



PXL-Digital

Graduaat systeem- en netwerkbeheer

Virtualization

Lector(en)

Stijn Jacobs en Jeroen Jean

I. Inhoudsopgave

I. Inhoudsopgave	2
1 Inleiding Docker, Podman en Kubernetes	12
1.1 Introductie	12
1.2 Containers	12
1.2.1 Inleiding	12
1.2.2 IT Geschiedenis: The Bad Old Days	12
1.2.3 VMware.....	12
1.2.4 Containers.....	14
1.2.5 Podman Demo	14
1.2.6 Recap	18
1.3 Applicaties	18
1.3.1 Traditionele applicaties.....	18
1.3.2 Cloud native en microservices.....	20
1.4 Docker	22
1.4.1 Bedrijf vs technologie	22
1.4.2 Docker technologie	23
1.4.3 Versies Docker.....	23
1.4.4 Podman	24
1.4.5 Podman Demo	25
1.5 Kubernetes	28
1.5.1 Inleiding	28
1.5.2 Geschiedenis van Kubernetes.....	29
1.5.3 Werking Kubernetes.....	29
1.5.4 Kubernetes On Prem en in de Cloud	33
1.5.5 Productiegeschiktheid Kubernetes	34
1.5.6 Hosted Kubernetes services	34
1.6 Openshift	35
1.7 Geschikte Workloads containers en VM's	35
1.8 Overzicht.....	36
2 Installatie docker en podman	37
2.1 Inleiding	37
2.2 Hardware Virtualisatieondersteuning.....	37

2.3	VMWare Workstation Pro/Edu installeren op host.....	38
2.4	Installeer VM's	39
2.5	Docker vs Podman.....	41
2.6	Parallel installeren.....	41
2.7	Podman installeren	42
2.7.1	Installatie zelf	42
2.7.2	Controle installatie.....	42
2.8	Docker installeren	43
2.8.1	Installatie zelf	43
2.8.2	Controle	44
2.9	Play with Docker	44
3	Beheer vanop afstand.....	48
3.1.1	Verbinding maken met je Server vanaf je hostcomputer.....	48
3.1.2	Let op met IP-nummer!.....	48
3.1.3	Oorzaak probleem.....	48
3.1.4	Probleem oplossen	51
3.2	SSH	52
3.2.1	Inleiding	52
3.2.2	SSH-server installeren en configureren.....	52
3.2.3	Verbinding maken met SSH-server	53
3.3	Remote Docker	55
3.3.1	Remote docker via opstellen Docker-daemon	55
3.3.2	Remote Podman/Docker via SSH	57
4	Containers en container images	60
4.1	Inleiding	60
4.2	Aanmaken docker hub account.....	60
4.3	Een RHEL 10 container image downloaden en gebruiken	63
4.4	Podman commando's	71
4.4.1	Containers en container images downloaden, uitvoeren en logs bekijken	71
4.4.2	Een interactieve container uitvoeren	74
4.4.3	Een detached container uitvoeren.....	76
4.4.4	Info over container opvragen.....	78
4.5	Interactief een container bouwen	81
4.5.1	Webserver.....	81

4.5.2	DNS-server	85
4.6	Containers starten, stoppen en verwijderen	95
4.7	Een Container Image bouwen met een containerfile	101
4.7.1	Inleiding	101
4.7.2	Algemene procedure	101
4.7.3	Praktijk	101
4.8	Images uploaden naar de Docker hub	113
4.8.1	Inloggen Docker hub	113
4.8.2	Private Repository	114
4.8.3	Public Repository	115
4.8.4	Image pushen	116
4.8.5	Image pullen	117
4.9	Cockpit	118
4.9.1	Inleiding	118
4.9.2	Installatie	119
4.9.3	Gebruik	119
5	Containers netwerken en data volumes	128
5.1	Inleiding	128
5.2	Container netwerken	128
5.2.1	Inleiding	128
5.2.2	Eerste mogelijkheid: Twee Containers op de Zelfde Host	128
5.2.3	Tweede mogelijkheid: Een Container Communiceert met een Externe Service	
	129	
5.2.4	Derde mogelijkheid: Een Externe Omgeving Communiceert met een Container	
	130	
5.2.5	Vierde mogelijkheid: Communicatie tussen Containers op Verschillende Hosts	
	131	
5.3	Rootless netwerktoegang	131
5.3.1	Vooraf	131
5.3.2	Check instellingen podman	132
5.3.3	Standaard rootless netwerk	133
5.3.4	User-space netwerk	141
5.4	Rootful netwerktoegang	148
5.5	User-space netwerken vs bridge netwerken	153

5.6	Netwerkmodi.....	154
5.7	Host-netwerk.....	155
5.8	Macvlan-netwerk.....	156
5.8.1	Inleiding	156
5.8.2	Macvlan netwerk aanmaken.....	156
5.8.3	Container met macvlan netwerk	157
5.8.4	Scheiding met host.....	160
5.8.5	Controle macvlan	161
5.9	Ipvlan	162
5.9.1	Inleiding	162
5.9.2	Ipvlan-netwerk aanmaken	162
5.9.3	Container met ipvlan netwerk	163
5.9.4	Scheiding met host.....	166
5.10	Uitgebreide vergelijking netwerken podman.....	167
5.11	Lokale volumes koppelen aan containers.....	168
5.11.1	Inleiding.....	168
5.11.2	Stateless vs stateful	168
5.11.3	Named volume	169
5.11.4	Bind mount.....	172
5.11.5	tmpfs volume.....	174
5.11.6	Remote volumes toevoegen aan containers.....	175
6	Podman images.....	178
6.1	Inleiding	178
6.2	Webserver	178
6.3	Database-server	185
6.4	Wordpress	187
6.4.1	Images die we gebruiken	187
6.4.2	Podman volumes aanmaken	189
6.4.3	Podman netwerk aanmaken.....	189
6.4.4	Container MariaDB.....	190
6.4.5	Container Wordpress	191
6.4.6	Website Wordpress configureren	191
6.5	Tips	195
7	Podman Compose.....	197

7.1	Inleiding	197
7.2	Installatie Podman Compose	197
7.3	Componenten van een .yml-bestand.....	198
7.3.1	Services	199
7.3.2	Volumes.....	200
7.3.3	Netwerken.....	200
7.3.4	Depends_on.....	201
7.3.5	Environment-variabelen	201
7.3.6	Command	201
7.3.7	Folded block en literal block	202
7.4	Voorbeeld 1: gedeelde map	203
7.4.1	.yml-bestand	203
7.4.2	Podman-compose up	205
7.4.3	Logs bekijken	207
7.4.4	Podman-compose down.....	208
7.4.5	Podman-compose down versus podman-compose stop	209
7.4.6	Podman-compose up -d	209
7.4.7	Podman-compose help	210
7.5	Voorbeeld 2: Wordpress	210
7.5.1	.yml-bestand	210
7.5.2	Podman-compose up	213
7.5.3	Website instellen	214
8	Podman Compose in combinatie met Containerfile	215
8.1	Inleiding	215
8.2	Voorbeeld 1	215
8.2.1	Containerfile.....	215
8.2.2	Bash script (script.sh).....	216
8.2.3	compose.yml.....	216
8.2.4	Uitvoer	217
8.3	Voorbeeld 2	217
8.3.1	Containerfile.....	217
8.3.2	Index.html	218
8.3.3	Compose.yml.....	218
8.3.4	Uitvoer	219

9 Podman pods	221
9.1 Inleiding	221
9.2 Werken met pods.....	222
9.3 Wordpress	225
9.3.1 Zonder gegevens op ServerXX	225
9.3.2 Met gegevens op ServerXX	227
10 Storage Infrastructure.....	231
10.1 Opslagsystemen	231
10.1.1 DAS (Direct Attached Storage):.....	231
10.1.2 SAN (Storage Area Network):.....	232
10.1.3 NAS (Network Attached Storage):.....	232
10.2 iSCSI	233
10.2.1 Inleiding.....	233
10.2.2 Toevoegen netwerk voor SAN.....	234
10.2.3 Installatie TrueNAS	237
10.2.4 iSCSI instellen.....	255
10.2.5 Verbinding maken met iSCSI-target op ServerXX	265
10.2.6 Initialiseren en formatteren van doel.....	266
10.2.7 Podman met iSCSI.....	267
11 Kubernetes	269
11.1 Inleiding	269
11.2 Kubernetes architectuur	270
11.2.1 Clusters.....	270
11.2.2 Control Plane	271
11.2.3 Worker nodes.....	272
11.3 Installatie.....	274
11.3.1 K3s versus k8s	274
11.3.2 VM	275
11.3.3 Installatie Kubernetes	275
11.4 Cluster aanmaken.....	278
11.5 Pods	279
11.5.1 Inleiding.....	279
11.5.2 Lifecycle	281
11.5.3 Pod netwerk	281

11.5.4	Pod aanmaken	282
11.6	Kubectl.....	288
11.6.1	Algemeen	288
11.6.2	Context.....	289
11.6.3	CRUD.....	291
11.6.4	Kubectl cheat sheet.....	293
11.7	Namespaces	293
11.7.1	Basis	293
11.7.2	Resources toewijzen aan NS	295
11.7.3	Resources opvragen in NS.....	296
11.8	Services	297
11.8.1	Algemeen	297
11.8.2	Endpoint slices	299
11.8.3	Services aanmaken.....	299
11.9	Deployments	308
11.9.1	Algemeen	308
11.9.2	ReplicaSets	308
11.9.3	Voorbeeld.....	308
11.9.4	Rolling updates	314
11.9.5	Rollback	318
11.10	Storage	323
11.10.1	Soorten volumes	325
11.10.2	PersistentVolume (PV)	326
11.10.3	PersistentVolumeClaim (PVC).....	327
11.10.4	StorageClass.....	328
11.10.5	Access modes.....	329
11.10.6	Reclaim Policies	329
11.10.7	Voorbeelden.....	329
11.11	Disclaimer	339
12	Van Podman naar Kubernetes.....	340
12.1	Inleiding	340
12.2	Podman pods voor Wordpress.....	340
12.3	Genereren van een Kubernetes-manifest.....	341
12.4	Toepassen van manifestbestand	344

12.5	Back-up maken en manifestbestand aanpassen	346
12.6	PVC-manifest aanmaken	354
12.7	Toepassen van manifestbestanden.....	355
12.8	Instellen Wordpress	355
12.9	Check persistant storage in cluster	359
12.10	Meerdere manifestbestanden.....	361
12.11	Deployments	373
13	Installatie Proxmox	379
13.1	Inleiding	379
13.1.1	Hypervisors	379
13.1.2	Proxmox	380
13.2	Proxmox VE installeren.....	385
13.2.1	Download ISO	385
13.2.2	Aanmaken VM voor Proxmox	385
13.2.3	installatie in VM	391
13.3	Repositories aanpassen.....	397
13.4	Abonnementswaarschuwing uitzetten	401
13.5	Meerdere Proxmox Servers	402
14	Proxmox Standalone Server.....	405
14.1	Inleiding	405
14.2	GUI vs CLI	405
14.2.1	GUI	405
14.2.2	CLI	406
14.3	Belangrijke instellingen	406
14.3.1	Systeem software-updates	406
14.3.2	Netwerkconfiguratie.....	407
14.3.3	Tijdsynchronisatie.....	410
14.4	Overzicht node management	411
14.5	Overzicht datacenter management	418
15	VM's en containers in Proxmox	424
15.1	Opslag	424
15.1.1	Inleiding.....	424
15.1.2	Configuratie schijven en volumes	428
15.2	VM	434

15.2.1	ISO toevoegen voor VM	434
15.2.2	VM aanmaken	436
15.2.3	Nested virtualization	443
15.2.4	Instellingen VM	444
15.2.5	VM starten enz.	445
15.2.6	Verbinding maken met VM	445
15.2.7	VM instellen.....	447
15.3	Container	448
15.3.1	Vereisten.....	448
15.3.2	Container template downloaden	448
15.3.3	Container aanmaken	451
15.3.4	Container starten.....	459
16	Clusters in Proxmox	461
16.1	Inleiding	461
16.2	Nodes aanmaken en configureren	461
16.2.1	Mappen kopiëren.....	461
16.2.2	IP-nummer instellen	462
16.2.3	Hostname instellen.....	465
16.3	Installatie Cluster	467
16.3.1	Inleiding.....	467
16.3.2	Voorbereiding.....	468
16.3.3	PVE.....	468
16.3.4	PVE2.....	469
16.3.5	PVE3.....	470
16.4	High availability en live migratie	471
16.4.1	Algemeen.....	471
16.4.2	Migratie zonder gedeelde opslag.....	474
16.4.3	Migratie met gedeelde opslag	474
16.5	Voorbeeld migratie zonder gedeelde opslag	477
16.6	Ceph	481
16.6.1	Introductie tot Ceph-opslag	481
16.6.2	Belangrijke componenten van Ceph.....	484
16.6.3	Installatie	488
16.6.4	Check.....	499

16.7	Voorbeeld Live migratie	500
16.8	High availability	506
16.8.1	Algemeen	506
16.8.2	Management Tasks	507
16.8.3	Check	509
16.8.4	Affinity Rules	509
16.8.5	Inleiding	509
16.8.6	HA Node Affinity Rules	510
16.8.7	Fencing	513
17	Proxmox CLI	514
17.1	Inleiding	514
17.2	Installatie VM	514
17.3	Installatie container	520
17.4	Netwerken	521
17.4.1	Inleiding	521
17.4.2	Back-up netwerkconfiguratie	521
17.4.3	NAT-bridge	521
17.4.4	Bonding	523
17.4.5	Restore netwerkconfiguratie	530
18	Proxmox cloud-images	532
18.1	Inleiding	532
18.2	Formaat	532
18.3	Dowloadlinks	533
18.4	Voorbeelden	533
18.4.1	Via GUI	533
18.4.2	Via CLI	552
18.4.3	Cloud-Init YAML-bestand	555

1 Inleiding Docker, Podman en Kubernetes

1.1 Introductie

Docker, Podman en Kubernetes zijn momenteel veel besproken technologieën in de IT-wereld, en het wordt steeds belangrijker om ze goed te begrijpen.

We leren de basisprincipes van containers met een eenvoudige demo.

Daarnaast behandelen we ook concepten als microservices en cloud native.

1.2 Containers

1.2.1 Inleiding

We beginnen met een kort overzicht van de geschiedenis, waarbij we virtual machines (VM's) en hun voor- en nadelen behandelen. Daarna duiken we in de wereld van containers, tonen een demonstratie, en vatten de belangrijkste punten samen.

1.2.2 IT Geschiedenis: The Bad Old Days

Om containers goed te begrijpen, is het belangrijk om de context te kennen waarin ze zijn ontstaan. Fundament van IT is dat applicaties de kern van elk bedrijf vormen. Of je nu in de bankwereld, retail, luchtvaart of een andere sector werkt... applicaties staan centraal in de bedrijfsvoering. Zonder applicaties geen bedrijf, en dat wordt steeds meer de realiteit. Tot midden jaren 2000 werd elke applicatie vaak op een eigen fysieke server gehost. Deze servers waren duur en groot, en het aankopen ervan ging gepaard met hoge kosten, zowel kapitaaluitgaven (CapEx) als operationele kosten (OpEx), zoals stroom, koeling en personeel voor beheer.

Bij het kiezen van de juiste server voor een nieuwe applicatie was de vraag "hoe groot en hoe snel moet de server zijn?" vaak lastig te beantwoorden. IT afdelingen speelden op zeker door krachtige en snelle servers aan te schaffen om performanceproblemen te voorkomen. Dit leidde vaak tot servers die slechts 5-10% van hun capaciteit gebruikten... Een verspilling van bedrijfskapitaal en middelen.



1.2.3 VMware

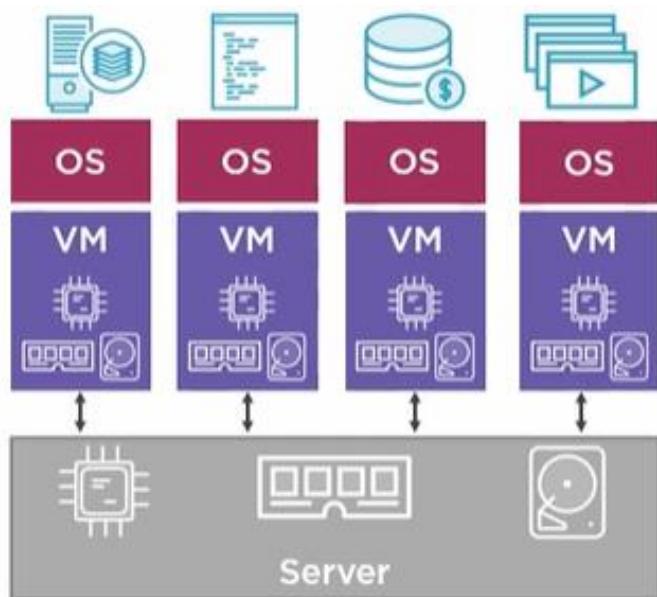
VMware bracht hier verandering in door technologie te introduceren waarmee we meer uit dezelfde overgespecificeerde fysieke servers konden halen. Dit leidde tot een efficiënter gebruik van middelen

doordat meerdere applicaties veilig op één fysieke server konden draaien door middel van virtualisatie. Dit betekende dat nieuwe applicaties niet automatisch nieuwe servers vereisten, wat de IT-wereld aanzienlijk verbeterde. Toch bracht dit model ook uitdagingen met zich mee.



1.2.3.1 Tekortkomingen van hypervisors

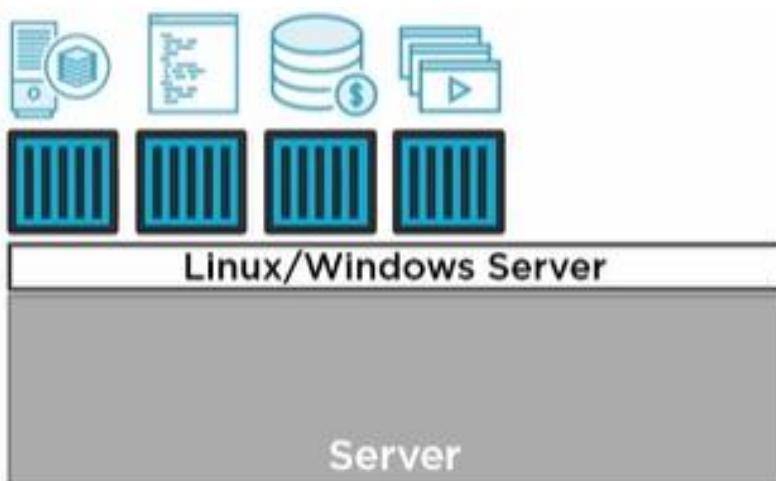
Hoewel virtualisatie via VMware en andere hypervisors veel voordelen bood, bleef het model inefficiënt in sommige opzichten.



Elke virtuele machine (VM) draait op een eigen besturingssysteem, dat aanzienlijke systeembronnen verbruikt (CPU, geheugen, schijfruimte) zonder dat er nog maar één applicatie draait.

Daarnaast brengt elk besturingssysteem extra operationele lasten met zich mee, zoals beveiligingspatches en updates, en potentiële licentiekosten. Dit "hongerige besturingssysteemmodel" maakt het hele proces duurder en complexer dan nodig zou zijn.

1.2.4 Containers



Containers bieden een slank alternatief. In plaats van meerdere VM's met hun eigen besturingssysteem bovenop een hypervisor, draait een containeromgeving op één enkel besturingssysteem.

In plaats van volledige VM's worden er kleinere, efficiëntere containers gecreëerd die enkel de benodigde bronnen van het besturingssysteem gebruiken.

Voordelen van containers

Elke container herbergt één applicatie, maar zonder de overhead van een volledig besturingssysteem per applicatie. Dit resulteert in schnellere opstarttijden en een efficiënter gebruik van systeembronnen.

Containers maken gebruik van één besturingssysteem dat wordt gedeeld door alle containers, wat leidt tot minder belasting van systeembronnen en minder administratieve taken. Hierdoor ontstaat ruimte om meer applicaties te draaien, waardoor bedrijven hun IT-investeringen beter kunnen benutten.

Bovendien zijn containerized applicaties ideaal voor situaties waar flexibiliteit en snelheid vereist zijn, zoals bij het op- en afschalen van diensten op aanvraag.

1.2.5 Podman Demo

Laten we een eenvoudige demo tonen om te laten zien hoe containers werken met Podman.

Voor deze demo gebruiken we een machine met Podman geïnstalleerd.

Docker en Podman zijn beide tools om containers te beheren, maar Podman is een container-engine die:

- Daemonless werkt (geen centrale daemon; elk commando is een eigen proces).
- Rootless containers ondersteunt (veiliger in multi-user omgevingen).
- Docker-compatibel is (images, Dockerfiles en CLI-patronen).
- Pods biedt (groep containers die netwerk- en andere namespaces delen, zoals in Kubernetes).

- Integreert met systemd (containers als services beheren).

We downloaden nu het image nginx (webserver).

```
student@serverXX:~$ podman image pull docker.io/library/nginx:latest
```

```
student@serverXX:~$ podman image pull docker.io/library/nginx:latest
Trying to pull docker.io/library/nginx:latest...
Getting image source signatures
Copying blob a2da0c0f2353 done    |
Copying blob e5d9bb0b85cc done    |
Copying blob 14e422fd20a0 done    |
Copying blob 14a859b5ba24 done    |
Copying blob 716cdf61af59 done    |
Copying blob b1badc6e5066 done    |
Copying blob c3741b707ce6 done    |
Copying config ad5708199e done    |
Writing manifest to image destination
ad5708199ec7d169c6837fe46e1646603d0f7d0a0f54d3cd8d07bc1c818d0224
student@serverXX:~$
```

We hebben nu een Podman-image gedownload met de naam nginx. Denk aan een image als een vooraf verpakte applicatie. Een image bevat alles wat nodig is om een applicatie te draaien. In dit geval bevat onze image een webserver die statische content serveert.

We listen de image.

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

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
docker.io/library/nginx	latest	ad5708199ec7	3 weeks ago	197 MB

We achterhalen het IP-nummer van de netwerkkaart.

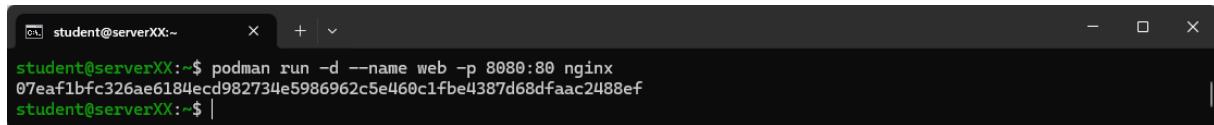
```
student@serverXX:~$ nmcli device show ens160 | grep 'IP4.ADDRESS'
```

```
student@serverXX:~$ nmcli device show ens160 | grep 'IP4.ADDRESS'
IP4.ADDRESS[1]:                         192.168.112.100/24
student@serverXX:~$
```

Om een container van deze image te starten, gebruiken we een commando dat instructies geeft om een nieuwe container te starten, gebaseerd op de gedownloade image, met een specifieke naam en gekoppeld aan een bepaalde netwerkpoort op de container host. We kiezen op de containerhost voor poort 8080. Zodra de container draait, krijgen we een uniek ID als bevestiging. Dit proces duurt slechts een fractie van een seconde.

```
student@serverXX:~$ podman run -d --name web -p 8080:80 nginx
```

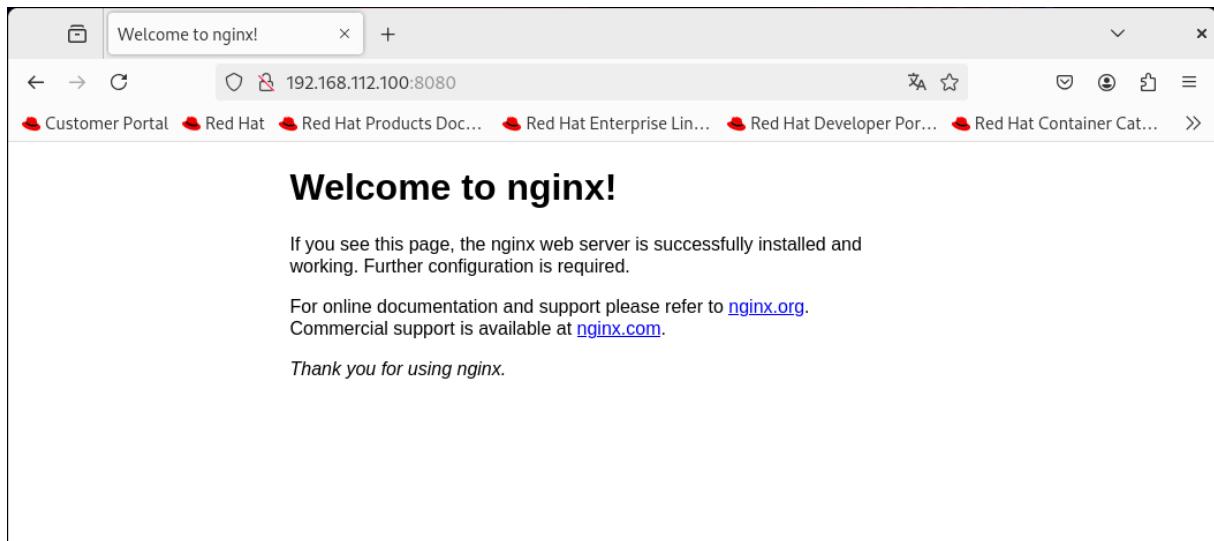
```
07eaf1bf326ae6184ecd982734e5986962c5e460c1fbe4387d68dfaac2488ef
```



```
student@serverXX:~$ podman run -d --name web -p 8080:80 nginx
07eaf1bfc326ae6184ecd982734e5986962c5e460c1fbe4387d68dfaac2488ef
student@serverXX:~$ |
```

Om te kijken of de webserver draait maken we gebruik van het IP-adres van de laptop en de blootgestelde poort, in dit geval poort 8080.

Door deze gegevens in een browser in te voeren, kunnen we de webserver direct zien draaien.



Samenvattend: we hebben een image gedownload (denk aan een vooraf verpakte applicatie), die we hebben gestart als een container. Na het starten hebben we gecontroleerd of alles correct werkt door de webserver te bezoeken via de browser.

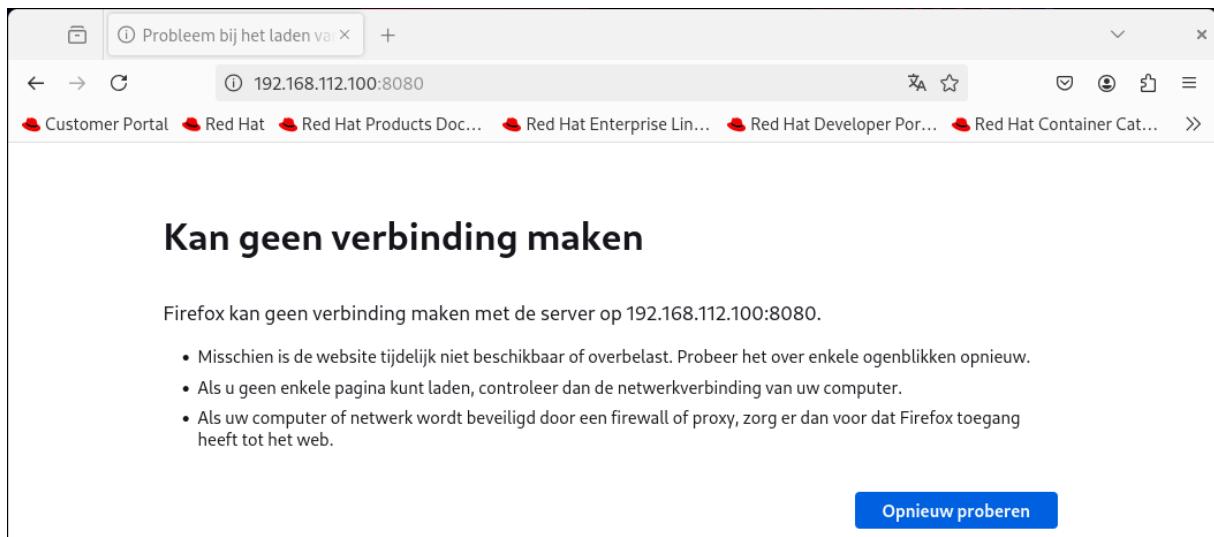
We kunnen de container ook eenvoudig stoppen.

```
student@serverXX:~$ podman stop web
web
```



```
student@serverXX:~$ podman stop web
web
student@serverXX:~$ |
```

U zal zien dat de webpagina nu niet meer bereikbaar is.



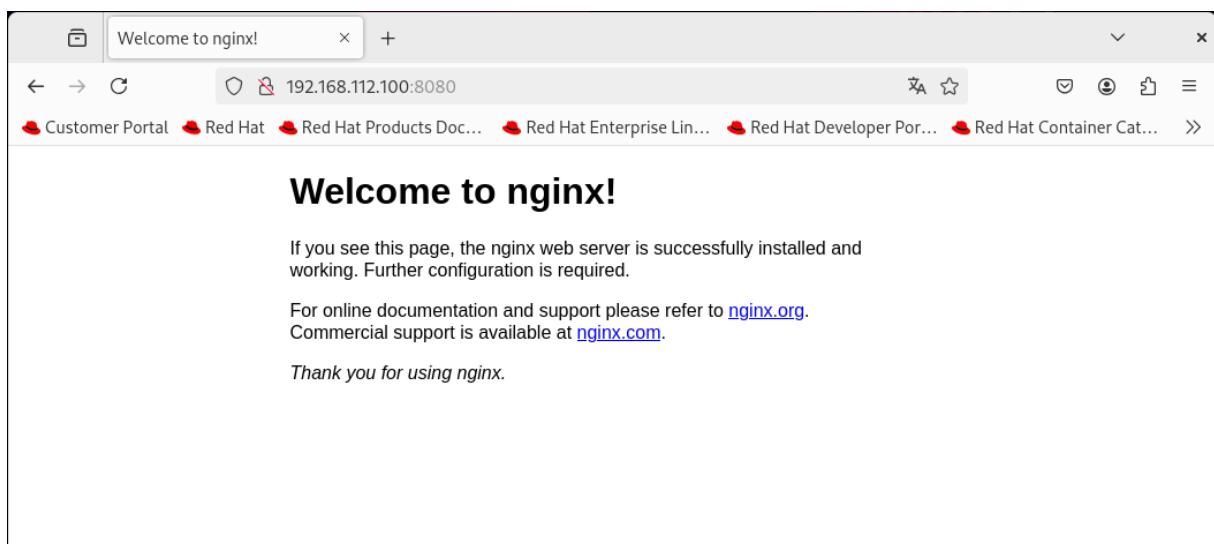
We kunnen de container nu opnieuw starten.

```
student@serverXX:~$ podman start web
```

```
web
```



Uiteraard zal de webpagina nu terug bereikbaar zijn.



Omdat containers vergelijkbaar zijn met virtuele machines, maar lichter en sneller, kunnen we de container eenvoudig stoppen en starten met enkele commando's. Dit maakt het beheer van applicaties bijzonder efficiënt. Containers bieden flexibiliteit: u kunt ze stoppen, herstarten en zelfs verwijderen, allemaal met eenvoudige commando's.

1.2.6 Recap

Applicaties zijn cruciaal voor bedrijfsprocessen; zonder applicaties is er geen bedrijf.

Traditioneel gezien kostte het opzetten en onderhouden van applicaties veel tijd en geld, en leidde dit tot veel verspilling. De introductie van virtualisatie door VMware en andere technologieën vermindeerde de doorlooptijden, verlaagde de kosten en verhoogde de efficiëntie drastisch.

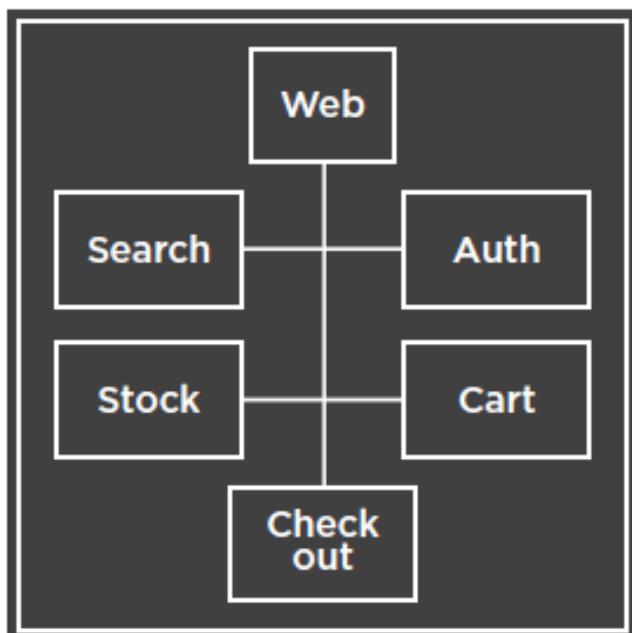
De ontwikkeling van containers door de Linux-gemeenschap ging echter verder en bood oplossingen die sneller, goedkoper en efficiënter zijn dan traditionele virtualisatie.

Containers bieden de mogelijkheid om traditionele applicaties te moderniseren door ze in containers te draaien zoals bijvoorbeeld d.m.v. Docker. Containers zijn ideaal voor het implementeren van microservices en cloud-native architecturen.

1.3 Applicaties

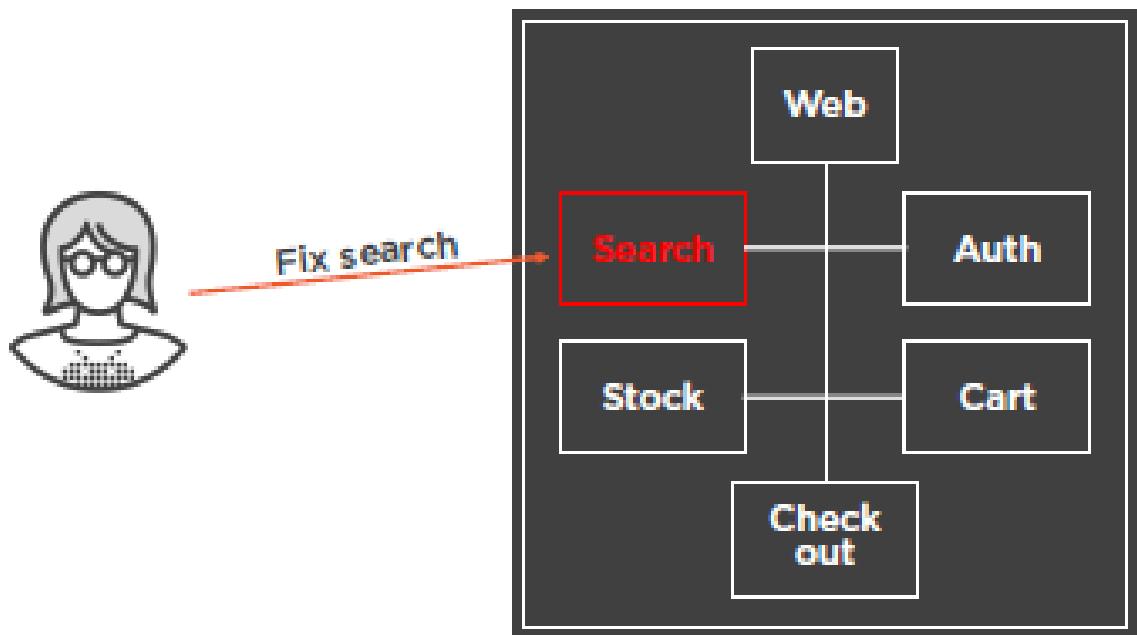
1.3.1 Traditionele applicaties

Traditionele of monolithische applicaties bundelen alle functionaliteiten in één enkel programma, wat het onderhoud en updates uitdagend maakt.



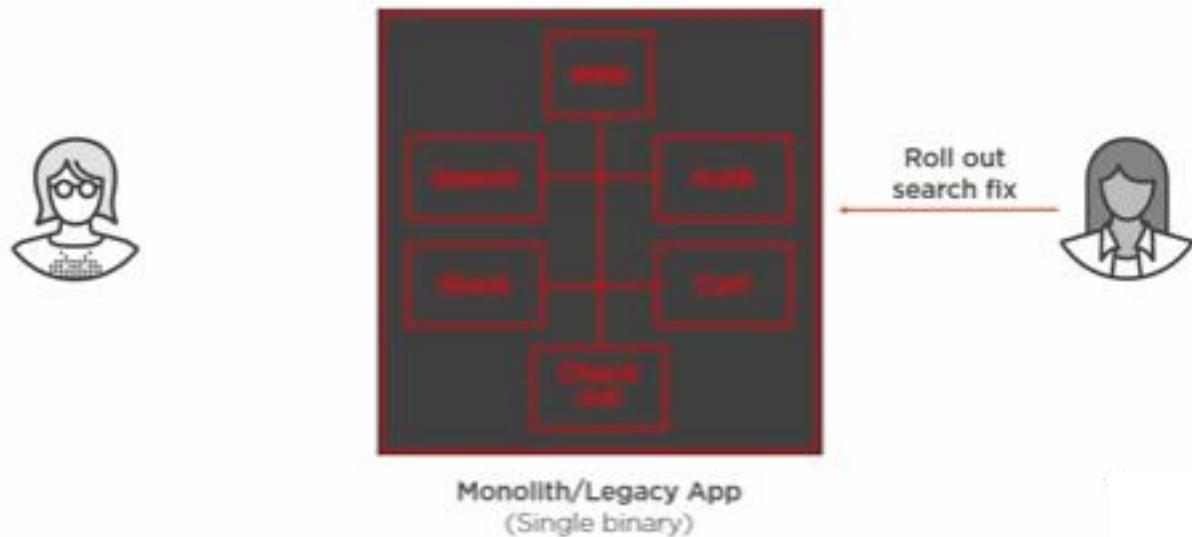
Monolith/Legacy App
(Single binary)

Als er een update uitgevoerd moet worden past de ontwikkelaar de code aan

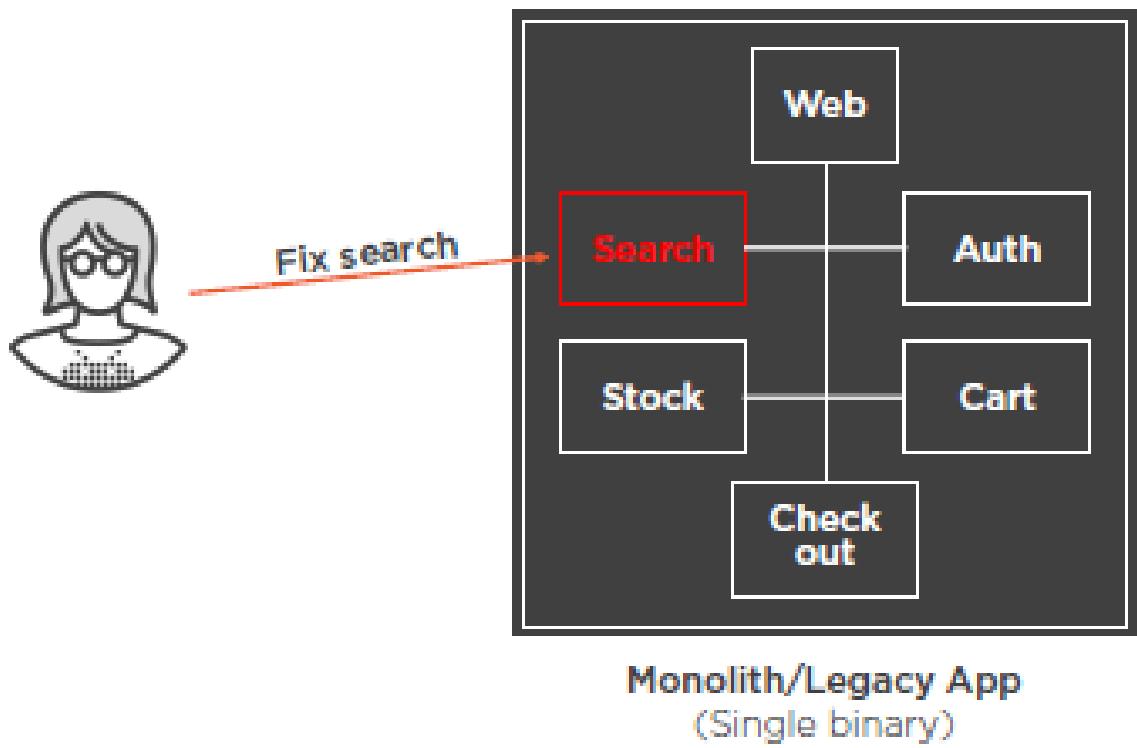


**Monolith/Legacy App
(Single binary)**

Bij legacy apps (monolithic apps) is de enige manier om een fout te herstellen de hele app uit te zetten.



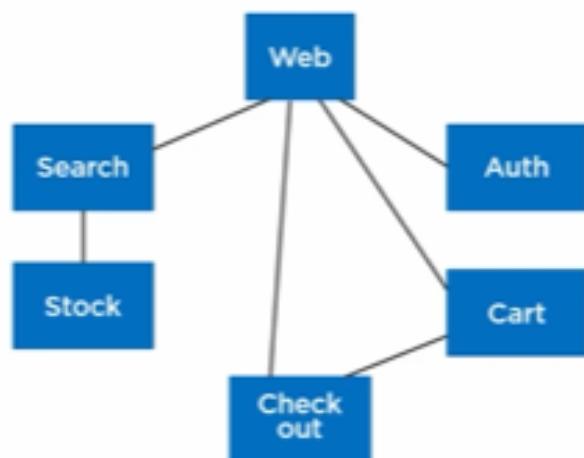
Cloud-native microservices splitsen deze functionaliteiten op in afzonderlijke, beheersbare diensten, waardoor ontwikkelaars en operators veel gerichter en efficiënter kunnen werken. Updates kunnen uitgevoerd worden op specifieke onderdelen zonder de gehele applicatie offline te hoeven halen, wat het risico en de impact op de bedrijfsvoering minimaliseert.



1.3.2 Cloud native en microservices

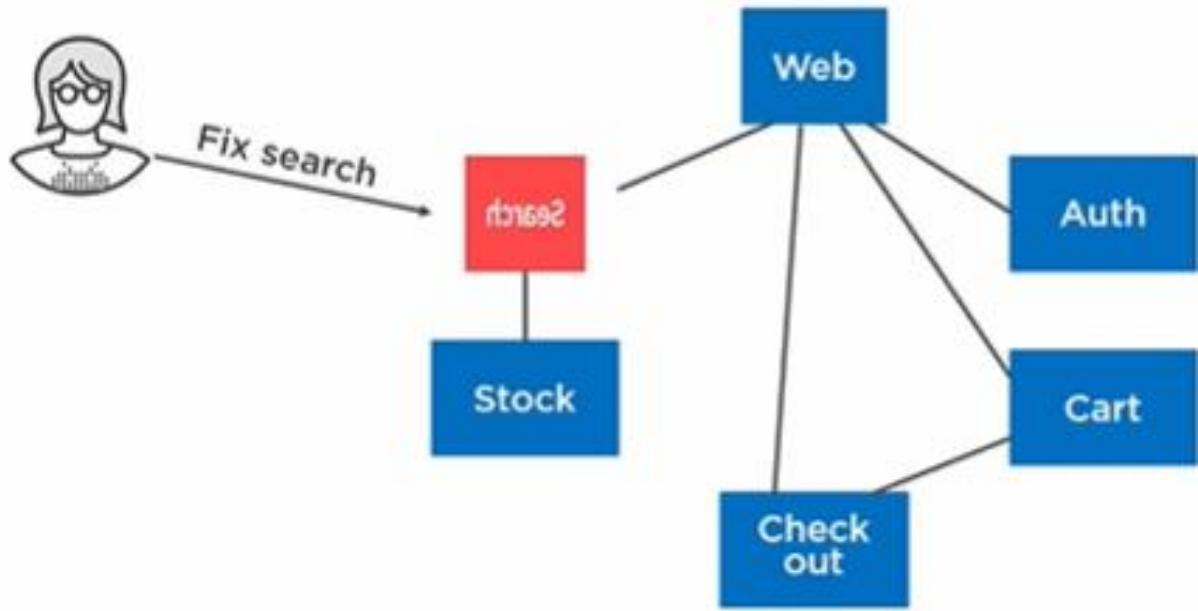
Cloud-native is een benadering die applicaties bouwt en beheert op een manier die voldoet aan de moderne eisen van bedrijfsvoering en cloud computing. Dit betekent dat cloud-native applicaties niet per se in de cloud draaien; ze kunnen ook in een on-premises datacenter gehost worden. Het draait allemaal om hoe de applicatie is opgebouwd en beheerd, met flexibiliteit en schaalbaarheid als kernprincipes.

Bij cloud native en microservices ga je al alle verschillende componenten splitsen en maak van elk component een kleine mini-app of mini-service. Mini-apps of mini-services communiceren met elkaar.

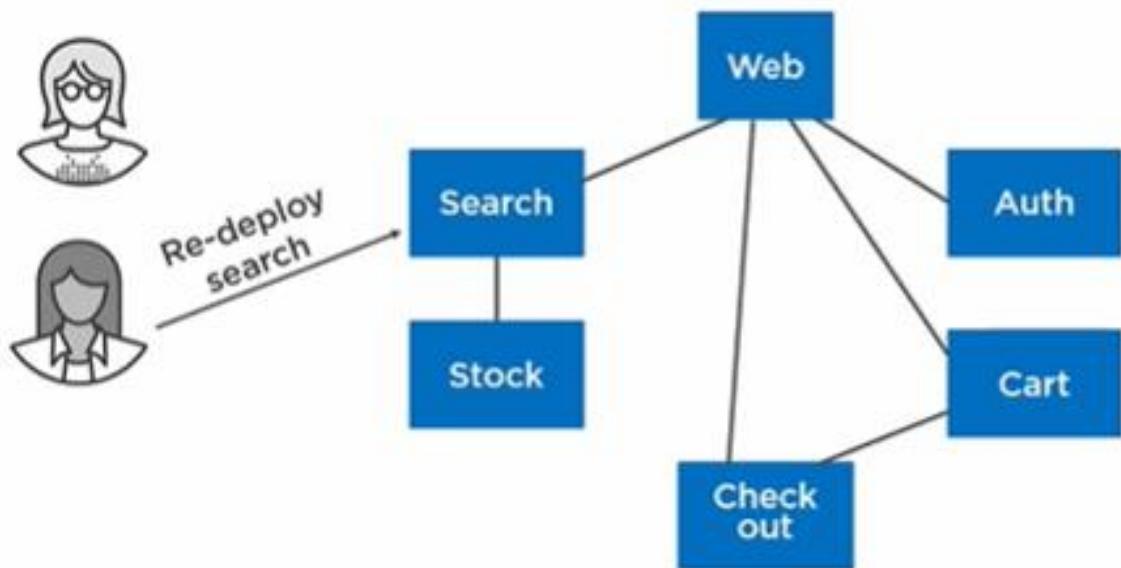


Bij cloud-native en microservices werkt de ontwikkelaar een fix bij.

De ontwikkelaar moet bijvoorbeeld alleen de zoekcode bijwerken wanneer de zoekfunctie wordt bijgewerkt.



De sys admin hoeft dan alleen de nieuwe versie van de search service uit te rollen.



Je kan een cloud native app runnen in een on-prem datacenter. Cloud native gaat over de manier hoe een applicatie is gebouwd en wordt beheerd. Zo kan je bijvoorbeeld de front-end onafhankelijk van de back-end schalen.

Containers kunnen gezien worden als Virtualisatie 2.0 - ze verbeteren de meeste aspecten van hypervisors en leggen de basis voor moderne, cloud-native en microservices-applicaties.

Hoewel containers veel voordelen bieden, zullen ze in veel gevallen naast virtuele machines blijven bestaan, vooral in grotere ondernemingen en traditionele IT-omgevingen.

1.4 Docker

1.4.1 Bedrijf vs technologie

Er is het bedrijf, Docker Inc., en er is de technologie Docker.

Zoals we zullen zien, zijn ze nauw verbonden, maar ze zijn niet hetzelfde. Dus, we zullen ze apart bespreken, en we beginnen met het bedrijf.

Docker, inc. is een technologie-startup uit San Francisco en de belangrijkste sponsor van de open-source containertechnologie met dezelfde naam. Maar het is veel meer dan dat.

Docker, het bedrijf, begon als dotCloud, een bedrijf dat een ontwikkelaarsplatform bovenop Amazon Web Services aanbood. Dat bleek echter niet zo succesvol als bedrijf en rond 2013 was er behoefte aan iets nieuws.

Ze gebruikten containers om hun platform bovenop AWS te bouwen, en, dit is het belangrijkste, ze hadden zelfgemaakte technologie ontwikkeld als intern hulpmiddel om hun containers op te zetten en te beheren.

Kort samengevat: ze hadden iets nieuws nodig, keken naar hun interne technologie voor het bouwen van containers en dachten: wat als we de containertechnologie verspreiden en er een bedrijf omheen bouwen?

De naam Docker komt van een Britse uitdrukking, een samentrekking van 'doc' en 'worker' (dokwerker).

Rond 2013 maakte het bedrijf dotCloud een grote draai en veranderde het zijn bedrijfsmodel van een ontwikkelaarsplatform op AWS naar een bedrijf dat de manier veranderde waarop we software bouwen, verzenden en draaien. Het gaf ons de containers. In de beginjaren leek het erop dat ze weliswaar hard werkten aan de technologie, maar dat er geen helder plan was om er een levensvatbaar bedrijf van te maken. Tegenwoordig hebben ze een solide bedrijfsplan, en één van de belangrijkste focuspunten van Docker Inc. is nu het verkopen van een enterprise-grade container management platform en het bieden van ondersteuning.

Kortom, Docker Inc. is een technologie-startup die de wereld Docker en eenvoudig te gebruiken containers heeft gebracht.

Tegenwoordig richten ze zich op orchestration en het ondersteunen van gecontaineriseerde applicaties op schaal met een focus op ondernemingen.

1.4.2 Docker technologie

Docker maakt het eenvoudig om applicaties in containers te draaien. De Docker-applicatie is open-source en, zoals veel open-source software tegenwoordig, beschikbaar op GitHub.

De open-source Docker-technologie, ook wel de Community Edition of CE genoemd, is gratis te gebruiken en iedereen kan bijdragen aan de ontwikkeling. Bedrijven zoals Red Hat, IBM, Microsoft, en vele andere grote spelers dragen bij.

Naast de Community Edition biedt Docker Inc. ook een Enterprise Edition aan, die in wezen dezelfde technologie stack gebruikt, maar op een langzamer releaseschema draait voor meer stabiliteit, met extra functies en uiteraard een enterprise-class ondersteuningscontract.

Beide edities zijn gericht op het draaien en beheren van applicaties in containers, ofwel gecontaineriseerde apps.

1.4.3 Versies Docker

Aan de technologiekant zijn er twee hoofdversies: de Community Edition (CE) en de Enterprise Edition (EE). Beide kunnen zowel on-premises als in de cloud draaien, maar de namen zeggen al veel.

Community Edition (CE)

- Free
- Quick release cycle
- Edge channel (fun/scary stuff)

Enterprise Edition (EE)

- Costs money
- Official support
- Patches
- Cautious release cycle
- Stability
- Extras
 - Enterprise web UI
 - Security features
 - AD/LDAP
 - Private registry
 - Policies
 - FIPS
 - Pipelines...

De Community Edition is gericht op de community: het is gratis, heeft een snelle release-cyclus en een edge-kanaal als je de nieuwste ontwikkelingen wilt uitproberen.

De Enterprise Edition, daarentegen, is een betaalde versie en biedt ondersteuning, zoals telefonische hulp wanneer iets misgaat, en langere ondersteuning met beveiligingsupdates en patches (vaak tot twee jaar), vergeleken met ongeveer zes maanden bij de Community Edition.

Qua release- en ondersteuningsbeleid heeft de Enterprise Edition een tragere, stabielere release-cyclus; alles moet grondig getest zijn. EE biedt ook extra functionaliteiten, zoals een uitgebreidere

webinterface, verbeterde beveiligingsmogelijkheden (bijv. gedetailleerd gebruikers- en groepsbeheer, integratie met Active Directory of LDAP), en een privéregister voor het opslaan van Docker-images. Of je nu on-premises of in een privécloud werkt, je kunt je images opslaan in een beveiligd, door jou beheerd register.

Daarnaast ondersteunt EE extra features die belangrijk zijn voor grotere organisaties.

1.4.4 Podman

Podman is een alternatief voor Docker en wordt vooral gepromoot binnen de Red Hat-ecosystemen. Net als Docker is Podman open-source en bedoeld om containers te bouwen, draaien en beheren. Het grote verschil is dat Podman zonder centrale daemon werkt (daemonless), waardoor het veiliger en flexibeler is.

De command line interface van Podman is grotendeels compatibel met Docker, waardoor veel Docker-commando's bijna één-op-één gebruikt kunnen worden. Dit maakt de overstap eenvoudig voor ontwikkelaars en beheerders die al met Docker vertrouwd zijn.

Podman is volledig gratis en open-source software.

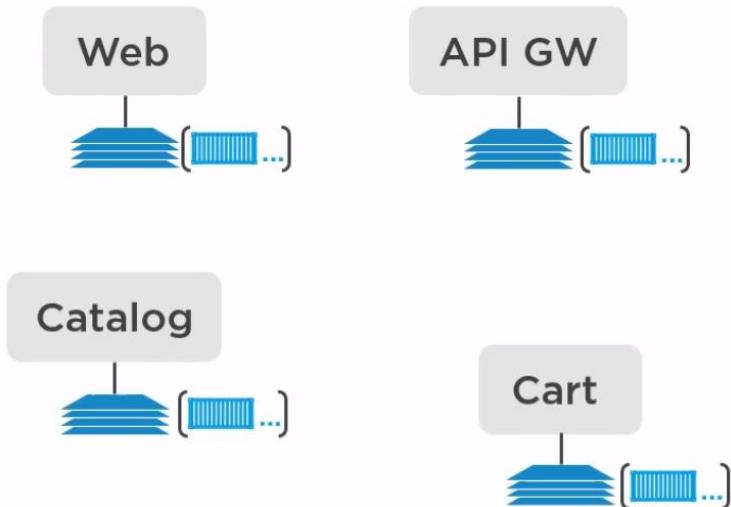
Je kan Podman zien als een doorontwikkeling van Docker.

1.4.5 Podman Demo

Eerder in dit hoofdstuk hebben we het kort gehad over microservices.

We zeiden dat moderne cloud-native apps zijn opgebouwd uit veel kleine onderdelen die samenwerken om één nuttige app te vormen.

Dus stel je een app voor met een web frontend service, een API gateway, een catalogus, een winkelwagentje, enzovoort. In de cloud-native microservices wereld wordt elke service apart gecodeerd en leeft elk in zijn eigen container.



Verschillende teams kunnen zelfs verantwoordelijk zijn voor verschillende services. Dit betekent dat elke service onafhankelijk van de rest kan worden aangepast of bijgewerkt. Natuurlijk communiceren ze allemaal met elkaar om de app als geheel te vormen. Dit is eenvoudige code die een webserver draait. Voor ontwikkelaars is dit dagelijkse kost, maar voor systeembeheerders is het gewoon broncode die, wanneer uitgevoerd, een webpagina toont. Laten we kort bekijken hoe we code kunnen uitvoeren als een container.

1.4.5.1 Stap één: coderen

We maken in onderstaand voorbeeld code en bouwen deze om tot een Podman-image.

Een image is als een gestopte container, of een sjabloon voor hoe je een container bouwt. We bouwen een image, zetten het in een registry, en starten een container ervan op.



Podman en docker zijn zeer flexibel en kan projectbestanden uitvoeren die in veel verschillende programmeertalen en omgevingen zijn geschreven. Podman en docker containers zijn namelijk in

staat om vrijwel elke software te draaien zolang er een geschikte runtime of omgeving beschikbaar is binnen de container. Hier zijn enkele van de meest voorkomende talen en omgevingen die je in Docker kunt uitvoeren:

Python: Via officiële Python Docker images kun je Python scripts en applicaties uitvoeren.
Node.js: Voor JavaScript en TypeScript applicaties gebruik je Node.js Docker images.
Java: Met officiële images van Java (OpenJDK of Oracle JDK) kun je Java applicaties uitvoeren.
C# (.NET): Docker ondersteunt .NET Core en .NET Framework voor C# applicaties.
Go (Golang): Voor het uitvoeren van Go applicaties zijn er officiële Go images.
Ruby: Ruby on Rails en andere Ruby applicaties kunnen draaien in Ruby Docker containers.
PHP: PHP en frameworks zoals Laravel kunnen worden uitgevoerd met PHP Docker images.
C++/C: Met aangepaste Docker images kun je C++ of C programma's compileren en uitvoeren.
R: Voor statistische analyses en data science toepassingen kun je R in containers draaien.
Perl: Met Perl Docker images kun je Perl scripts uitvoeren.
Rust: Rust applicaties kunnen worden uitgevoerd met Docker images die de Rust toolchain bevatten.
PowerShell: Je kan PowerShell scripts uitvoeren met de juiste Docker images.
Bash: je kunt Bash scripts gebruiken binnen Docker containers, en het wordt vaak gebruikt voor verschillende taken zoals automatisering, configuratie, en het opzetten van omgevingen.

We zullen in dit voorbeeld kiezen voor Bash. Misschien ga je nog niet alles begrijpen maar het is een demo. Later wordt alles duidelijker als je zelf aan de slag gaat.

Maak een nieuwe directory voor je project, bijvoorbeeld docker-linux, en ga naar deze directory.
student@serverXX:~\$ mkdir docker-linux

Maak in deze directory een bestand genaamd Dockerfile aan zonder extensie.

student@serverXX:~\$ touch docker-linux/Dockerfile

De inhoud van de containerfile is de volgende.

```
student@serverXX:~$ nano docker-linux/Dockerfile
# Gebruik een officiële Fedora image als basis

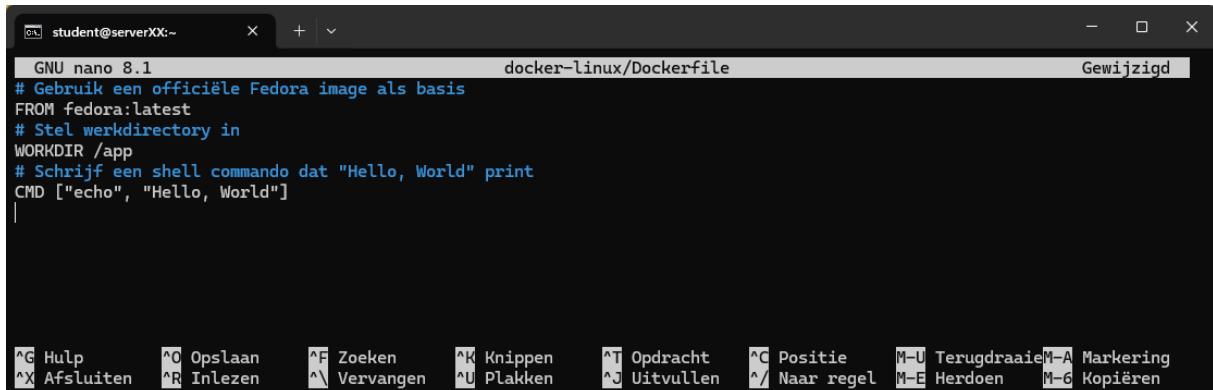
FROM fedora:latest

# Stel werkdirectory in

WORKDIR /app

# Schrijf een shell commando dat "Hello, World" print

CMD ["echo", "Hello, World"]
```



```
student@serverXX:~$ nano docker-linux/Dockerfile
# Gebruik een officiële Fedora image als basis
FROM fedora:latest
# Stel werkdirectory in
WORKDIR /app
# Schrijf een shell comando dat "Hello, World" print
CMD ["echo", "Hello, World"]
```

^G Hulp ^O Opslaan ^F Zoeken ^K Knippen ^T Opdracht ^C Positie M-U Terugdraaien M-A Markering
^X Afsluiten ^R Inlezen ^V Vervangen ^U Plakken ^J Uitvullen ^/ Naar regel M-E Herdoen M-G Kopiëren

Uitleg van de Dockerfile:

FROM fedora:latest

Dit commando gebruikt een officiële fedora image als basis. Deze image bevat Bash en is geschikt voor Linux-omgevingen.

WORKDIR /app

Stelt de werkdirectory binnen de container in. Dit is in dit geval optioneel.

CMD ["echo", "Hello, World"]

Dit commando voert bash uit met de instructie om de tekst "Hello, World" naar de standaarduitvoer te schrijven.

Het CMD commando bepaalt wat er wordt uitgevoerd wanneer de container start.

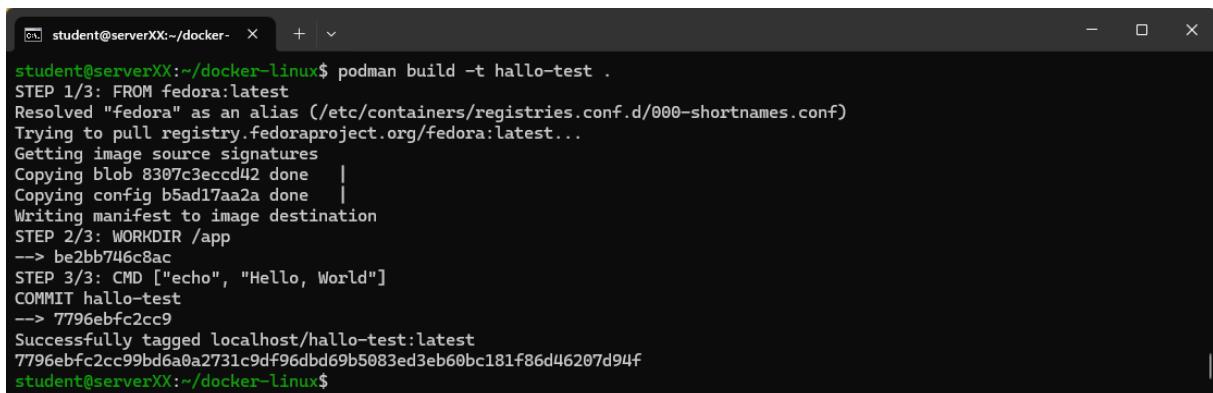
1.4.5.2 Stap twee: Podman image aanmaken en opslaan

Ga naar de directory waarin de Dockerfile staat.

```
student@serverXX:~$ cd docker-linux/
```

Bouw de Podman-image met de naam hallo-test vanuit Bash:

```
student@serverXX:~/docker-linux$ podman build -t hallo-test .
```



```
student@serverXX:~/docker-linux$ podman build -t hallo-test .
STEP 1/3: FROM fedora:latest
Resolved "fedora" as an alias (/etc/containers/registries.conf.d/000-shortnames.conf)
Trying to pull registry.fedoraproject.org/fedora:latest...
Getting image source signatures
Copying blob 8307c3eccd42 done    |
Copying config b5ad17aa2a done    |
Writing manifest to image destination
STEP 2/3: WORKDIR /app
--> be2bb746c8ac
STEP 3/3: CMD ["echo", "Hello, World"]
COMMIT hallo-test
--> 7796ebfc2cc9
Successfully tagged localhost/hallo-test:latest
7796ebfc2cc9bda60a2731c9df96dbd69b5083ed3eb60bc181f86d46207d94f
student@serverXX:~/docker-linux$
```

We hebben nu een image.

Wanneer je "podman build -t hallo-test ." uitvoert, doet Docker het volgende:

Lezen van de Dockerfile

Docker zoekt in de huidige directory (.) naar een bestand genaamd Dockerfile.

Uitvoeren van Instructies

Podman voert de instructies in de Dockerfile één voor één uit. Dit kan bijvoorbeeld het ophalen van een basisimage, het kopiëren van bestanden, het installeren van software zijn enzovoort.

Image Bouwen

Elke uitgevoerde instructie creëert een laag in de nieuwe Docker image. Zodra alle stappen zijn uitgevoerd, resulteert dit in een complete image.

Image Taggen

De voltooide image wordt opgeslagen met de naam en tag die je hebt opgegeven (hallo-test in dit geval).

Opslag in Lokale Docker Registry

De nieuwe image wordt opgeslagen in je lokale Docker image registry op je computer.

Je kan een image ook bewaren op het internet, zoals bijvoorbeeld de Docker Hub. We komen daar later op terug.

Je kan de lokale images als volgt opvragen.

```
student@serverXX:~/docker-linux$ podman images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
localhost/hallo-test	latest	7796ebfc2cc9	About a minute ago	169 MB
registry.fedoraproject.org/fedora	latest	b5ad17aa2a7f	7 hours ago	169 MB
docker.io/library/nginx	latest	ad5708199ec7	3 weeks ago	197 MB

1.4.5.3 Stap drie: podman container uitvoeren

Je kunt deze image nu gebruiken om containers te draaien met het docker run commando. Het maakt nu trouwens niet meer uit in welke directory je je bevindt.

```
student@serverXX:~/docker-linux$ cd
```

```
student@serverXX:~$ podman run hallo-test
```

```
Hello, World
```

```
student@serverXX:~$ cd
student@serverXX:~$ podman run hallo-test
Hello, World
student@serverXX:~$ |
```

1.5 Kubernetes

1.5.1 Inleiding

Het is niet eenvoudig om veel containers te gelijk te beheren. Daar heb je extra software voor nodig.

Er zijn hierbij 2 mogelijkheden...

Kubernetes wordt het meest gebruikt

Docker Swarm is goed voor beginners maar wordt niet veel meer gebruikt.



!Scale!



Kubernetes

1.5.2 Geschiedenis van Kubernetes

In het verleden begon Google een groot deel van het internet te domineren en veel van de technologieën die ze gebruikten, zoals de zoekmachine en Gmail, draaiden op containers. Dit was ruim voordat Docker bestond.

Google gebruikte containers op enorme schaal en ontwikkelde in-house systemen genaamd Borg en later Omega om deze operaties te beheren. Uiteindelijk creëerde Google Kubernetes, gebaseerd op Borg en Omega, en maakte het open-source. Tegenwoordig is Kubernetes een belangrijk project binnen de Cloud Native Computing Foundation (CNCF) en heeft het brede steun van grote cloudproviders en traditionele IT-leveranciers.

Kubernetes is ontworpen om een breed scala aan functionaliteiten te bieden, van stateless en stateful applicaties tot batchverwerking, beveiliging, opslag, netwerken, en zelfs machine learning. Kubernetes kan overal worden ingezet: in de cloud, on-premises, in datacenters en zelfs op laptops tijdens het ontwikkelen. Het biedt ondersteuning voor naadloze migraties tussen verschillende omgevingen, zoals tussen verschillende clouds of tussen on-premises en de cloud.

Kubernetes kan veel dingen doen: stateless, stateful, batch work, long running, security, storage, networking, serverless, ...

Sommige mensen noemen KuberneTEs K8S (keights) omdat er 8 letters staan tussen de K en de S.

1.5.3 Werking Kubernetes

VMware vCenter (gebruikt tijdens WPL2) is een centraal beheertool voor het beheren van ESXi-hypervisors en de VMs die daarop draaien.

Kubernetes is een open-source platform voor het orkestreren van containergebaseerde applicaties.



Scheduling, scaling, healing,
updating...



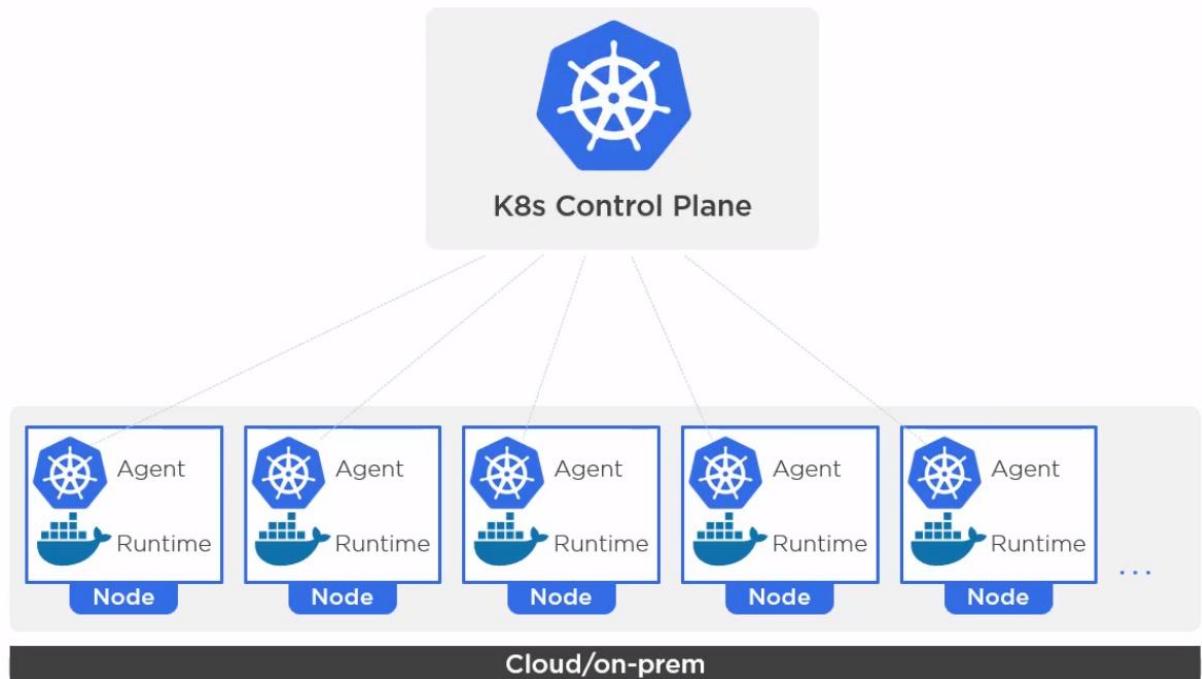
start | stop | delete ...



Kubernetes beheert containerized applicaties en richt zich op aspecten zoals hoeveel containers er moeten draaien, welke nodes gebruikt moeten worden, wanneer opschalen of afschalen noodzakelijk is en hoe updates zonder downtime uitgevoerd kunnen worden.

Kubernetes fungeert als een dirigent die orkestreert welke containers wanneer en waar moeten draaien, vergelijkbaar met hoe een dirigent een orkest leidt.

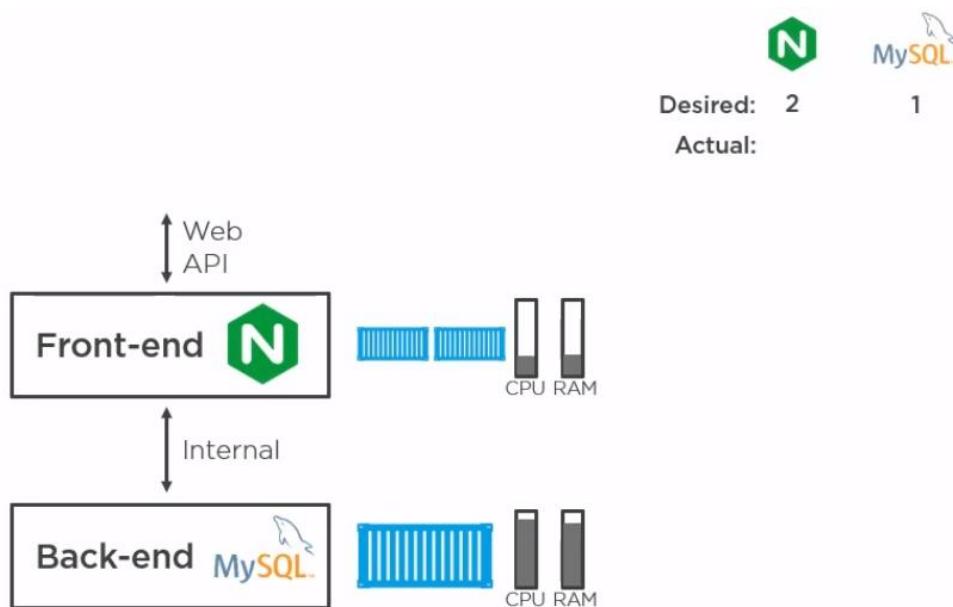
Elke node draait een container runtime zodat elke node containers kan draaien.



Een voorbeeld hiervan is een applicatie met een webfrontend (bijv. Nginx) en een persistente backend (bijv. MySQL). Kubernetes bepaalt op welke nodes deze containers draaien en kan automatisch opschalen bij toenemende belasting of terugscalen bij afname van de belasting. Dit zelfherstellende vermogen zorgt ervoor dat Kubernetes proactief reageert op veranderingen zoals uitval van nodes.

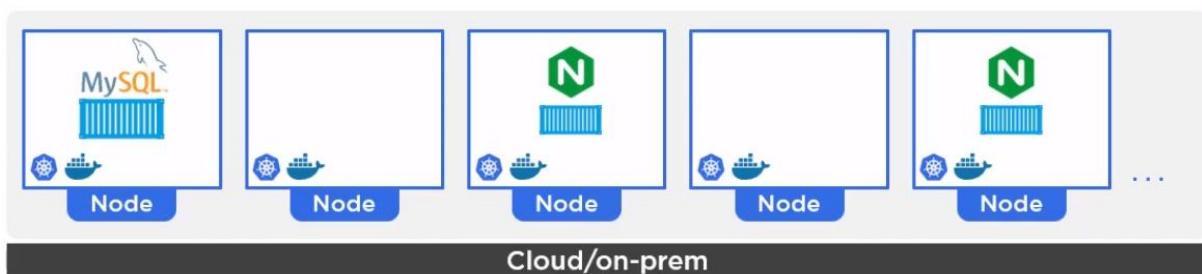
Stel bijvoorbeeld het volgende in in Kubernetes.

Front-end: 2 Kleine containers
Back-end: 1 container met veel bronnen
zoals CPU en RAM.

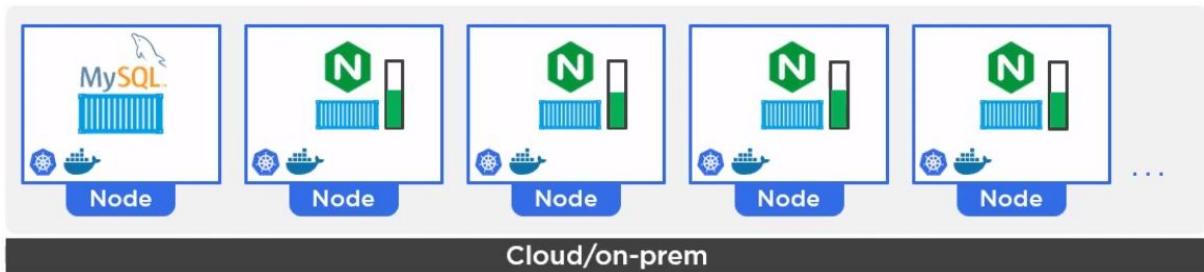


Kubernetes ontkoppelt applicaties van de onderliggende infrastructuur, waardoor het mogelijk is om applicaties eenvoudig te migreren tussen verschillende omgevingen, wat belangrijk kan zijn voor toekomstbestendige bedrijfsvoering.

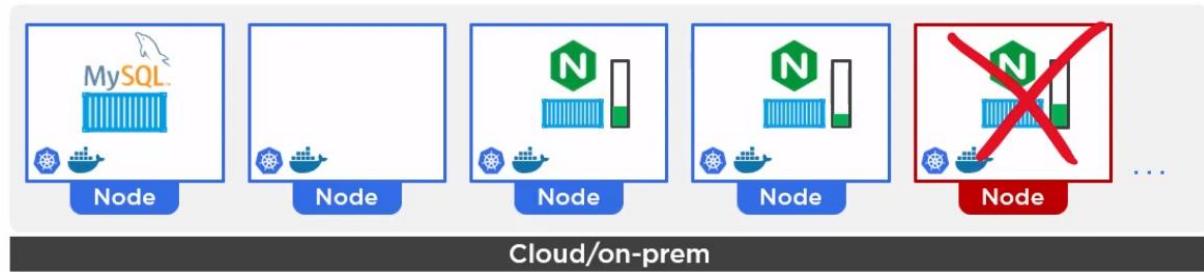
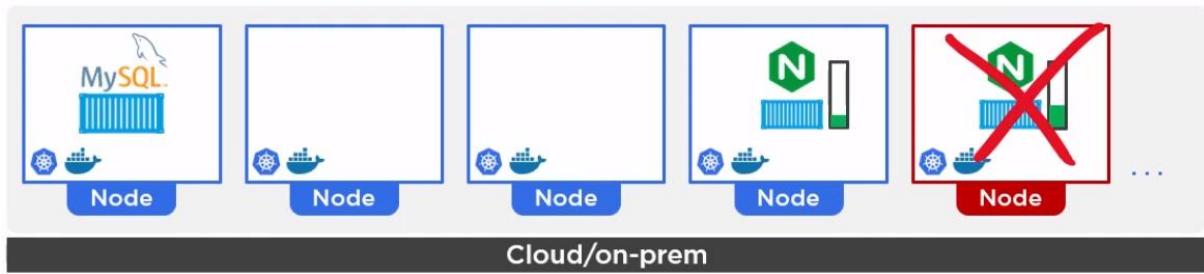
Kubernetes beslist vanaf welke nodes apps draaien en ziet er als volgt uit.



Stel dat belasting front-end toeneemt. Kubernetes voegt automatisch 2 containers toe zonder dat er een mens aan te pas komt



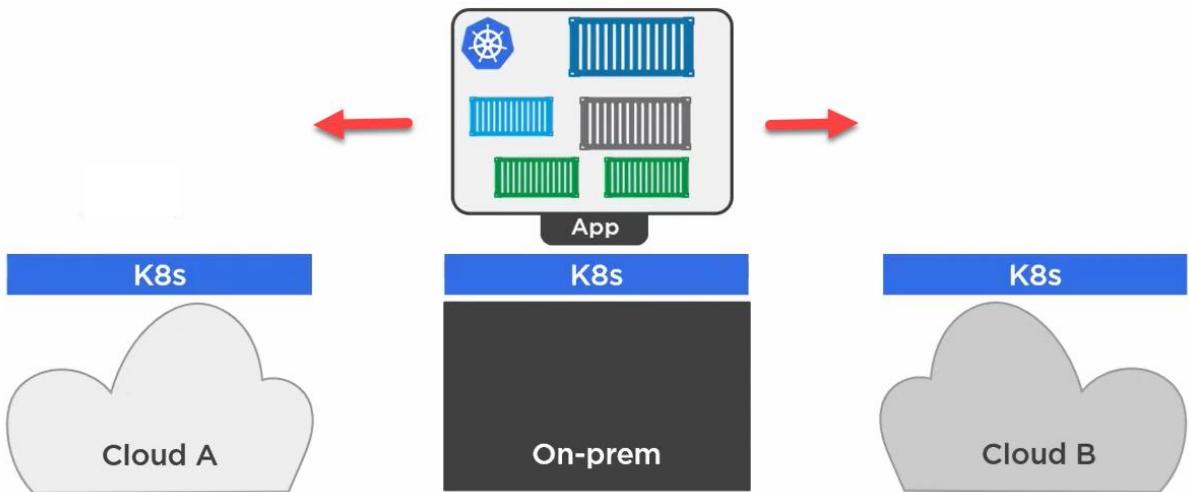
Stel dat er een node niet meer werkt, zal Kubernetes een andere node voor de container gebruiken.



Kubernetes is de absolute business voor het loskoppelen van uw applicaties van de onderliggende infrastructuur.

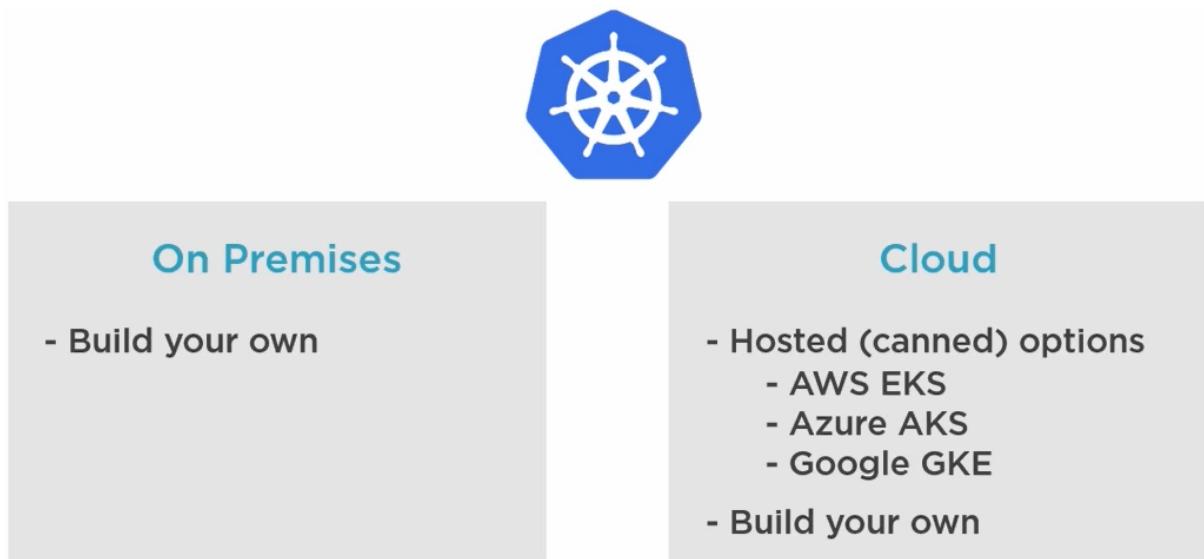
Zoals gezegd kan Kubernetes overal draaien: on-prem en in de cloud.

Als uw apps op Kubernetes draaien, is het een fluitje van een cent om ze naar en uit de cloud of zelfs van de ene cloud naar de andere te migreren (ervan uitgaande dat u uw apps niet schrijft voor een specifieke cloud).



1.5.4 Kubernetes On Prem en in de Cloud

Kubernetes is open source en geniet enorm veel steun van de community.



Alle grote cloudproviders en traditionele techbedrijven ondersteunen het, en er zijn vele innovatieve startups die erop inzetten. Net als Docker en Podman kan Kubernetes zowel on-premises als in de cloud draaien. Als je voor de cloud kiest, zijn er diverse kant-en-klare opties zoals AWS's Elastic Kubernetes Service (AWS EKS), Azure Kubernetes Service (AKS), en Google Kubernetes Engine (GKE). Je kan Kubernetes ook on-premises bouwen.

Kubernetes is veel omvangrijker dan Docker en Podman.

1.5.5 Productiegeschiktheid Kubernetes

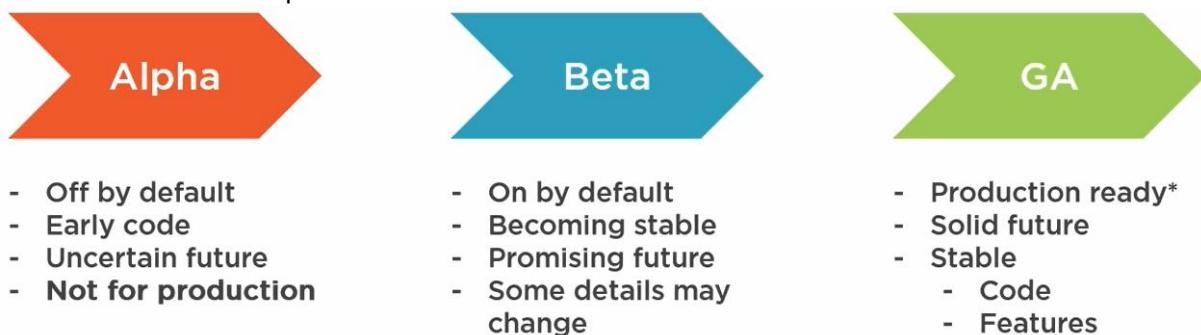
De functionaliteit is indrukwekkend, maar het bijhouden van de vele functies kan een uitdaging zijn.

Gelukkig doorlopen Kubernetes-functies gestructureerde stadia: Alpha, Beta en GA (General Availability).

Alphafuncties zijn standaard uitgeschakeld, kunnen buggy zijn en worden soms zonder waarschuwing verwijderd.

Betafuncties zijn meestal stabiel, ingeschakeld en veranderen niet zomaar, maar kunnen nog steeds evolueren.

GA is de norm voor productie: stabiel en betrouwbaar.



* Production readiness is ultimately your choice!

1.5.6 Hosted Kubernetes services

Hoewel Kubernetes open source is, blijven bedrijven zoals Google, Microsoft en IBM enorm investeren in de ontwikkeling ervan, vooral binnen hun clouddiensten. Hosted Kubernetes-diensten zijn een aanrader als je niet vastzit aan een specifieke cloud.



AWS Elastic Kubernetes Service
Azure Kubernetes Service
Google Kubernetes Engine
Others...

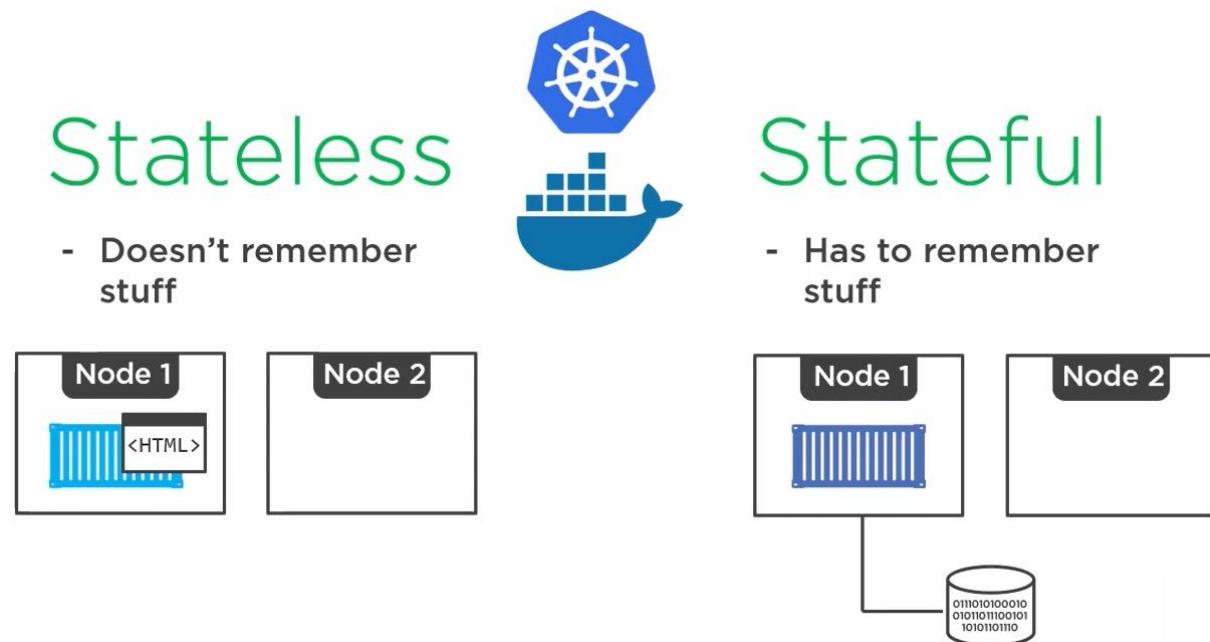
1.6 Openshift

OpenShift is een containerplatform ontwikkeld door Red Hat dat gebouwd is bovenop Kubernetes. Het helpt organisaties bij het ontwikkelen, uitrollen en beheren van containerapplicaties, zowel in de cloud als on-premises.

Waar Kubernetes de basisorchestratie van containers regelt (zoals scheduling, scaling en networking), voegt OpenShift daar extra functies aan toe zoals:

- Gebruiksvriendelijke webconsole en CLI (oc).
- Ingebouwde CI/CD pipelines voor DevOps-automatisering.
- Geïntegreerde container registry om images op te slaan en beheren.
- Uitgebreide security (striktere policies, rootless containers).
- Enterprise support.

1.7 Geschikte Workloads containers en VM's



Stateless

- Onthoudt geen data of status, bijvoorbeeld een eenvoudige webserver.
- Als de applicatie verplaatst wordt naar een andere node, maakt dat niet uit — hij heeft geen geheugen van vorige sessies of data.image.jpg

Stateful

- Onthoudt gegevens en status, zoals een database.
- Als de applicatie verplaatst wordt, moet hij toegang blijven houden tot zijn opgeslagen data, vaak via een gekoppeld opslagvolume.

Services die vaak in containers draaien:

- Webservers & API's
Bijvoorbeeld Nginx, Apache, of kleine microservices die frontend- of backend-functionaliteit aanbieden.
- Lightweight databases of caches (voor test/ontwikkel)
Bijvoorbeeld tijdelijke MySQL-instances.
(In productie worden databases vaak toch op VMs of bare metal gezet vanwege persistente opslag en performance.)
- Monitoring tools:
Bijvoorbeeld Prometheus.

Services die vaak in VM's draaien:

- Legacy applicaties:
Bijvoorbeeld oude ERP-systeem of applicaties die ontworpen zijn voor een volwaardig OS en kernel toegang nodig hebben.
- Volwaardige databases in productie:
Oracle DB, Microsoft SQL Server of grote PostgreSQL-omgevingen. Dit vooral omwille van data-integriteit, persistente opslag en sterkere isolatie.
- Windows-based workloads:
Denk aan Active Directory, Exchange servers, of andere Windows-apps die geen container images ondersteunen.

1.8 Overzicht

Hieronder vind je een overzicht van de verschillen technologieën die we hebben besproken.

- Infrastructuur (Bare metal Linux / VM / Cloud)
- Containers (bijv. Podman, Docker)
- Kubernetes (orkestratie, scheduling, scaling)
- OpenShift (security, CI/CD, webconsole, registry, support)

Uiteraard kunnen we niet alles behandelen in deze lessenreeks. De basis wordt wel aangeleerd.

2 Installatie docker en podman

2.1 Inleiding

We gaan in dit hoofdstuk uitleggen hoe je docker kan uitvoeren in een VM.

2.2 Hardware Virtualisatieondersteuning

We hebben dit reeds ingesteld in bij de OLODs Windows Essentials en Linux Essentials maar hier volgt een herhaling.

Het inschakelen van hardwarevirtualisatieondersteuning (zoals Intel VT-x of AMD-V) wordt gedaan in het BIOS/UEFI van je computer. Hieronder volgen de algemene stappen om dit in te schakelen. De exacte stappen kunnen enigszins variëren afhankelijk van het merk en model van je moederbord of computer.

Stappen om hardwarevirtualisatieondersteuning in te schakelen:

- Opstarten naar BIOS/UEFI:
 - o Start je computer opnieuw op.
 - o Tijdens het opstarten, druk herhaaldelijk op de toets om naar de BIOS/UEFI te gaan. De toets die je moet indrukken varieert, maar is vaak Delete, F2, F10, Esc, of F12. De exacte toets staat meestal kort vermeld op het opstartscherms van je computer.
- Zoek naar de Virtualisatie-instellingen:
 - o In het BIOS/UEFI-menu, zoek naar een sectie genaamd "Advanced," "CPU Configuration," "Chipset," of iets dergelijks.
 - o De optie kan variëren afhankelijk van het systeem, maar zoek naar termen zoals:
 - Intel VT-x of Intel Virtualization Technology voor Intel-processors.
 - AMD-V of SVM Mode (Secure Virtual Machine) voor AMD-processors.
- Inschakelen van de virtualisatie-optie:
 - o Selecteer de virtualisatie-optie (bijvoorbeeld Intel VT-x of AMD-V) en zet deze op Enabled.
 - o Gebruik de navigatietoetsen (meestal pijltjestoetsen) om te navigeren en de Enter-toets om te selecteren.
 - o Verander de instelling naar Enabled.
- Instellingen opslaan en afsluiten:
 - o Nadat je de instelling hebt gewijzigd, ga je naar de optie "Save & Exit," of druk op de toets die in het BIOS/UEFI wordt aangegeven om de wijzigingen op te slaan en af te sluiten (vaak F10).
 - o Bevestig dat je de wijzigingen wilt opslaan als daarom wordt gevraagd.
- Herstart de computer:
 - o De computer zal nu opnieuw opstarten met hardwarevirtualisatieondersteuning ingeschakeld.

Mogelijke problemen:

- BIOS/UEFI niet toegankelijk: Als je de BIOS/UEFI niet kunt openen, controleer dan of de toetsen correct zijn voor jouw systeem of raadpleeg de handleiding van je moederbord of computer.

- Optie niet beschikbaar: Als de optie niet beschikbaar is, controleer of je processor hardwarevirtualisatie ondersteunt. Sommige oudere of goedkopere CPU's hebben deze mogelijkheid niet. Ik denk niet dat er vanaf 2025 iemand in de klas zal zijn die dit probleem nog kan hebben.

Na het volgen van deze stappen zou hardwarevirtualisatie ingeschakeld moeten zijn, waardoor je VM's en software zoals Docker die virtualisatie gebruiken, kunt draaien op je computer.

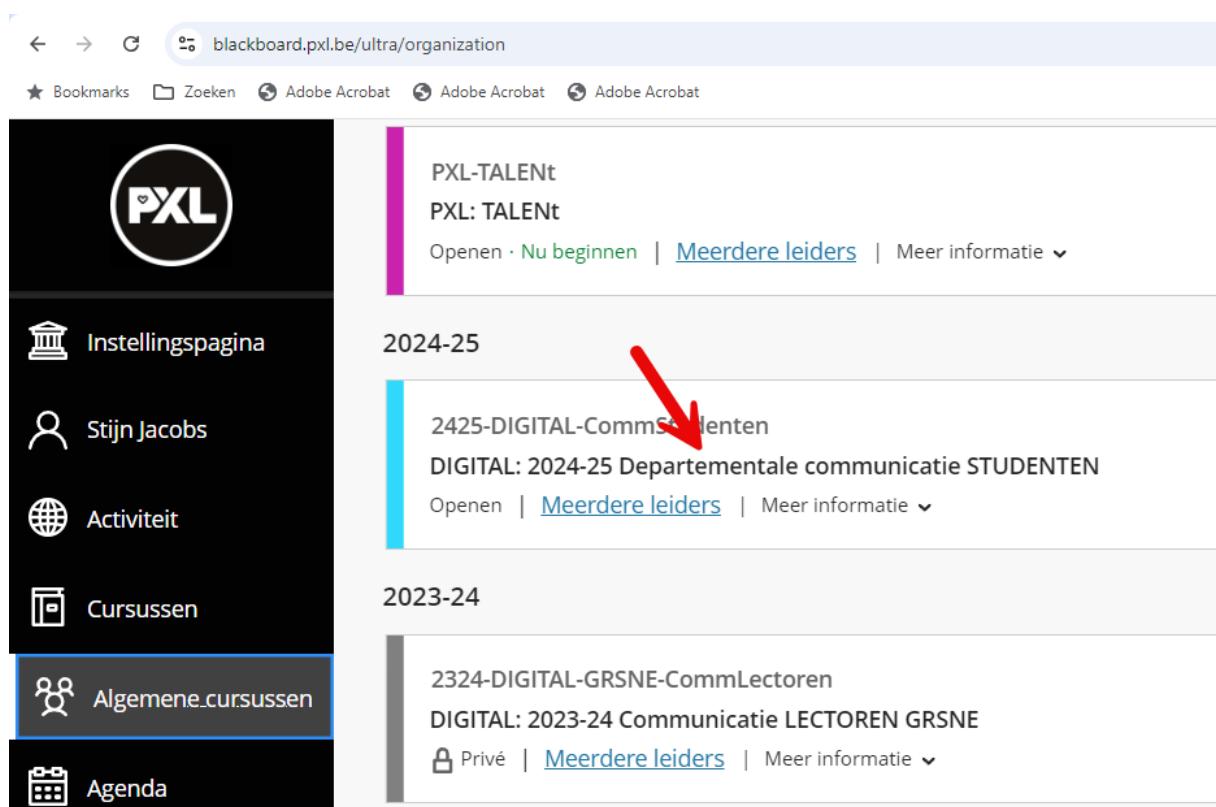
Opgelet!

Dit is niet voldoende! We gaan docker draaien in een VM (zie volgende paragraaf).

2.3 VMWare Workstation Pro/Edu installeren op host

De procedure om VMWare Workstation Pro op je computer te installeren is veranderd.

Je vindt de informatie terug op BlackBoard bij Algemene cursussen, DIGITAL: 2024-25 Departementale communicatie STUDENTEN.



The screenshot shows the BlackBoard Ultra interface. On the left is a sidebar with icons for 'Instellingspagina', 'Stijn Jacobs', 'Activiteit', 'Cursussen', 'Algemene cursussen' (which is highlighted with a blue border), and 'Agenda'. The main content area displays course information for the 2024-25 year. It shows 'PXL-TALENT' and 'PXL: TALENT' with links to 'Nu beginnen', 'Meerdere leiders', and 'Meer informatie'. Below that is a section for '2024-25' with '2425-DIGITAL-CommLectoren' and 'DIGITAL: 2024-25 Departementale communicatie STUDENTEN' with similar links. At the bottom is a section for '2023-24' with '2324-DIGITAL-GRSNE-CommLectoren' and 'DIGITAL: 2023-24 Communicatie LECTOREN GRSNE' with links. A red arrow points from the text above to the '2425-DIGITAL-CommLectoren' link.

Klik erna op onderstaande.

The screenshot shows a digital communication platform interface for the course 'DIGITAL: 2024-25 Departementale communicatie STUDENTEN'. The left sidebar includes icons for privacy, visibility, and sharing. The main content area displays course information and a list of resources:

- Organisatie-inhoud** (Organization Content):
 - IOPGELET! Klik EERST éénmalig in de menuregel bovenaan op "Groepen" en daarna op Deelnemen achter jouw groep! Dan zie je al jouw info! Klik hier voor meer info...
 - Zichtbaar voor deelnemers
 - ICT** (highlighted with a red arrow)
 - Zichtbaar voor deelnemers
 - ICT Algemeen** (highlighted with a red arrow)
 - Zichtbaar voor deelnemers
 - Handleidingen bij de ICT-voorzieningen van Hogeschool PXL.
 - ICT Cluster IT**
 - Vrijgavevoorraarden [Vrijgavevoorraarden bewerken](#)
 - Handleiding bij de ICT-voorzieningen en softwarepakketten nodig in functie van je specifieke opleiding.
 - VMware Workstation Pro (of Fusion) for Education** (highlighted with a red arrow)
 - Zichtbaar voor deelnemers
 - Azure portal (Microsoft producten downloaden)**
 - Zichtbaar voor deelnemers

Cursusleiders voor deze algemene cursus

- Veerle Asaert **LEIDER**
- Wim Bervoets **LEIDER**

Meer weergeven

Details en acties

- Naamlijst [lederen in je organisatie weergeven](#)
- Voortgang volgen [inschakelen](#)
- Afbeelding van organisatie [Weergave-instellingen bewerken](#)
- Organisatie is open [Deelnemers hebben toegang tot deze organisatie](#)
- Aanwezigheid [Markeer aanwezigheid](#)
- Boeken en tools [Tools van organisatie en instelling weergeven](#)
- Vragenbanken [Banken beheren](#)

Volg deze informatie

Opgellet

Kies voor versie 17.5.2 indien de installatie van een hogere versie mislukt.

2.4 Installeer VM's

Gebruik bijlage A om de VM's in te stellen!

Samenvattend:

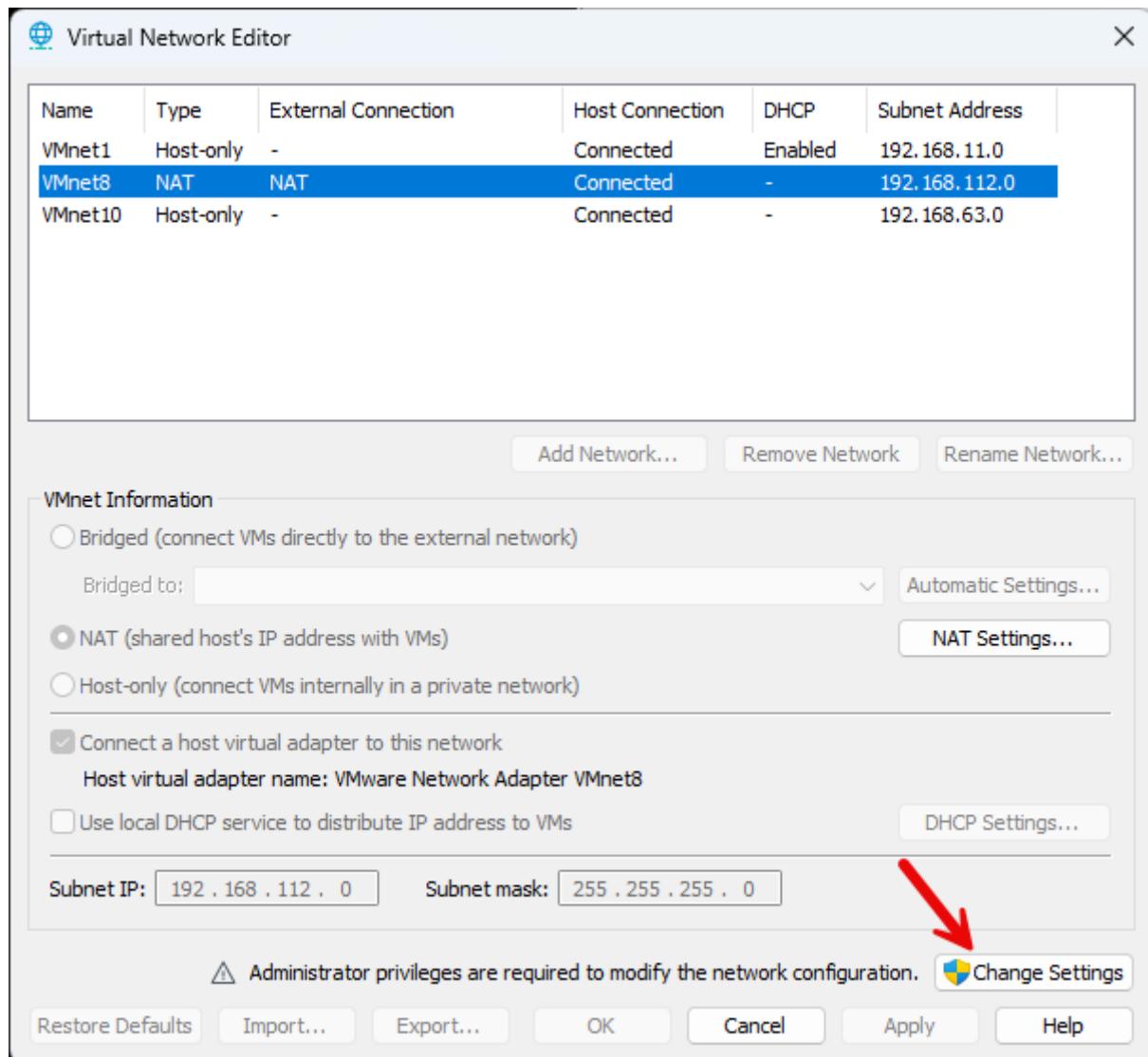
2 VM's met een zuivere installatie van RHEL 10 in Vmware Workstation op je Windows host met 1 netwerkkaart die via NAT verbonden is in VMWare met het internet.

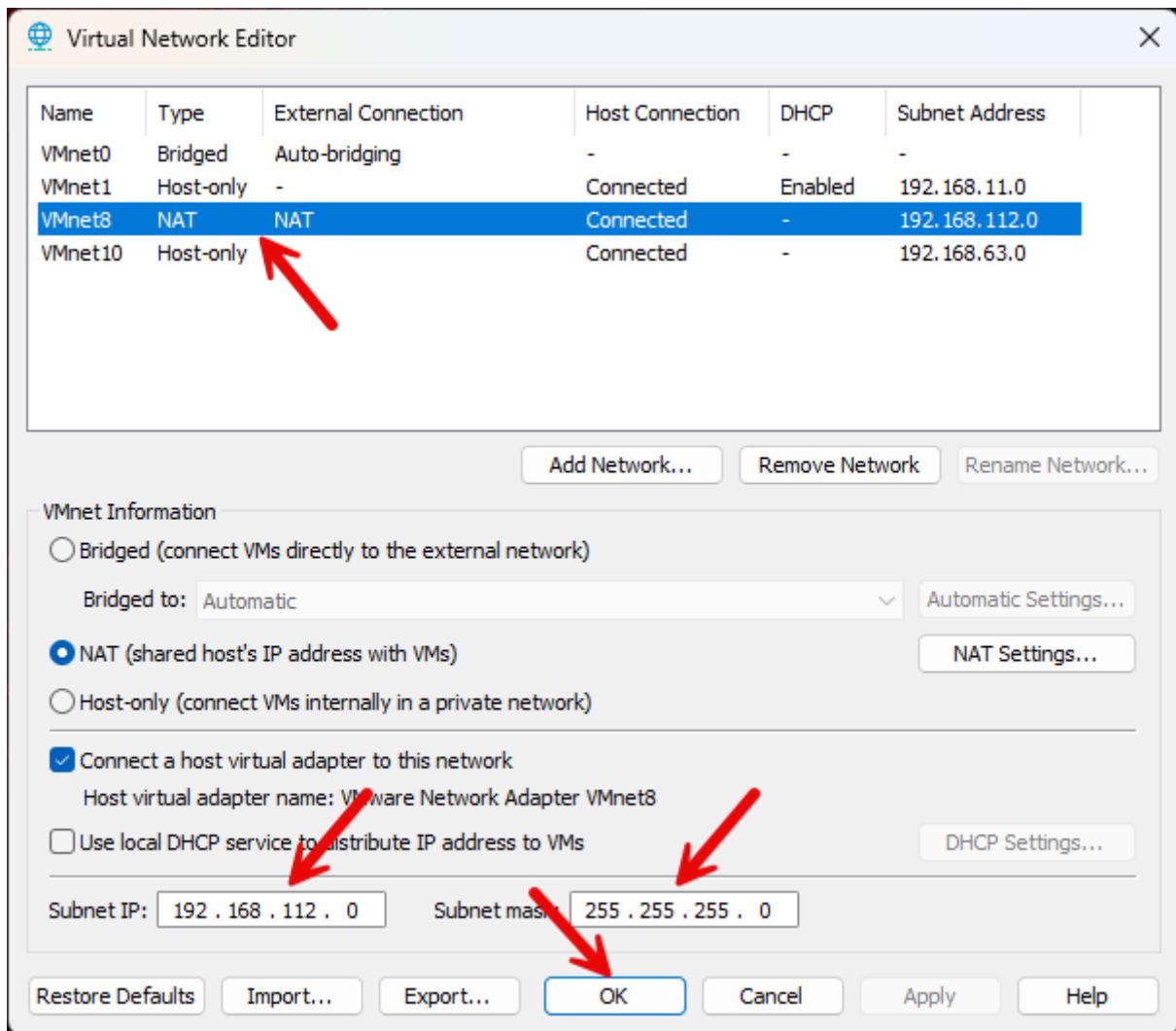
- 1^{ste} VM
 - o Naam: server[je_initialen]
 - o IP-nummer: 192.168.112.100
 - o gebruiker student met sudo-rechten
- 2^{de} VM
 - o Naam: client[je_initialen]
 - o IP-nummer: 192.168.112.200

- gebruiker student met sudo-rechten
- Registreer de VM's bij RedHat zoals je gewend bent.
- Kies voor een schijf van minimaal 100 GB voor elke VM.

Hou deze VM's altijd bij!

Let ook op dat je de juiste instellingen hebt bij de netwerkinstellingen (Edit, Virtual Network Editor) voor NAT in VMWare Workstation Pro!





2.5 Docker vs Podman

Zoals reeds besproken is Podman een daemonless, veiliger alternatief voor Docker. Het gebruikt dezelfde containerstandaarden en heeft geen centrale daemon: containers draaien als gewone processen, direct onder de gebruiker die ze start. Daardoor is Podman beter geschikt voor "rootless" werken (veiliger), en eenvoudiger te integreren in systemen die geen centrale service willen.

We zullen zowel docker als podman leren installeren in RHEL 10.

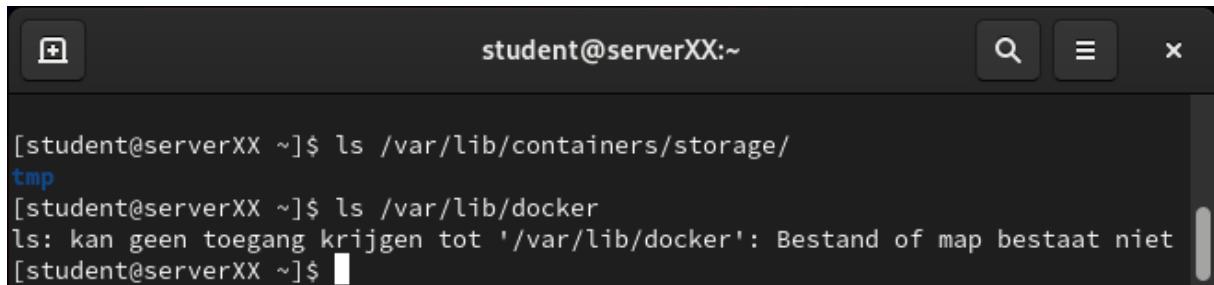
2.6 Parallel installeren

Docker en Podman kunnen naast elkaar op hetzelfde systeem staan, omdat Podman geen daemon gebruikt en containers als gewone processen draait.

Opgelet:

- Docker en Podman hebben hun eigen storage en netwerk namespaces hebben (tenzij je expliciet dezelfde instellingen forceert).
- Docker gebruikt standaard /var/lib/docker, Podman gebruikt /var/lib/containers/storage.

Na installatie van RHEL10 zal je zien dat /var/lib/containers/storage bestaat en /var/lib/docker niet. Logisch, podman is standaard geïnstalleerd, docker niet.



```
[student@serverXX ~]$ ls /var/lib/containers/storage/
tmp
[student@serverXX ~]$ ls /var/lib/docker
ls: kan geen toegang krijgen tot '/var/lib/docker': Bestand of map bestaat niet
[student@serverXX ~]$
```

2.7 Podman installeren

2.7.1 Installatie zelf

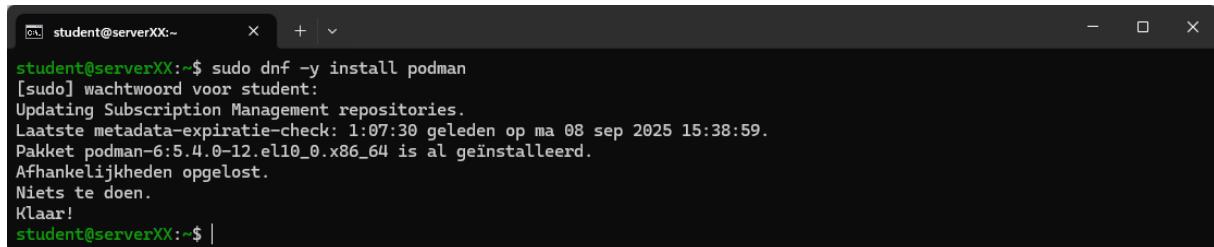
RedHat promoot Podman als de standaard containertechnologie vanaf RHEL 8.

Het is out-of-the-box ondersteund en volledig geïntegreerd (ook in documentatie en support: [Red Hat Enterprise Linux 9 Building, running, and managing containers](#)).

Podman is volledig compatible met Docker images en Docker CLI (je kunt zelfs “alias docker=podman” gebruiken).

Installatie gebeurt als volgt:

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



```
student@serverXX:~$ sudo dnf -y install podman
[sudo] wachtwoord voor student:
Updating Subscription Management repositories.
Laatste metadata-expiratie-check: 1:07:30 geleden op ma 08 sep 2025 15:38:59.
Pakket podman-6:5.4.0-12.el10_0.x86_64 is al geïnstalleerd.
Afhankelijkheden opgelost.
Niets te doen.
Klaar!
student@serverXX:~$ |
```

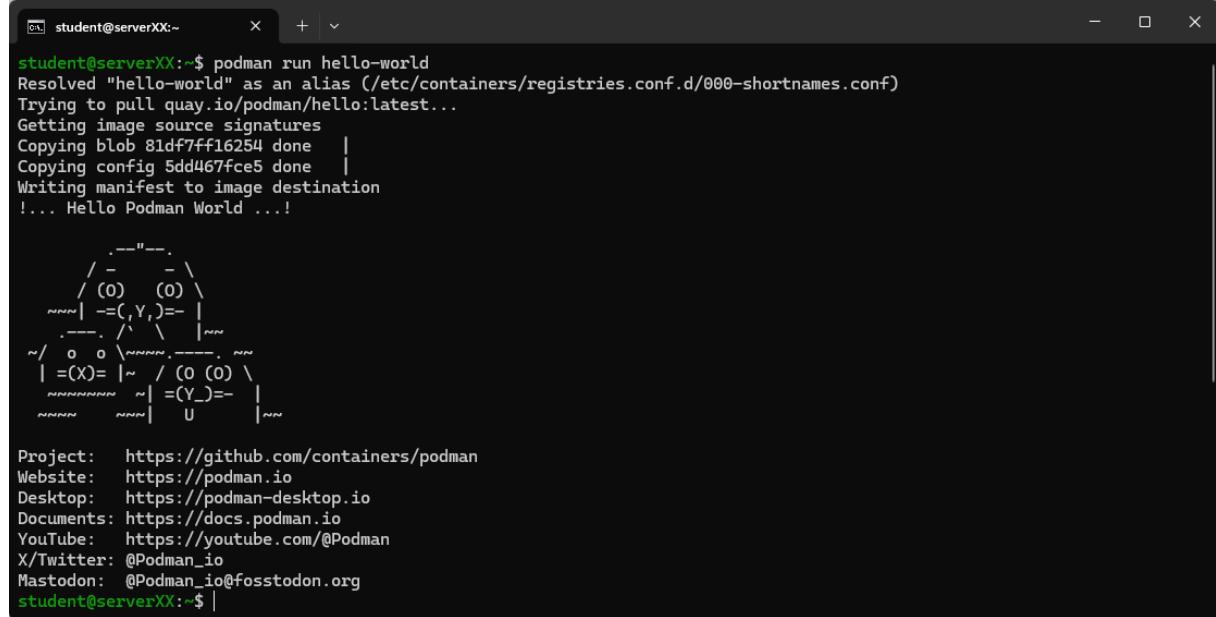
Zoals je ziet is podman op mijn systeem reeds geïnstalleerd.

2.7.2 Controle installatie

Voer hiervoor uit:

```
student@serverXX:~$ podman run hello-world
```

Als je onderstaande krijgt, is podman correct geïnstalleerd en functioneel.



```
student@serverXX:~$ podman run hello-world
Resolved "hello-world" as an alias (/etc/containers/registries.conf.d/000-shortnames.conf)
Trying to pull quay.io/podman/hello:latest...
Getting image source signatures
Copying blob 81df7ff16254 done    |
Copying config 5dd467fce5 done    |
Writing manifest to image destination
!... Hello Podman World ...!

          .--"--
         /   \ 
        (o)  (o) \
       ~~~| -=(,v,)=- |
       .---. / \ | ~
      / \ o o \~~~.----. ~~
     | = (X) = | ~ / (o (o) \
     ~~~~~~ ~| = (Y,) =- |
     ~~~~ ~~~| U | ~

Project: https://github.com/containers/podman
Website: https://podman.io
Desktop: https://podman-desktop.io
Documents: https://docs.podman.io
YouTube: https://youtube.com/@Podman
X/Twitter: @Podman_io
Mastodon: @Podman_io@fosstodon.org
student@serverXX:~$ |
```

2.8 Docker installeren

2.8.1 Installatie zelf

Docker wordt momenteel niet officieel ondersteund voor RHEL 10 volgens de officiële Docker-website: [RHEL | Docker Docs..](#)

Prerequisites

To install Docker Desktop successfully, you must:

- Meet the [general system requirements](#).
- Have a 64-bit version of either [RHEL 8 or RHEL 9](#)
- If `pass` is not installed, or it can't be installed, you must enable [CodeReady Linux Builder \(CRB\)](#) repository and [Extra Packages for Enterprise Linux \(EPEL\)](#).

Er is wel een oplossing maar voor een productieomgeving is aldus RHEL 9 aan te raden.

Hieronder staan de stappen beknopt beschreven waarmee het gaat (gebaseerd op Centos Stream: <https://docs.docker.com/engine/install/centos/>).

- Repository toevoegen:

```
student@serverXX:~$ sudo dnf config-manager --add-repo
https://download.docker.com/linux/centos/docker-ce.repo
```

- Docker packages installeren

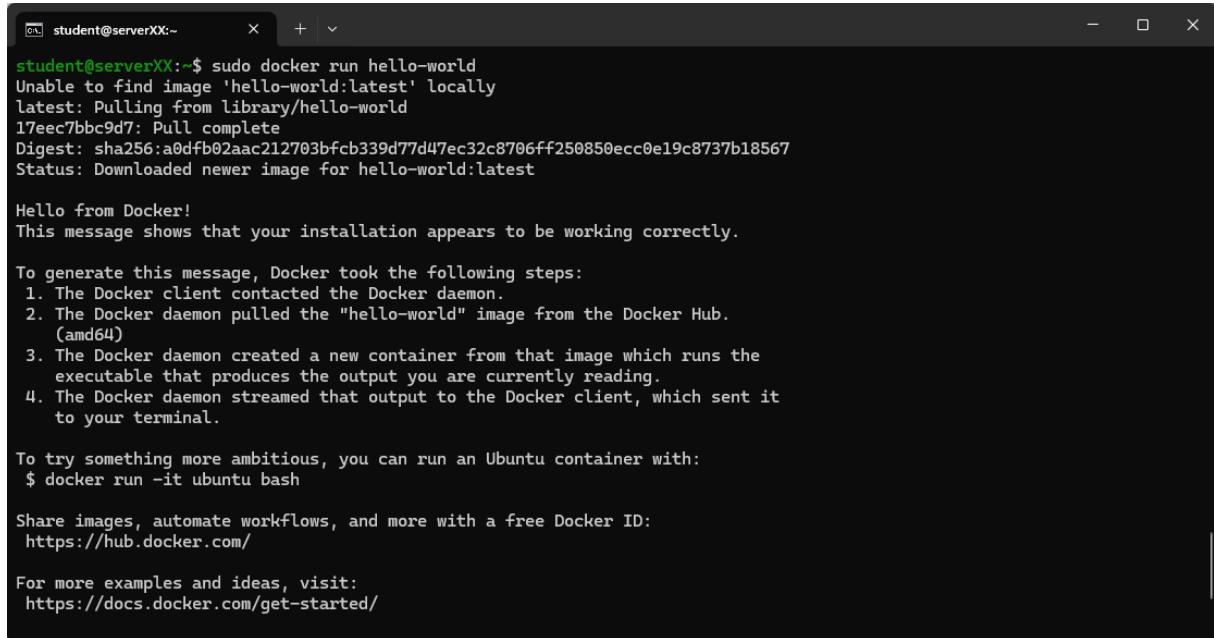
```
student@serverXX:~$ sudo dnf install -y docker-ce docker-ce-cli  
containerd.io docker-buildx-plugin docker-compose-plugin  
- Docker service activeren  
student@serverXX:~$ sudo systemctl enable --now docker
```

2.8.2 Controle

Voer hiervoor uit

```
student@serverXX:~$ sudo docker run hello-world
```

Als je onderstaande krijgt werkt het.



The screenshot shows a terminal window with the following text:

```
student@serverXX:~$ sudo docker run hello-world  
Unable to find image 'hello-world:latest' locally  
latest: Pulling from library/hello-world  
17eec7bbc9d7: Pull complete  
Digest: sha256:a0dfb02aac212703bfcb339d77d47ec32c8706ff250850ecc0e19c8737b18567  
Status: Downloaded newer image for hello-world:latest  
  
Hello from Docker!  
This message shows that your installation appears to be working correctly.  
  
To generate this message, Docker took the following steps:  
1. The Docker client contacted the Docker daemon.  
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.  
(amd64)  
3. The Docker daemon created a new container from that image which runs the  
executable that produces the output you are currently reading.  
4. The Docker daemon streamed that output to the Docker client, which sent it  
to your terminal.  
  
To try something more ambitious, you can run an Ubuntu container with:  
$ docker run -it ubuntu bash  
  
Share images, automate workflows, and more with a free Docker ID:  
https://hub.docker.com/  
  
For more examples and ideas, visit:  
https://docs.docker.com/get-started/
```

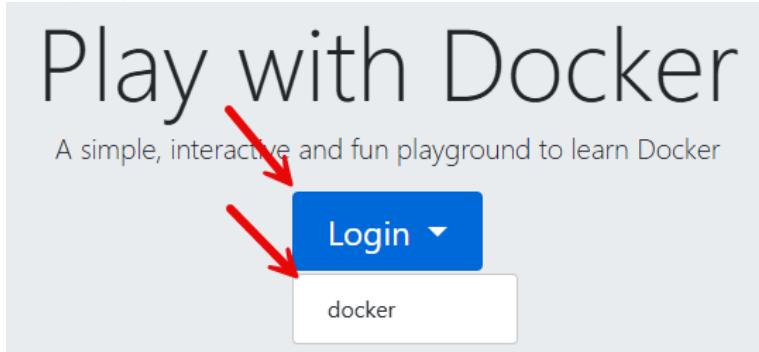
Let op: sudo is hiervoor nodig!

2.9 Play with Docker

Play with Docker is een gratis online omgeving waarmee je Docker kunt uitproberen zonder dat je iets hoeft te installeren op je eigen computer. Het biedt een sandbox-achtige omgeving waar je virtuele Linux Docker-instanties (containers) kunt draaien en beheren in je browser.

- Surf naar op labs.play-with-docker.com.

- Klik op Login en kies voor docker.



- Je bekomt nu onderstaand venster.



Sign in

Using Docker for work? We recommend signing in with your work email address.

Username or email address*

Continue

OR

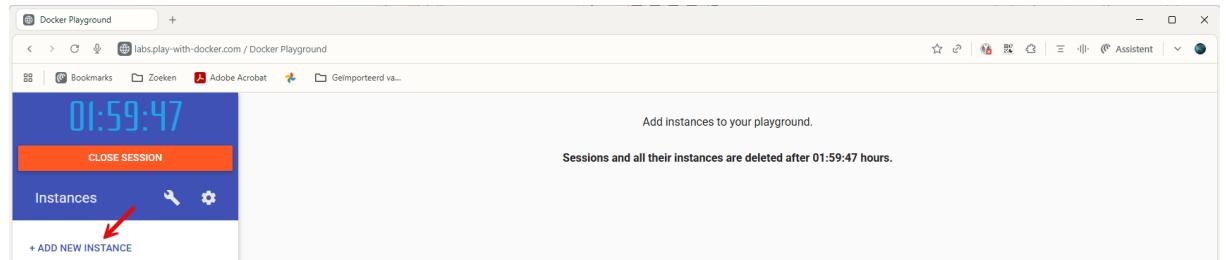
 Continue with Google

 Continue with GitHub

 Continue with SSO

Don't have an account? [Sign Up](#)

- Log in met je Docker Hub-account of één van de alternatieve methodes: Google, GitHub of SSO.
- Na het inloggen kun je een nieuwe sessie starten. Dit creëert een virtuele machine waarop je Docker 2 uur kunt gebruiken.



- Na inloggen kan kan je docker gebruiken.

```

d3a3je46_d3a3jn469qi000armrrg
IP: 192.168.0.14 OPEN PORT
Memory: 1.13% (45.39MB / 3.906GB) CPU: 0.09%
SSH: ssh ip172-18-0-20-d3a3je469qi000armrrg@direct.labs.play-with-docker.com
DELETE EDITOR

#####
# WARNING!!!!#
# This is a sandbox environment. Using personal credentials #
# is HIGHLY! discouraged. Any consequences of doing so are #
# completely the user's responsibilities. #
# The FWD team. #
#####
[node1] (local) root@192.168.0.14 ~
$ docker run hello-world
latest: Pulling from library/hello-world
17ee07bbc9d7: Pull complete
Digest: sha256:54e66cc1d1fcb1c13c58bd8017914dbed8701e2d8c74d9262e26bd9cc1642d31
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

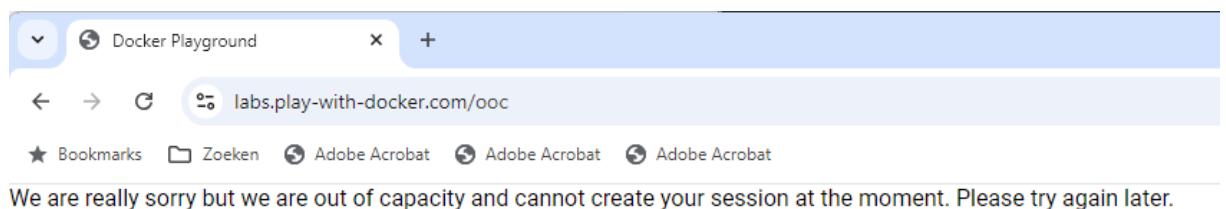
To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
[node1] (local) root@192.168.0.14 ~
$ 

```

Opgelet: de site kan overbelast zijn zoals hieronder staat aangegeven.



We zullen hier verder geen gebruik van maken.

3 Beheer vanop afstand

3.1.1 Verbinding maken met je Server vanaf je hostcomputer

3.1.2 Let op met IP-nummer!

We veranderen het IP-adres van ServerXX naar 192.168.112.1.

We vragen eerst de huidige connectienaam op.

```
student@serverXX:~$ sudo nmcli con show

[sudo] wachtwoord voor student:

Warning: nmcli (1.52.0) and NetworkManager (1.48.10) versions don't match.
Restarting NetworkManager is advised.

NAME      UUID                                  TYPE      DEVICE
ens160    8f18258e-ee43-3d8a-8ed6-b452b3bc8045  ethernet  ens160
docker0   22924712-c569-41ed-b3d2-3321eff411a  bridge    docker0
lo        4b771ea1-a470-470f-84ed-a6ef80af6342  loopback  lo
```

Zoals je ziet is de connectienaam ens160 aangezien docker0 te maken heeft met docker en lo de loopback is.

We stellen nu de nieuwe IP-nummer in.

```
student@serverXX:~$ sudo nmcli con mod ens160 ipv4.addresses 192.168.112.1/24
```

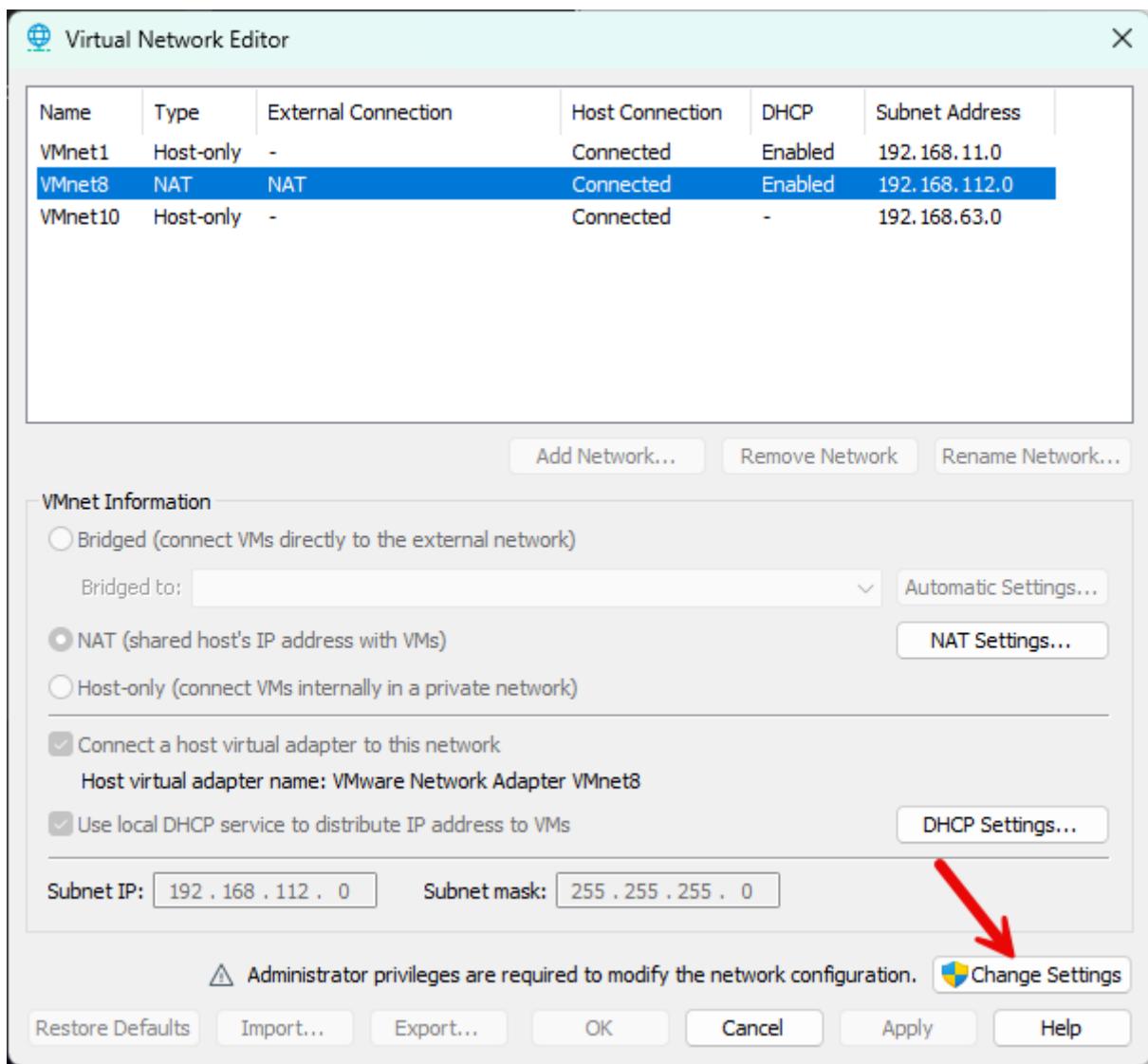
Herstart nu de netwerkverbinding.

```
student@serverXX:~$ sudo nmcli con down ens160 && sudo nmcli con up ens160
...
Fout: Activeren van verbinding is mislukt: Reserveren van de IP-configuratie is
mislukt (geen beschikbare adressen, time-out, enz.)
Hint: use 'journalctl -xe NM_CONNECTION=2318b781-c803-36d2-8642-bb3d5bb513f9 +
NM_DEVICE=ens160' to get more details.
```

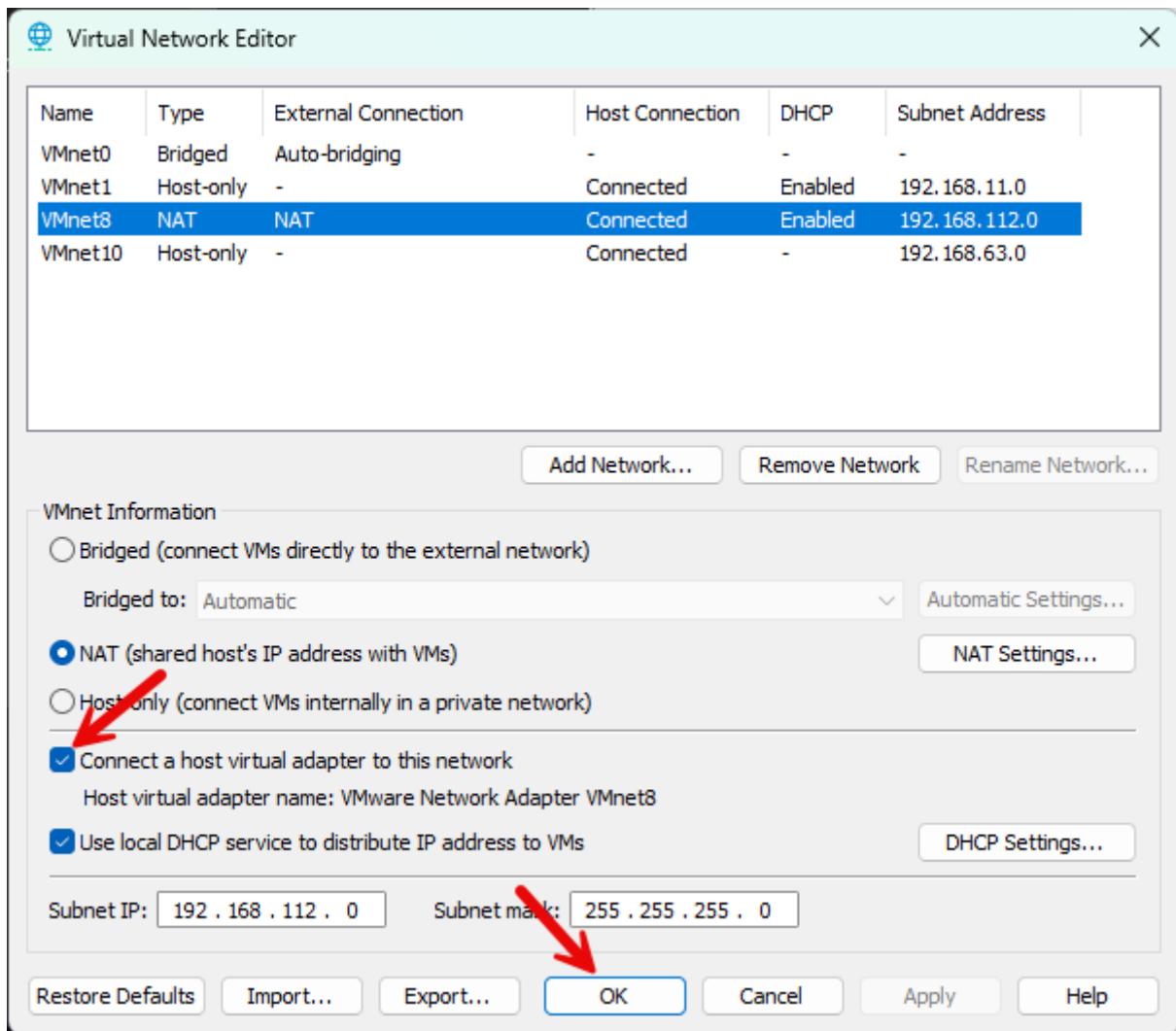
3.1.3 Oorzaak probleem

Er is een IP-conflict.

Klik daarom op Edit, Virtual Network Editor... en selecteer NAT. Klik erna rechtsonder op Change Settings en klik op Ja.

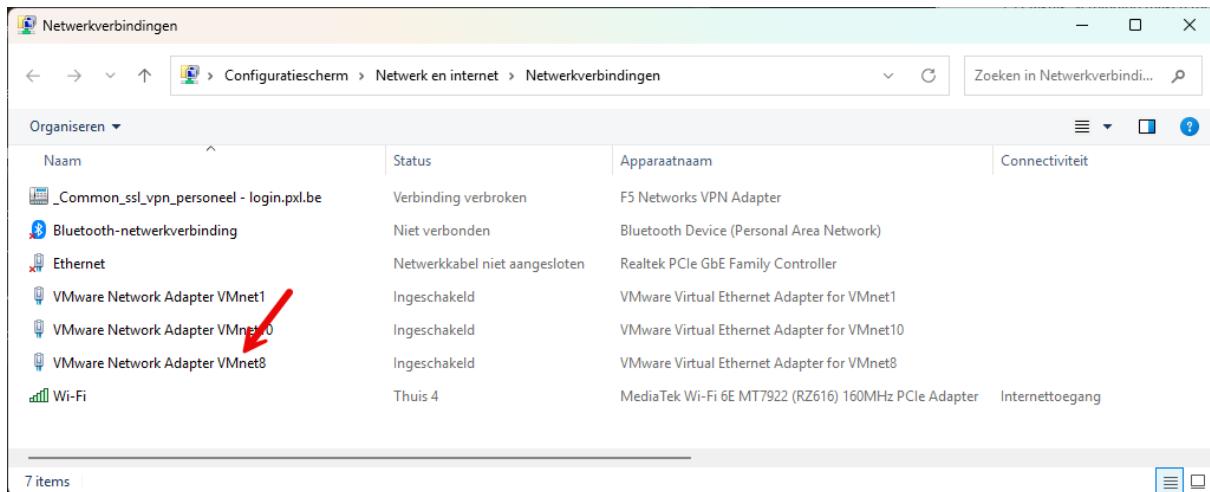


Selecteer NAT en zet dat er een vinkje staat voor Connect a host virtual adapter to this network (we hebben dat zo ingesteld tijdens instructies bijlage A).

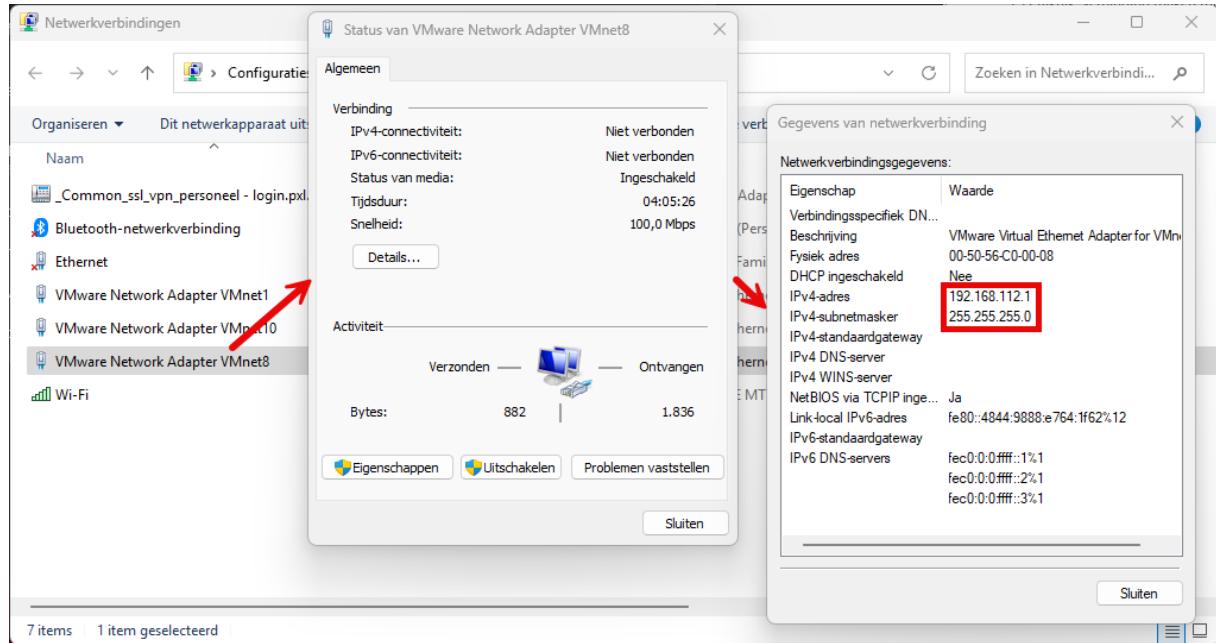


Klik erna op OK.

Ga nu naar de netwerkverbindingen op je host (Windows 11 computer normaliter).



We bekijken het IP-nummer van deze virtuele netwerkkaart zoals hieronder grafisch staat aangegeven.



Zoals je weet is dat ook het IP-nummer die ingesteld is voor de server! Hier zit dus het probleem.

3.1.4 Probleem oplossen

We vragen het IP-nummer van ens160 op.

```
student@serverXX:~$ nmcli -g IP4.ADDRESS dev show ens160
```

Je ziet dat je geen antwoord krijgt... Dat is omdat het IP-adres niet toegewezen kan worden omdat van het conflict.

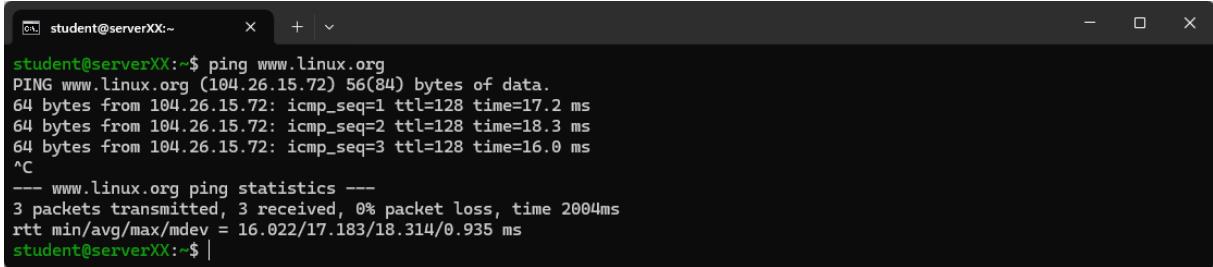
We zullen ens160 op de server terug het IP-adres 192.168.112.100/24 geven om het op te lossen.

```
student@serverXX:~$ sudo nmcli con mod ens160 ipv4.addresses 192.168.112.100/24
```

```
student@serverXX:~$ sudo nmcli con up ens160
```

```
Verbinding is met succes geactiveerd (actief D-Bus-pad:  
/org/freedesktop/NetworkManager/ActiveConnection/6)
```

Vanaf je server kan je nu terug op het internet.



```
student@serverXX:~$ ping www.linux.org
PING www.linux.org (104.26.15.72) 56(84) bytes of data.
64 bytes from 104.26.15.72: icmp_seq=1 ttl=128 time=17.2 ms
64 bytes from 104.26.15.72: icmp_seq=2 ttl=128 time=18.3 ms
64 bytes from 104.26.15.72: icmp_seq=3 ttl=128 time=16.0 ms
^C
--- www.linux.org ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 16.022/17.183/18.314/0.935 ms
student@serverXX:~$ |
```

Ping nu vanaf de host naar je server.... Ook dit gaat nu.



3.2 SSH

3.2.1 Inleiding

SSH maakt het mogelijk om op afstand een SSH-sessie uit te voeren op een andere computer. Je kan via SSH opdrachten uitvoeren, bestanden kopiëren, en interactieve shells openen. Dit is reeds behandeld bij Linux Advanced maar we herhalen het hier beknopt.

3.2.2 SSH-server installeren en configureren

Dit is reeds uitvoerig besproken in Linux Advanced maar wordt hier beknopt herhaald.

De SSH-server is al geïnstalleerd als je bijlage A hebt gevolgd maar hier staat hoe je het kan doen als je niet de nodige pakketten geïnstalleerd hebt.

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

Start de SSH-service op.

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

Start de SSH-service op tijdens opstarten van het systeem.

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

Stel de firewall in zodat er een gat gemaakt wordt voor SSH (ook na opstarten van het systeem).

```
student@serverXX:~$ sudo firewall-cmd --permanent --add-service=ssh
```

```
student@serverXX:~$ sudo firewall-cmd --reload
```

3.2.3 Verbinding maken met SSH-server

We maken vanaf de Linux-client verbinding met server<jeinitialen>.

```
[student@clientXX ~]$ ssh student@192.168.112.100
```

```
The authenticity of host '192.168.112.100 (192.168.112.100)' can't be established.
```

```
ED25519 key fingerprint is SHA256:F1750QXtB+sxrh3GrCCpXqTM6MZ+uzzmXhXFbmRDD8.
```

```
This key is not known by any other names
```

```
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes # eerste keer
```

```
Warning: Permanently added '192.168.112.100' (ED25519) to the list of known hosts.
```

```
student@192.168.112.100's password: # geef wachtwoord in van student op 192.168.112.100
```

```
...
```

```
student@serverXX:~$
```

Door exit in te geven ga je terug naar de client.

```
student@serverXX:~$ exit
```

```
uitgelogd
```

```
Connection to 192.168.112.100 closed.
```

```
[student@clientXX ~]$
```

Je kan, zoals ook reeds gezien bij Linux Advanced, ook gebruik maken van een sleutelpaar.

- Maak dat aan op de client.

```
[student@clientXX ~]$ ssh-keygen
```

```
Generating public/private rsa key pair.
```

```
Enter file in which to save the key (/home/student/.ssh/id_rsa): <Enter>
```

```
Enter passphrase (empty for no passphrase): <Enter>
```

```
Enter same passphrase again: <Enter>
```

```
Your identification has been saved in /home/student/.ssh/id_rsa
```

```
Your public key has been saved in /home/student/.ssh/id_rsa.pub
```

```
The key fingerprint is:
```

```
SHA256:NkwuIpRJav9p2errYVPAdfkc72Kd3FRzh3eC8qX0Fnk student@clientXX
```

```
The key's randomart image is:
```

```
+---[RSA 3072]---
```

```
| . . . . . |
```

```
| o o. . . . +.*|
```

```
| ..+ o . ooo+ o*|
```

```
| ... .+ o=.E. |
```

```
| ... .S o+o+ |
```

```
| ...=o . o+= . |
```

```
| 0 . ... |
```

```
| o + |
```

```
| o=. |
```

```
+---[SHA256]---
```

- Kopieer erna de publieke sleutel naar de account op je server waar je zonder wachtwoord verbinding mee wil maken.

```
[student@clientXX ~]$ ssh-copy-id student@192.168.112.100
```

```
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: ssh-add -L
```

```
The authenticity of host '192.168.112.100 (192.168.112.100)' can't be established.
```

```
ED25519 key fingerprint is SHA256:ZP8PS4Cs2UGaLLEyFunY/Y3C8X/qRzF8354RSj/upRs.
```

```
This key is not known by any other names.
```

```
Are you sure you want to continue connecting (yes/no/[fingerprint])? <yes>
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter
out any that are already installed

/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are
prompted now it is to install the new keys

student@192.168.112.100's password: <Geef wachtwoord in>
```

Number of key(s) added: 1

Now try logging into the machine, with: "ssh 'student@192.168.112.100'"
and check to make sure that only the key(s) you wanted were added.

Je kan nu verbinding maken zonder wachtwoord.

```
[student@clientXX ~]$ ssh student@192.168.112.100
...
student@serverXX:~$
```

3.3 Remote Docker

3.3.1 Remote docker via opstellen Docker-daemon

In dit onderdeel bespreken we twee methoden om de Docker-daemon extern toegankelijk te maken. De eerste methode is relatief eenvoudig, maar brengt belangrijke beveiligingsrisico's met zich mee: het openstellen van de Docker-daemon via een TCP-poort. Hierdoor kunnen externe clients verbinding maken en opdrachten naar de Docker-daemon sturen.

Het grote nadeel van deze methode is dat iedereen met toegang tot deze poort potentieel commando's kan uitvoeren, zelfs zonder beheerdersrechten. Dit kan leiden tot ernstige beveiligingsproblemen.

Een nuttige bron op dit gebied is de website "[Protect the Docker daemon socket](#)".

Hoewel het mogelijk is om de verbinding te beveiligen met TLS-certificaten, behandelen we dat hier niet.

Deze configuratie maakt gebruik van poort 2375 zonder TLS, en is niet veilig voor productieomgevingen en gebruiken we hier uitsluitend voor demonstratie- of testdoeleinden. In productie wordt uiteraard altijd aangeraden om TLS (poort 2376) te gebruiken.

We stellen eerst in dat Docker ook luistert op TCP-poort 2375 naar aanvragen zonder TLS.

```
student@serverXX:~$ sudo nano /etc/docker/daemon.json
```

```
{
    "hosts": ["tcp://0.0.0.0:2376", "unix:///var/run/docker.sock"]
}
```

```
student@student@serverXX:~$ nano /etc/docker/daemon.json
{
    "hosts": ["tcp://0.0.0.0:2376", "unix:///var/run/docker.sock"]
}
student@student@clientXX:~$
```

Zoals je ziet wordt er geluisterd naar:

- tcp://0.0.0.0:2376
Hier stel je in dat er geluisterd wordt op poort 2376.
- unix:///var/run/docker.sock

Hiermee stel je in dat Docker CLI (client) verbinding maakt met de Docker-daemon (server). Aangezien je nu in een apart bestand hebt ingesteld waarnaar geluisterd wordt dien je dit te verwijderen uit de systemd service unit file.

```
student@student@serverXX:~$ sudo nano /usr/lib/systemd/system/docker.service
```

...

```
ExecStart=/usr/bin/dockerd -H fd:// --
containerd=/run/containerd/containerd.sock
```

...

Pas deze regel aan naar onderstaande.

```
ExecStart=/usr/bin/dockerd --containerd=/run/containerd/containerd.sock
```

-H fd:// betekent eigenlijk: “gebruik de socket die systemd al voor mij heeft klaargezet”.

Docker hoeft dan zelf geen nieuwe socket aan te maken, maar neemt de file descriptor (de verwijzing naar dat socket-bestand) over van systemd.

Herlaad systemd en herstart Docker.

```
student@student@serverXX:~$ sudo systemctl daemon-reload
student@student@serverXX:~$ sudo systemctl restart docker
```

Erna kan je uiteraard de status controleren.

```
student@student@serverXX:~$ sudo systemctl status docker
```

Stel nu de firewall in.

```
student@student@serverXX:~$ sudo firewall-cmd --permanent --add-port=2376/tcp
```

```
student@serverXX:~$ sudo firewall-cmd --reload
```

Nu kan Docker TCP-verkeer ontvangen op poort 2376.

We kunnen nu op afstand (clientXX) verbinding maken met de Docker-host en commando's uitvoeren.
Op je client moet uiteraard wel de docker-client geïnstalleerd zijn.

We voegen hiervoor eerst de repository van docker zelf toe.

```
[student@clientXX ~]$ sudo dnf config-manager --add-repo  
https://download.docker.com/linux/centos/docker-ce.repo
```

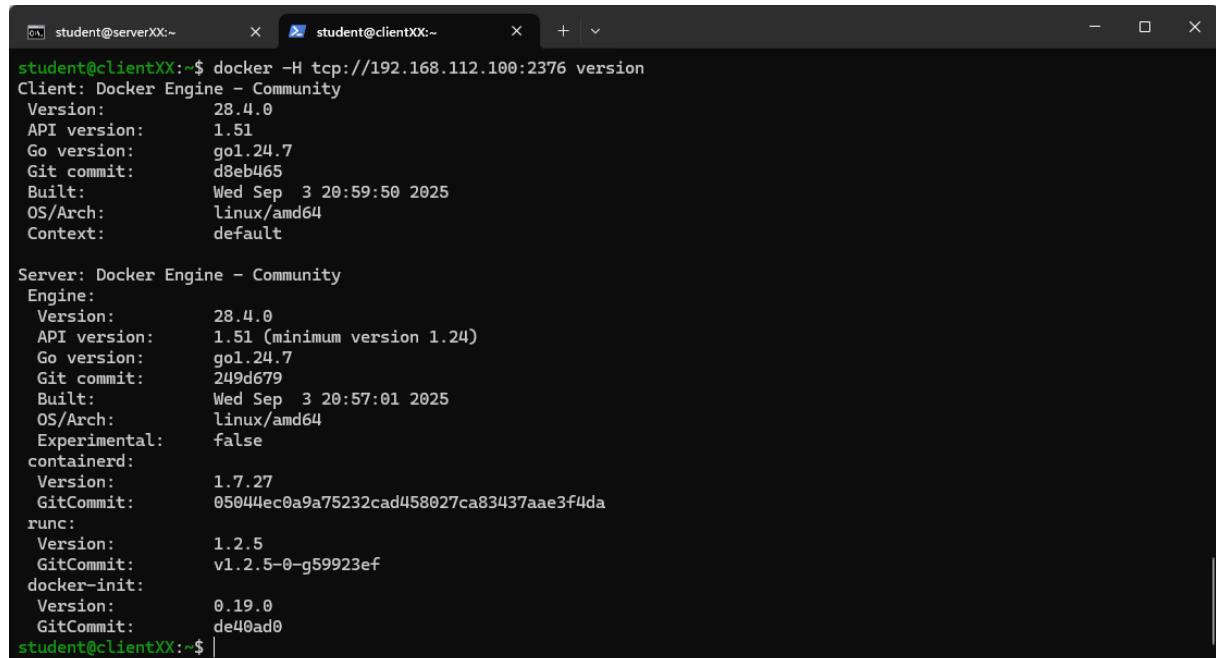
We installeren nu enkel de Docker CLI zodat we verbinding kunnen maken met de server.

```
[student@clientXX ~]$ sudo dnf install docker-ce-cli
```

De client is nu beschikbaar, maar niet de server van docker.

We maken nu verbinding met Docker op je server vanaf de client.

```
[student@clientXX ~]$ docker -H tcp://192.168.112.100:2376 version
```



```
student@clientXX:~$ docker -H tcp://192.168.112.100:2376 version
Client: Docker Engine - Community
Version: 28.4.0
API version: 1.51
Go version: go1.24.7
Git commit: d8eb465
Built: Wed Sep 3 20:59:50 2025
OS/Arch: linux/amd64
Context: default

Server: Docker Engine - Community
Engine:
Version: 28.4.0
API version: 1.51 (minimum version 1.24)
Go version: go1.24.7
Git commit: 249d679
Built: Wed Sep 3 20:57:01 2025
OS/Arch: linux/amd64
Experimental: false
containerd:
Version: 1.7.27
GitCommit: 05044ec0a9a75232cad458027ca83437aae3f4da
runc:
Version: 1.2.5
GitCommit: v1.2.5-0-g59923ef
docker-init:
Version: 0.19.0
GitCommit: de40ad0
student@clientXX:~$ |
```

Hoewel de methode functioneert is deze methode, zoals hier ingesteld, erg onveilig zoals reeds besproken.

- Geen SSH
- Iedereen kan verbinding maken met de poort vanaf een andere computer.

3.3.2 Remote Podman/Docker via SSH

Na het inloggen via SSH vanaf de client kun je Docker en Podman-commando's op de server uitvoeren.

Je kan ook onmiddellijk een commando meegeven met podman.

```
student@clientXX ~]$ ssh student@192.168.112.100 "podman version"

student@192.168.112.100's password:

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 docker krijg je een probleem wanneer je "docker version" meegeeft.

```
student@clientXX ~]$ ssh student@192.168.112.100 "docker version"

Client: Docker Engine - Community
Version: 28.4.0
API version: 1.51
Go version: go1.24.7
Git commit: d8eb465
Built: Wed Sep 3 20:59:50 2025
OS/Arch: linux/amd64
Context: default

permission denied while trying to connect to the Docker daemon socket at
unix:///var/run/docker.sock: Get
"http://%2Fvar%2Frun%2Fdocker.sock/v1.51/version": dial unix
/var/run/docker.sock: connect: permission denied
```

Zoals je ziet krijg je een melding dat er geen toegang is (permission denied). Dit kan je oplossen door sudo-rechten te gebruiken. Zoals je weet heeft docker immers sudo-rechten nodig.

We trachten het als volgt op te lossen.

```
[student@clientXX ~]$ ssh student@192.168.112.100 "sudo docker version"

student@192.168.112.100's password:
```

```
sudo: a terminal is required to read the password; either use the -S option to  
read from standard input or configure an askpass helper
```

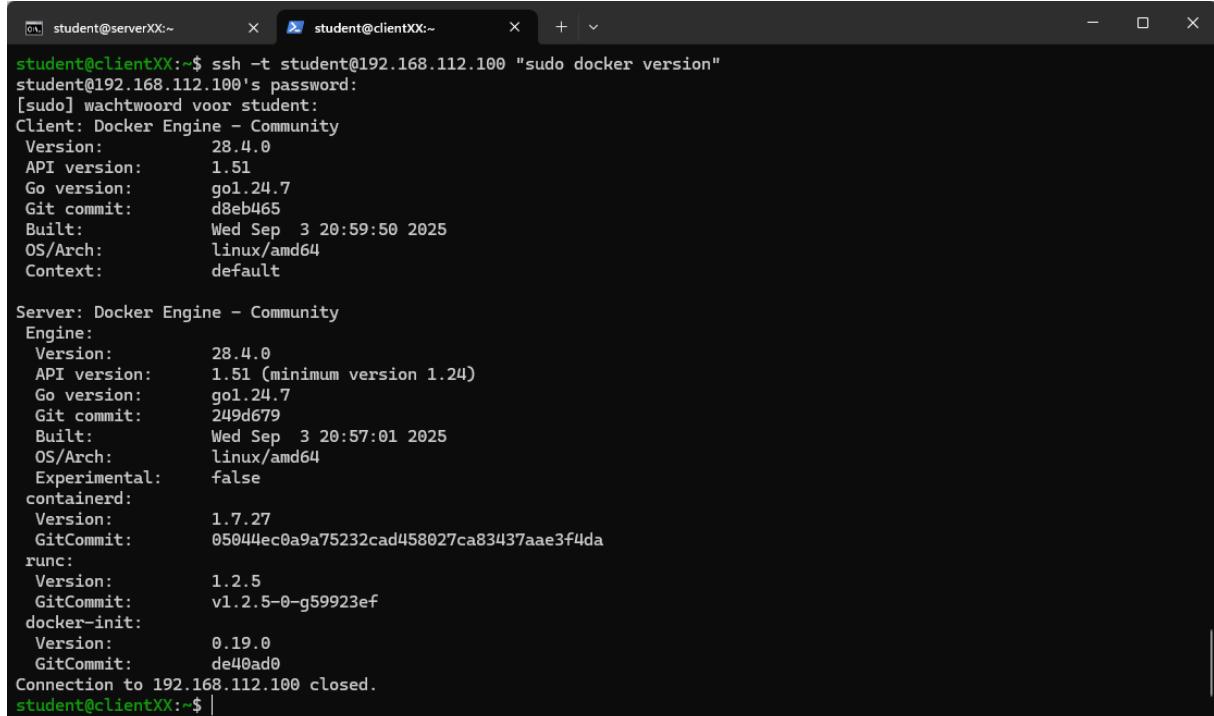
```
sudo: een wachtwoord is verplicht
```

Sudo heeft op de remote host geen interactieve terminal om het wachtwoord te vragen. Daarom krijg je:

```
sudo: een wachtwoord is verplicht
```

Je kan dat o.a. oplossen door een interactieve terminal bij SSH te forceren.

```
[student@clientXX ~]$ ssh -t student@192.168.112.100 "sudo docker version"
```



```
student@student@serverXX:~$ ssh -t student@192.168.112.100 "sudo docker version"  
student@192.168.112.100's password:  
[sudo] wachtwoord voor student:  
Client: Docker Engine - Community  
  Version:          28.4.0  
  API version:     1.51  
  Go version:      go1.24.7  
  Git commit:       d8eb465  
  Built:            Wed Sep  3 20:59:50 2025  
  OS/Arch:          linux/amd64  
  Context:          default  
  
Server: Docker Engine - Community  
Engine:  
  Version:          28.4.0  
  API version:     1.51 (minimum version 1.24)  
  Go version:      go1.24.7  
  Git commit:       249d679  
  Built:            Wed Sep  3 20:57:01 2025  
  OS/Arch:          linux/amd64  
  Experimental:    false  
containerd:  
  Version:          1.7.27  
  GitCommit:        05044ec0a9a75232cad458027ca83437aae3f4da  
runc:  
  Version:          1.2.5  
  GitCommit:        v1.2.5-0-g59923ef  
docker-init:  
  Version:          0.19.0  
  GitCommit:        de40ad0  
Connection to 192.168.112.100 closed.  
student@student@serverXX:~$ |
```

4 Containers en container images

4.1 Inleiding

In dit hoofdstuk bespreken we achtereenvolgens:

- Aanmaken docker hub account
- Een RHEL 10 container image downloaden en uitvoeren
- Een interactieve container uitvoeren
- Geïsoleerde containers uitvoeren
- Containers beheren met de CLI (podman of docker)
- Een container image interactief bouwen
- Containers stoppen, starten en verwijderen
- Een container image bouwen met Dockerfiles
- Container images uploaden naar een container registry (bijv. Red Hat Quay of Docker Hub)

4.2 Aanmaken docker hub account

Om een Docker Hub-account aan te maken, volg je deze stappen:

- Ga naar Docker Hub
 - o Surf naar <https://hub.docker.com>.
- Aanmelden
 - o Klik rechtsboven op de startpagina op de knop "Sign Up" (Registreren).
- Vul je gegevens in.

- Voer je e-mailadres, gebruikersnaam en wachtwoord in.



Create your account

Email

We suggest signing up with your work email address.

Username 

Password 

Send me occasional product updates and announcements. 

Sign up

OR

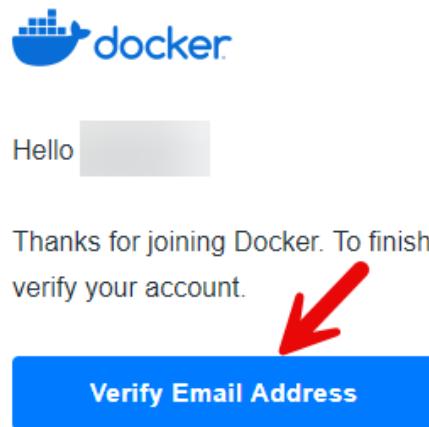
 Continue with Google

 Continue with GitHub

[Already have an account? Sign in](#)

- Je kunt je ook registreren met je GitHub- of Google-account voor een snellere aanmelding.
- Bevestig je e-mail.

- Nadat je je gegevens hebt ingevuld, stuurt Docker Hub een verificatie-e-mail. Open deze e-mail en klik op de verificatielink.



- Profiel voltooien
 - Na de verificatie kun je inloggen bij Docker Hub en je profiel verder invullen met extra informatie zoals je organisatie en voorkeuren. Je kan ook beveiligingsinstellingen wijzigen.

Zodra je klaar bent, kun je Docker Hub gebruiken om container images te hosten en te delen!

4.3 Een RHEL 10 container image downloaden en gebruiken

Kenmerken:

- RHEL UBI images zijn gratis te gebruiken, maar vereisen soms een Red Hat account voor updates.
- UBI (Universal Base Image) is de aanbevolen basis voor RHEL 10 containers.

Log je nu in op <https://hub.docker.com> met de gegevens die je juist hebt aangemaakt.

Typ in de zoekbalk bovenaan redhat in en druk uiteraard <Enter>.

The screenshot shows the Docker Hub interface with a search query 'redhat' entered in the search bar. The search results page displays 28 of 1,708 results. The results are organized into four columns of cards, each representing a Docker image. The cards include the image name, a thumbnail, the publisher (Red Hat, Inc.), a brief description, pull counts, star counts, and the last updated date. The interface includes a sidebar for filtering by products, trusted content, categories, operating systems, and architectures. A red arrow points to the search bar.

Image Name	Publisher	Description	Pulls	Stars	Last Updated
redhat/granite-3-2b-inst...	Red Hat, Inc.	IBM's Granite 3.0 large language model (LLM), optimized for local large language...	7.3K	1	9 months
redhat/granite-7b-lab-gg...	Red Hat, Inc.	4-bit quantized version of model Granite-7b-lab	4.8K	5	10 months
redhat/granite-3-8b-inst...	Red Hat, Inc.	IBM's Granite 3.0 large language model (LLM), optimized for local large language...	1.9K	3	9 months
redhat/granite-3-1b-a40...	Red Hat, Inc.	IBM's Granite 3.0 large language model (LLM), optimized for local large language...	1.1K	8	9 months
redhat/granite-3-3b-a80...	Red Hat, Inc.	IBM's Granite 3.0 large language model (LLM), optimized for local large language...	711	1	9 months
redhat/granite-3-1-8b-in...	Red Hat, Inc.	IBM's Granite 3.1 large language model (LLM), optimized for local large language...	693	8	7 months
redhat/ubi8	Red Hat, Inc.	Red Hat Universal Base Image 8	10M+	162	7 days
redhat/ubi9	Red Hat, Inc.	Red Hat Universal Base Image 9	5M+	67	6 days
redhat/ubi8-minimal	Red Hat, Inc.	Red Hat Universal Base Image 8 Minimal	10M+	52	8 days
redhat/ubi9-minimal	Red Hat, Inc.	Red Hat Universal Base Image 9 Minimal	1M+	22	6 days
redhat/ubi8-init	Red Hat, Inc.	Red Hat Universal Base Image 8 Init	1M+	18	6 days
redhat/ubi8-micro	Red Hat, Inc.	Red Hat Universal Base Image 8 Micro	500K+	28	8 days

Je krijgt nu bijna allemaal images van Redhat als resultaat.

Klik op Red Hat, inc. bij één van je resultaten zoals je hieronder ziet.

IMAGE

redhat/granite-3-2b-inst... ✓
Red Hat, Inc.

IBM's Granite 3.0 large language model (LLM), optimized for local large language...

Pulls 7.3K
Stars 1
Last Updated 9 months

Je ziet nu enkel resultaten van Red Hat inc..

Je kan hier ook rechtstreeks naartoe gaan: [Red Hat | Docker Hub](#).

Als je ubi10 in het zoekvenster intypt zie je onderstaand resultaat.

hub Explore My Hub

Red Hat ✓ Verified Publisher

Repositories

ubi10 Displaying 1 to 4 of 4 repositories

IMAGE	repository	publisher	description	Pulls	Stars	Last Updated
	redhat/ubi10	Red Hat, Inc.	Red Hat Universal Base Image 10	10K+	3	about 20 hours
	redhat/ubi10-init	Red Hat, Inc.	Red Hat Universal Base Image 10 Init	6.7K	0	2 days
	redhat/ubi10-minimal	Red Hat, Inc.	Red Hat Universal Base Image 10 Minimal	10K+	0	6 days
	redhat/ubi10-micro	Red Hat, Inc.	Red Hat Universal Base Image 10 Micro	9.3K	0	6 days

Als je ubi9 intypt zie je onderstaand resultaat.

hub Explore My Hub

Red Hat ✓ Verified Publisher

Repositories

ubi9 Displaying 1 to 4 of 4 repositories

IMAGE	repository	publisher	description	Pulls	Stars	Last Updated
	redhat/ubi9-init	Red Hat, Inc.	Red Hat Universal Base Image 9 Init	500K+	10	2 days
	redhat/ubi9	Red Hat, Inc.	Red Hat Universal Base Image 9	5M+	67	6 days
	redhat/ubi9-minimal	Red Hat, Inc.	Red Hat Universal Base Image 9 Minimal	1M+	22	6 days
	redhat/ubi9-micro	Red Hat, Inc.	Red Hat Universal Base Image 9 Micro	Pulls	Stars	Last Updated

Zoals je kan gokken maakt ubi10 gebruik van RHEL 10 en ubi9 maakt gebruik van RHEL 9.

De 4 varianten van Red Hat Universal Base Image hebben elk een ander doel en grootte:

- redhat/ubi10: De standaard Universal Base Image 10.
- redhat/ubi10-init: UBI 10 met een init-systeem (bijv. systemd) toegevoegd.
- redhat/ubi10-minimal: Een afgeslankte versie van UBI 10.
- redhat/ubi10-micro: De extreem minimalistische variant.

We bekijken redhat/ubi10 verder. Klik hiervoor op redhat/ubi10.

IMAGE

 **redhat/ubi10** 
Red Hat, Inc.

Red Hat Universal Base Image 10

Pulls	10K+
Stars	3
Last Updated	about 20 hours

Klik op Tags.



redhat/ubi10  Verified Publisher
By [Red Hat, Inc.](#) • Updated about 21 hours ago
Red Hat Universal Base Image 10

[IMAGE](#) [SECURITY](#) [OPERATING SYSTEMS](#)

 3  10K+



[Overview](#) [Tags](#)

Een tag is een label dat verwijst naar een specifieke versie of variant van een image.

Een image wordt meestal aangeduid als:

<image-naam>:<tag>

Meestal gebruik je de laatste versie. Het label is latest.

Je ziet hier de exacte syntax staan.

TAG	OS/ARCH	Compressed size
latest	linux/amd64	75.24 MB
719b672e7031	linux/arm64/v8	72.88 MB
7f23918a9fa4	linux/ppc64le	79.75 MB
6f4708918ad4		
+1 more...		

We trachten nu het image te pullen naar de server maar krijgen een foutmelding.

```
student@serverXX:~$ docker pull redhat/ubi10:latest
permission denied while trying to connect to the Docker daemon socket at
unix:///var/run/docker.sock: Post
"http://%2Fvar%2Frun%2Fdocker.sock/v1.51/images/create?fromImage=docker.io%2Fre
dhat%2Fubi10&tag=latest": dial unix /var/run/docker.sock: connect: permission
denied
```

Je krijgt een foutmelding omdat je docker niet met sudo-rechten uitvoert.

We voeren het nu correct uit.

```
student@serverXX:~$ sudo docker pull redhat/ubi10:latest
[sudo] wachtwoord voor student:
latest: Pulling from redhat/ubi10
b657dc667c04: Pull complete
Digest: sha256:b51579c045599bc2f2ba407f28fcc26e13b87409e062e227b185ae69d699bf95
Status: Downloaded newer image for redhat/ubi10:latest
docker.io/redhat/ubi10:latest
```

In Docker betekent een image pullen dat je een containerimage downloadt van een externe registry (zoals Docker Hub (wij), Red Hat Registry, ...) naar je lokale Docker-cache, zodat je het later kunt gebruiken om containers te starten.

Bij Docker wordt de data in ons systeem bewaard in (verschillend submappen) van /var/lib/docker.

We trachten nu hetzelfde image te pullen via podman.

```
student@serverXX:~$ podman pull redhat/ubi10:latest
? Please select an image:
```

```
▶ registry.access.redhat.com/redhat/ubi10:latest  
registry.redhat.io/redhat/ubi10:latest  
docker.io/redhat/ubi10:latest
```

Je kan nu via de pijltjes van je toetsenbord selecteren waar je het image van wil downloaden. We kiezen nu voor registry.redhat.io/redhat/ubi10:latest en drukken <enter>.

Je krijgt nu een foutmelding...

```
✓ registry.redhat.io/redhat/ubi10:latest  
Trying to pull registry.redhat.io/redhat/ubi10:latest...  
  
Error: initializing source docker://registry.redhat.io/redhat/ubi10:latest:  
unable to retrieve auth token: invalid username/password: unauthorized: Please  
login to the Red Hat Registry using your Customer Portal credentials. Further  
instructions can be found here:  
https://access.redhat.com/RegistryAuthentication
```

We loggen in op de Red Hat registry als volgt (met je RedHat-account):

```
student@serverXX:~$ podman login registry.redhat.io  
  
Username: stijnjacobs  
  
Password:  
  
Login Succeeded!
```

We trachten nu opnieuw via het registry registry.redhat.io van Red Hat te downloaden.

```
student@serverXX:~$ podman pull redhat/ubi10:latest  
? Please select an image:  
  
registry.access.redhat.com/redhat/ubi10:latest  
▶ registry.redhat.io/redhat/ubi10:latest  
docker.io/redhat/ubi10:latest  
  
✓ registry.redhat.io/redhat/ubi10:latest  
Trying to pull registry.redhat.io/redhat/ubi10:latest...  
  
Error: initializing source  
docker://registry.redhat.io/redhat/ubi10:latest: reading  
manifest latest in registry.redhat.io/redhat/ubi10:  
unauthorized: access to the requested resource is not authorized
```

Dit werkt dus nog altijd niet. Er staat wel niet meer dat je moet inloggen. Er staat dat je geen toegang hebt tot de resource. Je hebt wel toegang tot ubi10/ubi. Op registry.redhat.io/redhat/ bestaat ubi10 dus niet maar wel ubi10/ubi wel.

Daarnaast vind je hieronder info over de 3 locaties waar je kan downloaden.

- registry.access.redhat.com
 - o Dit is de officiële Red Hat Container Catalog registry.
 - o Bevat door Red Hat onderhouden en ondersteunde images.
- registry.redhat.io
 - o Dit is ook een Red Hat registry, maar dan de “authenticatie vereiste” variant. Dit hebben we juist gemerkt.
- docker.io
 - o Dit is de Docker Hub registry (waar we op de website gekeken hebben).
 - o Images hier zijn meestal door Red Hat zelf gesynchroniseerd, maar soms wat minder up-to-date dan de officiële Red Hat registries.
 - o Niet altijd de meest betrouwbare of ondersteunde bron.

We trachten dus opnieuw te pullen, maar nu met met ubi10/ubi.

```
student@serverXX:~$ podman pull ubi10/ubi:latest
? Please select an image:
registry.access.redhat.com/ubi10/ubi:latest
▶ registry.redhat.io/ubi10/ubi:latest
docker.io/ubi10/ubi:latest
```

Het downloaden lukt nu wel 😊

```
✓ registry.redhat.io/ubi10/ubi:latest
Trying to pull registry.redhat.io/ubi10/ubi:latest...
Getting image source signatures
Checking if image destination supports signatures
Copying blob b657dc667c04 done    |
Copying config d84e7638c2 done    |
Writing manifest to image destination
Storing signatures
d84e7638c2c4ad5390ee79228164903c95992a8af5311dacaacbaaa8069ec780
```

Je kan ook de registry specifiëren bij het downloaden zoals hieronder. Zo moet je niet meer selecteren vanwaar je wil downloaden.

```
student@serverXX:~$ podman pull registry.access.redhat.com/ubi10/ubi:latest
Trying to pull registry.access.redhat.com/ubi10/ubi:latest...
Getting image source signatures
```

```
Checking if image destination supports signatures
```

```
Copying blob b657dc667c04 skipped: already exists
```

```
Copying config d84e7638c2 done |
```

```
Writing manifest to image destination
```

```
Storing signatures
```

```
d84e7638c2c4ad5390ee79228164903c95992a8af5311dacaacbaaa8069ec780
```

PS Gebruik `registry.access.redhat.com` als je geen login hebt.

De catalogus van `registry.access.redhat.com` vind je terug op

<https://catalog.redhat.com/en/search?searchType=containers>.

Je kan jammer genoeg niet zoeken naar `ubi10` maar je moet zoeken naar `ubi` of `Red Hat Universal Base Image 10`.

The screenshot shows the Red Hat Ecosystem Catalog interface. At the top, there's a navigation bar with links for Solutions, Products, Artifacts, and Partners. Below that is a search bar with the placeholder text "Search for". A red arrow points to this search bar. The main content area shows a search result for "Red Hat Universal Base Image 10". The result is a card with the title "Red Hat Universal Base Image 10" and the tag "ubi10". It includes dropdown menus for "Architecture" (amd64), "Tag" (10.0-1755007654), and "Repo". Below the card, there are several other options under the heading "Red Hat Universal Base Image 10": "Red Hat Universal Base Image 10", "Red Hat Universal Base Image 10 Init", "Red Hat Universal Base Image 10 Micro", "Red Hat Universal Base Image 10 Minimal", and "Red Hat Universal Base Image 10 Minimal". At the bottom of the card, there are links for "Overview", "Security", and "Technical Information".

Klik nu weer op `Red Hat Universal Base Image 10`.

This screenshot shows the same search results page as the previous one, but now the result for "Red Hat Universal Base Image 10" is selected. A red arrow points to this specific result card. The card displays the title "Red Hat Universal Base Image 10" and the tag "ubi10". It includes a "Red Hat" logo icon. Below the title, it says "Provides the latest release of Red Hat Universal Base Image 10." There are also small icons for "Container image", "10.0", and "ppc64le". On the left, there's a sidebar with a "Type" dropdown set to "Containers" and a list of "Software" and "Containers". At the top right, there are filters for "Sort by: Relevance" and "1-20 of 27".

Je ziet nu allerlei info. Voor ons is alleen het laatste tabblad nu belangrijk. Klik dus op Get this image.

The screenshot shows the Red Hat Ecosystem Catalog interface. At the top, there's a navigation bar with links for Solutions, Products, Artifacts, and Partners. A search bar is also present. Below the navigation, the page title is "Red Hat Universal Base Image 10". On the right side, there's a "Provided by Red Hat" logo. The main content area shows details about the image, including its architecture (amd64), tag (10.0-1755007654), and repository structure (Single-stream). A red arrow points to the "Get this image" button.

Klik nu op Unauthenticated.

Je ziet nu onderaan hoe je het image kan downloaden.

This screenshot shows the same Red Hat Ecosystem Catalog page as the previous one, but with a different focus. A red arrow points to the "Unauthenticated" button in the "Get this image" section. Below this, there are sections for "Using podman" and "Using docker", each with a command-line example. The command for podman is "podman pull registry.access.redhat.com/ubi10:10.0-1755007654" and the command for docker is "docker pull registry.access.redhat.com/ubi10:10.0-1755007654". Both commands are highlighted with a red box.

Na de : staat de versie.

- Je kan vaak gebruik maken van lastest maar hier staat een specifieke versie: 10.0-1755007654.
- ubi10/ubi:latest is aan te raden i.p.v. ubi10:latest. Red Hat plaatst hun UBI-images onder subdirectories zoals ubi10/ubi en daardoor krijg je zeker geen foutmeldingen.

We proberen “podman pull registry.access.redhat.com/ubi10/ubi:latest” uit te voeren en krijgen geen foutmelding...

```
student@serverXX:~$ podman pull registry.access.redhat.com/ubi10:latest
Trying to pull registry.access.redhat.com/ubi10:latest...
Getting image source signatures
Checking if image destination supports signatures
Copying blob b657dc667c04 skipped: already exists
Copying config d84e7638c2 done  |
Writing manifest to image destination
Storing signatures
d84e7638c2c4ad5390ee79228164903c95992a8af5311dacaacbaaa8069ec780
```

Zoals verwacht is het image al gedownload.

- ubi10:latest → probeert een image in de root van ubi10 te vinden → soms niet beschikbaar.
- ubi10/ubi:latest → correcte repository en image naam → werkt en haalt de UBI 10 base image op.

Docker-images zijn vaak opgebouwd uit meerdere lagen, waarbij elke laag een set van veranderingen of toevoegingen aan de vorige laag bevat.

Als bepaalde lagen al op je systeem aanwezig zijn (bijvoorbeeld van een eerdere image die je hebt gedownload), worden deze lagen hergebruikt en worden alleen de nieuwe of gewijzigde lagen (de delta) gedownload.

4.4 Podman commando's

4.4.1 Containers en container images downloaden, uitvoeren en logs bekijken

PS We voeren alle commando's uit op de server.

Voer uit.

```
student@serverXX:~$ podman run registry.access.redhat.com/ubi10/ubi:latest
```

- podman run
Het Podman-commando dat een nieuwe container opstart op basis van een opgegeven image. Als het image nog niet lokaal aanwezig is, zal Podman deze eerst downloaden.
- registry.access.redhat.com/ubi10/ubi:latest
Dit is het image die Docker moet gebruiken om de container te starten. In dit geval is het een registry.access.redhat.com/ubi10/ubi:latest

- registry.access.redhat.com
Dit is de RHEL Registry, een plaats waar RHEL Podman-images host zoals reeds gezien.
- ubi10/ubi
Dit is de Universal Base Image (UBI). Universal Base Image is een officiële, onderhoudbare en herdistribueerbare basisimage die je vrij mag gebruiken, zelfs in gesloten of commerciële projecten, zonder dat je per se een Red Hat-subscriptie nodig hebt.
- latest: Zoals reeds uitgelegd verwijst dit naar de laatste versie.

Podman zal een nieuwe container op basis van deze image aanmaken en starten. Dit betekent dat je een geïsoleerde RHEL-omgeving krijgt waarin je RHEL-gebaseerde applicaties kunt draaien. Gezien deze container niets te doen heeft zal deze onmiddellijk stoppen.

We zullen nu ervoor zorgen dat de container iets te doen heeft.

Voer uit.

```
student@serverXX:~$ podman run registry.access.redhat.com/ubi10/ubi:latest
whoami
```

```
root
```

- De container wordt gestart: Podman start een nieuwe container op basis van de ubi10/ubi:latest-image.
- Podman voert whoami direct uit als een proces binnen de container, zonder dat er een shell wordt gestart.
- De container stopt: Zodra de opdracht is voltooid (whoami), zal de container stoppen, tenzij er specifieke opties zijn toegevoegd om de container draaiende te houden.

Het commando “podman ps” toont een lijst van actieve containers die momenteel draaien op je systeem. Standaard worden alleen de containers weergegeven die op dat moment actief zijn.

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

CONTAINER	ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
student@serverXX:~\$							

Als je een lijst wil tonen van alle containers, zowel actieve als gestopte containers, gebruik je onderstaande.

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

CONTAINER	ID	IMAGE	COMMAND	...
...				...
3f9fa2c21088	registry.access.redhat.com/ubi10/ubi:latest	whoami		...
...				...

Om een container te starten die al eens actief is geweest maak je gebruik van de unieke container ID.

We zien bijvoorbeeld hier dat de unieke container ID voor de container die het commando “whoami” heeft uitgevoerd 3f9fa2c21088 is.

Je kan deze container als volgt opnieuw starten.

```
student@serverXX:~$ podman start 3f9fa2c21088  
3f9fa2c21088
```

Je hoeft niet de volledige container ID in te geven wanneer je een Podman-commando uitvoert. Je kunt een gedeeltelijke container ID, zolang de opgegeven ID uniek genoeg is om naar één specifieke container te verwijzen. Je kan ook de containernaam gebruiken om de container te starten.

Podman laat je dus meestal toe om alleen de eerste paar tekens van die ID te gebruiken, zolang er geen andere container is met een ID die begint met dezelfde tekens.

```
student@serverXX:~$ podman start 3f  
3f
```

Als je nu de lijst opvraagt van de actieve containers krijg je onderstaand resultaat.

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

CONTAINER	ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
-----------	----	-------	---------	---------	--------	-------	-------

Dat komt omdat de containers al zijn uitgevoerd.

Je kan logbestanden van een container bekijken.

```
student@serverXX:~$ podman logs 3f  
root  
root  
root
```

- podman logs: Dit podman-commando toont de standaard output (stdout) en foutmeldingen (stderr) die door de container worden gegenereerd sinds het opstarten van de container. Dit is nuttig om te zien wat er binnen de container gebeurt, zoals foutmeldingen, uitvoer van scripts, of andere meldingen.
- 3f: Dit is weer de gedeeltelijke container-ID van de container waarvan je de logs wilt zien.
 - o Als 3f de eerste tekens van een unieke container-ID zijn, werkt het commando.
 - o Als meerdere containers dezelfde eerste tekens hebben, moet je meer van de ID invoeren.
 - o Je kunt ook een eigen containernaam gebruiken als je de container een naam hebt gegeven toen je hem startte. Daar komen we later op terug.

We zullen nu een keer kijken welke bestanden/mappen er in de container gelist kunnen worden met het commando ls zonder path aan te geven. Net zoals zojuist voegen we het commando toe.

```
student@serverXX:~$ podman run registry.access.redhat.com/ubi10/ubi:latest ls
```

```
afs
```

```
...
```

```
var
```

Als we nu terug een lijst tonen van alle containers, ongeacht of ze momenteel draaien of gestopt zijn, zie je dat deze container er is bijgekomen.

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

CONTAINER ID	IMAGE	COMMAND	...
3f9fa2c21088	registry.access.redhat.com/ubi10/ubi:latest	whoami	...
0444ba18e5e2	registry.access.redhat.com/ubi10/ubi:latest	ls	...

We vragen nu de logs op van de juist uitgevoerde container.

```
student@serverXX:~$ podman logs 04
```

```
afs
```

```
...
```

```
var
```

4.4.2 Een interactieve container uitvoeren

U wilt mogelijk niet dat een container simpelweg de toegewezen taak uitvoert en daarna meteen stopt.

In sommige gevallen heeft u een container nodig die voor een bepaalde periode actief blijft. Dit stelt u ook in staat om verbinding te maken met de container om verschillende beheertaken uit te voeren, zoals u normaal zou doen op een RHEL server.

Een interactieve container:

- Dwingt de container om door te blijven draaien.
- Geeft toegang tot de console van de container.

Met -it toe te voegen voer je een container uit die blijft draaien.

```
student@serverXX:~$ podman run -it registry.access.redhat.com/ubi10/ubi:latest
```

-it is een samentrekking van -i en -t.

-i betekent interactive.

Deze houdt de standaardinvoer open, ook als je er niets naar stuurt.

-t geeft een terminal.

Meestal worden -i en -t gecombineerd.

Na uitvoeren van bovenstaande opdracht ben je nu in de container.

Voer nu whoami uit.

```
[root@594bc4040809 /]# whoami  
root
```

Voer nu 'ls' uit.

```
[root@594bc4040809 /]# ls  
afs bin boot dev ... tmp usr var
```

Voer nu 'cat /etc/hostname' uit.

```
[root@594bc4040809 /]# cat /etc/hostname  
594bc4040809
```

Uiteraard zie je de naam van de container (hostname) ook na @ staan in je huidige prompt.

Voer nu "cat /etc/os-release" uit.

```
[root@594bc4040809 /]# cat /etc/os-release  
NAME="Red Hat Enterprise Linux"  
VERSION="10.0 (Coughlan)"  
...  
...
```

We gaan nu in een venster met ssh verbinding maken met ServerXX en we kijken of er nu containers actief zijn.

```
student@serverXX:~$ podman ps  
CONTAINER ID IMAGE COMMAND ...  
594bc4040809 registry.access.redhat.com/ubi10/ubi:latest /bin/bash ...
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
594bc4040809	registry.access.redhat.com/ubi10/ubi:latest	/bin/bash	6 minutes ago	Up 6 minutes		objectiv

Je ziet dat de hostname van de container en de container ID standaard dezelfde is.



```
[student@serverXX ~]$ podman ps
CONTAINER ID  IMAGE          COMMAND   CREATED    STATUS      PORTS     NAMES
594bc4040809  registry.access.redhat.com/ubi10/ubi:latest /bin/bash  6 minutes ago Up 6 minutes          objectiv
[student@serverXX ~]$
```

```
[root@594bc4040809 /]# exit
exit
[student@serverXX ~]$
```

Sluit de container nu af door in de container exit in te typen.

```
[root@594bc4040809 /]# exit
exit
```

Toon daarna terug de actieve containers.

```
student@serverXX:~$ podman ps
CONTAINER ID  IMAGE          COMMAND   CREATED    STATUS      PORTS     NAMES
```

De actieve container is nu verdwenen.



```
[root@594bc4040809 /]# exit
exit
[student@serverXX ~]$
```



```
[student@serverXX ~]$ podman ps
CONTAINER ID  IMAGE          COMMAND   CREATED    STATUS      PORTS     NAMES
[student@serverXX ~]$
```

Er zijn twee belangrijke manieren waarop je een container kunt draaien: interactief en detached.

Hier is het verschil tussen beide:

- Interactieve container
Bij het draaien van een container in interactieve modus, krijg je direct toegang tot de terminal binnen de container. Dit stelt je in staat om direct met de container te communiceren.
- Detached container
Bij het draaien van een container in detached modus, wordt de container op de achtergrond uitgevoerd zonder dat je er direct interactie mee hebt via de terminal.

4.4.3 Een detached container uitvoeren

Start nu een detached container.

```
student@serverXX:~$ podman run -d registry.access.redhat.com/ubi10/ubi:latest
tail -f /dev/null
```

```
035e6040adf0620bae7322eeb27760768cf80c0d1877bb52e81629bfec7f7698
```

De container zal nu draaien op de achtergrond.

```
--detach (of -d)
```

Deze vlag zorgt ervoor dat de container in de achtergrond begint te draaien.

```
tail -f /dev/null
```

Dit is het commando dat in de container wordt uitgevoerd. Een container die niets te doen sluit zichzelf af!

Dit is het daadwerkelijke commando dat wordt uitgevoerd binnen de container nadat deze is gestart. Laten we het opsplitsen:

tail: dit commando toont normaal de laatste regels van een bestand.

-f: betekent "follow", dus blijf luisteren naar nieuwe regels die aan het bestand worden toegevoegd.

/dev/null: dit is een speciaal bestand dat altijd leeg is, en waar nooit iets in geschreven wordt (ook wel de bit bucket genoemd).

tail -f /dev/null start een proces dat eindeloos blijft draaien en nooit output geeft.

Je zal dus ook zien dat de container nog draait.

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

CONTAINER ID	IMAGE	COMMAND	...
035e6040adf0	registry.access.redhat.com/ubi10/ubi:latest	tail -f /dev/null	...

Je ziet het commando ook bij COMMAND staan.

We gaan nu het docker log van deze container bekijken.

```
student@serverXX:~$ podman logs 03
```

Je ziet dat er niets in het log staat aangezien tail -f /dev/null geen output heeft.

Ga nu naar de container die nog uitgevoerd wordt en voer onderstaande uit.

```
student@serverXX:~$ podman exec -it 03 /bin/bash
```

```
[root@035e6040adf0 /]#
```

We vragen nu op welke shell we draaien.

```
[root@035e6040adf0 /]# echo $SHELL
```

```
/bin/bash
```

Verlaat de container door exit in te typen.

```
[root@035e6040adf0 /]# exit  
exit  
student@serverXX:~$
```

De container voert nog steeds op de achtergrond “tail -f /dev/null” uit.

```
student@serverXX:~$ podman ps  
CONTAINER ID IMAGE COMMAND ...  
035e6040adf0 registry.access.redhat.com/ubi10/ubi:latest tail -f /dev/null ...
```

4.4.4 Info over container opvragen

We willen gedetailleerde informatie bekomen over een docker image. Dit doe je met docker image inspect.

```
student@serverXX:~$ podman image inspect registry.access.redhat.com/ubi10/ubi:latest  
[  
 {  
   "Id": "620bc887852d1339346bfccbf61fa201ebd6fae470e2119462f1eb6dfe9a7288",  
   "Digest":  
   ...  
 }
```

We bespreken hier beknopt een aantal punten uit.

- Veld Layers: Dit toont de verschillende lagen (met hun hashes) waaruit het image is opgebouwd.

```
"RootFS": {  
  "Type": "layers",  
  "Layers": [  
    "sha256:09bf1696a91e6b849cc8e091e7995881e207b58dc31ff0b0d243913667b2570d"  
  ]  
},
```

Elke laag kan veranderingen bevatten, zoals toegevoegde bestanden of gewijzigde configuraties. Bij Docker en Podman zie je vaak meerdere lagen. Docker en Podman gebruiken een gelaagd bestandssysteem om efficiënt om te gaan met data, waarbij lagen worden gedeeld tussen images als er geen verschillen zijn. Hier is er maar één laag. Dit komt waarschijnlijk omdat het een “scratch” image is.

We zullen een andere image pullen om meer lagen te zien.

```
student@serverXX:~$ podman pull registry.access.redhat.com/ubi10/httpd-24  
student@serverXX:~$ podman image inspect  
registry.access.redhat.com/ubi10/httpd-24
```

```
"RootFS": {
    "Type": "layers",
    "Layers": [
        "sha256:c7469a1dad060aaa503c09b0cb15187f7794e4a7285a7205504316c18f863dec",
        "sha256:c046546934321a2adc537d0beca0baaa20573e6f5e442c6ade8b8258b23bd5f4",
        "sha256:ca10e1da07d6d77a42bf8d64668fd120e5e877fd18e9ec5c82f2464b93f75d52"
    ]
},
```

- Velden Architecture, Os en OsVersion

```
"Architecture": "amd64",
"Os": "linux",
```

Het veld Architecture geeft aan voor welke hardware architectuur de Docker-image bedoeld is.

Het veld Os geeft aan voor welk besturingssysteem de Docker-image is bedoeld. Dit kan Windows of Linux zijn.

Het veld OsVersion (indien aanwezig) geeft de specifieke versie van het besturingssysteem aan waarop het image is gebaseerd. Dit veld is vooral van belang voor Windows-containers, omdat er verschillende versies van Windows Server en Windows-besturingssystemen zijn die invloed hebben op de compatibiliteit van het image. Dit vind je hier dan ook niet terug.

- Veld Entrypoint

Het UBI10 base image heeft geen Entrypoint gedefinieerd omdat het een generiek basisimage is. UBI10/httpd-24 heeft dat wel.

```
"Entrypoint": [
    "container-entrypoint"
],
```

Definieert het vaste hoofdcommando dat altijd wordt uitgevoerd wanneer een container start. Dit betekent dat, wanneer je de container start (bijvoorbeeld met podman run), Podman automatisch het commando uitvoert dat in de Entrypoint is gedefinieerd.

- Veld Cmd

Het UBI10 base image heeft geen Cmd gedefinieerd omdat het een generiek basisimage is. UBI10/httpd-24 heeft dat wel.

```
"Cmd": [
    "/usr/bin/run-httpd"
],
```

Dit is de default command die het image zal draaien, tenzij je zelf iets anders meegeeft bij podman run <image>

Entrypoint wordt ALTIJD uitgevoerd. Cmd kan worden overschreven.

In een UBI10-container met Apache wordt Cmd ingesteld op "/usr/run-httpd". Dit zorgt ervoor dat de Apache-webserver in de voorgrond blijft draaien (tenzij je iets anders meegeeft uiteraard).

Als het hoofdproces (bijvoorbeeld run-httpd) stopt, dan ziet Podman dat PID 1 is beëindigd en wordt de hele container als gestopt beschouwd.

In een typische Podman Linux-omgeving blijft de container actief zolang het hoofdproces, zoals een webserver of database, actief is.

We inspecteren nu een ander bekend image “`docker.io/library/nginx:latest`”.

We downloaden eerst het image.

```
student@serverXX:~$ podman pull docker.io/library/nginx:latest
```

Docker-entrypoint.sh doet een aantal voorbereidingen voordat nginx start.

```
student@serverXX:~$ podman image inspect docker.io/library/nginx:latest | grep -A2 Entrypoint
```

```
    "Entrypoint": [
        "/docker-entrypoint.sh"
    ],
```

Bij Cmd staat dat nginx op de voorgrond moet worden uitgevoerd.

```
student@serverXX:~$ podman image inspect docker.io/library/nginx:latest | grep -A4 Cmd
```

```
    "Cmd": [
        "nginx",
        "-g",
        "daemon off;"
    ],
```

Hieronder zie je de verschillende lagen van het image.

```
student@serverXX:~$ podman image inspect docker.io/library/nginx:latest | grep -A8 Layers
```

```
    "Layers": [
        "sha256:eb5f13bce9936c760b9fa73aeb1b608787daa36106cc888104132e353ed37252",
        "sha256:dab69e9f41e95b695c830dd40de509cac70228535889014e632834462f5683fa",
        "sha256:39bc11fab5202a44b1cd5ce2dff1db01a5e4fb6dbf3382be700b2b5b337fe8",
        "sha256:2988603ca26438649eac2757fb9dd9ba08319f432d4971420cfa2134a4dcfa7",
        "sha256:a0e5983a25a50e0523bf004411cbbbfc08b6735096460ec2e3e0e919633632c1",
        "sha256:129b375526fcf7be9015a58cd3abf6f623b6f82244e575de11905b9bad61e8f7",
        "sha256:45c2d10807fb1f09c464da6f87fc08f863031b0cff7689378e2fa8f9ac6956bd"
    ]
```

4.5 Interactief een container bouwen

4.5.1 Webserver

We gaan nu zelf een Apache-webserver interactief bouwen in een container op basis van UBI10.

We starten hiervoor een interactieve container op.

```
student@serverXX:~$ podman run -it -p 8080:80  
registry.access.redhat.com/ubi10/ubi:latest /bin/bash
```

```
[root@cbc5aa4d0583 /]#
```

-it: deze optie is al reeds besproken.

-p <HOST_POORT>:<CONTAINER_POORT>: de webserver draait op poort 80 in de container en zal beschikbaar zijn via poort 8080 op de host.

Opmerking: als je wil dat de website op poort 80 beschikbaar is op de host (-p 80:80) moet je sudo gebruiken omdat een niet-root gebruiker geen poorten onder 1024 mag openen.

Je bent nu root in de container.

We installeren nu apache in de container.

```
[root@549bead12a05 /]# dnf install -y httpd  
Updating Subscription Management repositories.  
...  
Complete!
```

Maak nu een eenvoudige webpagina aan.

```
[root@549bead12a05 /]# echo "Hallo vanaf Apache op UBI10!" >  
/var/www/html/index.html
```

Start nu httpd op in de container.

```
[root@549bead12a05 /]# /usr/sbin/httpd  
AH00558: httpd: Could not reliably determine the server's fully qualified  
domain name, using 192.168.112.100. Set the 'ServerName' directive globally to  
suppress this message
```

De melding die je krijgt heeft te maken met DNS maar de webserver is opgestart.

Vraag het IP-adres op via ip (nmcli werkt niet).

```
[root@cbc5aa4d0583 /]# ip addr  
1: lo: <LOOPBACK,UP,LOWER_UP> ...
```

```

...
2: ens160: <BROADCAST,MULTICAST,UP,LOWER_UP> ...

    link/ether 6a:18:4a:39:ad:28 brd ff:ff:ff:ff:ff:ff

    inet 192.168.112.100/24 brd 192.168.112.255 scope global noprefixroute ens160

...

```

Het IP-adres vinden we terug bij ens160: 192.168.112.100. Het lijkt alsof de container hetzelfde IP-adres als dat van de container host heeft. Dat komt omdat je de container in rootless mode hebt gestart.

```

@549bead12a05/ x + v
[root@549bead12a05 /]# ip addr
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
                valid_lft forever preferred_lft forever
2: ens160: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 65520 qdisc fq_codel state UNKNOWN group default qlen 1000
    link/ether 3a:f4:17:46:43:cd 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::3af4:17ff:fe46:43cd/64 scope link
                valid_lft forever preferred_lft forever
[root@549bead12a05 /]#

```

Start een tweede SSH-verbinding en check welke containers actief zijn.

CONTAINER ID	IMAGE	COMMAND	...
...			
549bead12a05	registry.access.redhat.com/ubi10/ubi:latest	/bin/bash	...

Zoals je ziet is de container 549bead12a05 nog steeds actief.

Je kan ook vanaf de container host het IP-adres opvragen van de container.

```

student@serverXX:~$ podman exec -it 54 ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> ...
...
2: ens160: <BROADCAST,MULTICAST,UP,LOWER_UP> ...
    link/ether 6a:18:4a:39:ad:28 brd ff:ff:ff:ff:ff:ff
    inet 192.168.112.100/24 brd 192.168.112.255 scope global noprefixroute ens160
...

```

We testen nu vanuit de container of de webserver draait.

```

[root@549bead12a05 /]# curl 192.168.112.100
Hallo vanaf Apache op UBI10!

```

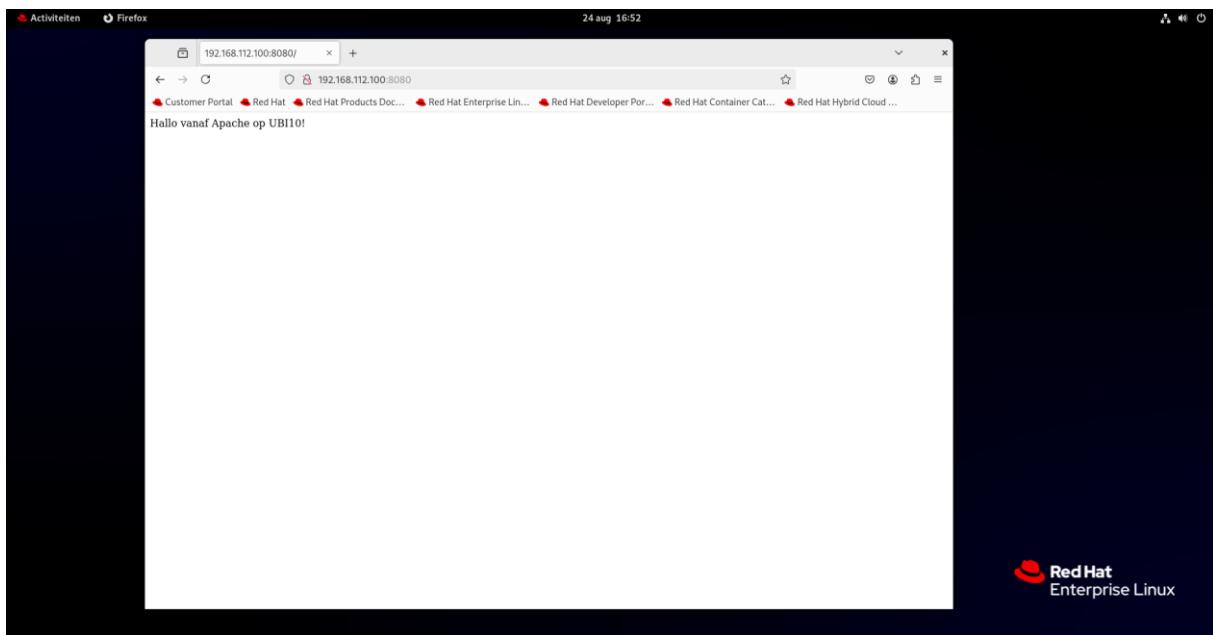
```
@549bead12a05/ x student@serverXX:~ x + v
link/Loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
    valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
    valid_lft forever preferred_lft forever
2: ens160: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 65520 qdisc fq_codel state UNKNOWN group default qlen 1000
link/ether 3a:f4:17:46:43:cd 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::38f4:17ff:fe06:43cd/64 scope link
    valid_lft forever preferred_lft forever
[root@549bead12a05 /]# curl 192.168.112.100
Hallo vanaf Apache op UBI10!
[root@549bead12a05 /]# |
```

We testen nu vanuit de host of de webserver draait (let op: hier is het op poort 8080!).

```
student@serverXX:~$ curl 192.168.112.100:8080
Hallo vanaf Apache op UBI10!
```

```
student@serverXX:~ x + v
student@serverXX:~$ curl 192.168.112.100:8080
Hallo vanaf Apache op UBI10!
student@serverXX:~$ |
```

Uiteraard kan je dit ook testen vanuit de grafische omgeving.



Sluit de container met de webserver af.

```
[root@549bead12a05 /]# exit
exit
```

Zoals je hieronder kan zien draait de container met de webserver nu niet meer.

```
student@serverXX:~$ podman ps
CONTAINER ID  IMAGE
COMMAND ...
```

...

Als je alle containers op je systeem toont, zowel de actieve als gestopte met de optie -a achter ps, zie je de container uiteraard nog staan.

```
student@serverXX:~$ podman ps -a  
CONTAINER ID  IMAGE                      COMMAND ...  
...  
549bead12a05  registry.access.redhat.com/ubi10/ubi:latest  /bin/bash  ...
```

4.5.2 DNS-server

We gaan een DNS-server bouwen in een container.

Om te beginnen starten we een nieuwe interactive detached container.

```
student@serverXX:~$ podman run -d registry.access.redhat.com/ubi10/ubi:latest
sleep infinity
```

```
b2071b2fa6506f5ea6fae5ea522c4558f9fb40b1fa7362440e72352eb2d05b94
```

- -d: start de container in de achtergrond (detached).
- sleep infinity: houdt de container in leven zodat je kunt inloggen en configureren.

We gaan nu in de container als volgt

```
student@serverXX:~$ podman exec -it b2 bash
```

- podman exec -it dns bash: open een interactieve shell in de container.

We vragen nu de hostname op en kijken naar de IP-instellingen.

```
[root@b2071b2fa650 /]# cat /etc/hostname
b2071b2fa650

[root@b2071b2fa650 /]# ip addr show dev ens160
2: ens160: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 65520 qdisc fq_codel state
UNKNOWN group default qlen 1000
    link/ether 9a:41:78:f9:0e:74 brd ff:ff:ff:ff:ff:ff
        inet 192.168.112.100/24 brd 192.168.112.255 scope global noprefixroute
            ens160
```

- De container toont, zoals verwacht, 192.168.112.100 als IP-nummer.

We installeren BIND en bijhorende tools in de container.

```
[root@b2071b2fa650 /]# dnf -y install bind bind-utils nano
```

```
...
```

We maken nu een primaire DNS-zone abc.pri aan op een DNS-server. Geef /etc/named.conf onderstaande inhoud.

```
[root@b2071b2fa650 /]# nano /etc/named.conf
options {
    directory "/var/named";
    recursion yes;
    allow-recursion { any; };
```

```

// kies forwarders die vanaf jouw netwerk bereikbaar zijn:
forwarders { 8.8.8.8; 8.8.4.4; };

forward only;

// in containers vaak handig, tenzij je klok/anchors perfect zijn:
dnssec-validation no;

// (of: dnssec-validation auto; als je tijd NTP-juist is en validatie
wilt)

};

logging {
    channel default_debug {
        file "data/named.run";
        severity dynamic;
    };
};

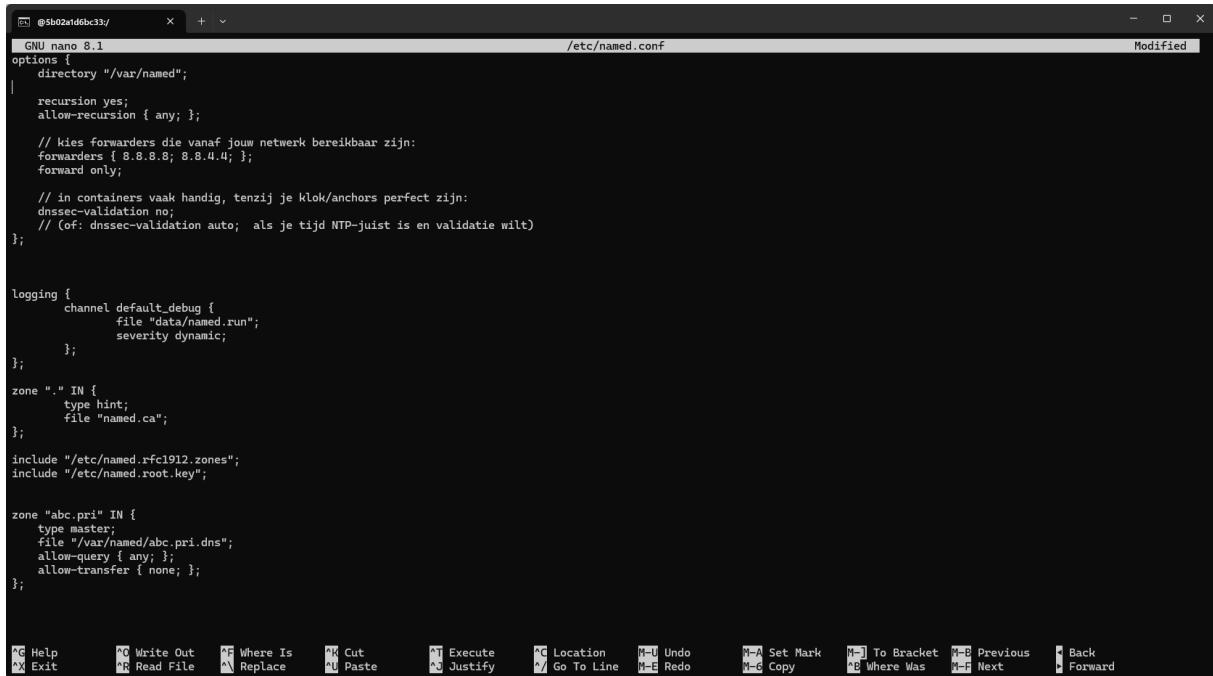
zone "." IN {
    type hint;
    file "named.ca";
};

include "/etc/named.rfc1912.zones";
include "/etc/named.root.key";

zone "abc.pri" IN {
    type master;
    file "/var/named/abc.pri.dns";
    allow-query { any; };
    allow-transfer { none; };
};

```

```
};
```



```
GNU nano 8.1                               /etc/named.conf                                Modified
options {
    directory "/var/named";
    recursion yes;
    allow-recursion { any; };

    // kies forwarders die vanaf jouw netwerk bereikbaar zijn:
    forwarders { 8.8.8.8; 8.8.4.4; };
    forward only;

    // in containers vaak handig, tenzij je klok/anchors perfect zijn:
    dnssec-validation no;
    // (of: dnssec-validation auto; als je tijd NTP-juist is en validatie wilt)
};

logging {
    channel default_debug {
        file "data/named.run";
        severity dynamic;
    };
};

zone "." IN {
    type hint;
    file "named.ca";
};

include "/etc/named.rfc1912.zones";
include "/etc/named.root.key";

zone "abc.pri" IN {
    type master;
    file "/var/named/abc.pri.dns";
    allow-query { any; };
    allow-transfer { none; };
};
```

- In dit geval wordt een DNS-zone met de naam abc.pri aangemaakt. Het gebruik van .pri (privé) is een gangbare praktijk voor interne netwerken, hoewel dit een niet-geregistreerde topleveldomeinnaam is. Dit betekent dat het domein abc.pri niet publiek toegankelijk is op internet, maar alleen binnen je interne netwerk wordt gebruikt.
- De zonefile waarnaar verwezen wordt, specificeert de bestandsnaam waarin de DNS-zone wordt opgeslagen. In dit geval wordt het zonebestand opgeslagen als abc.pri.dns.

We zullen nu een zonebestand aanmaken.

```
[root@b2071b2fa650 /]# nano /var/named/abc.pri.dns
```

```
$TTL 8h
$ORIGIN abc.pri.

@ IN SOA ns1.abc.pri. hostmaster.abc.pri. (
    2025031701 ; serial (YYYYMMDDnn)
    1d          ; vernieuwingsperiode
    3h          ; herhalingsperiode
    3d          ; vervaltijd
    3h          ; minimum TTL
)
IN NS ns1.abc.pri.

ns1 IN A 127.0.0.1
host IN A 192.168.1.100
```

```

GNU nano 8.1                               /var/named/abc.pri.dns
$TTL 8h
$ORIGIN abc.pri.
@ IN SOA ns1.abc.pri. hostmaster.abc.pri. (
    2025031701 ; serial (YYYYMMDDnn)
    1d          ; vernieuwingsperiode
    3h          ; herhalingsperiode
    3d          ; vervaltijd
    3h          ; minimum TTL
)
IN NS ns1.abc.pri.
ns1 IN A 127.0.0.1
host IN A 192.168.1.100

```

Help Write Out Where Is Cut Execute Location Undo Set Mark To Bracket Previous Back
Exit Read File Replace Paste Justify Go To Line Undo Redo Copy Where Was Next Forward

Onderstaande commando's zorgen ervoor dat de zonefiles alleen gelezen kunnen worden door user en group named.

```
[root@b2071b2fa650 /]# chown -R named:named /var/named
[root@b2071b2fa650 /]# chmod 770 -R /var/named
```

We checken nu de syntax van de configuratiebestanden.

```
- named:conf
[root@b2071b2fa650 /]# named-checkconf -z /etc/named.conf

zone localhost.localdomain/IN: loaded serial 0

zone localhost/IN: loaded serial 0

zone
1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.ip6.arpa/IN:
loaded serial 0

zone 1.0.0.127.in-addr.arpa/IN: loaded serial 0

zone 0.in-addr.arpa/IN: loaded serial 0

zone abc.pri/IN: loaded serial 2025031701

- abc.pri.dns
[root@ b2071b2fa650 /]# named-checkzone abc.pri /var/named/abc.pri.dns

zone abc.pri/IN: loaded serial 2025031701

OK
```

We starten nu named op in de voorgrond (-g) met de user named.

```
[root@b2071b2fa650 /]# named -g -u named
```

```

24-Aug-2025 17:01:10.815 command channel listening on 127.0.0.1#953
24-Aug-2025 17:01:10.816 configuring command channel from '/etc/rndc.key'
24-Aug-2025 17:01:10.816 command channel listening on :1#953
24-Aug-2025 17:01:10.816 not using config file logging statement for logging due to -g option
24-Aug-2025 17:01:10.817 managed-keys-zone: loaded serial 0
24-Aug-2025 17:01:10.817 zone 1.0.0.127.in-addr.arpa/IN: loaded serial 0
24-Aug-2025 17:01:10.817 zone abc.pri/IN: loaded serial 2025031701
24-Aug-2025 17:01:10.817 zone 0.in-addr.arpa/IN: loaded serial 0
24-Aug-2025 17:01:10.818 zone localhost.localdomain/IN: loaded serial 0
24-Aug-2025 17:01:10.818 zone 1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.ip6.arpa/IN: loaded serial 0
24-Aug-2025 17:01:10.819 zone localhost/IN: loaded serial 0
24-Aug-2025 17:01:10.819 all zones loaded
24-Aug-2025 17:01:10.819 running
24-Aug-2025 17:01:10.846 managed-keys-zone: Key 20326 for zone . is now trusted (acceptance timer complete)
24-Aug-2025 17:01:10.846 managed-keys-zone: Key 38696 for zone . is now trusted (acceptance timer complete)

```

We checken eerst dat we verbinding kunnen maken met het internet in de container.

Check dit door terug verbinding te maken met dezelfde container in een ander venster of tabblad.

```
student@serverXX:~$ podman exec -it b2 bash
```

Installeer iputils om ping te kunnen gebruiken

```
[root@b2071b2fa650 /]# dnf install -y iputils
[root@ b2071b2fa650 /]# ping -c 1 www.google.be
PING www.google.be (142.250.179.99) 56(84) bytes of data.
...
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 25.454/25.454/25.454/0.000 ms
```

We gaan nu in de container een nslookup doen van een DNS-naam gebruik makend van de juist geïnstalleerde DNS-server (voer het enkele keren uit als het niet werkt – het duurt even).

```
[root@b2071b2fa650 /]# nslookup host.abc.pri 127.0.0.1
Server:      127.0.0.1
Address:     127.0.0.1#53
```

```
Name:   host.abc.pri
```

```
Address: 192.168.1.100
```

Je bekomt uiteraard hetzelfde resultaat met “nslookup host.abc.pri 192.168.112.100” aangezien we in de container reeds hebben vastgesteld dat IP-nummer van de container als 192.168.112.100 wordt gezien.

```
[root@7b4382105731 /]# nslookup host.abc.pri 192.168.112.100
```

```
Server:      192.168.112.100
```

```
Address:     192.168.112.100#53
```

```
Name:   host.abc.pri
```

```
Address: 192.168.1.100
```

We verlaten de container door exit te typen.

```
[root@b2071b2fa650 /]# exit
```

```
exit
```

```
student@serverXX:~$
```

We gaan nu de actieve containers tonen.

```
student@serverXX:~$ podman ps  
  
CONTAINER ID  IMAGE                      COMMAND ...  
b2071b2fa650  registry.access.redhat.com/ubi10/ubi:latest  sleep infinity ...
```

Stop nu de container waarin we de DNS-server hebben geïnstalleerd.

```
student@serverXX:~$ podman stop b2  
WARN[0010] StopSignal SIGTERM failed to stop container crazy_feistel in 10  
seconds, resorting to SIGKILL b2
```

Podman stuurt een SIGTERM-signal naar het primaire proces (PID 1) dat binnen de container draait. Dit signaal geeft aan dat het proces de kans om netjes af te sluiten.

Podman wacht standaard 10 seconden na het SIGTERM-signal. Als het proces binnen deze tijd beëindigt, wordt de container netjes gestopt en gaat het in de status "stopped".

Als het proces na de time-outperiode nog steeds niet is gestopt, stuurt Docker een SIGKILL-signal naar het proces. Dit forceert een onmiddellijke beëindiging van het proces zonder de mogelijkheid om nog schoonmaakwerkzaamheden uit te voeren.

Je kan dit probleem oplossen door --init toe te voegen zoals je hieronder terugvindt. Je moet dan wel de hele paragraaf opnieuw doorlopen...

```
podman run --init -d registry.access.redhat.com/ubi10/ubi:latest sleep infinity  
--init zal ervoor zorgen dat er een klein init-systeem wordt gestart als PID 1. Dit init-proces stuurt signalen door naar het hoofdproces.
```

Nadat het primaire proces van de container is gestopt (of geforceerd is beëindigd), gaat de container naar de gestopte status (stopped), maar de container zelf blijft op het systeem aanwezig. Dit betekent dat je de container later opnieuw kunt starten met het commando docker start <container-id>.

We gaan nu een container image afleiden van deze container.

```
student@serverXX:~$ podman commit b2 contdns  
Getting image source signatures  
  
Copying blob 09bf1696a91e skipped: already exists  
  
Copying blob cf6810d4c970 done  |  
  
Copying config 25f89a133b done  |  
  
Writing manifest to image destination  
  
25f89a133bdd719f5b6ad1fcea4e03973adbc7895dd6a1f7d4c77d97316ae930
```

Het commando podman commit wordt gebruikt om een nieuwe Podman-image te maken op basis van de huidige staat van een draaiende of gestopte container. Dit betekent dat je de wijzigingen die je in een container hebt aangebracht (zoals geïnstalleerde software of aangepaste configuraties) kunt opslaan in een nieuwe image, zodat je deze later opnieuw kunt gebruiken of delen.

Bekijk nu de container images op je computer.

```
student@serverXX:~$ podman images
REPOSITORY          TAG      IMAGE ID      ...
localhost/contdns  latest   25f89a133bdd  ...
...
...
```

Je ziet dat er een container image is bijgekomen met de naam contdns.

Voer nu onderstaande uit.

```
student@serverXX:~$ podman run --detach contdns
6eacec2c8f7fcfd8e295d1e8c85a65583f4c1fc0bf3f95fe9ba66399ce15743
```

De container zal:

- Opstarten;
- Sleep infinity uitvoeren;

We starten in de container named op.

```
student@serverXX:~$ podman exec -it 6e bash
[root@6eacec2c8f7f /]# named -g -u named
```

Maak in een ander venster of tabblad nogmaals verbinding met de container en doe een DNS-aanvraag.

```
student@serverXX:~$ podman exec -it 6e bash
[root@6eacec2c8f7f /]# nslookup host.abc.pri 127.0.0.1
Server:      127.0.0.1
Address:     127.0.0.1#53

Name:   host.abc.pri
Address: 192.168.1.100
```

We verlaten nu deze container.

```
[root@6eacec2c8f7f /]# exit
exit
```

```
student@serverXX:~$
```

Aangezien we geen portmapping hebben ingesteld voor de container kunnen we niet vanuit de container host verbinding maken met de DNS-server.

```
student@serverXX:~$ nslookup host.abc.pri 192.168.112.100  
;; connection timed out; no servers could be reached  
...
```

We zullen dat oplossen.

We sluiten hiervoor eerst de container waarin de DNS-server is geïnstalleerd af. Ik gebruik nu kill i.p.v. stop omdat podman kill onmiddellijk een SIGKILL-signal stuurt.

```
student@serverXX:~$ podman kill 6e
```

Voor een DNS-server heb je poort 53 nodig, en wel in twee varianten:

- UDP/53 wordt gebruikt voor bijna alle standaard DNS-queries (snelle vraag/antwoord).
- TCP/53 wordt gebruikt voor zone-transfers en voor antwoorden die groter zijn dan één UDP-pakket.

We trachten nu terug een container instantie af te leiden van het image contdns maar zetten UDP en TCP-poort 53 open.

```
student@serverXX:~$ podman run -p 53:53/udp -p 53:53/tcp --detach contdns  
Error: pasta failed with exit code 1:  
Failed to bind port 53 (Permission denied) for option '-t 53-53:53-53', exiting
```

Je krijgt nu een foutmelding omdat een gewone gebruiker geen poorten kleiner dan 1024 open kan zetten.

We proberen dit dan eens als root.

```
[student@...] $ sudo podman run -p 53:53/udp -p 53:53/tcp --detach contdns  
[sudo] wachtwoord voor student:  
? Please select an image:  
▶ registry.access.redhat.com/contdns:latest  
registry.redhat.io/contdns:latest  
docker.io/library/contdns:latest
```

Je kan eender welk online register gebruiken. Het zal niet werken. Je wil immers gebruik maken van de lokale image contdns.

Een image die je als gewone user hebt gemaakt (localhost/contdns:latest) is **niet zichtbaar** voor root of eender welke andere gebruiker, en omgekeerd.

We tonen de container images van student.

```
student@serverXX:~$ podman images

REPOSITORY          TAG      IMAGE ID ...
localhost/contdns  latest   25f89a133bdd ...

...
```

We tonen nu de container images van root.

```
student@serverXX:~$ sudo podman images

REPOSITORY  TAG      IMAGE ID      CREATED      SIZE
```

Je moet het image exporteren als student en opnieuw importeren als root om het image voor de root beschikbaar te maken.

```
student@serverXX:~$ podman save -o contdns.tar localhost/contdns:latest

Copying blob 09bf1696a91e done    |
Copying blob cf6810d4c970 done    |
Copying config 25f89a133b done    |
Writing manifest to image destination
```

Ga nu over naar de gebruiker root en importeer het image.

```
[root@serverXX ~]# suo podman load -i /home/student/contdns.tar

student@serverXX:~$ sudo podman images

REPOSITORY          TAG      IMAGE ID      CREATED      SIZE
localhost/contdns  latest   25f89a133bdd  About an hour ago  276 MB
```

We leiden nu via sudo terug een container instantie af van het image contdns en zetten UDP en TCP-poort 53 open.

```
[student@... ~]$ sudo podman run -p 53:53/udp -p 53:53/tcp --detach contdns
dc983788962c85c57846bffb6446fa96f3f3bdcf6f78240ad2c2741a4d34da3
```

We starten in de container named op.

```
student@serverXX:~$ sudo podman exec -it dc bash
[root@6eacec2c8f7f /]# named -g -u named
```

Ga nu naar een ander venster of tabblad waar je aangemeld bent op serverXX.

Doe een nslookup van www.google.be om te kijken of forwarden werkt.

```
student@serverXX:~$ nslookup www.google.be 192.168.112.100
Server:      192.168.112.100
Address:     192.168.112.100#53
```

Non-authoritative answer:

```
Name:   www.google.be
Address: 142.250.179.99
Name:   www.google.be
Address: 2a00:1450:4007:818::2003
```

Als bovenstaande niet werkt wil dat zeggen dat er vanuit de container geen verbinding kan gemaakt worden met 8.8.8.8. Dit los je als volgt op (op de container host):

```
student@serverXX:~$ sudo iptables -t nat -A POSTROUTING -s 10.88.0.0/16 -o ens160 -j MASQUERADE
```

```
student@serverXX:~$ sudo iptables -I FORWARD 1 -s 10.88.0.0/16 -o ens160 -j ACCEPT
```

```
student@serverXX:~$ sudo iptables -I FORWARD 1 -d 10.88.0.0/16 -m state --state RELATED,ESTABLISHED -i ens160 -j ACCEPT
```

Doe een lookup van host.abc.pri.

```
student@serverXX:~$ nslookup host.abc.pri 192.168.112.100
Server:      192.168.112.100
Address:     192.168.112.100#53
```

```
Name:   host.abc.pri
Address: 192.168.1.100
```

Ook dit werkt!

We sluiten de DNS-server, af.

```
student@serverXX:~$ sudo podman kill dc
```

4.6 Containers starten, stoppen en verwijderen

Voor dit onderdeel verwijderen we eerst alle containers.

```
student@serverXX:~$ podman rm $(podman ps -a -q) -f
```

We komen later erop terug hoe dit werkt.

Je kan ook als volgt alle containers verwijderen.

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

Voor deze paragraaf starten we eerst 2 keer ubi10/ubi op op een dusdanige manier dat de container blijft draaien.

```
student@serverXX:~$ podman run -i -t -d ubi10/ubi
```

```
54f1575ffceca735d92b256cf7181ed8eef1993005d2700b89e2f027fed0a484
```

```
student@serverXX:~$ podman run -it -d ubi10/ubi
```

```
88a2cf918c2d5f4985be19efd3e00557d512fc7942cf37bc2c32352d2a7392f
```

- *-i* zorgt ervoor dat STDIN (standaardinvoer) open blijft, zelfs als je niet bent verbonden.
- *-t* zorgt voor een pseudo-terminal zodat de container zich gedraagt alsof je in een normale terminal zit.
- Container draait in de achtergrond door optie *-d*.
- Het bash-proces is het hoofdproces in de container. Omdat Bash actief blijft, blijft de container ook draaien. Zodra Bash stopt (bijvoorbeeld als je exit uitvoert in de container), zal de container automatisch stoppen.
- Je mag de opties ook combineren (*-i -t -d* kan je ook schrijven als *-it*).

-i en *-d* lijken tegenstrijdig maar als je geen *-it* meegeeft en je start een container in detached mode gebeurt het volgende:

- De container start.
- */bin/bash* merkt dat er geen terminal (TTY) beschikbaar is.
- Bash gaat dan in non-interactive mode → het voert niets uit, sluit meteen af.
- Resultaat → je container stopt onmiddellijk.

We zullen ook 2 containers opstarten die zichzelf onmiddellijk afsluiten.

```
student@serverXX:~$ podman run -d ubi10/ubi
```

```
8c728e96ce0e939ec839fb1ff20e583924a1a1f056c33de0ecc3feb2c0edb9b6
```

```
student@serverXX:~$ podman run -d ubi10/ubi
```

```
553cb8581f98265a4358b103cf655f8a41aa5e72e892e882b2ccb6fc1d467d98
```

We tonen nu alle containers van de gebruiker student.

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

CONTAINER ID	IMAGE	COMMAND	...
54f1575ffcec	registry.access.redhat.com/ubi10/ubi:latest	/bin/bash ...	
88a2cf918c2d	registry.access.redhat.com/ubi10/ubi:latest	/bin/bash ...	
8c728e96ce0e	registry.access.redhat.com/ubi10/ubi:latest	/bin/bash ...	
553cb8581f98	registry.access.redhat.com/ubi10/ubi:latest	/bin/bash ...	

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
54f1575ffcec	registry.access.redhat.com/ubi10/ubi:latest	/bin/bash	2 minutes ago	Up 2 minutes		flamboyant_bhabha
88a2cf918c2d	registry.access.redhat.com/ubi10/ubi:latest	/bin/bash	2 minutes ago	Up 2 minutes		pedantic_shamir
8c728e96ce0e	registry.access.redhat.com/ubi10/ubi:latest	/bin/bash	46 seconds ago	Exited (0) 47 seconds ago		awesome_torvalds
553cb8581f98	registry.access.redhat.com/ubi10/ubi:latest	/bin/bash	45 seconds ago	Exited (0) 45 seconds ago		sleepy_shannon

Dit geeft een lijst weer van alle containers op het systeem van de gebruiker student, zowel actieve (draaiende) als gestopte containers.

We tonen nu alle actieve containers van de gebruiker student.

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

CONTAINER ID	IMAGE	COMMAND	...
54f1575ffcec	registry.access.redhat.com/ubi10/ubi:latest	/bin/bash ...	
88a2cf918c2d	registry.access.redhat.com/ubi10/ubi:latest	/bin/bash ...	

We gaan nu de draaiende container met de container ID beginnend met 54 afsluiten.

```
student@serverXX:~$ podman stop 54
WARN[0010] StopSignal SIGTERM failed to stop container flamboyant_bhabha in 10
seconds, resorting to SIGKILL
```

54

Een proces dat als PID 1 draait, krijgt standaard geen signalen door zoals je zou verwachten. Dit betekent dat /bin/bash in de container geen SIGTERM goed afhandelt. Vandaar dat na 10 seconden SIGKILL is gebruikt om de container af te sluiten.

Het UBI10-init image is speciaal gemaakt om precies dit probleem op te lossen.

We proberen dit uit door eerst de ubi10-init image naar de lokale registry te downloaden.

```
student@serverXX:~$ podman pull registry.access.redhat.com/ubi10/ubi-
init:latest

Trying to pull registry.access.redhat.com/ubi10/ubi-init:latest...
Getting image source signatures
Checking if image destination supports signatures
Copying blob c6edde6ffca0 skipped: already exists
```

```

Copying blob 1094cf3c2cfb done    |
Copying config a561029cd0 done    |
Writing manifest to image destination
Storing signatures
a561029cd0f8c767f5c3a73c1937651bb7c14bab8f366a14b766610b8c9dffe8

```

We starten nu terug een container die blijft draaien, maar nu van ubi-init.

```

student@serverXX:~$ podman run -i -t -d ubi10/ubi-init
457172280bf896feb4dfc6d4528a1653a3406dea071eed51ae6e380c76fd329f

```

We checken of de container draait.

```

student@serverXX:~$ podman ps

CONTAINER ID  IMAGE                                     COMMAND ...
88a2cf918c2d  registry.access.redhat.com/ubi10/ubi:latest   /bin/bash ...
457172280bf8  registry.access.redhat.com/ubi10/ubi-init:latest /sbin/init ...

```

Zoals je ziet draait de container nog en bij het commando (COMMAND) staat /sbin/init i.p.v. /bin/bash.

Hieronder vind je het verschil tussen ubi10 en ubi10-init terug.

Kenmerk	ubi10 (standaard)	ubi10-init (met init-proces)
Init-proces aanwezig	Nee	Ja (systemd)
PID 1 = /bin/bash	Ja, als je bash start	Nee, systemd draait als PID 1
Signaalafhandeling	SIGTERM vaak genegeerd	SIGTERM correct doorgegeven
Container stopt netjes	Vaak niet	Ja
Gebruikssituatie	kleine containers en testcases	productieservices en lange sessies

We trachten nu de container 45 te stoppen.

```

student@serverXX:~$ podman stop 45
45

```

De container wordt nu op een mooie manier afgesloten.

De container zal trouwens niet worden verwijderd!

Om de container terug te starten gebruik je onderstaande.

```
student@serverXX:~$ podman start 45  
45
```

Om een container te verwijderen gebruik je “podman rm”.

```
student@serverXX:~$ podman rm 45  
Error: cannot remove container  
457172280bf896feb4dfc6d4528a1653a3406dea071eed51ae6e380c76fd329f as it is  
running - running or paused containers cannot be removed without force:  
container state improper
```

De container kan niet verwijderd worden omdat de container niet gestopt is. Om te verwijderen voeren we dan ook het volgende uit.

```
student@serverXX:~$ podman stop 45; podman rm 45  
45  
45
```

Om alle (zowel draaiende als gestopte) container ID's op je container host te tonen doe je het volgende.

```
student@serverXX:~$ podman ps -a -q  
54f1575ffcec  
88a2cf918c2d  
8c728e96ce0e  
553cb8581f98
```

Je kan, gebruik makend van command substitution (remember Linux essentials...), alle containers stoppen.

```
student@serverXX:~$ podman stop $(podman ps -a -q)  
WARN[0010] StopSignal SIGTERM failed to stop container pedantic_shamir in 10  
seconds, resorting to SIGKILL  
88a2cf918c2d  
8c728e96ce0e  
553cb8581f98  
54f1575ffcec
```

In plaats van “`podman stop $(podman ps -a -q)`” kan je ook gebruik maken van “`podman stop --all`”.

```
student@serverXX:~$ podman stop --all
```

...

We vragen de containers op die draaien.

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

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

Uiteraard zijn er nu zo geen meer.

Je kan alle containers op een eenvoudige manier verwijderen.

```
student@serverXX:~$ podman rm $(podman ps -a -q) --force  
8c728e96ce0e  
553cb8581f98  
54f1575ffcec  
88a2cf918c2d
```

Ook zou je alle containers kunnen verwijderen met podman rm --all.

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

Als je alle containers nu opvraagt zie je dat er geen meer zijn.

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

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

Om gestopte containers te verwijderen.

```
student@serverXX:~$ podman container prune  
WARNING! This will remove all non running containers.  
Are you sure you want to continue? [y/N] y
```

Er zijn uiteraard geen containers verwijderd omdat we hiervoor al alle containers (actieve en niet-actieve) hebben verwijderd.

We gaan nu de container images tonen.

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

REPOSITORY	TAG	IMAGE ID ...
localhost/contdns	latest	25f89a133bdd ...
registry.access.redhat.com/ubi10/ubi-init	latest	a561029cd0f8 ...
registry.access.redhat.com/ubi10/ubi	latest	620bc887852d ...
...		

Om het docker image localhost/contdns te verwijderen voer je het volgende uit.

```
student@serverXX:~$ podman image rm localhost/contdns
Untagged: localhost/contdns:latest
Deleted: 25f89a133bdd719f5b6ad1fce4e03973adbc7895dd6a1f7d4c77d97316ae930
```

In plaats van “podman image rm” kan je ook gebruik maken van “podman rmi” om een image te verwijderen.

Alleen de localhost/contdns -image wordt verwijderd en geen andere images. Andere images blijven onaangeraakt tenzij je ze expliciet verwijdert.

```
student@serverXX:~$ podman images
REPOSITORY                      TAG      IMAGE ID ...
registry.access.redhat.com/ubi10/ubi-init  latest   a561029cd0f8 ...
registry.access.redhat.com/ubi10/ubi       latest   620bc887852d ...
...
...
```

Om alle container images te verwijderen gebruik je onderstaande.

```
student@serverXX:~$ podman image rm --all --f
Untagged: localhost/contdns:latest
Untagged: registry.access.redhat.com/ubi10/ubi-init:latest
Untagged: registry.access.redhat.com/ubi10/ubi:latest
...
...
```

4.7 Een Container Image bouwen met een containerfile

4.7.1 Inleiding

Wat we tot nu toe hebben gedaan, was een interactieve manier om containers te bouwen.

Het is echter niet de meest efficiënte manier.

- Toegang verkregen tot de container.
- Instructies gegeven over hoe deze zichzelf moet configureren.
- De container gestopt.
- Een image gecreëerd op basis van wat we hadden gemaakt.

Daarnaast hebben we de container op een slechte manier afgesloten.

Het proces bestond uit een reeks commando's.

Met containerfiles kun je een container-image creëren. Dit zijn speciale bestanden die uniek zijn voor Docker en een eigen syntax hebben. Je kan Dockerfiles gebruiken met podman.

4.7.2 Algemene procedure

Om een container-image te bouwen aan de hand van een Dockerfile, volg je onderstaande stappen:

- Maak een containerfile
Plaats een bestand genaamd Dockerfile (mag ook andere naam hebben maar dan moet je later extra parameters meegeven) in een directory en schrijf daarin de instructies die het image opbouwen.
 - o FROM: identificeert van welke initiële image we dit proces willen starten.
 - o COPY: kopieert de inhoud van een bestand op je host naar een locatie binnen het image.
 - o RUN: hierachter plaats je een opdracht die uitgevoerd wordt als de container is gestart.
 - o CMD of ENTRYPOINT: standaardcommando dat de container start. We bespreken dit later.
- Bouw het image
Navigeer naar de directory met de Dockerfile en voer het commando uit.
podman build --format=docker -t <image-naam> .
 - o Het -t-vlaggetje geeft de naam en optioneel een tag aan het image, en de . verwijst naar de huidige directory met de Dockerfile.
- Run de container
Nadat het image is aangemaakt kan je een container draaien gebaseerd op de zelfgemaakte image.

Na dit proces wordt het image aangemaakt en kun je deze gebruiken om containers te starten.

4.7.3 Praktijk

4.7.3.1 Container bouwen met Dockerfile

We maken eerst een mappenstructuur aan in de homedirectory van student.

```
student@serverXX:~$ mkdir -p ~/oefening/scripts
```

Maak Dockerfile aan met nano.

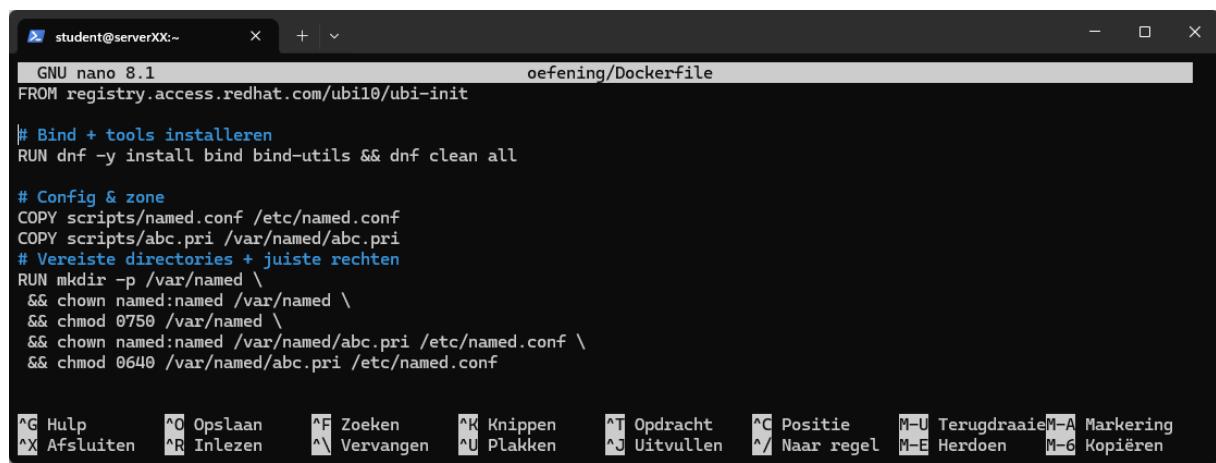
```
student@serverXX:~$ nano oefening/Dockerfile
```

We gaan nu onderstaande inhoud (zoals reeds uitgelegd) toevoegen aan de Dockerfile.

```
FROM registry.access.redhat.com/ubi10/ubi-init

# Bind + tools installeren
RUN dnf -y install bind bind-utils && dnf clean all

# Config & zone
COPY scripts/named.conf /etc/named.conf
COPY scripts/abc.pri /var/named/abc.pri
# Vereiste directories + juiste rechten
RUN mkdir -p /var/named \
&& chown named:named /var/named \
&& chmod 0750 /var/named \
&& chown named:named /var/named/abc.pri /etc/named.conf \
&& chmod 0640 /var/named/abc.pri /etc/named.conf
```



The screenshot shows a terminal window titled 'student@serverXX:~' with the file 'oefening/Dockerfile' open in the nano editor. The content of the Dockerfile is displayed, including the FROM instruction, the RUN command for installing bind and its utilities, and the COPY commands for the configuration files. The nano editor interface includes standard keyboard shortcuts for operations like Help (^G), Save (^O), Find (^F), Cut (^K), Paste (^T), Undo (^C), Redo (^U), and Copy (^M).

```
GNU nano 8.1          oefening/Dockerfile
FROM registry.access.redhat.com/ubi10/ubi-init

# Bind + tools installeren
RUN dnf -y install bind bind-utils && dnf clean all

# Config & zone
COPY scripts/named.conf /etc/named.conf
COPY scripts/abc.pri /var/named/abc.pri
# Vereiste directories + juiste rechten
RUN mkdir -p /var/named \
&& chown named:named /var/named \
&& chmod 0750 /var/named \
&& chown named:named /var/named/abc.pri /etc/named.conf \
&& chmod 0640 /var/named/abc.pri /etc/named.conf
```

Sla op en sluit af met <CTRL> + X en J (Gewijzigde buffer opslaan) .

We maken nu de bestanden aan die gekopieerd worden van serverXX naar de container.

```
student@serverXX:~$ nano oefening/scripts/named.conf
```

```
options {
```

```

        directory "/var/named";

        listen-on port 53 { any; };

        allow-query { any; };

        recursion yes;

        allow-recursion { any; };

        forwarders { 8.8.8.8; 8.8.4.4; };

        forward only;

        dnssec-validation no;

};


```

```

zone "abc.pri" {

    type master;

    file "abc.pri";

};


```

Sla op en sluit af met <CTRL> + X en J (Gewijzigde buffer opslaan) .

```

student@serverXX:~$ nano oefening/scripts/abc.pri

$TTL 8h

$ORIGIN abc.pri.

@ IN SOA ns1.abc.pri. hostmaster.abc.pri. (
    2025031701 ; serial (YYYYMMDDnn)
    1d          ; vernieuwingsperiode
    3h          ; herhalingsperiode
    3d          ; vervaltijd
    3h          ; minimum TTL
)

IN NS ns1.abc.pri.

ns1 IN A 127.0.0.1

host IN A 192.168.1.100


```

Sla op en sluit af met <CTRL> + X en J (Gewijzigde buffer opslaan) .

We gaan nu naar de directory waarin de dockerfile staat en bouwen een container image aan de hand hiervan.

```
student@serverXX:~$ cd oefening

[student@serverXX oefening]$ podman build --format=docker -t contdns .

STEP 1/5: FROM registry.access.redhat.com/ubi10/ubi-init
STEP 2/5: RUN dnf -y install bind bind-utils && dnf clean all
...
STEP 3/5: COPY scripts/named.conf /etc/named.conf
--> c0547c29517c

STEP 4/5: COPY scripts/abc.pri      /var/named/abc.pri
--> bf6f089e3b79

STEP 5/5: RUN mkdir -p /var/named && chown named:named /var/named && chmod 0750 /var/named && chown named:named /var/named/abc.pri /etc/named.conf && chmod 0640 /var/named/abc.pri /etc/named.conf

COMMIT contdns

--> a0690336fccc

Successfully tagged localhost/contdns:latest
dde70813780a32d9acace35b7ce8de22421ffa8fd643e9d9362646acad1f31ee
```

We checken nu dat het image contdns bestaat.

```
[student@serverXX oefening]$ podman images

REPOSITORY          TAG      IMAGE ID ...
localhost/contdns  latest   dde70813780a ...
...
...
```

We starten nu een container aan de hand van de contdns image.

```
[student@serverXX oefening]$ podman run -d -p 1053:53/udp -p 1053:53/tcp
contdns /usr/sbin/named -g -u named -c /etc/named.conf
```

We kijken nu of de container draait.

```
[student@serverXX oefening]$ podman ps

CONTAINER ID  IMAGE          COMMAND ...
6e7cd2aab566  localhost/contdns:latest  /usr/sbin/named -...
```

In dit voorbeeld gebruiken we poort 1053 op de host zodat we niet het image moeten kopiëren naar de root.

We kunnen nu met dig kijken of de container vertalingen kan maken.

```
[student@serverXX oefening]$ dig @127.0.0.1 -p 1053 host.abc.pri
```

```
...  
;; ANSWER SECTION:  
  
host.abc.pri.          28800    IN      A       192.168.1.100  
...
```

4.7.3.2 Naam van container en hostname kiezen

Er is een willekeurige container ID gekozen bij het aanmaken van de container aan de hand van het image contdns.

Als je “podman ps” uitvoert zie je altijd het eerste stukje van deze container ID bij een container staan.

```
[student@serverXX oefening]$ podman ps  
  
CONTAINER ID  IMAGE ...          NAMES  
6e7cd2aab566  localhost/contdns:latest...  busy_wozniak
```

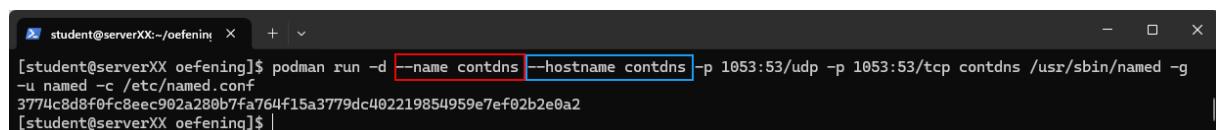
Als je een container aanmaakt in Podman zonder explicet een naam op te geven, genereert Podman automatisch een unieke naam voor de container. Deze naam wordt gegenereerd op basis van een combinatie van een willekeurig gekozen bijvoeglijk naamwoord (busy) en de achternaam van een bekende wetenschapper, uitvinder of hacker (wozniak in dit geval), gescheiden door een underscore (_).

We gaan nu de juist aangemaakte container verwijderen aan de hand van die naam.

```
[student@serverXX oefening]$ podman rm busy_wozniak -f  
-f zorgt ervoor dat je de container kan verwijderen, ondanks dat deze gestart is.
```

We gaan nu een nieuwe container starten met de naam contdns en de hostname contdns en gebaseerd op het image contdns.

```
[student@serverXX oefening]$ podman run -d --name contdns --hostname contdns -p 1053:53/udp -p 1053:53/tcp contdns /usr/sbin/named -g -u named -c /etc/named.conf
```



A screenshot of a terminal window titled "student@serverXX:~/oefening". The command entered is "podman run -d --name contdns --hostname contdns -p 1053:53/udp -p 1053:53/tcp contdns /usr/sbin/named -g -u named -c /etc/named.conf". The output shows a long container ID: "3774c8d8f0fc8ec902a280b7fa764f15a3779dc402219854959e7ef02b2e0a2". The terminal prompt "[student@serverXX oefening]\$ |" is visible at the bottom right.

Je kan nu de zelfgekozen naam gebruiken om in de container te gaan.

```
[student@serverXX oefening]$ podman exec -it contdns bash
```



```
[student@serverXX oefening]$ podman exec -it contdns bash
[root@contdns /]# cat /etc/hostname
contdns
[root@contdns /]# |
```

Je kan de container ook op een gemakkelijke manier verwijderen (nadat je uit de container bent gegaan uiteraard met exit).

```
[student@serverXX oefening]$ podman rm contdns -f
WARN[0010] StopSignal (37) failed to stop container contdns in 10 seconds,
resorting to SIGKILL
contdns
```

Voor alle duidelijkheid: je hebt nu de container en niet het image verwijderd.

4.7.3.3 CMD en ENTRYPOINT instructies in dockerfiles

Tot nu toe moet je veel intypen om een nieuwe container te starten.

```
podman run -d --name contdns --hostname contdns -p 1053:53/udp -p 1053:53/tcp
contdns /usr/sbin/named -g -u named -c /etc/named.conf
```

Je kan hetgeen dat nodig is om de container draaiende te houden in de dockerfile plaatsen.

- Voeg CMD-commando toe: dit wordt uitgevoerd wanneer de container wordt geïnitialiseerd.
- CMD kan overruled worden door parameters toe te voegen als je de container start.

We voegen nu onderstaande toe aan de Dockerfile.

```
# Zorgt dat netjes gestopt wordt
STOPSIGNAL SIGTERM
# Uitvoeren bij initialiseren container
CMD ["/usr/sbin/named", "-g", "-u", "named", "-c", "/etc/named.conf"]
```

```

student@serverXX:~/oefening_1 Dockerfile
GNU nano 8.1
FROM registry.access.redhat.com/ubi10/ubi-init

# Bind + tools installeren
RUN dnf -y install bind bind-utils && dnf clean all

# Config & zone
COPY scripts/named.conf /etc/named.conf
COPY scripts/abc.pri /var/named/abc.pri

# Vereiste directories + juiste rechten
RUN mkdir -p /var/named \
&& chown named:named /var/named \
&& chmod 0750 /var/named \
&& chown named:named /var/named/abc.pri /etc/named.conf \
&& chmod 0640 /var/named/abc.pri /etc/named.conf

# Zorgt dat netjes gestopt wordt
STOPSIGNAL SIGTERM

# Uitvoeren bij initialiseren container
CMD ["/usr/sbin/named", "-g", "-u", "named", "-c", "/etc/named.conf"]

^G Hulp      ^O Opslaan      ^F Zoeken      ^K Knippen      ^T Opdracht      ^C Positie      M-U Terugdraaien      M-A Markering
^X Afsluiten  ^R Inlezen      ^V Vervangen     ^U Plakken      ^J Uitvullen     ^/ Naar regel    M-E Herdoen      M-G Kopiëren

```

Deze syntax bij CMD lijkt vrij ingewikkeld maar als je goed kijkt zie je dat in feite “/usr/sbin/named -g -u named -c /etc/named.conf” wordt uitgevoerd. Je kan ook mogelijk vereenvoudigen naar

```

# Uitvoeren bij initialiseren container
CMD /usr/sbin/named -g -u named -c /etc/named.conf

```

maar dan draait het via /bin/sh -c ... i.p.v. door Podman rechtstreeks. Signalen en PID1 gedrag werken via de syntax die wij hebben ingesteld zeker correct!

De CMD-instructie wordt gebruikt om een standaard commando of argumenten op te geven die moeten worden uitgevoerd wanneer een container start, tenzij de gebruiker zelf een ander commando opgeeft bij het starten van de container.

ENTRYPOINT wordt gebruikt om een vast commando of script in te stellen dat altijd wordt uitgevoerd wanneer de container start, ongeacht welke commando's of argumenten worden meegegeven bij het opstarten van de container.

We passen de dockerfile aan zodat ENTRYPOINT i.p.v. CMD gebruikt wordt.

```

# Uitvoeren bij initialiseren container
ENTRYPOINT ["/usr/sbin/named", "-g", "-u", "named", "-c", "/etc/named.conf"]

```

```

student@serverXX:~/oefening X + v
GNU nano 8.1 Dockerfile Gewijzigd

# Bind + tools installeren
RUN dnf -y install bind bind-utils && dnf clean all

# Config & zone
COPY scripts/named.conf /etc/named.conf
COPY scripts/abc.pri /var/named/abc.pri

# Vereiste directories + juiste rechten
RUN mkdir -p /var/named \
&& chown named:named /var/named \
&& chmod 0750 /var/named \
&& chown named:named /var/named/abc.pri /etc/named.conf \
&& chmod 0640 /var/named/abc.pri /etc/named.conf

# Zorgt dat netjes gestopt wordt
STOPSIGTERM

# Uitvoeren bij initialiseren container
ENTRYPOINT ["/usr/sbin/named", "-g", "-u", "named", "-c", "/etc/named.conf"]

^G Hulp      ^O Opslaan    ^F Zoeken      ^K Knippen      ^T Opdracht      ^C Positie      M-U Terugdraaien M-A Markering
^X Afsluiten  ^R Inlezen    ^R Vervangen    ^U Plakken      ^J Uitvullen    ^/ Naar regel   M-E Herdoen    M-6 Kopiëren

```

Maak nu een gewijzigd image aan.

```

[student@serverXX oefening]$ podman build --format=docker -t contdns .

STEP 1/6: FROM registry.access.redhat.com/ubi10/ubi-init

STEP 2/6: RUN dnf -y install bind bind-utils && dnf clean all
--> Using cache
17ba2f59f7459e88b844ec2797fdf39e15c5ea2a35df2fc5e9e46ea37373b535
--> 17ba2f59f745

STEP 3/6: COPY scripts/named.conf /etc/named.conf
--> 3badead4e3bf

STEP 4/6: COPY scripts/abc.pri      /var/named/abc.pri
--> d0d0cf430cf2

STEP 5/6: RUN mkdir -p /var/named && chown named:named /var/named && chmod
0750 /var/named && chown named:named /var/named/abc.pri /etc/named.conf &&
chmod 0640 /var/named/abc.pri /etc/named.conf
--> 54af3d1b3da7

STEP 6/6: ENTRYPOINT ["/usr/sbin/named", "-g", "-u", "named", "-c",
"/etc/named.conf"]

COMMIT contdns
--> 0ee7fc766ef8

Successfully tagged localhost/contdns:latest
0ee7fc766ef800830d765daf17ede3ea8e67c80d0af0b453f23f60f1a826fbdc

```

```
student@serverXX:~/oefening$ nano Dockerfile
student@serverXX:~/oefening$ podman build --format=docker -t contdns .
STEP 1/7: FROM registry.access.redhat.com/ubi10/ubi-init
STEP 2/7: RUN dnf -y install bind bind-utils && dnf clean all
--> Using cache 92dabc71ef7f1e76c04e446f8e92e504aba0a66f0d2d378eceebed71da576287
--> 92dabc71ef7f
STEP 3/7: COPY scripts/named.conf /etc/named.conf
--> Using cache 4e32f3b7c191be2062e86b2e2642eb779247b1b8dc0306bf6658efa64a629011
--> 4e32f3b7c191
STEP 4/7: COPY scripts/abc.pri /var/named/abc.pri
--> Using cache 036f9a85033bc141848ea8d370d1b22fd08417a5cd395d276611b91e3ef90044
--> 036f9a85033b
STEP 5/7: RUN mkdir -p /var/named && chown named:named /var/named && chmod 0750 /var/named && chown named:named /var/named/abc.pri /etc/named.conf && chmod 0640 /var/named/abc.pri /etc/named.conf
--> Using cache a590e2c7c4568076d018a81e0e7a016613d051cd3a18defcb07a25261d6e1c94
--> a590e2c7c456
STEP 6/7: STOPSIGAL SIGTERM
--> Using cache 419d2c910188b47afclaa5c97344ec00e71c7e4ce5456d9ce5f88e1a7da474d6
--> 419d2c910188
STEP 7/7: ENTRYPOINT ["/usr/sbin/named", "-q", "-u", "named", "-c", "/etc/named.conf"]
COMMIT contdns
--> b7f1fa953616
Successfully tagged localhost/contdns:latest
b7f1fa95361604fea0484bbb26e87e783ccb998f9151c3a5d1d59bc972e91f91
student@serverXX:~/oefening$ |
```

We inspecteren nu het aangemaakte image.

```
[student@serverXX oefening]$ podman image inspect contdns
```

```
student@serverXX:~/oefening > + <
34c65b62517d4198/work"
}
},
"RootFS": {
    "Type": "layers",
    "Layers": [
        "sha256:09bf1696a91e6b849cc8e091e7995881e207b58dc31ff0b0d243913667b2570d",
        "sha256:039ea7f02cb5a0c394b46a4d6bf16claa88fdc9d62c0bcc63ece9b54b3460a",
        "sha256:c95693e4c10adff2e372ef6b323329b5e45ebc9d34701c29478fa1a290f9a8a",
        "sha256:d2fcbaa6b5ba9b50af720b8fdcc824754a8eb98ed12637240fefef6b9d26b3730",
        "sha256:37230d64dede62be08a04lf03cb4ee4702d8f9a97elb983dfc57058477a49d68",
        "sha256:f3dc5c9dfa475662d05c09645e30a4f81fac065713006d81fcddf2a7f627d274"
    ]
},
"Labels": {
    "architecture": "x86_64",
    "build-date": "2025-08-21T08:19:38",
    "com.redhat.component": "ubi10-init-container",
    "com.redhat.license_terms": "https://www.redhat.com/en/about/red-hat-end-user-license-agreements#UBI",
    "description": "The Universal Base Image Init is designed to run an init system as PID 1 for running multi-services inside a container. This base image is freely redistributable, but Red Hat only supports Red Hat technologies through subscriptions for R"
}
```

Je ziet de verschillende lagen die gebruikt zijn.

We gaan een nieuwe container starten op basis van deze image. Zoals je juist gezien hebt zal je nu minder argumenten moeten meegeven (`/usr/sbin/named -g -u named -c /etc/named.conf` is niet meer nodig).

```
[student@serverXX oefening]$ podman run -d --name contdns --hostname contdns -p 1053:53/udp -p 1053:53/tcp contdns
```

bc1e8bc8e440012af0b372260f375b10032fbdb2f4f59beb94dac8e3c5369da7

Kijk nu eens naar de containers die actief zijn.

```
[student@serverXX oefening]$ podman ps
```

CONTAINER ID	IMAGE	COMMAND
bc1e8bc8e440	localhost/contdns:latest	

Je kan de container nu deftig stoppen en terug starten.

```
student@serverXX:~/oefening$ podman stop contdns  
contdns  
student@serverXX:~/oefening$ podman start contdns  
contdns
```

4.7.3.4 HEALTHCHECK instructie in dockerfile

HEALTHCHECK is een optie in een Dockerfile waarmee je Docker vertelt hoe te bepalen of een container gezond (healthy) of ongezond (unhealthy) is.

Een retourwaarde van nul geeft aan dat de container in goede gezondheid verkeert, terwijl elke andere waarde aangeeft dat de container ongezond is. Met deze instructies kan bijvoorbeeld een test op de DNS-server worden uitgevoerd om te controleren of de services en applicaties in de container correct functioneren en gezond zijn.

Om dit verder uit te leggen zullen we eerst naar de DNS-container gaan.

```
[student@serverXX oefening]$ podman exec -it contdns /bin/bash
```

We gaan gebruik maken van pgrep en installeren dat programma in de container.

```
[root@contdns /]# dnf install procps-ng
```

Het commando pgrep is nu beschikbaar. We checken in de contdns container of de DNS-server draait.

```
[root@contdns /]# pgrep named
```

1

- zoekt in de process-lijst (ps) naar een proces dat named heet (dat is de BIND DNS-daemon).
- Als het proces gevonden wordt → pgrep print het PID.
 - 1 in dit geval (dat klopt!)

De exit-code van pgrep zit in de speciale shellvariabele \$?, en die zie je alleen als je echo \$? Uitvoert.

```
[root@contdns /]# echo $?  
0
```

Je ziet hier exit-code 0: dat wil zeggen dat named is gevonden. Anders kreeg je exit-code 1.

We hebben de PID niet nodig om te bepalen of named draait. We kunnen dus "> /dev/null" toevoegen aan "pgrep named" om de output te verbergen.

```
[root@contdns /]# pgrep named > /dev/null
```

```
[root@contdns /]# echo $?
```

```
0
```

Met deze kennis passen we de Dockerfile aan.

```
[root@contdns /]# exit  
exit  
[student@serverXX oefening]$ nano Dockerfile  
  
FROM registry.access.redhat.com/ubi10/ubi-init  
  
# Bind + tools installeren  
  
RUN dnf -y install bind bind-utils procps-ng && dnf clean all  
  
  
# Config & zone  
  
COPY scripts/named.conf /etc/named.conf  
COPY scripts/abc.pri      /var/named/abc.pri  
  
  
# Vereiste directories + juiste rechten  
  
RUN mkdir -p /var/named \  
    && chown named:named /var/named \  
    && chmod 0750 /var/named \  
    && chown named:named /var/named/abc.pri /etc/named.conf \  
    && chmod 0640 /var/named/abc.pri /etc/named.conf  
  
  
# Zorgt dat netjes gestopt wordt  
  
STOPSIGNAL SIGTERM  
  
  
# Uitvoeren bij initialiseren container  
  
ENTRYPOINT ["/usr/sbin/named", "-g", "-u", "named", "-c", "/etc/named.conf"]  
  
  
#Healthcheck (wordt named uitgevoerd)  
HEALTHCHECK --interval=30s --timeout=5s --retries=3 CMD \  
pgrep named >/dev/null || exit 1
```

```

  GNU nano 5.6.1
FROM registry.access.redhat.com/ubi10/ubi-init
# Bind + tools installeren
RUN dnf -y install bind bind-utils procps-ng && dnf clean all

# Config & zone
COPY scripts/named.conf /etc/named.conf
COPY scripts/abc.pri    /var/named/abc.pri

# Vereiste directories + juiste rechten
RUN mkdir -p /var/named \
&& chown named:named /var/named \
&& chmod 0750 /var/named \
&& chown named:named /var/named/abc.pri /etc/named.conf \
&& chmod 0640 /var/named/abc.pri /etc/named.conf

# Uitvoeren bij initialiseren container
ENTRYPOINT ["/usr/sbin/named", "-g", "-u", "named", "-c", "/etc/named.conf"]

#Healthcheck (wordt named uitgevoerd)
HEALTHCHECK --interval=30s --timeout=5s --retries=3 CMD \
pgrep named >/dev/null || exit 1

```

Hulp Opslaan Zoeken Knippen Opdracht Positie Terugdraaien Markering Naar haakje Voorgaande
Afsluiten Inlezen Vervangen Plakken Uitvullen Naar regel Herdoen Kopiëren Terugzoeken Volgende

We kijken nu uiteraard enkel naar hetgeen er in de Dockerfile bijgekomen is.

- Bij de installatie is er procps-ng bijgekomen zodat pgrep uitgevoerd kan worden.
- Onderstaande vraagt uitleg:
pgrep named >/dev/null || exit 1

Als de returnvalue niet 0 is wordt hetgeen dat achter || staat uitgevoerd. In dit geval wordt dan een exit 1 doorgegeven aan HEALTHCHECK.

- Onderstaande vraagt uitleg:
HEALTHCHECK --interval=30s --timeout=5s --retries=3
--interval=30s → hoe vaak de check wordt uitgevoerd (default 30s)
--timeout=5s → maximale tijd dat de check mag duren
--retries=3 → aantal mislukte checks vóórdat container unhealthy wordt

Laat het image nu hermaken.

```
[student@serverXX oefening]$ podman build --format=docker --no-cache -t contdns
```

Verwijder nu de huidige container met DNS-server.

```
[student@serverXX oefening]$ podman rm contdns -f
```

We gaan nu een nieuwe container starten met de aangepaste versie van de DNS-server.

```
[student@serverXX oefening]$ podman run -d --name contdns --hostname contdns -p 1053:53/udp -p 1053:53/tcp contdns
```

We gaan kijken of de container actief en healthy is.

```
[student@serverXX oefening]$ podman ps
```

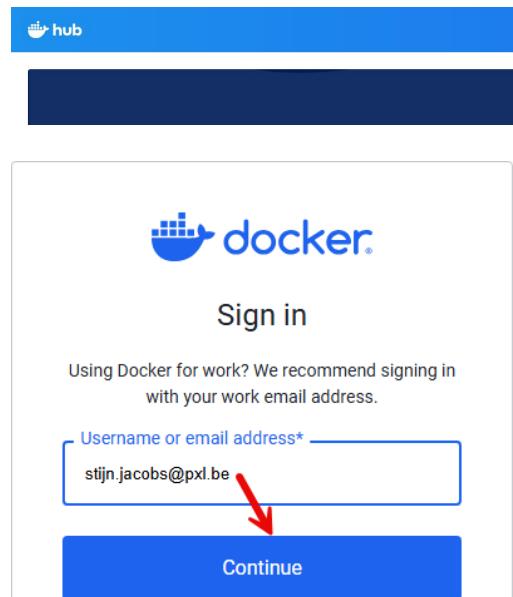
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
a4bbcf7f3a19f	localhost/contdns:latest		About a minute ago	Up About a minute (healthy)	0.0.0.0:1053->53/tcp, 0.0.0.0:1053->53/udp	contdns

Houd er rekening mee dat Docker, zonder een aanvullende orchestratie-oplossing, geen acties onderneemt op basis van de statuswaarden "gezond" of "ongezond". Containers worden niet automatisch gestopt of opnieuw opgestart.

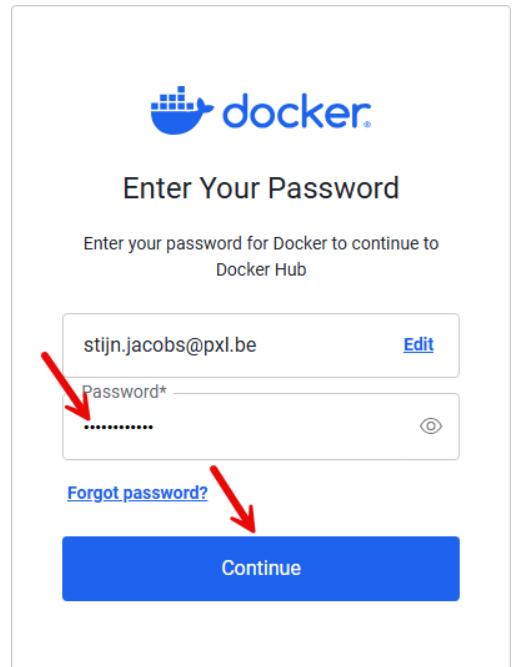
4.8 Images uploaden naar de Docker hub

4.8.1 Inloggen Docker hub

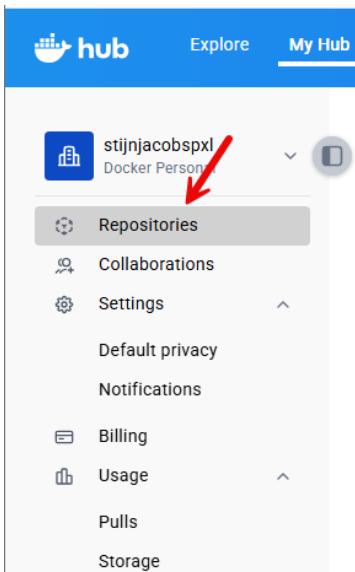
Ga naar <https://hub.docker.com> en nog je in met je docker-account.



The screenshot shows the Docker Hub sign-in page. At the top, there is a blue header bar with the Docker logo, a search bar, and navigation icons. Below the header, a large dark blue banner spans the width of the page. The main content area has a white background with the Docker logo at the top. The text "Sign in" is centered above a form field. Below the form field, a note says "Using Docker for work? We recommend signing in with your work email address." A red arrow points from the bottom of the "Username or email address*" input field to the "Continue" button at the bottom of the form. The input field contains the email "stijn.jacobs@pxl.be".



The screenshot shows the Docker Hub password entry page. The top part is identical to the sign-in page. Below it, the text "Enter Your Password" is centered. A note below it says "Enter your password for Docker to continue to Docker Hub". There are two input fields: one for the email "stijn.jacobs@pxl.be" and one for the password. A red arrow points from the bottom of the "Password*" input field to the "Forgot password?" link. Another red arrow points from the bottom of the "Forgot password?" link to the "Continue" button at the bottom of the form. The password field contains several dots.



Klik links op Repositories zo je je daar niet bevindt. Je komt nu op deze locatie terecht:

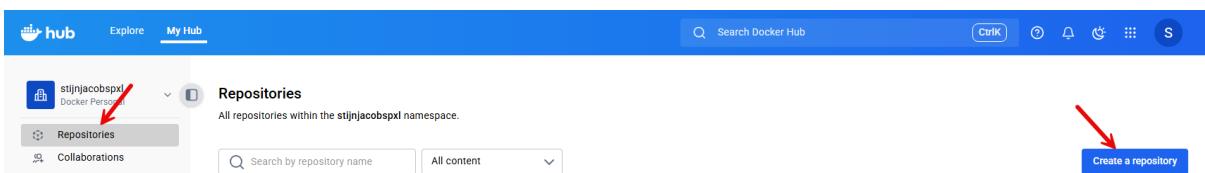
<https://hub.docker.com/repositories/<username>>.

4.8.2 Private Repository

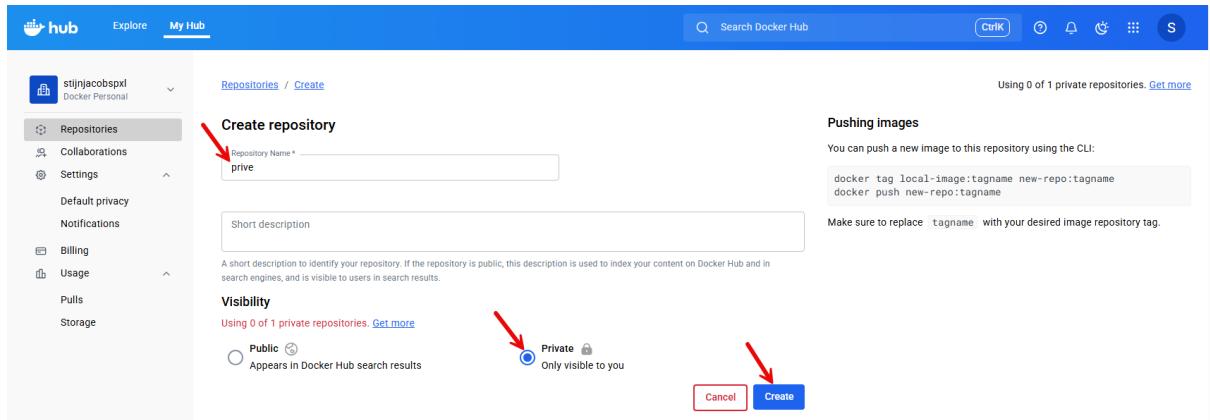
Een private repository in Docker Hub is een opslagplaats waar je container images kunt bewaren, maar die alleen toegankelijk is voor jou en de personen of teams aan wie je expliciet toegang verleent. Met een private repository kun je je container images beveiligen en ervoor zorgen dat alleen geautoriseerde gebruikers toegang hebben om ze te bekijken, te downloaden, of aan te passen. Dit is handig voor gevoelige projecten of bedrijfsmilieus waar je je software niet publiekelijk wilt delen.

We gaan nu een private repository aanmaken.

Klik hiervoor rechtsboven op Create repository.



Vul de gegevens voor jouw namespace in zoals hieronder staat aangegeven. Klik erna op Create.



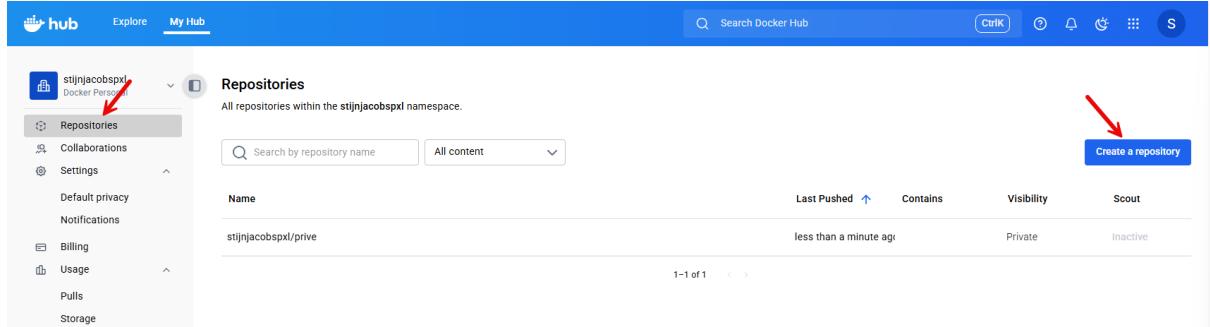
4.8.3 Public Repository

De tegenhanger van een private repository is een public repository. Dit is een opslagplaats die toegankelijk is voor iedereen. Iedereen kan de inhoud van een public repository bekijken en downloaden, zonder authenticatie.

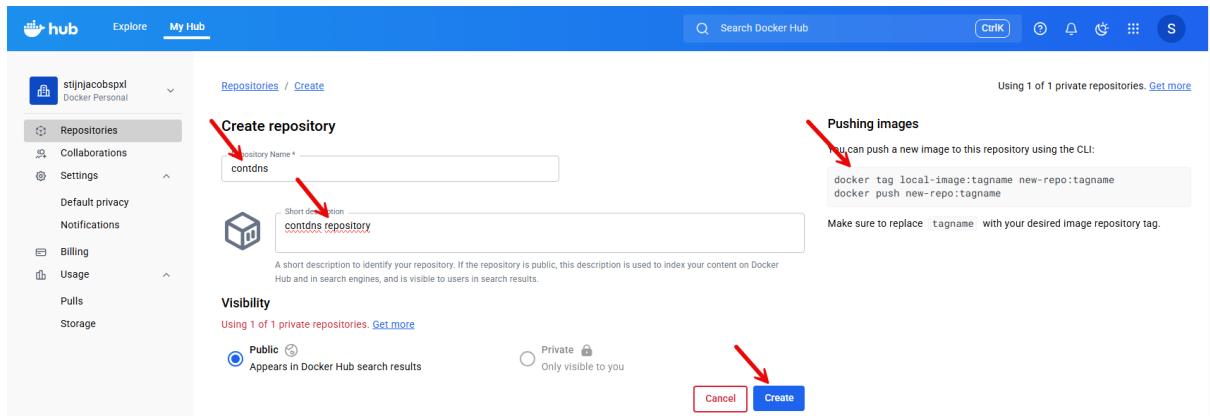
We gaan nu een public repository aanmaken.

Klik hiervoor eerst op **Repositories** aan de linkerkant indien nodig.

Klik nu weer rechtsboven op **Create repository**.



Vul de gegevens voor jouw namespace in zoals hieronder staat aangegeven. Klik erna op **Create**.



Je ziet rechts hoe je een nieuw image op de repository kan zetten.

PS Zoals je ziet kan je maar gebruik maken van 1 private repository met een gratis account.

4.8.4 Image pushen

Voor dit onderdeel moet je uiteraard een image hebben dat je wil uploaden.

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

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
localhost/contdns	latest	bc3854be343f	2 hours ago	263 MB

We moeten nu uiteraard via CLI inloggen bij podman. Aangezien we met podman werken is de procedure lichtjes anders dan bij docker.

```
student@serverXX:~$ podman login docker.io
```

Username: stijn.jacobs@pxl.be

Password:

Login Succeeded!

PS bij Docker zou je als volgt inloggen (dit doen we nu niet).

```
student@serverXX:~$ docker login -u <dockergebruikersnaam>.
```

Voer erna uiteraard je wachtwoord in.

Je moet nu je lokale image taggen met de naam van de repository op Docker Hub.

```
student@serverXX:~$ podman tag mijnimage:latest docker.io/<jouw-gebruikersnaam>/<repo-naam>:latest
```

In dit geval:

```
student@serverXX:~$ podman tag contdns:latest  
docker.io/stijnjacobspxl/contdns:latest
```

We kijken nu terug naar het images op ons systeem.

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

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
docker.io/stijnjacobspxl/contdns	latest	bc3854be343f	2 hours ago	263 MB
localhost/contdns	latest	bc3854be343f	2 hours ago	263 MB

We pushen het image nu naar de Docker Hub.

```
student@serverXX:~$ podman push docker.io/stijnjacobspxl/contdns:latest
```

```
[student@serverXX ~]$ podman push docker.io/stijnjacobspxl/contdns:latest
Getting image source signatures
Copying blob 697bef2cdb5c done
Copying blob 082c7390184d done
Copying blob 32825e41cccl done
Copying blob 7dc7bec5d446 done
Copying blob 039ea7f02cbe done
Copying blob 09bf1696a91e done
Copying config bc3854be34 done
Writing manifest to image destination
[student@serverXX ~]$
```

Wanneer je een container maakt die gebaseerd is op een bestaand image, kan Docker proberen om de basislagen van die image opnieuw te pushen naar Docker Hub. Dit gebeurt meestal om een paar redenen:

- Private Layers: Als de basisimage lagen bevatten die privé zijn, moet Docker deze lagen opnieuw pushen om te verifiëren dat je toegang hebt tot die content1.
- Image Updates: Als er updates zijn geweest aan de basisimage sinds je deze hebt gedownload, kan Docker proberen de nieuwste versie te pushen1.
- Layer Differences: Zelfs als de lagen hetzelfde lijken, kunnen er kleine verschillen zijn die Docker ertoe aanzetten om de lagen opnieuw te pushen1.
- Tagging Issues: Als je het image niet correct hebt getagd, kan Docker proberen de lagen opnieuw te pushen omdat het denkt dat ze nieuw zijn2.

Surf nu naar <https://hub.docker.com/repositories/<jouwloginnaam>>

Je ziet dat de repository nu een image bevat.

Name	Last Pushed	Contains	Visibility	Scout
stijnjacobspxl/contdns	33 minutes ago	IMAGE	Public	Inactive
stijnjacobspxl/prive	about 2 hours ago		Private	Inactive

4.8.5 Image pullen

We zullen voor de eenvoud alle images van ons systeem verwijderen om vervolgens het image van contdns te pullen.

```
student@serverXX:~$ podman image rm --all -f
Untagged: docker.io/stijnjacobspxl/contdns:latest
Untagged: localhost/contdns:latest
Untagged: registry.access.redhat.com/ubi10/ubi-init:latest
```

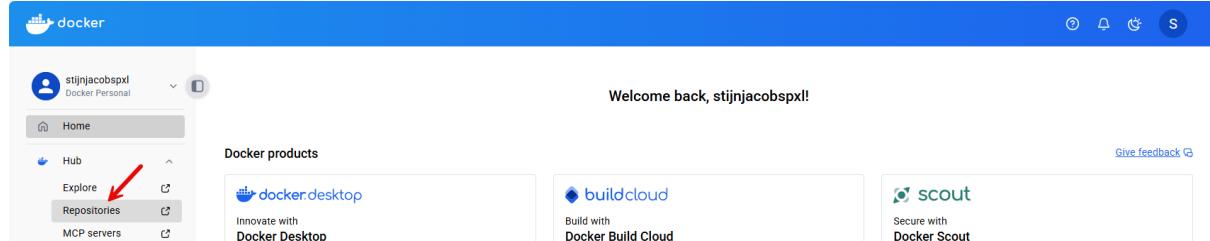
...

We pullen nu het image van contdns.

```
student@serverXX:~$ podman pull docker.io/stijnjacobspxl/contdns:latest
Trying to pull docker.io/stijnjacobspxl/contdns:latest...
Getting image source signatures
Copying blob c6edde6ffca0 done |
Copying blob 349ee6ea37a7 done |
Copying blob 1094cf3c2cfb done |
Copying blob e9f79a762181 done |
Copying blob ac7f61692b52 done |
Copying blob a93174cfe616 done |
Copying config bc3854be34 done |
Writing manifest to image destination
```

bc3854be343f67c433722fe824db4db5fd06310d65946aeaaf0cba9702730add

Als je surft naar <https://www.docker.com> en inlogt (Sign in) kan je links klikken op Hub, Repositories.



Containers beheren

4.9 Cockpit

4.9.1 Inleiding

Cockpit is een open-source, webgebaseerde tool voor serverbeheer in Linux die een grafische interface biedt om Linux-systeem te beheren en te monitoren, zowel lokaal als op afstand. Via een plugin kan je ook containers beheren.

De officiële website: <https://cockpit-project.org/documentation>.

4.9.2 Installatie

De installatie is erg eenvoudig...

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

Cockpit draait als een service. We starten de service onmiddellijk en na opstarten.

```
student@serverXX:~$ sudo systemctl enable --now cockpit.socket
```

```
Created symlink /etc/systemd/system/sockets.target.wants/cockpit.socket →
/usr/lib/systemd/system/cockpit.socket.
```

We stellen de firewall ook in voor cockpit.

```
student@serverXX:~$ sudo firewall-cmd --add-service=cockpit --permanent (--zone=...)
```

Geef eventueel de zone mee maar niet de zone van docker. Die wordt niet gebruikt voor inkomende verbindingen naar de host zelf (we komen daar later op terug).

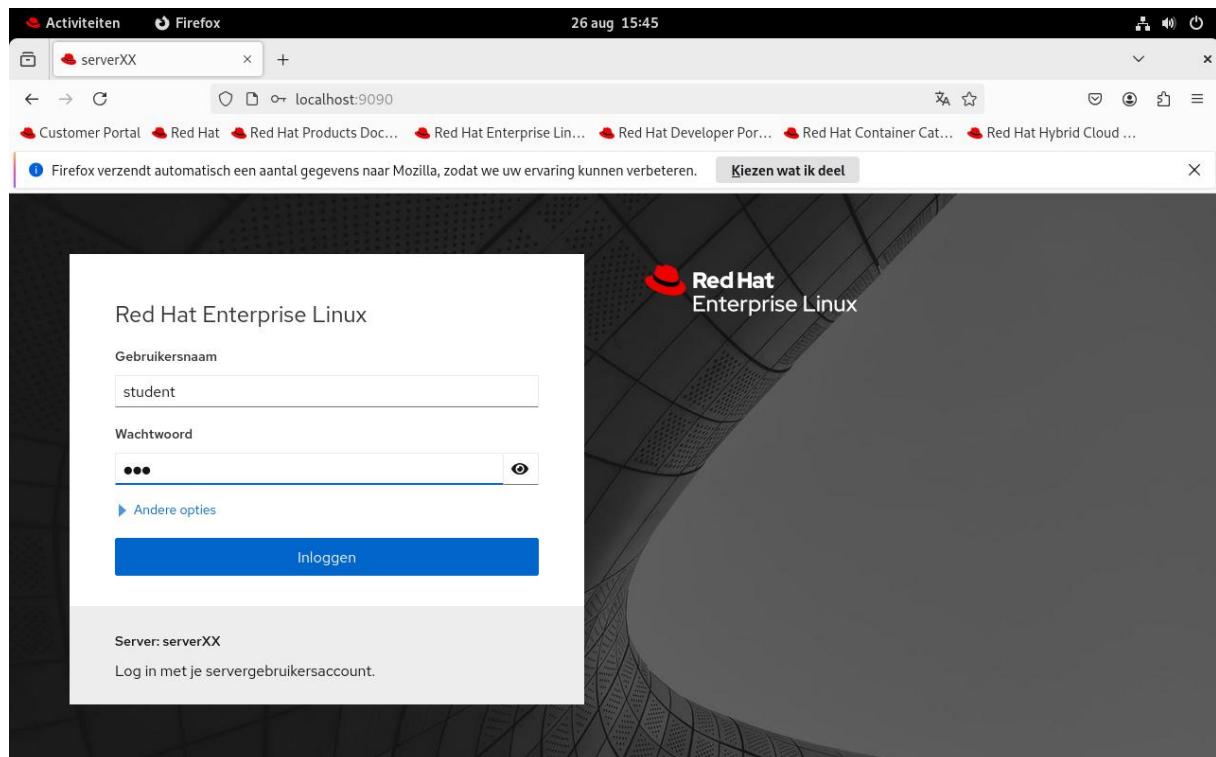
```
student@serverXX:~$ sudo firewall-cmd --reload
```

4.9.3 Gebruik

We gaan nu naar de browser en typen onderstaande in:

<http://localhost:9090>

Log in met de gegevens van student.

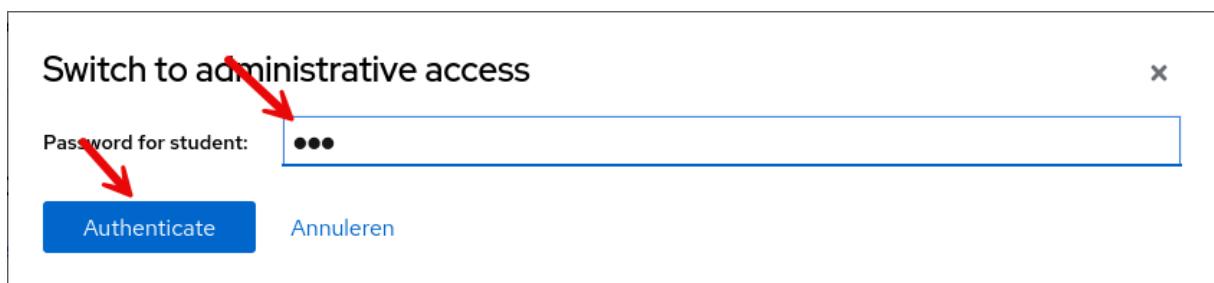


Je krijgt nu onderstaand venster.

The screenshot shows the Cockpit web interface running in Firefox. A banner at the top reads "De webconsole wordt uitgevoerd in de beperkte toegang modus." (The webconsole is running in limited access mode) and contains a blue button labeled "Turn on administrative access". A red arrow points to this button. The left sidebar shows navigation links like Overzicht, Logboeken, Opslag, Netwerken, Podman containers, Accounts, Services (with a red exclamation mark), and Gereedschappen. The main content area displays system status under "Gezondheid" and "Gebruik".

Klik op Turn on administrative access.

Nu wordt weer het wachtwoord van student gevraagd. Geef dat in en kies voor Authenticate.



Zoals je ziet is Cockpit een algemene GUI-tool om je server te beheren.

Op dit moment zijn we echter enkel geïnteresseerd in Podman containers. Je kan daar links op klikken.

The screenshot shows the Red Hat Customer Portal interface. On the left, there's a sidebar with various links: Systeem, Overzicht, Logboeken, Opslag, Netwerken, Podman containers (which is highlighted with a blue background), Accounts, Services (with a red notification dot), and Gereedschappen. The main content area has tabs for 'Images' and 'Containers'. Under 'Images', there's a table with one row: 'Image' (docker.io/stijnjacobspnx/contndslatest), Owner (student), Created (3 hours ago), ID (bc3854be343f), Disk s... (263 MB), and Use... (unused). A 'Create container' button is at the bottom right of this section. Under 'Containers', it says 'No containers'. At the top right of the main content area, there are buttons for 'Administratieve toegang', 'Hulp', and 'Sessie'.

Je ziet hier het image staan die op ons systeem staat.

We zullen een container aanmaken door op de blauwe knop Create container te klikken.

We geven onderstaande gegevens in.

The screenshot shows the 'Create container' dialog box. It has fields for 'Name' (Hello-world), 'Owner' (User: student), 'Image' (docker.io/library/hello-world), and 'Command'. There's a checked checkbox for 'With terminal'. At the bottom are 'Create and run', 'Create', and 'Cancel' buttons. The background shows the same Red Hat Customer Portal interface as the previous screenshot, with the 'Podman containers' sidebar selected.

Nadat je op Create and run hebt geklikt wordt de container in cockpit altijd detached uitgevoerd. Je ziet dat de container in de state Exited verkeerd. Dat wil zeggen dat de container al gestopt is. Je ziet ook dat het image bewaard is.

The screenshot shows the Red Hat Enterprise Linux cockpit interface in Firefox. The left sidebar has a dark theme with white text. The 'Podman containers' section is selected. The main area has a light background.

Images Section:

Image	Owner	Created	ID	Disk s...	Used by	Action
docker.io/library/hello-world:latest	user: student	3 weeks ago	1b44b5a3e06a	26.7 kB	1 container	Create container
docker.io/stijnjacobspxl/contdns:latest	user: student	4 hours ago	bc3854be343f	263 MB	unused	Create container

Containers Section:

Container	Owner	CPU	Memory	State	Action
Hello-world docker.io/library/hello-world:latest /hello	user: student			Exited	Logs

A red arrow points to the 'Exited' status of the 'Hello-world' container in the 'Containers' section.

Je kan het log van de container als volgt bekijken. Klik hiervoor op > voor de container en klik op Logs.

The screenshot shows the Red Hat Enterprise Linux desktop environment with the Podman containers interface. The left sidebar is open with 'Podman containers' selected. The main window shows a list of containers, with one named 'Hello-world' currently running. The 'Logs' tab is selected, displaying the output of the container's logs.

Als je op de 3 puntjes rechts van de container klikt kan je handelingen op de container uitvoeren.

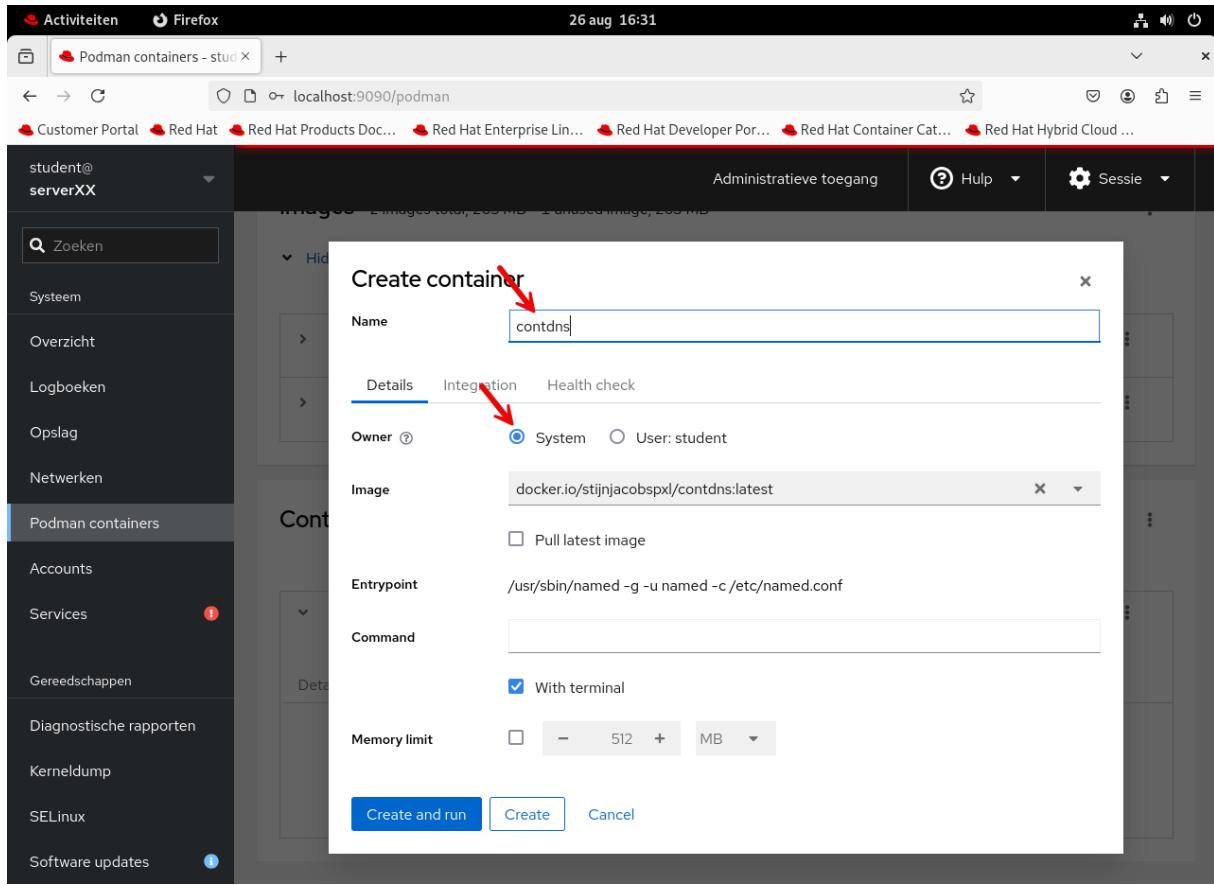
- Start: Probeert de container opnieuw te starten.
- Rename: Geeft de container een nieuwe naam. Handig als je overzicht wilt houden.
- Commit: Maakt een nieuw image van de huidige staat van de container. Dit is vergelijkbaar met een snapshot: je slaat alle wijzigingen in het containerbestandssysteem op als een nieuw image. Handig als je configuraties hebt gedaan binnen de container en die permanent als image wilt bewaren.
- Delete: Verwijdt de container (niet het image).

We zullen nu een DNS-server draaien aan de hand van docker.io/stijnjacobspxl/contdns:latest.

We doen het nu eens door te klikken op Create container naast het image docker.io/stijnjacobspxl/contdns:latest.

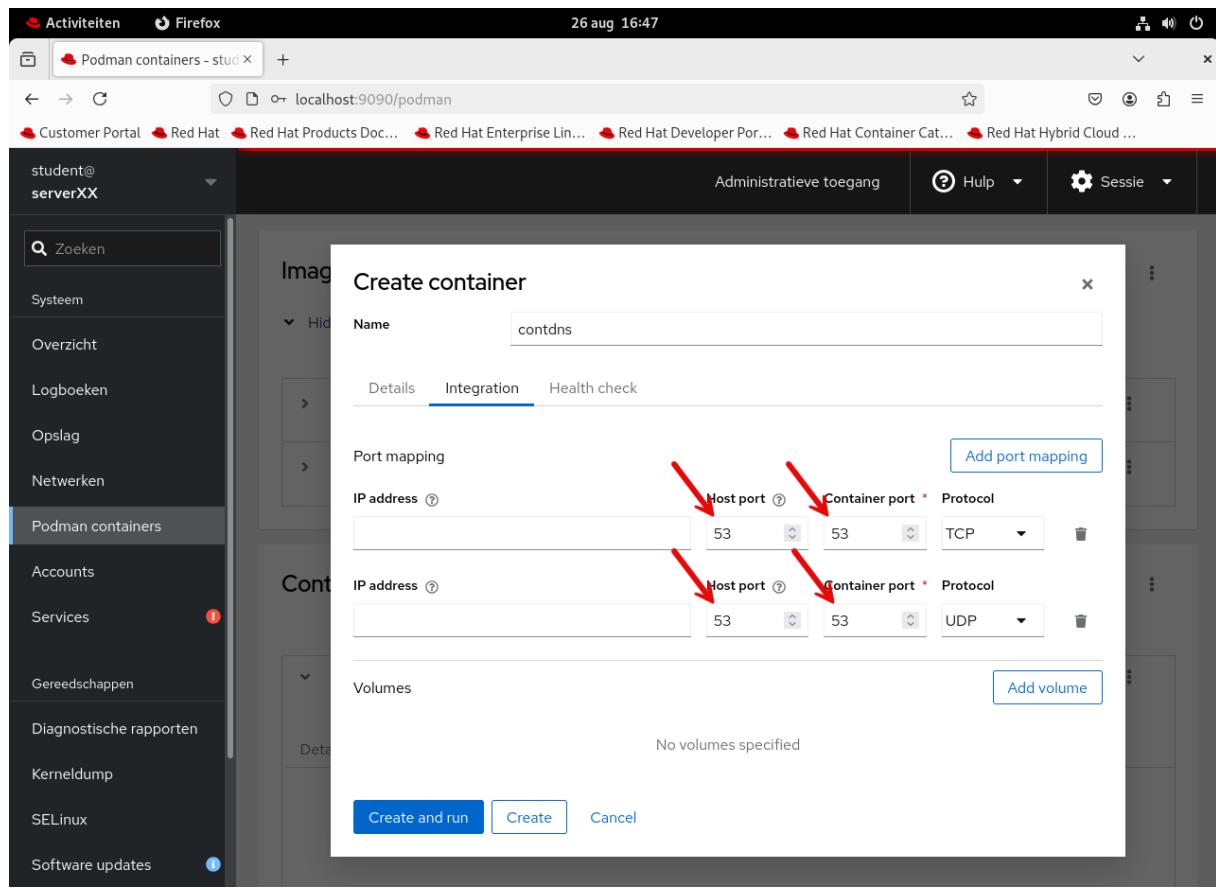
Images						
Image	Owner	Created	ID	Disk s...	Used by	
> docker.io/library/hello-world:latest	user: student	3 weeks ago	1b44b5a3e06a	26.7 kB	1 container	Create container :::
> docker.io/stijnjacobspxl/contdns:latest	user: student	4 hours ago	bc3854be343f	263 MB	unused	Create container :::

Geef nu onderstaande in op het tabblad Details.

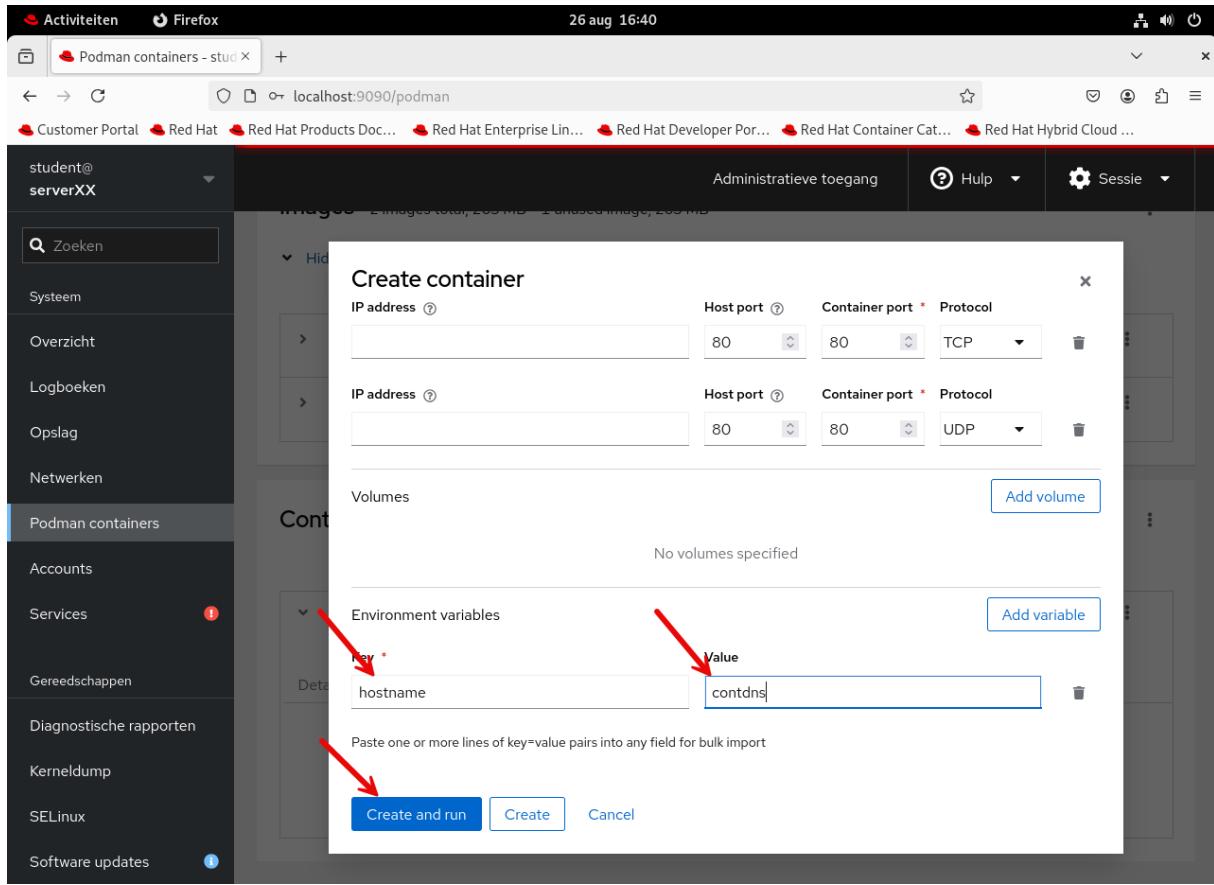


Door te kiezen voor system kunnen we gebruik maken van poorten onder 1024 😊.

Klik nu op integration en geef onderstaande in (klik 2 keer op port mapping).



Ga in het tabblad Integration naar beneden en klik op Add variable en vul onderstaande waarden in en klik op Create and run.



Je ziet nu volgende...

Containers		Show	All		Create pod	Create container	⋮
Container	↑	Owner	CPU	Memory	State		⋮
contdns		system	2.21%	0%	Running Healthy	4.03 MB	⋮

Met nslookup kan je nu een A-record opvragen.

```
student@serverXX:~$ nslookup host.abc.pri 127.0.0.1
Server:      127.0.0.1
Address:     127.0.0.1#53
```

Name: host.abc.pri

Address: 192.168.1.100

Je ziet ook dat het image docker.io/library/hello-world:latest in cockpit 2 keer aanwezig is... 1 keer voor student en één keer voor system.

Dat komt overeen met hetgeen wij hebben ingesteld via de CLI.

The screenshot shows the Red Hat Customer Portal interface for managing Podman containers. The left sidebar has a dark theme with white text and icons. The main content area is light gray.

Images

Image	Owner	Created	ID	Disk s...	Used by	Actions
docker.io/library/hello-world:latest	user: student	3 weeks ago	1b44b5a3e06a	26.7 kB	1 container	Create container More
docker.io/stijnjacobspxl/contdns:latest	user: student	4 hours ago	bc3854be343f	263 MB	unused	Create container More
docker.io/stijnjacobspxl/contdns:latest	system	4 hours ago	bc3854be343f	263 MB	1 container	Create container More

Containers

Container	Owner	CPU	Memory	State	Actions
contdns	system	0.00%	1% 9.73 MB	Running Healthy	More
Hello-world	user: student			Exited	More

At the bottom of the container table, there are tabs: Details, Integration, Logs, and Console.

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.

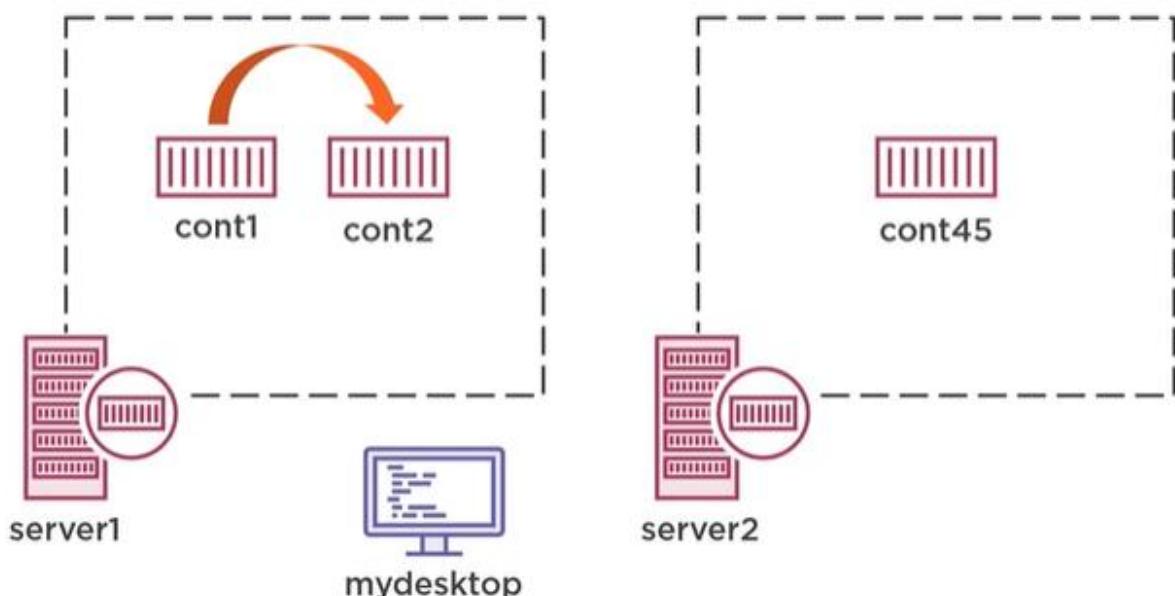
5.2 Container netwerken

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:

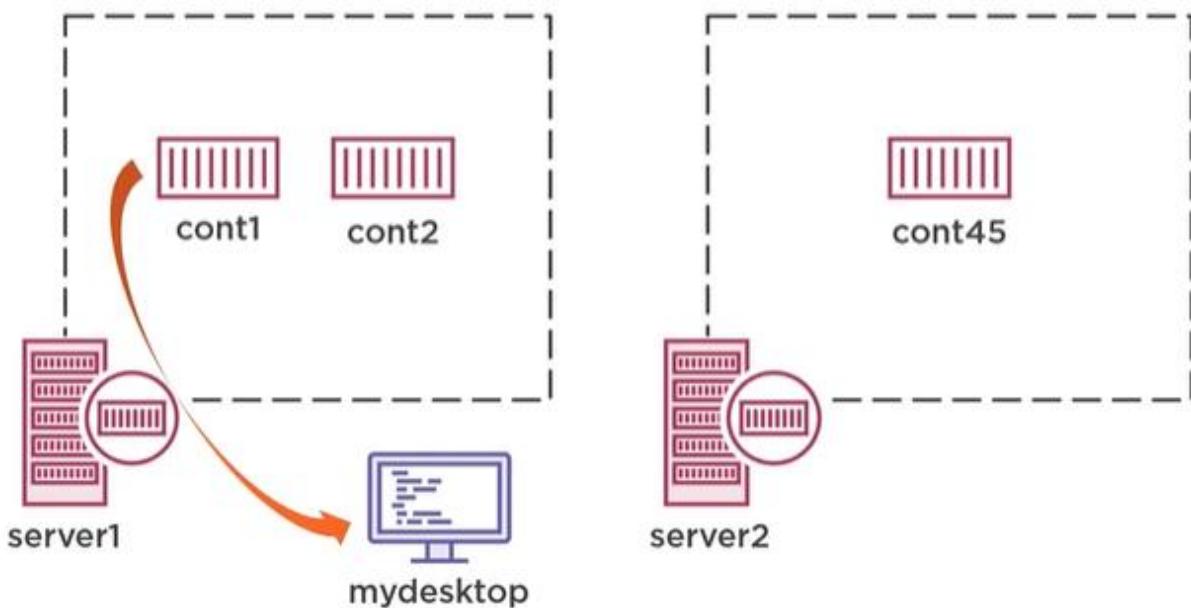
Lage latentie omdat beide containers op dezelfde fysieke host draaien.

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.

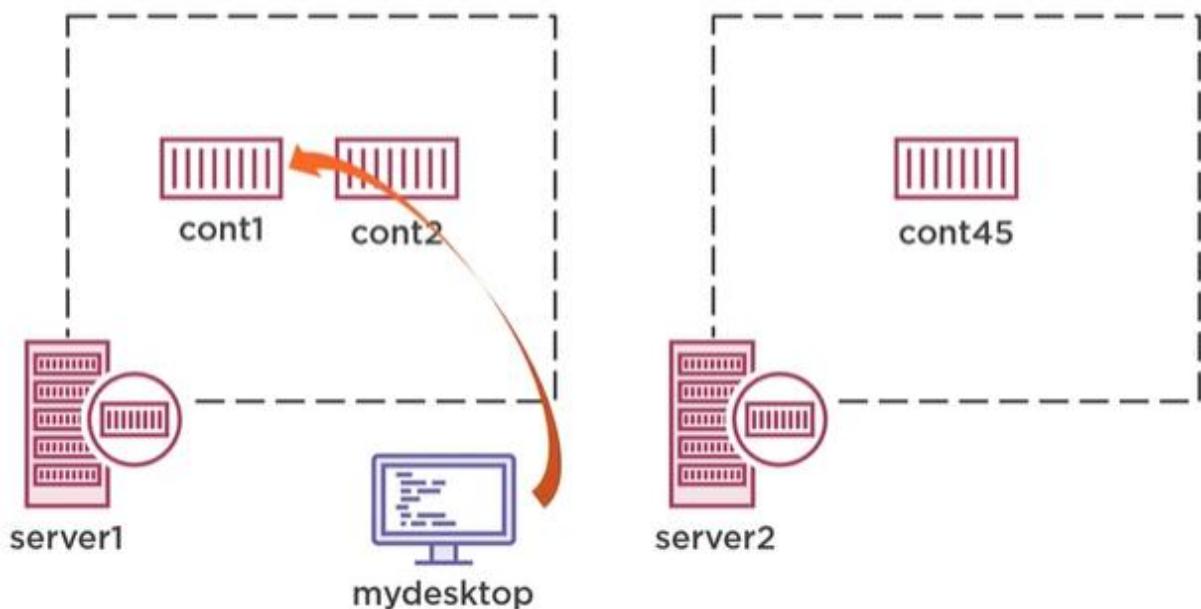
Voordelen:

- 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:

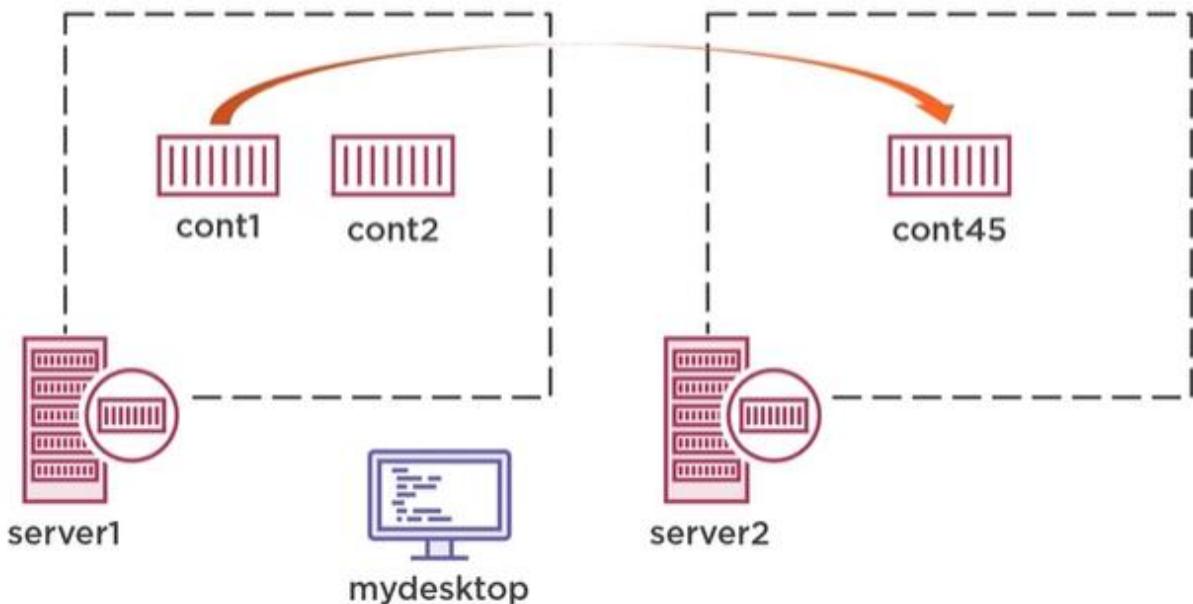
Externe systemen kunnen gemakkelijk toegang krijgen tot de services binnen containers.

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

- pasta: container gebruikt hetzelfde IP-adres als de host.
- slirp4netns: container krijgt een eigen virtueel IP (bijv. 10.0.2.100).

Gateway/NAT

- pasta: gebruikt de host-gateway, geen NAT.
- slirp4netns: maakt eigen gateway en doet NAT.

Interface

- pasta: neemt de host-interface over (bijv. ens160).
- slirp4netns: creëert een virtuele tap0 interface.

IPv6

- 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

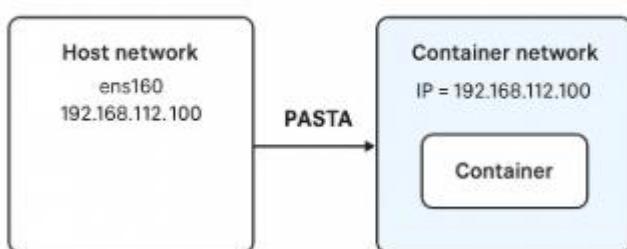
--- www.google.be ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.583/1.583/1.583/0.000 ms
```

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"
```

```
222f4f4b35a473a9bd3aaea6f2f6706008791e306584e3957d5730b4f8957cdb
```

-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.

- 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/xhtml1/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.containers.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:

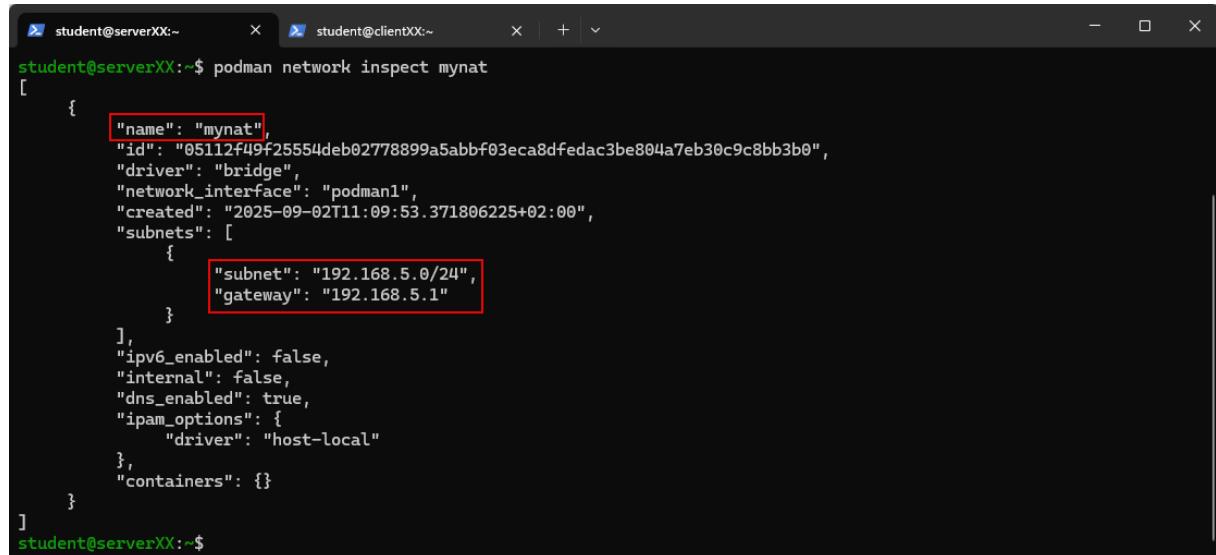
```
student@serverXX:~$ podman network ls
```



```
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
```



```
student@serverXX:~$ podman network inspect mynat
[
  {
    "name": "mynat",
    "id": "05112f49f2554deb02778899a5abbff03eca8dfedac3be804a7eb30c9c8bb3b0",
    "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
90a401c67a1d63221925aa893690db03e54ecca4a1313234cdcf920f7510f4dc
```

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 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_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
```



The screenshot shows a terminal window with two tabs. The active tab displays the command `ip route` and its output:

```
[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
  altnode enp3s0
  altnode 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 show 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 green 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.

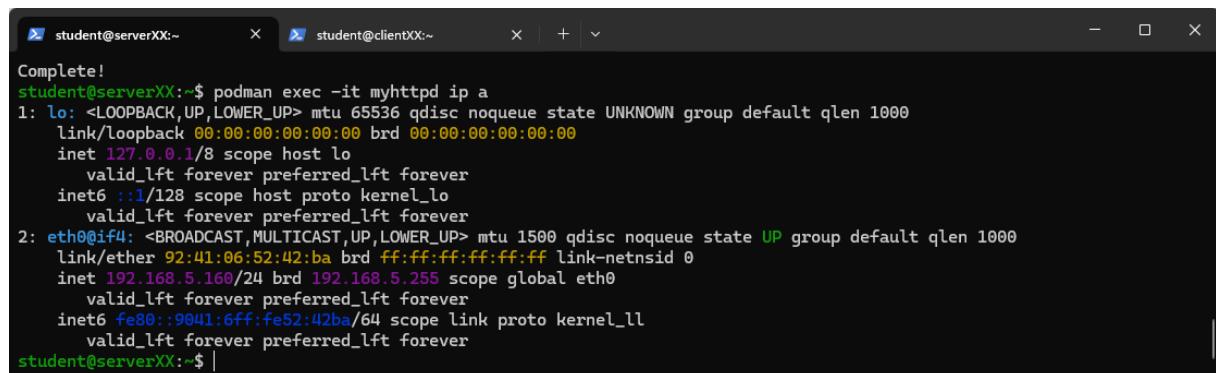
--ip 192.168.5.160: de container krijgt een specifiek IP-nummer

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
```



A screenshot of a terminal window titled 'student@serverXX:~'. It shows two tabs: 'student@serverXX:~' and 'student@clientXX:~'. The main pane displays the output of the command 'podman exec -it myhttpd ip a'. The output shows two network interfaces: 'lo' (loopback) and 'eth0@if4'. The 'lo' interface has an IP of 127.0.0.1/8. The 'eth0@if4' interface has an IP of 192.168.5.160/24. Both interfaces have MTU values of 1500 and queueing discipline (qdisc) of 'noqueue'.

```
student@serverXX:~$ podman exec -it myhttpd 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@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">
<head>
```

...

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
...
[root@C10 ~]#
```

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": [{"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
```

```

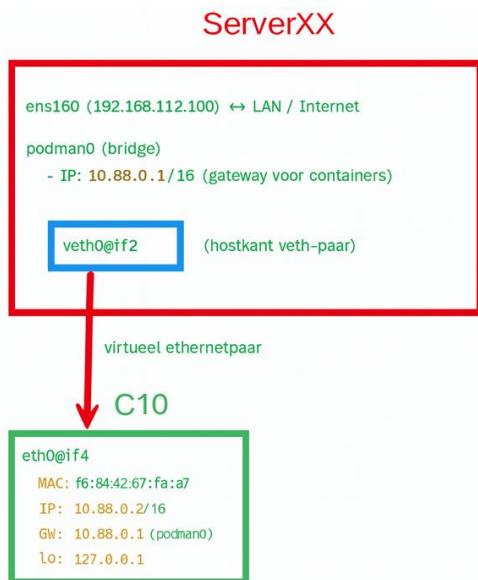
student@serverXX:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
        inet6 ::1/128 scope host noprefixroute
            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:fe6:238b/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.



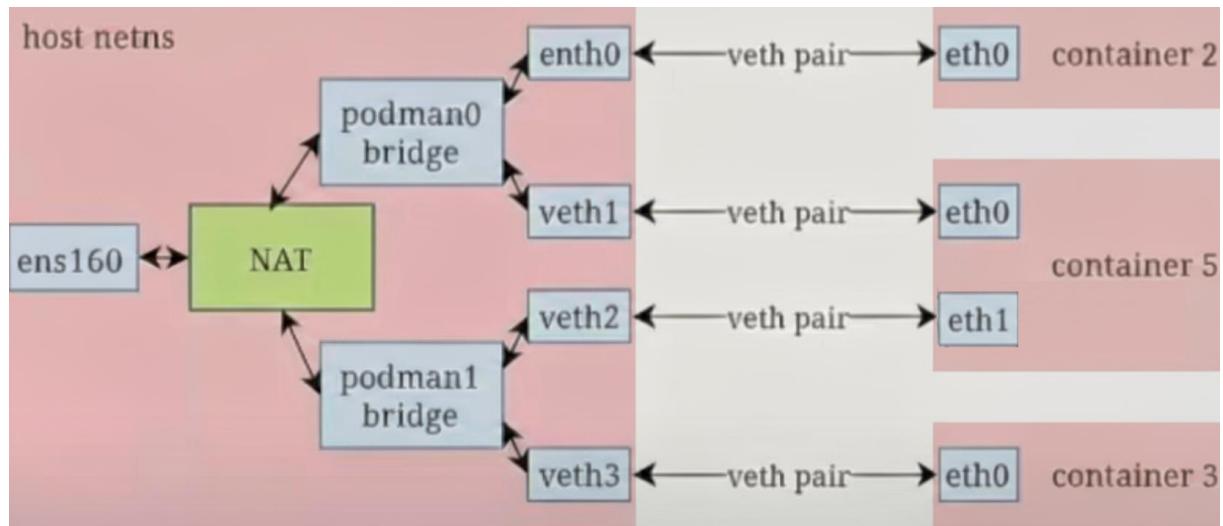
Hoe het werkt...

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/xhtml1/DTD/xhtml1.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
...

```

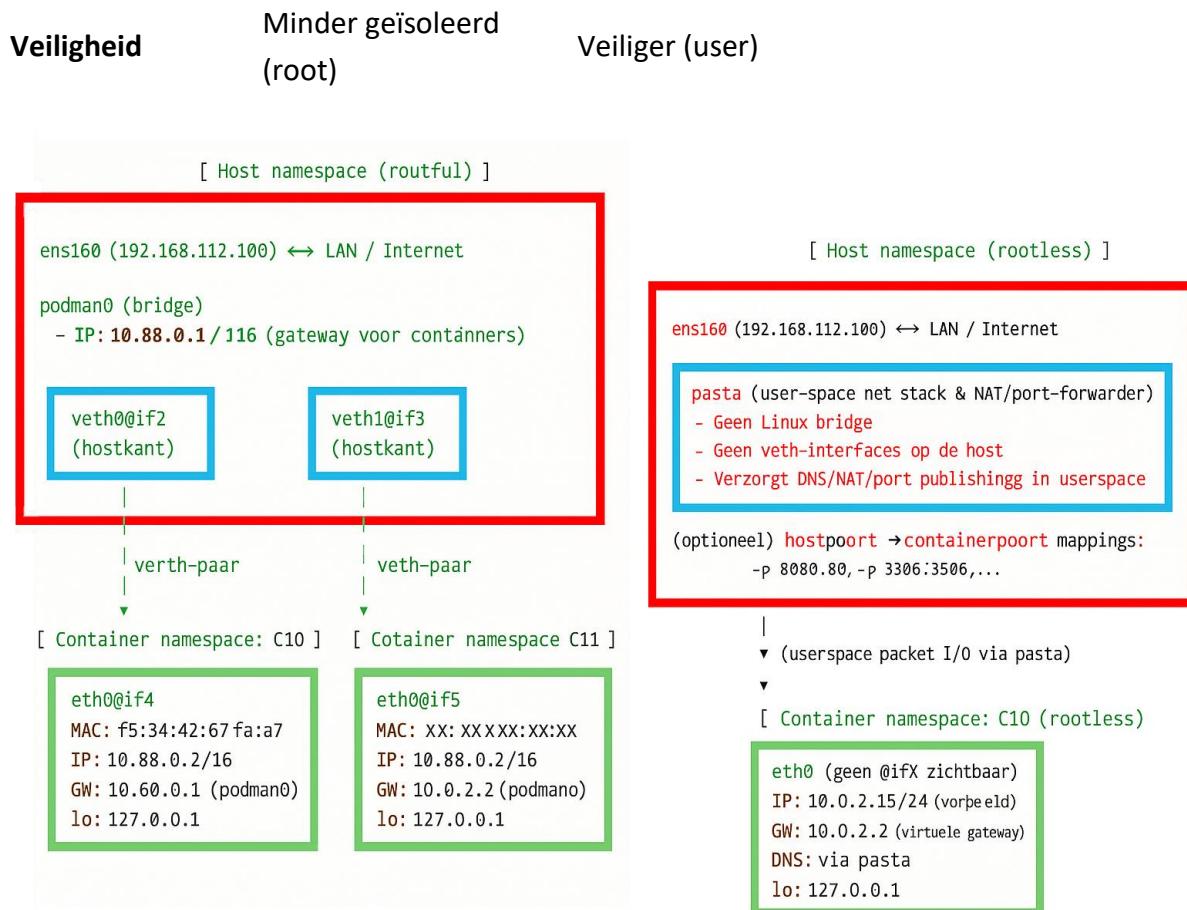
Je kan nu deze website beschikbaar maken via het netwerk.

5.5 User-space netwerken vs bridge netwerken

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

Eigenschap	Rootful (<code>sudo podman</code>)	Rootless (<code>podman</code>)
Netwerkdriver	bridge (podman0)	slirp4netns / pasta
IP-adressen	Echte kernel-bridge IP's	Virtueel (userspace)
Poorten <1024	Mogelijk	Niet mogelijk

Eigenschap	Rootful (sudo podman)	Rootless (podman)
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
```

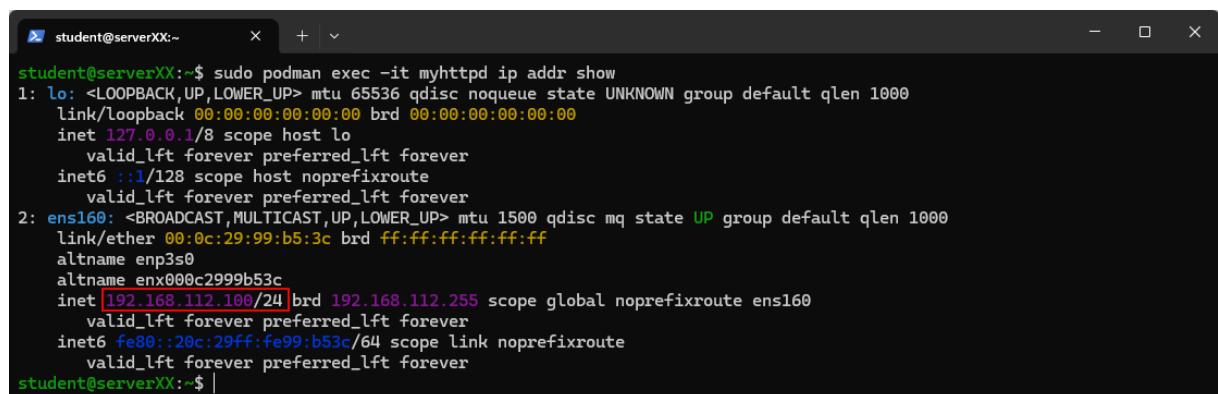
Start een webserver met het host-netwerk en test de toegang.

```
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>
  ...
</html>
```

--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_netwerk
```

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_netwerk	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_netwerk --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_ll
            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@contiis3 /]# cat /etc/resolv.conf
nameserver 169.254.1.1
nameserver 192.168.112.2
[root@contiis3 /]#
```

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

```
[root@contiis3 /]# dnf install -y iputils
```

```
[root@contiis3 /]# ping -c 1 www.google.be
```

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

We installeren in de container een webserver.

```
[root@contiis3 /]# 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">

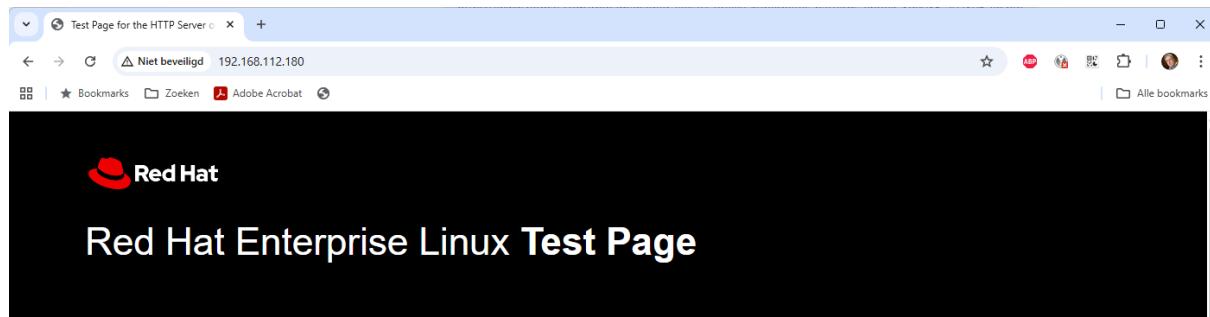
...

```

[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.



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, you should send e-mail to ["webmaster@example.com"](mailto: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`.



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">
...
</html>
```

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 contiiis3.

```
student@serverXX:~$ sudo podman exec -it contiiis3 /bin/sh -c "ip link show"
```

```
student@serverXX:~$ sudo podman exec -it containerX /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:
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:f4:4e:34:b5 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 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 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::c299: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@contiiis4 /]# cat /etc/resolv.conf
```

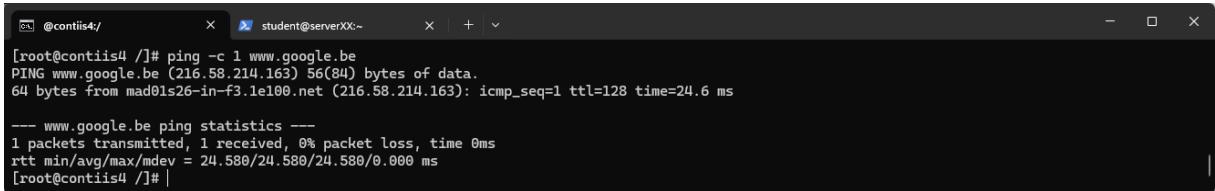


```
[root@contiiis4 /]# cat /etc/resolv.conf
nameserver 192.168.112.2
[root@contiiis4 /]# |
```

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

```
[root@contiiis4 /]# dnf install -y iputils
```

```
[root@contiiis4 /]# ping -c 1 www.google.be
```



```
[root@contiiis4 /]# 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@contiiis4 /]# |
```

We installeren in de container een webserver.

```
[root@contiiis4 /]# dnf install -y httpd
```

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

```
[root@contiiis4 /]# 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
```

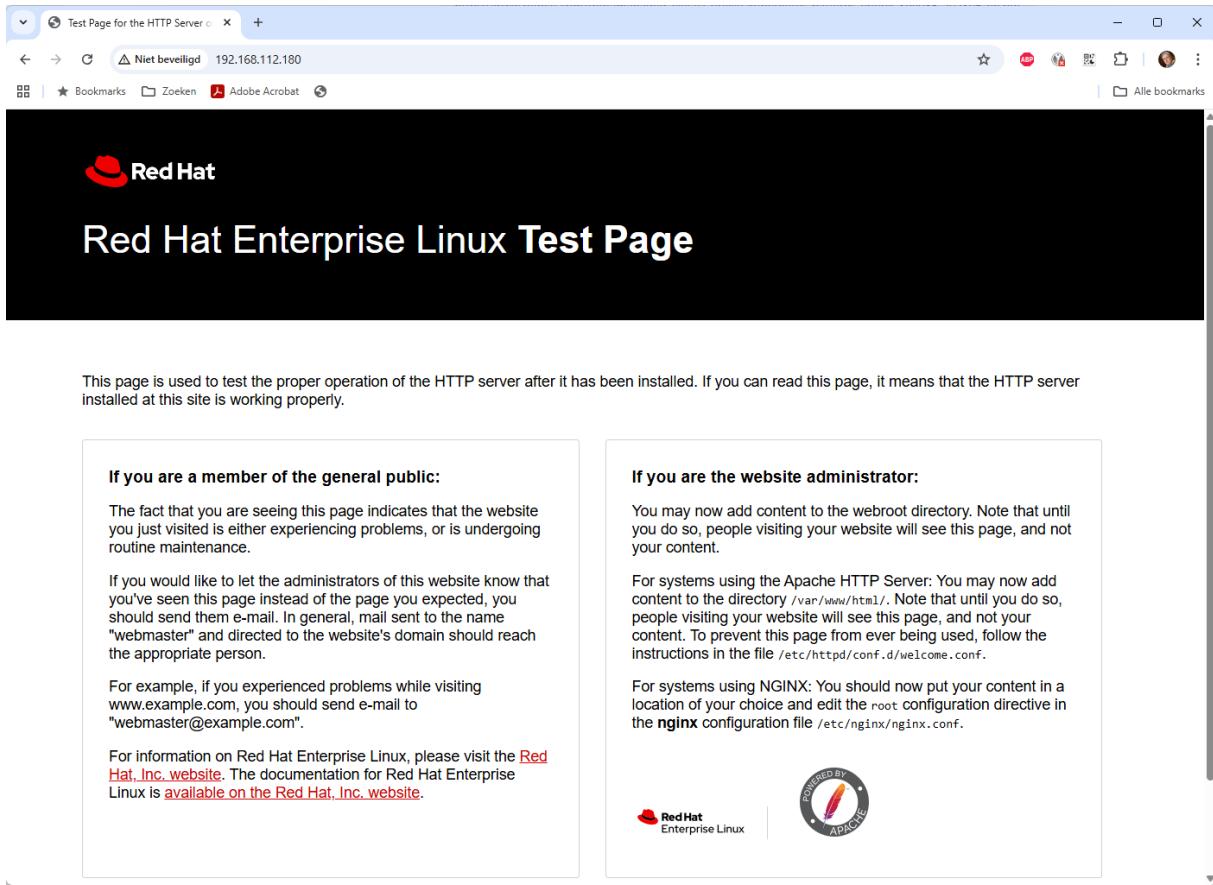
```
[root@contiiis4 /]# curl 192.168.112.180
```

```
<!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@contiiis4 /]# exit
```

```
exit
```

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



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>
        <h1>Test Page for the HTTP Server on Red Hat Enterprise Linux</h1>
        <p>This page is generated by an Apache server running on Red Hat Enterprise Linux. It is a test page for the HTTP server, showing basic styling and layout.</p>
        <ul>
            <li>Apache Version: 2.4.18</li>
            <li>Server OS: Red Hat Enterprise Linux 7.6</li>
            <li>Processor: Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz</li>
            <li>Memory: 16.0 GiB</li>
            <li>Disk: 1.7 TiB</li>
            <li>Network: eth0</li>
            <li>Kernel: 3.10.0-1057.el7.x86_64</li>
        </ul>
    </body>
</html>

```

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
Toegang lage poorten (<1024)	Ja	Ja	Ja	Nee (tenzij workaround)

Eigenschap	macvlan L2	ipvlan L2	bridge rootful	pasta rootless
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

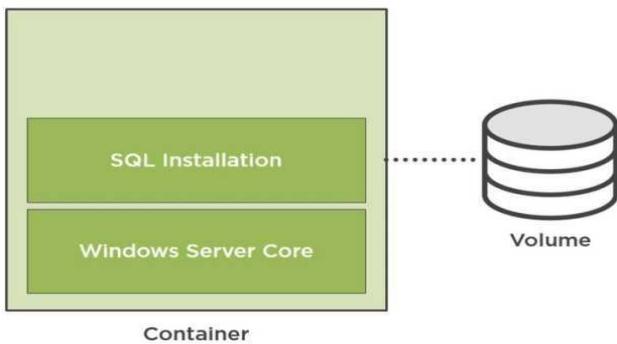
Een stateless applicatie kan worden gecreëerd en verwijderd zonder dat er zorgen zijn over het behouden van gegevens. Je kunt de instantie eenvoudig opnieuw maken zonder dat dit gevolgen heeft voor de werking of configuratie.

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.

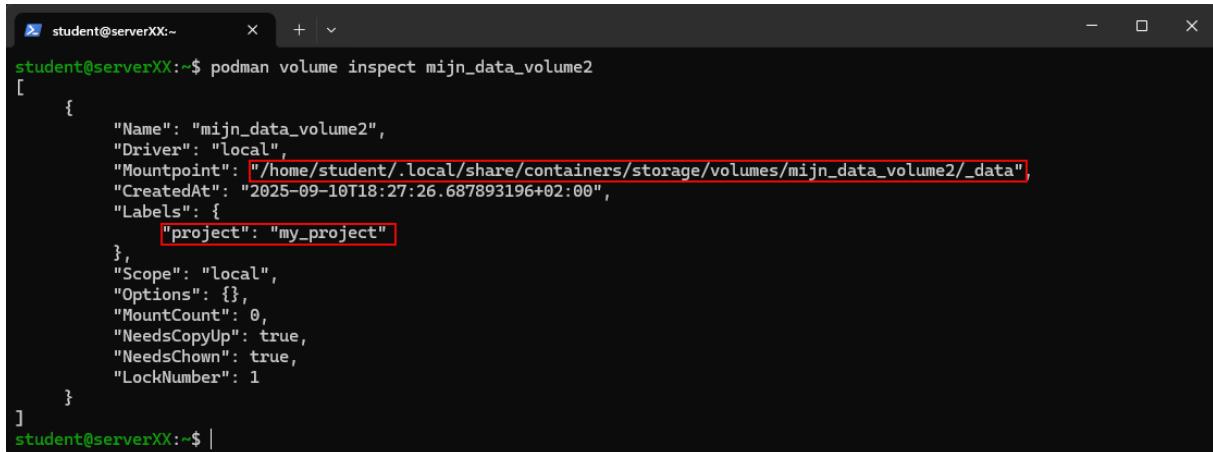
```
student@serverXX:~$ podman volume create mijn_data_volume2 --label  
"project=my_project"
```

```
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.

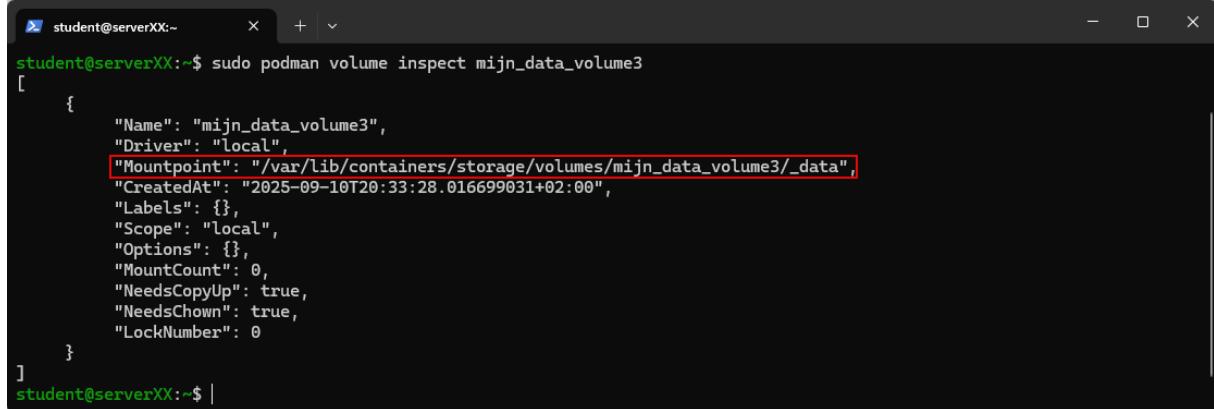
```
student@serverXX:~$ podman rm conta -f  
contb
```

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.

6 Podman images

6.1 Inleiding

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

6.2 Webserver

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

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

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

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

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

We zoeken eens naar een nginx-container.

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

NAME                                     DESCRIPTION
...
...
```

Je ziet zeer veel images.

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

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

registry.access.redhat.com/ubi10/nginx-126      ...
registry.redhat.io/ubi10/nginx-126             ...
```

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

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

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

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

The screenshot shows the Red Hat Ecosystem Catalog interface. At the top, there is a navigation bar with links for Solutions, Products, Artifacts, Partners, Contact us, Resources, All Red Hat, and Log In. Below the navigation bar, a search bar contains the text "Containers" and a magnifying glass icon. A red arrow points from the text "Kies nu voor containers bovenaan en typen in het zoekvenster nginx-126 in." to the search bar. To the right of the search bar is a search icon. Further down the page, a section titled "Container results for 'nginx-126'" displays eight search results. Each result card includes a small red hat icon, the name "Nginx 1.26", the provider name (e.g., rhel10/nginx-126), a brief description, and a timestamp indicating when it was published. A red arrow points from the text "Klik op Nginx 1.26 bij ubi10 zoals hierboven staat aangegeven." to the second result card, which has "ubi10/nginx-126" listed under the provider name.

Provider	Description	Published
rhel10/nginx-126	Platform for running nginx 1.26 or building nginx-based application	16 uur geleden
ubi10/nginx-126	Platform for running nginx 1.26 or building nginx-based application	gisteren
ubi10/nginx-126	Platform for running nginx 1.26 or building nginx-based application	16 uur geleden
rhel10/nginx-126	Platform for running nginx 1.26 or building nginx-based application	16 uur geleden
ubi9/nginx-126	Platform for running nginx 1.26 or building nginx-based application	gisteren
rhel9/nginx-126	Platform for running nginx 1.24 or building nginx-based application	gisteren

Klik op Nginx 1.26 bij ubi10 zoals hierboven staat aangegeven.

We krijgen nu onderstaande pagina.

The screenshot shows the Red Hat Ecosystem Catalog interface. At the top, there's a navigation bar with links for Home, Artifacts, Solutions, Products, Artifacts, Partners, Containers, a search bar (searching for 'nginx-126'), and user account options (Contact us, Resources, All Red Hat, Log In). Below the navigation is a breadcrumb trail: Home > Artifacts > All container results.

The main content area displays the details for the 'Nginx 1.26' container image. It includes the image name 'nginx-126', the provider 'Red Hat' (represented by a red hat icon), and a 'Provided by' section. Below this, there are dropdown menus for architecture ('amd64') and version ('10.0').

The central part of the page shows the image's version history: '10.0-1760573692' (latest) and '10.0'. Below this is a navigation bar with tabs: Overview (which is selected), Security, Technical information, Packages, Containerfile, and Get this image.

Description: Nginx is a web server and a reverse proxy server for HTTP, SMTP, POP3 and IMAP protocols, with a strong focus on high concurrency, performance and low memory usage. The container image provides a containerized packaging of the nginx 1.26 daemon. The image can be used as a base image for other applications based on nginx 1.26 web server. Nginx server image can be extended using OpenShift's Source build feature.

Usage in OpenShift: In this example, we assume that you are using the ubi9/nginx-126 image, available through the nginx:1.26 imagestream tag in OpenShift. To build a simple `test-app` application in OpenShift:

```
oc new-app nginx:1.26--https://github.com/scrlorg/nginx-container.git --context-dir=1.26/test/test-app/
To access the application:
$ oc get pods
$ oc exec <pod> -- curl 127.0.0.1:8080
```

Source-to-Image framework and scripts: This image supports the Source-to-Image (S2I) strategy in OpenShift. The Source-to-Image is an OpenShift framework which makes it easy to write images that take application source code as an input, use a builder image like this Nginx container image, and produce a new image that runs the assembled application as an output.

In case of Nginx container image, the application source code is typically either static HTML pages or configuration files. To support the Source-to-Image framework, important scripts are included in the builder image:

The `/usr/libexec/s2i/run` script is set as the default command in the resulting container image (the new image with the application artifacts). The `/usr/libexec/s2i/assemble` script inside the image is run to produce a new image with the application artifacts. The script takes sources of a given application (HTML pages), Nginx configuration files, and places them into appropriate directories inside the image. The structure of nginx-app can look like this:

- ./nginx.conf-- The main nginx configuration file
- ./nginx-cfg/*.conf Should contain all nginx configuration we want to include into image
- ./nginx-default-cfg/*.conf Contains any nginx config snippets to include in the default server block

Bij nieuwe packages is de info vaak niet beschikbaar.

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

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

Erna inspecteren we `ubi10/nginx-126`.

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

Je krijgt nu nogal onoverzichtelijke output.

```
student@serverXX:~$ podman inspect registry.access.redhat.com/ubi10/nginx-126
[
  {
    "Id": "4f9e549bdfc2f4560f6571a16db5c4ae2e908952351b9c5f4182708f09ebe165",
    "Digest": "sha256:230140087c189b3b9dd25590469da5b4b63735c4849c46d2eacb9eddb57ee939",
    "RepoTags": [
      "registry.access.redhat.com/ubi10/nginx-126:latest"
    ],
    "RepoDigests": [
      "registry.access.redhat.com/ubi10/nginx-126@sha256:230140087c189b3b9dd25590469da5b4b63735c4849c46d2eacb9eddb57ee939",
      "registry.access.redhat.com/ubi10/nginx-126@sha256:87d5fc56619129e86b27d81b86803381dbc0575657
15c6cc4ba9b72c0cf2e1df"
    ],
    "Parent": "",
    "Comment": "",
    "Created": "2025-09-03T15:27:01.757060557Z",
    "Config": {
      "User": "1001",
      "ExposedPorts": {
        "8080/tcp": {},
        "8443/tcp": {}
      },
      "Env": [
        "container=oci",
        "STI_SCRIPTS_URL=image:///usr/libexec/s2i",
        "STI_SCRIPTS_PATH=/usr/libexec/s2i",
        "APP_ROOT=/opt/app-root",
        "HOME=/opt/app-root/src",
        "PATH=/opt/app-root/src/bin:/opt/app-root/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/
usr/bin:/sbin:/bin",
        "PLATFORM=el10",
        "NAME=nginx",
        "NGINX_VERSION=1.26",
      ]
    }
  }
]
```

Dit komt omdat het JSON-formaat is.

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

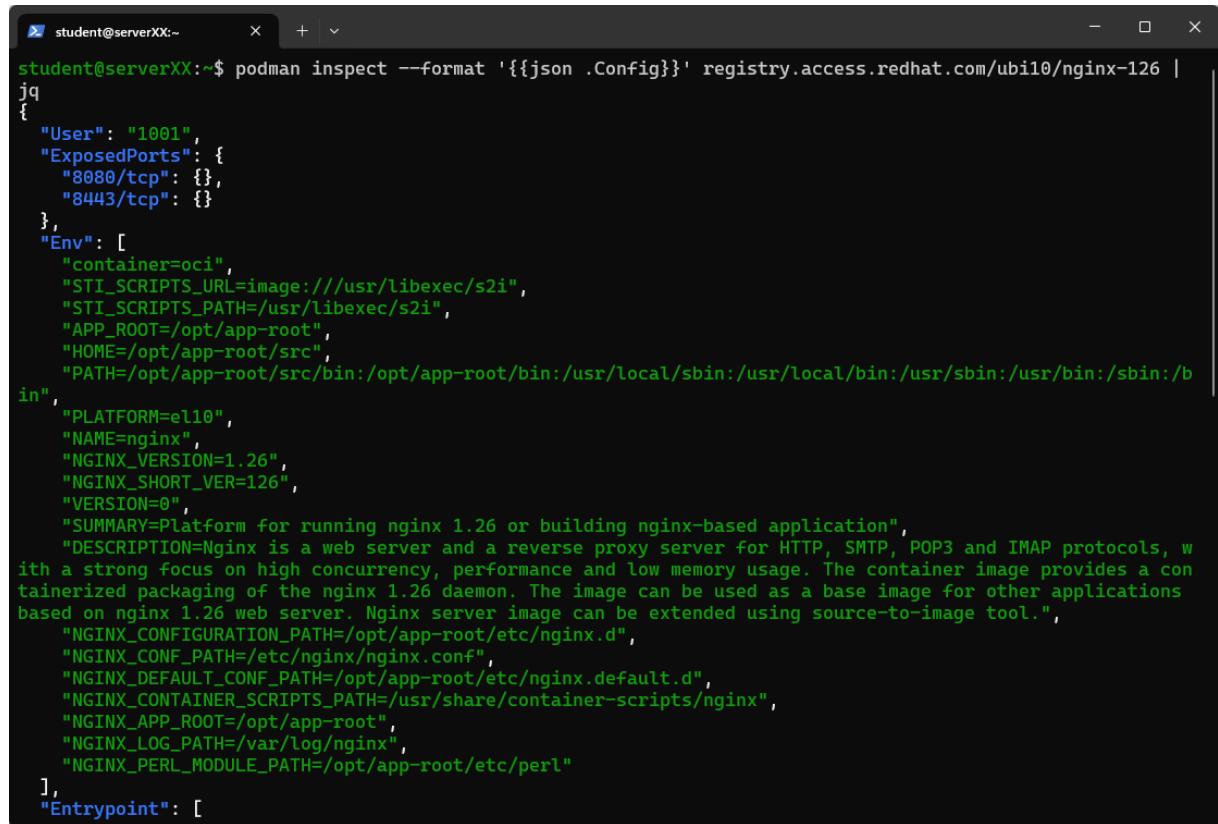
```
student@serverXX:~$ podman inspect registry.access.redhat.com/ubi10/nginx-126 | jq
[
  {
    "Id": "4f9e549bdfc2f4560f6571a16db5c4ae2e908952351b9c5f4182708f09ebe165",
    "Digest": "sha256:230140087c189b3b9dd25590469da5b4b63735c4849c46d2eacb9eddb57ee939",
    "RepoTags": [
      "registry.access.redhat.com/ubi10/nginx-126:latest"
    ],
    "RepoDigests": [
      "registry.access.redhat.com/ubi10/nginx-126@sha256:230140087c189b3b9dd25590469da5b4b63735c4849c46d2eac
b9eddb57ee939",
      "registry.access.redhat.com/ubi10/nginx-126@sha256:87d5fc56619129e86b27d81b86803381dbc057565715c6cc4ba
9b72c0cf2e1df"
    ],
    "Parent": "",
    "Comment": "",
    "Created": "2025-09-03T15:27:01.757060557Z",
    "Config": {
      "User": "1001",
      "ExposedPorts": {
        "8080/tcp": {},
        "8443/tcp": {}
      },
      "Env": [
        "container=oci",
        "STI_SCRIPTS_URL=image:///usr/libexec/s2i",
        "STI_SCRIPTS_PATH=/usr/libexec/s2i",
        "APP_ROOT=/opt/app-root",
        "HOME=/opt/app-root/src",
        "PATH=/opt/app-root/src/bin:/opt/app-root/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/
usr/bin",
        "PLATFORM=el10",
        "NAME=nginx",
        "NGINX_VERSION=1.26",
      ]
    }
  }
]
```

Podman heeft ook een mogelijkheid om het verder te filteren.

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

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

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



```
student@serverXX:~$ podman inspect --format '{{json .Config}}' registry.access.redhat.com/ubi10/nginx-126 | jq  
{  
  "User": "1001",  
  "ExposedPorts": {  
    "8080/tcp": {},  
    "8443/tcp": {}  
  },  
  "Env": [  
    "container=oci",  
    "STI_SCRIPTS_URL=image:///usr/libexec/s2i",  
    "STI_SCRIPTS_PATH=/usr/libexec/s2i",  
    "APP_ROOT=/opt/app-root",  
    "HOME=/opt/app-root/src",  
    "PATH=/opt/app-root/src/bin:/opt/app-root/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",  
    "PLATFORM=el10",  
    "NAME=nginx",  
    "NGINX_VERSION=1.26",  
    "NGINX_SHORT_VER=126",  
    "VERSION=0",  
    "SUMMARY=Platform for running nginx 1.26 or building nginx-based application",  
    "DESCRIPTION=Nginx is a web server and a reverse proxy server for HTTP, SMTP, POP3 and IMAP protocols, with a strong focus on high concurrency, performance and low memory usage. The container image provides a containerized packaging of the nginx 1.26 daemon. The image can be used as a base image for other applications based on nginx 1.26 web server. Nginx server image can be extended using source-to-image tool.",  
    "NGINX_CONFIGURATION_PATH=/opt/app-root/etc/nginx.d",  
    "NGINX_CONF_PATH=/etc/nginx/nginx.conf",  
    "NGINX_DEFAULT_CONF_PATH=/opt/app-root/etc/nginx.default.d",  
    "NGINX_CONTAINER_SCRIPTS_PATH=/usr/share/container-scripts/nginx",  
    "NGINX_APP_ROOT=/opt/app-root",  
    "NGINX_LOG_PATH=/var/log/nginx",  
    "NGINX_PERL_MODULE_PATH=/opt/app-root/etc/perl"  
  ],  
  "Entrypoint": [
```

Je kan verder verfijnen naar .Config.Labels.

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

```

student@serverXX:~$ podman inspect --format '{{json .Config.Labels}}' registry.access.redhat.com/ubi10/nginx-126 | jq
{
  "architecture": "x86_64",
  "build-date": "2025-09-03T15:26:56",
  "com.redhat.component": "nginx-126-container",
  "com.redhat.license_terms": "https://www.redhat.com/en/about/red-hat-end-user-license-agreements#UBI",
  "description": "Nginx is a web server and a reverse proxy server for HTTP, SMTP, POP3 and IMAP protocols, with a strong focus on high concurrency, performance and low memory usage. The container image provides a containerized packaging of the nginx 1.26 daemon. The image can be used as a base image for other applications based on nginx 1.26 web server. Nginx server image can be extended using source-to-image tool.",
  "distribution-scope": "public",
  "help": "For more information visit https://github.com/sclorg/nginx-container",
  "io.buildah.version": "1.40.1",
  "io.k8s.description": "Nginx is a web server and a reverse proxy server for HTTP, SMTP, POP3 and IMAP protocols, with a strong focus on high concurrency, performance and low memory usage. The container image provides a containerized packaging of the nginx 1.26 daemon. The image can be used as a base image for other applications based on nginx 1.26 web server. Nginx server image can be extended using source-to-image tool.",
  "io.k8s.display-name": "Nginx 1.26",
  "io.openshift.expose-services": "8443:https",
  "io.openshift.s2i.scripts-url": "image:///usr/libexec/s2i",
  "io.openshift.tags": "builder,nginx,nginx-126",
  "io.s2i.scripts-url": "image:///usr/libexec/s2i",
  "maintainer": "SoftwareCollections.org <sclorg@redhat.com>",
  "name": "ubi10/nginx-126",
  "release": "1756913183",
  "summary": "Platform for running nginx 1.26 or building nginx-based application",
  "url": "https://catalog.redhat.com/en/search?searchType=containers",
  "usage": "s2i build <SOURCE-REPOSITORY> ubi10/nginx-126 <APP-NAME>",
  "vcs-ref": "d38a13af239207ddef9285fec786e3766f89e9e9",
  "vcs-type": "git",
  "vendor": "Red Hat, Inc.",
  "version": "1"
}
student@serverXX:~$
```

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

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

```

student@serverXX:~$ podman inspect --format '{{json .Config.Labels.help}}' registry.access.redhat.com/ubi10/nginx-126 | jq
"For more information visit https://github.com/sclorg/nginx-container"
```

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

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

Versions

Nginx versions currently provided are:

- [nginx-1.20](#)
- [nginx-1.22](#)
- [nginx-1.22 micro](#)
- [nginx-1.1](#)
- [nginx-1.26](#)

Hier vind je zéér interessante info terug.

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

3. Prepare an application inside a container

This step usually consists of at least these parts:

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

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

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

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

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

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

3.2 is niet interessant voor ons op de website.

Onderstaande stappen komen ons bekend voor...

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

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

5. Run the resulting image with the final application

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

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

Direct usage with a mounted directory

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

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

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

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

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

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

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

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

Maak eerst de directorystructuur aan.

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

Maak de startpagina index.html aan.

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

```
Dit is een eenvoudige website
```

We controleren de webpagina.

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

```
Dit is een eenvoudige website
```

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

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

```
...
```

```
45bc04da9ccfc27077875faaad64e09499ab587adb340b06fbb7846b75a2f418
```

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

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



6.3 Database-server

We zoeken nu eens een mariadb-image.

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

```
...
```

```
registry.redhat.io/rhel10/mariadb-1011
```

```
...
```

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

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

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

```
Username: stijnjacobs
```

```
Password:
```

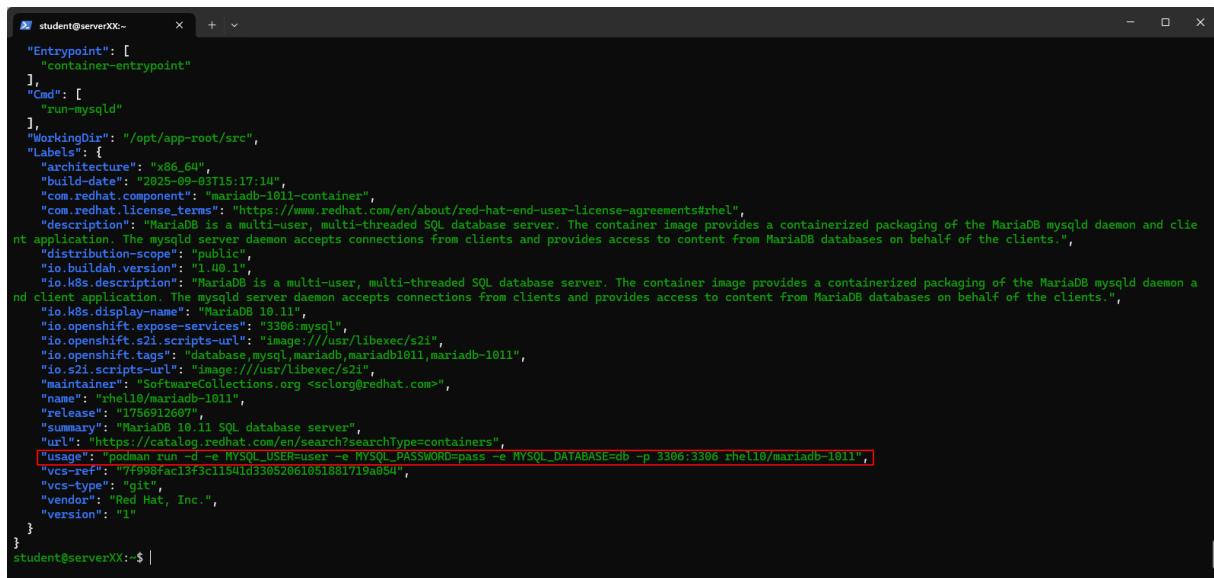
```
Login Succeeded!
```

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

```
...
```

```
69da3594e8f55d55013d1307b53e0c2e5cde7e19147fcf4270aa9fdb4935120b
```

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



```
student@serverXX:~$ podman inspect --format '{{json .Config}}' registry.redhat.io/rhel10/mariadb-1011 | jq
{
  "Entrypoint": [
    "container-entrypoint"
  ],
  "Cmd": [
    "run-mysqld"
  ],
  "WorkingDir": "/opt/app-root/src",
  "Labels": {
    "architecture": "x86_64",
    "build-date": "2025-09-03T15:17:14",
    "com.redhat.component": "mariadb-1011-container",
    "com.redhat.license-terms": "https://www.redhat.com/en/about/red-hat-end-user-license-agreements#rhel",
    "description": "MariaDB is a multi-user, multi-threaded SQL database server. The container image provides a containerized packaging of the MariaDB mysqld daemon and client application. The mysqld server daemon accepts connections from clients and provides access to content from MariaDB databases on behalf of the clients.",
    "distribution-scope": "public",
    "io.buildah.version": "1.40.1",
    "io.k8s.description": "MariaDB is a multi-user, multi-threaded SQL database server. The container image provides a containerized packaging of the MariaDB mysqld daemon and client application. The mysqld server daemon accepts connections from clients and provides access to content from MariaDB databases on behalf of the clients.",
    "io.k8s.display-name": "MariaDB 10.11",
    "io.openshift.expose-services": "3306:mysql",
    "io.openshift.s2i.repositories": "image:///usr/libexec/s2i",
    "io.openshift.tags": "database_mysql,mariadb,mariadb1011,mariadb-1011",
    "io.s2i.repositories": "image:///usr/libexec/s2i",
    "maintainer": "SoftwareCollections.org <sclorg@redhat.com>",
    "name": "rhel10/mariadb-1011",
    "release": "1756912607",
    "summary": "MariaDB 10.11 SQL database server",
    "url": "https://catalog.redhat.com/en/search?searchType=containers",
    "usage": "podman run -d -e MYSQL_USER=user -e MYSQL_PASSWORD=pass -e MYSQL_DATABASE=db -p 3306:3306 rhel10/mariadb-1011",
    "vcs-ref": "7f998fac13f3c11501d33052061051881719a054",
    "vcs-type": "git",
    "vendor": "Red Hat, Inc.",
    "version": "1"
  }
}
student@serverXX:~$
```

Hier staat heel duidelijk hoe je de containerimage moet gebruiken!

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

```
student@serverXX:~$ podman inspect --format '{{json .Config.Labels.usage}}' registry.redhat.io/rhel10/mariadb-1011 | jq
{
  "podman run -d -e MYSQL_USER=user -e MYSQL_PASSWORD=pass -e MYSQL_DATABASE=db -p 3306:3306 rhel10/mariadb-1011"
}
```

Aan de hand hiervan kunnen we onderstaande opbouwen.

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

```
17c89d9b9ffdae9508c46b33b39e5d7a4bc2c8b9f8f3becf3a2950fb7ba1e01b
```

De database-server is gestart en blijft draaien.

```
student@serverXX:~$ podman ps
CONTAINER ID  IMAGE                                     COMMAND      ...
17c89d9b9ffd  registry.redhat.io/rhel10/mariadb-1011:latest run-mysqld ...
```

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

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

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

We kunnen nu verbinding maken met het wachtwoord abc.

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

We gaan niet zelf de database configureren en verlaten MariaDB.

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

6.4 Wordpress

6.4.1 Images die we gebruiken

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

Jullie hebben toen daarvoor:

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

We gaan Wordpress installeren en configureren gebruik makend van containers.

We gaan nu gebruik maken van de officiële image van Wordpress op de Docker Hub:

<https://hub.docker.com>.

Zoek bovenaan naar wordpress en selecteer links Docker Official Image.

The screenshot shows the Docker Hub interface. In the search bar at the top, the word 'wordpress' is typed. Below the search bar, there is a sidebar with a 'Trusted content' section containing a checked checkbox for 'Docker Official Image'. The main area displays a single search result for 'wordpress' under the heading 'Docker Official Images'. This result includes a thumbnail with the WordPress logo, the text 'The WordPress rich content management system can utilize plugins, widgets, and...', and metrics: Pulls (18+), Stars (5824), and Last Updated (1 day). The entire search result card is highlighted with a red box.

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

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

How to use this image

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

Copy

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

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

Zoek nu naar mariadb.

1 - 2 of 2 results for mariadb.

mariadb Docker Official Image

MariaDB Server is a high performing open source relational database, forked from...

	Pulls	Stars	Last Updated
Docker Official Image	1B+	6026	9 days

phpmyadmin Docker Official Images

phpMyAdmin - A web interface for MySQL and MariaDB.

	Pulls	Stars	Last Updated
Docker Official Images	50M+	1116	1 day

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

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

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

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

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

6.4.2 Podman volumes aanmaken

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

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

6.4.3 Podman netwerk aanmaken

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

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

6.4.4 Container MariaDB

We zullen nu een container starten met MariaDB.

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

-d zorgt ervoor dat de container detached start.

-- name mariadb geeft de container de naam mariadb.

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

-e is een environment variable.

MARIADB_ROOT_PASSWORD = wachtwoord van de databasebeheerder root.

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

MARIADB_USER = er wordt een nieuwe gebruiker wp aangemaakt.

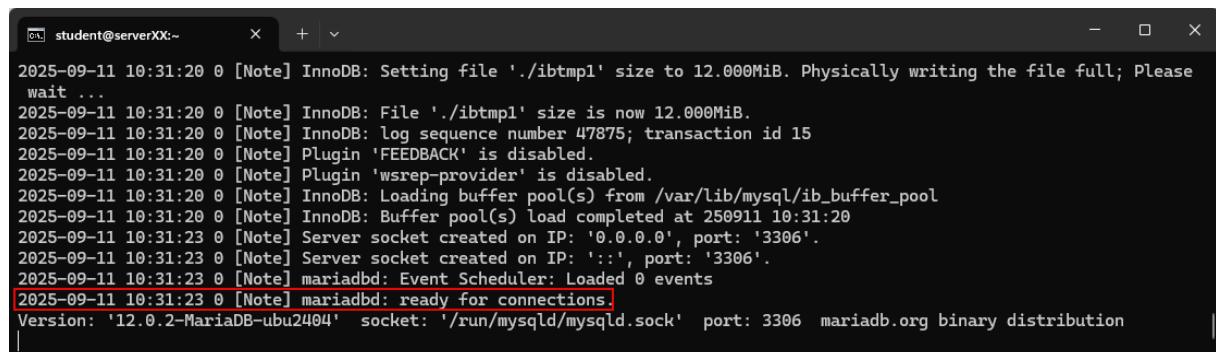
MARIADB_PASSWORD = wachtwoord van die gebruiker (wp).

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

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

Ter controle kan je kijken of MariaDB connecties aanvaardt.

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



```
student@serverXX:~ 2025-09-11 10:31:20 0 [Note] InnoDB: Setting file './ibtmp1' size to 12.000MiB. Physically writing the file full; Please wait ...  
2025-09-11 10:31:20 0 [Note] InnoDB: File './ibtmp1' size is now 12.000MiB.  
2025-09-11 10:31:20 0 [Note] InnoDB: log sequence number 47875; transaction id 15  
2025-09-11 10:31:20 0 [Note] Plugin 'FEEDBACK' is disabled.  
2025-09-11 10:31:20 0 [Note] Plugin 'wsrep-provider' is disabled.  
2025-09-11 10:31:20 0 [Note] InnoDB: Loading buffer pool(s) from /var/lib/mysql/ib_buffer_pool  
2025-09-11 10:31:20 0 [Note] InnoDB: Buffer pool(s) load completed at 250911 10:31:20  
2025-09-11 10:31:23 0 [Note] Server socket created on IP: '0.0.0.0', port: '3306'.  
2025-09-11 10:31:23 0 [Note] Server socket created on IP: '::', port: '3306'.  
2025-09-11 10:31:23 0 [Note] mariadb: Event Scheduler: Loaded 0 events  
2025-09-11 10:31:23 0 [Note] mariadb: ready for connections.  
Version: '12.0.2-MariaDB-ubuntu2404' socket: '/run/mysqld/mysqld.sock' port: 3306 mariadb.org binary distribution
```

Ga eruit door <CTRL> + C te typen.

De container blijft draaien zolang mariadb draait.

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

6.4.5 Container Wordpress

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

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

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

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

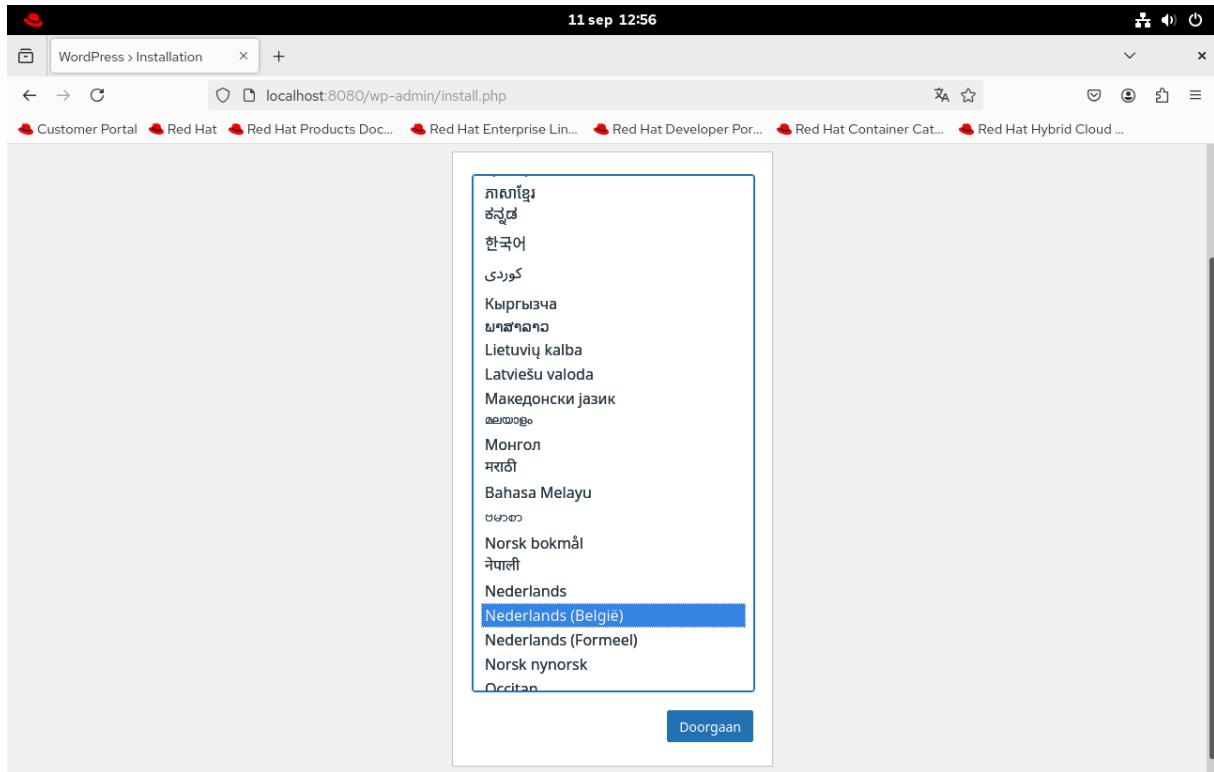
```
student@serverXX:~$ netstat -tlpn | grep 8080
...
tcp6      0      0 ::::8080      ::::*      LISTEN      5279/rootlessport
```

Er is dus een website beschikbaar op poort 8080.

6.4.6 Website Wordpress configureren

Ga naar de browser van ServerXX en typ <http://localhost:8080> in.

Kies in het eerste vensster voor Nederlands (België) zoals je hieronder ziet.



Klik erna op Doorgaan.

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

Vul nu gegevens in analoog met onderstaande.

11 sep 12:58

WordPress > installatie

localhost:8080/wp-admin/install.php?step=1

Customer Portal Red Hat Red Hat Products Doc... Red Hat Enterprise Lin... Red Hat Developer Por... Red Hat Container Cat... Red Hat Hybrid Cloud ...

gebruiken.

Benodigde informatie

De volgende informatie invoeren. Maak je geen zorgen, deze instellingen kunnen steeds worden gewijzigd.

Website titel test wordpress

Gebruikersnaam student
Gebruikersnamen mogen alleen alfanumerieke karakters, spaties, underscores, koppeltekens, punten en het @ symbool bevatten.

Wachtwoord student Verbergen
Erg zwak
Belangrijk: Dit wachtwoord is nodig om in te loggen. Zorg ervoor dat je het bewaart op een geheime locatie.

Bevestig wachtwoord Bevestig het gebruik van een zwak wachtwoord

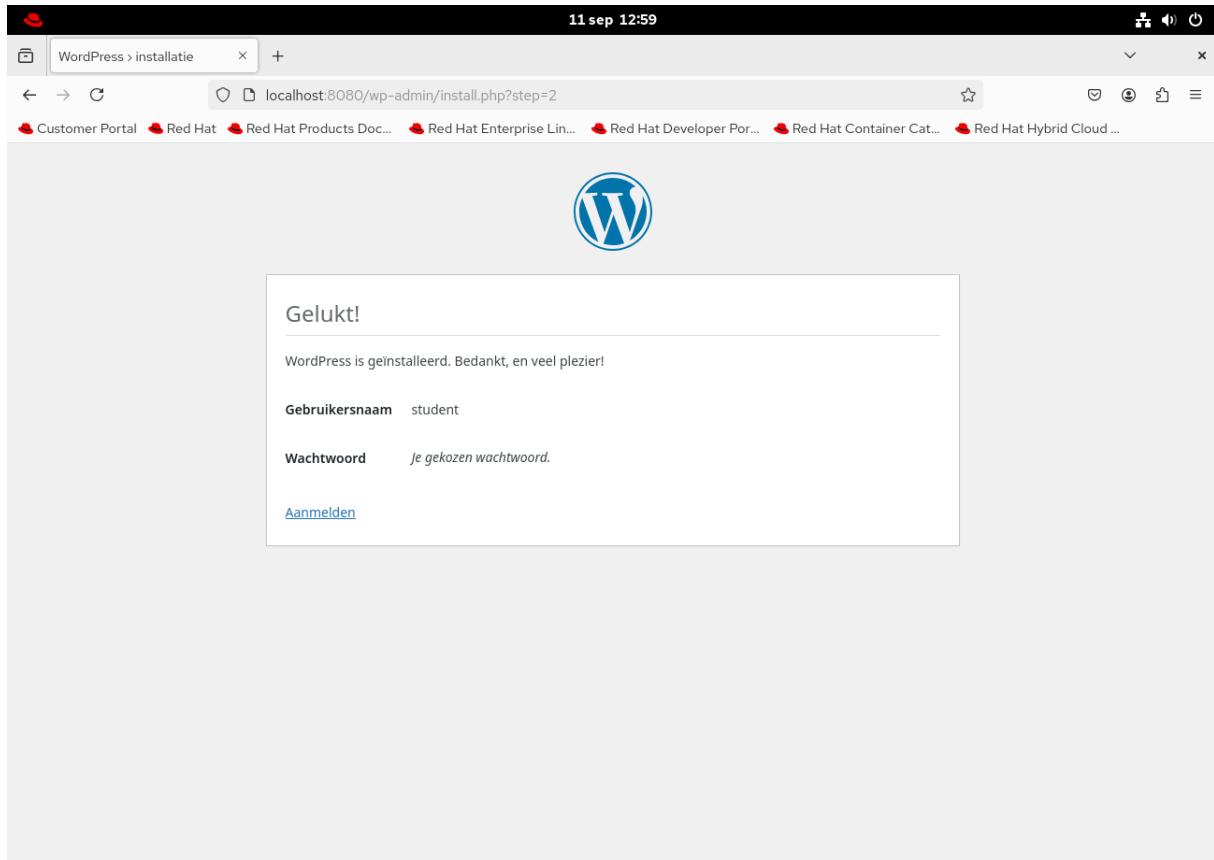
Je e-mailadres stijn.jacobs@pxl.be
Controleer zorgvuldig of je het e-mailadres goed hebt ingevuld voordat je verder gaat.

Zoekmachine zichtbaarheid Blokkeer zoekmachines deze website te indexeren
Het is aan de zoekmachines of ze gehoor geven aan dit verzoek.

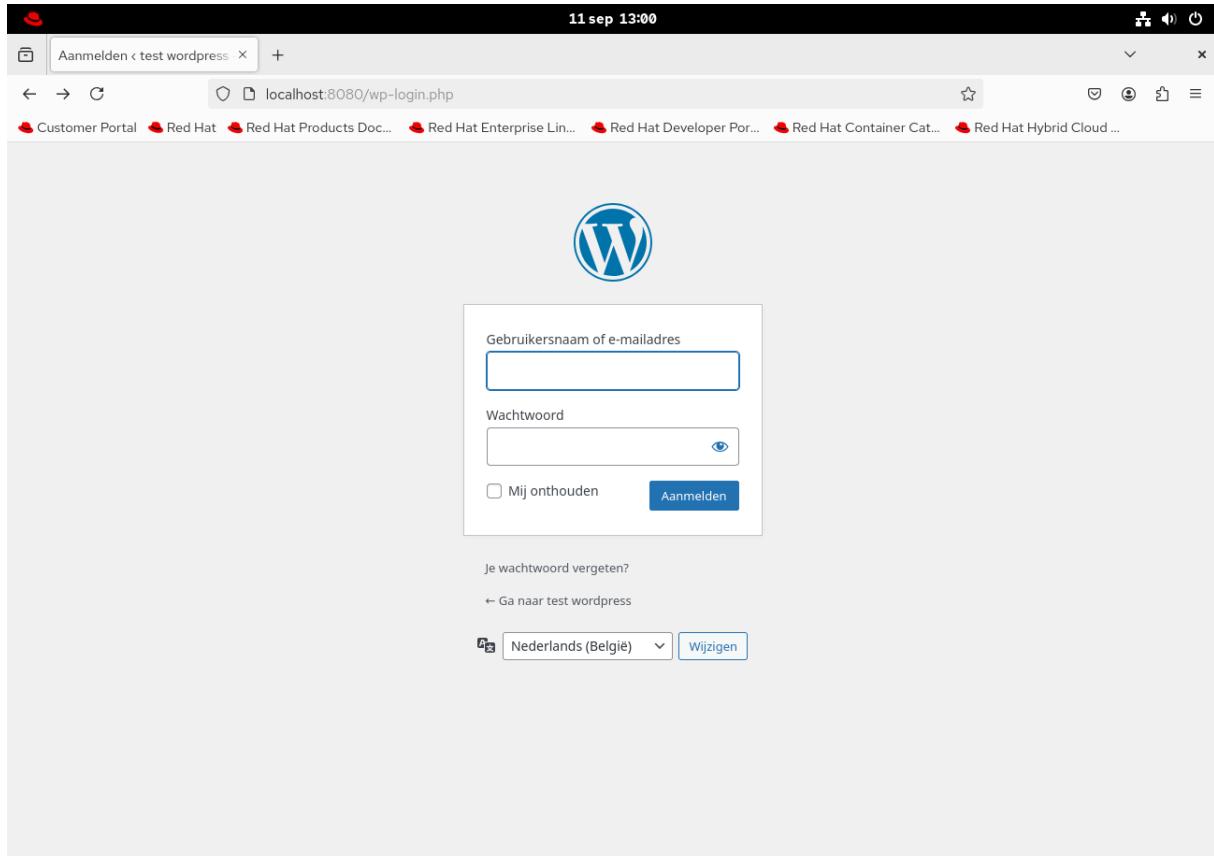
WordPress installeren

Klik erna uiteraard op WordPress installeren.

Je krijgt nu de melding dat WordPress is geïnstalleerd.



Klik uiteraard op Aanmelden.



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

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

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

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

6.5 Tips

Elke officiële image heeft een uitgebreide beschrijving.

Daar vind je info over o.a.:

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

7 Podman Compose

7.1 Inleiding

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

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

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

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

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

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

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

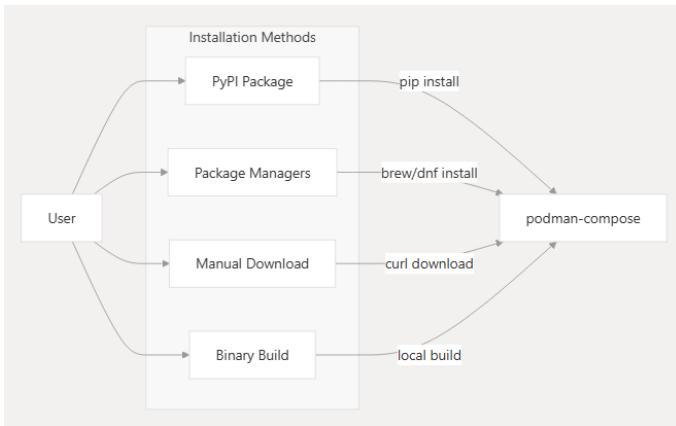
Podman-compose gebruikt dezelfde Compose-spec (het YAML-formaat) als Docker Compose. Het verschil zit alleen in de engine erachter:

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

7.2 Installatie Podman Compose

We installeren podman-compose op ServerXX.

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



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

Installeer hiervoor eerst pip3.

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

Daarna installeer je via pip3 podman-compose.

```
student@serverXX:~$ pip3 install podman-compose
Defaulting to user installation because normal site-packages is not writeable
Collecting podman-compose
...

```

Na installatie is het commando uiteraard beschikbaar.

```
student@serverXX:~$ podman-compose
usage: podman-compose [-h] [-v] [--in-pod in_pod] [--pod-args pod_args] [--env-file env_file] [-f file]
...

```

We checken de versie van podman-compose.

```
student@serverXX:~$ podman-compose --version
podman-compose version 1.5.0
podman version 5.4.0
```

7.3 Componenten van een .yml-bestand

7.3.1 Services

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

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

Hieronder de verschillende onderdelen:

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

Let op met inspingen:

- Je moet spaties gebruiken, geen tabs. Tabs zijn verboden omdat ze op verschillende systemen anders geïnterpreteerd worden.

- Je mag zelf kiezen hoeveel spaties je gebruikt per niveau — 2, 4 of zelfs 6 — zolang je binnen hetzelfde bestand consequent blijft.
- De aanbevolen standaard is 2 spaties per niveau, omdat het overzichtelijk en compact is.

7.3.2 Volumes

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

```
volumes:
  shared_volume:
```

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

Toewijzen van volumes aan een service...

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

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

7.3.3 Netwerken

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

```
networks:
  - rhel_network
```

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

Een netwerk gebruiken in services...

```
services:
  rhel_worker:
    ...
    networks:
      - rhel_network
```

- `rhel_network`

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

7.3.4 Depends_on

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

`services:`

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

Dat betekent: start eerst `config_writer`, dan pas `app`.

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

7.3.5 Environment-variabelen

Deze worden gebruikt om de containerinstellingen te beheren.

`services:`

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

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

7.3.6 Command

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

Hieronder enkele voorbeelden.

```
services:
```

```
  app:
```

```
    command: "echo Hallo wereld"
```

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

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

Dit is de lijstvorm.

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

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

Gebruik dit niet!

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

Dit werkt dan uiteraard niet!!!

7.3.7 Folded block en literal block

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

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

Voorbeeld folded block:

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

Dit wordt geïnterpreteert als volgd:

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

Voorbeeld literal block:

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

Dit wordt geïnterpreteerd als volgt:

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

7.4 Voorbeeld 1: gedeelde map

7.4.1 .yml-bestand

Genoeg uitleg hierover...

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

```
services:  
  config_writer:  
    image: registry.access.redhat.com/ubi10/ubi  
    container_name: config_writer  
    volumes:  
      - shared_volume:/config:z  
    command: >  
      /bin/bash -c "sleep 5;  
      echo 'config=true' > /config/appsettings.txt;  
      echo 'Configuratiebestand geschreven naar /config/appsettings.txt';  
      sleep 20"  
  
  app:  
    image: registry.access.redhat.com/ubi10/ubi  
    container_name: app  
    depends_on:  
      - config_writer  
    volumes:  
      - shared_volume:/app_config:z  
    command: >  
      /bin/bash -c "sleep 10;  
      if [ -f /app_config/appsettings.txt ]; then  
        cat /app_config/appsettings.txt;  
      else
```

```
    echo 'Configuratiebestand niet gevonden.';  
fi"
```

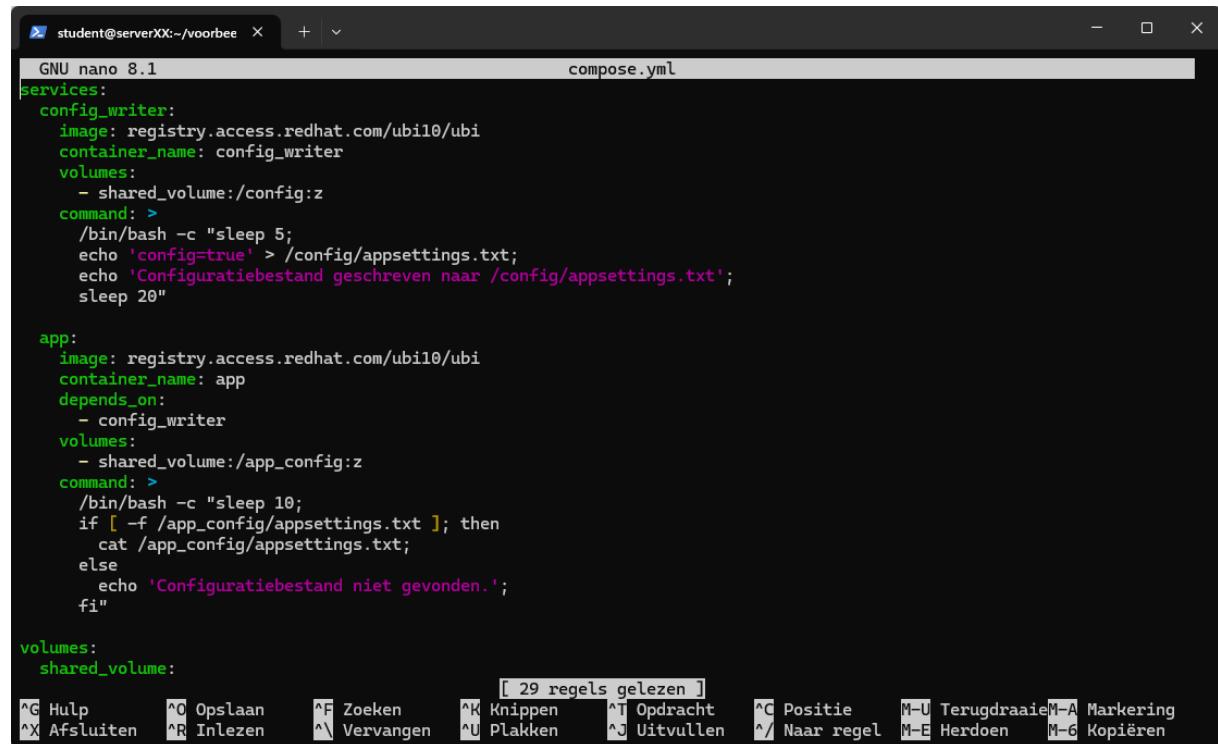
volumes:

shared_volume:

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

```
student@serverXX:~$ mkdir voorbeeld1
```

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



```
student@serverXX:~/voorbee compose.yml  
GNU nano 8.1  
services:  
  config_writer:  
    image: registry.access.redhat.com/ubi10/ubi  
    container_name: config_writer  
    volumes:  
      - shared_volume:/config:z  
    command: >  
      /bin/bash -c "sleep 5;  
      echo 'config=true' > /config/appsettings.txt;  
      echo 'Configuratiebestand geschreven naar /config/appsettings.txt';  
      sleep 20"  
  
  app:  
    image: registry.access.redhat.com/ubi10/ubi  
    container_name: app  
    depends_on:  
      - config_writer  
    volumes:  
      - shared_volume:/app_config:z  
    command: >  
      /bin/bash -c "sleep 10;  
      if [ -f /app_config/appsettings.txt ]; then  
        cat /app_config/appsettings.txt;  
      else  
        echo 'Configuratiebestand niet gevonden.';  
      fi"  
  
volumes:  
  shared_volume:  
    [ 29 regels gelezen ]  
  ^G Hulp      ^O Opslaan      ^F Zoeken      ^K Knippen      ^T Opdracht      ^C Positie      M-U Terugdraaien  
  ^X Afsluiten  ^R Inlezen      ^V Vervangen     ^U Plakken      ^J Uitvullen     ^/ Naar regel   M-E Herdoen    M-6 Kopiëren
```

Je kan uiteraard ook gebruik maken van Visual Studio Code.

```

student@192.168.112.100: ~$ cat voorbeeld1/compose.yml
version: '3'
services:
  config_writer:
    image: registry.access.redhat.com/ubi10/ubi
    container_name: config_writer
    volumes:
      - shared_volume:/config:z
    command: >
      /bin/bash -c "sleep 5;
      echo 'config=true' > /config/appsettings.txt;
      echo 'Configuratiebestand geschreven naar /config/appsettings.txt';
      sleep 20"
  app:
    image: registry.access.redhat.com/ubi10/ubi
    container_name: app
    depends_on:
      - config_writer
    volumes:
      - shared_volume:/app_config:z
    command: >
      /bin/bash -c "sleep 10;
      if [ -f /app_config/appsettings.txt ]; then
        cat /app_config/appsettings.txt;
      else
        echo 'Configuratiebestand niet gevonden.';
      fi"
    volumes:
      - shared_volume:

```

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

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

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

7.4.2 Podman-compose up

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

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

Voer het volgende commando uit:

```
student@serverXX:~/voorbeeld1$ podman-compose up
c9f5c430823dcf4e1f090bd53282efbfe17f983c29cfcc3bd0ddb53b895ecc0b
9ae589470b1a4334261ccc37796aa1eb996215cb82edef29255962578c348e72
9a29ac43fad3c6ab1b8f6f28e65c87be3ccb3455afb2dca317cdf9bb7a49f678
[config_writer] | Configuratiebestand geschreven naar /config/appsettings.txt
```

```
[app] | config=true
```

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

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

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

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

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

```
student@serverXX:~/voorbeeld1$ podman-compose up  
[config_writer] | Configuratiebestand geschreven naar /config/appsettings.txt  
[app] | config=true  
student@serverXX:~/voorbeeld1$
```

We leggen dit nu gedetailleerd uit.

1. Lezen en verwerken van compose.yml

In dit geval detecteert Podman Compose twee services:

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

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

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

2. Starten van de config_writer service

Docker Compose start de config_writer container met de volgende instellingen:

- De container gebruikt de Docker-image registry.access.redhat.com/ubi10/ubi.
- Het volume shared_volume wordt gemount naar de map /config in de container.
- Het opgegeven command wordt uitgevoerd, namelijk een Bash-script.
 - o /bin/bash -c "sleep 5;
De container wacht 5 seconden voordat hij verder gaat.
 - o echo 'config=true' > /config/appsettings.txt;
Daarna schrijft hij de string 'config=true' naar een bestand genaamd appsettings.txt in de map /config, die wordt gedeeld via het volume.

- echo 'Configuratiebestand geschreven naar /config/appsettings.txt': Deze regel wordt naar de container-uitvoer geschreven om aan te geven dat het configuratiebestand succesvol is geschreven.
- sleep 20: Ten slotte wacht de container nog eens 20 seconden voordat hij volledig stopt.

3. Starten van de app service (na depends_on)

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

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

De app container wordt vervolgens gestart met de volgende instellingen:

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

4. Duur van het uitvoeren van de containers

- config_writer
Deze container blijft draaien gedurende de “sleep 20” fase, maar zal daarna stoppen.
- App
Deze container zal direct na het uitvoeren van het script stoppen.

7.4.3 Logs bekijken

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

```
student@serverXX:~/voorbeeld1$ podman logs config_writer
Configuratiebestand geschreven naar /config/appsettings.txt

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

- Je ziet 2 keer het weggeschreven bericht omdat je 2x “podman-compose up” hebt uitgevoerd.
`student@serverXX:~/voorbeeld1$ podman logs app`

`config=true`

`config=true`

- o Als het bestand appsettings.txt bestaat, zal de inhoud (config=true) worden weergegeven.

- o Dat is hier 2x gebeurd omdat je 2x “podman-compose up” hebt uitgevoerd.

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

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

`9a29ac43fad3 config=true`

`9a29ac43fad3 config=true`

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

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

7.4.4 Podman-compose down

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

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

`app`

`config_writer`

`app`

`config_writer`

`c9f5c430823dcf4e1f090bd53282efbfe17f983c29cfcc3bd0ddb53b895ecc0b`

`voorbeeld1_default`

De containers en netwerken zijn nu verwijderd.

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

`...`

`student@serverXX:~/voorbeeld1$ podman network ls`

NETWORK ID	NAME	DRIVER
2f259bab93aa	podman	bridge

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

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

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

```
student@serverXX:~/voorbeeld1$ podman-compose up  
...  
student@serverXX:~/voorbeeld1$ podman-compose down -v  
  
student@serverXX:~/voorbeeld1$ ls  
/home/student/.local/share/containers/storage/volumes/  
  
student@serverXX:~/voorbeeld1$
```

7.4.5 Podman-compose down versus podman-compose stop

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

7.4.6 Podman-compose up -d

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

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

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

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

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

```
11284f1dc973 Configuratiebestand geschreven naar /config/appsettings.txt  
d1e2f29cfa5e config=true
```

Je ziet nu de log entries van alle containers.

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

```
student@serverXX:~/voorbeeld1$ podman-compose logs config_writer  
Configuratiebestand geschreven naar /config/appsettings.txt
```

7.4.7 Podman-compose help

Voer onderstaande hiervoor uit.

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

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

```
student@serverXX:~/voorbeeld1$ podman-compose --help  
help show help  
version show version  
wait wait running containers to stop  
systemd create systemd unit file and register its compose stacks  
pull pull stack images  
push push stack images  
build build stack images  
up Create and start the entire stack or some of its services  
down tear down entire stack  
ps show status of containers  
run create a container similar to a service to run a one-off command  
exec execute a command in a running container  
start start specific services  
stop stop specific services  
restart restart specific services  
logs show logs from services  
config displays the compose file  
port Prints the public port for a port binding.  
pause Pause all running containers  
unpause Unpause all running containers  
kill Kill one or more running containers with a specific signal  
stats Display percentage of CPU, memory, network I/O, block I/O and PIDs for services.  
images List images used by the created containers  
student@serverXX:~/voorbeeld1$ |
```

7.5 Voorbeeld 2: Wordpress

In vorig hoofdstuk hebben we Wordpress geïnstalleerd in 2 containers.

We kunnen dat ook opbouwen met Podman-compose.

7.5.1 .yml-bestand

We maken eerst weer een directory aan.

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

```

services:

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

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

```

```
networks:
```

```
- wpNet
```

```
volumes:
```

```
mysql-data:
```

```
wp-content:
```

```
networks:
```

```
wpNet:
```

The screenshot shows a terminal window titled "student@serverXX:~". The file being edited is "wordpressmysql/compose.yml". The content of the file is as follows:

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

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

volumes:
  mysql-data:
  wp-content:

networks:
  wpNet:
```

At the bottom of the terminal window, there is a menu bar with the following options: Hulp (Help), Opslaan (Save), Zoeken (Search), Knippen (Cut), Opdracht (Command), Positie (Position), Terugdraaien (Undo), Afsluiten (Close), Inlezen (Paste), Vervangen (Replace), Plakken (Paste), Uitvullen (Fill), Naar regel (To line), Herdoen (Redo), and Kopiëren (Copy).

Of met Visual Studio Code natuurlijk...

```

compose.yml

wordpressmysql > compose.yml
1 services:
2   mariadb:
3     image: mariadb:latest
4     container_name: mariadb
5     restart: always
6     environment:
7       MARIADB_ROOT_PASSWORD: ServerXXdocker007
8       MARIADB_DATABASE: wordpress
9       MARIADB_USER: wp
10      MARIADB_PASSWORD: ServerXXdocker1
11      volumes:
12        - mysql-data:/var/lib/mysql:Z
13      networks:
14        - wnet
15
16  wordpress:
17    image: wordpress:latest
18    container_name: wordpress
19    restart: always
20    ports:
21      - "8080:80"
22    environment:
23      WORDPRESS_DB_HOST: mariadb:3306
24      WORDPRESS_DB_USER: wp
25      WORDPRESS_DB_PASSWORD: ServerXXdocker1
26      WORDPRESS_DB_NAME: wordpress
27    volumes:
28      - wp-content:/var/www/html/wp-content:Z
29    networks:
30      - wnet
31
32  volumes:
33    mysql-data:
34    wp-content:
35
36  networks:
37    wnet:

```

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

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

Je ziet ook de poortkoppeling bij Wordpress.

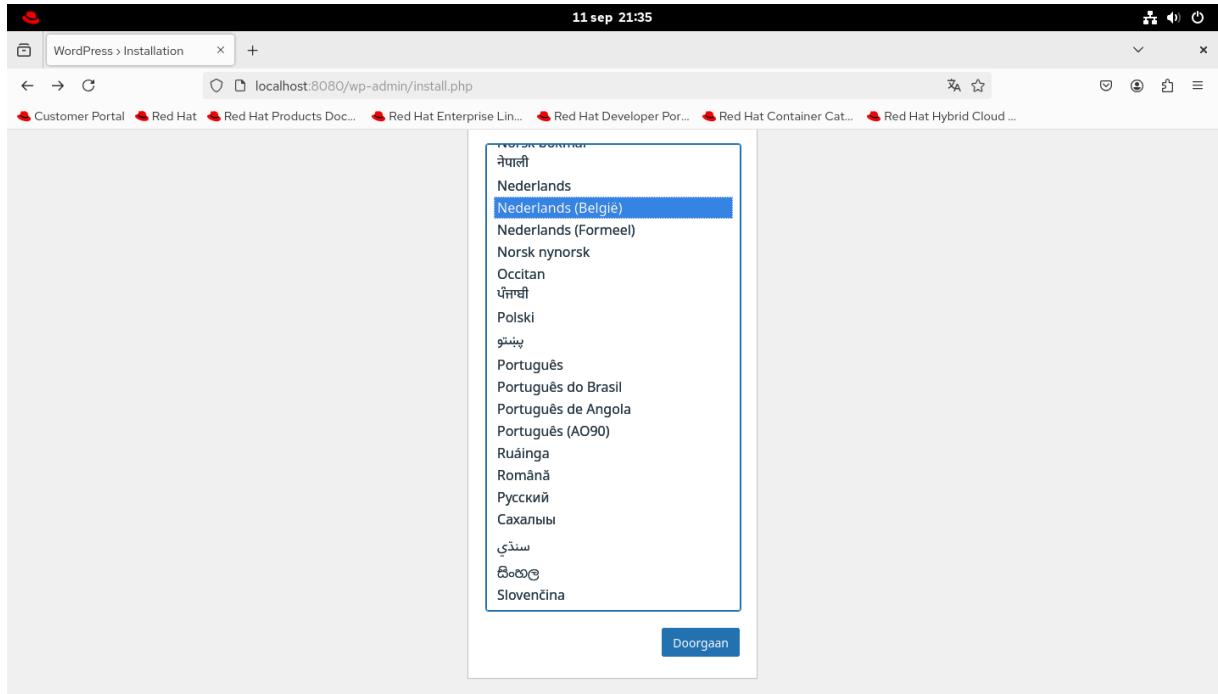
7.5.2 Podman-compose up

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

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

7.5.3 Website instellen

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



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

8 Podman Compose in combinatie met Containerfile

8.1 Inleiding

Eén of meerdere container files en compose.yml kunnen samenwerken om het bouwen, configureren en beheren van containers te vereenvoudigen.

- Een Containerfile is een script met instructies voor het bouwen van een Podman-image. Het bevat alle stappen die nodig zijn om een omgeving te creëren zoals al behandeld:
 - o De basisimage (FROM).
 - o Het kopiëren van bestanden (COPY).
 - o Installatie (RUN).
 - o Het definiëren van een startcommando (CMD/ENTRYPOINT).
 - o Status checken (HEALTHCHECK)
- Met compose.yml kan je meerdere containers definiëren aan de hand van images. Dit bestand maakt het mogelijk om:
 - o Verschillende services te definiëren (zoals databases en applicaties).
 - o Netwerken en volumes aan te maken.
 - o Poorten door te sturen en omgevingsvariabelen in te stellen.

8.2 Voorbeeld 1

We bouwen een eenvoudige container die een bash-script uitvoert.

Maak een map vb1.

```
student@serverXX:~$ mkdir vb1
```

Hierin zullen we Dockerfile, script.sh en compose.yml plaatsen.

8.2.1 Containerfile

In dit voorbeeld vertrekken we van een registry.access.redhat.com/ubi10/ubi-image en voegen we een simpel bash-script toe dat in de container wordt uitgevoerd.

```
student@serverXX:~$ nano vb1/Dockerfile

FROM registry.access.redhat.com/ubi10/ubi

COPY script.sh /scripts/script.sh

RUN chmod +x /scripts/script.sh

WORKDIR /scripts

CMD ["/bin/bash", "./script.sh"]
```

Dit Containerfile-bestand doet het volgende:

- Het gebruikt registry.access.redhat.com/ubi10/ubi -image als basis.

- Kopieert een bash-script (script.sh) naar de container.
- Geeft het script uitvoerrechten.
- Stelt de werkdirectory in op /scripts
- Voert het bash-script uit bij het starten van de container.

8.2.2 Bash script (script.sh)

Hier is een voorbeeld van het bash-script dat in de container wordt uitgevoerd. Dit script print gewoon een simpele bericht naar de console.

```
student@serverXX:~$ nano vb1/script.sh

#!/bin/bash

echo "Hallo, dit is een bash-script dat draait in een RHEL-container!"
```

8.2.3 compose.yml

Dit is het Compose-bestand dat een container aanmaakt op basis van de dockerfile.

```
student@serverXX:~$ nano vb1/compose.yml

services:

bashrunner:

build:
  context: .
  dockerfile: Dockerfile
  container_name: bashrunner

networks:
  - bashnet

networks:
  bashnet:
```

```
GNU nano 8.1
services:
  bashrunner:
    build:
      context: .
      dockerfile: Dockerfile
      container_name: bashrunner
    networks:
      - bashnet

networks:
  bashnet:
```

Dit Compose-bestand doet het volgende:

- Definieert een service genaamd bashrunner.
- Gebruikt de Dockerfile om de container te bouwen.
- Plaatst de container op een netwerk genaamd bashnet.

8.2.4 Uitvoer

Voer het volgende uit in de command line vanuit de directory waar je docker-compose.yml staat:

```
student@serverXX:~$ cd vb1
student@serverXX:~/vb1$ podman-compose up --build
...
COMMIT vb1_bashrunner
...
[bashrunner] | Hallo, dit is een bash-script dat draait in een RHEL-container!
```

Aangezien we geen imagenaam gespecificeerd hebben krijgt deze een naam als volgt: <projectnaam>_<servicenaam>. De projectnaam bij het gebruik van podman-compose wordt standaard afgeleid van de naam van de directory waarin het compose-bestand zich bevindt. Vandaar de naam vb1_bashrunner voor de image.

Je kan erna de container ook starten zonder eerst de image te builden omdat deze nu reeds bestaat.

```
student@serverXX:~/vb1$ podman-compose up
[bashrunner] | Hallo, dit is een bash-script dat draait in een RHEL-container!
```

8.3 Voorbeeld 2

Hier bouwen zelf een eenvoudige webserver op basis van UBI10 met Apache.

Maak een map vb2 aan met daarin:

- Bestand Containerfile
- Map mywebsite/ met daarin bestand index.html
- Bestand compose.yml

We maken eerst de map vb2 aan.

```
student@serverXX:~$ mkdir -p vb2/mywebsite
```

8.3.1 Containerfile

```
student@serverXX:~$ nano vb2/Containerfile
FROM registry.access.redhat.com/ubi10/httpd-24
COPY mywebsite/ /var/www/html/
```

EXPOSE 8080

In dit Containerfile-bestand:

- We gebruiken de httpd-server gebaseerd op ubi10.
- We plaatsen de inhoud van de subdirectory mywebsite in /var/www/html/
- We documenteren poort 8080 voor verkeer (↔ publiceren!). Dit is niet verplicht.

De container moet niet draaiende gehouden worden aangezien

registry.access.redhat.com/ubi10/httpd-24 daar zelf voor zorgt.

8.3.2 Index.html

We plaatsen dit bestand in mywebsite/index.html.

```
student@serverXX:~$ nano vb2/mywebsite/index.html

<head>

    <title>Voorbeeld 2 website</title>

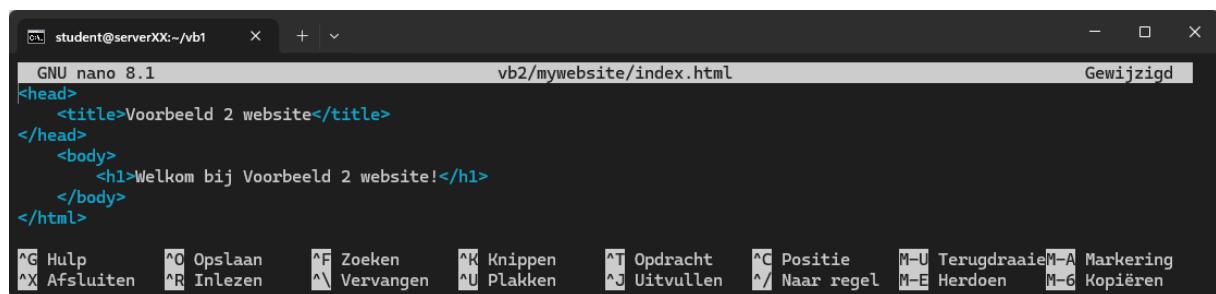
</head>

<body>

    <h1>Welkom bij Voorbeeld 2 website!</h1>

</body>

</html>
```



A screenshot of a terminal window titled "student@serverXX:~/vb1". The window shows the file "vb2/mywebsite/index.html" being edited in the nano text editor. The content of the file is displayed in the terminal window. The terminal window has a dark background and light-colored text. At the bottom, there is a menu bar with options like "Hulp", "Opslaan", "Zoeken", etc., and a status bar indicating "Gewijzigd".

```
GNU nano 8.1                                     vb2/mywebsite/index.html                                     Gewijzigd
<head>
    <title>Voorbeeld 2 website</title>
</head>
<body>
    <h1>Welkom bij Voorbeeld 2 website!</h1>
</body>
</html>
```

8.3.3 Compose.yml

We geven dit bestand onderstaande inhoud.

```
student@serverXX:~$ nano vb2/compose.yml

services:

  webserver:

    build:
      context: .

    dockerfile: Containerfile
```

```

image: wwwimage

container_name: webserver

ports:
  - "8080:8080"

networks:
  - webnet

networks:
  webnet:

```

```

student@serverXX:~/vb1/vb2  ~  + | v
GNU nano 8.1                               compose.yml                                Gewijzigd
services:
  webserver:
    build:
      context: .
      dockerfile: Containerfile
    image: wwwimage
    container_name: webserver
    ports:
      - "8080:8080"
    networks:
      - webnet

networks:
  webnet:
|_
```

^G Hulp ^O Opslaan ^F Zoeken ^K Knippen ^T Opdracht ^C Positie M-U Terugdraaien M-A Markering
 ^X Afsluiten ^R Inlezen ^V Vervangen ^U Plakken ^J Uitvullen ^/ Naar regel M-E Herdoen M-G Kopiëren

In dit Compose-bestand:

- definiëren we een service webserver die de Containerfile gebruikt om de container te bouwen.
- Hier is gekozen om zelf een naam toe te wijzen voor de image die wordt aangemaakt: `wwwimage`.
- De poort 80 van de container wordt gekoppeld aan poort 8080 op de hostmachine, zodat je de website kunt bezoeken via o.a. <http://localhost:8080>.
- De container draait op een webnet-netwerk.

8.3.4 Uitvoer

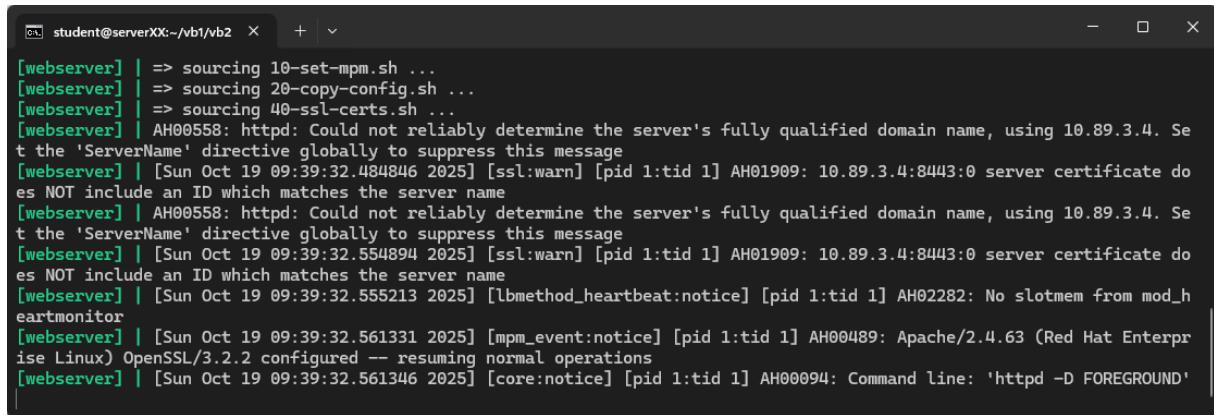
Voer in de map vb2 het volgende commando uit om de container te bouwen en te draaien:

```

student@serverXX:~$ cd vb2
student@serverXX:~/vb2$ podman-compose up --build

```

Zoals je ziet blijft de webserver draaien.



A screenshot of a terminal window titled "student@serverXX:~/vb1/vb2". The window contains several lines of Apache log messages. Key entries include:

- [webserver] | => sourcing 10-set-mpm.sh ...
- [webserver] | => sourcing 20-copy-config.sh ...
- [webserver] | => sourcing 40-ssl-certs.sh ...
- [webserver] | AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 10.89.3.4. Set the 'ServerName' directive globally to suppress this message
- [webserver] | [Sun Oct 19 09:39:32.484846 2025] [ssl:warn] [pid 1:tid 1] AH01909: 10.89.3.4:8443:0 server certificate does NOT include an ID which matches the server name
- [webserver] | AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 10.89.3.4. Set the 'ServerName' directive globally to suppress this message
- [webserver] | [Sun Oct 19 09:39:32.554894 2025] [ssl:warn] [pid 1:tid 1] AH01909: 10.89.3.4:8443:0 server certificate does NOT include an ID which matches the server name
- [webserver] | [Sun Oct 19 09:39:32.555213 2025] [lbmethod_heartbeat:notice] [pid 1:tid 1] AH02282: No slotmem from mod_heartbeatmonitor
- [webserver] | [Sun Oct 19 09:39:32.561331 2025] [mpm_event:notice] [pid 1:tid 1] AH00489: Apache/2.4.63 (Red Hat Enterprise Linux) OpenSSL/3.2.2 configured -- resuming normal operations
- [webserver] | [Sun Oct 19 09:39:32.561346 2025] [core:notice] [pid 1:tid 1] AH00094: Command line: 'httpd -D FOREGROUND'

Ga naar een ander terminalvenster en voer curl localhost:8080 of curl 192.168.112.100:8080 uit.

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

<head>

<title>Voorbeeld 2 website</title>

</head>

<body>

<h1>Welkom bij Voorbeeld 2 website!</h1>

</body>

</html>
```

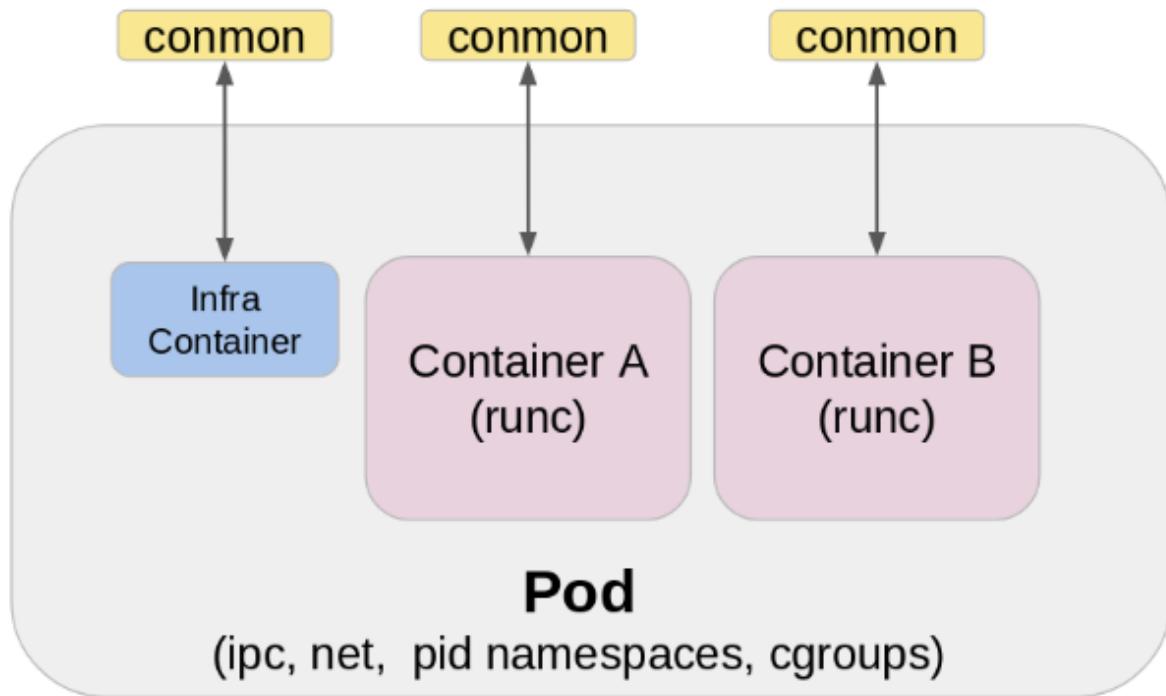
Je ziet dat de webpagina beschikbaar is.

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
```

...

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES	POD ID
PODNAME							
3136cb2b750	localhost/podman-pause:5.4.0-1750809600		About an hour ago	Exited (0) About a minute ago		a518b698331a-infra	a518b6
98331a mypod	6f297a9b39d3 registry.access.redhat.com/ubi10/ubi:latest	/bin/bash	10 minutes ago	Exited (137) About a minute ago		myubi	a518b6
98331a mypod							

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 rm opdracht.

```
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" -e MYSQL_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
```

```
...
```

```
student@serverXX:~$ podman ps -a --pod
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES POD ID PODNAME
b082fbfdcc2f7 localhost/podman-pause:5.4.0-1750809609 mariadb 6 minutes ago Up 31 seconds 0.0.0.0:8080->80/tcp 527e0e3a0cfc1-infra 527e0e3a0cfc1 wp-pod
57e09fc0bf99 docker.io/library/mariadb:latest 39 seconds ago Up 31 seconds 0.0.0.0:8080->80/tcp, 3306/tcp wp-db 527e0e3a0cfc1 wp-pod
student@serverXX:~$
```

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.

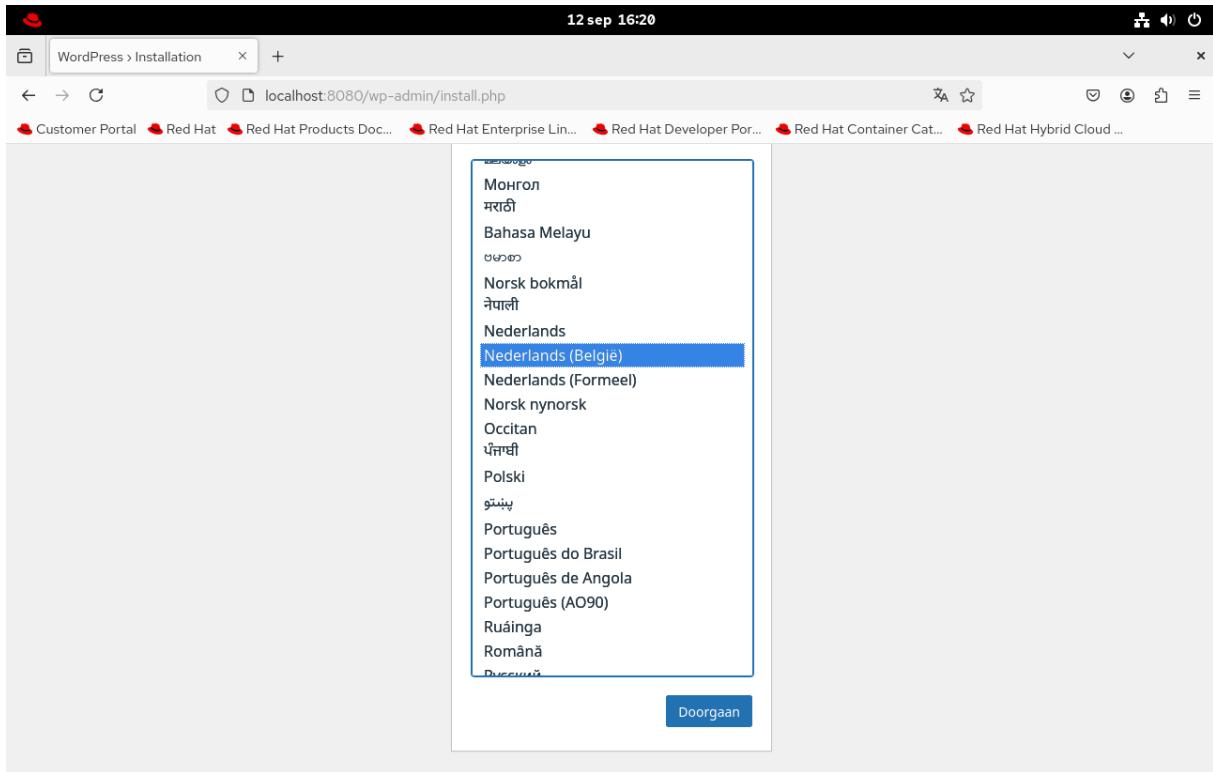
```
student@serverXX:~$ podman ps -a --pod
```

```
...
```

```
student@serverXX:~$ podman ps -a --pod
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES POD ID PODNAME
PODNAME
b082fbfdcc2f7 localhost/podman-pause:5.4.0-1750809609 12 minutes ago Up 6 minutes 0.0.0.0:8080->80/tcp 527e0e3a0cfc1-infra 527e0e3a0cfc1
wp-pod
57e09fc0bf99 docker.io/library/mariadb:latest mariadb 6 minutes ago Up 6 minutes 0.0.0.0:8080->80/tcp, 3306/tcp wp-db 527e0e3a0cfc1
wp-pod
c06cecef4e7c docker.io/library/wordpress:latest apache2-foreground... 2 minutes ago Up 2 minutes 0.0.0.0:8080->80/tcp wp-web 527e0e3a0cfc1
wp-pod
student@serverXX:~$
```

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
```

...

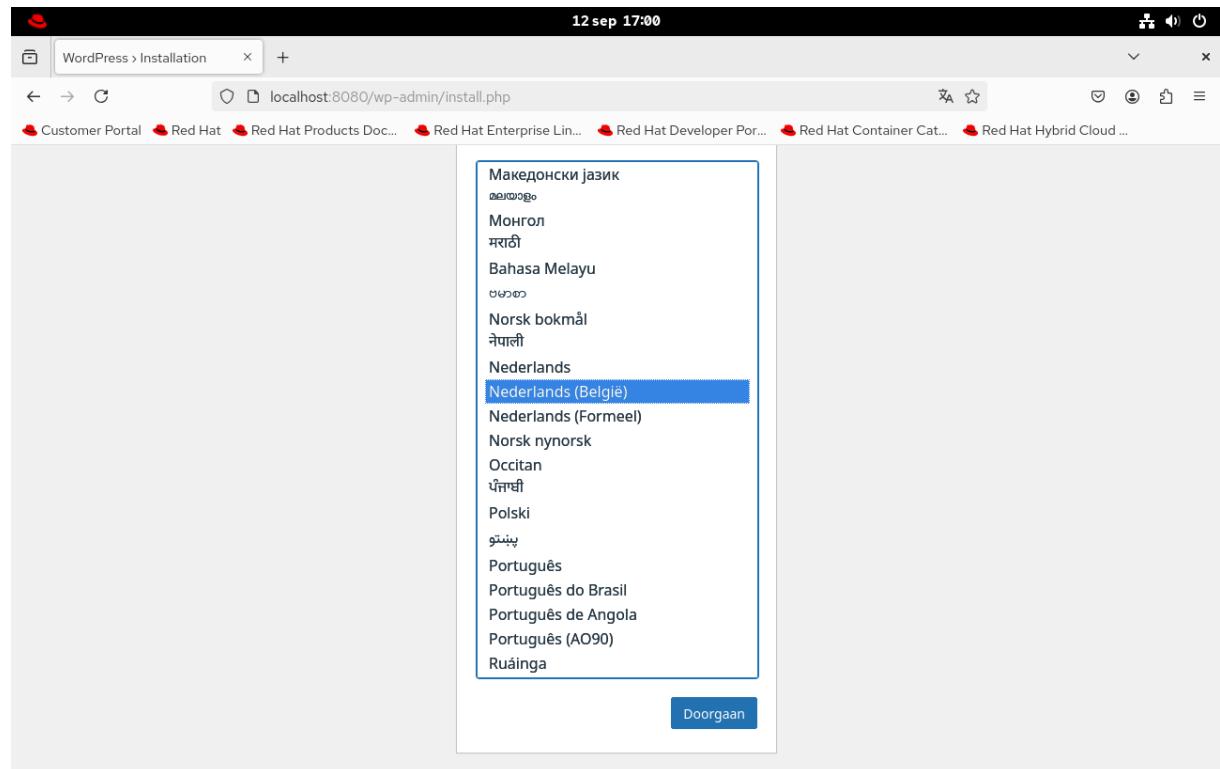
```

student@serverXX:~$ podman ps -a --pod
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES POD_ID PODNAME
1df93f41ed9ea localhost/podman-pause:5.4-175089600 mariadb 4 minutes ago Up 4 minutes 0.0.0.0:8880->80/tcp c3b3f436a2e5 wp-pod
366306c21f0a docker.io/library/mariadb:latest mariadb 4 minutes ago Up 4 minutes 0.0.0.0:8880->80/tcp, 3306/tcp wp-db c3b3f436a2e5 wp-pod
bd14ebea558e docker.io/library/wordpress:latest apache2-foreground... About a minute ago Up About a minute 0.0.0.0:8880->80/tcp wp-app c3b3f436a2e5 wp-pod
student@serverXX:~$ |

```

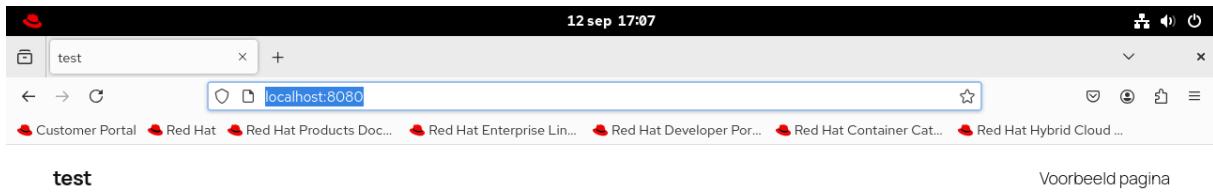
Je ziet nu dat de 3 containers draaien.

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.



Blog

Hallo wereld!

Welkom bij WordPress. Dit is je eerste bericht. Bewerk of verwijder het en start dan met schrijven!

12 september 2025

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 dezelfde 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

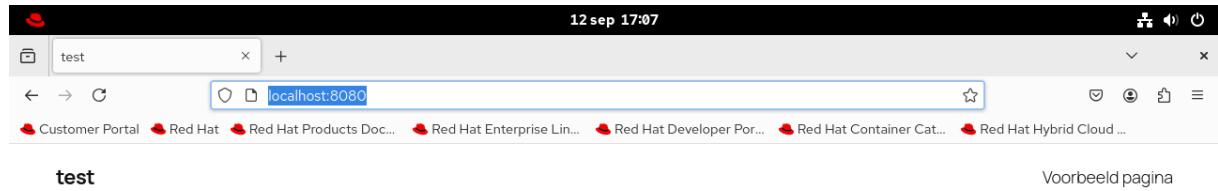
student@serverXX:~$ podman run -d --pod wp-pod2 --name wp-db -e
MYSQL_ROOT_PASSWORD="admin123" -e MYSQL_DATABASE="wordpress" -e
MYSQL_USER="wpuser" -e MYSQL_PASSWORD="user123" -v
/home/student/wpdata/db:/var/lib/mysql:Z docker.io/library/mariadb:latest
08360a21128d5d20797fe20cae0fdb019eb1806b4786ccc255c094e46addced1

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.



Blog

Hallo wereld!

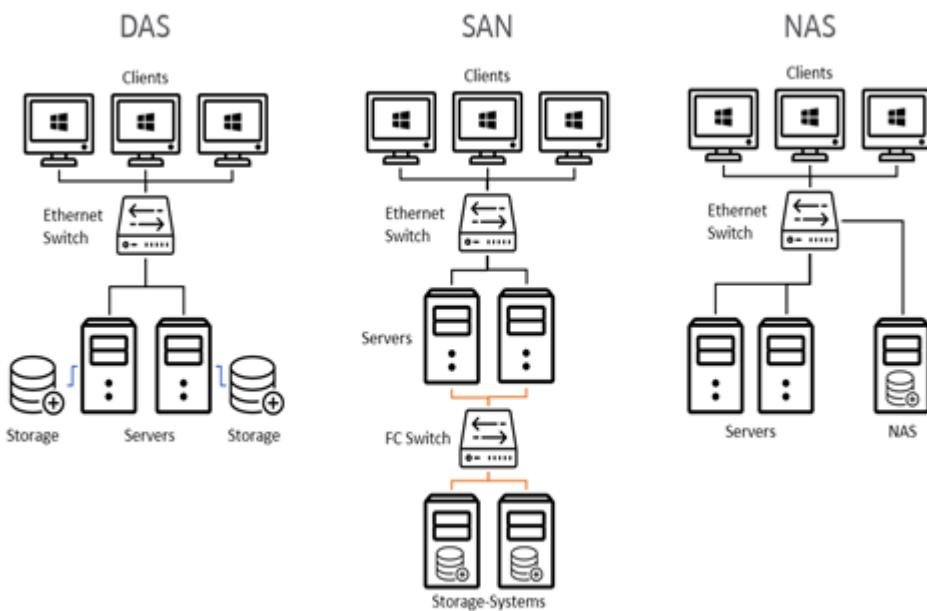
Welkom bij WordPress. Dit is je eerste bericht. Bewerk of verwijder het en start dan met schrijven!

12 september 2025

10 Storage Infrastructure

10.1 Opslagsystemen

SAN (Storage Area Network), DAS (Direct Attached Storage) en NAS (Network Attached Storage) zijn drie veelvoorkomende manieren om opslag in IT-infrastructuren te beheren en te gebruiken. Ze verschillen in hoe ze toegang bieden tot opslagbronnen en in hun typische gebruiksscenario's.



10.1.1 DAS (Direct Attached Storage):

DAS is opslag die direct is aangesloten op een server of computer, zonder een netwerk tussen de opslag en de computer. Het is het eenvoudigste type opslag en komt vaak voor in de vorm van interne harde schijven of externe opslag die via interfaces zoals SATA, SAS of USB direct met één computer is verbonden.

Voorbeelden:

Een externe harde schijf die via USB aan een computer is gekoppeld of een RAID-array die in een server is ingebouwd.

Voordelen:

- Eenvoudig op te zetten en te gebruiken.
- Lage latency omdat er geen netwerk bij betrokken is.
- Goedkoop voor kleine opstellingen.

Nadelen:

- Niet geschikt voor delen van opslag over meerdere computers.
- Schaalbaarheid is beperkt tot de fysieke capaciteit van de host.

10.1.2 SAN (Storage Area Network):

SAN is een speciaal high-performance netwerk dat meerdere servers verbindt met gecentraliseerde, block-level opslag. Het gebruikt vaak gespecialiseerde netwerken zoals Fibre Channel of iSCSI om servers directe toegang te geven tot gedeelde opslag, alsof het een lokale schijf is. SAN's worden vaak gebruikt in datacenters en grote ondernemingen voor veeleisende toepassingen zoals databases en virtualisatie.

Voorbeelden:

Een SAN-oplossing zoals die van Dell EMC, NetApp, of een systeem dat gebruik maakt van Fibre Channel-switches en gedeelde storage arrays.

Voordelen:

- Hoge prestaties door dedicated opslagnetwerken.
- Block-level toegang, wat efficiënt is voor virtualisatie en databases.
- Hoge schaalbaarheid en betrouwbaarheid, geschikt voor enterprise omgevingen.

Nadelen:

- Complex en duur om op te zetten en te beheren.
- Vereist vaak gespecialiseerde hardware/software en kennis.

10.1.3 NAS (Network Attached Storage):

NAS is een dedicated opslagapparaat dat via een standaard lokaal netwerk (LAN) verbonden is en bestanden deelt met meerdere clients (computers, servers, etc.). Het biedt bestand-gebaseerde toegang tot opslag en werkt meestal via netwerkprotocollen zoals SMB/CIFS (voor Windows), NFS (voor Linux/Unix), en FTP.

Voorbeelden:

Een NAS-apparaat zoals die van merken als Synology, QNAP of een zelfgebouwde NAS-server die via het netwerk wordt gedeeld.

Voordelen:

- Eenvoudig toegang tot gedeelde opslag voor meerdere apparaten op het netwerk.
- Goed voor het delen van bestanden binnen een thuis- of bedrijfsomgeving.
- Gemakkelijk schaalbaar door het toevoegen van schijven of nieuwe NAS-apparaten aan het netwerk.

Nadelen:

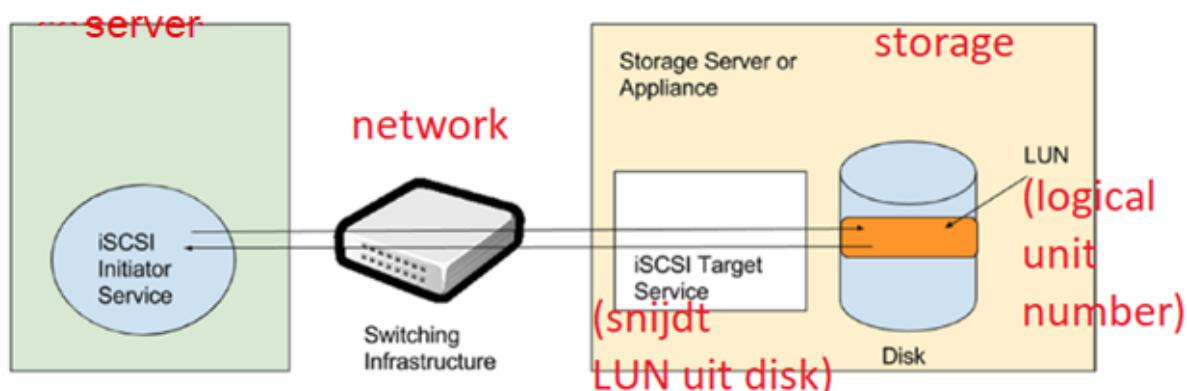
- Prestaties kunnen beperkt worden door netwerksnelheden (zoals de snelheid van je ethernetverbinding).
- Beperkt tot bestand-gebaseerde toegang, wat minder efficiënt kan zijn voor intensieve toepassingen zoals databases of virtualisatie.

10.2 iSCSI

10.2.1 Inleiding

We gaan een open source storage-oplossing implementeren met behulp van TrueNAS Community Edition, dat onder de GNU General Public License valt. Deze tool maakt gebruik van het iSCSI-protocol om opslag op een efficiënte manier te delen over een netwerk.

iSCSI gebruikt dezelfde opslagcommando's als het traditionele SCSI-protocol, maar stuurt deze via TCP/IP. Hierdoor kunnen opslagbronnen en rekenkracht eenvoudig worden gescheiden en over meerdere locaties worden verspreid. Dit zorgt voor meer flexibiliteit en schaalbaarheid in je IT-infrastructuur.



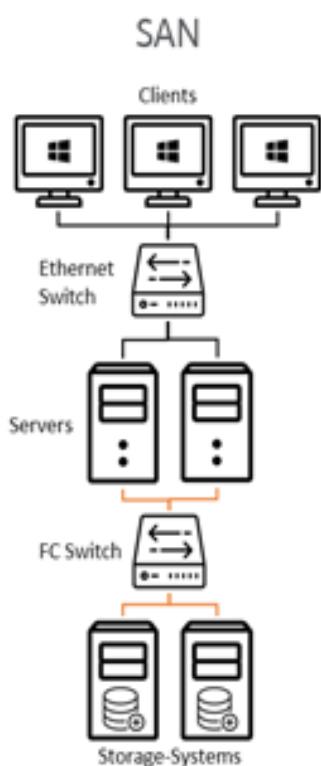
- Server met iSCSI Initiator Service (links):
 - o Dit is de machine (bijvoorbeeld een fysieke server of virtuele machine) die gebruik wil maken van opslag die extern beschikbaar is.
 - o Op deze server draait de iSCSI Initiator Service, een softwarecomponent die verantwoordelijk is voor het leggen van de netwerkverbinding met de externe opslag.
 - o De initiator stuurt via het netwerk SCSI-opdrachten naar de opslagserver, ook wel de iSCSI Target Service genoemd.
- Netwerk (midden):
 - o Tussen de server en de opslag bevindt zich het netwerk, meestal bestaande uit switches en routers.
 - o Over dit netwerk worden de data en commando's verstuurd met het TCP/IP-protocol.
 - o De verbinding loopt van de iSCSI initiator op de server naar de iSCSI target op de opslagserver.
- Opslagserver of Appliance (rechts):
 - o Deze server bevat de iSCSI Target Service, die de taak heeft om opslagbronnen aan te bieden aan clientmachines (de initiators).
 - o De target service ontvangt en verwerkt de SCSI-opdrachten, en vertaalt deze naar acties op de fysieke schijven of andere opslagmedia.
- Disk en LUN (Logical Unit Number):
De fysieke opslag op de opslagserver wordt opgedeeld in logische eenheden, zogenaamde LUN's.
 - Een LUN is een deel van een fysieke schijf dat als afzonderlijke schijf wordt gepresenteerd aan de initiator.

- Bijvoorbeeld, een LUN van 100 GB is een specifiek deel van een fysieke schijf dat de server als een zelfstandige opslag ziet.
- iSCSI Target Service (rechts, in opslagserver):
 - o Deze service maakt de LUN's vrij en beschikbaar aan de initiator.
 - o Wanneer de initiator verbinding maakt met de target, gebruikt deze de LUN alsof het een lokale schijf is, ook al bevindt de opslag zich fysiek op een andere locatie.

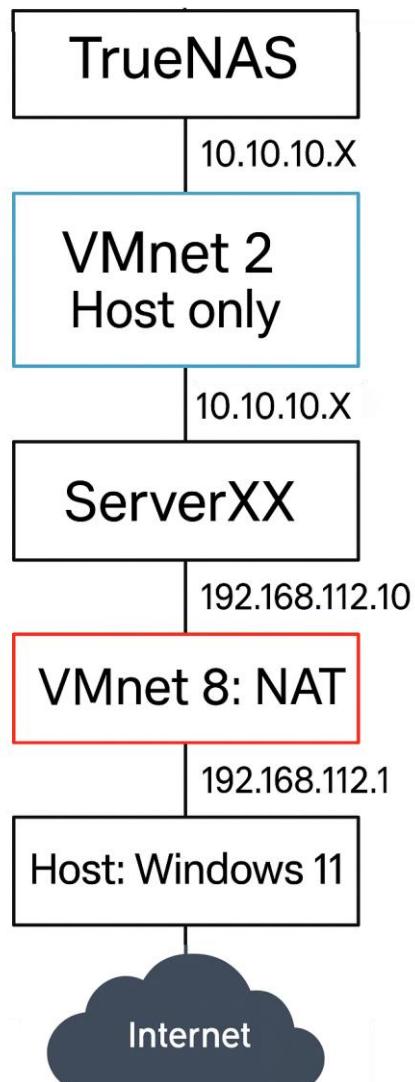
We zullen een iSCSI target configureren op TrueNAS Community Edition en deze iSCSI benaderen vanuit RHEL10. We zullen daarna met Podman gegevens hierop opslaan.

10.2.2 Toevoegen netwerk voor SAN

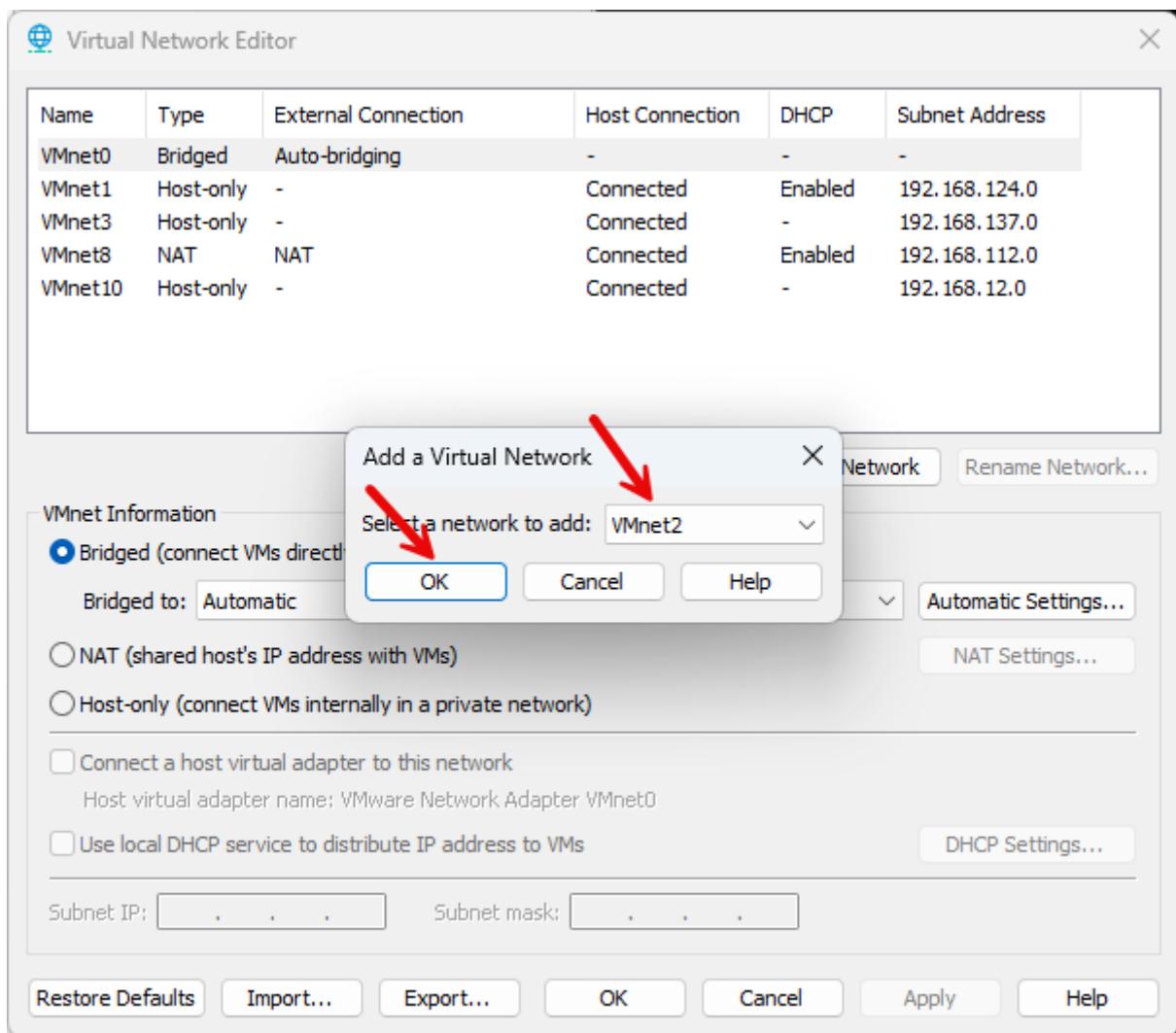
Zoals reeds aangehaald in de paragraaf over opslagsystemen zijn er 2 netwerken nodig om iSCSI correct te implementeren.



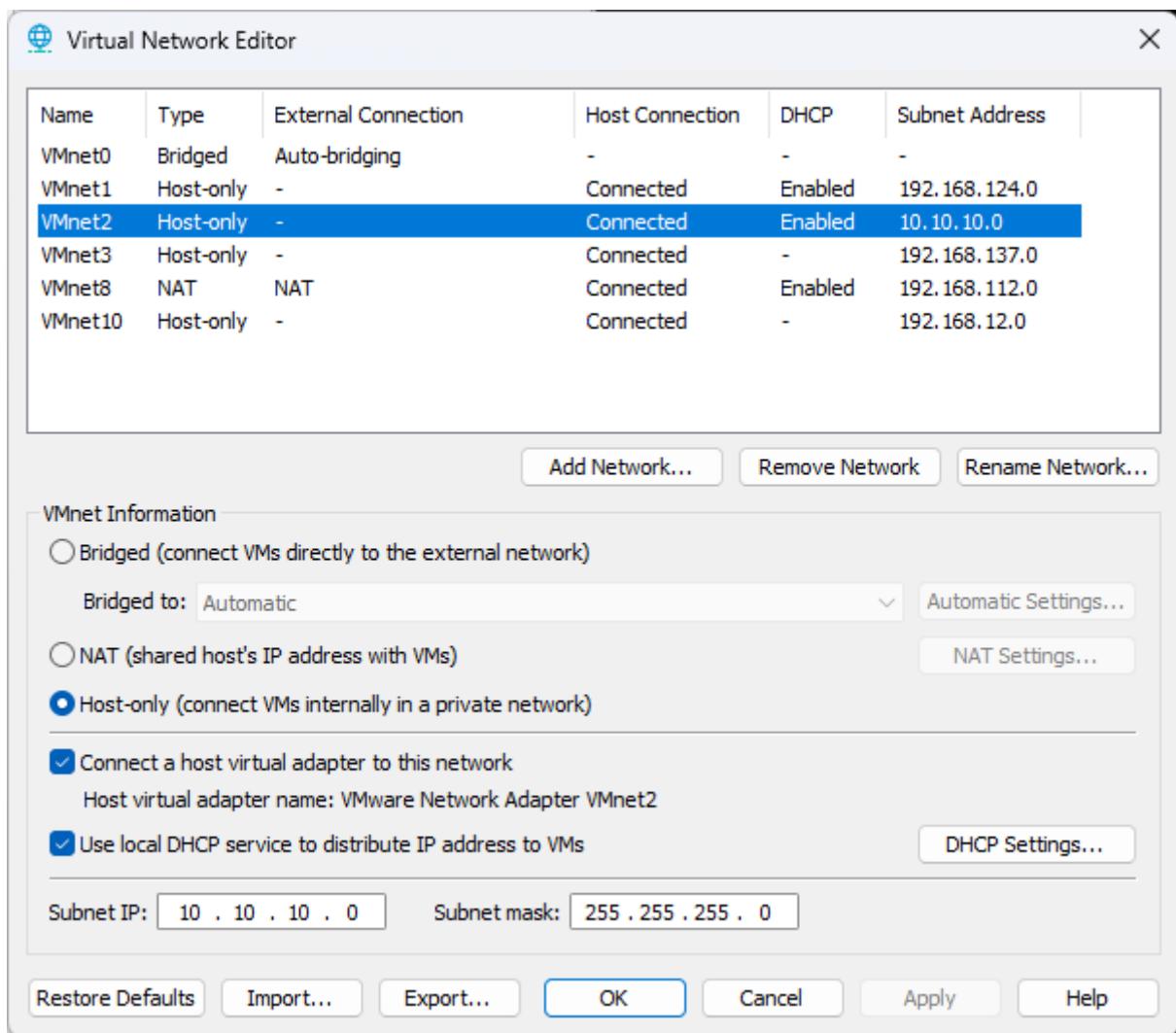
We zullen dit als volgt implementeren.



We voegen dus Vmnet2 toe aan Vmware via Edit, Virtual Network Editor. Klik erna op Add network en kies voor VMnet2.



Laat host only staan. Kies voor subnet 10.10.10.0/24 zoals hieronder staat weergegeven.



10.2.3 Installatie TrueNAS

Ga naar <https://www.truenas.com/download-truenas-community-edition/>.

Klik rechts onder op No Thank you, I have already signed up.

Klik nu op onderstaande.

TrueNAS 25.04.2.6

Previous Stable Version



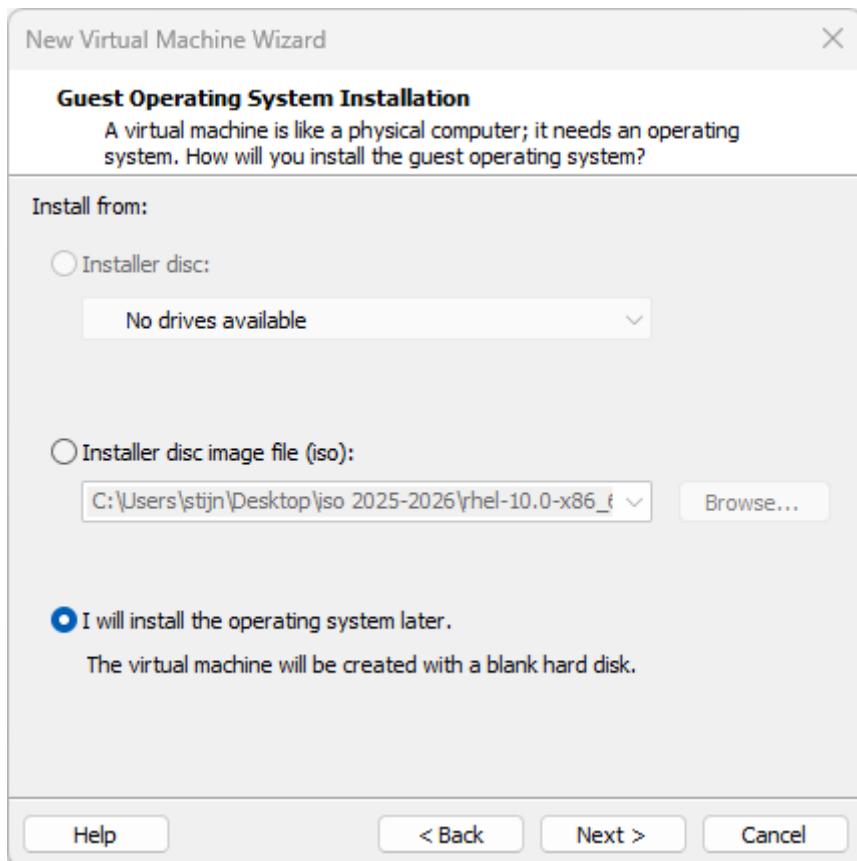
[Download LEGACY >](#)

In versie 25.10.0 zit een bug betreffende detectie van schijven. Dit zou opgelost worden bij 25.10.1.

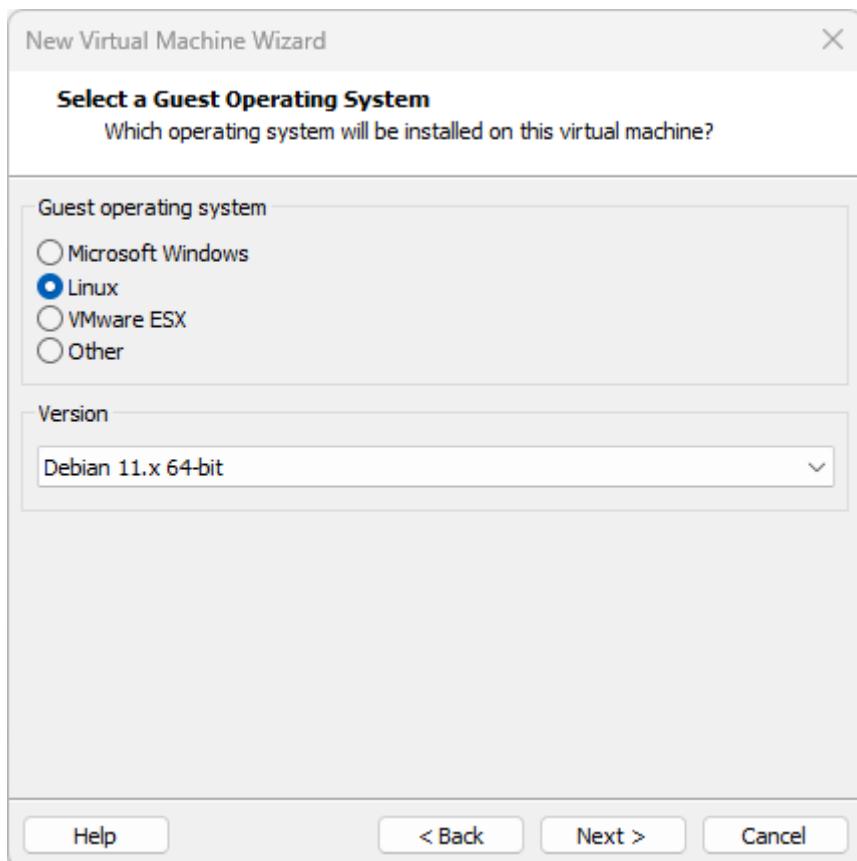
Open VMware Workstation en klik op "Create a New Virtual Machine".



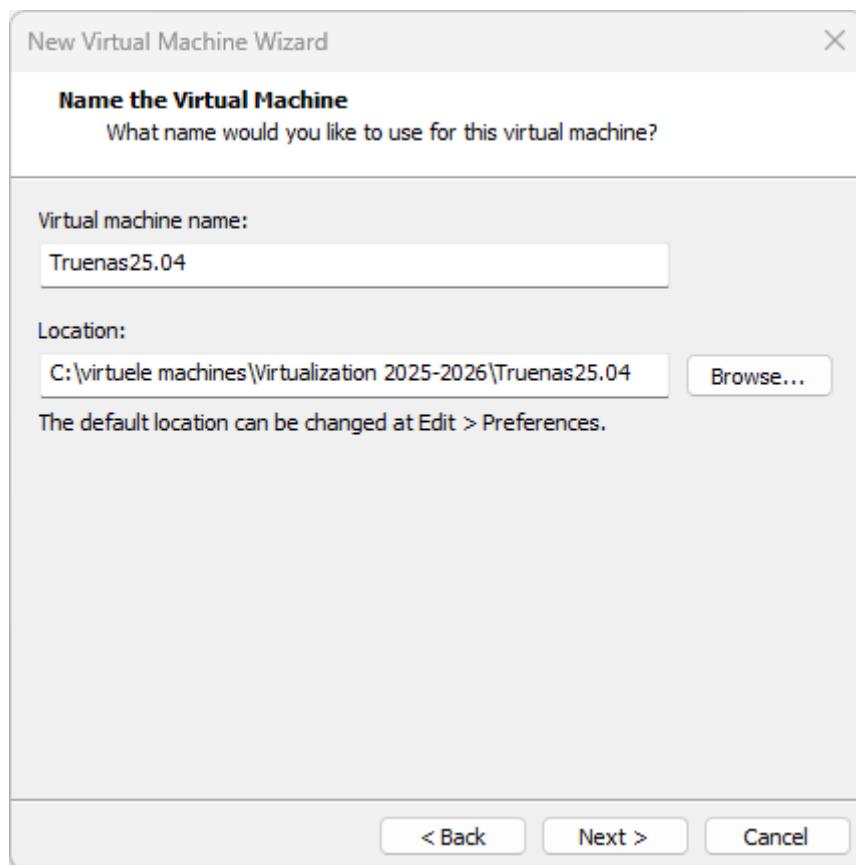
Kies nu voor de Optie "I will install the operating system later".



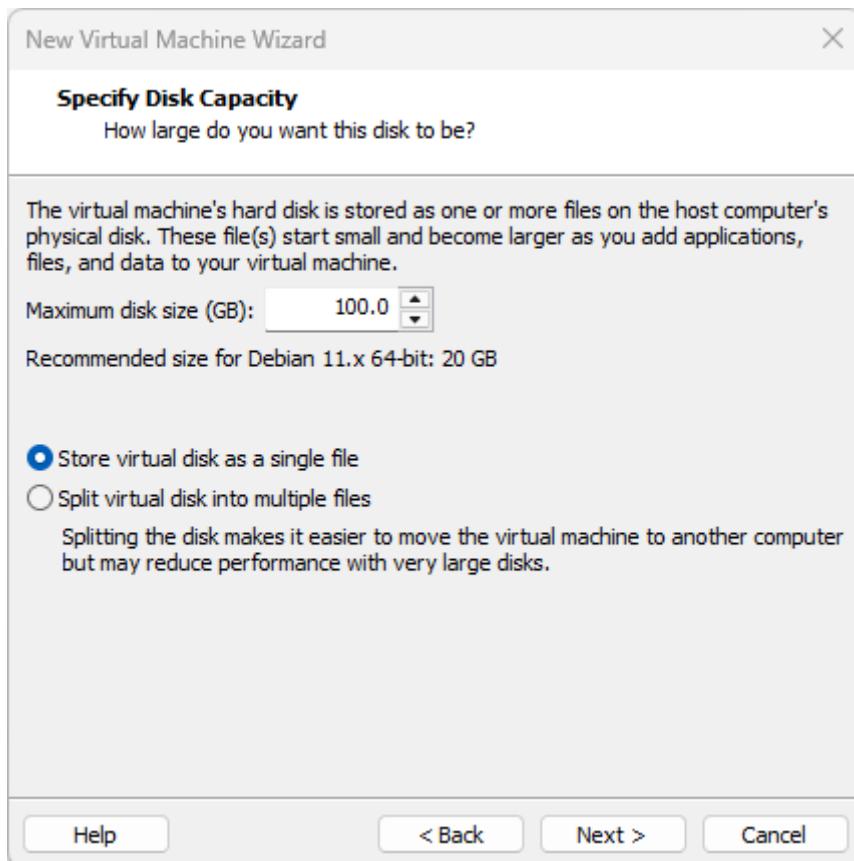
Kies nu voor Linux, Debian 11.x 64-bit.



Bepaal nu zelf waar je de VM opslaat en geef deze als naam TrueNAS 25.04.

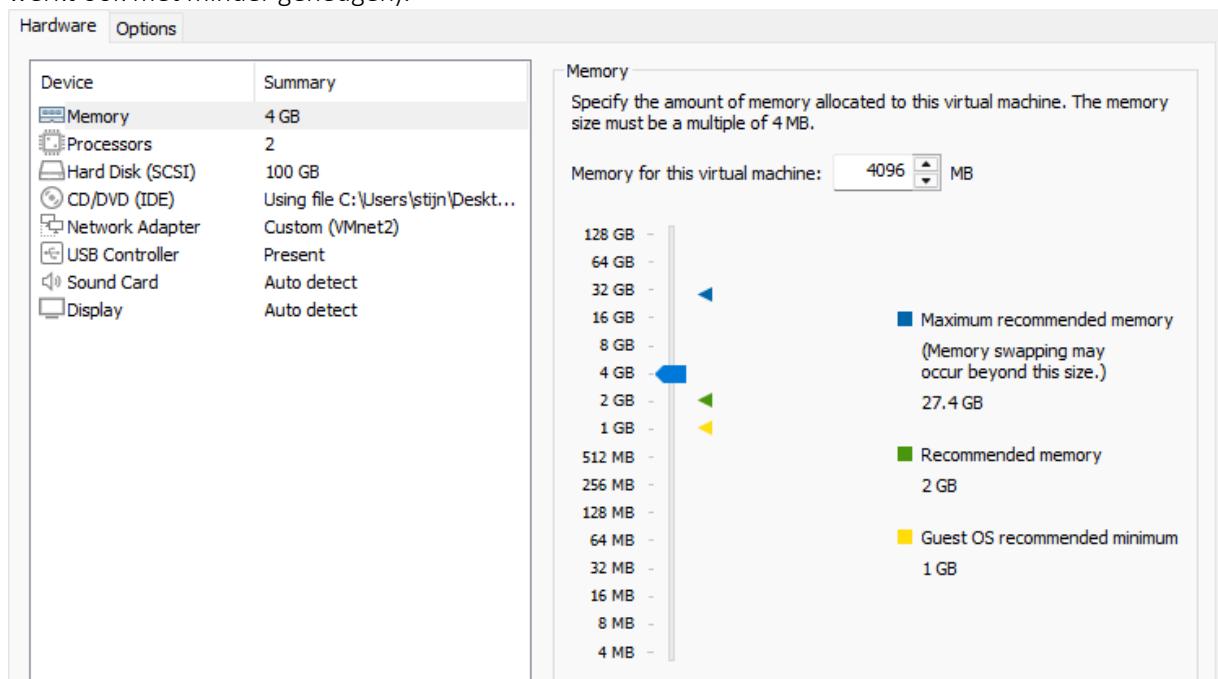


Wijs minimaal 100 GB opslagruimte toe aan de VM.



Klik op "Customize Hardware" om de hardware-instellingen te optimaliseren.

- Geheugen (RAM): We kiezen 4G (meer is beter, minimaal 8 GB wordt aangeraden maar het werkt ook met minder geheugen).



- Processors: Kies minimaal 2 cores.

The screenshot shows the 'Hardware' tab selected in the VM settings. In the 'Processors' section, the number of processors is set to 2, and the number of cores per processor is set to 1. The total processor cores are 2. Under 'Virtualization engine', the 'Virtualize Intel VT-x/EPT or AMD-V/RVI' checkbox is checked, while 'Virtualize CPU performance counters' is unchecked.

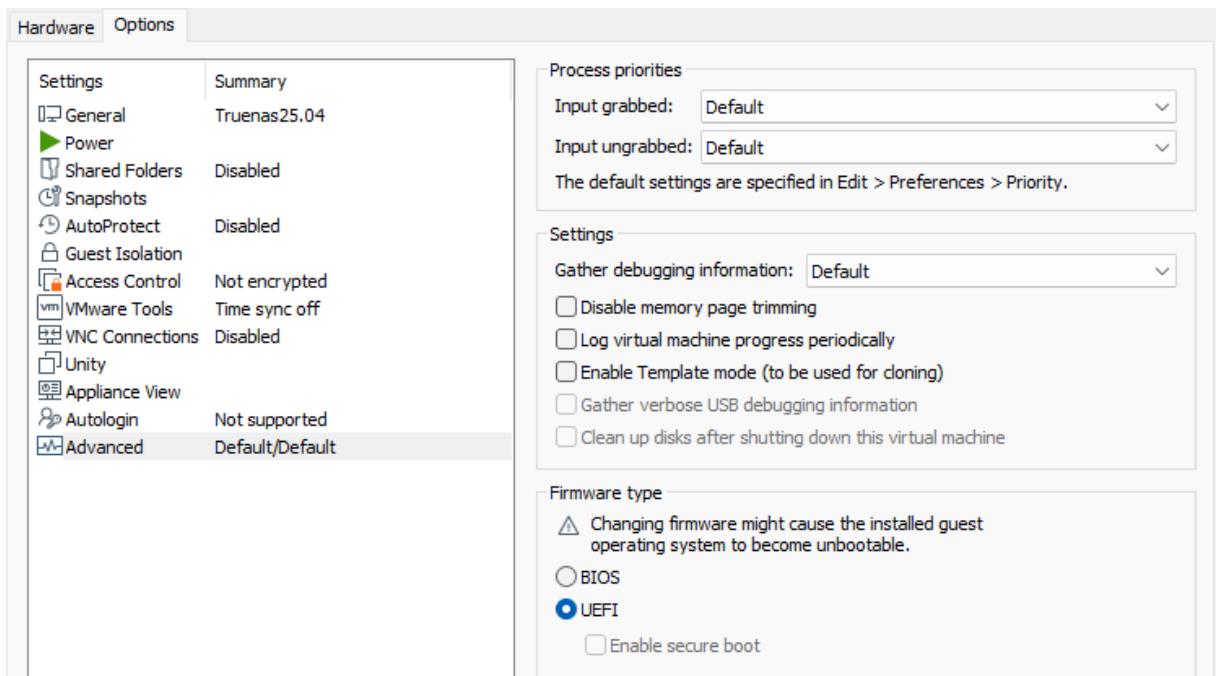
- CD/DVD (IDE): Selecteer Use ISO image file:. Klik op browse en selecteer de ISO van TrueNAS. Zet zeker vinkje voor Connect at power on.

The screenshot shows the 'Hardware' tab selected in the VM settings. In the 'Device' list, the 'CD/DVD (IDE)' entry is highlighted. Under 'Connection', the 'Use ISO image file:' option is selected, and the path 'C:\Users\stijn\Desktop\iso 2025-2026\TrueNAS-SCA.iso' is specified. The 'Connect at power on' checkbox is checked.

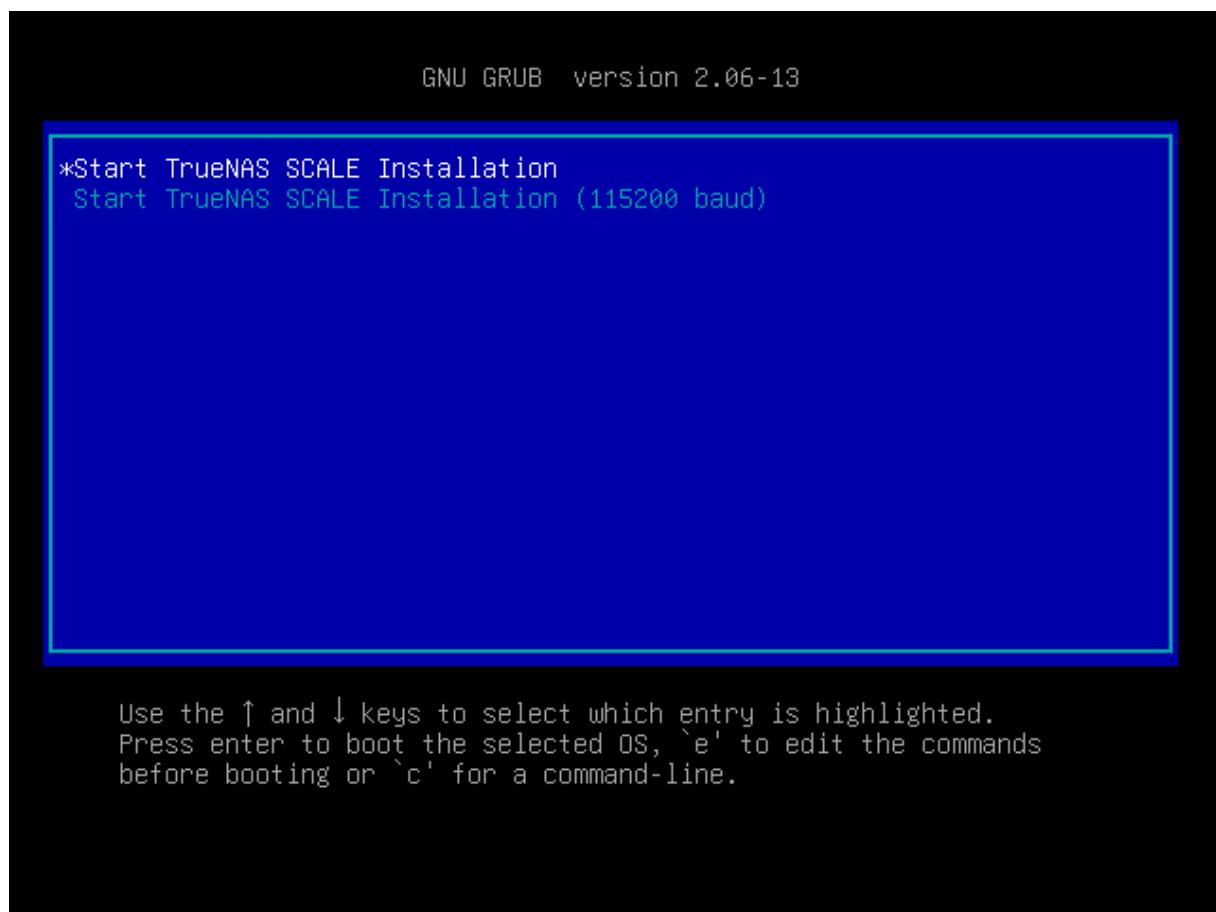
- Network Adapter: Stel de netwerkadapter in op "Custom (VMnet2)" zodat TrueNAS toegang heeft tot ServerXX.

The screenshot shows the 'Hardware' tab selected in the VM settings. In the 'Device' list, the 'Network Adapter' entry is highlighted and set to 'Custom (VMnet2)'. Under 'Network connection', the 'Custom: Specific virtual network' option is selected, with 'VMnet2' chosen from the dropdown. Other options like 'Bridged', 'NAT', and 'Host-only' are also shown.

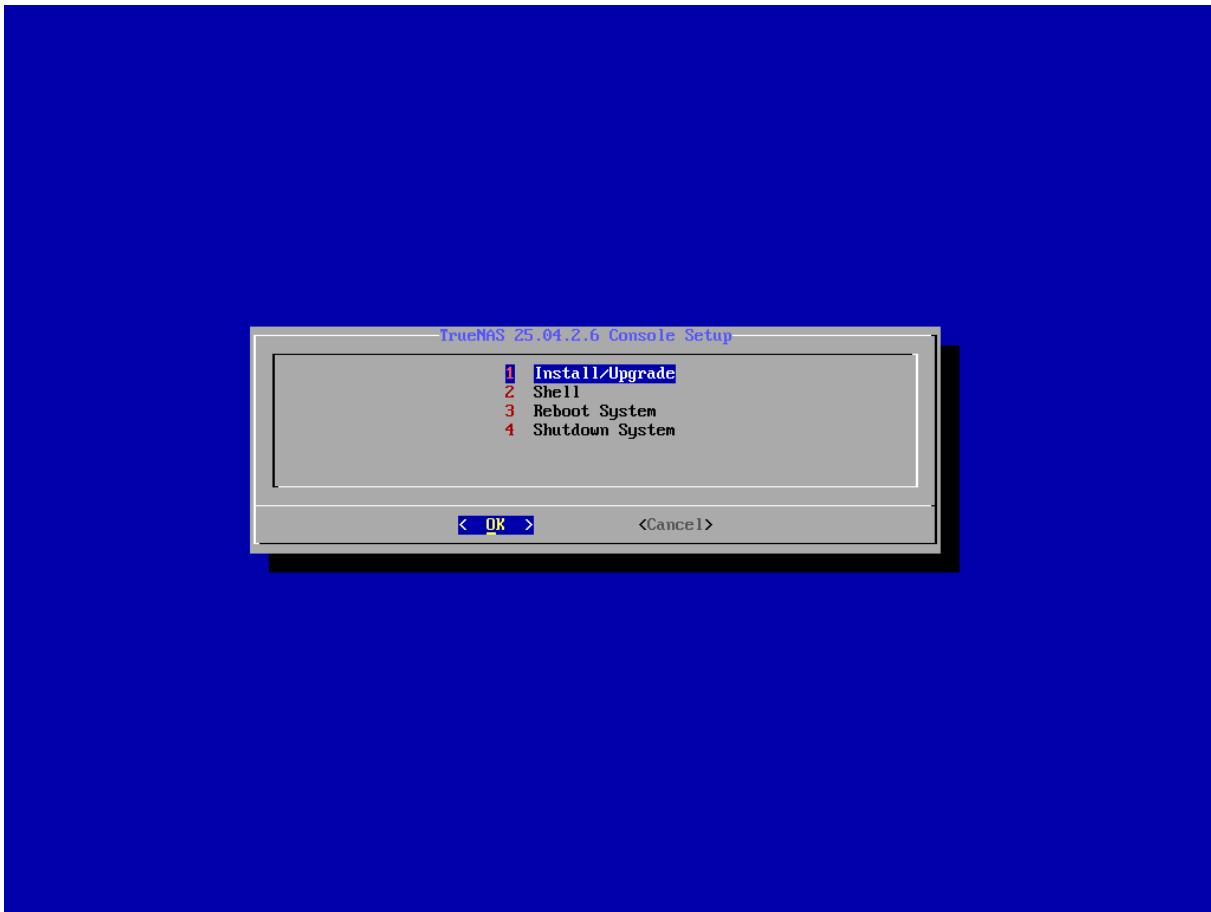
- Klik Close, Finish.
- Klik op Edit virtual machine settings.
- Op het tabblad Options, Advanced kies je bij Firmware type voor UEFI.



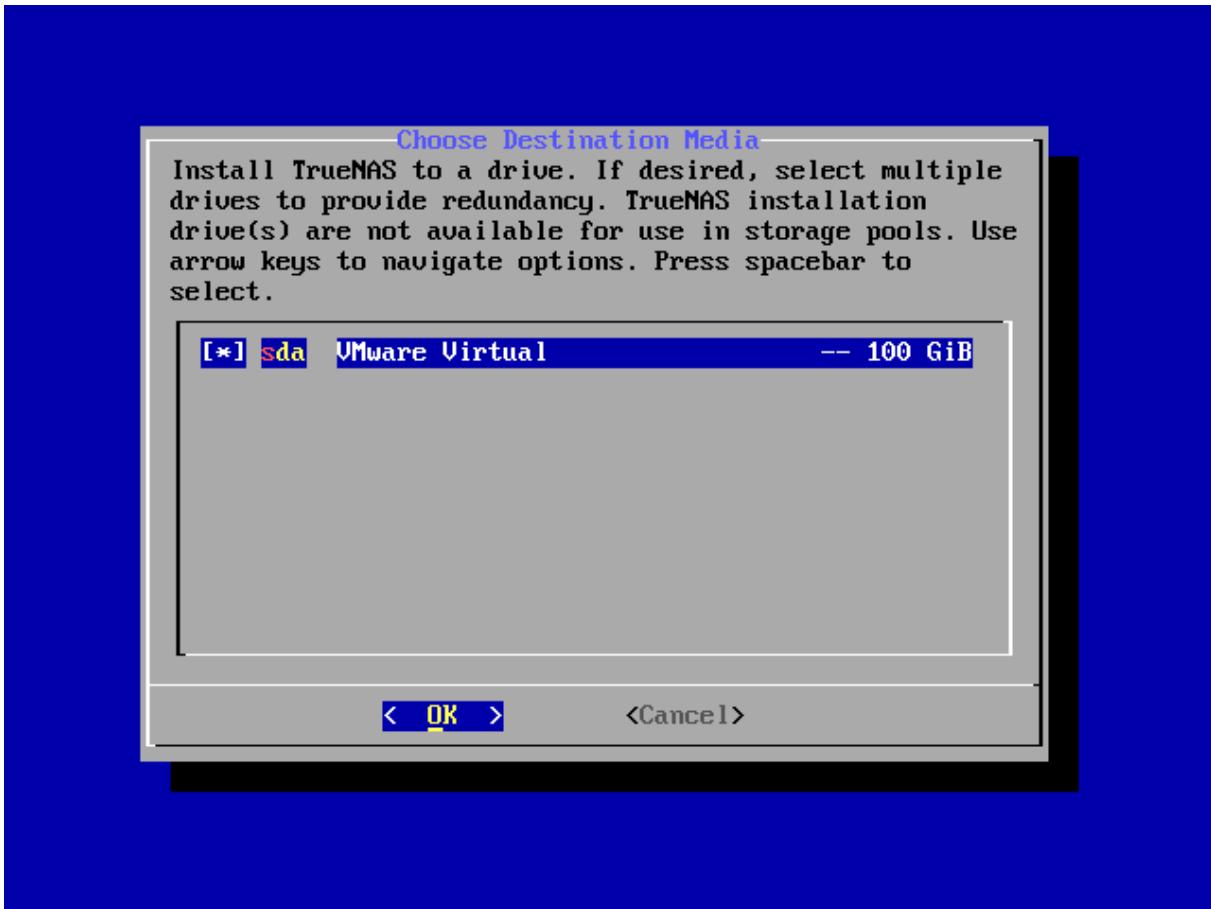
- Klik op OK om de hardwareconfiguratie op te slaan.
Start de VM op. Kies voor Start TrueNAS Installation en druk <Enter>.



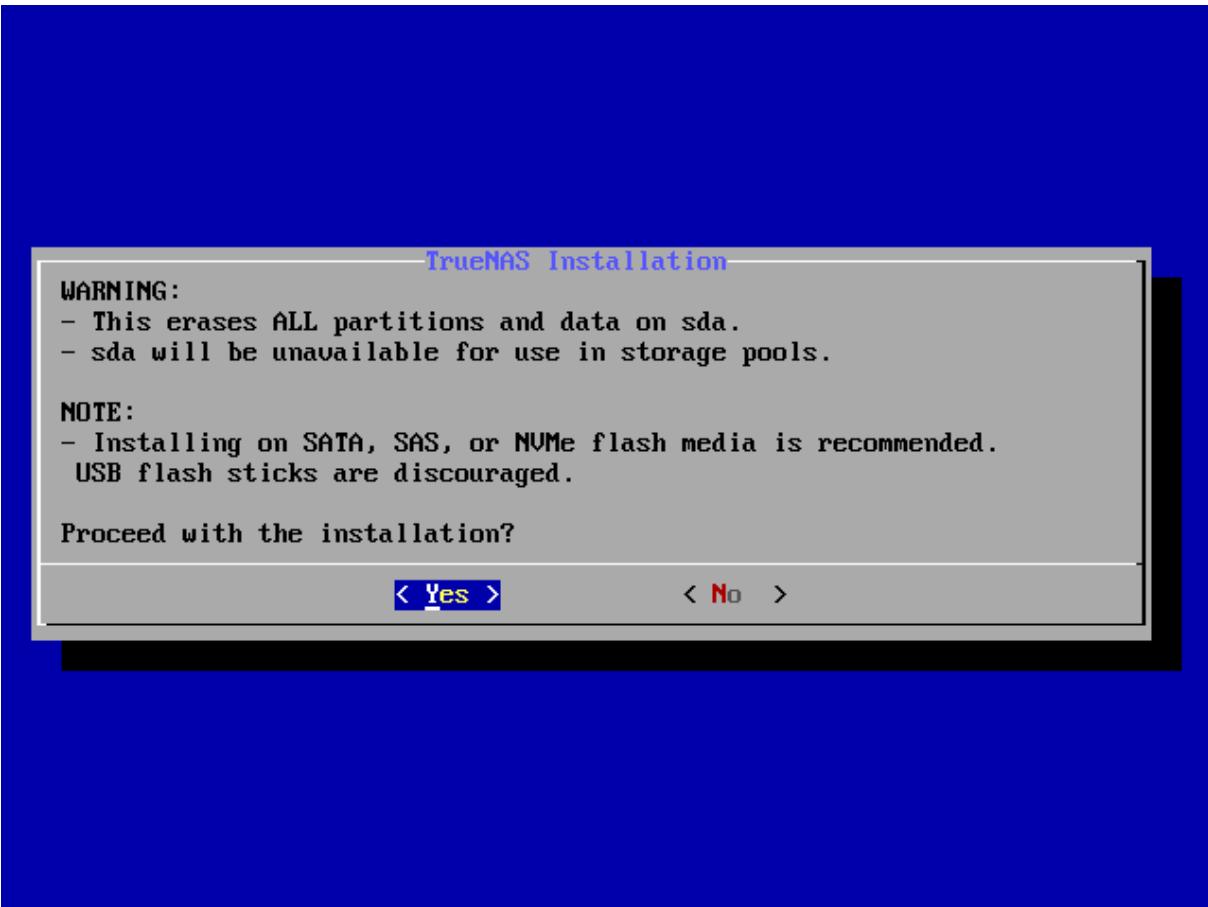
Kies in onderstaand venster voor 1 en druk <Enter>.



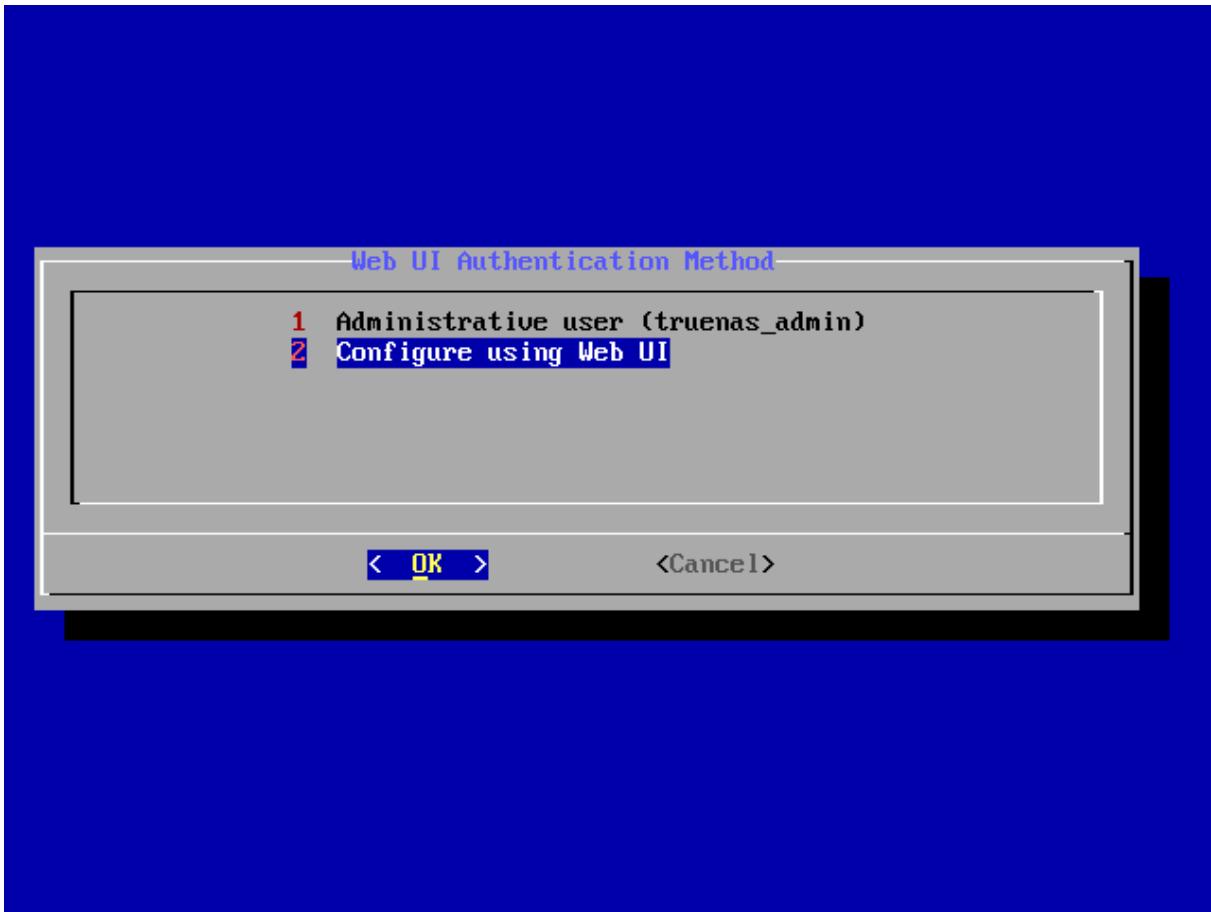
Selecteer de virtuele schijf die je hebt ingesteld als installatieschijf (dit is de schijf waar TrueNAS op zal worden geïnstalleerd). Selecteer de schijf met de spatiebalk en klik op OK door op <enter> te klikken.



Je krijgt een melding alle partities zullen verwijderd worden van sda. Dat is nu ok uiteraard. Druk op <Enter>.

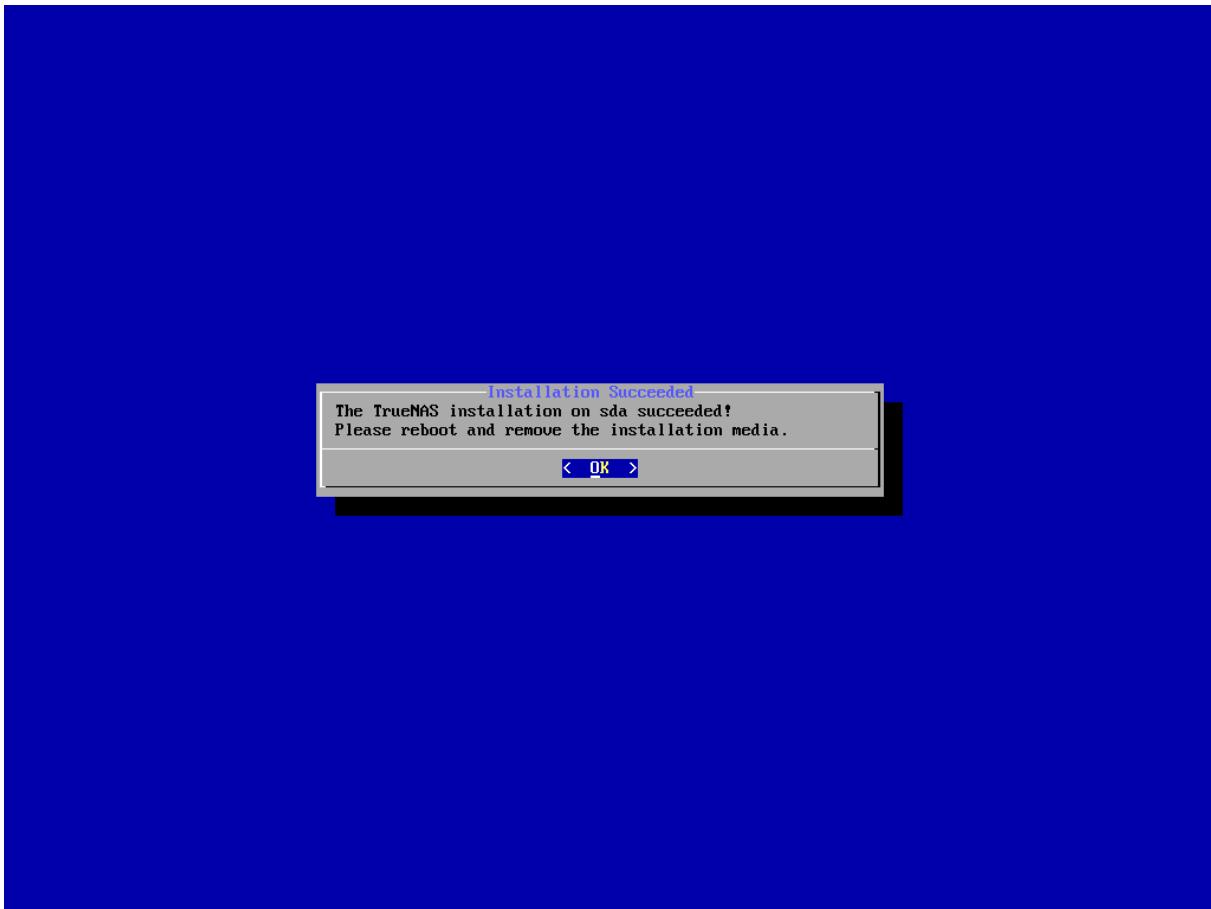


Kies bij Authentication Method voor 2 en druk <Enter>.

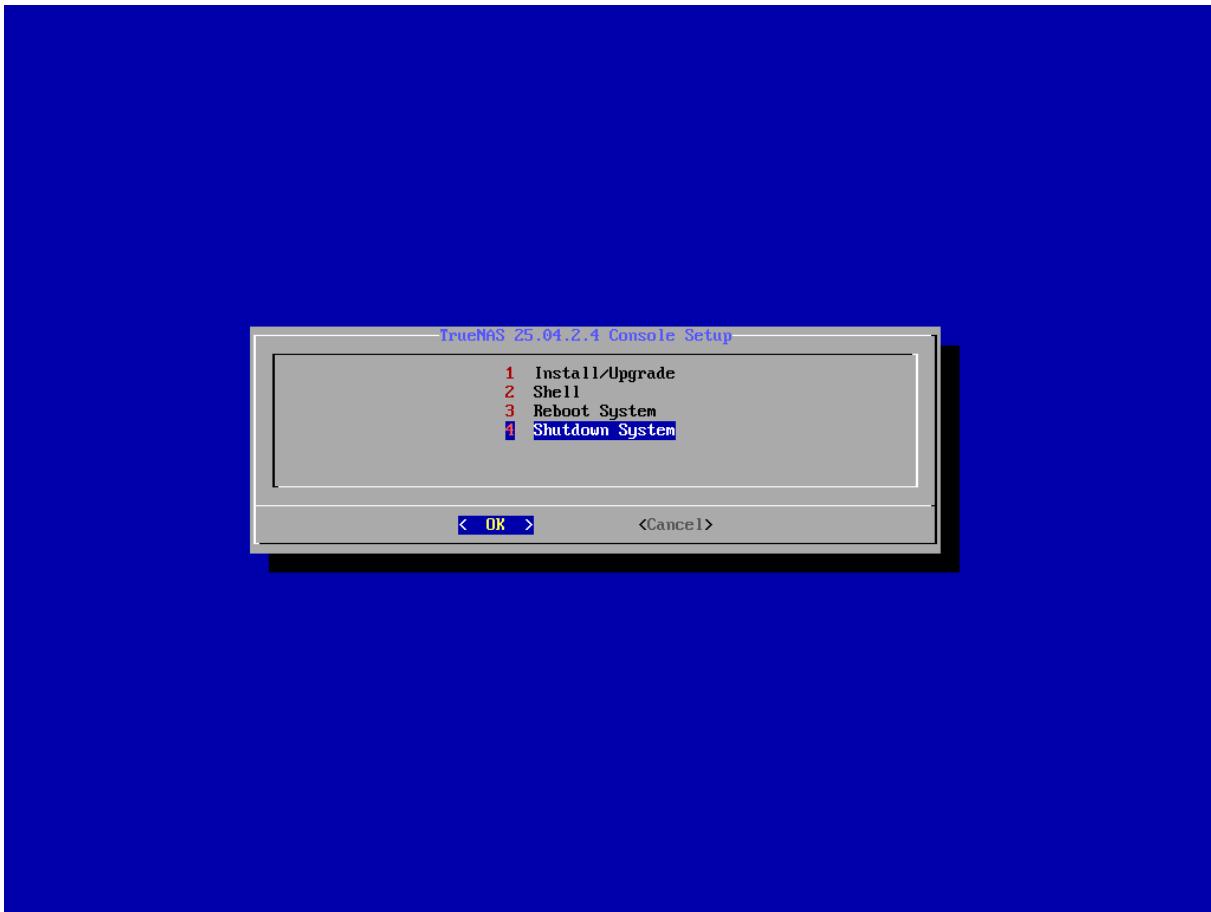


De installatie gaat nu van start.

Na installatie klik je weer OK.



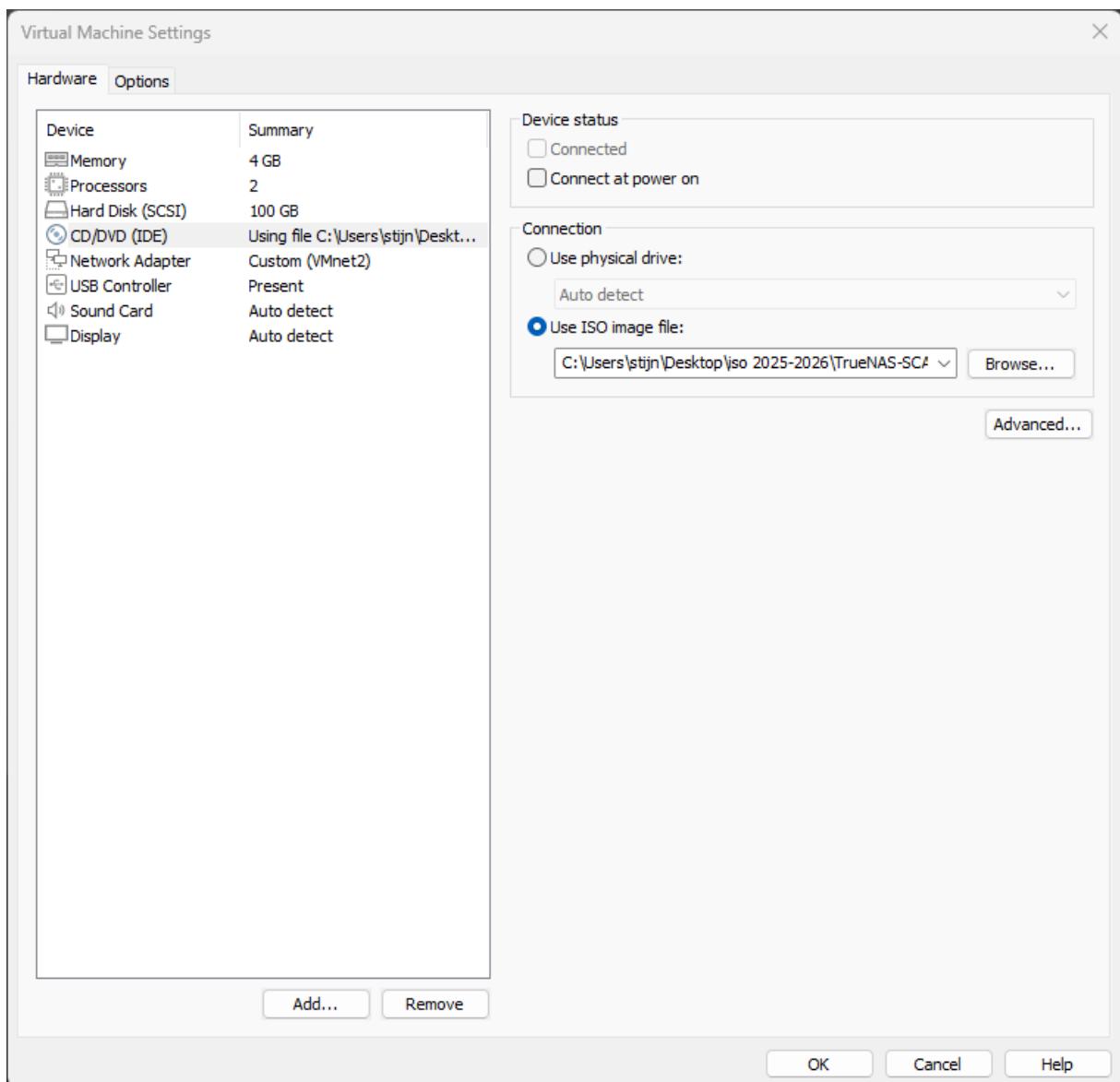
Kies nu voor nummer 4 en klik <Enter>.



We gaan nu de virtuele CD/DVD deactiveren. Die heb je niet meer nodig en je wil er niet meer van opstarten.

Ga naar virtual machine settings en selecteer links CD/DVD (IDE).

Verwijder het vinkje voor “Connect at power on”.



We voegen ook 2 SCSI-schijven van 200 GB toe door op Add te klikken. Dat wijst zich uit.

uiteindelijk krijg je volgende instellingen.

Device	Summary
Memory	4 GB
Processors	2
Hard Disk (SCSI)	100 GB
New Hard Disk (SCSI)	200 GB
New Hard Disk (SCSI)	200 GB
CD/DVD (IDE)	Using file C:\Users\stijn\Desktop\...
Network Adapter	Custom (VMnet2)
USB Controller	Present
Sound Card	Auto detect
Display	Auto detect

Start TrueNAS op.

De netwerkkaart van TrueNAS zal een IP-adres verkrijgen via DHCP op dit moment.

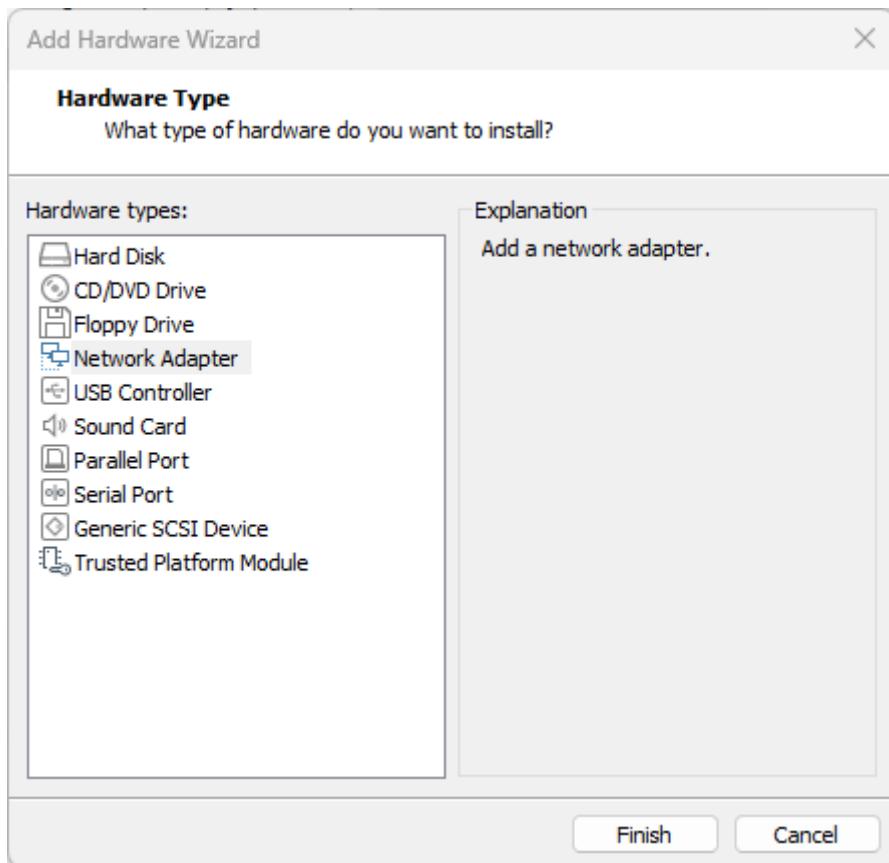
Noteer het IP-adres dat op het TrueNAS-console wordt weergegeven.

```
Console setup
-----
The web user interface is at:
http://10.10.10.128
https://10.10.10.128

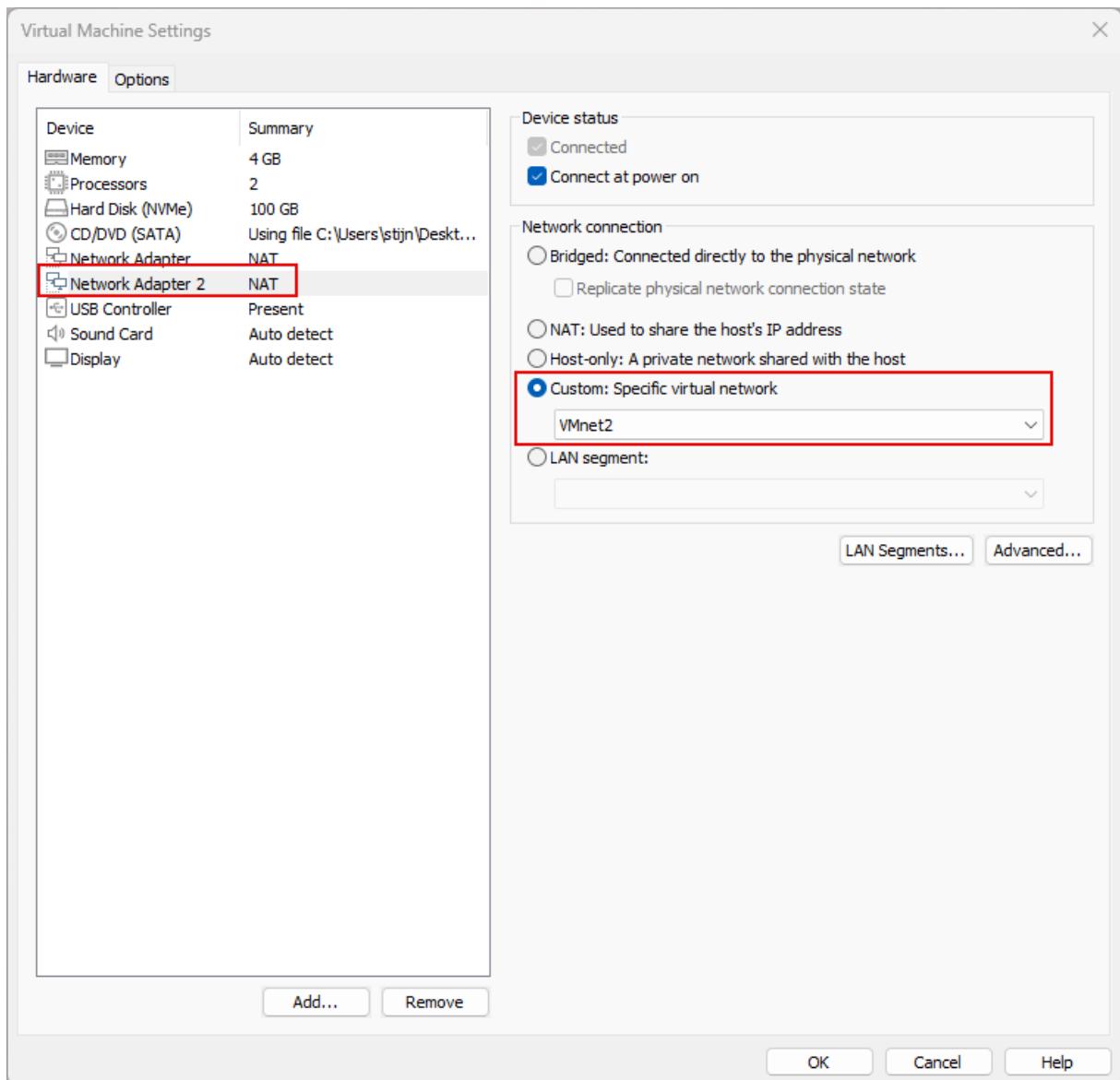
1) Configure network interfaces
2) Configure network settings
3) Configure static routes
4) Set up local administrator
5) Reset configuration to defaults
6) Open TrueNAS CLI Shell
7) Open Linux Shell
8) Reboot
9) Shutdown

Enter an option from 1-9: _
```

Voeg nu een tweede netwerkkaart toe aan ServerXX via VM, Settings. Klik op Add... , Network Adapter. Klik op Finish.



Verander nu de instellingen van de toegevoegde netwerkkaart zodat deze gebruik maakt van Vmnet2.



We dienen nu op ServerXX de tweede netwerkkaart te verbinden.

```
student@serverXX:~$ nmcli
...
ens224: niet verbonden
    "VMware VMXNET3"
    ethernet (vmxnet3), 00:0C:29:99:B5:46, hw, mtu 1500
...
```

We connecteren met de netwerkinterface ens224.

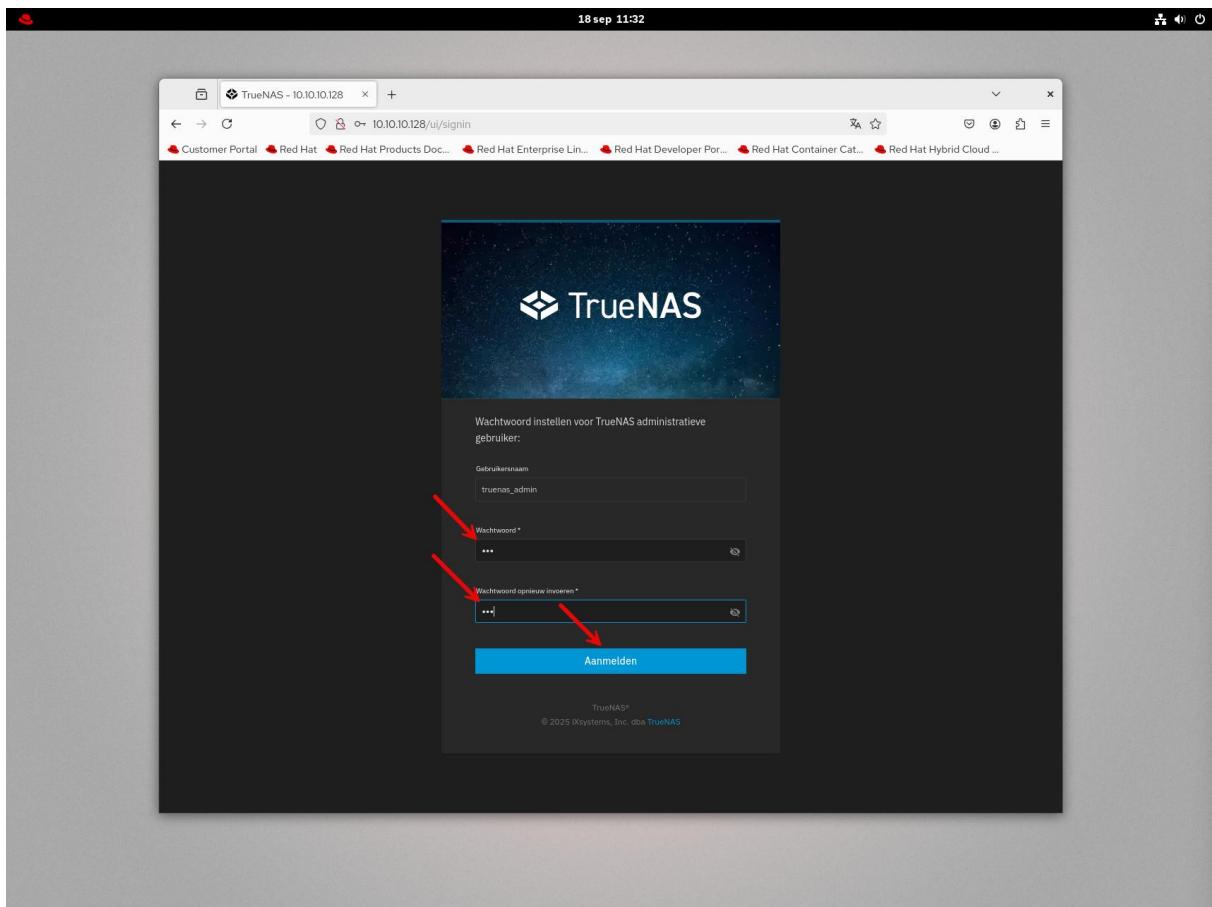
```
student@serverXX:~$ sudo nmcli device connect ens224
```

We vragen nu info op over ens224.

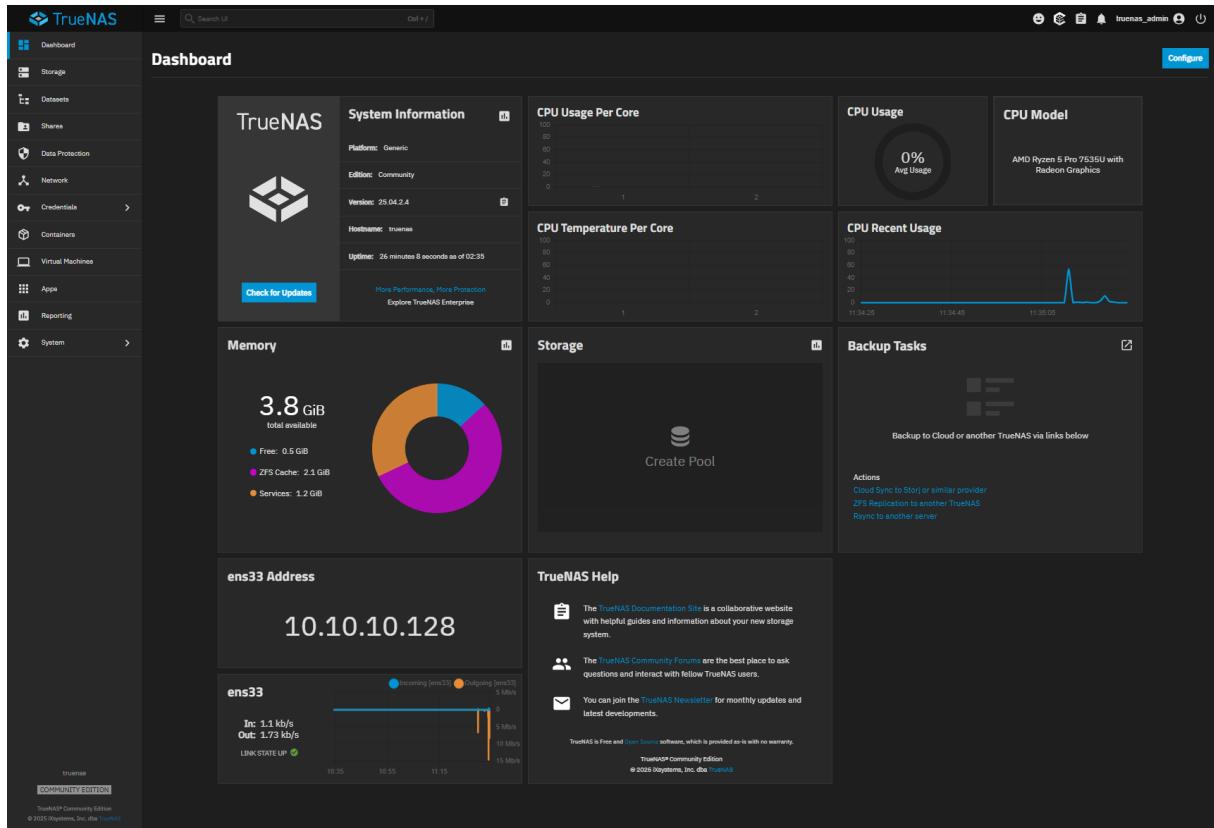
```
student@serverXX:~$ nmcli device show ens224 | grep IP4
IP4.ADDRESS[1]:           10.10.10.129/24
IP4.GATEWAY:              --
IP4.ROUTE[1]:             dst = 10.10.10.0/24, nh = 0.0.0.0, mt = 101
IP4.DNS[1]:                10.10.10.1
IP4.DOMAIN[1]:             localdomain
```

Open een webbrowser op ServerXX (of op je host als je een virtuele netwerkkaart voor VMnet2 hebt toegevoegd voor de host) en voer het IP-adres in dat Truenas heeft gekregen, in dit geval 10.10.10.128.

We stellen nu een wachtwoord in voor de gebruiker truenas_admin. Aangezien het een testomgeving is kies ik voor 123 en klik op Aanmelden.



Je bekomt nu onderstaand dashboard.

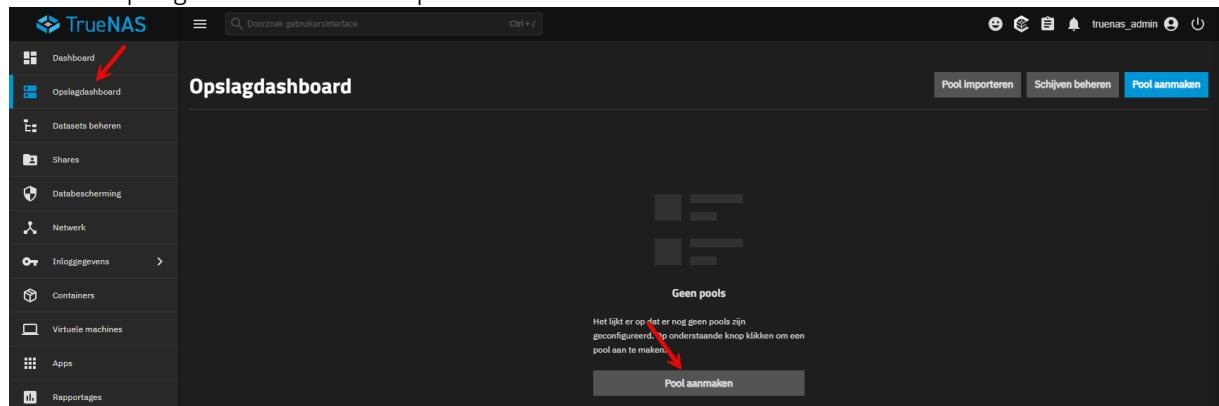


10.2.4 iSCSI instellen

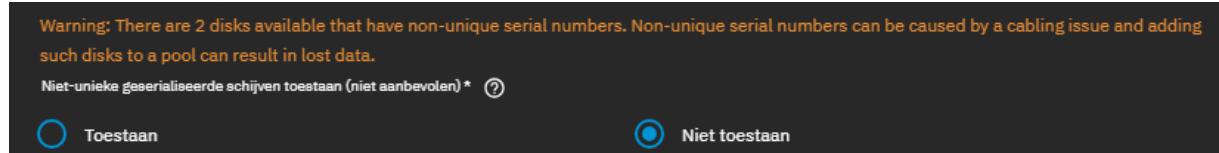
10.2.4.1 Opslagpool aanmaken

Voordat je een iSCSI-opslagbron kunt configureren, moet je een opslagpool aanmaken.

- Ga naar Opslagdashboard en klik op Pool aanmaken.



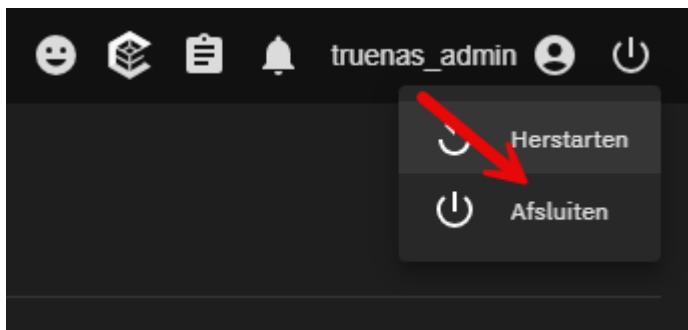
- Klik op Pool aanmaken
- Je ziet dat er een probleem is...



Dit moet je aanpassen in VMware zelf.

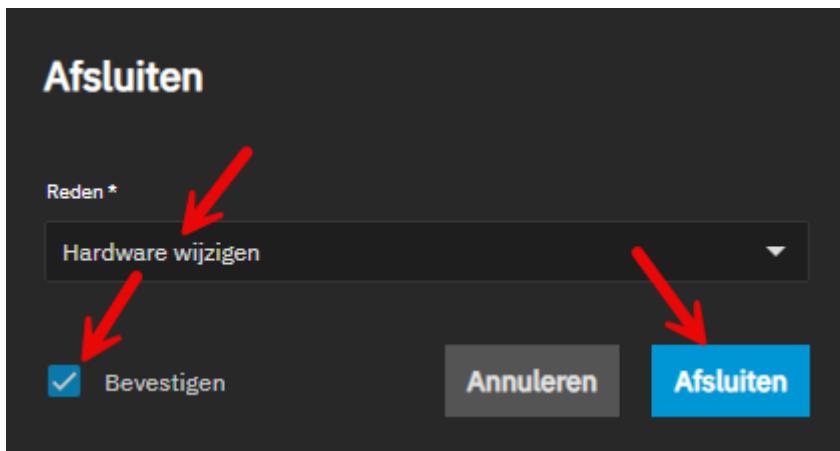
We moeten TrueNAS hiervoor eerst afsluiten.

Opgelet: gebruik Chrome of een andere moderne browser... Anders zie je niet alles goed gepositioneerd op het scherm.

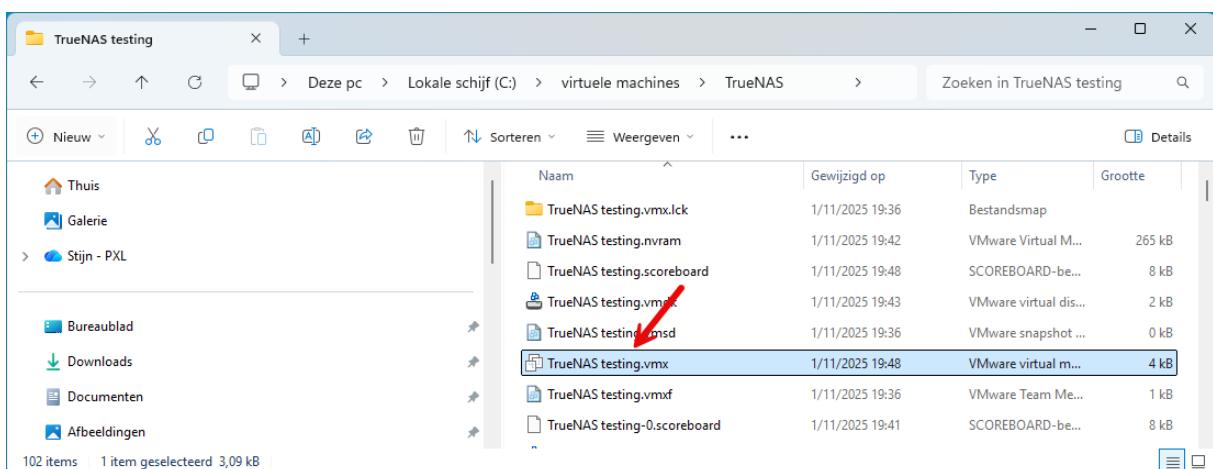


Je moet een reden opgeven... Kies voor hardware wijzigen.

Zet een vinkje voor Bevestigen en kies voor Afsluiten.



Ga op zoek naar het .vmx-bestand van TrueNAS op je host.



Voeg volgende regel toe met de kladblok of een andere teksteditor.

disk.EnableUUID = "true"

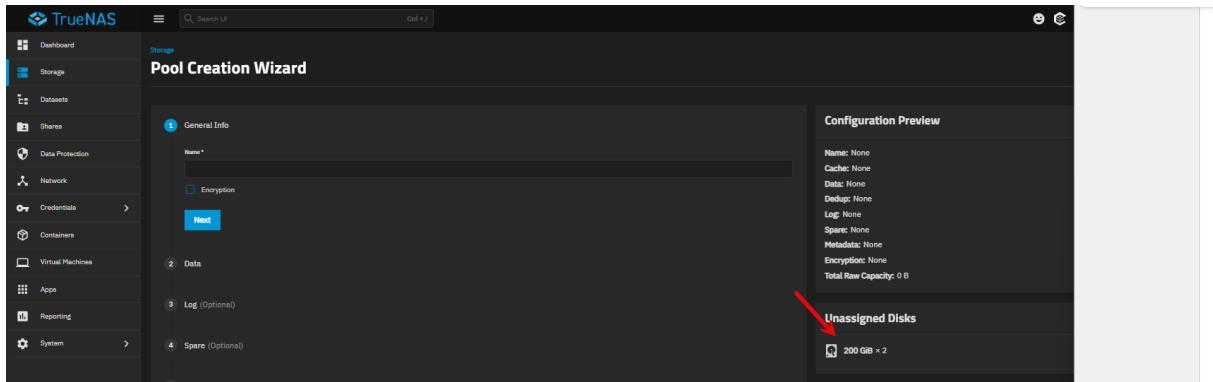


```
Truenas25.04.vmdk
Bestand Bewerken Weergeven
viv_chsbase = "true"
firmware = "efi"
ide1:0.startConnected = "FALSE"
scs1:0:1.fileName = "Truenas25.04-0-000001.vmdk"
scs1:0:1.present = "TRUE"
scs1:0:2.fileName = "Truenas25.04-1-000001.vmdk"
scs1:0:2.present = "TRUE"
scs1:0:1.redo = ""
scs1:0:2.redo = ""
usb:0.present = "TRUE"
usb:0.deviceType = "hid"
usb:0.port = "0"
usb:0.parent = "-1"
svga.guestBackedPrimaryAware = "TRUE"
guestInfo.detailedData = "architecture='X86' bitness='64' distroAddlVersion='12 (bookworm)' distroName='Debian GNU/Linux' distroVersion='12' familyName='Linux' kernelVersion='6.12.15-production+truenas' prettyName='Debian GNU/Linux 12 (bookworm)'"  
disk.EnableUUID = "true"

Ln 99, Col 1 3:155 tekens Tekst zonder opmaak 100% Windows (CRLF) UTF-8
```

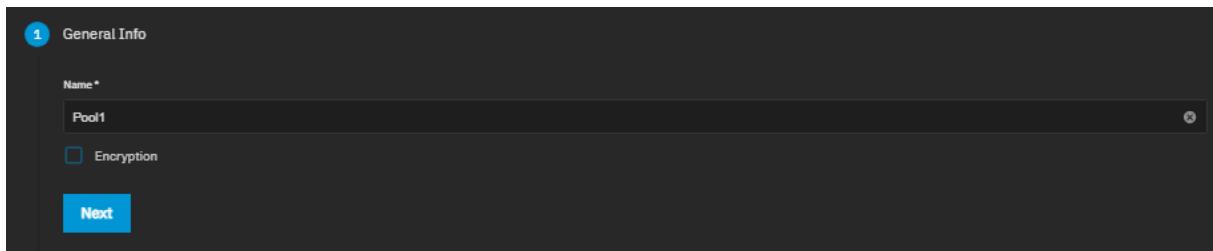
Start de VM terug op.

Ga terug naar Storage, Create pool. Je zal zien dat er nu geen foutmelding meer is.



Typ voor de naam Pool1. Je kan encryptie instellen op poolniveau, wat in de praktijk inhoudt dat de root-dataset (en alle nieuwe datasets/volumes die erin aangemaakt worden) automatisch encryptie gebruiken.

Klik Next.



Je bepaalt nu hoe de disks gecombineerd worden voor opslag, snelheid en redundantie.

Je ziet hier veel mogelijkheden:

- Stripe
Combineert alle disks tot één grote, snelle opslag. Géén redundantie: als één disk uitvalt,

verlies je alles.

Voordeel: snel, hele opslagruimte bruikbaar.

Nadeel: geen bescherming tegen uitval.

- Mirror

Elk blok data wordt naar twee (of meer) disks tegelijk geschreven. Als één disk uitvalt, blijft je data intact.

Voordeel: hoge veiligheid, eenvoudige recovery.

Nadeel: de helft (bij 2 disks) van je opslag verliest je aan "spiegeling".

- RAIDZ1

Vergelijkbaar met RAID5: één disk mag uitvallen zonder dataverlies. Data en pariteitsinformatie worden verspreid.

Voordeel: iets meer capaciteit over dan mirror, basisbescherming.

Nadeel: trager bij kleine schrijfacties, maar sommige risico's bij helaas twee falende disks tegelijk.

- RAIDZ2

Vergelijkbaar met RAID6: twee disks mogen tegelijk falen.

Voordeel: veel veiligere bescherming dan RAIDZ1.

Nadeel: iets minder bruikbare capaciteit, nog iets trager.

- RAIDZ3

Drie disks mogen tegelijk falen.

Voordeel: maximale bescherming.

Nadeel: meeste capaciteit kwijt, prestaties vooral bij kleine IO lager.

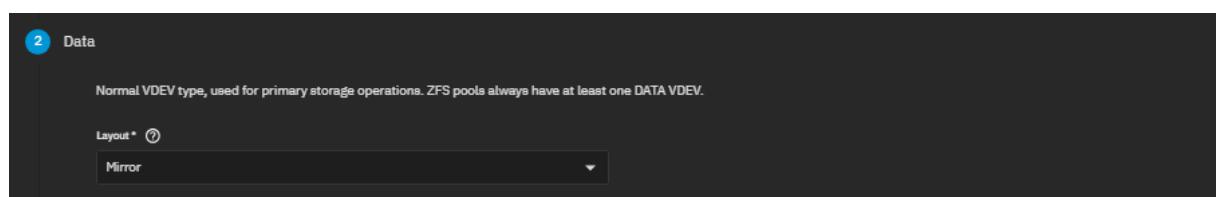
- dRAID1, dRAID2, dRAID3

Dit zijn gedistribueerde RAID-varianten, nieuw in recentere ZFS/TrueNAS versies, geoptimaliseerd voor zeer grote aantallen drives.

Voordeel: Sneller rebuild bij uitval, geschikt voor grote installaties.

Nadeel: Complexer, vaak alleen relevant bij tientallen schijven.

Wij kiezen voor Mirror.



De schijven zijn al geselecteerd.

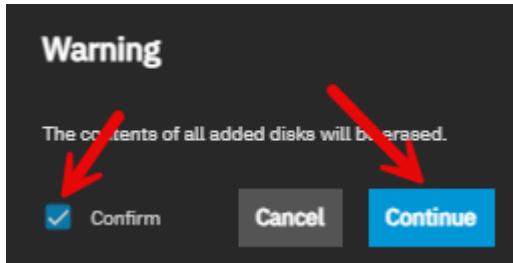
Kies voor Save And Go To Review (de andere instellingen zijn optioneel).



Klik erna op Create Pool.



Je krijgt de melding dat de contents van de toegevoegde schijven zal worden verwijderd. Bevestig dit en klik op Create Pool.



De pool wordt nu aangemaakt.

Je krijgt nu een overzicht van de Storage.

10.2.4.2 Zvol aanmaken

Een Zvol (ZFS virtual volume) is een virtual block device, vergelijkbaar met een fysieke harde schijf of SSD, maar dan virtueel en binnen je bestaande storage pool.

Een Zvol (ZFS-volume) is dan ook nodig om opslagruimte te reserveren voor iSCSI.

- Ga naar Datasets aan de linkerkant.
- Selecteer Pool1.
- Klik rechtsboven op Add Zvol

- Geef het Zvol de naam opslag1 en stel een grootte van 50GiB in. Deze grootte zal het beschikbare iSCSI-opslagvolume bepalen. Bij compression level kezen we voor Off. Laat de overige instellingen staan.

Add Zvol

Zvol Name* ⓘ

opslag1

Comments ⓘ

Size for this Zvol* ⓘ

50 GiB

Force size ⓘ

Sync* ⓘ

Inherit (standard)

Compress level* ⓘ

Off

ZFS Deduplication* ⓘ

Inherit (off)

Sparse ⓘ

Read-only* ⓘ

Inherit (off)

Block size* ⓘ

16 KiB

Snapdev* ⓘ

Inherit (hidden)

Encryption Options

Inherit (non-encrypted) ⓘ

Save

The dialog is titled 'Add Zvol'. It contains several configuration options: 'Zvol Name*' with value 'opslag1', 'Comments', 'Size for this Zvol*' with value '50 GiB', a 'Force size' checkbox, 'Sync' set to 'Inherit (standard)', 'Compress level*' set to 'Off', 'ZFS Deduplication' set to 'Inherit (off)', a 'Sparse' checkbox, 'Read-only' set to 'Inherit (off)', 'Block size*' set to '16 KiB', and 'Snapdev*' set to 'Inherit (hidden)'. At the bottom, there is an 'Encryption Options' section with a checked 'Inherit (non-encrypted)' checkbox and a large blue 'Save' button.

- Klik op Save om het Zvol aan te maken.
- Je ziet nu opslag1 in Pool1 staan.

Datasets				
	Dataset Name	Used / Available	Encryption	Roles
▼ E:	Pool1	50.79 GiB / 140.09 GiB	Unencrypted	Cloud
	opslag1	50.78 GiB / 190.87 GiB	Unencrypted	

10.2.4.3 iSCSI-service activeren

Voordat je verder kunt met het configureren van iSCSI, moet de service worden geactiveerd.

Ga naar Shares in het hoofdmenu. Je ziet hier dat je naast de SMB en NFS-shares service ook block (iSCSI)-service kan activeren.

Klik rechtsboven op de 3 puntjes bij Block (iSCSI) Shares Targets en kies voor Turn On Service.

Als je wil dat iSCSI telkens bij het opstarten geactiveerd wordt ga je naar System aan de linkerkant, Services.

Zet de schakelaar aan bij "Start Automatically".

10.2.4.4 iSCSI configuratie

Ga terug naar Shares aan de linkerkant.

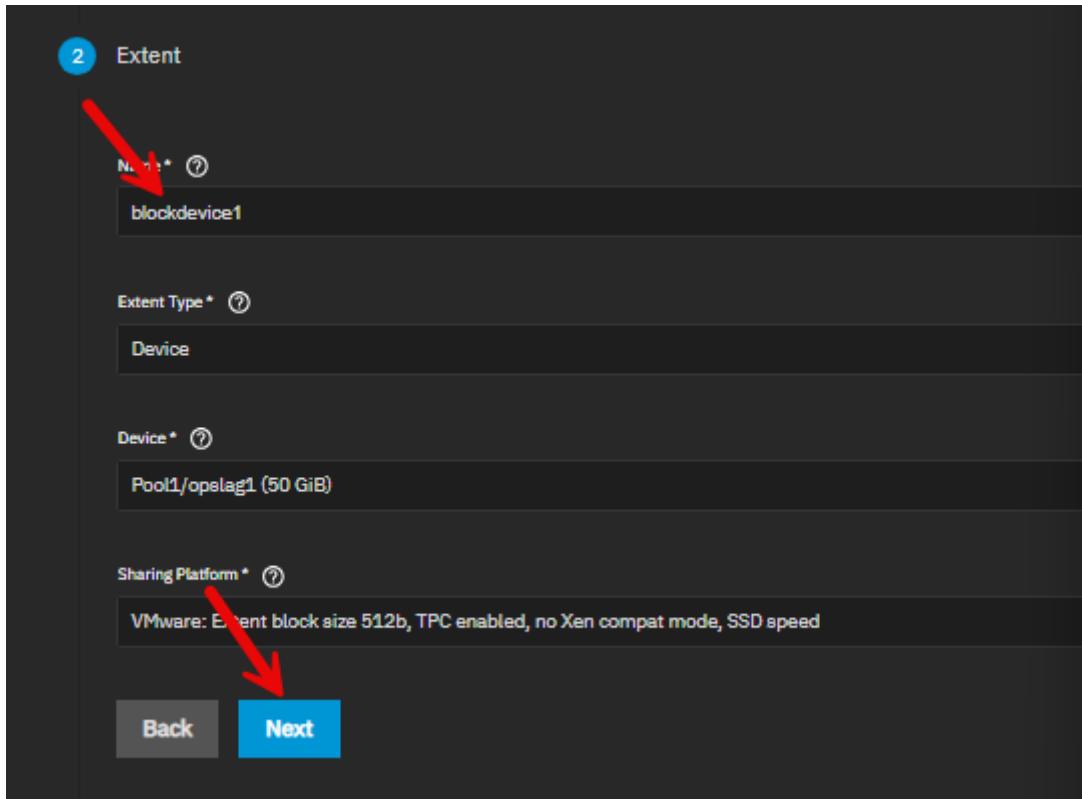
Op dit moment heb je nog geen target/extents aangemaakt. Je ziet: "No records have been added yet".

Dat betekent dat je ZVOL opslag1 nog niet is gekoppeld aan de iSCSI service die je draait.

TrueNAS heeft een ingebouwde wizard die het configureren van een iSCSI-share.

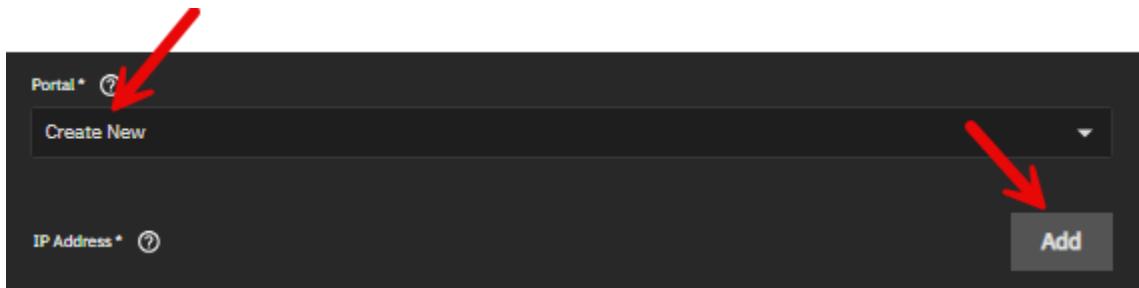
Na het klikken kies je bij Target voor Create New en Next.

Bij Extent kiezen we de naam blockdevice1. Bij extent type laat je Device staan. Bij device kies je uiteraard voor Pool1/opslag1(50GiB). De overige instellingen laat je staan en klik op Next.



In de volgende stap configureren we de Protocol options.

We kiezen het pijltje naar beneden en kiezen voor Create New bij Portal. Een portal is de combinatie van IP-adres + poort (standaard 3260) waarop TrueNAS iSCSI aanbiedt.

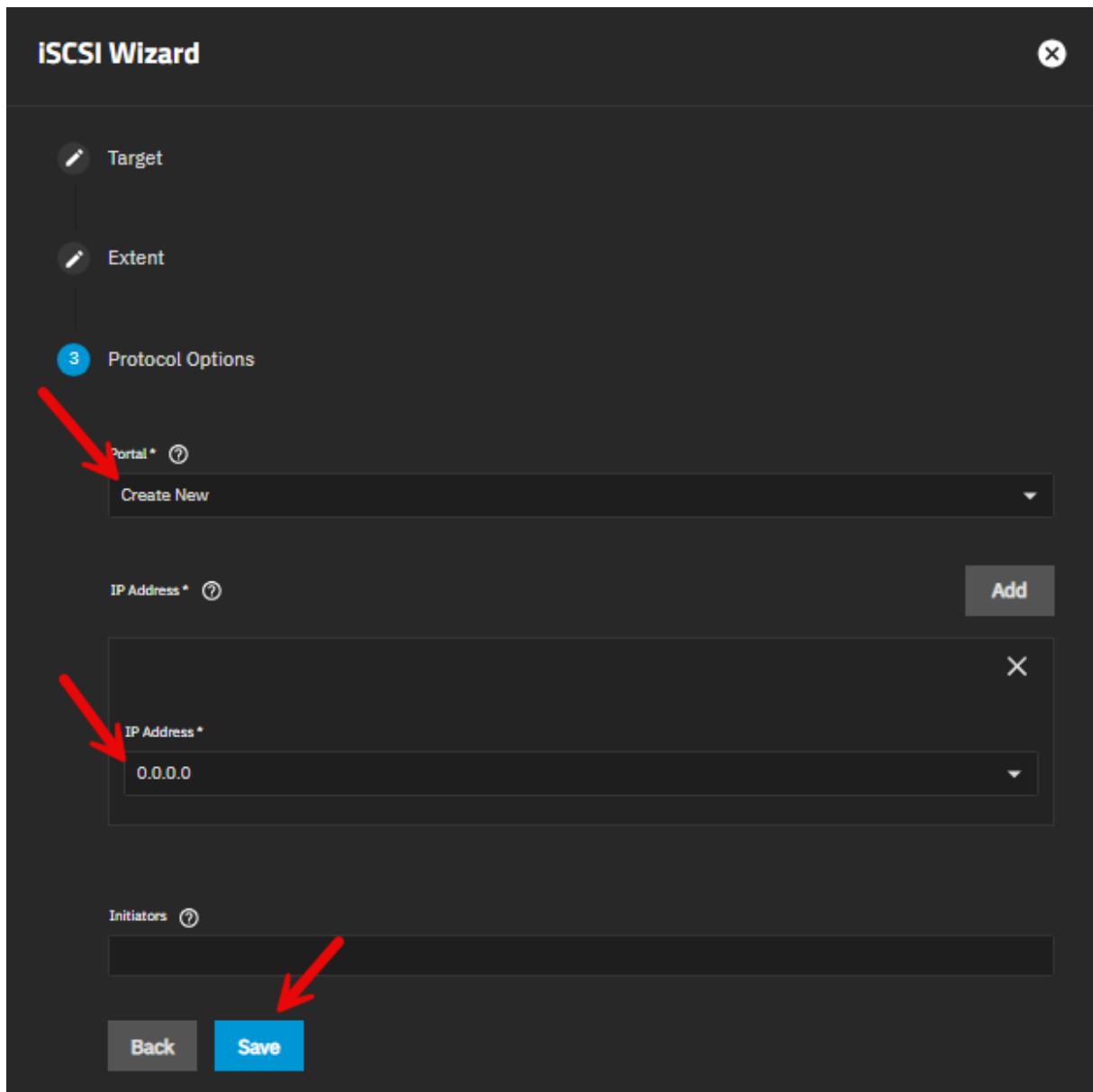


Aangezien we nog geen fixed IP-adres hebben ingesteld kiezen we voor 0.0.0.0 bij IP-adres. Klik hiervoor weer het pijltje naar beneden en selecteer 0.0.0.0.

In deze stap configureren we ook welke initiators (iSCSI-clients) toegang hebben tot het iSCSI-doel.

Initiators: Als je wilt dat alle iSCSI-clients toegang hebben, laat je het veld initiators leeg. Wij kiezen deze optie. Anders vul je een lijst in van hostnamen gescheiden door Enter.

Klik op Save.



Je kan de instellingen wijzigen of het overzicht opvragen als je op het potloodje klikt bij blockdevice1.

Edit iSCSI Target

Basic Info

Target Name * ⓘ
blockdevice1

Target Alias ⓘ
Target Alias

Authorized Networks

No items have been added yet.

ISCSI Group

Add groups

Add

Portal Group ID * ⓘ
1 (blockdevice1)

Initiator Group ID * ⓘ
1 (ALL Initiators Allowed)

Authentication Method * ⓘ
None

Authentication Group Number * ⓘ

Save

Wij veranderen niets en klikken op het kruisje rechtsboven.

10.2.5 Verbinding maken met iSCSI-target op ServerXX

Om verbinding te maken moet je eerst iSCSI initiator tools installeren.

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

We starten de de iSCSI service op.

```
student@serverXX:~$ sudo systemctl enable --now iscsid
```

```
Created symlink '/etc/systemd/system/multi-user.target.wants/iscsid.service' →
'/usr/lib/systemd/system/iscsid.service'.
```

We zoeken nu naar targets op het TrueNAS IP-adres.

```
student@serverXX:~$ sudo iscsiadm -m discovery -t sendtargets -p 10.10.10.128  
10.10.10.128:3260,1 iqn.2005-10.org.freenas.ctl:blockdevice1
```

Zoals je ziet wordt er 1 target gevonden.

Gebruik nu het IQN (de naam van de target) uit de discovery-commandoutput om verbinding te maken.

```
student@serverXX:~$ sudo iscsiadm -m node -T iqn.2005-  
10.org.freenas.ctl:blockdevice1 -p 10.10.10.128 --login  
  
Logging in to [iface: default, target: iqn.2005-  
10.org.freenas.ctl:blockdevice1, portal: 10.10.10.128,3260]  
  
Login to [iface: default, target: iqn.2005-10.org.freenas.ctl:blockdevice1,  
portal: 10.10.10.128,3260] successful.
```

Controleer nu of de node geregistreerd is.

```
student@serverXX:~$ sudo iscsiadm -m node  
10.10.10.128:3260,1 iqn.2005-10.org.freenas.ctl:blockdevice1
```

10.2.6 Initialiseren en formatteren van doel

Check nu of de blockdevice beschikbaar is.

```
student@serverXX:~$ lsblk  
  
NAME      MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS  
  
sda        8:0    0   50G  0 disk  
  
sr0       11:0    1 816,4M  0 rom  /run/media/student/RHEL-10-0-BaseOS-  
x86_64  
  
nvme0n1    259:0   0 100G  0 disk  
  
  └─nvme0n1p1  259:1   0  600M  0 part /boot/efi  
  
  └─nvme0n1p2  259:2   0     1G  0 part /boot  
  
  └─nvme0n1p3  259:3   0 98,4G  0 part  
  
    ├─rhel-root 253:0   0 64,8G  0 lvm  /  
  
    ├─rhel-swap 253:1   0     2G  0 lvm  [SWAP]  
  
    └─rhel-home 253:2   0 31,6G  0 lvm  /home
```

We zien sda met een grootte van 50GB 😊.

We maken hierop één partitie aanmaken. We gaan het hier doen met fdisk zoals jullie reeds kennen.

```
student@serverXX:~$ sudo fdisk /dev/sda

Command (m for help): n

Select (default p): <Enter>

Partition number (1-4, default 1): <Enter>

First sector (16384-104857631, default 16384): <Enter>

Last sector, +/-sectors or +/-size{K,M,G,T,P} (16384-104857631, default 104857631): <Enter>

Command (m for help): w
```

We voeren terug lsblk uit om te kijken of de partitie herkend wordt.

```
student@serverXX:~$ lsblk

NAME      MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
sda        8:0    0   50G  0 disk
└─sda1     8:1    0   50G  0 part
...

```

We maken nu een xfs-filesystem aan op deze partitie.

```
student@serverXX:~$ sudo mkfs.xfs /dev/sda1
...
Discarding blocks...Done.
```

We maken nu een mounting point en mounten.

```
student@serverXX:~$ sudo mkdir /mnt/iscsi
student@serverXX:~$ sudo mount /dev/sda1 /mnt/iscsi
```

10.2.7 Podman met iSCSI

Het zal niemand verwonderen dat het héél eenvoudig is om Podman met iSCSI in te stellen.

We stellen student als eigenaar en groep in voor /mnt/iscsi.

```
student@serverXX:~$ sudo chown student:student /mnt/iscsi/
```

We vragen de vrije ruimte op van /mnt/iscsi.

```
student@serverXX:~$ df /mnt/iscsi/  
Bestandssysteem 1K-blokken Gebruikt Beschikbaar Geb% Aangekoppeld op  
/dev/sda1 52355088 1035568 51319520 2% /mnt/iscsi
```

Je ziet heel duidelijk dat deze directory gekoppeld is aan /dev/sda1.

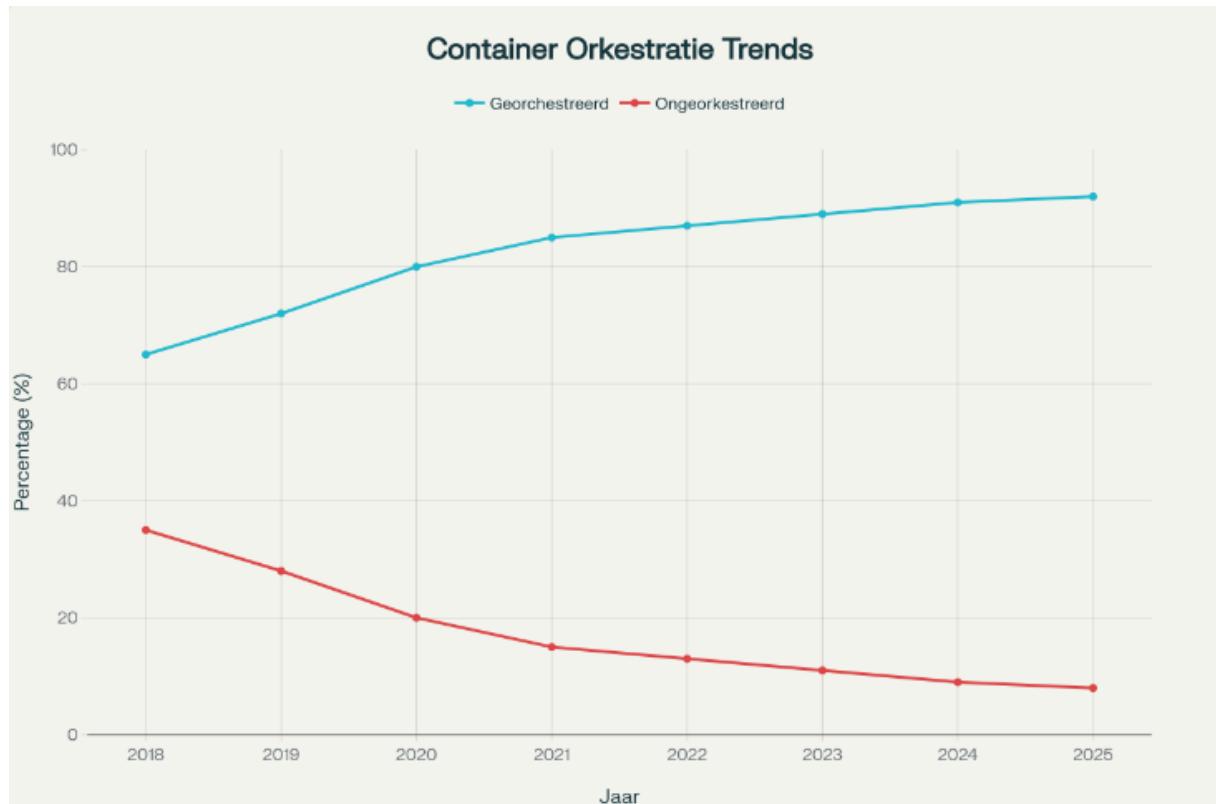
We maken een nieuwe container aan van UBI10-init en koppelen /mnt/iscsi in de container aan /iscsi en vragen op hoeveel vrije ruimte er is op /iscsi.

```
student@serverXX:~$ podman run --rm -it -v /mnt/iscsi:/iscsi  
registry.access.redhat.com/ubi10/ubi-init /bin/bash -c "df -h /iscsi"  
  
Filesystem Size Used Avail Use% Mounted on  
/dev/sda1 50G 1012M 49G 2% /iscsi
```

11 Kubernetes

11.1 Inleiding

De tijd dat we manueel containers moeten starten en stoppen is stilaan voorbij. Tegenwoordig worden de meeste containers (voornamelijk in de grotere bedrijven) beheerd door een orchestration tool.

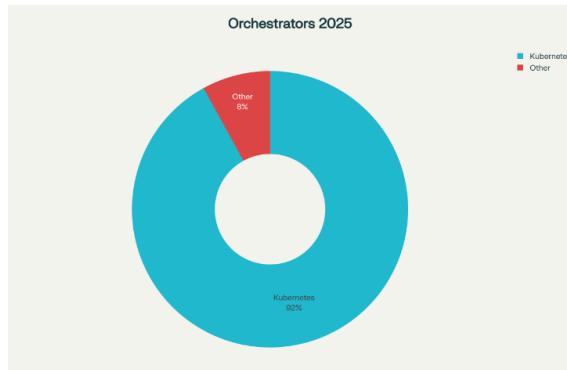


Kubernetes is het belangrijkste open-source platform om containergebaseerde applicaties automatisch te deployen, te schalen en te beheren. Het zorgt ervoor dat applicaties betrouwbaar draaien op een cluster van computers, ongeacht waar ze worden gehost (zowel in de cloud als on-premise).

Kubernetes kan bijvoorbeeld:

- Containers starten en stoppen
- Schaalvergroting of -verkleining op basis van vraag
- Load balancing
- Zelfherstel bij fouten

Het helpt om applicaties stabiel en efficiënt te draaien zonder handmatig beheer van servers. Wereldwijd wordt Kubernetes aanschouwd als de beste tool voor Container Orchestration.



Enkele alternatieven op een rij:

- Docker Swarm (met Mirantis-ondersteuning)
Eenvoudiger te beheren, maar minder schaalbare orchestrator dan Kubernetes.
- Amazon ECS
De managed container orchestrator van AWS, vooral populair bij wie volledig in AWS-cloud werkt.
- OpenShift
Gebaseerd op Kubernetes, ontwikkeld door RedHat; wordt als compleet platform beschouwd met extra features voor ontwikkelaars en security.

11.2 Kubernetes architectuur

11.2.1 Clusters

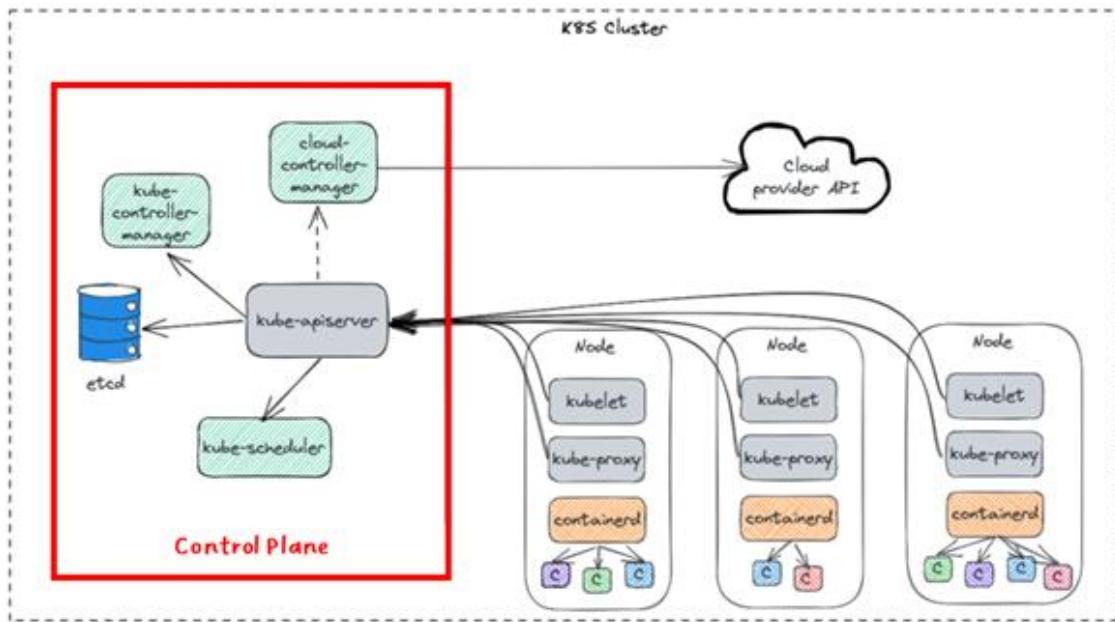
Binnen Kubernetes wordt met een cluster een groep machines (hosts) bedoeld die samenwerken om containerized applicaties te draaien en te beheren. Er zijn bij Kubernetes verschillende hosts die samenwerken die één bepaalde functie hebben (worker node of control plane) om een applicatie te hosten.

Binnen een Kubernetes-cluster wordt een host een "node" genoemd. Een node kan een fysieke machine zijn, maar ook een virtuele machine, bijvoorbeeld in een cloudomgeving.

Er zijn verschillende cloudproviders die Kubernetes als beheerde dienst aanbieden. Om echter een goed begrip te krijgen en te leren werken met Kubernetes, beperken we ons tot het gebruik van lokale virtuele nodes.

Naast de gewone worker nodes, is er ook de control plane node. Waar de worker nodes de uitvoerders zijn van de taken binnen het cluster, fungeert de control plane als de beheerder die het cluster coördineert en controleert. Voordat we beginnen met het opzetten van een Kubernetes-cluster, is het belangrijk om beide componenten goed te begrijpen.

Hieronder vind je de architectuur terug van Kubernetes.



11.2.2 Control Plane

Het brein van een Kubernetes-cluster is de Control Plane node. Dit is een verzameling van verschillende services die samenwerken om de volledige cluster correct te laten functioneren.

In een productieomgeving draait een Control Plane node nooit alleen. Voor voldoende betrouwbaarheid en beschikbaarheid worden altijd meerdere Control Plane nodes ingezet, vaak in een High Availability (HA) configuratie met minstens 3 of 5 nodes. Dit zorgt ervoor dat het cluster blijft werken, zelfs als één of meerdere Control Plane nodes uitvallen.

Voor test- en ontwikkelomgevingen volstaat meestal één enkele Control Plane node. Maar in een productieomgeving is het aan te raden om altijd minimaal drie Control Plane nodes te gebruiken om redundantie en fouttolerantie te garanderen.

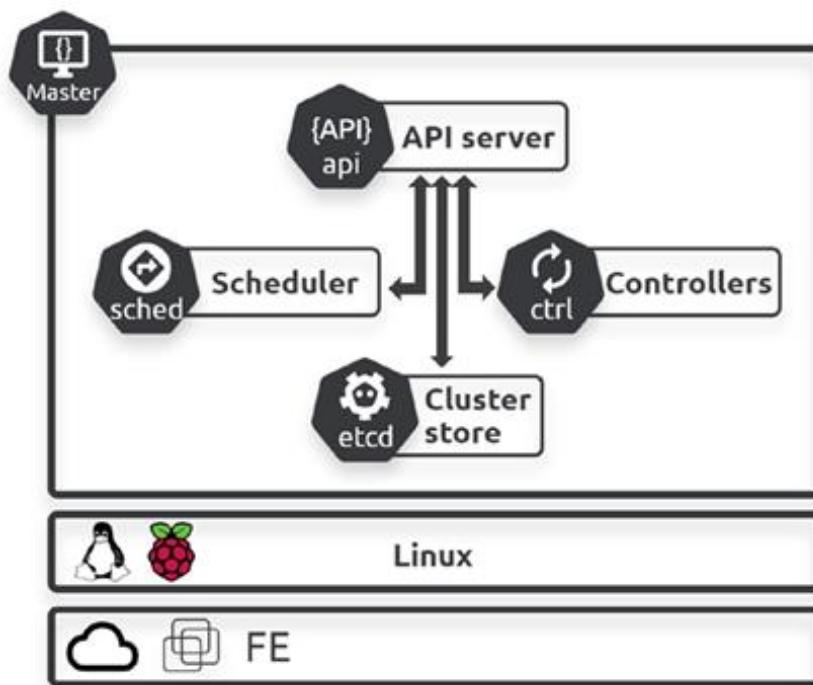
Kortom, de Control Plane beheert het cluster, verzorgd planning, statuscontrole, en coördinatie, en draait in HA setups meerdere keren om continuïteit te waarborgen.

Een Control Plane node in Kubernetes bestaat uit verschillende belangrijke services, die elk een specifieke taak binnen het cluster hebben. Zonder diep op alle details in te gaan, volgt hier een korte uitleg van de belangrijkste componenten:

- API server
De API-server is het centrale aanspreekpunt waarmee alle andere nodes binnen een Kubernetes-cluster communiceren met het control plane.
De API server luistert standaard op poort 6443 en ontvangt alle verzoeken binnen het cluster.
- Cluster Store (etcd)
Dit is de opslagplaats waarin alle configuraties en de huidige status van het gehele cluster worden bewaard.
Het functioneert als een consistente en betrouwbare database.

- Controller Manager
Dit is een verzameling controllers die verschillende aspecten van het cluster beheren. De hoofdtaak is om ervoor te zorgen dat de werkelijke toestand van het cluster altijd overeenkomt met de gewenste toestand, bijvoorbeeld dat de juiste applicaties draaien zoals geconfigureerd.
- Scheduler
De scheduler is verantwoordelijk voor het toewijzen van nieuwe taken aan gezonde nodes. Hij beslist waar nieuwe pods moeten worden gestart door de status van de nodes te controleren via de API server.
- Cloud Controller Manager
Deze service maakt het mogelijk om het Kubernetes-cluster te integreren met cloudproviders. Het biedt een centraal punt voor communicatie en beheer van cloudspecifieke onderdelen zoals load balancers en opslag.

Deze services vormen samen het Control Plane en zorgen ervoor dat het Kubernetes-cluster betrouwbaar, schaalbaar en automatisch beheerd wordt.



Uiteraard zal je nooit user applicaties uitvoeren op je Control Plane Node.

11.2.3 Worker nodes

Worker nodes zijn de primaire werkpaarden van een Kubernetes-cluster. Zij zijn verantwoordelijk voor het hosten en uitvoeren van de applicaties.

De belangrijkste taken van een worker node zijn:

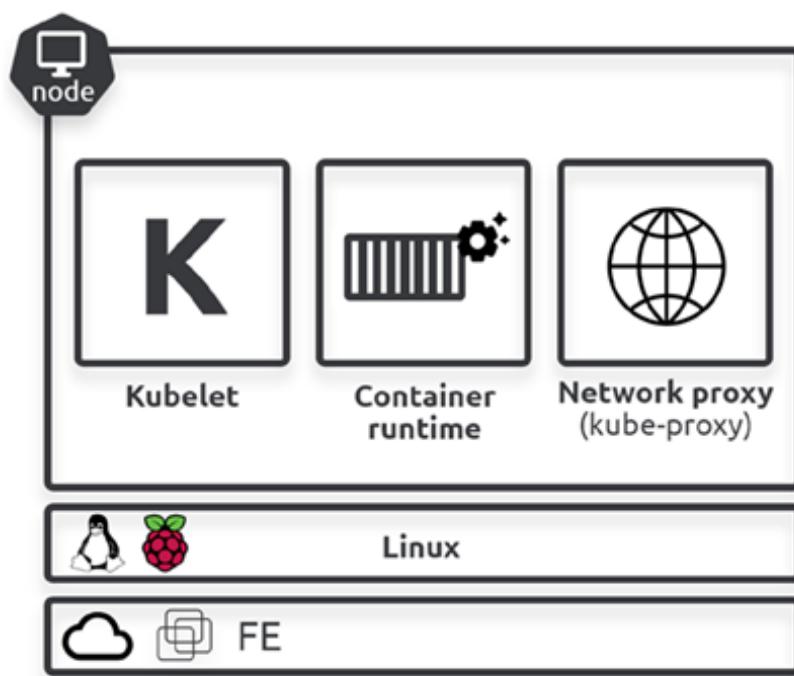
- Continu de Control Plane raadplegen via de API-server voor nieuwe taken (work assignments).

- Uitvoeren van toegewezen taken zodra deze beschikbaar zijn.
- Regelmatig rapporteren aan de Control Plane over hun status.

Een worker node bestaat uit verschillende essentiële componenten:

- Kubelet
Een agent die ervoor zorgt dat de containers binnen pods correct draaien en onderhoudt de communicatie met de Control Plane.
- Container runtime
De software die containers uitvoert, zoals containerd of CRI-O.
- Kube-proxy
Verantwoordelijk voor de netwerkregels en het afhandelen van netwerkverkeer naar de containers binnen de node.

Deze taken worden continu herhaald zolang de node actief is, waardoor de applicaties betrouwbaar en schaalbaar kunnen draaien binnen het cluster.



Laat ons eerst even kort ingaan op deze verschillende onderdelen. In tegenstelling tot de Control Plane, gaan we 2 van de 3 onderdelen van een Worker Node uitgebreider bestuderen. De Network-Proxy service gaan we in een later verder uitleggen.

11.2.3.1 Kubelet

De Kubelet is het belangrijkste onderdeel van een worker node in Kubernetes.

Wanneer een worker node aan het cluster wordt toegevoegd, registreert de Kubelet de node bij de API-server van de Control Plane. Daarna onderhoudt de Kubelet continu verbinding met de API-server om te controleren of er nieuwe taken (pods) aan die node zijn toegewezen.

Wanneer er een nieuwe taak is, zorgt de Kubelet ervoor dat deze wordt uitgevoerd door een container te starten. Daarnaast onderhoudt de Kubelet een voortdurende communicatie met de Control Plane om voortdurend statusupdates te versturen over de uitvoering van de taak.

Als de Kubelet een taak niet kan uitvoeren, rapporteert hij dit aan de Control Plane, die dan beslist wat de volgende stap is, bijvoorbeeld om de taak toe te wijzen aan een andere node. De Kubelet wordt automatisch geïnstalleerd en geactiveerd zodra een node zich bij het cluster aansluit, en speelt een cruciale rol in het draaiende houden van containerized applicaties binnen Kubernetes.

11.2.3.2 Container Runtime

Kubernetes zorgt voor de orkestratie van containers, wat betekent dat de nodes in staat moeten zijn om containers uit te voeren. Daarom is er een Container Runtime nodig op elke node. De Container Runtime haalt de benodigde container images op en start of stopt containers op de node op basis van de taken die de Kubelet ontvangt.

Vroeger had Kubernetes native ondersteuning voor Docker als container runtime. Inmiddels maakt Kubernetes gebruik van containerd, een lichtgewicht container runtime die is ontstaan uit de kerncomponenten van Docker. Deze container runtime, containerd, is door Docker Inc. aan de CNCF (Cloud Native Computing Foundation, de organisatie achter Kubernetes) gedoneerd en wordt nu universeel ondersteund in Kubernetes-omgevingen.

In plaats van containerd kan je ook gebruik maken van CRI-O.

Voor pure Kubernetes-gebruik is CRI-O een gespecialiseerde, lichte en veilige runtime. Containerd biedt iets meer functionaliteit en bredere compatibiliteit en is momenteel iets populairder in cloudgestuurde en algemene Kubernetes-omgevingen

11.3 Installatie

11.3.1 k3s versus k8s

Kubernetes draait doorgaans op clusters van meerdere machines (nodes), waarbij er een control plane en meerdere worker nodes zijn, die samen containerized applicaties beheren. Dit type architectuur is ideaal voor productieomgevingen die schaalbaarheid, hoge beschikbaarheid en veerkracht vereisen.

Nu we de basisarchitectuur van een K8's cluster onder de loep hebben genomen, is het tijd om zelf onze eerste cluster op te zetten. We zullen gebruik maken van K3s.

K3s is een lichtgewicht, compacte Kubernetes-distributie ontworpen voor o.a. ontwikkel- en testomgevingen. Het draait Kubernetes functionaliteit op één machine (single-node cluster) of een kleine cluster van machines met minimale resources.

In k3s kunnen control plane en worker nodes gecombineerd binnen dezelfde node. Dit maakt k3s ideaal om snel een complete maar eenvoudige Kubernetes-omgeving lokaal of in resourcebeperkte

situaties op te zetten. K3s kan ook in een multi-node setup draaien maar dat zullen we niet bespreken.

Het draaien van een Kubernetes-cluster op één machine heeft vooral zin voor:

- Leer- en testdoeleinden
Het is veel eenvoudiger en sneller om Kubernetes te leren en te experimenteren op één machine zonder meerdere fysieke servers of complexe infrastructuur op te zetten.
- Ontwikkelomgevingen
Developers kunnen lokaal applicaties in containers draaien en testen met dezelfde orkestratiertools die in productie gebruikt worden, wat consistentie bevordert.
- Kleine productieomgevingen
Voor kleine bedrijven of specifieke workloads die geen volledige schaalbaarheid of hoge beschikbaarheid nodig hebben, biedt zo'n setup toch veel voordelen van container orkestratie.

Het opzetten van K8s valt buiten de doelstellingen van de cursus.

11.3.2 VM

Maak een nieuwe RHEL 10 VM aan die aan volgende eisen voldoet:

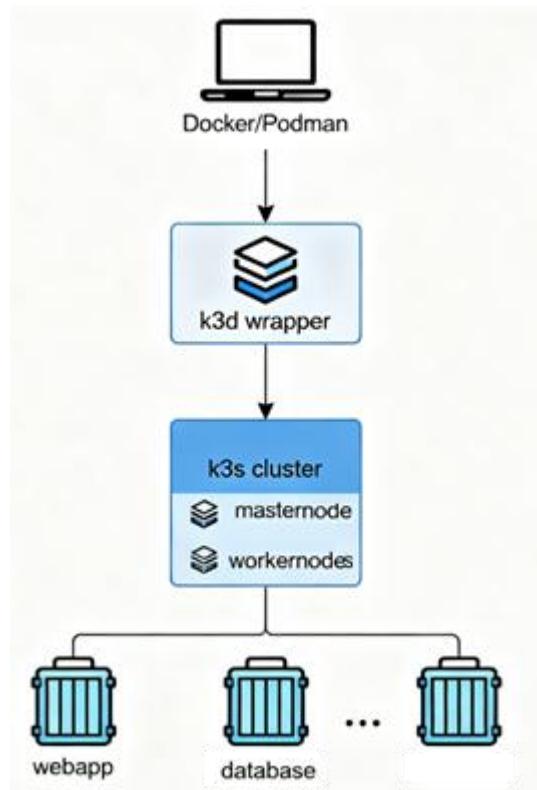
- Geen GUI.
- Hostname: virt-K8s-XX (xx zijn je initialen).
- Gebruiker student met wachtwoord PXL.
- Statisch IP: 192.168.112.10.
- Bereikbaar is via SSH.

11.3.3 Installatie Kubernetes

11.3.3.1 Mogelijkheden

Zoals reeds aangehaald laat k3s ons toe om de basis van Kubernetes onder de knie te krijgen.

Voor deze cursus gaan we werken met k3d. Dit is een wrapper die k3s draait in docker of podman.



Installatieprocedures verschillen per versie van Red Hat Enterprise Linux (RHEL). Op RHEL 10 ondersteunt Red Hat geen Docker meer (je kan wel via een alternatieve repository Docker installeren zoals we geleerd hebben) en promoot het Podman als container runtime.

Hoewel OpenShift van Red Hat een Kubernetes-platform biedt, is het voorlopig niet wijdverspreid.

Wij focussen ons op k3d op RHEL 10 met Docker maar bieden ook installatierichtlijnen voor k3d met Podman op RHEL 10 en voor k3d met Docker op RHEL 9.

Wil je experimenteren met Podman, dan is Minikube een betere keuze dan k3d, omdat het breder en stabiever is in ondersteuning voor diverse container runtimes, inclusief Podman. Zo heb je een soepelere en betrouwbare Kubernetes-ervaring tijdens het leren en uitproberen.

11.3.3.2 RHEL 9

- Docker installeren.
`student@virt-K8s-XX :~$ sudo dnf config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo`
`...`
`student@virt-K8s-XX :~$ sudo dnf install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin`
`...`
`student@virt-K8s-XX :~$ sudo systemctl enable --now docker`
- K3d installeren.

```
[student@virt-K8s-XX ~]$ sudo wget -q -O -
https://raw.githubusercontent.com/k3d-io/k3d/main/install.sh |
```

```
bash
Preparing to install k3d into /usr/local/bin
k3d installed into /usr/local/bin/k3d
Run 'k3d --help' to see what you can do with it.
```

- k3d heeft toegang nodig tot de Docker-daemon (docker.sock) om containers te kunnen starten en het k3s-cluster op te zetten. Om als niet-root gebruiker Docker-commando's te kunnen uitvoeren, moet je lid zijn van de dockergroep. Het commando newgrp docker zorgt ervoor dat je huidige shell-sessie overschakelt naar de groep docker zonder dat je uit- en weer hoeft in te loggen.

```
[student@virt-K8s-XX ~]$ sudo usermod -aG docker $USER
[student@virt-K8s-XX ~]$ newgrp docker
```

- Kubectl installeren.
Hoewel k3d het cluster opzet, moet je nog steeds iets hebben om ertegen te praten.

```
[student@virt-K8s-XX ~]$ sudo cat <<EOF | sudo tee
/etc/yum.repos.d/Kubernetes.repo
[Kubernetes]
name=Kubernetes
baseurl=https://pkgs.K8s.io/core:/stable:/v1.32/rpm/
enabled=1
gpgcheck=1
gpgkey=https://pkgs.K8s.io/core:/stable:/v1.32/rpm/repo
md.xml.key
EOF
[student@virt-K8s-XX ~]$ sudo dnf update && sudo dnf install
kubectl -y
```

11.3.3.3 RHEL 10

Volg onderstaande stappen:

- Docker installeren (zoals beschreven in hoofdstuk 2). De stappen staan hieronder ter herhaling.

```
student@virt-K8s-XX :~$ sudo dnf config-manager --add-repo
https://download.docker.com/linux/centos/docker-ce.repo
...
student@virt-K8s-XX :~$ sudo dnf install -y docker-ce docker-ce-
cli containerd.io docker-buildx-plugin docker-compose-plugin
...
student@virt-K8s-XX :~$ sudo systemctl enable --now docker
```

- k3d heeft toegang nodig tot de Docker-daemon (docker.sock) om containers te kunnen starten en het k3s-cluster op te zetten. Om als niet-root gebruiker Docker-commando's te kunnen uitvoeren, moet je lid zijn van de dockergroep. Het commando newgrp docker zorgt ervoor dat je huidige shell-sessie overschakelt naar de groep.

```
student@virt-K8s-XX :~$ sudo usermod -aG docker $USER
```

```
student@virt-K8s-XX :~$ newgrp docker
```

- K3D installeren.

```
student@virt-K8s-XX :~$ wget -q -O -
https://raw.githubusercontent.com/k3d-io/k3d/main/install.sh | bash
```

```
Preparing to install k3d into /usr/local/bin
```

```
k3d installed into /usr/local/bin/k3d
```

```
Run 'k3d --help' to see what you can do with it.
```

- Kubectl installeren.

```
student@virt-K8s-XX :~$ sudo cat <<EOF | sudo tee
/etc/yum.repos.d/Kubernetes.repo
[Kubernetes]
name=Kubernetes
baseurl=https://pkgs.K8s.io/core:/stable:/v1.32/rpm/
enabled=1
gpgcheck=1
gpgkey=https://pkgs.K8s.io/core:/stable:/v1.32/rpm/repo
md.xml.key
EOF
student@virt-K8s-XX :~$ sudo dnf update
student@virt-K8s-XX :~$ sudo dnf -y install kubectl
```

- Optioneel (afgeraden): Podman configureren zodat K3D er gebruik van kan maken:
Podman in K3D is experimenteel. Daarom gaan we enkel met Docker werken voor K8S en mag je deze stap overslaan.

- Podman socket beschikbaar maken voor k3d.

```
student@virt-K8s-XX :~$ sudo systemctl enable --now
podman.socket
```

- Zorgen dat K3D naar de juiste "docker"-socket verwijst. Dit doen we door een symbolische link aan te maken.

```
student@virt-K8s-XX :~$ sudo ln -s /run/podman/podman.sock
/var/run/docker.sock
```

11.4 Cluster aanmaken

Nadat alles is geïnstalleerd, kunnen we onze eerste k3s cluster opzetten via k3d. Hiervoor kan je volgende cmd uitvoeren:

```
student@virt-K8s-XX :~$ k3d cluster create dev-cluster --servers
1 --agents 1
```

```
...
```

--servers 1 betekent dat er 1 server node wordt aangemaakt.

--agents 1 betekent dat er 1 worker node wordt aangemaakt.

Om nu te kijken welke nodes er allemaal aangemaakt zijn kan men gebruik maken volgend cmd:

NAME	STATUS	ROLES	AGE
VERSION			
k3d-dev-cluster-agent-0 v1.31.5+k3s1	Ready	<none>	3m5s
k3d-dev-cluster-server-0 v1.31.5+k3s1	Ready	control-plane, master	3m8s

De output van dit cmd toont duidelijk dat er 2 nodes draaien, waarvan 1 node de Control plane is en de andere node een werker.

In de praktijk zal je meestal maar één k3d-cluster tegelijk draaien.

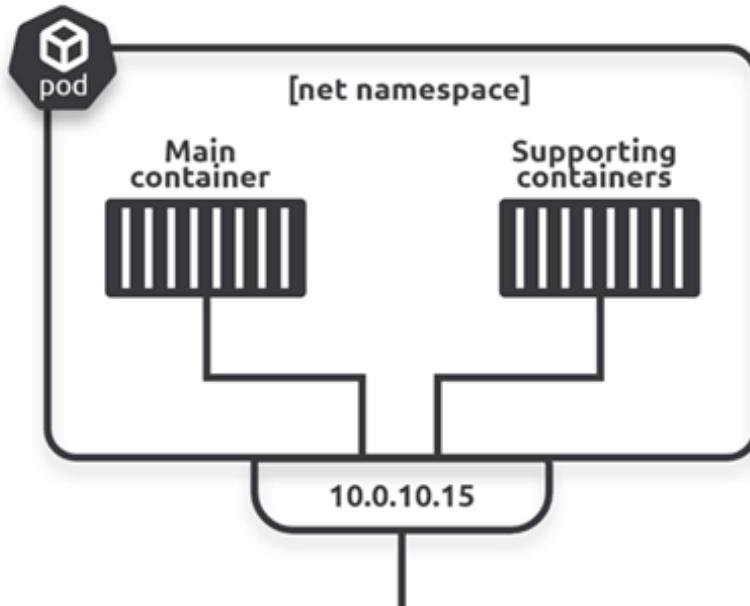
11.5 Pods

11.5.1 Inleiding

Een Kubernetes-pod is de kleinste beheersbare eenheid in Kubernetes en fungeert als een afgeschermd omgeving waarin één of meer containers draaien. De pod zelf draait geen applicatie, maar beheert de container(s) die dat doen. Containers binnen dezelfde pod delen dezelfde netwerkruimte (IP-adres) en opslag, wat zorgt voor directe en efficiënte communicatie tussen deze containers via localhost.

Dit is vergelijkbaar met Podman, waar een pod ook een verzameling containers is die nauw samenwerken. Het belangrijkste onderscheid is dat de pod de operationele context en resource-sharing regelt, terwijl de container de daadwerkelijke applicatie en bijbehorende processen draait.

Kortom: binnen Kubernetes draait elke applicatie-container in een pod, en de pod zorgt voor de isolatie, netwerk- en opslagresources die nodig zijn om die containers als één samenhangende eenheid te laten functioneren.

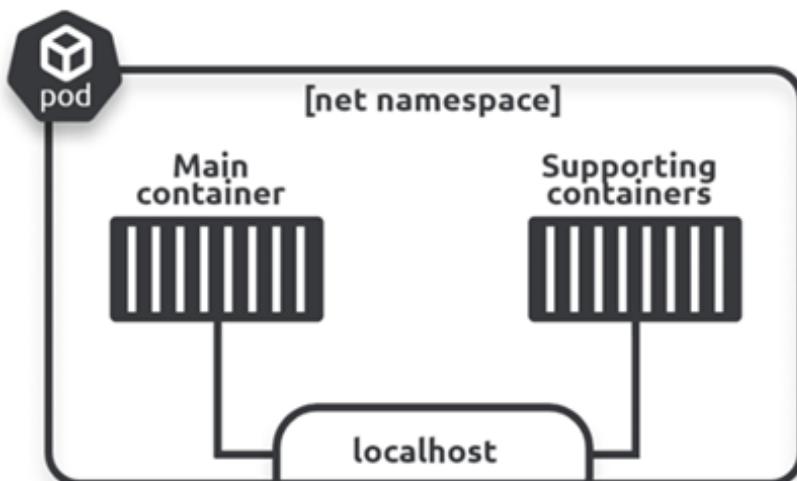


Elk Kubernetes-pod krijgt een eigen netwerkstack en kernel namespace toegewezen, wat betekent dat de pod een geïsoleerde netwerk- en systeemomgeving heeft waarin container(s) kunnen draaien. Dit maakt de pod een afgeschermd eenheid binnen het cluster.

Over het algemeen wordt aangeraden om voor elke applicatie of microservice een aparte pod te maken, zodat ze onafhankelijk kunnen schalen en beheerd worden. Het gebruik van meerdere containers in één pod is meestal voor geavanceerdere toepassingen nodig, waarbij voorbeeld een helper-container of sidecar-container (container voor extra functionaliteit) ondersteuning biedt. Dit vereist meer diepgaande kennis van Kubernetes.

Samengevat delen containers in een pod:

- De netwerkstack (IP-adres en netwerkinterfaces)
- Storage volumes (gevolumeerde opslag)
- Kernel namespace (system-level isolatie)
- Ze draaien op dezelfde fysieke node

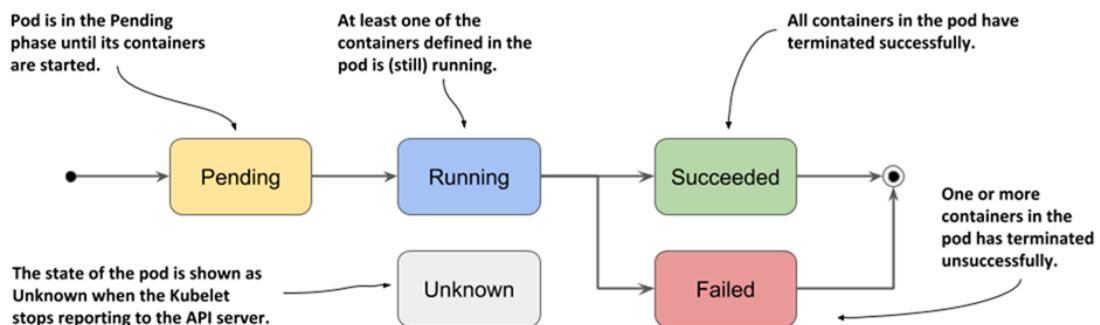


11.5.2 Lifecycle

Een pod bevindt zich steeds in een bepaalde status en heeft een eindige levenscyclus. Elke pod zal immers ooit eens stoppen met bestaan. De levenscyclus van een pod doorloopt meestal de volgende fasen:

- Pending
De pod is aangemaakt en de control plane probeert deze op een geschikte, gezonde node op te zetten, maar de containers zijn nog niet allemaal gestart.
- Running
Ten minste één container binnen de pod draait actief en de pod functioneert.
- Succeeded
Alle containers binnen de pod zijn succesvol gestopt (beëindigd), bijvoorbeeld bij batchtaken die klaar zijn.
- Failed
Eén of meer containers zijn ongepland gestopt met een fout.
- Unknown
De status van de pod kan tijdelijk niet worden bepaald.

Zolang een pod correct werkt (de container in de pod draait correct), zal deze dus in een running status blijven.



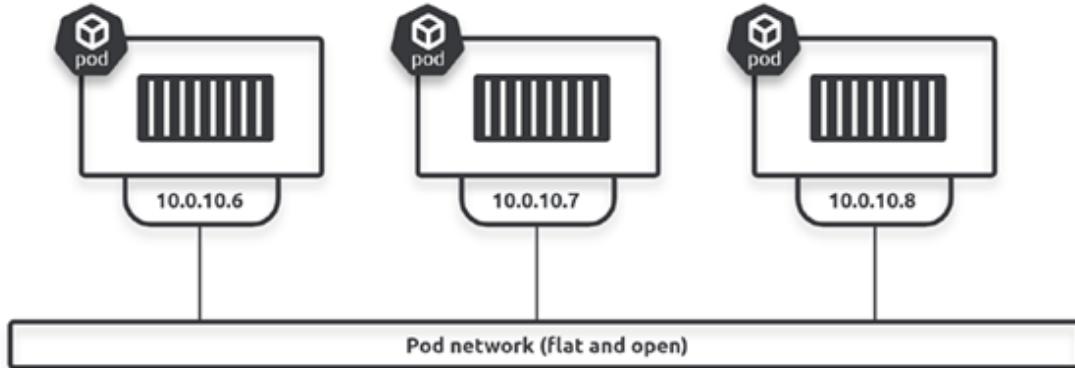
Een pod kan je niet aanpassen. Indien er iets moet gewijzigd worden aan een pod, moet je deze laten sterven en een nieuwe pod deployen.

11.5.3 Pod netwerk

In Kubernetes krijgt elke pod een eigen IP-adres toegekend, afkomstig uit een vooraf gedefinieerd IP-bereik dat wordt toegewezen aan pods in het cluster. Hierdoor heeft elke pod een uniek IP-adres dat gedeeld wordt door alle containers in die pod.

Het netwerk binnen een Kubernetes-cluster is standaard plat en open binnen de cluster, wat betekent dat alle pods zonder beperkingen met elkaar kunnen communiceren in dezelfde cluster, onafhankelijk van de node. Om deze communicatie te beperken en de veiligheid te verhogen, kan men gebruikmaken van Network Policies. Deze policies definiëren regels die bepalen welke pods met elkaar mogen communiceren, waardoor netwerkverkeer binnen het cluster kan worden afgeschermd.

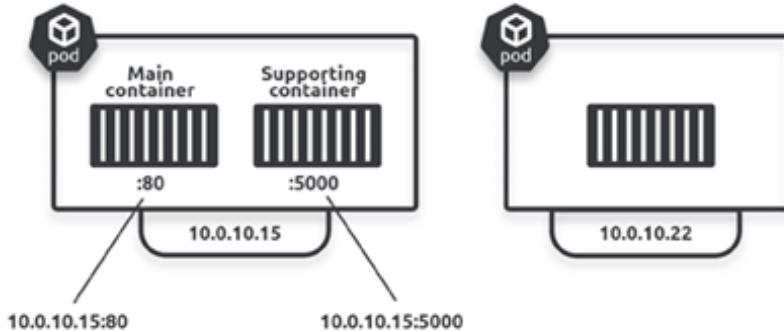
en beveiligd.



Elke pod krijgt zijn eigen netwerk namespace, waarin alle containers binnen die pod dezelfde netwerkconfiguratie delen. Dit betekent dat alle containers in een pod hetzelfde IP-adres hebben, dat gedeeld wordt door alle containers binnen die pod. Communicatie tussen containers in dezelfde pod verloopt via localhost op de wijze van de eigen IP en hun toegewezen poorten.

Wanneer externe systemen contact willen opnemen met een container in de pod, gebeurt dat via het IP-adres van de pod zelf.

Indien men een multicontainer pod heeft, moet er dus ook op het niveau van de pod met poorten gewerkt worden om de specifieke container binnen de pod te kunnen bereiken.



11.5.4 Pod aanmaken

Een pod kan op twee manieren aangemaakt worden:

- Imperatief
 - Een pod aanmaken door het invoeren van commando's.
- Declaratief
 - Een pod (of andere objecten) aanmaken door het gebruik van een manifest bestand.

Wij gaan steeds gebruik maken van de declaratieve manier om een pod op te zetten. Hiervoor moeten we dus leren hoe een manifest bestand wordt opgebouwd.

11.5.4.1 Imperatief

Je maakt de Pod dus rechtstreeks aan door een commando in te voeren in de terminal.

```
student@serverXX:~$ kubectl run nginx-pod --image=nginx --restart=Never --port=80 --labels="zone=prod,version=v1"
```

```
pod/mypod created
```

- kubectl run: maakt een Pod aan
- nginx-pod: naam van de Pod
- --image=nginx: container image dat gebruikt wordt
- --restart=Never: pod zal niet herstarten bij crash of verwijdering
- --labels="zone=prod,version=v1": stelt labels in

Je kan de pod erna zien door onderstaande uit te voeren:

```
student@serverXX:~$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
nginx-pod	1/1	Running	0	4m18s

11.5.4.2 Declaratief

Een manifestbestand wordt in Kubernetes doorgaans gedefinieerd in YAML-formaat (hier: second-pod.yaml). Hieronder staat een voorbeeld van een eenvoudig manifestbestand waarmee een Pod met een Nginx-server wordt aangemaakt.

```
kind: Pod (1)
apiVersion: v1 (2)
metadata:
  name: nginx-pod2 (3)
  labels: (4)
    zone: prod
    version: v1
spec:
  containers: (5)
    - name: nginxcont (6)
      image: nginx (7)
      ports:
        - containerPort: 80 (8)
```

Kind

Hiermee definiëren we welk soort object we willen aanmaken.

apiVersion

Geeft aan welke apiversie we gebruiken in de manifest file.

metadata.name

Hiermee geven we een unieke naam aan het object.

metadata.labels

Labels worden gebruikt voor het koppelen van services aan pods (meer hierover in de paragraaf Services).

spec.containers

In dit deel gaan we onze specifieke containers gaan definiëren.

spec.containers.name

Unieke naam die we aan onze container geven.

Containernaam is niet instelbaar als je imperatief werkt. Bij imperatief maakt Kubernetes de containernaam automatisch gelijk aan de Pod-naam

spec.containers.image

Geeft aan welke docker image er gebruikt wordt voor de container.

... containerport

Specificeert welke poort er open moet gezet worden op de container.

Zoals je kan zien zijn er heel wat belangrijke elementen die we moeten definiëren in een pod. Om deze pod te laten draaien volstaat het om de yaml file lokaal te bewaren en deze te laten deployen op de cluster.

- Maak een nieuwe cluster aan voor de declaratieve pod:

```
student@virt-k8s-XX:~$ k3d cluster create test-cluster -s 1 -a 1
INFO[0000] Prep: Network
INFO[0000] Created network 'k3d-test-cluster'
INFO[0000] Created image volume k3d-test-cluster-images
INFO[0000] Starting new tools node...
INFO[0000] Starting node 'k3d-test-cluster-tools'
INFO[0001] Creating node 'k3d-test-cluster-server-0'
INFO[0001] Creating node 'k3d-test-cluster-agent-0'
INFO[0001] Creating LoadBalancer 'k3d-test-cluster-serverlb'
INFO[0001] Using the k3d-tools node to gather environment
information
INFO[0001] HostIP: using network gateway 172.19.0.1 address
INFO[0001] Starting cluster 'test-cluster'
INFO[0001] Starting servers...
INFO[0001] Starting node 'k3d-test-cluster-server-0'
INFO[0003] Starting agents...
INFO[0004] Starting node 'k3d-test-cluster-agent-0'
INFO[0006] Starting helpers...
INFO[0006] Starting node 'k3d-test-cluster-serverlb'
```

```
INFO[0012] Injecting records for hostAliases (incl.  
host.k3d.internal) and for 3 network members into CoreDNS  
configmap...
```

```
INFO[0014] Cluster 'test-cluster' created successfully!
```

```
INFO[0014] You can now use it like this:
```

```
kubectl cluster-info
```

- Maak een file aan genaamd: second-pod.yaml met daarin de inhoud van de voorbeeld yaml. Let op met de spaties en dat je geen tabs gebruikt!

```
student@virt-k8s-XX:~$ nano second-pod.yaml
```

```
kind: Pod
apiVersion: v1
metadata:
  name: nginx-pod2
  labels: (4)
    zone: prod
    version: v1
spec:
  containers:
    - name: nginxcont
      image: nginx
      ports:
        - containerPort: 80
```

- Deploy de pod door gebruik te maken van volgende cmd:

```
student@virt-k8s-XX:~$ kubectl apply -f second-pod.yaml
pod/nginx-pod2 created
```

- Bekijk de status van de pod door volgende cmd:

```
student@virt-k8s-XX:~$ kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE     IP          ...
nginx-pod2  1/1     Running   0          46s    10.42.1.6 ...
```

Zoals je ziet zie je alleen de pods van de cluster waar je nu mee werkt.

- Je kan alle clusters opvragen met onderstaand commando.

```
student@serverXX:~$ kubectl config get-clusters
```

```
NAME
```

```
k3d-test-cluster
```

```
k3d-dev-cluster
```

- Indien je details van de pod wil weten, kan je gebruik maken van volgende cmd:

```
student@virt-k8s-XX:~$ kubectl describe pod nginx-pod2
```

```
Name:           nginx-pod2
Namespace:      default
Priority:       0
Service Account: default
Node:           k3d-test-cluster-server-0/172.19.0.2
Start Time:     Fri, 07 Nov 2025 12:49:53 +0100
Labels:         version=v1
                zone=prod
Annotations:    <none>
Status:         Running
IP:             10.42.1.6
...
```

- Om deze pod te verwijderen, volstaat het om het volgende commando uit te voeren:

```
student@virt-k8s-XX:~$ kubectl delete pod nginx-pod2
```

```
pod "nginx-pod2" deleted
```

```
student@virt-k8s-XX:~$ kubectl get pods -o wide
```

```
No resources found in default namespace.
```

K8s maakt gebruik maakt van de Dockerhub voor het downloaden van de containers.

Natuurlijk is het aanmaken van een pod niet het enige wat we willen doen. Vaak moet men ook in interactie gaan met de pod om in de container bepaalde cmd's uit te voeren. Hiervoor kan je gebruiken maken van "kubectl exec". Dit laat toe om een cmd uit te voeren in onze pod en de output weer te geven. Indien we onze vorige pod terug uitvoeren en een ls willen uitvoeren in de container, doen we dit op de volgende manier:

```
student@virt-k8s-XX:~$ kubectl apply -f second-pod.yaml
```

```
pod/nginx-pod2 created
```

```
student@virt-k8s-XX:~$ kubectl exec nginx-pod2 -- ls -al
```

```
total 4
```

```
drwxr-xr-x. 1 root root 39 Oct 30 10:44 .
```

```

drwxr-xr-x.  1 root root   39 Oct 30 10:44 ..
lrwxrwxrwx.  1 root root    7 Aug 24 16:20 bin -> usr/bin
drwxr-xr-x.  2 root root   6 Aug 24 16:20 boot
drwxr-xr-x.  5 root root  360 Oct 30 10:44 dev
drwxr-xr-x.  1 root root   41 Oct 28 21:50 docker-entrypoint.d
-rwxr-xr-x.  1 root root 1620 Oct 28 21:49 docker-entrypoint.sh
drwxr-xr-x.  1 root root   32 Oct 30 10:44 etc
drwxr-xr-x.  2 root root   6 Aug 24 16:20 home
lrwxrwxrwx.  1 root root    7 Aug 24 16:20 lib -> usr/lib
lrwxrwxrwx.  1 root root   9 Aug 24 16:20 lib64 -> usr/lib64
drwxr-xr-x.  2 root root   6 Oct 20 00:00 media
drwxr-xr-x.  2 root root   6 Oct 20 00:00 mnt
drwxr-xr-x.  2 root root   6 Oct 20 00:00 opt
dr-xr-xr-x. 379 root root   0 Oct 30 10:44 proc
drwx-----.  2 root root  37 Oct 20 00:00 root
drwxr-xr-x.  1 root root   38 Oct 30 10:44 run
lrwxrwxrwx.  1 root root   8 Aug 24 16:20 sbin -> usr/sbin
drwxr-xr-x.  2 root root   6 Oct 20 00:00 srv
dr-xr-xr-x. 13 root root   0 Oct 30 08:47 sys
drwxrwxrwt.  2 root root   6 Oct 20 00:00 tmp
drwxr-xr-x.  1 root root  66 Oct 20 00:00 usr
drwxr-xr-x.  1 root root  19 Oct 20 00:00 var

```

Het teken -- in het commando dient als scheidingsteken tussen de opties van kubectl exec en het commando dat je wilt uitvoeren binnen de container van de pod. Alles wat na -- komt, wordt gezien als het commando dat binnen de container moet worden uitgevoerd.

We kunnen het “kubectl exec” ook gebruiken voor een interactieve sessie met de pod op te starten:

```

student@virt-k8s-XX:~$ kubectl exec -it nginx-pod2 -- bash
root@nginx-pod2:/# apt update

```

```

Hit:1 http://deb.debian.org/debian trixie InRelease
Hit:2 http://deb.debian.org/debian trixie-updates InRelease
Hit:3 http://deb.debian.org/debian-security trixie-security
InRelease

Reading package lists... Done
root@hello-pod:/# apt upgrade

Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done

0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
root@nginx-pod2:/# exit

exit
student@virt-k8s-XX:~$
```

11.6 Kubectl

11.6.1 Algemeen

Bij het aanmaken van onze eerste pods, hebben we reeds een paar keer leren werken met het kubectl cmd. Dit is het standaard tool voor het werken met een K8s cluster.

Deze tool heeft enorm veel mogelijkheden, dewelke opgebouwd zijn volgens een duidelijke syntax.

kubectl [command] [TYPE] [NAME] [flags]

- command
Commando of operatie, bv apply, create, delete, get
- type
Het type van de resource of object
- name
De naam van de resource of object
- flags
Optionele vlaggen

Voorbeelden:

- kubectl create -f mypod.yaml
- kubectl get pods
- kubectl get pod mypod

- kubectl delete pod mypod

De installatie van de cluster gebeurt in ons geval met k3d, terwijl het werken met de functionaliteit van de cluster via kubectl verloopt.

11.6.2 Context

Bij het onderdeel over het opzetten van clusters zagen we dat het mogelijk is om meerdere Kubernetes-clusters op één server te draaien.

Met kubectl kun je met al deze clusters werken via zogenoemde contexten.

Een context (werkomgeving) is een configuratie-item in de kubeconfig die een specifieke combinatie beschrijft van:

- een cluster (met zijn API-server),
- een gebruiker (voor authenticatie (authenticatie wordt verder niet besproken)),
- en een namespace (dit bespreken we later).

Door een context te gebruiken weet kubectl precies:

met welk cluster, namens welke gebruiker en binnen welke namespace het moet communiceren.

Zo kun je eenvoudig wisselen tussen verschillende omgevingen (bijvoorbeeld de dev, test en prod-omgeving) zonder telkens je configuratie te moeten aanpassen.

We laten de volledige cubeconfig zien.

```
student@serverXX:~$ kubectl config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://0.0.0.0:45273
    name: k3d-dev-cluster
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://0.0.0.0:44735
    name: k3d-test-cluster
contexts:
- context:
    cluster: k3d-dev-cluster (1)
    user: admin@k3d-dev-cluster (2)
    name: k3d-dev-cluster (3)
- context:
    cluster: k3d-test-cluster
    user: admin@k3d-test-cluster
    name: k3d-test-cluster
current-context: k3d-test-cluster
kind: Config
preferences: {}
users:
- name: admin@k3d-dev-cluster
  user:
    client-certificate-data: DATA+OMITTED
    client-key-data: DATA+OMITTED
```

```

- name: admin@k3d-test-cluster
  user:
    client-certificate-data: DATA+OMITTED
    client-key-data: DATA+OMITTED

```

1. cluster: k3d-dev-cluster
Dit geeft aan met welk cluster deze context verbinding maakt.
Het verwijst naar één van de clusters die gedefinieerd zijn onder clusters.
In jouw geval: het cluster dat draait op https://0.0.0.0:44909.
2. user: admin@k3d-dev-cluster
Dit geeft aan welke gebruiker (authenticatie) gebruikt wordt om toegang te krijgen tot het cluster.
Het verwijst naar een entry in users: in de kubeconfig.
We hebben dit niet ingesteld.
3. name: k3d-dev-cluster
Dit is de naam van de context zelf.
Een context is een combinatie van cluster + user (+ namespace optioneel).
Je gebruikt deze naam om te wisselen van context

Toont de naam van de actieve context.

```
student@serverXX:~$ kubectl config current-context
k3d-test-cluster
```

We tonen nu een overzicht van alle contexts.

```
student@serverXX:~$ kubectl config get-contexts
CURRENT      NAME          CLUSTER          AUTHINFO          ...
            k3d-dev-cluster   k3d-dev-cluster   admin@k3d-dev-cluster ...
*           k3d-test-cluster   k3d-test-cluster   admin@k3d-test-cluster
```

Zoals verwacht zie je dat k3d-test-cluster nu actief is.

We switchen nu naar de context k3d-dev-cluster.

```
student@serverXX:~$ kubectl config use-context k3d-dev-cluster
Switched to context "k3d-dev-cluster".
```

As je nu de pods opvraagt krijg je onderstaande.

```
student@serverXX:~$ kubectl get pods
NAME        READY   STATUS    RESTARTS   AGE
nginx-pod   1/1     Running   0          64m
```

We willen nu bijvoorbeeld een nieuwe context maken met de naam dev-cluster, met gebruiker admin@k3d-dev-cluster en namespace default.

```
student@serverXX:~$ kubectl config set-context dev-context --
cluster=k3d-dev-cluster --user=admin@k3d-dev-cluster --
namespace=default
```

We tonen nu terug een overzicht van alle contexts.

```
student@serverXX:~$ kubectl config get-contexts
CURRENT   NAME          CLUSTER      AUTHINFO ...
           dev-context    k3d-dev-cluster admin@k3d-dev-cluster ...
*         k3d-dev-cluster  k3d-dev-cluster  admin@k3d-dev-cluster
           k3d-test-cluster k3d-test-cluster admin@k3d-test-cluster
```

We switchen terug naar de context k3d-test-cluster.

```
student@serverXX:~$ kubectl config use-context k3d-test-cluster
Switched to context "k3d-first-pod".
```

11.6.3 CRUD

Kubectl maakt het mogelijk om de vier basisbewerkingen binnen een Kubernetes-cluster uit te voeren volgens het CRUD-principe:

- Create (aanmaken)
- Read (uitlezen)
- Update (bijwerken)
- Delete (verwijderen)

CRUD-actie	Betekenis	Voorbeeld commando's
Create	Maak een resource aan	kubectl apply -f <naam>.yaml kubectl create -f <naam>.yaml
Read	Bekijk bestaande resources	kubectl get pods kubectl describe svc my-service
Update	Pas een bestaande resource aan	kubectl edit deployment my-app kubectl apply -f updated.yaml
Delete	Verwijder een resource	kubectl delete pod my-pod kubectl delete -f deployment.yaml

11.6.3.1 Create

Voor het aanmaken van een object binnen K8s kan men gebruik maken van 2 commando's:

- `kubectl create`
`kubectl create <type> <parameters>`
`kubectl create -f <path to manifest>`

Dit commando zal een resource aanmaken op een imperatieve wijze (via de stdin) of door het aanleveren van een manifest file door gebruik te maken van de -f parameter.

- `kubectl apply`
`kubectl apply -f <path to manifest>`

Het commando `kubectl apply -f <path naar manifest>` maakt, net als het `create`-commando, een resource aan op basis van de opgegeven manifestfile. Het verschil is dat `apply` ook wordt gebruikt om wijzigingen aan een bestaande resource door te voeren, waarbij het de configuratie bijwerkt zonder de hele resource te verwijderen en opnieuw aan te maken.

11.6.3.2 Read

Kubectl laat ook toe om details op te vragen van de verschillende resources die men heeft aangemaakt.

```
kubectl get <type>
kubectl get <type> <name>
kubectl get <type> <name> -o <output format>
```

Voorbeeld:

```
student@virt-k8s-XX:~$ kubectl get pod
NAME        READY   STATUS    RESTARTS   AGE
nginx-pod2  1/1     Running   0          22h

student@virt-k8s-XX:~$ kubectl get pod -o wide
NAME        READY   STATUS    RESTARTS   AGE      IP           ...
nginx-pod2  1/1     Running   0          22h     10.42.1.7 ...
```

11.6.3.3 Update

Met `kubectl edit <type> <object name>` kunnen we resources aanpassen. Dit commando zal een kopie van het object openen in een teksteditor om aan te passen.

Voorbeeld:

```
kubectl edit pod nginx-pod2 #voer best niet uit/verlaten: typ :q
```

Omdat handmatige wijzigingen meestal niet worden vastgelegd in versiebeheer of manifestbestanden, wordt het gebruik van kubectl edit in productieomgevingen meestal afgeraden. Het kan leiden tot configuratieafwijkingen die lastig te reproduceren of traceren zijn.

11.6.3.4 Delete

De laatste CRUD bewerking die we hebben, is het verwijderen van resources. Dit doen we aan de hand van het volgende commando:

```
kubectl delete <type> <name>
```

Voorbeeld:

```
student@virt-k8s-XX:~$ kubectl get pod
NAME        READY   STATUS    RESTARTS   AGE
nginx-pod2  1/1     Running   0          4s

student@virt-k8s-XX:~$ kubectl delete pod nginx-pod2
pod "nginx-pod2" deleted

student@virt-k8s-XX:~$ kubectl get pod
No resources found in default namespace.
```

11.6.4 Kubectl cheat sheet

<https://Kubernetes.io/docs/reference/kubectl/cheatsheet/>

11.7 Namespaces

11.7.1 Basis

In Kubernetes dienen namespaces om resources logisch te groeperen en te isoleren binnen één cluster. Ze zijn als het ware virtuele omgevingen binnen hetzelfde fysieke (of virtuele) cluster.

Wanneer je een nieuw cluster opstelt, zijn er standaard vier namespaces aanwezig:

Namespace	Doel
default	De standaard namespace. Als je geen -n opgeeft bij een kubectl-commando, gebruikt Kubernetes automatisch deze namespace. Alle “gewone” resources komen hier terecht, tenzij je anders specificeert.

kube-system	Hier draait de infrastructuur van Kubernetes zelf, zoals de API-server, kube-dns (CoreDNS), scheduler, controller-manager, enz. Je hoort hier meestal niet zelf applicaties te plaatsen.
kube-public	Bevat publieke informatie, zoals cluster-informatie, die voor iedereen toegankelijk is (wordt zelden gebruikt).
kube-node-lease	Wordt intern door Kubernetes gebruikt om node leases bij te houden. Dit helpt de control plane snel te detecteren of een node nog leeft. (Elke node krijgt hier een "lease object").

```
student@virt-k8s-XX:~$ kubectl get namespaces
```

NAME	STATUS	AGE
default	Active	17m
kube-node-lease	Active	17m
kube-public	Active	17m
kube-system	Active	17m

Namespaces maken het mogelijk om je Kubernetes-cluster logisch te partitioneren. Dit helpt je om:

- Meerdere teams veilig in één cluster te laten werken zonder elkaar's resources te beïnvloeden.
- Verschillende omgevingen zoals ontwikkeling (dev), testen (test) en productie (prod) te scheiden.
- Resource quota's en beleidsregels (policies) af te dwingen.
- Het beheer van je cluster overzichtelijk en beheersbaar te houden.

Zo krijg je betere isolatie, toegangscontrole en resourcebeheer binnen één shared clusteromgeving.

Een nieuwe namespace aanmaken doe je door gebruik te maken van het kubectl create ns <name>

Voorbeeld:

```
student@virt-k8s-XX:~$ kubectl create ns pwl
namespace/pwl created
```

```
student@virt-k8s-XX:~$ kubectl get namespaces
```

NAME	STATUS	AGE
------	--------	-----

default	Active	22m
kube-node-lease	Active	22m
kube-public	Active	22m
kube-system	Active	22m
pxl	Active	27s

Uiteraard kan je ook een namespace aanmaken op de declaratieve wijze. Hiervoor volstaat het om volgende YAML (ns.yaml) file aan te maken:

```
apiVersion: v1
kind: Namespace
metadata:
  name: dev
```

Vervolgens moeten we deze nog aanmaken:

```
student@virt-k8s-XX:~$ kubectl apply -f ns.yaml
namespace/dev created
```

```
student@virt-k8s-XX:~$ kubectl get namespaces
NAME      STATUS   AGE
default   Active   25h
dev       Active   16s
kube-node-lease Active  25h
kube-public  Active  25h
kube-system  Active  25h
pxl       Active   5m13s
```

11.7.2 Resources toewijzen aan NS

By default zal elke pod (of elke andere resource) die je aanmaakt in de default namespace worden geplaatst. Indien je met verschillende namespaces werkt, is dit niet meteen het gewenste resultaat. Je wilt immers je pods (of andere resources) in specifieke namespaces plaatsen. Afhankelijk van welke wijze je pods (of andere resources) aanmaakt, is hiervoor een andere werkwijze nodig:

- Imperatief
Indien je bijvoorbeeld een pod wilt aanmaken in de dev namespace moet je dit aan de hand van de -n parameter.

```
student@virt-k8s-XX:~$ kubectl run nginx-dev --image nginx -n
dev
pod/nginx-dev created
```

- Declaratief

Bij het aanmaken van de manifest file volstaat het om bij de metadata de correcte namespace te specificeren.

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  namespace: dev
spec:
  containers:
  - name: nginx
    image: nginx
```

11.7.3 Resources opvragen in NS

Ook bij het opvragen van de details van de resource moeten we steeds de -n parameter meegeven. Kubectl zal immers steeds in de default namespace zoeken naar de resources die hij wil opvragen.

```
student@virt-k8s-XX:~$ kubectl get pods
kubectl run nginx-dev --image nginx -n dev
```

```
student@virt-k8s-XX:~$ kubectl get pods -n dev
NAME        READY   STATUS    RESTARTS   AGE
nginx-dev   1/1     Running   0          4m39s
```

Indien je met meerdere namespaces aan het werken bent, kan het gebruik van de -n parameter snel lastig worden om mee te werken. Zeker indien je verschillende commando's wilt uitvoeren voor een bepaalde namespace. Je kan echter je kubeconfig aanpassen zodat deze standaard in een bepaalde namespace werkt. Dit doe je aan de hand van het volgende cmd:

```
kubectl config set-context --current --namespace <ns-name>
```

Voorbeeld:

```
student@virt-k8s-XX:~$ kubectl config set-context --current --
namespace dev
Context "k3d-test-nginx" modified.
```

```
student@virt-k8s-XX:~$ kubectl get pods
NAME        READY   STATUS    RESTARTS   AGE
nginx-dev   1/1     Running   0          9m18s
```

Zorg wel dat je steeds goed beseft in welke namespace je aan het werken bent.

Je kan ook alle pods in alle namespaces als volgt opvragen.

```
student@virt-k8s-XX:~$ kubectl get pods --all-namespaces
```

...

11.8 Services

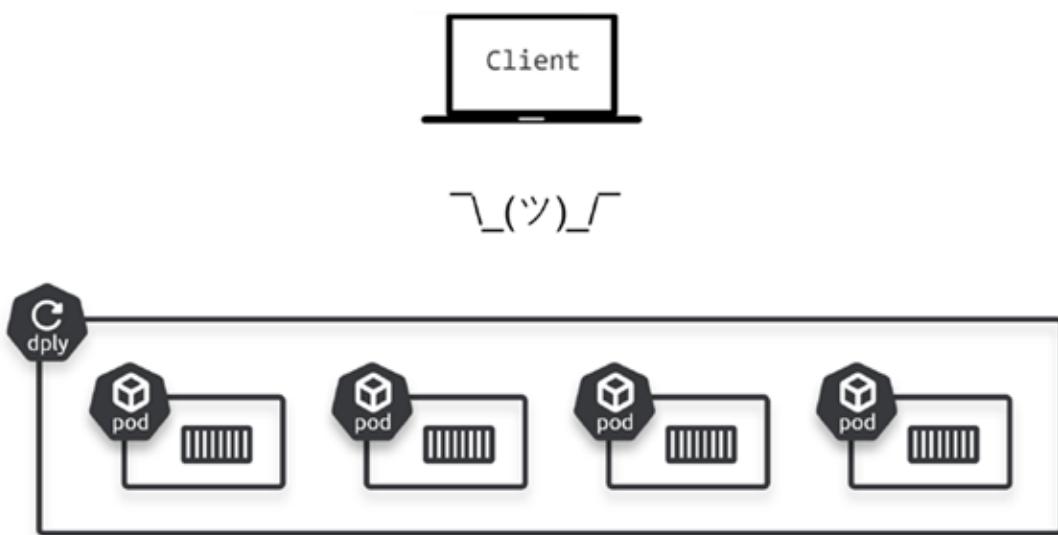
11.8.1 Algemeen

Hoewel het netwerk binnen een Kubernetes-cluster standaard plat en open is, waardoor alle pods rechtstreeks met elkaar kunnen communiceren in een cluster, is het niet aan te raden om pods direct via IP-adressen te laten communiceren.

Stel dat je een pod hebt waarop een website draait en een andere pod waarop een database draait. In de configuratie van de website kun je het IP-adres van de database-pod gebruiken om verbinding te maken. Dit werkt, omdat pods in een Kubernetes-cluster standaard onbeperkt met elkaar kunnen communiceren.

Echter, als je de database-pod vervolgens verwijdert en opnieuw aanmaakt — wat gebruikelijk is bij updates of scaling — dan krijgt de nieuwe pod een ander IP-adres. Hierdoor verliest je website de verbinding, omdat het nog steeds naar het oude IP verwijst.

Daarnaast, als je meerdere website-pods wil draaien voor schaalbaarheid, moet je een manier hebben om die pods collectief te benaderen, bijvoorbeeld door een enkele toegangspoort voor externe gebruikers te creëren.

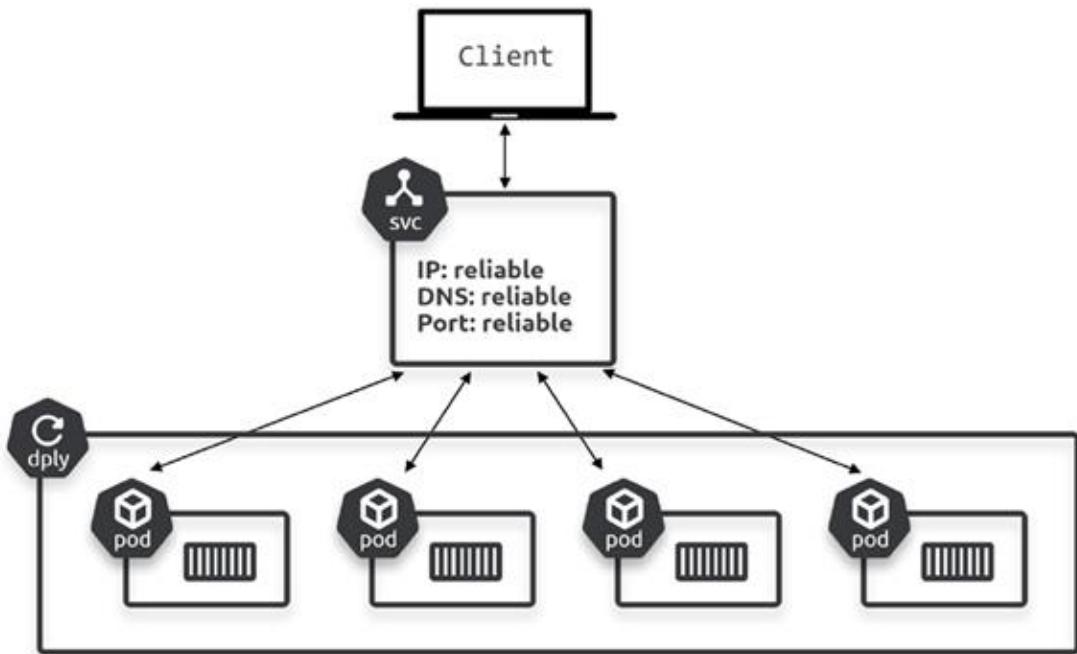


Om de problemen uit de vorige voorbeelden op te lossen, maak je binnen Kubernetes gebruik van Services. Services zorgen ervoor dat pods binnen het cluster altijd op een consistente en betrouwbare manier bereikbaar zijn, zelfs als de pods dynamisch veranderen, bijvoorbeeld door opschalen of herstarten.

Een Service is een speciaal type resource dat fungeert als een stabiel aanspreekpunt voor een groep pods. Dit maakt het mogelijk om pods te bereiken via een onveranderlijk netwerkendpoint, onafhankelijk van de levenscyclus van de individuele pods.

Elke Service heeft een aantal vaste eigenschappen die gedurende de hele levensduur van de Service blijven bestaan:

- Een vast (clusterbreed) IP-adres
- Een consistente DNS-naam binnen het cluster
- Een of meerdere TCP- of UDP-poorten waardoor de service bereikbaar is



Door gebruik te maken van een Service in Kubernetes beschik je over een betrouwbaar en stabiel netwerkendpoint dat toegang biedt tot een groep pods, ongeacht veranderingen in het aantal pods achter deze Service. Dit betekent dat het aantal pods naadloos kan schalen – omhoog of omlaag – zonder dat gebruikers of verbonden applicaties iets hoeven te veranderen.

Een kracht van een Service is dat het automatisch zorgt voor load balancing. Alle pods die gekoppeld zijn aan een Service via labels worden continu gemonitord. Als een pod niet meer gezond is, zal de Service deze automatisch uitsluiten van de load balancing, zodat alleen gezonde pods verkeer ontvangen.

Kubernetes kent verschillende servicetypes met elk hun eigen kenmerken en gebruik:

Type	Toegang vanaf	Gebruik
ClusterIP	Alleen binnen het cluster	Standaardtype, voor interne communicatie tussen pods en services

NodePort	Buiten het cluster via node-IP en poort	Eenvoudige externe toegang, vaak voor lokale of testomgevingen
LoadBalancer	Buiten het cluster via extern IP	Cloud-native toegang met geïntegreerde load balancer (bv. AWS, GCP, Azure)

Hoewel het Kubernetes-netwerk standaard open is, is het altijd aan te raden om een Service te definiëren wanneer een Pod via specifieke poorten bereikbaar moet zijn

11.8.2 Endpoint slices

Zoals eerder besproken voert een Service in Kubernetes verkeer-distributie (load balancing) uit over de achterliggende pods. Hiervoor heeft de Service een actuele lijst nodig van actieve en gezonde endpoints (pods) die bij de Service horen.

Deze lijst wordt samengesteld op basis van labelselectie en vastgelegd in EndpointSlices.

Wanneer een Service wordt aangemaakt, creëert Kubernetes automatisch één of meer bijbehorende EndpointSlices. Deze bevatten subsets van de IP-adressen en poorten van de pods die via de labelselector aan de Service gekoppeld zijn.

De control plane (met name de kube-controller-manager en endpoint controller) houdt continu bij welke pods aan de labelselector voldoen. Nieuwe pods worden aan de EndpointSlices toegevoegd zodra ze beschikbaar zijn. Verwijderde of niet-gezonde pods worden eruit verwijderd.

Op deze manier zorgt Kubernetes ervoor dat de lijst met actieve en gezonde pods altijd up-to-date is — essentieel voor betrouwbare load balancing.

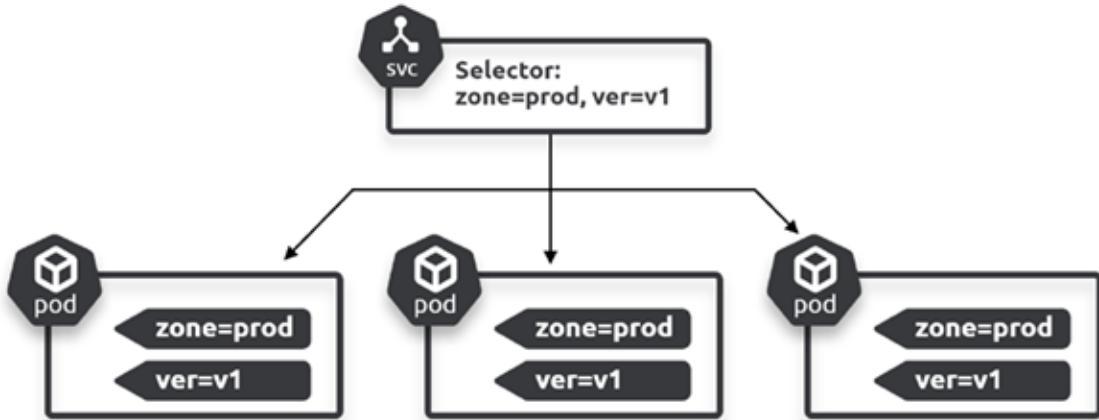
Let op: alleen pods die Ready zijn, worden in de EndpointSlices opgenomen.

11.8.3 Services aanmaken

Het aanmaken van een Service kan op twee manieren, maar wij richten ons hier uitsluitend op de declaratieve aanpak met manifestbestanden.

Voordat we ingaan op het maken van een Service, is het belangrijk te begrijpen hoe een Service wordt gekoppeld aan pods. In Kubernetes koppelen we altijd pods aan een Service, nooit andersom.

Deze koppeling gebeurt via labels.



Zolang de manifestfile van een pod de juiste labels bevat die overeenkomen met de labelselector van een Service, zal de pod automatisch worden geregistreerd bij die Service en opgenomen worden in de load balancing.

Bovenaan zie je een Service (svc) met een labelselector: zone=prod, ver=v1.

Onder de Service staan drie pods, elk voorzien van dezelfde labels: zone=prod en ver=v1.

Hiermee wordt duidelijk dat het labelmatchen tussen pods en services de sleutel is tot automatische koppeling en load balancing binnen Kubernetes.

11.8.3.1 Cluster IP

Zoals aangehaald in het overzicht van soorten services zorgt dit soort van service enkel voor communicatie in een cluster tussen pods en andere services. Volgende manifestfile svc.yaml zorgt voor het aanmaken van een ClusterIP-service.

```
apiVersion: v1
kind: Service (1)
metadata:
  name: nginx-svc (2)
spec:
  type: ClusterIP (3)
  selector:
    app: nginx (4)
  ports:
    - port: 8080 (5)
      targetPort: 80 (6)
      protocol: TCP
```

1. kind: Service
Dit geeft aan wat voor soort Kubernetes-object je aanmaakt.
In dit geval is het een Service

2. name: nginx-svc
De naam van de Service.
Deze naam wordt gebruikt:
 - om de Service in de cluster te identificeren;
 - en als DNS-naam binnen de cluster.
3. type: ClusterIP
Bepaalt hoe de Service toegankelijk is binnen of buiten de cluster.
ClusterIP → enkel binnen de cluster bereikbaar (standaard).
4. App: nginx
Dit koppelt de Service aan de juiste Pods. Dit betekent dat alle Pods die het label app=nginx hebben, behoren tot deze Service.
5. port: 8080
De poort waarop de Service binnen de cluster luistert.
Andere Pods binnen dezelfde namespace kunnen dus verbinden naar nginx-svc:8080.
6. targetPort: 80
De poort in de container (of Pod) waarop de applicatie effectief luistert.
De Service stuurt inkomend verkeer op port: 8080 door naar targetPort: 80 in de Pods.
client → nginx-svc:8080 → (doorsturen) → nginx-pod:80

Vervolgens moeten we ook nog een pod aanmaken met de effectieve server (svc-nginx.yaml).

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: nginx
spec:
  containers:
    - name: nginx
      image: nginx
      ports:
        - containerPort: 80
```

Daarnaast maken we ook nog een gewone debian pod aan dewelke we gaan gebruiken als client (svc-debian.yaml).

```
apiVersion: v1
kind: Pod
metadata:
  name: debian-pod
spec:
  containers:
    - name: debian
      image: debian:latest
      command: ["sleep", "infinity"]
      tty: true
```

The image displays three terminal windows side-by-side, each showing a different YAML configuration file:

- Top Terminal:** Shows the `svc.yaml` file. It defines a Service named `nginx-svc` with type `ClusterIP`. The selector `app: nginx` matches the Pod. A port `port: 8080` is mapped to `targetPort: 80` and `protocol: TCP`.
- Middle Terminal:** Shows the `svc-nginx.yaml` file. It defines a Pod named `nginx-pod` with a single container named `nginx` running the `nginx` image. The container port is `containerPort: 80`.
- Bottom Terminal:** Shows the `svc-deb.yaml` file. It defines a Pod named `debian-pod` with a single container named `debian` running the `debian:latest` image. The command for the container is `["sleep", "infinity"]` and `tty: true`.

Nu moeten we onze deployments ook daadwerkelijk uitvoeren. Het is aan te raden om eerst de Service te deployen en daarna de Pods.

```
student@virt-k8s-XX:~$ kubectl apply -f svc.yaml
```

```
service/nginx-svc created
```

```
student@virt-k8s-XX:~$ kubectl get svc -o wide
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE	SELECTOR
nginx-svc	ClusterIP	10.43.126.13	<none>	8080/TCP	73s	app=nginx

```
student@virt-k8s-XX:~$ kubectl apply -f svc-deb.yaml
```

```
pod/debian-pod created
```

```
student@virt-k8s-XX:~$ kubectl apply -f svc-nginx.yaml
```

```
pod/nginx-pod created
```

```
student@virt-k8s-XX:~$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
debian-pod	1/1	Running	0	33s

```
nginx-pod     1/1     Running     0          5m37s
```

We kunnen nu de Debian pod gebruiken om onze service te testen.

```
student@virt-k8s-XX:~$ kubectl exec -it debian-pod -- sh  
  
# apt update && apt install -y curl  
# curl http://nginx-svc:8080  
  
<!DOCTYPE html>  
  
<html>  
  
<head>  
  
<title>Welcome to nginx!</title>  
  
<style>  
  
...  

```

Uit de bovenstaande output blijkt duidelijk dat de Service (nginx-svc) correct functioneert. De Debian-pod kan via de interne DNS-naam nginx-svc:8080 succesvol verbinding maken met de Nginx-pod, wat bevestigt dat de Service goed werkt en het verkeer correct doorstuurt.

11.8.3.2 NodePort

11.8.3.2.1 Basis

Waar een ClusterIP enkel binnen de cluster een vast aanspreekpunt biedt voor interne communicatie, gaat een NodePort een stap verder.

Met een NodePort maak je de service namelijk ook extern bereikbaar — via een specifieke poort op elke node in de cluster.

In het onderstaande voorbeeld zie je hoe je een NodePort-service kunt aanmaken met behulp van een manifestbestand (nodeport.yaml).

```
apiVersion: v1  
  
kind: Service (1)  
  
metadata:  
  
  name: nginx-srv2 (2)  
  
spec:
```

```

type: NodePort (3)

selector:

app: nginx (4)

ports:

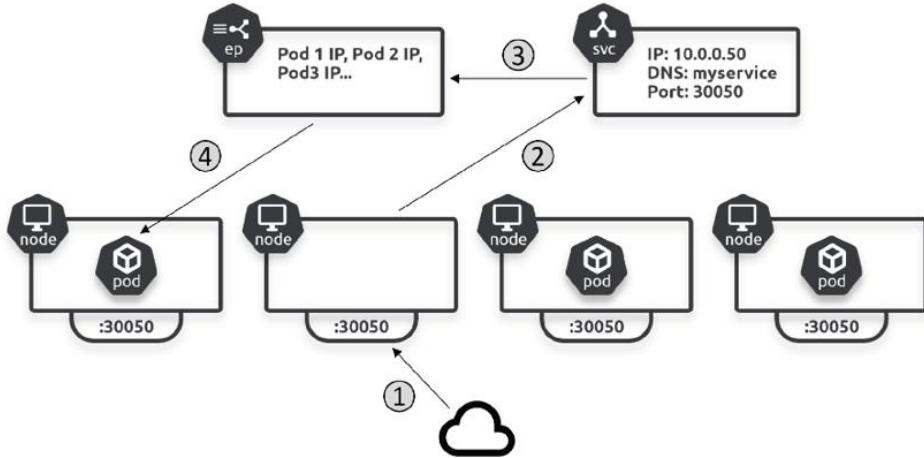
  - port: 8080 (5)

    targetPort: 80 (6)

    nodePort: 30050 (7)

```

1. **kind: Service**
Dit geeft aan wat voor soort Kubernetes-object je aanmaakt.
In dit geval is het een Service
2. **name: nginx-srv2**
De naam van de Service.
Deze naam wordt gebruikt:
 - om de Service in de cluster te identificeren;
 - en als DNS-naam binnen de cluster.
3. **type: NodePort**
Bepaalt hoe de Service toegankelijk is binnen of buiten de cluster.
NodePort → maakt de Service bereikbaar via een vaste poort op elke node, ook van buiten de cluster.
4. **App: nginx**
Dit koppelt de Service aan de juiste Pods. Dit betekent dat alle Pods die het label `app=nginx` hebben, behoren tot deze Service.
5. **port: 8080**
Dit is de poort waarop de Service luistert binnen de cluster.
Andere Pods (of Services) kunnen via `demo-node:8080` verbinding maken.
6. **targetPort: 80**
De poort in de container (of Pod) waarop de applicatie effectief luistert.
De Service stuurt inkomend verkeer op port: 8080 door naar targetPort: 80 in de Pods.
7. **nodePort: 30050**
Dit is de poort op elke node van de cluster waarmee de Service van buiten de cluster bereikbaar is.
Externe clients kunnen verbinding maken via `<NodeIP>:30050`, waarna het verkeer automatisch wordt doorgestuurd naar de Service (port: 8080) en vervolgens naar de bijbehorende Pod (targetPort: 30050).
Als je nodePort niet expliciet opgeeft, kiest Kubernetes automatisch een poort in het bereik 30000–32767. Je mag zelf ook geen poorten buiten dat bereik kiezen voor de nodePort.
Door zelf een nodePort te definiëren, weet je precies welke poort extern bereikbaar is, wat handig is voor firewalls of voorspelbare toegang.



Wanneer je deze NodePort Service deployt, wordt op elke node in het cluster poort 30050 geopend. Via deze poort kunnen externe clients het cluster binnengaan. De communicatie verloopt volgens de volgende stappen, zoals in de afbeelding:

- Een externe client maakt verbinding met het IP-adres van een willekeurige node op poort 30050.
- Het inkomende verkeer wordt ontvangen door de NodePort Service op die node.
- De Service raadpleegt de bijbehorende EndpointSlice, waar een actuele lijst van gezonde pods (endpoints) voor deze Service wordt bijgehouden.
- Op basis van load balancing stuurt de Service de aanvraag door naar een van de gezonde pods (1 in tekening), ongeacht op welke node deze pod draait.

Belangrijk: het maakt niet uit op welke node je cluster binnengaat; het verkeer wordt altijd op dezelfde manier verwerkt en gespreid over de beschikbare, gezonde pods. Welke pod de aanvraag uiteindelijk afhandelt, wordt bepaald door de loadbalancing van de Service.

Je mag het IP-adres van eender welke worker node gebruiken omdat NodePort-services in Kubernetes op alle nodes van het cluster dezelfde externe poort openen.

Nu moeten we onze deployments ook daadwerkelijk uitvoeren.

```
student@virt-k8s-XX:~$ kubectl apply -f nodeport.yaml
service/nginx-srv2 created
```

Bij k8s zou onderstaande nu werken.

```
student@serverXX:~$ curl localhost:30050
curl: (7) Failed to connect to localhost port 30050 after 0 ms:
Could not connect to server
```

11.8.3.2.2 K3d en nodeports

De fout treedt op omdat we met k3d werken... Bij een echte K8s installatie zou het dus werken.

Kubernetes zelf draait in Docker-containers bij k3d. Je kan dat eenvoudig checken.

```
student@serverXX:~$ docker ps --filter name=k3d

CONTAINER ID        IMAGE               COMMAND
207bbb1703fa      ghcr.io/k3d-io/k3d-proxy:5.8.3   "/bin/sh -c nginx-pr..." ...
3964528f5a9e      rancher/k3s:v1.31.5-k3s1       "/bin/k3d-entrypoint..." ...
ce4898349338      rancher/k3s:v1.31.5-k3s1       "/bin/k3d-entrypoint..." ...
5173339da660      ghcr.io/k3d-io/k3d-proxy:5.8.3   "/bin/sh -c nginx-pr..." ...
c2d2fac65183      rancher/k3s:v1.31.5-k3s1       "/bin/k3d-entrypoint..." ...
4d3afdc4107f      rancher/k3s:v1.31.5-k3s1       "/bin/k3d-entrypoint..." ...
```

Daardoor zit de NodePort-routing binnen dat containernetwerk, en is poort 30050 niet automatisch beschikbaar op je VM-host tenzij je die poort expliciet hebt gepubliceerd.

Je kan dat gelukkig (op 2 manieren) oplossen...

- Optie A

Voor kubectl port-forward op de service zoals hieronder staat om te forwarden.

```
student@serverXX:~$ kubectl port-forward svc/nginx-srv2
30050:8080 -n dev &
```

```
[1] 44857
```

```
student@serverXX:~$ Forwarding from 127.0.0.1:30050 -> 80
Forwarding from [::1]:30050 -> 80
```

```
student@serverXX:~$ curl localhost:30050
```

```
Handling connection for 30050
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<title>Welcome to nginx!</title>
```

```
...
```

Het werkt 😊.

- Optie B

Als je persistent NodePort-toegang wil vanaf de VM (zonder port-forward na reboot), maak (of hermaak) je de cluster waarbij je de po(o)rt(en) publiceert op de loadbalancer.

We zullen hiervoor eerst de bestaande clusters verwijderen (om niet te veel resources te gebruiken).

```
student@serverXX:~$ k3d cluster list

NAME          SERVERS   AGENTS   LOADBALANCER
dev-cluster    1/1       1/1      true
first-pod      1/1       1/1      true

student@serverXX:~$ k3d cluster delete dev-cluster
...
student@serverXX:~$ k3d cluster delete test-cluster
...
```

We maken nu een nieuwe cluster aan met publicatie van poort 30050 via de k3d loadbalancer.

```
student@serverXX:~$ k3d cluster create mycluster -p
30050:30050@server:0
...
...
```

We voeren nu de nodige deployments uit.

```
student@serverXX:~$ kubectl apply -f  nodeport.yaml
service/nginx-srv2 created
student@serverXX:~$ kubectl apply -f  svc-nginx.yaml
pod/nginx-pod created
```

We trachten nu de webpagina op te vragen.

```
student@serverXX:~$ curl localhost:30050
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...
...
```

Ook dit werkt 😊.

11.8.3.3 Loadbalancer

LoadBalancer Services bieden eenvoudige externe toegang tot je Kubernetes workloads door automatisch een internet-facing load balancer in te stellen via je cloudprovider. Je krijgt een publiek IP-adres of DNS-naam die altijd bereikbaar is en die automatisch verkeer verdeelt naar je Service. Hierdoor hoef je geen IP-adressen van individuele nodes te beheren en kun je eventueel een eigen, herkenbare DNS-naam koppelen voor gemakkelijke toegang.

Let op: LoadBalancer Services zijn alleen beschikbaar op cloudplatforms die dit ondersteunen, zoals AWS, Azure of GCP.

In deze cursus focussen we ons alleen op ClusterIP en NodePort, omdat een LoadBalancer Service in een lokale testomgeving meestal niet beschikbaar is.

11.9 Deployments

11.9.1 Algemeen

Een Deployment is een Kubernetes-resource waarmee je eenvoudig applicaties uitrolt, bijwerkt en beheert. Je definieert hierin hoeveel pods je wilt draaien, en Kubernetes zorgt ervoor dat dit aantal altijd up-to-date blijft (self-healing). Bij wijzigingen aan je applicatie voert een Deployment gecontroleerde updates uit (rolling updates) en maakt het eventueel terugdraaien (rollback) mogelijk. Elke microservice krijgt idealiter een eigen Deployment, zodat je elke service onafhankelijk kunt updaten en schalen voor maximale flexibiliteit.

11.9.2 ReplicaSets

Deployments gebruiken ReplicaSets om het gewenste aantal pods te beheren. ReplicaSets zorgen ervoor dat altijd het juiste aantal werkende pods actief is: als er één uitvalt, wordt er automatisch een nieuwe gestart. Ze maken zo self-healing en eenvoudig schalen mogelijk.

11.9.3 Voorbeeld

11.9.3.1 Basis

Een deployment resource in Kubernetes maak je aan met een YAML-manifestbestand. Dit manifest beschrijft precies hoe jouw applicatie uitgerold moet worden: hoeveel pods je wilt, welke container-image er gebruikt wordt, en op welke poort je app bereikbaar is.

```
apiVersion: apps/v1  
kind: Deployment (1)  
  
metadata:  
  name: nginx-deployment (2)  
  
spec:
```

```

replicas: 2 (3)

selector:

matchLabels:

app: nginx

template: (4)

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:latest

ports:

- containerPort: 80

```

1. kind: Deployment

Hiermee geef je aan dat het object een Deployment is.

2. name: nginx-deployment

Dit is de unieke naam voor de Deployment binnen de namespace. Gebruik een geldige DNS-naam zonder spaties of speciale tekens.

3. replicas: 2

Hiermee geef je aan hoeveel exemplaren van de pod je wilt draaien. Hier zijn dat er 2, wat zorgt voor redundantie en eventueel high availability.

4. template:

Dit is een pod-template. Het beschrijft hoe elke pod eruit moet zien die door de Deployment wordt gemaakt. Hier definieer je onder andere:

- Labels die bepalen welke pods bij deze deployment horen.
- De container(s) met naam, image en poortinstellingen.

In Kubernetes hoef je niet per se een apart manifest voor een Pod te maken als je al een deployment manifest hebt. Een deployment bevat namelijk een pod-template dat beschrijft hoe elke pod eruit moet zien. Kubernetes gebruikt deze template om automatisch de juiste Pods te creëren en te beheren.

In onderstaand stappenplan gaan we aan de slag met de basis deployment file zoals hierboven uitgelegd:

1. Verwijder alle clusters.

```
student@virt-k8s-XX:~$ k3d cluster list  
student@virt-k8s-XX:~$ k3d cluster delete ...
```

2. Maak nieuwe cluster aan voor de test.

```
student@virt-k8s-XX:~$ k3d cluster create mycluster --agents 1
```

3. Vervolgens maken we een yaml file aan met de inhoud van bovenstaand voorbeeld.

```
student@virt-k8s-XX:~$ nano dep.yaml
```

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

```
  name: nginx-deployment
```

```
spec:
```

```
  replicas: 2
```

```
  selector:
```

```
    matchLabels:
```

```
      app: nginx
```

```
  template:
```

```
    metadata:
```

```
      labels:
```

```
        app: nginx
```

```
  spec:
```

```
    containers:
```

```
      - name: nginx
```

```
        image: nginx:latest
```

```
      ports:
```

```
        - containerPort: 80
```

4. Als laatste stap volstaat het om deze nieuw aangemaakt manifest file te deployen.

```
student@virt-k8s-XX:~$ kubectl apply -f dep.yaml
```

```
deployment.apps/nginx-deployment created
```

Ten alle tijden kan men de status van een bepaalde deployment nagaan. Dit doen we door gebruik te maken van volgende cmd:

```
student@virt-k8s-XX:~$ kubectl get deployment nginx-deployment
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   2/2      2           2          90s
```

Zoals je kan zien hebben we hier een duidelijk overzicht van de status van de deployment met een aantal belangrijke statistieken:

- READY = geeft aan hoeveel pods er reeds klaar zijn van het totaal aantal replica's dat we hebben gespecificeerd.
- UP-TO-DATE = Welke pods de laatste image hebben volgens de manifest file (later zien we meer over updates via een deployment)
- AVAILABLE = hoeveel pods er beschikbaar zijn.
- AGE = hoelang de deployment reeds aan het draaien is (vanaf eerste apply)

In onze manifest file hebben we aangegeven dat we 2 replica's willen voor deze deployment. Dit kunnen we controleren op volgende manier via de pods:

```
student@virt-k8s-XX:~$ kubectl get pod
NAME                           READY   STATUS    RESTARTS   AGE
nginx-deployment-54b9c68f67-8sqwj   1/1     Running   0          4m3s
nginx-deployment-54b9c68f67-nfhlz   1/1     Running   0          4m3s
```

11.9.3.2 Self-healing

Een van de belangrijkste verantwoordelijkheden van een deployment was de self-healing functie van de pods. Dit wil zeggen dat wanneer een bepaalde pod niet meer correct draait, de deployment processen er automatisch voor zullen zorgen dat we zo snel als mogelijk terug in de state zitten die in de manifest file staat aangegeven. In dit geval 2 replica's. Deze functionaliteit kunnen we ook zelf uittesten door manueel een pod te verwijderen. K8s zal automatisch een nieuwe pod aanmaken.

```
student@virt-k8s-XX:~$ kubectl delete pod nginx-deployment-...
pod "nginx-deployment-54b9c68f67-8sqwj" deleted
```

```
student@virt-k8s-XX:~$ kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
nginx-deployment-54b9c68f67-m6f9h   1/1     Running   0          21s
```

```
nginx-deployment-54b9c68f67-nfh1z    1/1      Running     0          8m10s
```

Zoals je duidelijk kunt zien, heeft de self-healing-functie ingegrepen en automatisch een nieuwe pod aangemaakt.

11.9.3.3 Aanpassen deployment

Het grote voordeel van een Deployment is dat je altijd de gewenste staat (desired state) van je applicatie kan aanpassen. Dit kan op twee manieren:

- Imperatief, door rechtstreeks commando's te gebruiken (bijvoorbeeld via kubectl scale).
- Declaratief, door de manifest file aan te passen en deze opnieuw toe te passen met kubectl apply.

Gebruik bij voorkeur altijd de declaratieve manier.

Als je namelijk imperatieve commando's gebruikt, pas je de deployment aan in de cluster, maar niet de manifest file. Hierdoor ontstaat er een mismatch: bij een volgende apply van de manifest file kan je deployment teruggezet worden naar een oude configuratie.

De gouden regel is daarom om altijd veranderingen in de manifest file door te voeren en deze opnieuw te apply'en.

Als je bijvoorbeeld het aantal replica's wil wijzigen van 2 naar 6 pas je dit aan in de manifest file en voer je daarna "kubectl apply" uit. Kubernetes herkent aan de hand van de naam dat het dezelfde deployment is en voert alleen de gewijzigde instellingen door.

```
student@virt-k8s-XX:~$ kubectl get deployment
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
nginx-deployment	2/2	2	2	16m

We zullen nu instellen dat er 6 replica's moeten zijn.

```
student@virt-k8s-XX:~$ nano dep.yaml
```

...

spec:

replicas: 6

...

We voeren de wijzigingen door.

```
student@virt-k8s-XX:~$ kubectl apply -f dep.yaml
```

```
deployment.apps/nginx-deployment configured
```

We checken de nieuwe situatie.

```
student@virt-k8s-XX:~$ kubectl get deployment
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
nginx-deployment	6/6	6	6	20m

```
student@virt-k8s-XX:~$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
nginx-deployment-54b9c68f67-dpc98	1/1	Running	0	2m16s
nginx-deployment-54b9c68f67-g2q9m	1/1	Running	0	2m16s
nginx-deployment-54b9c68f67-j57vx	1/1	Running	0	2m16s
nginx-deployment-54b9c68f67-l2srk	1/1	Running	0	2m16s
nginx-deployment-54b9c68f67-m6f9h	1/1	Running	0	13m
nginx-deployment-54b9c68f67-nfh1z	1/1	Running	0	21m

11.9.3.4 Verwijderen deployment

Indien je geen gebruik meer wenst te maken van een bepaalde deployment, kan men deze verwijderen door gebruik te maken volgend cmd:

```
student@virt-k8s-XX:~$ kubectl delete deployment nginx-deployment  
deployment.apps "nginx-deployment" deleted  
student@virt-k8s-XX:~$ kubectl get deployment  
No resources found in default namespace.
```

Het kan even duren voor de pods weg zijn.

```
student@virt-k8s-XX:~$ kubectl get pods  
No resources found in default namespace.
```

Men kan ook een deployment aan de hand van de manifest file verwijderen:

```
student@virt-k8s-XX:~$ kubectl apply -f dep.yaml  
deployment.apps/nginx-deployment created  
student@virt-k8s-XX:~$ kubectl get deployment
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
nginx-deployment	6/6	6	6	20m

nginx-deployment	6/6	6	6	14s
------------------	-----	---	---	-----

```
student@virt-k8s-XX:~$ kubectl delete -f dep.yaml
deployment.apps "nginx-deployment" deleted
```

Het verwijderen van een deployment is definitief. Alle pods worden verwijderd. Let dus steeds goed op dat je de juiste deployment verwijderd.

11.9.4 Rolling updates

We zullen eerst een agent toevoegen aan de cluster mycluster.

```
student@serverXX:~$ k3d node create agent-02 --role agent --
-cluster mycluster
INFO[0000] Adding 1 node(s) to the runtime local cluster
'mycluster'...
INFO[0000] Using the k3d-tools node to gather environment
information
INFO[0000] Starting new tools node...
INFO[0000] Starting node 'k3d-mycluster-tools'
INFO[0000] HostIP: using network gateway 172.18.0.1 address
INFO[0000] Starting node 'k3d-agent-02-0'
INFO[0003] Successfully created 1 node(s)!
```

```
student@serverXX:~$ k3d cluster list
```

NAME	SERVERS	AGENTS	LOADBALANCER
mycluster	1/1	2/2	true

Naast het aanpassen van het aantal replica's voor de self-healing functionaliteit, heeft men ook de mogelijkheid om aan de hand van een deployment rolling updates uit te voeren. Dit is een manier van updaten waarbij men ervoor zorgt dat de microservice nooit down zal gaan. Er wordt maar steeds een klein aantal van de replica's geüpdatet. Zo blijft men doorgaan tot de hele deployment is geüpdatet. Om dit mogelijk te maken in een deployment moet men een manifest file (noem die dep-rolling.yaml) aanmaken met bepaalde instellingen:

```
apiVersion: apps/v1
kind: Deployment (1)
```

```
metadata: (2)
  name: nginx-deployment
  annotations:
    kubernetes.io/change-cause: "Initial release"
spec:
  replicas: 10 (3)
  selector: (4)
  matchLabels:
    app: nginx
  revisionHistoryLimit: 5 (5)
  progressDeadlineSeconds: 300 (6)
  minReadySeconds: 10 (7)
  strategy: (8)
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1 (9)
      maxSurge: 1 (10)
  template:
    metadata:
      labels:
        app: nginx
  spec:
    containers:
      - name: nginx
        image: nginx:1.24
    ports:
```

– **containerPort: 80**

1. kind: Deployment
Dit geeft aan dat het een Deployment-resource betreft.
2. Metadata:
Hierin staan metadata over het Deployment-object, zoals de naam en eventuele labels of annotaties voor identificatie en management. Meer uitleg hier later over.
3. replicas: 10
Het gewenste aantal gelijke pods dat altijd actief moet zijn. Hier zijn dat er 10.
4. Selector:
Dit is een label-selector die bepaalt welke pods onder beheer van deze Deployment vallen. De labels in .spec.template.metadata.labels moeten hieraan matchen.
5. revisionHistoryLimit: 5
Het aantal oude ReplicaSets dat wordt bewaard om terug te kunnen rollen naar een vorige versie.
6. progressDeadlineSeconds: 300
De maximale tijd in seconden waarin een update als succesvol moet worden beschouwd. Lukt dat niet, dan slaat de Deployment status met een foutmelding.
7. minReadySeconds: 10
Hoe lang een pod "ready" moet zijn voordat deze als beschikbaar wordt gezien en bijvoorbeeld oude pods worden verwijderd.
8. strategy:
De update-strategie die voor de Deployment wordt gebruikt. RollingUpdate zorgt voor gefaseerde updates met minimale downtime.
9. maxUnavailable: 1
Het maximum aantal pods dat tijdens een update tijdelijk niet beschikbaar mag zijn (hier 1).
10. maxSurge: 1
Het maximum aantal extra pods dat tijdelijk boven het gewenste replica-aantal mag draaien tijdens een update (hier 1).

Alvorens we deze manifest toepassen is het noodzakelijk om even wat dieper in te gaan op de rollingupdate strategie. Hierin hebben we gespecificeerd dat we maar maximum 1 pod willen hebben die boven de desired state van 10 replica's is aan de hand van de maxSurge parameter. Dit wil zeggen dat er dus nooit meer dan 11 pods tegelijk mogen draaien tijdens het update proces. Daarnaast hebben we de maxUnavailable parameter aangegeven dat we nooit minder dan 9 pods willen hebben tijdens de update. De combinatie van deze twee parameters zal ervoor zorgen dat er maximum 2 pods tegelijk zullen worden geüpdatet.

Indien je je manifest aanpast met bovenstaande wijzigingen zal de deployment gewoon naar 10 replica's worden geschaald. Er zullen echter geen updates worden uitgevoerd daar de image versie nog steeds dezelfde is in container template. De updates zullen enkel uitgevoerd worden volgens de aangegeven strategie indien de image versie aangepast wordt of er een andere image voor deze deployment wordt gebruikt.

Wij gaan er van uit dat de deployment correct verloopt en er 10 replica's in onze deployment draaien met versie 1.0:

```
student@virt-k8s-XX:~$ kubectl apply -f dep-rolling.yaml
```

```
student@virt-k8s-XX:~$ kubectl get deployment
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   0/10    10           0          15s
```

Na enige tijd...

```
student@virt-k8s-XX:~$ kubectl get deployment
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   10/10   10           10         30s
```

We checken nu of de pods gestart zijn.

```
student@virt-k8s-XX:~$ kubectl get pods
NAME                               READY   STATUS    RESTARTS   AGE
nginx-deployment-64dd99d679-2z8zd   1/1     Running   0          45s
...
nginx-deployment-64dd99d679-zqpfq   1/1     Running   0          45s
```

Om het update process nu correct te starten gaan we de manifestfile aanpassen naar een nieuwere image. Pas hiervoor dep-rolling.yaml aan.

```
image: nginx:1.26
```

Daarnaast gaan we ook bij de metadata in de manifest file de annotation aanpassen hetgeen we later kunnen gebruiken om eventueel snel terug te draaien naar een vorige versie. Pas de manifest file als volgt aan:

```
kubernetes.io/change-cause: "Update naar nginx:1.26"
```

Omdat we nu een aanpassing hebben gemaakt aan de manifest file gaan we deze terug apply'en.

```
student@virt-k8s-XX:~$ kubectl apply -f dep-rolling.yaml
deployment.apps/nginx-deployment configured
```

De updates die nu gestart zijn, kunnen we volgen. Hiervoor kan je gebruik maken van het cmd "kubectl rollout status deployment". Dit zal op de console steeds aangeven wat de status is van de rollout. Om dit te stoppen volstaat het om "CTRL+C" in te drukken.

```
student@virt-k8s-XX:~$ kubectl rollout status deployment nginx-deployment
```

```
Waiting for deployment "nginx-deployment" rollout to finish: 2
out of 10 new replicas have been updated...
```

```
Waiting for deployment "nginx-deployment" rollout to finish: 2
out of 10 new replicas have been updated...
```

```
Waiting for deployment "nginx-deployment" rollout to finish: 2
out of 10 new replicas have been updated...
```

```
Waiting for deployment "nginx-deployment" rollout to finish: 1
old replicas are pending termination...
```

...

```
deployment "nginx-deployment" successfully rolled out
```

We tonen toont een overzicht van alle deployments in de huidige namespace van je Kubernetes cluster.

```
student@virt-k8s-XX:~$ kubectl get deployment
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
nginx-deployment	10/10	10	10	9m16s

In dit voorbeeld is het gehele update proces snel gedaan. Echter sommige updates kunnen heel wat tijd in beslag nemen, zeker wanneer je met grote deployments bezig bent. Daarom kan het soms noodzakelijk zijn om een rolling update even te pauzeren en terug te starten. Dit doe je door gebruik te maken van volgende twee cmd's:

```
kubectl rollout pause deploy <deployment-name>
```

```
kubectl rollout resume deploy <deployment-name>
```

11.9.5 Rollback

Het kan altijd voorkomen dat een bepaalde update voor problemen zorgt. In dat geval is het vaak noodzakelijk dat we een rollback doen naar een vorige versie. Dit kunnen we zoals altijd op 2 manieren doen:

- Imperatieve wijze (door gebruik van cmd's)
- Declaratieve wijze (door de manifest file terug aan te passen)

11.9.5.1 Imperatieve manier

Indien er echt een heel snelle rollback moet gebeuren, kan je altijd terug draaien naar een vorige versie door gebruik te maken van het kubectl rollout undo deployment cmd.

Alvorens we dit commando kunnen gebruiken, moeten we eerst achterhalen naar welke revision (replicaset) we de deployment willen terugdraaien. Hiervoor kan je gebruik maken van het kubectl rollout history deployment cmd.

```
student@virt-k8s-XX:~$ kubectl rollout history deployment nginx-deployment  
deployment.apps/nginx-deployment  
REVISION  CHANGE-CAUSE  
1          Initial release  
2          Update naar nginx:1.26
```

Nu wordt ook meteen duidelijk waarom we in de manifest file gebruik hebben gemaakt van de annotations. Deze geven ons bij het history cmd een duidelijk beeld van wat er in een bepaalde revision is gebeurd. Met dit commando zien we nu duidelijk dat we over 2 revisions beschikken en dus moeten terugdraaien naar revision 1. Om dit te bekomen volstaat het om het kubectl rollout undo deployment verder aan te vullen met de correcte revision voor de desbetreffende revision.

```
student@virt-k8s-XX:~$ kubectl rollout undo deployment nginx-deployment --to-revision=1  
deployment.apps/nginx-deployment rolled back
```

De opdracht kubectl rollout history deployment nginx-deployment toont een overzicht van alle revisies (versies) van het opgegeven deployment, inclusief de wijzigingsredenen.

```
student@virt-k8s-XX:~$ kubectl rollout history deployment nginx-deployment  
deployment.apps/nginx-deployment  
REVISION  CHANGE-CAUSE  
2          Update naar nginx:1.26  
3          Initial release
```

We tonen toont een lijst van alle pods in de huidige namespace, inclusief hun status, aantal containers dat draait, herstarts en leeftijd (tijdens rollback).

```
student@virt-k8s-XX:~$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
nginx-deployment-64dd99d679-585f5	1/1	Running	0	11s
nginx-deployment-64dd99d679-9mkgc	1/1	Running	0	23s
nginx-deployment-64dd99d679-b5s4h	1/1	Running	0	11s
nginx-deployment-64dd99d679-cx629	1/1	Running	0	34s
nginx-deployment-64dd99d679-gjrmgb	1/1	Running	0	23s
nginx-deployment-64dd99d679-jnjpz	1/1	Running	0	45s
nginx-deployment-64dd99d679-jwlt7	0/1	ContainerCreating	0	0s
nginx-deployment-64dd99d679-k59tm	0/1	ContainerCreating	0	0s
nginx-deployment-64dd99d679-lbpvd	1/1	Running	0	45s
nginx-deployment-64dd99d679-wxc98	1/1	Running	0	34s
nginx-deployment-75578ddfcc-2fbff	1/1	Running	0	10m
nginx-deployment-75578ddfcc-b6zxk	0/1	Completed	0	10m
nginx-deployment-75578ddfcc-r2z6l	1/1	Terminating	0	11m

Na enige tijd runnen alle pods uiteraard.

We tonen nu de replicasets.

```
student@virt-k8s-XX:~$ kubectl get rs
```

NAME	DESIRED	CURRENT	READY	AGE
nginx-deployment-64dd99d679	10	10	10	18m
nginx-deployment-75578ddfcc	0	0	0	11m

Door de twee laatste commando's zien we duidelijk de werking van de replicasets. Per update wordt er een replicaset gemaakt om zo een geschiedenis op te bouwen van revisions. In de manifest file hebben wij aangegeven dat we maar 5 revisions willen bijhouden. Hiermee bedoelen we unieke revisions die effectief een verandering teweeg hebben gebracht. Bij de eerste release is de eerste Replicaset opgebouwd, bij de tweede release is er een tweede replicaset opgebouwd met de nieuwe versies. Wanneer we echter een rollback hebben gedaan is nu geen derde nieuwe replicaset opgebouwd maar heeft K8s intern gewoon een rollback gedaan naar de vorige replicaset (of de replicaset waarnaar we hebben verwezen). Dit is zelfs correct gereflecteerd in de history. Daar zien

we dat bij revision 3 we terug op de initial release zitten. Hiermee hebben een duidelijke kracht van het gebruik van de replicaset aangetoond.

Deze manier van werken geniet, zoals reeds aangehaald, niet de voorkeur. Door het gebruik van imperatieve cmd's wordt de manifest file niet aangepast en heeft dit nu een potentieel gevaar gevormd. Als een andere engineer nu de manifest file terug deployed is deze deployment helemaal niet in de desired state dat we wensen.

11.9.5.2 Declaratieve manier

Beter is om de versie aan te passen in de manifest file en terug te deployen. Dit zorgt ervoor dat het manifest steeds up to date is.

Pas de deployment file terug aan naar de oorspronkelijke situatie:

```
student@virt-k8s-XX:~$ nano dep-rolling.yaml
```

...

```
kubernetes.io/change-cause: "Initial release"
```

...

```
image: nginx:1.24
```

...

```
student@virt-k8s-XX:~$ kubectl apply -f dep-rolling.yaml
```

```
deployment.apps/nginx-deployment configured
```

```
student@virt-k8s-XX:~$ kubectl get deployment
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
nginx-deployment	10/10	10	10	35m

```
student@virt-k8s-XX:~$ kubectl rollout history deployment nginx-deployment
```

```
deployment.apps/nginx-deployment
```

```
REVISION CHANGE-CAUSE
```

```
2 Update naar nginx:1.26
```

3 Initial release

De initiele deployment is weer aan het draaien. We passen de manifest file nu terug aan zodat we versie 2 aan het draaien zijn en apply'en de manifest file.

```
student@virt-k8s-XX:~$ nano dep-rolling.yaml
```

...

```
Kubernetes.io/change-cause: "Update naar nginx:1.26"
```

...

```
image: nginx:1.26
```

...

```
student@virt-k8s-XX:~$ kubectl apply -f dep-rolling.yaml
```

```
deployment.apps/nginx-deployment configured
```

```
student@virt-k8s-XX:~$ kubectl rollout history deployment nginx-deployment
```

```
deployment.apps/nginx-deployment
```

```
REVISION  CHANGE-CAUSE
```

```
3  Initial release
```

```
4  Update naar nginx:1.26
```

Wacht nu even tot de update volledig is afgerond. Dit kan je altijd nagaan door de status van de deployment te raadplegen:

```
student@virt-k8s-XX:~$ kubectl get deployment
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
nginx-deployment	10/10	10	10	13m

Pas nu de manifest file terug aan door zowel de image op nginx:1.24 te zetten en de annotation aan te passen met een passende boodschap vb: 'Rollback to nginx 1.24 due to bug'. Vervolgens de manifest file terug apply'en.

```
student@virt-k8s-XX:~$ nano dep-rolling.yaml
```

...

```
Kubernetes.io/change-cause: "Rollback to nginx 1.24 due to  
bug"
```

...

```
image: nginx:1.24
```

...

```
student@virt-k8s-XX:~$ kubectl apply -f dep-rolling.yaml  
deployment.apps/nginx-deployment configured
```

```
student@virt-k8s-XX:~$ kubectl rollout history deployment nginx-deployment
```

```
deployment.apps/nginx-deployment
```

```
REVISION  CHANGE-CAUSE
```

```
4        Update naar nginx:1.26
```

```
5        Rollback to nginx 1.24 due to bug
```

```
student@virt-k8s-XX:~$ kubectl get deployment
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
nginx-deployment	10/10	10	10	17m

11.10 Storage

Containers hebben vaak een kort leven. Als een container crasht, start Kubernetes automatisch een nieuwe container om de app beschikbaar te houden.

De data van de vorige container gaat daarbij verloren, want elke container heeft zijn eigen geïsoleerd bestandssysteem.

Voorbeeld opslag in filesystem van container:

```

apiVersion: v1
kind: Pod
metadata:
  name: temp-data-demo
spec:
  containers:
  - name: app
    image: busybox
    command: ['sh', '-c', 'echo "data" > /tmp/file.txt && sleep 3600']

```

Wanneer we deze zouden deployen zal er data worden weggeschreven in de /tmp/file.txt. Echter zodra de pod sterft zal deze data niet meer bestaan en zal de nieuwe pod starten met een nieuwe geïsoleerd filesystem. Gegevens opgeslagen in eigen, privé filesystem van containers gaan altijd verloren als de container crasht of de pod opnieuw opstartt.

Dit zouden we kunnen tegengaan door gebruik te maken van tijdelijke volumes (ephemeral volumes). Nemen we bijvoorbeeld volgende pod:

```

apiVersion: v1
kind: Pod
metadata:
  name: pod-with-volume
spec:
  containers:
  - name: app
    image: busybox
    command: ['sh', '-c', 'echo "data" > /data/file.txt && sleep 3600']
    volumeMounts:
    - name: my-volume
      mountPath: /data
  volumes:
  - name: my-volume
    emptyDir: {} # Simple empty directory that survives container restarts

```

In deze volume mount wordt het volume my-volume gekoppeld aan het pad /data binnen de container.

In de volumes sectie wordt my-volume aangemaakt als type emptyDir: {}.

Het emptyDir volume wordt aangemaakt zodra de Pod op een node wordt gestart. Aanvankelijk is het volume leeg.

Het volume blijft bestaan zolang de Pod actief is, ongeacht het starten of crashen van individuele containers binnen die Pod. Hierdoor blijft de data behouden bij containerrestarts, maar niet als de volledige Pod verwijderd wordt.

11.10.1 Soorten volumes

Kubernetes maakt onderscheid tussen container filesystem, tijdelijke volumes en persistente opslag. Het container filesystem is privé voor de container en verdwijnt bij een herstart. Tijdelijke volumes bestaan zolang de Pod leeft en verdwijnen zodra de Pod stopt, terwijl persistente opslag data bewaart onafhankelijk van de levenscyclus van de Pod.

- Container filesystem

Het standaard geïsoleerde bestandssysteem van een container. Data hierin gaat verloren zodra de container herstart, crasht of de Pod eindigt. Niet gedeeld tussen containers en niet geschikt voor persistente opslag.

- Tijdelijke volumes (ephemeral volumes)

Tijdelijke volumes bestaan zolang de Pod leeft.

Ze worden beheerd door Kubernetes en zijn gedeeld tussen containers binnen dezelfde Pod. Ze verdwijnen volledig wanneer de Pod wordt verwijderd.

- emptyDir

Een lege, tijdelijke map die Kubernetes aanmaakt wanneer de Pod start.

De map wordt gedeeld tussen containers in de Pod en verdwijnt zodra de Pod stopt.

- configMap

Een volume dat configuratiegegevens bevat.

Beschikbaar als alleen-lezen bestanden voor containers.

Niet bedoeld voor langdurige opslag; data komt uit de ConfigMap-resource.

- Secret

Een volume dat gevoelige gegevens (zoals wachtwoorden of tokens) bevat.

Wordt als beveiligde, alleen-lezen bestanden aangeboden aan containers.

- downwardApi

Een volume waarmee Pod-informatie (bijvoorbeeld naam, labels, resource limits) als bestanden beschikbaar wordt gemaakt in de container.

Handig voor applicaties die metadata van de Pod nodig hebben.

- Persistent storage

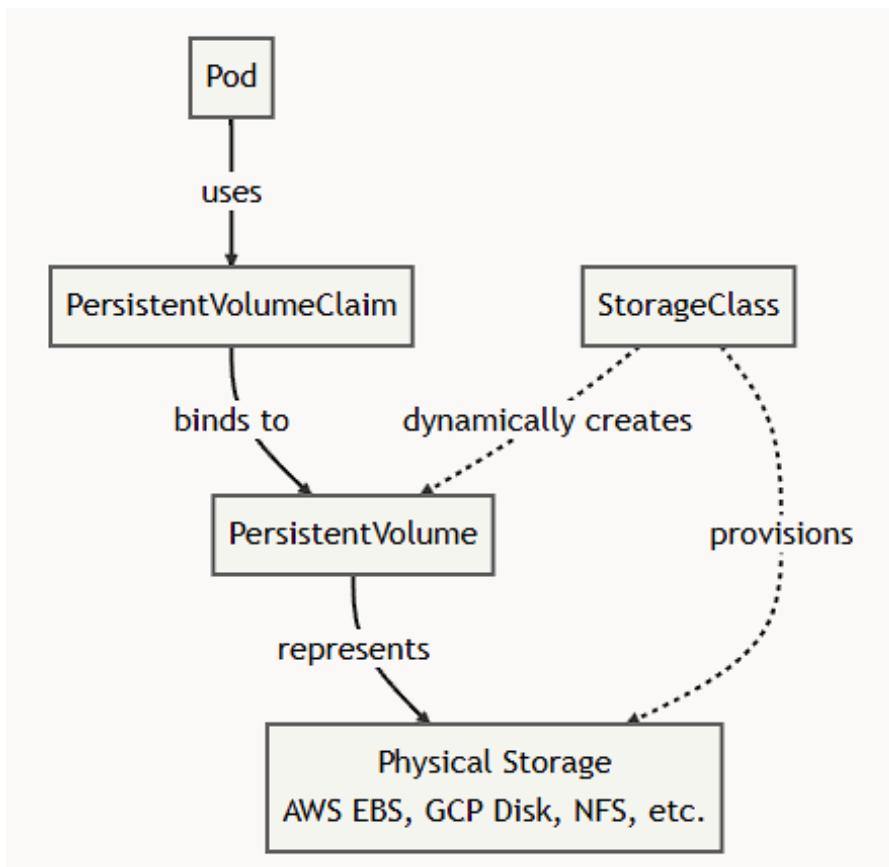
- PersistentVolume (PV)

Dit is een stuk opslagruijte dat door een beheerder is gecreëerd en beschikbaar is in het cluster, onafhankelijk van de levensduur van een Pod. Het vertegenwoordigt de fysieke opslagresource.

- PersistentVolumeClaim (PVC)

Dit is het verzoek van een gebruiker of applicatie om toegang te krijgen tot een PersistentVolume met bepaalde eigenschappen zoals grootte en toegangsmodus.

- StorageClass
Dit is een object waarmee de beheerder opslagklassen definieert met specifieke eigenschappen (zoals prestaties of back-upbeleid). Een StorageClass maakt dynamische provisioning van PersistentVolumes mogelijk, waardoor volumes automatisch worden aangemaakt wanneer een PVC wordt ingediend. Een StorageClass is niet verplicht maar wel aanbevolen.



Deze drie objecten werken samen om ervoor te zorgen dat applicaties duurzaam en flexibel opslag kunnen gebruiken binnen een Kubernetes-cluster.

Opgelet: persistent data wordt verwijderd wanneer de cluster wordt verwijderd tenzij je bij het aanmaken van de cluster

11.10.2 PersistentVolume (PV)

Een persistent volume vertegenwoordigt de effectieve opslag in een cluster. Dit volume is onafhankelijk van een specifieke Pod, wat betekent dat de data in het Persistent Volume behouden blijft zelfs als de Pod die er gebruik van maakt stopt, verwijderd wordt of opnieuw wordt gestart.

Hieronder zien we een voorbeeld van een persistent volume

```

apiVersion: v1
kind: PersistentVolume (1)
metadata:
  
```

```

    name: my-pv (2)
  spec:
    capacity:
      storage: 10Gi (3)
    accessModes:
      - ReadWriteOnce (4)
  persistentVolumeReclaimPolicy: Retain (5)
  hostPath:
    path: /mnt/data (6)

```

1. kind: PersistentVolume
We definiëren een Persistent volume.
2. name: my-pv
De naam van het volume is my-pv.
3. storage: 10Gi
Het volume stelt 10 gigabyte opslagruimte beschikbaar.
4. ReadWriteOnce
Dit betekent dat het volume door één node tegelijk voor lezen en schrijven kan worden gebruikt.
5. persistentVolumeReclaimPolicy: Retain
Dit bepaalt dat de opslag behouden blijft nadat de PersistentVolumeClaim (PVC) losgekoppeld is; data wordt dus niet automatisch verwijderd.
6. path: /mnt/data
Dit volume maakt gebruik van een fysieke map op de node zelf als opslag. Dit is alleen geschikt voor testomgevingen of eenvoudige setups, want bij productiesystemen worden meestal netwerk- of cloudopslag gebruikt.

11.10.3 PersistentVolumeClaim (PVC)

Een PersistentVolumeClaim is eigenlijk een verzoek om toegang te krijgen tot een bepaald volume. Hierbij zal men ook steeds een access mode moeten aangeven alsook laten weten hoeveel ruimte we effectief willen gebruiken van dat bepaald volume.

```

apiVersion: v1
kind: PersistentVolumeClaim (1)
metadata:
  name: my-pvc (2)
spec:
  accessModes:
    - ReadWriteOnce (3)
  resources:
    requests:
      storage: 8Gi (4)

```

1. kind: PersistentVolumeClaim
We definiëren een PVC.

2. name: my-pvc
De naam van deze aanvraag voor opslag.
3. ReadWriteOnce
Dit geeft aan dat de opslag later slechts door één node tegelijk gelezen en beschreven mag worden.
4. storage: 8Gi
De PVC opslagruimte vraagt minimaal 8 GiB.

Met bovenstaande volumeClaim kan je nu aan de slag om te gebruiken in bijvoorbeeld een pod:

```
apiVersion: v1
kind: Pod (1)
metadata:
  name: my-pod (2)
spec:
  containers:
    - name: app
      image: nginx
      volumeMounts:
        - name: storage (3)
          mountPath: /usr/share/nginx/html (4)
  volumes:
    - name: storage
      persistentVolumeClaim:
        claimName: my-pvc (5)
```

1. kind: Pod
Dit beschrijft, zoals jullie weten, een Pod.
2. name: my-pod
De naam van de Pod, namelijk "my-pod".
3. name: storage
Geeft aan dat een volume met de naam "storage" binnen de container wordt gekoppeld.
4. mountPath: /usr/share/nginx/html
Dit is de locatie in de container waar het volume wordt gemount, in dit geval de webroot voor Nginx.
5. claimName: my-pvc
Een PersistentVolumeClaim met de naam "my-pvc" wordt gebruikt.

11.10.4 StorageClass

Een StorageClass maakt het mogelijk om persistent volumes automatisch te creëren wanneer je een PersistentVolumeClaim aanmaakt. Dit concept wordt vooral veel gebruikt in cloudomgevingen. Cloudomgevingen vallen buiten de doelstellingen van dit hoofdstuk.

Bij k3s is standaard al een default StorageClass aanwezig (zie verder), genaamd local-path, die de lokale opslag van de nodes gebruikt.

11.10.5 Access modes

Perstistant volumes kunnen drie verschillende toegangsniveaus (access modes) hebben:

- **ReadWriteOnce (RWO)**
Volume kan gemount worden in Read-Write mode, maar enkel door 1 node.
- **ReadOnlyMany (ROX)**
Volume kan gemount worden als Read-Only door meerdere nodes.
- **ReadWriteMany (RWX)**
Volume kan gemount worden als Read-Write door meerdere nodes.

In de praktijk hangt de beschikbare access mode sterk af van het type opslag (storage) dat achter het volume zit. Niet elk opslagtype ondersteunt bijvoorbeeld gelijktijdige toegang door meerdere nodes.

11.10.6 Reclaim Policies

Bij het verwijderen van een PersistentVolume (PV) bepaalt de persistentVolumeReclaimPolicy wat er met de bijbehorende storage gebeurt. Er zijn twee mogelijke opties:

- **Delete:** de storage wordt automatisch verwijderd wanneer het PV wordt verwijderd (dit is de standaardoptie voor PV's die via een StorageClass automatisch worden aangemaakt)..
- **Retain:** de storage blijft bestaan, ook nadat de PV is verwijderd.

Voorbeeld in een PV-configuratie:

```
persistentVolumeReclaimPolicy: Retain
```

Opgelet: deze instelling geldt voor alle PV's: zowel degenen die handmatig als via storageclass zijn aangemaakt.

11.10.7 Voorbeelden

Om de verschillende types van storage duidelijk te maken, gaan we aan de slag met een nieuwe cluster. Om zoveel mogelijk resources te besparen, verwijder je best alle andere clusters. Zodra je dat gedaan hebt, kan je aan de slag om een nieuwe cluster aan te maken.

```
student@virt-k8s-XX:~$ k3d cluster list
student@virt-k8s-XX:~$ k3d cluster delete ...
student@virt-k8s-XX:~$ k3d cluster create storage-lab --agents 2
INFO[0000] Prep: Network
INFO[0000] Created network 'k3d-storage-lab'
```

```
student@virt-k8s-XX:~$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
				Pagina 329

k3d-storage-lab-agent-0	Ready	<none>	80s	v1.31.5+k3s1
k3d-storage-lab-agent-1	Ready	<none>	80s	v1.31.5+k3s1
k3d-storage-lab-server-0	Ready	control-plane,master	83s	v1.31.5+k3s1

We bekijken eerst de default storageclass is bij de cluster:

student@virt-k8s-XX:~\$ kubectl get storageclass				
NAME	PROVISIONER	RECLAIMPOLICY
local-path (default)	rancher.io/local-path	Delete

Het gaat hier om een lokale opslag StorageClass die dynamische provisioning voorziet binnen het cluster. Dit is typisch in test- of developmentomgevingen met lokale opslag.

```
student@virt-k8s-XX:~$ kubectl get storageclass local-path -o yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    defaultVolumeType: local
...

```

Dit wil zeggen dat wanneer er een PVC gemaakt wordt zonder StorageClassName, de default storageclass zal gebruikt worden.

11.10.7.1 Opslag in container filesystem

Soms kan het handig zijn om specifiek vluchtige storage te gebruiken. Hiervoor moeten we eerst even bestuderen hoe dit effectief in zijn werk gaat.

Maak een nieuwe manifest file (temptest.yaml) aan:

```
apiVersion: v1
kind: Pod
metadata:
  name: temp-data-pod
spec:
  containers:
    - name: writer
```

```
image: busybox
command: ['sh', '-c', 'while true; do echo "Log entry at $(date)" >> /tmp/app.log; sleep 5; done']
```

Deploy deze pod.

```
student@virt-k8s-XX:~$ kubectl apply -f temp-data-pod.yaml
kubectl apply -f temp-data-pod.yaml
```

Kijk naar de log file die wordt opgebouwd.

```
student@virt-k8s-XX:~$ kubectl exec temp-data-pod -- tail -f /tmp/app.log
Log entry at Sun Nov 16 11:32:56 UTC 2025
Log entry at Sun Nov 16 11:33:01 UTC 2025
^C
```

Bekijk de inhoud van de map /tmp/ binnen de container van de Pod temp-data-pod.

```
student@virt-k8s-XX:~$ kubectl exec temp-data-pod -- ls -la /tmp/
total 4
drwxrwxrwt    1 root      root            21 Nov  1 11:38 .
drwxr-xr-x    1 root      root           62 Nov  1 11:38 ..
-rw-r--r--    1 root      root          2310 Nov  1 11:43 app.log
```

Verwijder nu de pod.

```
student@virt-k8s-XX:~$ kubectl delete -f temp-data-pod.yaml
pod "temp-data-pod" deleted
```

Start pod metzelfde instellingen opnieuw op.

```
student@virt-k8s-XX:~$ kubectl apply -f temp-data-pod.yaml
pod/temp-data-pod created
```

Kijk terug naar de log file die wordt opgebouwd.

```
student@virt-k8s-XX:~$ kubectl exec temp-data-pod -- tail -f /tmp/app.log
Log entry at Sun Nov 16 11:41:07 UTC 2025
Log entry at Sun Nov 16 11:41:12 UTC 2025
^C
```

Zoals je kan zien, wordt het bestand gevuld met nieuwe entry's (kijk naar het uur).

11.10.7.2 Persistent storage - Manueel

Het eerste dat we moeten doen, is een Persistent Volume aanmaken. Daarvoor maak je een YAML-bestand met de naam manual-pv.yaml, waarin je de specificaties van het volume definieert

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: manual-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: manual
  hostPath:
    path: /tmp/k3d-manual-pv
```

Daarna moet je ook een PersistentVolumeClaim manifest aanmaken. Wij geven het de naam manual-pvc.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: manual-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: manual
```

Deze beide gaan we nu apply'en wat verder in detail bekijken.

```
student@virt-k8s-XX:~$ kubectl apply -f manual-pv.yaml
persistentvolume/manual-pv created

student@virt-k8s-XX:~$ kubectl apply -f manual-pvc.yaml
persistentvolumeclaim/manual-pvc created
```

We tonen nu een overzicht van alle Persistent Volumes (PV's) die in je Kubernetes-cluster zijn aangemaakt.

```
student@virt-k8s-XX:~$ kubectl get pv
```

NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY	...
manual-pv	1Gi	RWO	Retain	...

We tonen nu een overzicht van alle PersistentVolumeClaims (PVC's) in de Kubernetes-cluster.

```
student@virt-k8s-XX:~$ kubectl get pvc
```

NAME	STATUS	VOLUME	CAPACITY	...
manual-pvc	Bound	manual-pv	1Gi	...

We vragen nu meer details op over PVC manual-pvc.

```
student@virt-k8s-XX:~$ kubectl describe pvc manual-pvc
```

```
Name:           manual-pvc
Namespace:      default
StorageClass:   manual
Status:         Bound
Volume:         manual-pv
Labels:          <none>
Annotations:    pv.kubernetes.io/bind-completed: yes
                  pv.kubernetes.io/bound-by-controller: yes
Finalizers:     [kubernetes.io/pvc-protection]
Capacity:       1Gi
Access Modes:   RWO
VolumeMode:     Filesystem
Used By:        <none>
Events:          <none>
```

Vervolgens gaan we deze PVC gebruiken in een pod. Hiervoor volstaat het om een pod manifest file aan te maken genaamd pod-with-pvc.yaml.

```
apiVersion: v1
kind: Pod
```

```

metadata:
  name: pvc-demo
spec:
  containers:
    - name: app
      image: busybox
      command: ['sh', '-c', 'echo "Persistent data" > /mydata/file.txt && sleep 3600']
  volumeMounts:
    - name: persistent-storage
      mountPath: /mydata
  volumes:
    - name: persistent-storage
      persistentVolumeClaim:
        claimName: manual-pvc

```

Dan rest je alleen nog de pod te apply'en en een file aan te maken.

```

student@virt-k8s-XX:~$ kubectl apply -f pod-with-pvc.yaml
pod/pvc-demo created

```

We schrijven de tekst "Important data" naar het bestand /mydata/important.txt binnenvin de Pod pvc-demo, zodat het wordt opgeslagen op de gekoppelde PersistentVolume.

```

student@virt-k8s-XX:~$ kubectl exec pvc-demo -- sh -c 'echo
"Important data" > /mydata/important.txt'

```

We tonen nu de inhoud van het bestand /mydata/important.txt binnenvin de container van de pod pvc-demo.

```

student@virt-k8s-XX:~$ kubectl exec pvc-demo -- cat
/mydata/important.txt
Important data

```

We tonen nu de inhoud van het bestand /mydata/file.txt binnenvin de Pod pvc-demo.

```

student@virt-k8s-XX:~$ kubectl exec pvc-demo -- cat
/mydata/file.txt

```

Persistent data

Verwijder nu de pod.

```

student@virt-k8s-XX:~$ kubectl delete pod pvc-demo
pod "pvc-demo" deleted

```

We maken nu terug eenzelfde pod aan.

```
student@virt-k8s-XX:~$ kubectl apply -f pod-with-pvc.yaml
pod/pvc-demo created
```

We checken nu of de data nog beschikbaar is.

```
student@virt-k8s-XX:~$ kubectl exec pvc-demo -- cat
/mydata/important.txt
```

Important data

```
student@virt-k8s-XX:~$ kubectl exec pvc-demo -- cat
/mydata/file.txt
```

Persistent data

Zoals we constateren is dit inderdaad het geval!

We verwijderen nu alle resources van deze demo.

```
student@virt-k8s-XX:~$ kubectl delete pod pvc-demo
student@virt-k8s-XX:~$ kubectl delete pvc manual-pvc
student@virt-k8s-XX:~$ kubectl delete pv manual-pv
```

11.10.7.3 Persistent storage – Dynamische storage

In k3d kunnen we ook dynamische opslagprovisioning gebruiken. Dit gebeurt via de ingebouwde default StorageClass, meestal local-path, omdat er geen cloudprovider zoals AWS of GCP beschikbaar is om opslag te beheren.

We maken hiervoor eerst een nieuwe PVC-manifest genaamd dynamic-pvc.yaml:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: dynamic-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
```

StorageClassName is optioneel; als je het niet opgeeft, wordt in K3d standaard de local-path StorageClass gebruikt.

We maken ook een Pod manifest aan dynamic-pod.yaml die de PVC gebruikt:

```

apiVersion: v1
kind: Pod
metadata:
  name: dynamic-demo
spec:
  containers:
  - name: app
    image: nginx
    volumeMounts:
    - name: web-content
      mountPath: /usr/share/nginx/html
  volumes:
  - name: web-content
    persistentVolumeClaim:
      claimName: dynamic-pvc

```

We hebben geen persistent volume aangemaakt. Zoals aangehaald zal de local-path StorageClass gebruikt worden.

```

student@virt-k8s-XX:~$ kubectl get pv
No resources found

```

We deployen nu de persistent volume claim.

```

student@virt-k8s-XX:~$ kubectl apply -f dynamic-pvc.yaml
persistentvolumeclaim/dynamic-pvc created

```

We vragen nu persistent volume claim op.

```

student@virt-k8s-XX:~$ kubectl get pvc
NAME        STATUS    VOLUME   CAPACITY  ACCESS MODES  STORAGECLASS ...
dynamic-pvc  Pending

```

Je ziet bij STATUS Pending staan omdat Kubernetes wacht met binden tot er een Pod is die gebruik maakt van de persistent volume claim.

Uiteraard zijn er nu nog geen persistent volumes.

```

student@virt-k8s-XX:~$ kubectl get pv
No resources found

```

Dit zal immers pas aangemaakt worden wanneer er een pod gebruikt maakt van het PVC dat op zijn beurt automatisch een PersistentVolume zal aanmaken.

```
student@virt-k8s-XX:~$ kubectl apply -f dynamic-pod.yaml
pod/dynamic-demo created
```

We vragen nu terug persistent volumes claims en persistant volume op.

```
student@virt-k8s-XX:~$ kubectl get pvc
```

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES	STORAGECLASS	...
dynamic-pvc	Bound	pvc- ...	2Gi	RWO	local-path	...

```
student@virt-k8s-XX:~$ kubectl get pv
```

NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS	...
pvc-...	2Gi	RWO	Delete	Bound	...

Doordat de pod gebruikt maakt van de PVC met de default storageclass, is er nu ook automatisch een Volume aangemaakt. Deze heeft een dynamische naam die aangemaakt is door het K8s zelf.

Nu kunnen we in de pod een bestand (webpagina) wegschrijven, dat door nginx zal gebruikt worden.

```
student@virt-k8s-XX:~$ kubectl exec -it dynamic-demo -- /bin/sh
# echo "<h1>Persistent Web Content</h1>" >
/usr/share/nginx/html/index.html
# exit
```

We starten nu port forward.

```
student@virt-k8s-XX:~$ kubectl port-forward pod/dynamic-demo
8080:80 &
```

...

We vragen nu webpagina op vanaf de host.

```
student@virt-k8s-XX:~$ curl localhost:8080
Handling connection for 8080
<h1>Persistent Web Content</h1>
```

Wanneer we nu de pod verwijderen zal de storage nog steeds blijven bestaan.

Verwijder nu de pod.

```
student@virt-k8s-XX:~$ kubectl delete pod dynamic-demo
```

```
pod "dynamic-demo" deleted
```

Het opvragen van de webpagina gaat nu uiteraard niet meer.

```
student@virt-k8s-XX:~$ curl localhost:8080
```

```
Handling connection for 8080
```

```
...
```

```
curl: (52) Empty reply from server
```

```
error: lost connection to pod
```

We maken nu terug een nieuwe pod aan metzelfde instellingen.

```
student@virt-k8s-XX:~$ kubectl apply -f dynamic-pod.yaml
```

```
pod/dynamic-demo created
```

We starten nu port forward opnieuw op.

```
student@virt-k8s-XX:~$ kubectl port-forward pod/dynamic-demo
```

```
8080:80 &
```

```
...
```

We vragen nu de webpagina opnieuw op.

```
student@virt-k8s-XX:~$ curl localhost:8080
```

```
Handling connection for 8080
```

```
<h1>Persistent Web Content</h1>
```

We verwijderen de nieuwe pod.

```
student@virt-k8s-XX:~$ kubectl delete pod dynamic-demo
```

```
pod "dynamic-demo" deleted
```

Wanneer we de Claim verwijderen zal het volume nu ook automatisch verwijderd worden. Dit komt omdat de reclaimPolicy op delete staat voor local-path. Hieronder zie je dat duidelijk staan.

```
student@virt-k8s-XX:~$ kubectl get storageclass
```

NAME	PROVISIONER	RECLAIMPOLICY
local-path (default)	rancher.io/local-path	Delete

We verwijderen persistent volume claim dynamic-pvc.

```
student@virt-k8s-XX:~$ kubectl delete pvc dynamic-pvc
persistentvolumeclaim "dynamic-pvc" deleted
```

We vragen nu de persistent volumes op.

```
student@virt-k8s-XX:~$ kubectl get pv
No resources found
```

Zoals verwacht is er geen persistent volume meer.

11.11 Disclaimer

Het hoofdstuk van K8s is gebaseerd op het werk van Tom Cool (lector bachelor) aangevuld met eigen inbreng (ook gebruik makend van het internet).

12 Van Podman naar Kubernetes

12.1 Inleiding

In dit hoofdstuk wordt stap voor stap uitgelegd hoe je een bestaande Podman-pod kunt omzetten naar een Kubernetes-manifest met behulp van podman generate kube.

We tonen niet alleen hoe je de YAML-bestanden kunt genereren, maar ook hoe je ze moet aanpassen zodat ze compatibel zijn met Kubernetes (met name k3d-clusters).

Daarnaast gaan we dieper in op veelvoorkomende problemen bij de omzetting en tonen we hoe je deze kunt oplossen.

We zullen in een later stadium ook gebruik maken van deployments.i.p.v. pods

We gebruiken als casestudy Wordpress.

12.2 Podman pods voor Wordpress

Eerst zullen we alle clusters van Kubernetes verwijderen die aanwezig zijn.

```
student@virt-k8s-XX:~$ kubectl config get-clusters  
student@virt-k8s-XX:~$ k3d cluster delete [naam cluster]
```

Voer het laatste commando uit totdat alle clusters verdwenen zijn.

We zullen nu eerst Wordpress installeren via een podman pod zoals we reeds gezien hebben in hoofdstuk 8. Voor meer info hierover verwiss ik dan ook graag door naar hoofdstuk 9.

```
student@virt-k8s-XX:~$ podman pod create --name wp-pod -p  
8080:80  
  
student@virt-k8s-XX:~$ podman run -d --restart=always --pod=wp-  
pod -e MYSQL_ROOT_PASSWORD="dbpass" -e MYSQL_DATABASE="wp" -  
e MYSQL_USER="wordpress" -e MYSQL_PASSWORD="wppass" --name=wp-db  
mariadb  
  
student@virt-k8s-XX:~$ 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  
  
student@virt-k8s-XX:~$ podman ps -a --pod  
  
...
```

```

student@serverXX:~$ podman ps -a --pod
CONTAINER ID IMAGE COMMAND CREATED STATUS
PORTS NAMES POD ID PODNAME
d85de60d425e quay.io/podman/hello:latest /usr/local/bin/po... 5 weeks ago Exited (0) 5 weeks ago
          zealous_austin
3d823a7bbfffb localhost/podman-pause:5.4.0-1757462400 0.0.0.0:8080->80/tcp d2402d6d61aa-infra d2402d6d61aa wp-pod
          mariadb
          0.0.0.0:8080->80/tcp, 3306/tcp wp-db d2402d6d61aa wp-pod
          apache2-foreground...
          0.0.0.0:8080->80/tcp wp-web d2402d6d61aa wp-pod
student@serverXX:~$
```

12.3 Genereren van een Kubernetes-manifest

We zetten nu het bestaande Podman-pod automatisch om naar een Kubernetes YAML-bestand.

```
student@virt-k8s-XX:~$ podman generate kube -s wp-pod > wordpress.yaml
```

We bekijken nu de inhoud van dit bestand.

```
student@virt-k8s-XX:~$ cat wordpress.yaml

# Save the output of this file and use kubectl create -f to import

# it into Kubernetes.

#
# Created with podman-5.4.0

apiVersion: v1

kind: Service

metadata:

  creationTimestamp: "2025-11-10T10:19:38Z"

  labels:

    app: wp-pod

    name: wp-pod

spec:

  ports:

    - name: "80"

      nodePort: 30389
```

```

    port: 80
    targetPort: 80
  selector:
    app: wp-pod
  type: NodePort
---
apiVersion: v1
kind: Pod
metadata:
  annotations:
    io.kubernetes.cri-o.SandboxID/wp-db:
3d823a7bbffbf0b6f2ca346327dc86c580045eaee1a459a03e2bc72be7547f
    io.kubernetes.cri-o.SandboxID/wp-web:
3d823a7bbffbf0b6f2ca346327dc86c580045eaee1a459a03e2bc72be7547f
  creationTimestamp: "2025-11-10T10:19:38Z"
  labels:
    app: wp-pod
  name: wp-pod
spec:
  containers:
    - args:
        - mariadb
      env:
        - name: MYSQL_ROOT_PASSWORD
          value: dbpass
        - name: MYSQL_USER
          value: wordpress

```

```

- name: MYSQL_DATABASE
  value: wp

- name: MYSQL_PASSWORD
  value: wppass

image: docker.io/library/mariadb:latest

name: wp-db

ports:
- containerPort: 80

volumeMounts:
- mountPath: /var/lib/mysql
  name:
13db45d1db6fb6a94cd40dffca9c7b4f4aaed0ebb91ad5d9a86316283e12e5
-pvc

- args:
- apache2-foreground

env:
- name: WORDPRESS_DB_USER
  value: wordpress

- name: WORDPRESS_DB_PASSWORD
  value: wppass

- name: WORDPRESS_DB_HOST
  value: 127.0.0.1

- name: WORDPRESS_DB_NAME
  value: wp

image: docker.io/library/wordpress:latest

name: wp-web

volumeMounts:

```

```

    - mountPath: /var/www/html

      name:
fe3e9358634f981ebcbb20597b3a11b0be33be227ed32671e8df2684e40d4788
-pvc

      volumes:

      - name:
13db45d1db6fb6a94cd40dffca9c7b4f4aaeded0ebb91ad5d9a86316283e12e5
-pvc

      persistentVolumeClaim:

        claimName:
13db45d1db6fb6a94cd40dffca9c7b4f4aaeded0ebb91ad5d9a86316283e12e5

        - name:
fe3e9358634f981ebcbb20597b3a11b0be33be227ed32671e8df2684e40d4788
-pvc

      persistentVolumeClaim:

        claimName:
fe3e9358634f981ebcbb20597b3a11b0be33be227ed32671e8df2684e40d4788

```

Er is output gegenereerd.

Als je de output analyseert zal je zien dat volgende is aangemaakt:

- Service met de naam wp-pod
- Pod met de naam wp-pod
 - o Container met de naam wp-db
 - o Container met de naam wp-web

12.4 Toepassen van manifestbestand

We verwijderen nu eerst de podman pod met de bijhorende containers.

```

student@virt-k8s-XX:~$ podman pod rm wp-pod -f
d2402d6d61aad6720c457526f05cbbcb8c29b8aad448c40d838b70fd673a148d

student@virt-k8s-XX:~$ podman ps --pod

```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	...
--------------	-------	---------	---------	--------	-----

Uiteraard is de pod nu verdwenen.

We maken nu een cluster aan met onderstaande instellingen.

Let op:

De poort 30689 in het Service-manifest moet overeenkomen met de poort die je in het k3d-cluster hebt doorgestuurd met de -p optie (8080:30689@server:0).

Dat betekent dat verkeer van poort 8080 op je lokale machine wordt doorgestuurd naar poort 30677 in de Kubernetes-service. Bij het genereren van het manifestbestand is een willekeurige nodePort gekozen!

```
student@virt-k8s-XX:~$ cat wordpress.yaml
```

...

```
ports:  
- name: "80"  
  nodePort: 30389  
  port: 80  
  targetPort: 80
```

...

```
student@virt-k8s-XX:~$ k3d cluster create mijncluster --servers  
1 --agents 1 -p "8080:30389@server:0"
```

...

We trachten nu het Kubernetes-manifest toe te passen.

```
student@virt-k8s-XX:~$ kubectl apply -f wordpress.yaml  
service/wp-pod created
```

The Pod "wp-pod" is invalid:

```
* spec.volumes[0].name: Invalid value:  
"13db45d1db6fb6a94cd40dffca9c7b4f4aaeded0ebb91ad5d9a86316283e12e  
5-pvc": must be no more than 63 characters  
  
* spec.volumes[1].name: Invalid value:  
"fe3e9358634f981ebcbb20597b3a11b0be33be227ed32671e8df2684e40d478  
8-pvc": must be no more than 63 characters  
  
* spec.containers[0].volumeMounts[0].name: Not found:  
"13db45d1db6fb6a94cd40dffca9c7b4f4aaeded0ebb91ad5d9a86316283e12e  
5-pvc"
```

```
* spec.containers[1].volumeMounts[0].name: Not found:  
"fe3e9358634f981ebcbb20597b3a11b0be33be227ed32671e8df2684e40d478  
8-pvc"
```

Jammer genoeg werkt het dus niet 'out of the box'.

12.5 Back-up maken en manifestbestand aanpassen

Ik raad nu aan om eerst een kopie te maken van het originele manifestbestand aangezien we wijzigingen zullen moeten aanbrengen om het werkend te krijgen.

```
student@virt-k8s-XX:~$ cp wordpress.yaml wordpress.yaml~
```

Een ~ na een bestandsnaam in Linux betekent meestal dat het gaat om een back-upbestand of tijdelijke kopie van een ander bestand.

We zullen nu het manifestbestand aanpassen zodat het wel werkt.

Allereerst: er zijn een aantal regels die je zonder problemen mag verwijderen uit dit bestand. Ik heb het dan over regels met creationTimestamp en annotations met io.kubernetes.cri-o.SandboxID/wp-db. Ook de regels die commentaar zijn (beginnend met #) mag je verwijderen. Dit is uiteraard nog geen oplossing voor het probleem maar zo maak je het bestand in ieder geval al beter leesbaar.

Verwijder daarom volgende regels:

```
# Save the output of this file and use kubectl create -f to  
import  
# it into Kubernetes.  
  
#  
  
# Created with podman 5.4.0  
  
creationTimestamp: "2025-11-10T10:19:38Z"  
  
annotations:  
  
io.kubernetes.cri-o.SandboxID/wp-db:  
3d823a7bbffbf0b6f2ea346327de86e580045eaee1a459a03e2bc72be7547  
f  
  
io.kubernetes.cri-o.SandboxID/wp-web:  
3d823a7bbffbf0b6f2ea346327de86e580045eaee1a459a03e2bc72be7547  
f  
  
creationTimestamp: "2025-11-10T10:19:38Z"
```

Lange namen mogen niet gebruikt worden in een manifestbestand. Kubernetes verwacht namen van maximaal 63 tekens. We lossen dit op.

```
volumeMounts:
- mountPath: /var/lib/mysql
  name: 13db45d1db6fb6a94cd40dffca9c7b4f4aaeeded0ebb91ad5d9a86316283e12e5-pvc
- args:
- apache2-foreground
env:
- name: WORDPRESS_DB_USER
  value: wordpress
- name: WORDPRESS_DB_PASSWORD
  value: wppass
- name: WORDPRESS_DB_HOST
  value: 127.0.0.1
- name: WORDPRESS_DB_NAME
  value: wp
image: docker.io/library/wordpress:latest
name: wp-web
volumeMounts:
- mountPath: /var/www/html
  name: fe3e9358634f981ebcbb20597b3a11b0be33be227ed32671e8df2684e40d4788-pvc
volumes:
- name: 13db45d1db6fb6a94cd40dffca9c7b4f4aaeeded0ebb91ad5d9a86316283e12e5-pvc
  persistentVolumeClaim:
    claimName: 13db45d1db6fb6a94cd40dffca9c7b4f4aaeeded0ebb91ad5d9a86316283e12e5
- name: fe3e9358634f981ebcbb20597b3a11b0be33be227ed32671e8df2684e40d4788-pvc
  persistentVolumeClaim:
    claimName: fe3e9358634f981ebcbb20597b3a11b0be33be227ed32671e8df2684e40d4788
-----, -----, -----
  name: fe3e9358634f981ebcbb20597b3a11b0be33be227ed32671e8df2684e40d4788-pvc
volumes:
- name: 13db45d1db6fb6a94cd40dffca9c7b4f4aaeeded0ebb91ad5d9a86316283e12e5-pvc
  persistentVolumeClaim:
    claimName: 13db45d1db6fb6a94cd40dffca9c7b4f4aaeeded0ebb91ad5d9a86316283e12e5
- name: fe3e9358634f981ebcbb20597b3a11b0be33be227ed32671e8df2684e40d4788-pvc
  persistentVolumeClaim:
    claimName: fe3e9358634f981ebcbb20597b3a11b0be33be227ed32671e8df2684e40d4788
```

Ik vervang in mijn manifestbestand

13db45d1db6fb6a94cd40dffca9c7b4f4aaeeded0ebb91ad5d9a86316283e12e5 door wp-db-mount en
fe3e9358634f981ebcbb20597b3a11b0be33be227ed32671e8df2684e40d4788 door wp-web-mount.

```

! wordpress.yaml
!
1  apiVersion: v1
2  kind: Service
3  metadata:
4    labels:
5      app: wp-pod
6      name: wp-pod
7  spec:
8    ports:
9      - name: "80"
10     nodePort: 30389
11     port: 80
12     targetPort: 80
13   selector:
14     app: wp-pod
15     type: NodePort
16 ---
17  apiVersion: v1
18  kind: Pod
19  metadata:
20    labels:
21      app: wp-pod
22      name: wp-pod
23  spec:
24    containers:
25      - args:
26        - mariadb
27        env:
28          - name: MYSQL_ROOT_PASSWORD
29            value: dbpass
30          - name: MYSQL_USER
31            value: wordpress
32          - name: MYSQL_DATABASE
33            value: wp
34          - name: MYSQL_PASSWORD
35            value: wppass
36        image: docker.io/library/mariadb:latest
37        name: wp-db
38        ports:
39          - containerPort: 80
40        volumeMounts:
41          - mountPath: /var/lib/mysql
42            name: 13db45d1db6fb6a94cd40dffca9c7b4f4aaed0ebb91ad5d9a86316283e12e5-pvc
43        args:
44          - apache2-foreground
45        env:
46          - name: WORDPRESS_DB_USER
47            value: wordpress
48          - name: WORDPRESS_DB_PASSWORD
49            value: wppass
50          - name: WORDPRESS_DB_HOST
51            value: 127.0.0.1
52          - name: WORDPRESS_DB_NAME
53            value: wp
54        image: docker.io/library/wordpress:latest
55        name: wp-web

```

Uiteindelijk krijg je onderstaand yaml-bestand.

```
student@virt-k8s-XX:~$ cat wordpress.yaml
```

```
apiVersion: v1
```

```
kind: Service
```

```
metadata:
```

```
  labels:
```

```
    app: wp-pod
```

```
    name: wp-pod
```

```
spec:
```

```
  ports:
```

```
    - name: "80"
```

```
    nodePort: 30389
```

```

    port: 80
    targetPort: 80
  selector:
    app: wp-pod
  type: NodePort
---
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: wp-pod
  name: wp-pod
spec:
  containers:
    - args:
        - mariadb
      env:
        - name: MYSQL_ROOT_PASSWORD
          value: dbpass
        - name: MYSQL_USER
          value: wordpress
        - name: MYSQL_DATABASE
          value: wp
        - name: MYSQL_PASSWORD
          value: wppass
      image: docker.io/library/mariadb:latest

```

```

name: wp-db

ports:
  - containerPort: 80

volumeMounts:
  - mountPath: /var/lib/mysql
    name: wp-db-mount-pvc

args:
  - apache2-foreground

env:
  - name: WORDPRESS_DB_USER
    value: wordpress
  - name: WORDPRESS_DB_PASSWORD
    value: wppass
  - name: WORDPRESS_DB_HOST
    value: 127.0.0.1
  - name: WORDPRESS_DB_NAME
    value: wp

image: docker.io/library/wordpress:latest

name: wp-web

volumeMounts:
  - mountPath: /var/www/html
    name: wp-web-mount-pvc

volumes:
  - name: wp-db-mount-pvc

persistentVolumeClaim:
  claimName: wp-db-mount

```

```
- name: wp-web-mount-pvc
  persistentVolumeClaim:
    claimName: wp-web-mount
```

Als je het grondig analyseert zal je constateren dat hierin een duidelijke fout staat...

```
...
name: wp-db
ports:
- containerPort: 80
...
```

De database-server luistert op poort 3306 zoals we reeds vaak hebben besproken.

We passen het manifestbestand dan ook als volgt aan.

```
...
name: wp-db
ports:
- containerPort: 3306
...
```

Bij de container wp-web zie dan weer geen poort. Voeg dat ook volgende regens toe onder name: wp-web.

```
...
ports:
- containerPort: 80
...
```

Nu heeft wordpress.yaml onderstaande inhoud.

```
apiVersion: v1
kind: Service
metadata:
  labels:
```

```
    app: wp-pod

    name: wp-pod

spec:

  ports:

    - name: "80"

      nodePort: 30389

      port: 80

      targetPort: 80

  selector:

    app: wp-pod

    type: NodePort

---

apiVersion: v1

kind: Pod

metadata:

  labels:

    app: wp-pod

    name: wp-pod

spec:

  containers:

    - args:

        - mariadb

      env:

        - name: MYSQL_PASSWORD

          value: wppass

        - name: MYSQL_DATABASE
```

```

    value: wp

  - name: MYSQL_USER
    value: wordpress

  - name: MYSQL_ROOT_PASSWORD
    value: dbpass

image: docker.io/library/mariadb:latest

name: wp-db

ports:
  - containerPort: 3306

volumeMounts:
  - mountPath: /var/lib/mysql
    name: wp-db-mount-pvc

  - args:
      - apache2-foreground

env:
  - name: WORDPRESS_DB_PASSWORD
    value: wppass

  - name: WORDPRESS_DB_HOST
    value: 127.0.0.1

  - name: WORDPRESS_DB_NAME
    value: wp

  - name: WORDPRESS_DB_USER
    value: wordpress

image: docker.io/library/wordpress:latest

name: wp-web

ports:

```

```

      - containerPort: 80

    volumeMounts:

      - mountPath: /var/www/html
        name: wp-web-mount-pvc

  volumes:

    - name: wp-web-mount-pvc

    persistentVolumeClaim:

      claimName: wp-web-mount

    - name: wp-db-mount-pvc

    persistentVolumeClaim:

      claimName: wp-db-mount

```

12.6 PVC-manifest aanmaken

Zoals je reeds geleerd hebt moet je ook een PVC-manifest aanmaken. Dit moeten we dus nog zelf aanmaken.

```

student@virt-k8s-XX:~$ cat opslag.yaml

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

  name: wp-db-mount

spec:

  accessModes:
    - ReadWriteOnce

  resources:

    requests:
      storage: 1Gi

```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: wp-web-mount
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

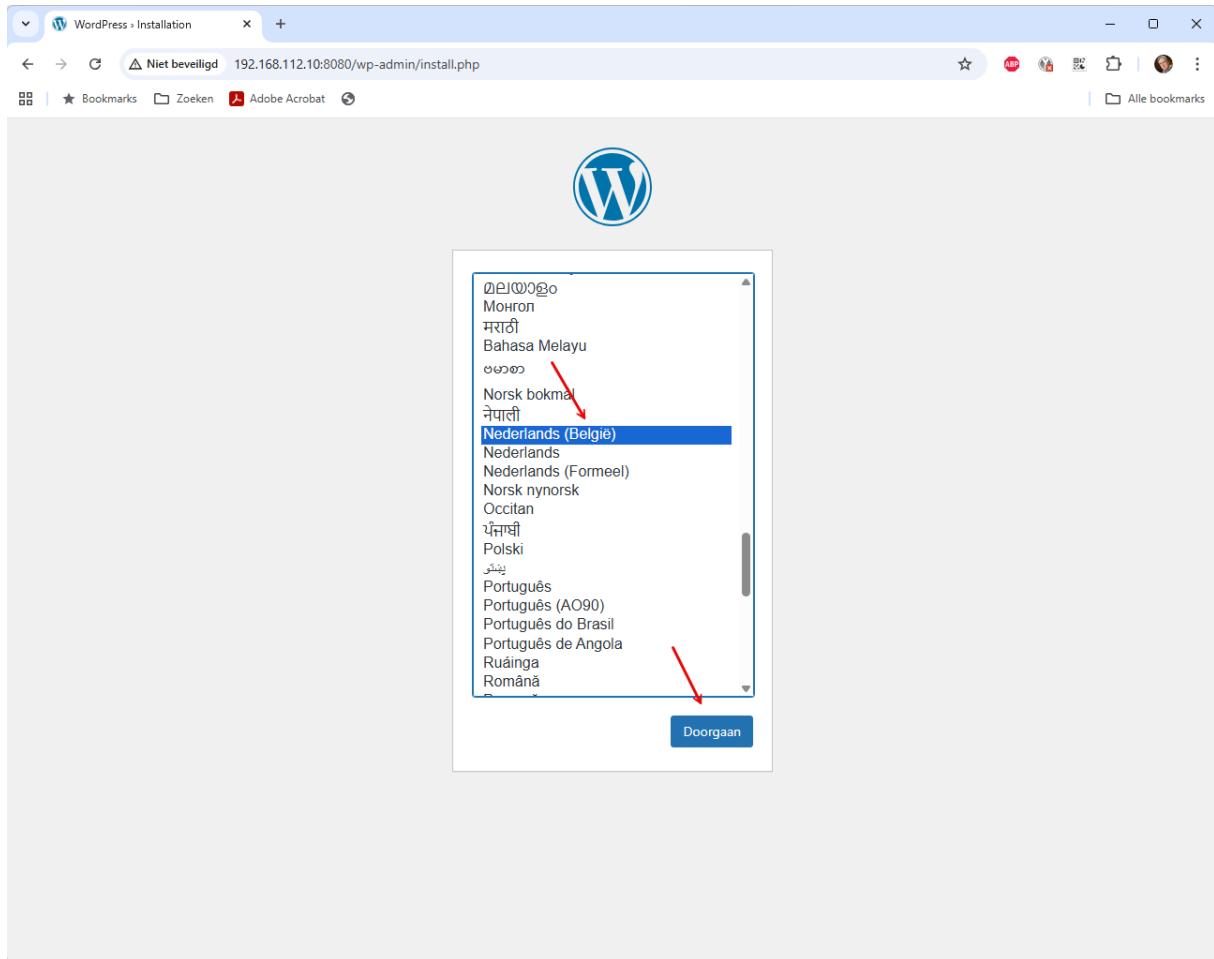
12.7 Toepassen van manifestbestanden

Voer nu onderstaande uit.

```
student@virt-k8s-XX:~$ kubectl apply -f opslag.yaml
persistentvolumeclaim/wp-db-mount created
persistentvolumeclaim/wp-web-mount created
student@virt-k8s-XX:~$ kubectl apply -f wordpress.yaml
service/wp-pod created
pod/wp-pod created
```

12.8 Instellen Wordpress

Je kan nu op een andere computer op het 192.168.112.0-netwerk onderstaande uitvoeren.



Welkom

Welkom bij het bekende vijf minuten installatieproces van WordPress! Vul gewoon de informatie hieronder in en je bent klaar om het meeste krachtige en uitbreidbare publicatieplatform van de wereld te gebruiken.

Benodigde informatie

De volgende informatie invoeren. Maak je geen zorgen, deze instellingen kunnen steeds worden gewijzigd.

Website titel	<input type="text" value="testwebsite"/>
Gebruikersnaam	<input type="text" value="student"/>
Wachtwoord	<input type="password" value="123"/> Verbergen <small>Erg zwak</small>
Bevestig wachtwoord	<input checked="" type="checkbox"/> Bevestig het gebruik van een zwak wachtwoord
Je e-mailadres	<input type="text" value="stijn.jacobs@pxl.be"/> <small>Controleer zorgvuldig of je het e-mailadres goed hebt ingevuld voordat je verder gaat.</small>
Zoekmachine zichtbaarheid	<input checked="" type="checkbox"/> Blokkeer zoekmachines deze website te indexeren <small>Het is aan de zoekmachines of ze gehoor geven aan dit verzoek.</small>
WordPress installeren	

WordPress > installatie

Niet beveiligd 192.168.112.10:8080/wp-admin/install.php?step=2

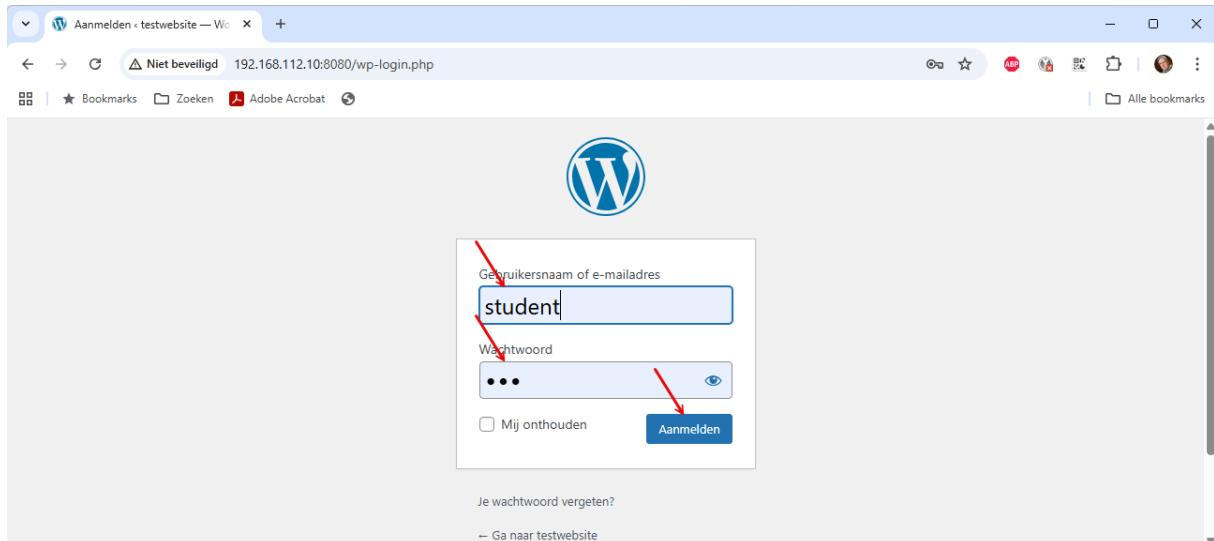
Gelukt!

WordPress is geïnstalleerd. Bedankt, en veel plezier!

Gebruikersnaam student

Wachtwoord Je gekozen wachtwoord.

[Aanmelden](#)



A screenshot of the WordPress dashboard. The URL in the address bar is 192.168.112.10:8080/wp-admin/. The top navigation bar includes links for 'testwebsite', 'Dashboard', 'Home', 'Updates', 'Berichten', 'Media', 'Pagina's', 'Reacties', 'Weergave', 'Plugins', 'Gebruikers', 'Tools', 'Instellingen', and 'Menu invouwen'. A red arrow points to the 'Dashboard' link. The main content area features a large 'Welkom bij WordPress!' heading and a link to 'Leer meer over de 6.8.3 versie.'. Below this are three promotional cards: one about writing rich content with blocks and patterns, another about customizing the theme, and a third about changing the website's look and feel with styles. Each card has a small icon and a 'Negeren' (Delete) button.



12.9 Check persistant storage in cluster

We vragen de service, pods en pvc op.

```
student@virt-k8s-XX:~$ kubectl get service
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
AGE				
kubernetes	ClusterIP	10.43.0.1	<none>	443/TCP 26m
wp-pod	NodePort	10.43.53.246	<none>	80:30389/TCP 25m

PS De service kubernetes wordt automatisch door de cluster zelf aangemaakt. Het is de interne service die verwijst naar de API-server van je Kubernetes-cluster.

```
student@virt-k8s-XX:~$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
wp-pod	2/2	Running	0	19m

Je ziet 2/2 omdat wp-pod 2 containers bevat.

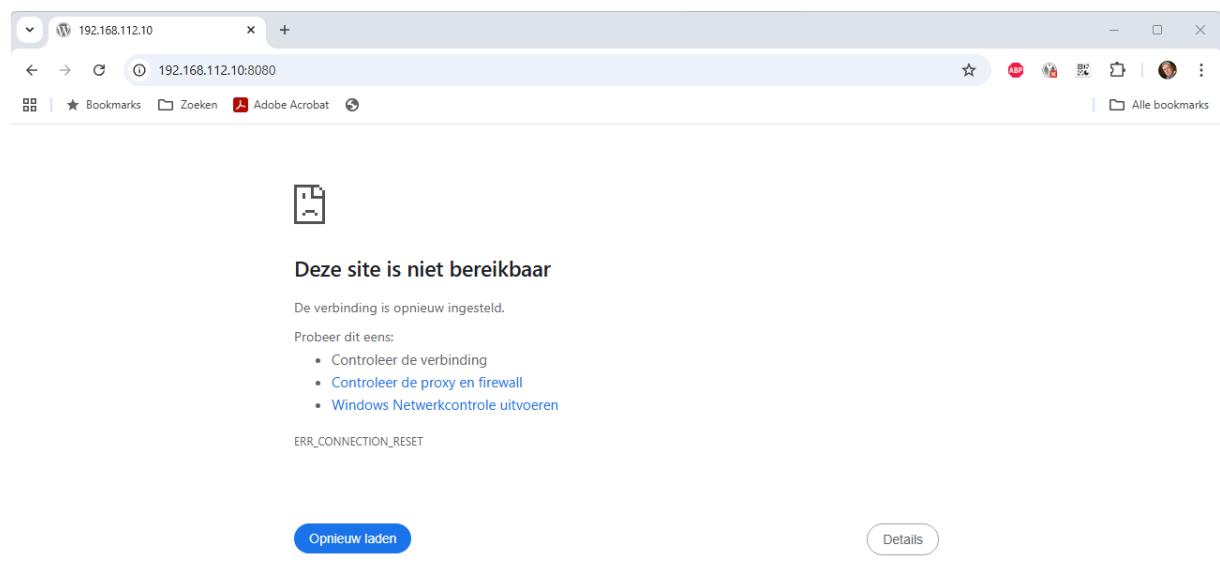
```
student@virt-k8s-XX:~$ kubectl get pvc
```

NAME	STATUS	VOLUME	...
wp-db-mount	Bound	pvc-aa4a6e08-0aa5-40ca-9123-ffd3fe5aa8f0	...
wp-web-mount	Bound	pvc-bb7b9a94-72bb-4ee9-b40a-740681eeb613	...

Om te checken of de gegevens in de cluster zijn opgeslagen verwijderen we de pods en de service. We verwijderen hiervoor de service en de pod.

```
student@virt-k8s-XX:~$ kubectl delete service wp-pod
service "wp-pod" deleted
student@virt-k8s-XX:~$ kubectl delete pod wp-pod
pod "wp-pod" deleted
```

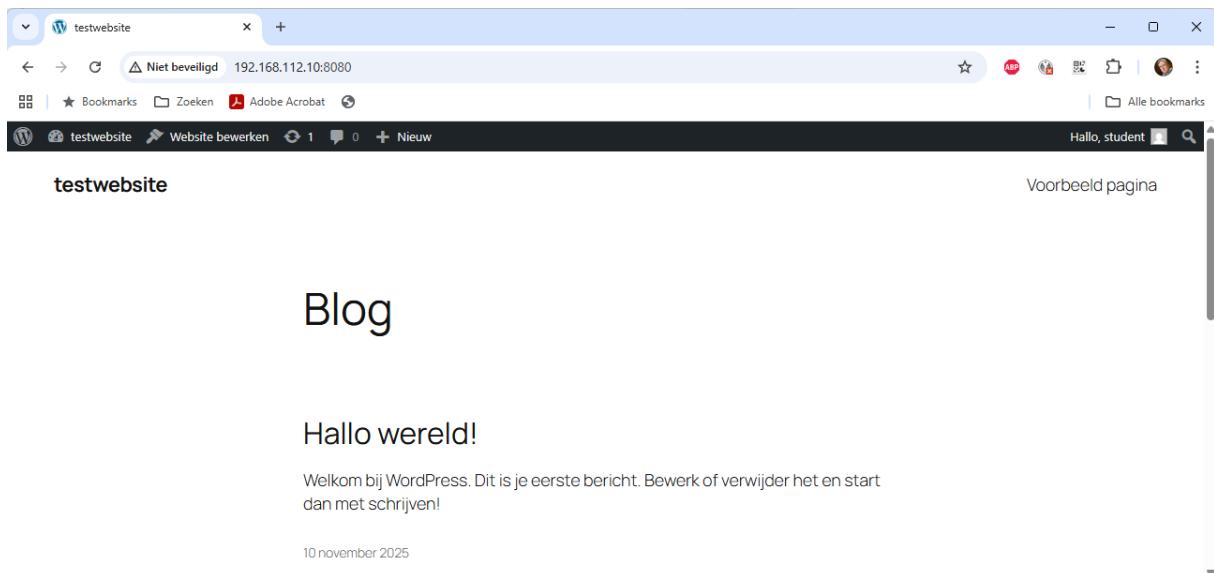
Uiteraard is nu de website niet meer beschikbaar.



We maken de service en de pod nu terug aan aan de hand van het manifestbestand.

```
student@virt-k8s-XX:~$ kubectl apply -f wordpress.yaml
service/wp-pod created
pod/wp-pod created
```

Als je nu de webpagina terug vraagt zal je zien dat deze terug beschikbaar is.



12.10 Meerdere manifestbestanden

Grottere YAML-bestanden, zoals het bestand dat we net gebruikten (*wordpress.yaml*), worden al snel onoverzichtelijk. Voor schaalbaarheid is het bovendien beter om verschillende YAML-bestanden te gebruiken — bijvoorbeeld één per service of component.

We maken eerst een map aan en kopiëren de inhoud van *wordpress.yaml* ernaartoe.

```
student@virt-k8s-XX:~$ mkdir wordpress
student@virt-k8s-XX:~$ cp wordpress.yaml wordpress
student@virt-k8s-XX:~$ cd wordpress
```

We doen dit als volgt. Alles boven --- hoort bij service. Verwijder dus alles onder --- uit *servicewp.yaml* die je hieronder aanmaakt.

```
student@virt-k8s-XX:~/wordpress$ cp wordpress.yaml servicewp.yaml
student@virt-k8s-XX:~/wordpress$ nano servicewp.yaml
apiVersion: v1
kind: Service
metadata:
  labels:
```

```

    app: wp-pod

    name: wp-pod

spec:

  ports:

    - name: "80"

      nodePort: 30389

      port: 80

      targetPort: 80

  selector:

    app: wp-pod

  type: NodePort

```

Alles onder --- hoort bij pod. Verwijder in podwp.yaml die je hieronder aanmaakt aldus alles boven ---.

```

student@virt-k8s-XX:~/wordpress$ cp wordpress.yaml podwp.yaml

student@virt-k8s-XX:~/wordpress$ nano podwp.yaml

student@virt-k8s-XX:~/wordpress$ cat podwp.yaml

apiVersion: v1

kind: Pod

metadata:

  labels:

    app: wp-pod

  name: wp-pod

spec:

  containers:

    - args:

        - mariadb

      env:

```

```

- name: MYSQL_PASSWORD
  value: wppass

- name: MYSQL_DATABASE
  value: wp

- name: MYSQL_USER
  value: wordpress

- name: MYSQL_ROOT_PASSWORD
  value: dbpass

image: docker.io/library/mariadb:latest

name: wp-db

ports:
- containerPort: 3306

volumeMounts:
- mountPath: /var/lib/mysql
  name: wp-db-mount-pvc

args:
- apache2-foreground

env:
- name: WORDPRESS_DB_PASSWORD
  value: wppass

- name: WORDPRESS_DB_HOST
  value: 127.0.0.1

- name: WORDPRESS_DB_NAME
  value: wp

- name: WORDPRESS_DB_USER
  value: wordpress

```

```

image: docker.io/library/wordpress:latest

name: wp-web

ports:
- containerPort: 80

volumeMounts:
- mountPath: /var/www/html
  name: wp-web-mount-pvc

volumes:
- name: wp-web-mount-pvc

persistentVolumeClaim:
  claimName: wp-web-mount

- name: wp-db-mount-pvc

persistentVolumeClaim:
  claimName: wp-db-mount

```

We kunnen nu uiteraard wordpress.yaml verwijderen aangezien we die niet meer nodig hebben.

```
student@virt-k8s-XX:~/wordpress$ rm wordpress.yaml
```

Het is een goed idee om te checken of nu met deze 2 maniefestbestand hetzelfde resultaat komt.

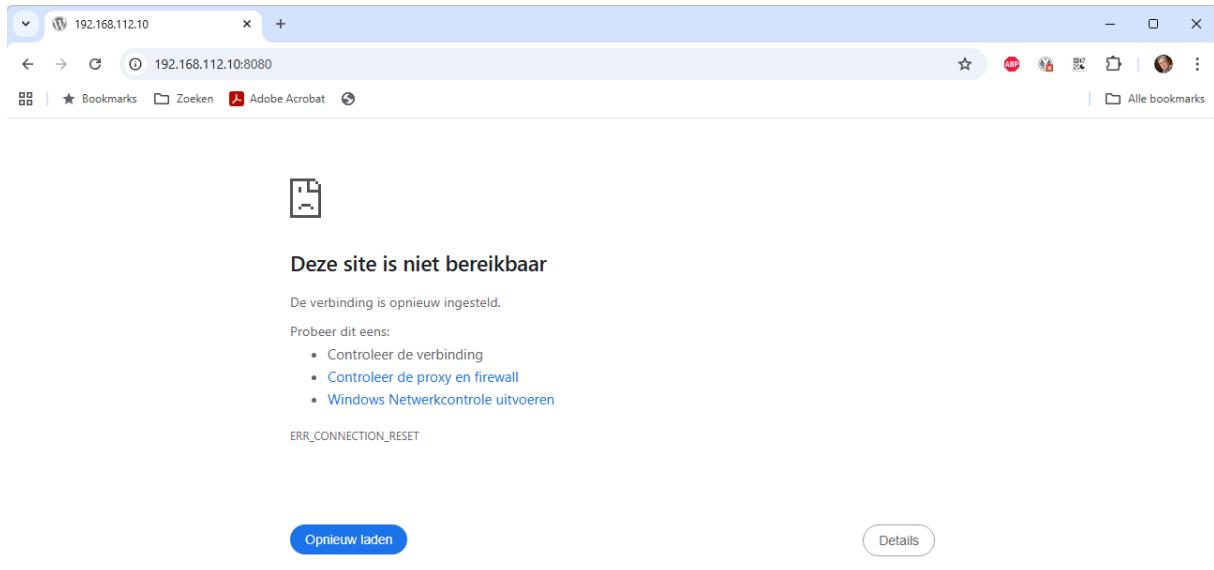
```
student@virt-k8s-XX:~/wordpress$ kubectl delete pod wp-pod
```

```
pod "wp-pod" deleted
```

```
student@virt-k8s-XX:~/wordpress$ kubectl delete service wp-pod
```

```
service "wp-pod" deleted
```

Uiteraard is de website nu niet meer beschikbaar.



We proberen nu de website op te zetten aan de hand van de 2 juist aangemaakte manifestbestanden.

```
student@virt-k8s-XX:~/wordpress$ kubectl apply -f servicewp.yaml
```

```
service/wp-pod created
```

```
student@virt-k8s-XX:~/wordpress$ kubectl apply -f podwp.yaml
```

```
pod/wp-pod created
```

We zullen constateren dat de website nu beschikbaar is.



We kunnen podwp.yaml beter ook nog in 2 splitsen door de database (MariaDB) en WordPress webserver elk in hun eigen Pod te zetten. Eén service per pod wordt om volgende redenen aanbevolen:

- Losse componenten (DB ↔ web) kun je onafhankelijk herstarten of schalen.
- Kleiner manifest per functie = beter beheer.
- Kubernetes best practice: één hoofdcontainer per Pod (niet twee containers samenzetten).

```
student@virt-k8s-XX:~/wordpress$ cp podwp.yaml podwordpress.yaml
```

```
student@virt-k8s-XX:~/wordpress$ cp podwp.yaml poddatabase.yaml
```

Pas poddatabase.yaml nu aan zodat het moet onderstaande overeenkomt.

```
student@virt-k8s-XX:~/wordpress$ nano poddatabase.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: database-pod
  name: database-pod
spec:
  containers:
    - args:
        - mariadb
      env:
        - name: MYSQL_PASSWORD
          value: wppass
        - name: MYSQL_DATABASE
          value: wp
        - name: MYSQL_USER
          value: wordpress
        - name: MYSQL_ROOT_PASSWORD
```

```

    value: dbpass

  image: docker.io/library/mariadb:latest

  name: wp-db

  ports:
    - containerPort: 3306

  volumeMounts:
    - mountPath: /var/lib/mysql
      name: wp-db-mount-pvc

  volumes:
    - name: wp-db-mount-pvc

  persistentVolumeClaim:
    claimName: wp-db-mount

```

Pas podwordpress.yaml nu aan zodat het moet onderstaande overeenkomt.

```

student@virt-k8s-XX:~/wordpress$ nano podwordpress.yaml

apiVersion: v1
kind: Pod
metadata:
  labels:
    app: wordpress-pod
  name: wordpress-pod
spec:
  containers:
    - args:
        - apache2-foreground
      env:
        - name: WORDPRESS_DB_PASSWORD

```

```

    value: wppass

  - name: WORDPRESS_DB_HOST
    value: database-service

  - name: WORDPRESS_DB_NAME
    value: wp

  - name: WORDPRESS_DB_USER
    value: wordpress

  image: docker.io/library/wordpress:latest

  name: wp-web

  ports:
    - containerPort: 80

  volumeMounts:
    - mountPath: /var/www/html
      name: wp-web-mount-pvc

  volumes:
    - name: wp-web-mount-pvc

  persistentVolumeClaim:
    claimName: wp-web-mount

```

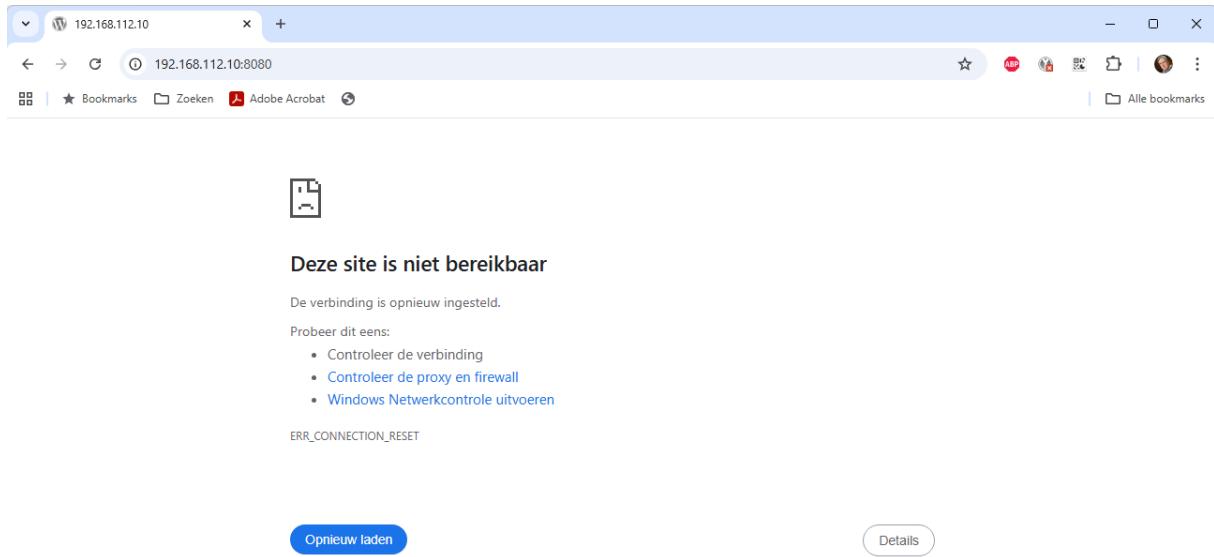
We verwijderen nu de pod met de 2 containers, genaamd wp-pod.

```

student@virt-k8s-XX:~/wordpress$ kubectl delete pod wp-pod
pod "wp-pod" deleted

```

Uiteraard is de website nu niet meer beschikbaar.



De service draait nog. We checken dit.

```
student@virt-k8s-XX:~/wordpress$ kubectl get service wp-pod
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	...
wp-pod	NodePort	10.43.109.19	<none>	...

We starten nu de 2 pods met telkens 1 container.

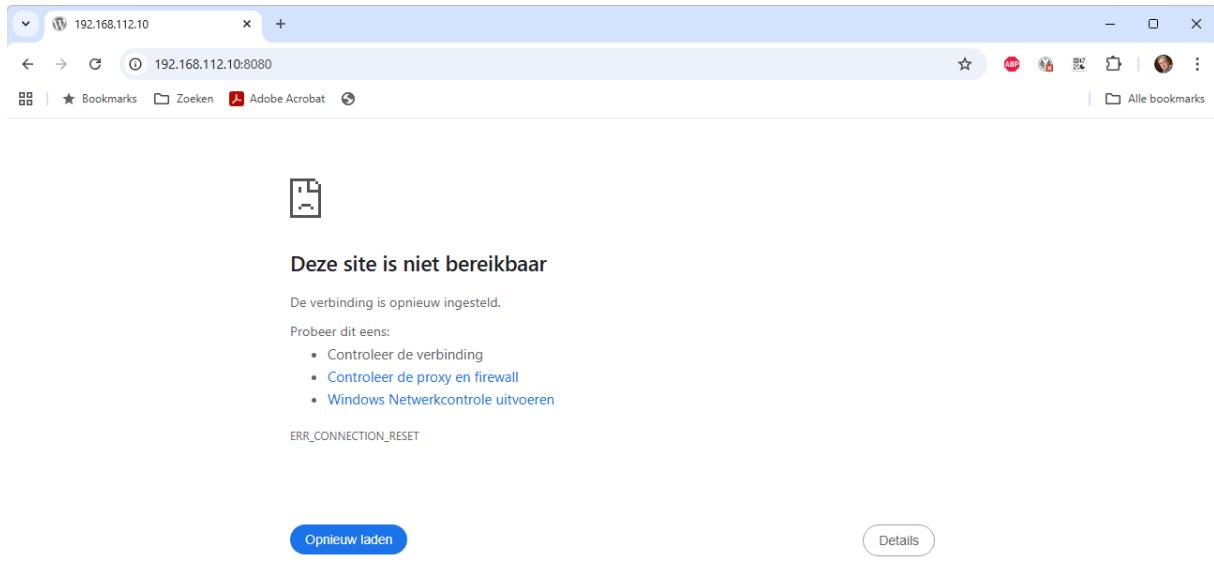
```
student@virt-k8s-XX:~/wordpress$ kubectl apply -f  
poddatabase.yaml
```

pod/database-pod created

```
student@virt-k8s-XX:~/wordpress$ kubectl apply -f  
podwordpress.yaml
```

pod/wordpress-pod created

Als je nu de website opvraagt krijg je onderstaande fout...



Er kan geen verbinding tot stand gebracht worden omdat servicewp.yaml wijst naar wp-pod en niet naar wordpress-pod.

```
student@virt-k8s-XX:~/wordpress$ cp servicewp.yaml
servicewordpress.yaml
```

Zorg dat servicewordpress.yaml onderstaande inhoud krijgt.

```
student@virt-k8s-XX:~/wordpress$ nano servicewordpress.yaml

apiVersion: v1

kind: Service

metadata:

  labels:

    app: wp

    name: wordpress-service

spec:

  ports:

  - name: "80"

    nodePort: 30389

    port: 80
```

```
    targetPort: 80

    selector:
        app: wordpress-pod

    type: NodePort
```

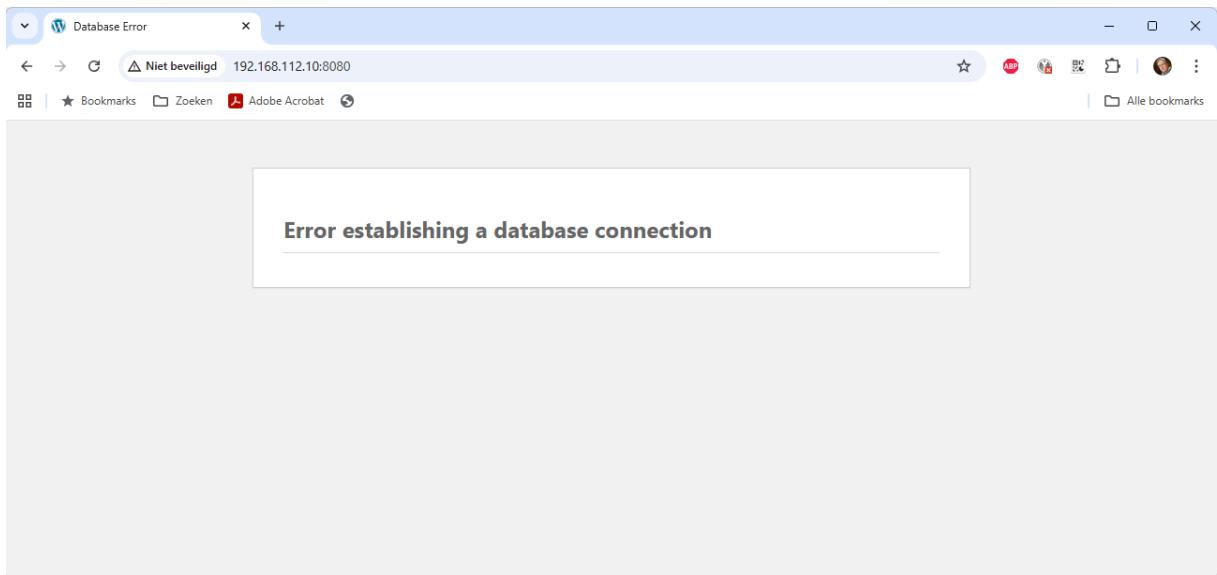
Stop de service wp-pod en pas de nieuwe service toe aan de hand van het manifestbestand.

```
student@virt-k8s-XX:~/wordpress$ kubectl delete service wp-pod
service "wp-pod" deleted

student@virt-k8s-XX:~/wordpress$ kubectl apply -f
servicewordpress.yaml

service/wordpress-service created
```

Als we nu naar de website gaan krijg je onderstaande foutmelding.



Dit komt omdat pods enkel met elkaar kunnen praten via IP-adressen, maar die zijn veranderlijk. Dit hebben we reeds besproken in vorig hoofdstuk.

We moeten dus een service bijmaken zodat de database-service bereikbaar wordt vanuit Wordpress-pod.

```
student@virt-k8s-XX:~/wordpress$ nano servicedatabase.yaml

apiVersion: v1
kind: Service
```

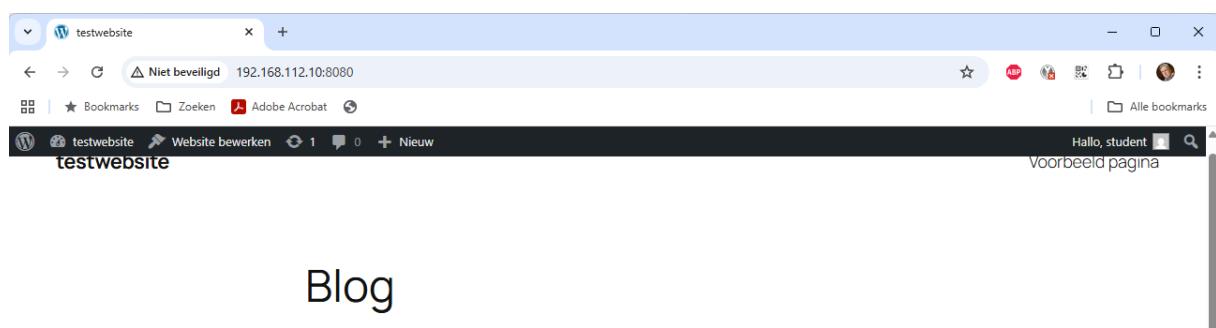
```
metadata:  
  labels:  
    app: wp  
    name: database-service  
  
spec:  
  selector:  
    app: database-pod  
  
  ports:  
    - port: 3306  
      targetPort: 3306
```

"TargetPort: 3306" is optioneel. Dit is de poort in de Pod waar verkeer naartoe wordt gestuurd. Als je targetPort weglaat, gebruikt Kubernetes automatisch dezelfde waarde als 'port'

We starten nu de service aan de hand van het manifestbestand.

```
student@virt-k8s-XX:~/wordpress$ kubectl apply -f  
servicedatabase.yaml  
  
service/database-service created
```

De website is nu opnieuw beschikbaar!



Blog

Hallo wereld!

Welkom bij WordPress. Dit is je eerste bericht. Bewerk of verwijder het en start dan met schrijven!

10 november 2025

12.11 Deployments

In plaats van pods kun je beter Deployments gebruiken.

Deployments zorgen ervoor dat pods automatisch herstarten als ze crashen, dat ze eenvoudig geschaald kunnen worden en dat updates gecontroleerd uitgerold kunnen worden. Voor onze WordPress-opstelling betekent dit dat zowel de MariaDB-pod als de WordPress-webserver in een Deployment geplaatst worden.

Het gebruik van deployments is trouwens best-practice...

Aan de hand van de pods maken we volgende deployments aan.

```
student@virt-k8s-XX:~/wordpress$ nano deployment-database.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: database-deployment
  labels:
    app: database-pod
spec:
  replicas: 1
  selector:
    matchLabels:
      app: database-pod
  template:
    metadata:
      labels:
        app: database-pod
    spec:
      containers:
        - name: wp-db
```

```

image: docker.io/library/mariadb:latest

args:
  - mariadb

env:
  - name: MYSQL_PASSWORD
    value: wppass
  - name: MYSQL_DATABASE
    value: wp
  - name: MYSQL_USER
    value: wordpress
  - name: MYSQL_ROOT_PASSWORD
    value: dbpass

ports:
  - containerPort: 3306

volumeMounts:
  - mountPath: /var/lib/mysql
    name: wp-db-mount-pvc

volumes:
  - name: wp-db-mount-pvc

persistentVolumeClaim:
  claimName: wp-db-mount

```

```

student@virt-k8s-XX:~/wordpress$ nano deployment-wordpress.yaml

apiVersion: apps/v1
kind: Deployment
metadata:

```

```
name: wordpress-deployment

labels:
  app: wp-pod

spec:
  replicas: 1
  selector:
    matchLabels:
      app: wordpress-pod
  template:
    metadata:
      labels:
        app: wordpress-pod
    spec:
      containers:
        - name: wp-web
          image: docker.io/library/wordpress:latest
          args:
            - apache2-foreground
      env:
        - name: WORDPRESS_DB_PASSWORD
          value: wppass
        - name: WORDPRESS_DB_HOST
          value: database-service
        - name: WORDPRESS_DB_NAME
          value: wp
        - name: WORDPRESS_DB_USER
```

```

    value: wordpress

  ports:
    - containerPort: 80

  volumeMounts:
    - mountPath: /var/www/html
      name: wp-web-mount-pvc

  volumes:
    - name: wp-web-mount-pvc

  persistentVolumeClaim:
    claimName: wp-web-mount

```

We verwijderen de pods en starten de deployments.

```

student@virt-k8s-XX:~/wordpress$ kubectl delete pod database-pod
pod "database-pod" deleted

student@virt-k8s-XX:~/wordpress$ kubectl delete pod wordpress-pod
pod "wordpress-pod" deleted

student@virt-k8s-XX:~/wordpress$ kubectl apply -f deployment-
database.yaml
deployment.apps/database-deployment created

student@virt-k8s-XX:~/wordpress$ kubectl apply -f deployment-
wordpress.yaml
deployment.apps/wordpress-deployment created

```

De website is nu uiteraard ook beschikbaar.



We vragen nu de pods op.

```
student@virt-k8s-XX:~/wordpress$ kubectl get pods
```

NAME	READY	STATUS	...
database-deployment-597cddbfc7-rvnk9	1/1	Running	...
wordpress-deployment-5cc9fd7ffc-rg4s8	1/1	Running	...

Je kan het aantal replicas (aantal pods) van een Deployment heel eenvoudig verhogen naar 2 op twee manieren: imperatief en declaratief.

Om het declaratief in te stellen open je een deployment YAML (we kiezen voor deployment-wordpress.yaml) en wijzigen bijvoorbeeld replicas: 1 naar replicas: 2 zodat er 2 i.p.v. 1 pod draait.

```
student@virt-k8s-XX:~/wordpress$ nano deployment-wordpress.yaml
```

```
...
spec:
```

```
    replicas: 2 # Pas dit aan
```

```
    selector:
```

```
...
```

Uiteraard dienen we de wijzigingen door te voeren.

```
student@virt-k8s-XX:~/wordpress$ kubectl apply -f deployment-wordpress.yaml
```

```
deployment.apps/wordpress-deployment configured
```

Als je de pods nu opvraagt zie je dat er 2 pods van Wordpress draaien.

```
student@virt-k8s-XX:~/wordpress$ kubectl get pods
```

NAME	READY	STATUS	
RESTARTS	AGE		
database-deployment-597cddbfc7-rvnk9	1/1	Running	0
8m23s			
wordpress-deployment-5cc9fd7ffc-clqln	1/1	Running	0
27s			
wordpress-deployment-5cc9fd7ffc-rg4s8	1/1	Running	0
8m19s			

Om het imperatief in te stellen zullen we het aantal pods van wordpress opschalen naar 3.

```
student@virt-k8s-XX:~/wordpress$ kubectl scale deployment  
wordpress-deployment --replicas=3
```

```
deployment.apps/wordpress-deployment scaled
```

```
student@virt-k8s-XX:~/wordpress$ kubectl get deployments
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
database-deployment	1/1	1	1	14m
wordpress-deployment	2/3	3	2	14m

En na enige tijd...

```
student@virt-k8s-XX:~/wordpress$ kubectl get deployments
```

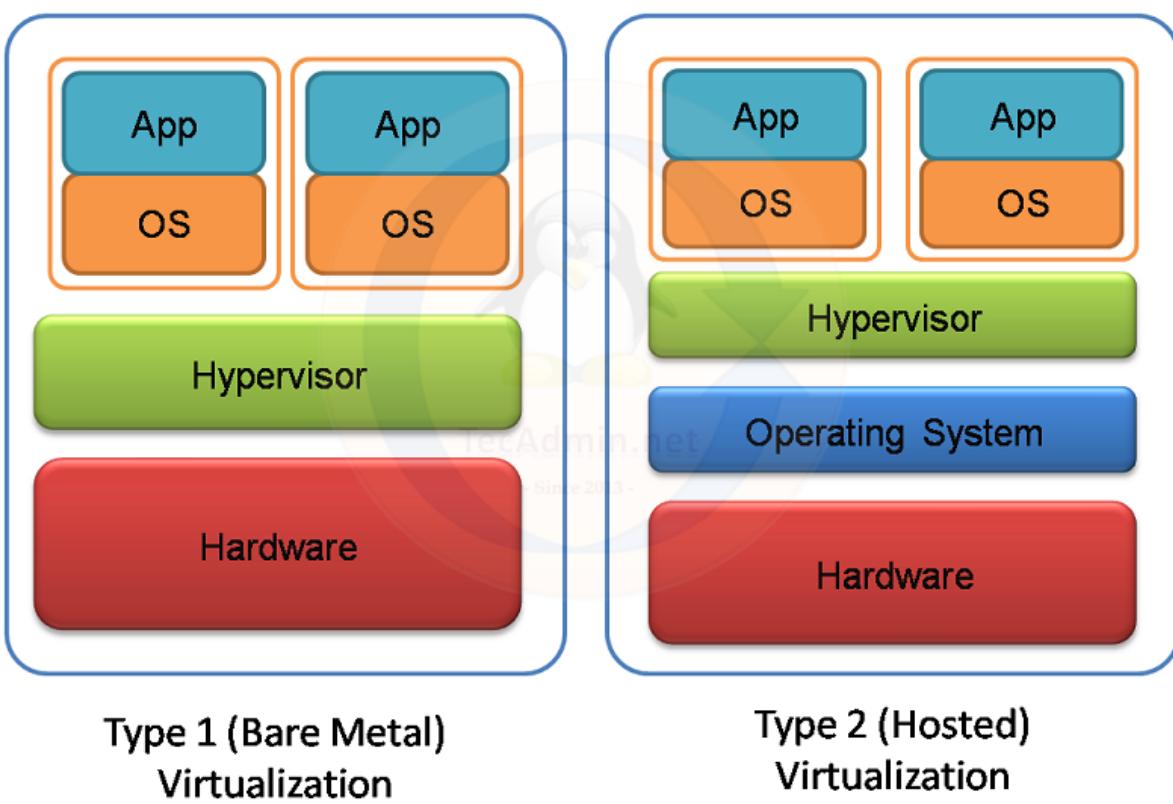
NAME	READY	UP-TO-DATE	AVAILABLE	AGE
database-deployment	1/1	1	1	15m
wordpress-deployment	3/3	3	3	14m

13 Installatie Proxmox

13.1 Inleiding

13.1.1 Hypervisors

Een Type 1 hypervisor heeft directe toegang tot het geheugen, de CPU en andere hardwarebronnen van de computer die de hypervisor direct virtualiseert, voorziet en beheert. Een Type 2 hypervisor moet toegang hebben tot de bronnen van de computer en deze virtualiseren, maar dit moet worden bereikt via het hostbesturingssysteem.



Voorbeelden:

- Type 2
 - VMware Workstation en Virtualbox
- Type 1
 - VMware ESXi, Microsoft Hyper-V en Proxmox

Proxmox VE, VMware ESXi en Hyper-V hebben verschillende gemeenschappelijke functies, die essentieel zijn voor virtualisatieplatformen:

- Virtualisatie van besturingssystemen: Alle drie ondersteunen het draaien van meerdere virtuele machines (VM's) op één fysieke server, waardoor hardware efficiënter wordt gebruikt.

- Live migratie: Ze bieden mogelijkheden om VM's live te migreren tussen hosts zonder downtime.
- Snapshot-functionaliteit: Ondersteuning voor het maken van snapshots van VM's, zodat snel teruggedraaid kan worden naar een vorige toestand.
- Clustering: Mogelijkheid om meerdere hosts te groeperen voor schaalbaarheid en failover-functionaliteit.
- Beheerinterface: Alle drie bieden beheer via een GUI (web- of applicatiegebaseerd) en command-line tools.
- Netwerkvirtualisatie: Ondersteuning voor virtuele netwerken, VLAN's en geavanceerde netwerkconfiguraties.
- Opslagopties: Ondersteuning voor verschillende opslagmethoden, zoals NFS, iSCSI en lokale opslag.
- Ondersteuning van diverse besturingssystemen: Ze kunnen een breed scala aan gast-OS'en draaien, waaronder Linux, Windows en BSD.
- Integratie met externe tools: Alle drie kunnen worden geïntegreerd met back-up-, monitoring- en beheertools.

13.1.2 Proxmox

Proxmox Virtual Environment (kortweg Proxmox VE) is een open source virtualisatieplatform, sinds 2008 ontwikkeld door het Oostenrijkse Proxmox Server Solutions en vrijgegeven onder de GNU AGPL licentie. Het ondersteunt zowel VM's (virtuele machines) op basis van KVM (Linux Kernel-based Virtual Machine) alsook containers op basis van Linux Container LXC. Hun inkomsten genereren ze voornamelijk via betaalde ondersteuningsabonnementen. Deze abonnementen bieden toegang tot de Proxmox Enterprise Repository, die stabiele en geteste updates bevat, evenals technische ondersteuning op verschillende niveaus.

Kenmerk	Proxmox VE	VMware ESXi	Hyper-V
Type	Open-source virtualisatieplatform	Proprietair virtualisatieplatform	Microsoft virtualisatieplatform
Licentie	GPLv3 (gratis, betaalde support mogelijk)	Gratis basisversie, commerciële licenties	Inclusief in Windows Server-licenties
Host OS	Debian-gebaseerd	Op Linux-gebaseerd, eigen OS	Windows Server of Windows 10/11 Pro/Edu (in beperkte mate)
Beheerinterface	Webinterface, CLI	Webinterface, VMware vSphere Client	Hyper-V Manager, PowerShell, Windows Admin Center

Ondersteunde VM-formaten	KVM, LXC	VMDK	VHD, VHDX
Opslagopties	ZFS, Ceph, LVM, NFS, iSCSI, en meer	VMFS, NFS, iSCSI	CSV, SMB, NFS, iSCSI
Back-ups	Ingebouwd (via GUI of CLI)	Externe tools zoals Veeam	Windows Backup, externe tools
Kosten	Gratis, optioneel supportabonnement	Hoog, afhankelijk van editie	Inclusief bij Windows Server
Schalingsopties	Schaalbaar met clustering en Ceph	Hoog, afhankelijk van licentie	Beperkt door Windows Server-functionaliteit
Integratie met containers	Ja (via LXC)	Nee	Beperkt (via Windows Containers)
Community Support	Groot, actief	Klein, afhankelijk van commerciële support	Gematigd, via Microsoft en gebruikersfora
Updates	Regelmatig, open-source	Regelmatig, vereist betaald contract	Regelmatig via Windows Update

13.1.2.1 Hardwarevereisten

Proxmox VE draait op Intel EMT64 of AMD64 CPU's met ondersteuning voor Intel VT of AMD-V. Je hebt minimaal 4GB RAM en 32GB opslag nodig.

Wij zullen Proxmox VE installeren in VMware Workstation. Nested Virtualization is aldus nodig.

In een testomgeving kan je aan de slag met een eenvoudige NUC en in een productieomgeving kan je server-grade hardware gebruiken. Wanneer de hardware onderdelen ondersteund worden door Debian Linux, zal Proxmox VE erop werken.

Opgelet: aangezien we Proxmox in een VM installeren en VM's gaan draaien in Proxmox is nested virtualization nodig! Zie bijlage C.

13.1.2.2 Hyperconverged Infrastructure (HCI)

Is niets nieuw, maar kent de laatste jaren een sterke groei in een kmo omgeving omwille van de kostenbesparing. Een traditionele high-availability omgeving bestaat bijvoorbeeld uit een cluster van 3 servers voor de compute nodes (de servers waarop de VM's draaien) en uit een cluster van 3 servers voor de centrale opslagruimte. Dat wilt zeggen: 6 fysieke servers. HCI combineert deze twee clusters tot een cluster waardoor de compute nodes en de opslag op dezelfde servers draaien.

Binnen Proxmox VE kan dat gedaan worden met behulp van Ceph. Zo kan je perfect een cluster van meerdere Proxmox VE nodes met geïntegreerd Ceph cluster voor gedeelde opslagruimte opzetten.

13.1.2.3 Licenties en het verdienmodel

Proxmox VE is vrijgegeven onder de open source licentie GNU AGPL v3 en is gratis te downloaden en te gebruiken.

Proxmox VE gebruikt als besturingssysteem Debian Linux. In de installatie is de gratis Proxmox "community" repository (repo) inbegrepen. Tegen betaling kan ook een "enterprise" repo gebruikt worden.

Het verschil tussen de "community" en de "enterprise" repository is tweeledig: de updates van Proxmox VE worden eerst in de "community" repo vrijgegeven en pas enkele weken nadat in de "enterprise" repo. Zo kan de "community" repo dus nog kleine bugs bevatten. Al wordt de "community" repo zeker niet als testplatform gezien, het is het "beta" stadium dan al gepasseerd. Daarnaast bevatten sommige "enterprise" abonnementen ook ondersteuning.

Tussen de updates van "community" en de "enterprise" repo zitten dus enkele weken verschil, maar qua functionaliteit zijn ze 1-op-1 identiek. De "enterprise" variant bevat dus niet meer functionaliteiten dan de "community" variant.

In een professionele omgeving kies je dus voor de "enterprise" repo, puur omwille van de zekerheid van stabiliteit. Alle nodes in een cluster moeten over dezelfde licentie beschikken, je kan ze niet mixen.

De "enterprise" licentie betaal je per bezette CPU socket. Je wordt dus niet op het aantal cores afgererekend, maar op het aantal bezette CPU sockets, onafhankelijk of je een Intel i3 1215U met 6-cores gebruikt of een AMD EPYC Milan 7643 met 48 cores. Sinds vele jaren en tot op dit moment zijn er slechts vier licentiemodellen, waardoor je niet hoeft te grasduinen doorheen tal van licenties.

Meer info: <https://www.proxmox.com/en/proxmox-virtual-environment/pricing>.

13.1.2.4 Beheer

Voor het beheer van Proxmox VE heb je geen speciale software of een management server nodig. Iedere Proxmox VE installatie bevat een webgebaseerde beheerdersinterface waarmee je die

specifieke server en het volledige cluster kan beheren. De Proxmox VE omgeving kan ook volledig via de command line en via API's beheerd worden, wat een verregaande automatisatie mogelijk maakt.

Het upgraden van een Proxmox VE installatie is daardoor ook eenvoudig. Iedere upgrade komt met duidelijke upgrade instructies, die vaak niet meer inhouden dan de upgrade uitvoeren via de webinterface. Wanneer je een cluster hebt, blijven alle VM's ook steeds online door de live-migratie mogelijkheid. Ook major updates van het Debian OS, bijvoorbeeld een upgrade van Debian 10 naar Debian 11, worden perfect en eenvoudig afgehandeld.

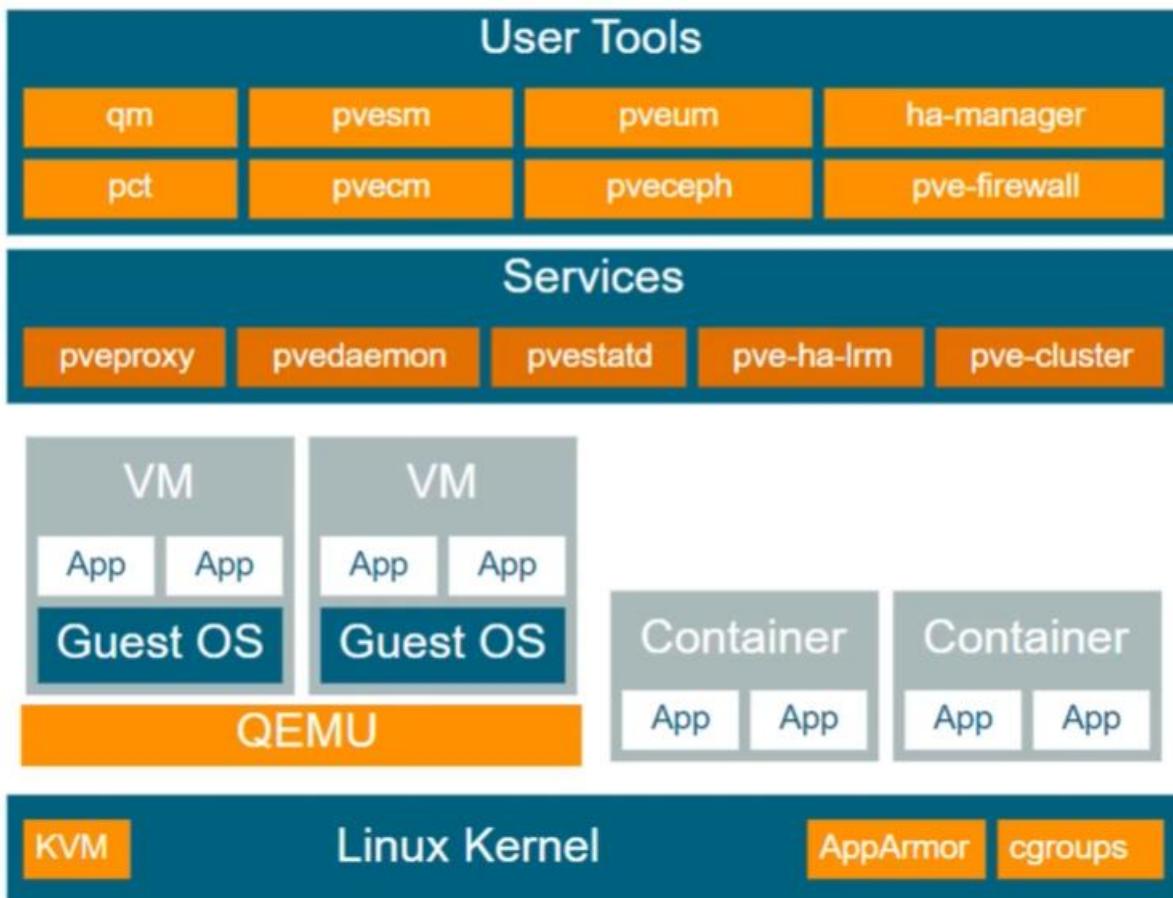
13.1.2.5 Backups en snapshots

Van iedere VM kunnen snapshots genomen worden. Proxmox VE komt ook met een geïntegreerde backup/restore oplossing om bijvoorbeeld dagelijks een back-up van de VM's en containers te maken. Wil je nog meer controle over de back-ups en een full-blown back-up oplossing, dan loont het de moeite om te kijken naar Proxmox BS (Proxmox Backup Server). Proxmox BS wordt op een externe server geïnstalleerd en doet dienst als een centrale back-up oplossing.

13.1.2.6 Beveiliging

De cluster nodes communiceren onderling via het beveiligde SSH. Proxmox VE komt met een geïntegreerde firewall op drie niveau's: cluster, host en VM. Op elk niveau kunnen specifieke firewall regels ingesteld worden met ondersteuning voor IPv4 en IPv6. Inloggen op de webinterface ondersteuning ook tweestapsverificatie via bijvoorbeeld Yubikeys.

13.1.2.7 Architectuur Proxmox



Dit diagram laat de architectuur van Proxmox Virtual Environment (PVE) zien. Hier is een beknopte uitleg van de verschillende lagen:

- Linux Kernel
 - De kernlaag van het systeem:
 - o KVM (Kernel-based Virtual Machine): Voor virtualisatie van volledige machines (VM's).
 - o AppArmor en cgroups: Voor beveiliging en resourcebeheer.
- QEMU
 - o Virtuele machines worden uitgevoerd met behulp van QEMU. Dit biedt hardwarevirtualisatie voor VM's.
- VM's en Containers
 - o VM's (Virtual Machines): Volledig geïsoleerde besturingssystemen (Guest OS) met applicaties.
 - o Containers: Lichtgewicht virtualisatie, waarbij alleen applicaties draaien in gescheiden omgevingen. Let op: dit zijn geen docker containers.
- Services
 - Deze laag bevat essentiële services voor het beheer van de Proxmox-omgeving:

- o pveproxy: Webinterface en API-toegang.
- o pvedaemon: Voor VM- en containerbeheer.
- o pvestatd: Voor het verzamelen van statusinformatie (resources en monitoring).

- pve-ha-lrm: High Availability (HA)-beheer.
 - pve-cluster: Voor clustering van meerdere nodes.
- User Tools
 - Hulpmiddelen voor het beheren van Proxmox:
 - qm: Voor het beheren van virtuele machines.
 - pct: Voor het beheren van containers.
 - pvesm: Opslagbeheer.
 - pvecm: Clusterbeheer.
 - pveum: Gebruikersbeheer.
 - ha-manager: High Availability-manager.
 - pveceph: Ceph-opslagbeheer.
 - pve-firewall: Firewallbeheer.

13.2 Proxmox VE installeren

13.2.1 Download ISO

Om Proxmox VE te installeren, download je eerst de installatiesoftware van <https://www.proxmox.com/en/downloads>.

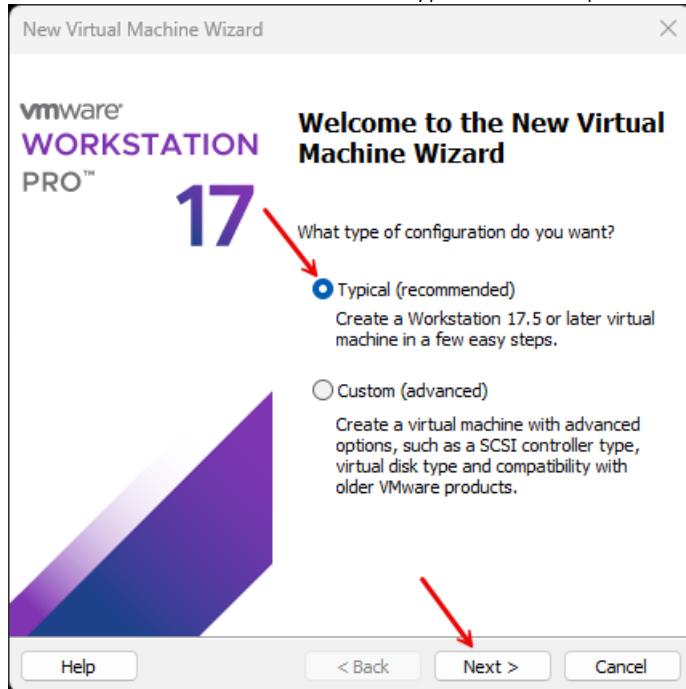
Latest Releases

Latest Releases			
Proxmox VE 9.1 ISO Installer	Version 9.1-1	File Size 1.83 GB	Last Updated November 19, 2025
	SHA256SUM 6d8f5afc78c0c66812d7272cde7c8b98be7eb54401ceb045400db05eb5ae6d22	Download	
Proxmox Backup Server 4.0 ISO Installer	Version 4.0-1	File Size 1.34 GB	Last Updated August 06, 2025
	SHA256SUM 208607b250164863b5731a29dd89569a123e6f385c5ec0939a4942357bf731e2	Download	Torrent
Proxmox Mail Gateway 9.0 ISO Installer	Version 9.0-1	File Size 1.63 GB	Last Updated October 01, 2025
	SHA256SUM 23bf1e50bba06f6fb50360740c0388462b921b414d2e4f3ea859e5150f498c8e	Download	Torrent

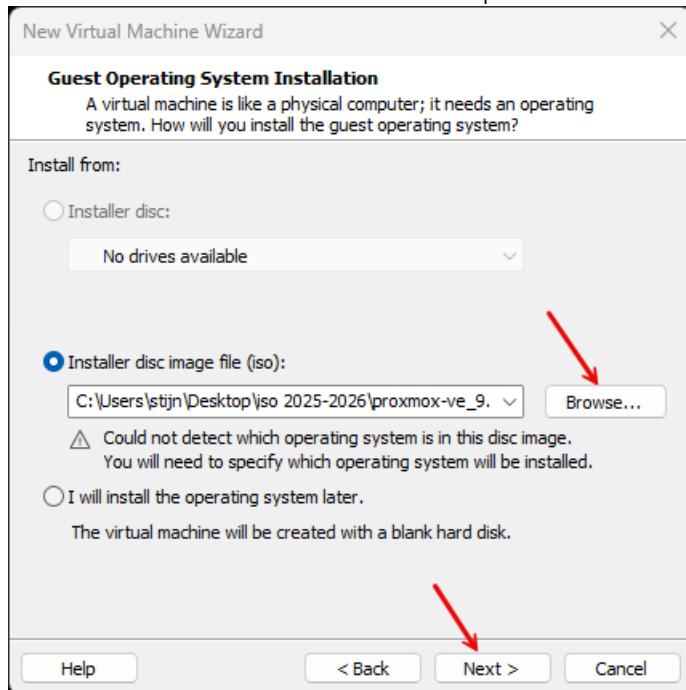
13.2.2 Aanmaken VM voor Proxmox

- Open uiteraard VMware Workstation 17 op uw systeem.
- Klik op File, New Virtual Machine.

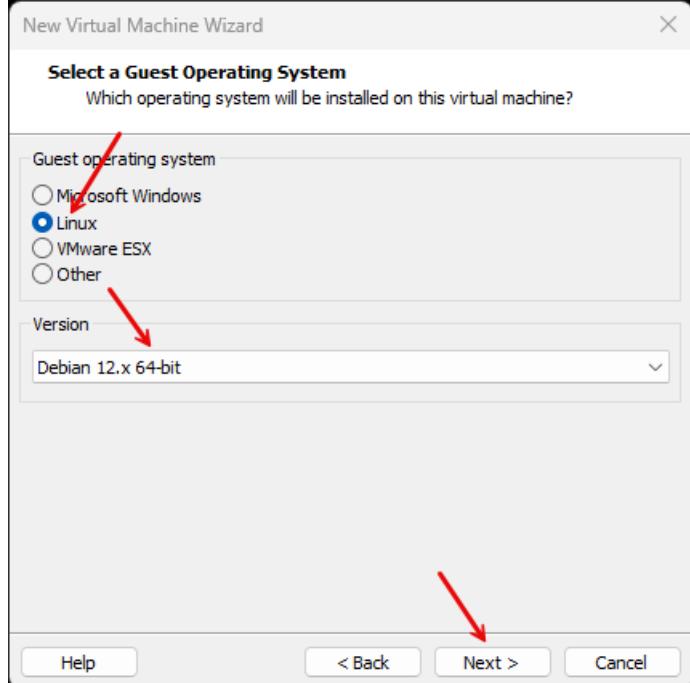
- Kies in onderstaand venster voor Typical en klik op Next.



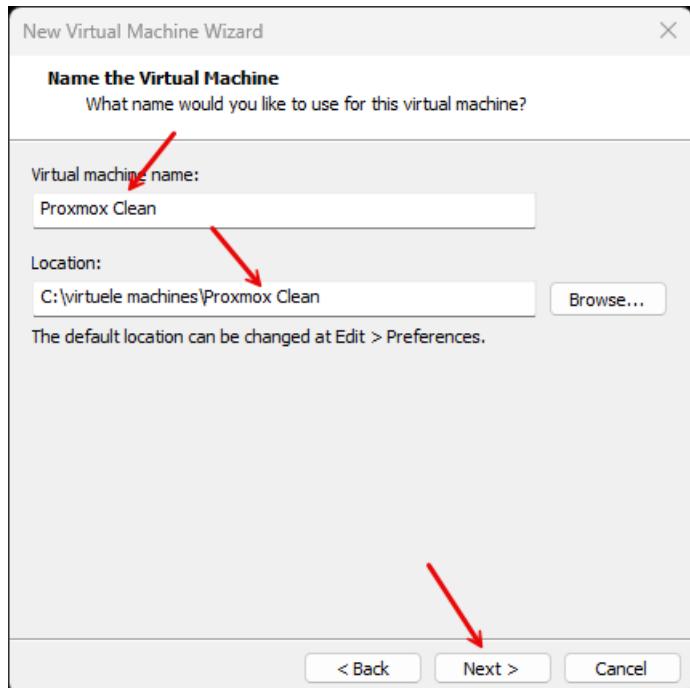
- Kies in het volgende venster de gedownloade Proxmox ISO-image door te klikken op Browse en het bestand te selecteren. Klik erna op Next.



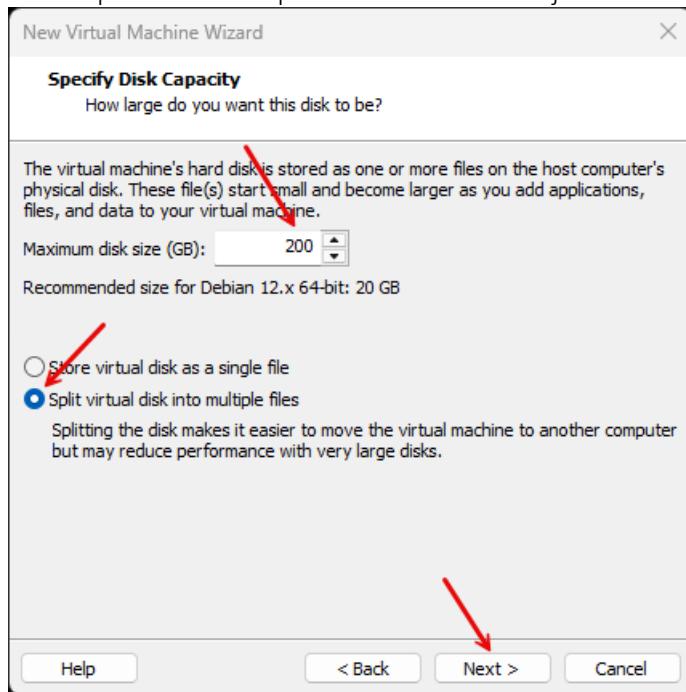
- Stel het gastbesturingssysteem in als Linux en de versie als Debian 12.x 64-bit.



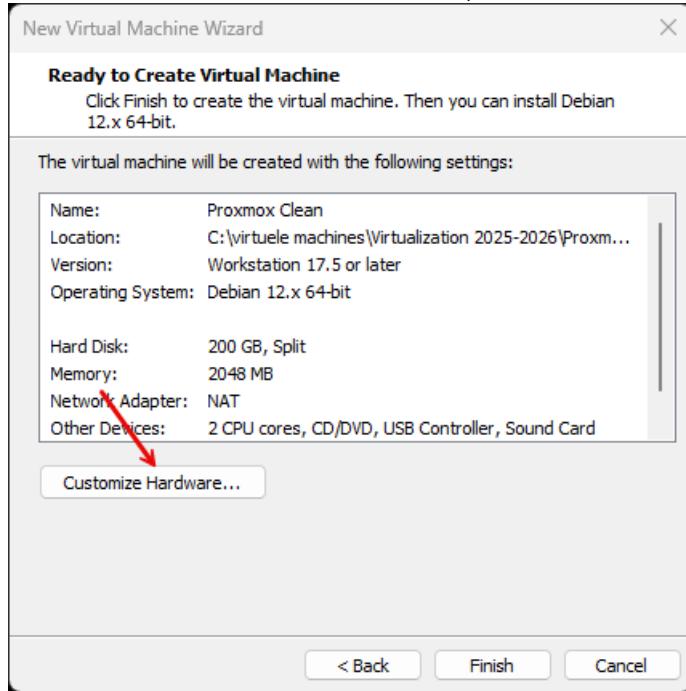
- Geef uw virtuele machine een naam en selecteer de locatie. Klik dan Next.



- Kies voor een schijf van 200 GB en klik op Next.
PS De optie i.v.m. het splitten van virtuele schijven is niet belangrijk voor ons.

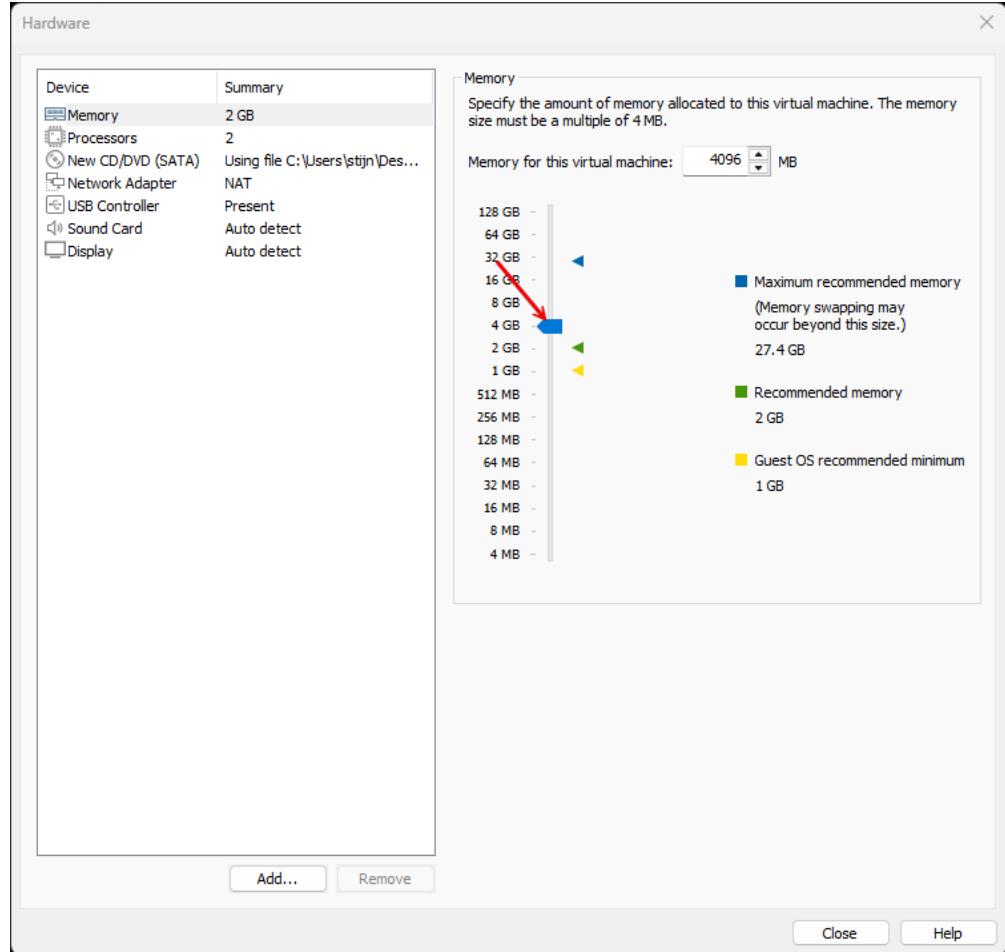


- Controleer de informatie en klik erna op Customize Hardware... .

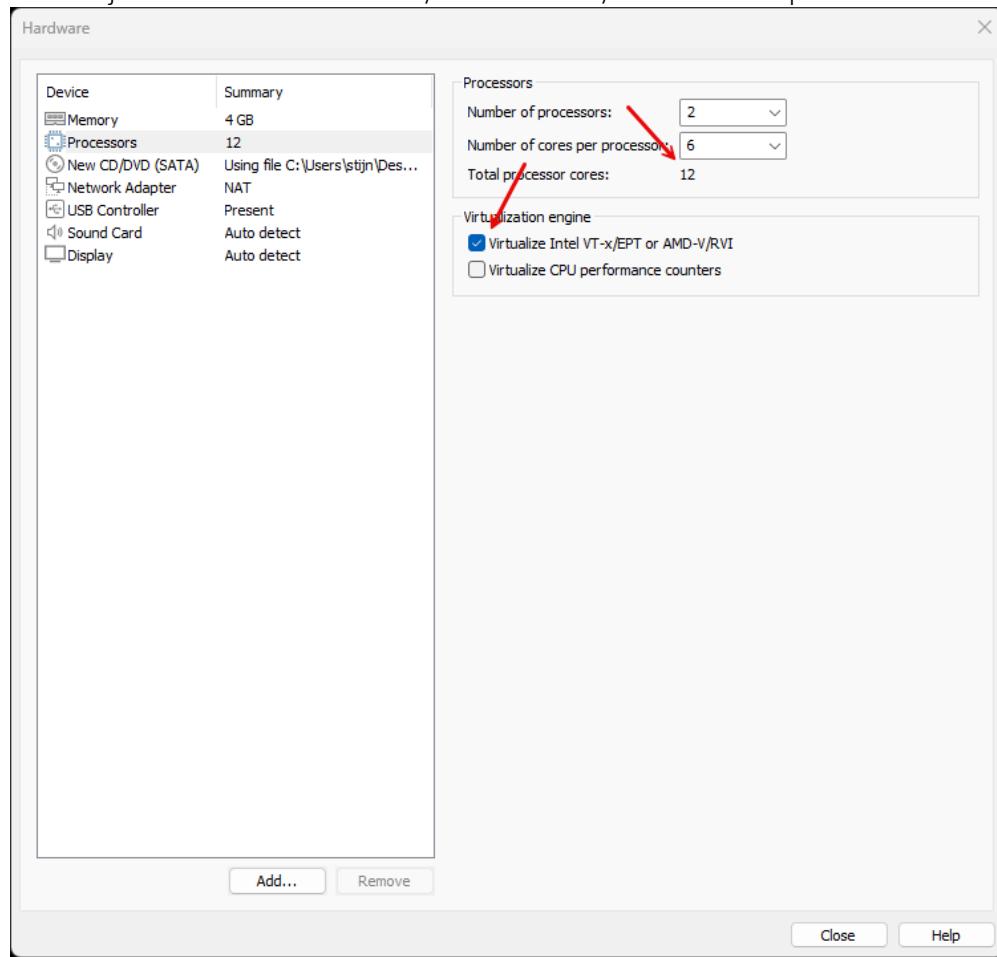


- Pas de instellingen aan.

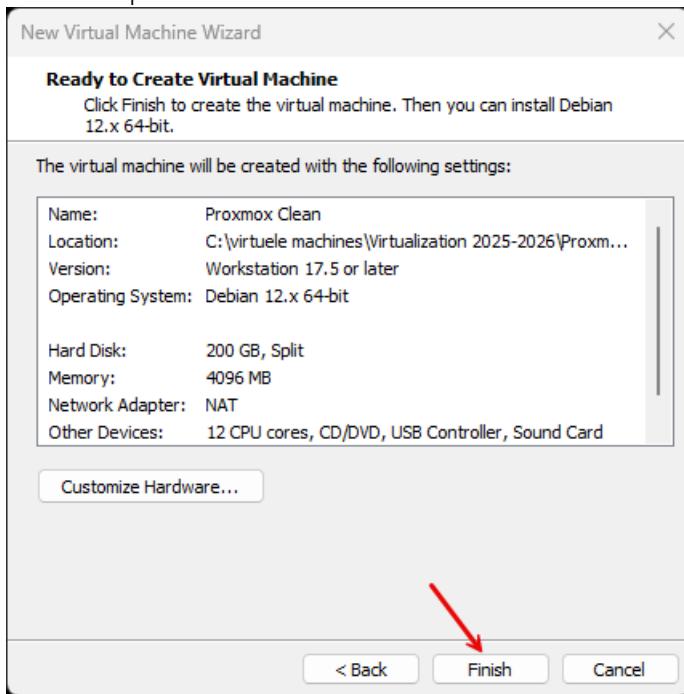
- Memory: minimaal 4GB.



- Processors: maximaliseer Total processor Cores (afhankelijk van je processor) en zet een vinkje voor Virtualize Intel VT-x/EPT or AMD-V/RVI. Klik erna op Close.

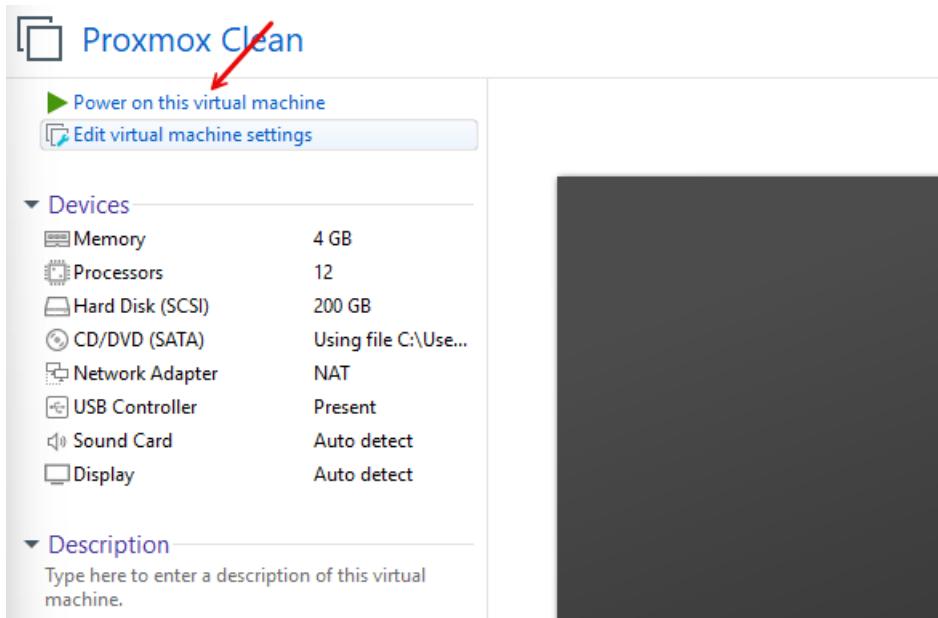


- Klik erna op Finish.



13.2.3 installatie in VM

- Zet de VM aan.

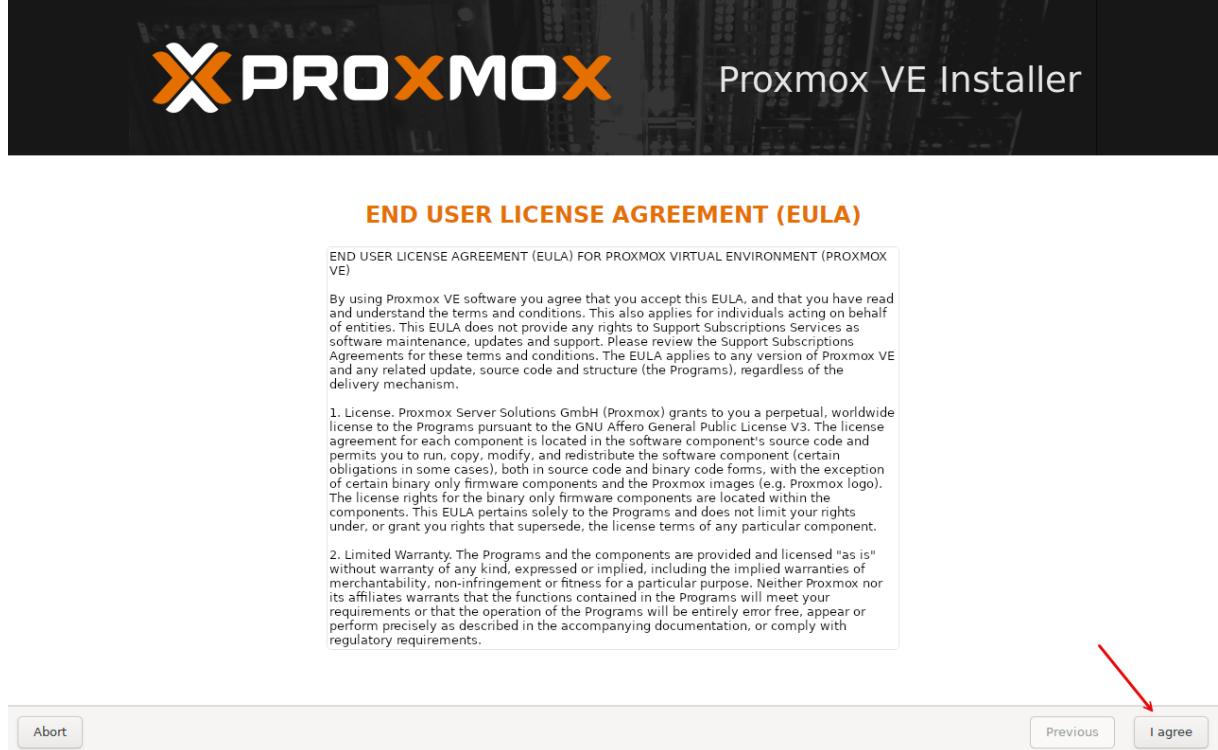


- Kies voor Install Proxmox VE (Graphical) door op <Enter> te drukken.

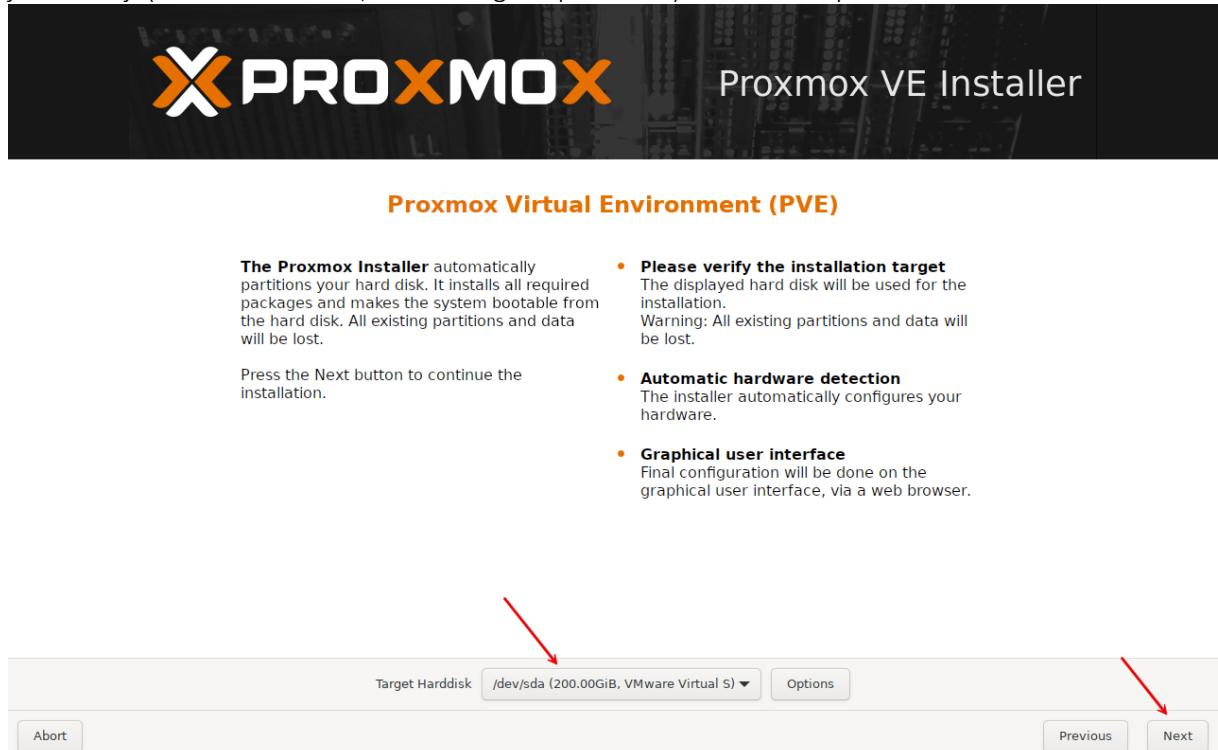
Proxmox VE 9.1 (iso release 1) - <https://www.proxmox.com/>



- Na een aantal tests volgt de gebruikerslicentie, klik op I agree.



- Onder in het scherm, een beetje onopvallend, staat de Target Harddisk. Selecteer hier de juiste schijf (er is er nu maar 1, dus dat is geen probleem) en klik dan op Next.



- Kies bij Country Belgium en bij Time Zone voor Europe/Brussels. Bij Layout selecteer je je toetsenbordindeling.

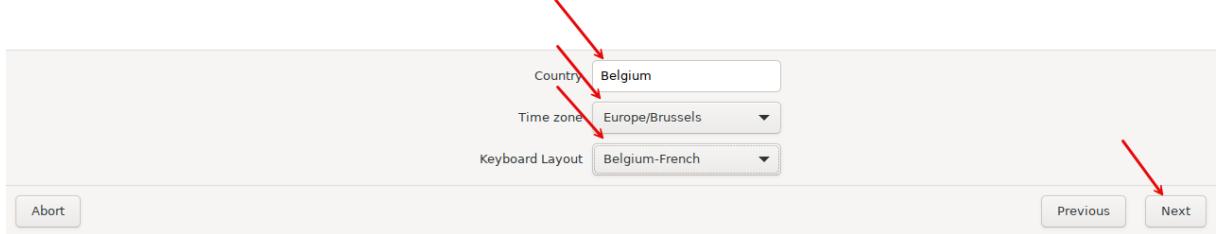


Location and Time Zone selection

The Proxmox Installer will set up your time zone and keyboard layout. Ensuring that your system behaves as intended once it is up and running.

Press the Next button to continue the installation.

- **Country:** Narrows down the available time zones to make selection easier.
- **Time Zone:** Automatically adjust daylight saving time.
- **Keyboard Layout:** Choose your keyboard layout.



- Configureer een wachtwoord (kijk naar de voorwaarden!) bij Password. Vul ook uw mailadres in.



Administration Password and Email Address

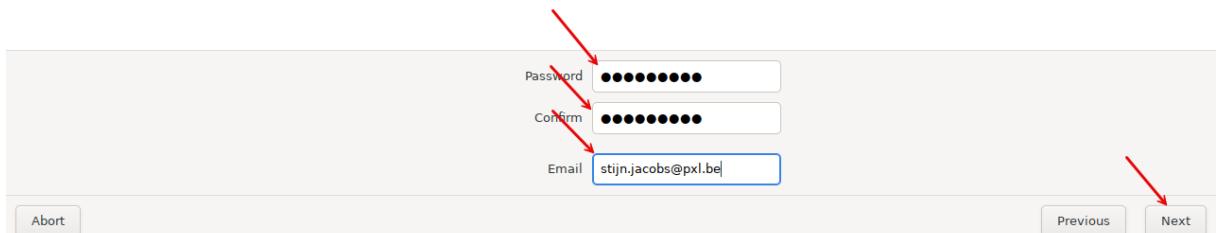
Proxmox Virtual Environment is a full featured, highly secure GNU/Linux system, based on Debian.

In this step, please provide the *root* password.

- **Password:** Please use a strong password. It must be at least 8 characters long, and contain a combination of letters, numbers, and symbols.

- **Email:** Enter a valid email address. Your Proxmox VE server will send important alert notifications to this email account (such as backup failures, high availability events, etc.).

Press the Next button to continue the installation.



- Als laatste kun je op het blad Management Network Configuration de naam en de netwerkconfiguratie kiezen. We laten de standaard staan aangezien we dit later kunnen

wijzigen.

The screenshot shows the 'Management Network Configuration' step of the Proxmox VE Installer. It includes instructions to verify network settings and a list of configuration options:

- Please verify** the displayed network configuration. You will need a valid network configuration to access the management interface after installing.
- After you have finished, press the Next button. You will be shown a list of the options that you chose during the previous steps.
- IP address (CIDR):** Set the main IP address and netmask for your server in CIDR notation.
- Gateway:** IP address of your gateway or firewall.
- DNS Server:** IP address of your DNS server.

Form fields include:
Management Interface: nic0 - 00:0c:29:14:f1:61 (e1000)
Hostname (FQDN): pve.localdomain
IP Address (CIDR): 192.168.112.133 / 24
Gateway: 192.168.112.2
DNS Server: 192.168.112.2
Checkboxes: Pin network interface names, Options
Buttons: Abort, Previous, Next

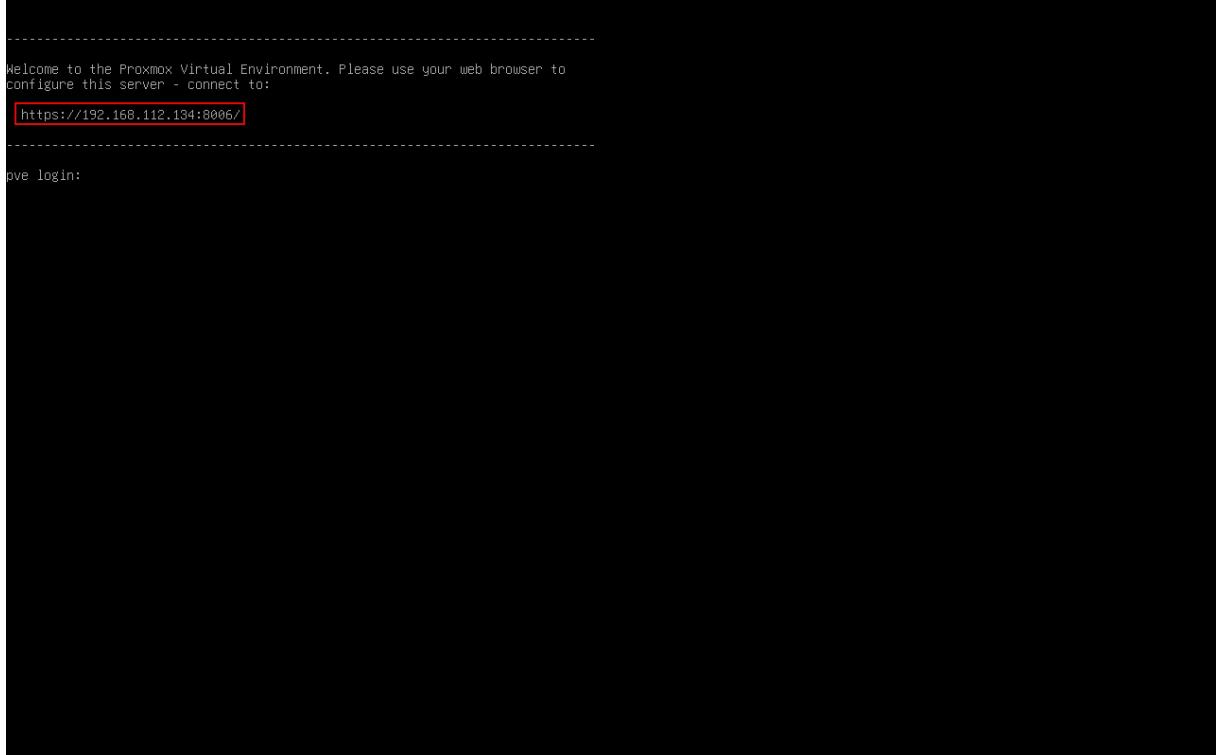
- Controleer tot slot de gekozen opties en klik op Install om Proxmox ook echt te installeren.

The screenshot shows the 'Summary' step of the Proxmox VE Installer. It displays the confirmed configuration options:

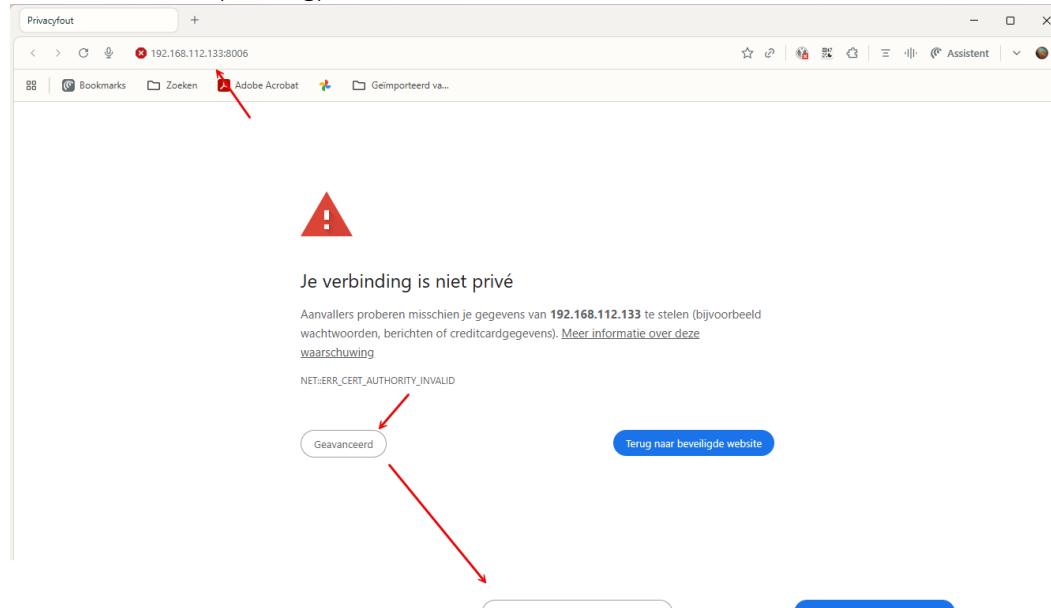
Option	Value
Filesystem:	ext4
Disk(s):	/dev/sda
Country:	Belgium
Timezone:	Europe/Brussels
Keymap:	fr-be
Email:	stijn.jacobs@pxl.be
Management Interface:	nic0
Hostname:	pve
IP CIDR:	192.168.112.133/24
Gateway:	192.168.112.2
DNS:	192.168.112.2

Checkboxes:
 Automatically reboot after successful installation
Buttons: Abort, Previous, Install

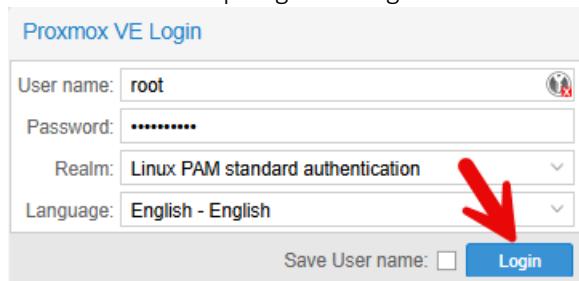
- Na installatie krijg je onderstaande na reboot...



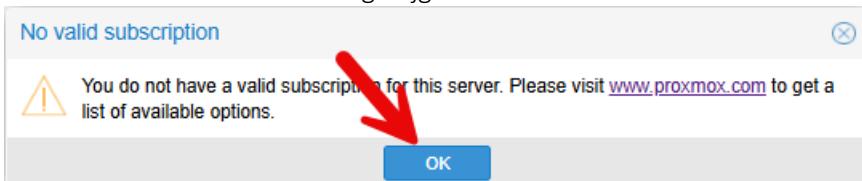
- Je ziet hoe je kan inloggen (<https://192.168.112.133:8006/>) via de website om Proxmox VE te beheren.
- Ga naar je browser op je hostmachine en typ <https://192.168.112.133:8006/> in.
 - o Aangezien er geen geldig certificaat is klik je op Geavanceerd, Doorgaan naar 192.168.112.133 (onveilig).



- Log u nu in met onderstaande gegevens:
 - o User name: root
 - o Password: het wachtwoord dat je gekozen hebt tijdens de installatie.
 - o We laten de taal op English – English staan.



- Je zal nu onderstaande melding krijgen.



Je ziet nu de grafische interface van Proxmox.

De interface is intuïtief en zeer functioneel zodra u ermee vertrouwd raakt.

De kracht van Proxmox ligt in de manier waarop het is ontworpen om beheerders toegang te geven tot de volledige omgeving via de webinterface van elk aangesloten knooppunt (node). Hier volgt een korte uitleg van de belangrijkste elementen:

- Datacenter als hoogste organisatieniveau
Het datacenter is de centrale organisatorische eenheid binnen Proxmox. Vanuit het datacenter worden alle Proxmox VE-knooppunten beheerd, samen met hun gekoppelde

virtuele machines (VM's), containers en opslagconfiguraties. Het datacenter biedt een overzicht van de gehele infrastructuur.

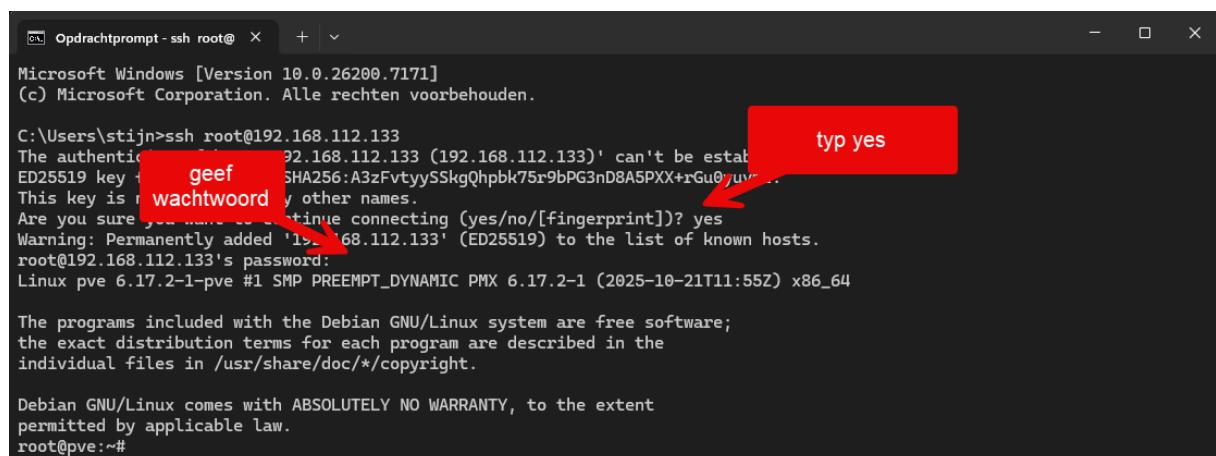
- Knooppunten en hun componenten
Elk knooppunt vertegenwoordigt een individuele host in het cluster. Hier is er maar 1 knooppunt: pve. Elk knooppunt heeft toegang tot zijn eigen netwerk, opslag en configuratie. Wanneer u een specifiek knooppunt selecteert, opent zich een instellingenpaneel waarin u de belangrijkste configuraties kunt beheren, inclusief netwerk- en opslaginstellingen, rechtstreeks vanuit de GUI.
- Taakbeheer en logging
Onderaan de interface bevindt zich een realtime "task log". Dit is een stroom van operationele logboekvermeldingen die elke actie of REST API-aanroep weergeven. Dit is vergelijkbaar met een activiteitenlogboek in andere platforms zoals Azure, en biedt transparantie over wat er in het systeem gebeurt. Het omvat ook clusterstatus, heartbeat-signalen en andere systeemmeldingen.
- Gebruiksvriendelijke bedieningsknoppen
Rechtsboven in de interface vindt u bedieningselementen voor virtuele machines en containers. Hier kunt u eenvoudig bulkacties uitvoeren, zoals het opstarten, stoppen of migreren van VM's en containers. Ook biedt deze sectie mogelijkheden voor externe toegang en geavanceerd beheer.

Proxmox biedt een goed doordachte webinterface die zowel beginners als ervaren beheerders een efficiënt en gecentraliseerd beheerplatform biedt. Met toegang tot zowel GUI-functionaliteit als command-line opties (via SSH of de CLI) is Proxmox veelzijdig genoeg om aan de behoeften van diverse organisaties te voldoen.

13.3 Repositories aanpassen

We maken een ssh-verbinding om de repositories aan te passen.

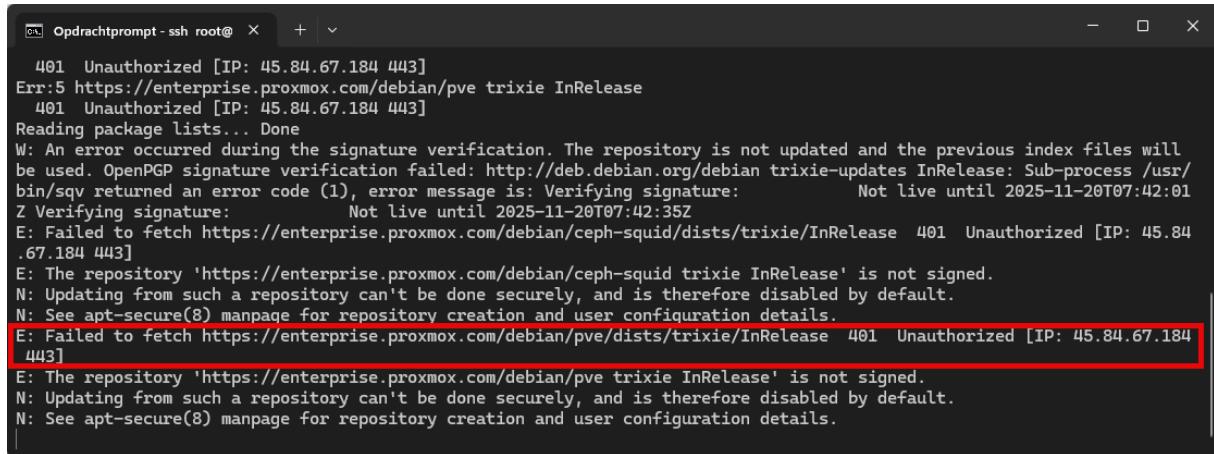
```
C:\Users\stijn>ssh root@192.168.112.133
```



```
Opdrachtprompt - ssh root@192.168.112.133 + ▾ Microsoft Windows [Version 10.0.26200.7171] (c) Microsoft Corporation. Alle rechten voorbehouden. C:\Users\stijn>ssh root@192.168.112.133 The authenticity of host '192.168.112.133 (192.168.112.133)' can't be established. ED25519 key fingerprint is SHA256:A3zFvtyySSkgQhpbk75r9bPG3nD8A5PXX+rGu0juw... This key is known by other names. Are you sure you want to continue connecting (yes/no/[fingerprint])? yes Warning: Permanently added '192.168.112.133' (ED25519) to the list of known hosts. root@192.168.112.133's password: Linux pve 6.17.2-1-pve #1 SMP PREEMPT_DYNAMIC PMX 6.17.2-1 (2025-10-21T11:55Z) x86_64 The programs included with the Debian GNU/Linux system are free software; the exact distribution terms for each program are described in the individual files in /usr/share/doc/*copyright. Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent permitted by applicable law. root@pve:~#
```

We voeren onderstaande uit.

```
root@pve:~# apt-get update && apt-get upgrade
```



```
401 Unauthorized [IP: 45.84.67.184 443]
Err:5 https://enterprise.proxmox.com/debian/pve trixie InRelease
  401 Unauthorized [IP: 45.84.67.184 443]
Reading package lists... Done
W: An error occurred during the signature verification. The repository is not updated and the previous index files will be used. OpenPGP signature verification failed: http://deb.debian.org/debian trixie-updates InRelease: Sub-process /usr/bin/sqv returned an error code (1), error message is: Verifying signature: Not live until 2025-11-20T07:42:01
Z Verifying signature: Not live until 2025-11-20T07:42:35Z
E: Failed to fetch https://enterprise.proxmox.com/debian/ceph-squid/dists/trixie/InRelease 401 Unauthorized [IP: 45.84.67.184:443]
E: The repository 'https://enterprise.proxmox.com/debian/ceph-squid trixie InRelease' is not signed.
N: Updating from such a repository can't be done securely, and is therefore disabled by default.
N: See apt-secure(8) manpage for repository creation and user configuration details.
E: Failed to fetch https://enterprise.proxmox.com/debian/pve/dists/trixie/InRelease 401 Unauthorized [IP: 45.84.67.184:443]
E: The repository 'https://enterprise.proxmox.com/debian/pve trixie InRelease' is not signed.
N: Updating from such a repository can't be done securely, and is therefore disabled by default.
N: See apt-secure(8) manpage for repository creation and user configuration details.
```

Je zal zien dat je foutmeldingen krijgt. Dit komt omdat Proxmox ervan uit gaat na installatie dat je een betalend abonnement hebt. Je bent niet geauthentificeerd om die repositories te gebruiken.

Je kan wel gebruik maken van de (volledig legale) gratis repositories.

- We stellen in dat we geen gebruik maken van de commerciële repository.

```
root@pve:~# nano /etc/apt/sources.list.d/pve-enterprise.sources
```

Zet voor elke regel een #.

```
#Types: deb
```

```
#URIs: https://enterprise.proxmox.com/debian/pve
```

```
#Suites: trixie
```

```
#Components: pve-enterprise
```

```
#Signed-By: /usr/share/keyrings/proxmox-archive-keyring.gpg
```

Sla het bestand op en sluit het.

- We maken nu een no-subscription repo.
Maak hiervoor onderstaand bestand aan.

```
root@pve:~# nano /etc/apt/sources.list.d/pve-no-subscription.sources
```

Geef dat bestand onderstaande inhoud.

```
Types: deb
```

```
URIs: http://download.proxmox.com/debian/pve
```

Suites: trixie
Components: pve-no-subscription
Signed-By: /usr/share/keyrings/proxmox-archive-keyring.gpg

Sla het bestand op en sluit het.

- Ga naar je Ceph-repo-configuratiebestand. Comment de “enterprise”-regels uit (zoals gedaan voor pve-enterprise) en voeg de no-subscription Ceph repo eronder toe:

```
root@pve:~# nano /etc/apt/sources.list.d/ceph.sources
```

```
#Types: deb
```

```
#URIs: https://enterprise.proxmox.com/debian/ceph-squid
```

```
#Suites: trixie
```

```
#Components: enterprise
```

```
#Signed-By: /usr/share/keyrings/proxmox-archive-keyring.gpg
```

```
Types: deb
```

```
URIs: http://download.proxmox.com/debian/ceph-squid
```

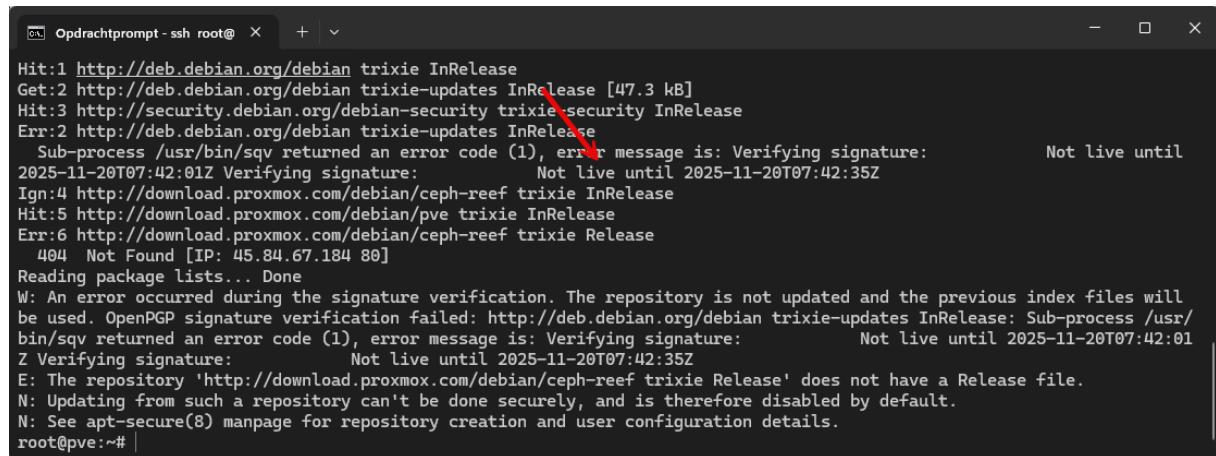
```
Suites: trixie
```

```
Components: no-subscription
```

```
Signed-By: /usr/share/keyrings/proxmox-archive-keyring.gpg
```

We voeren terug uit.

```
root@pve:~# apt-get update && apt-get upgrade
```

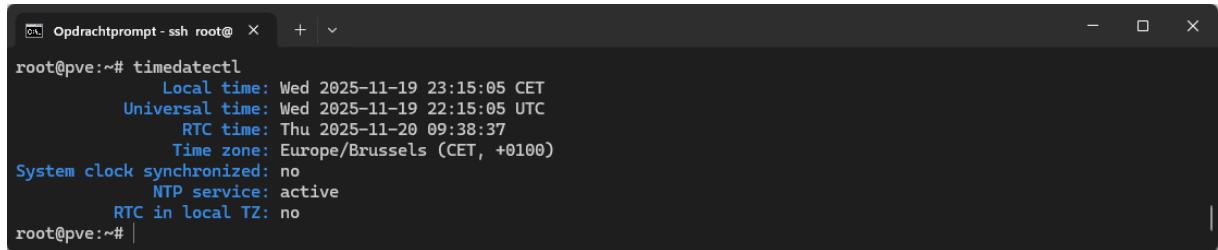


```
Opdrachtprompt - ssh root@ ~ + | x - □ ×
Hit:1 http://deb.debian.org/debian trixie InRelease
Get:2 http://deb.debian.org/debian trixie-updates InRelease [47.3 kB]
Hit:3 http://security.debian.org/debian-security trixie-security InRelease
Err:2 http://deb.debian.org/debian trixie-updates InRelease
  Sub-process /usr/bin/sqv returned an error code (1), error message is: Verifying signature: Not live until
  2025-11-20T07:42:01Z Verifying signature: Not live until 2025-11-20T07:42:35Z
Ign:4 http://download.proxmox.com/debian/ceph-reef trixie InRelease
Hit:5 http://download.proxmox.com/debian/pve trixie InRelease
Err:6 http://download.proxmox.com/debian/ceph-reef trixie Release
  404 Not Found [IP: 45.84.67.184:80]
Reading package lists... Done
W: An error occurred during the signature verification. The repository is not updated and the previous index files will
be used. OpenPGP signature verification failed: http://deb.debian.org/debian trixie-updates InRelease: Sub-process /usr/
bin/sqv returned an error code (1), error message is: Verifying signature: Not live until 2025-11-20T07:42:01
Z Verifying signature: Not live until 2025-11-20T07:42:35Z
E: The repository 'http://download.proxmox.com/debian/ceph-reef trixie Release' does not have a Release file.
N: Updating from such a repository can't be done securely, and is therefore disabled by default.
N: See apt-secure(8) manpage for repository creation and user configuration details.
root@pve:~# |
```

Deze foutmelding heeft te maken met de timeserver.

Check de huidige tijd

```
root@pve:~# timedatectl
```



```
root@pve:~# timedatectl
      Local time: Wed 2025-11-19 23:15:05 CET
      Universal time: Wed 2025-11-19 22:15:05 UTC
            RTC time: Thu 2025-11-20 09:38:37
           Time zone: Europe/Brussels (CET, +0100)
System clock synchronized: no
          NTP service: active
    RTC in local TZ: no
root@pve:~# |
```

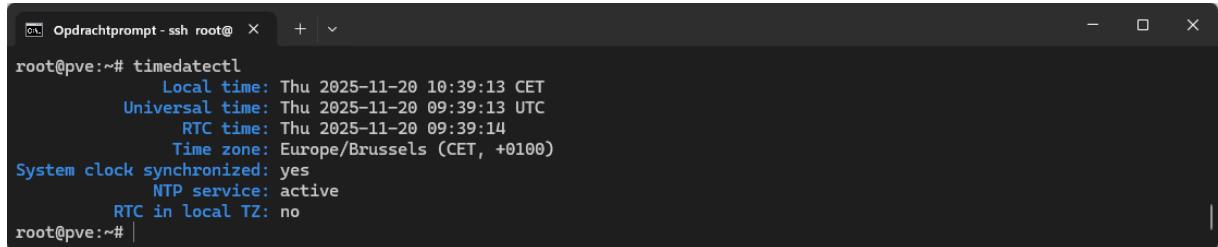
De tijd is niet correct...

Herstart de tijdserver.

```
root@pve:~# systemctl restart chronyd
```

Vraag nu terug de tijd op.

```
root@pve:~# timedatectl
```



```
root@pve:~# timedatectl
      Local time: Thu 2025-11-20 10:39:13 CET
      Universal time: Thu 2025-11-20 09:39:13 UTC
            RTC time: Thu 2025-11-20 09:39:14
           Time zone: Europe/Brussels (CET, +0100)
System clock synchronized: yes
          NTP service: active
    RTC in local TZ: no
root@pve:~# |
```

We voeren nu weer uit.

```
root@pve:~# apt-get update && apt-get upgrade
```

Er is nu geen fout meer die voorkomt...

```

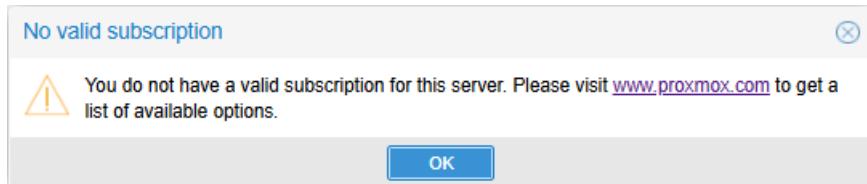
Opdrachtprompt - ssh root@ ~ + | -
Ign:4 http://download.proxmox.com/debian/ceph-reef trixie InRelease
Hit:5 http://download.proxmox.com/debian/pve trixie InRelease
Err:6 http://download.proxmox.com/debian/ceph-reef trixie Release
  404  Not Found [IP: 45.84.67.184 80]
Reading package lists... Done
E: The repository 'http://download.proxmox.com/debian/ceph-reef trixie Release' does not have a Release file.
N: Updating from such a repository can't be done securely, and is therefore disabled by default.
N: See apt-secure(8) manpage for repository creation and user configuration details.
root@pve:~# nano /etc/apt/sources.list.d/ceph.sources
root@pve:~# apt-get update && apt-get upgrade
Hit:1 http://security.debian.org/debian-security trixie-security InRelease
Hit:2 http://deb.debian.org/debian trixie InRelease
Hit:3 http://deb.debian.org/debian trixie-updates InRelease
Get:4 http://download.proxmox.com/debian/ceph-squid trixie InRelease [2,736 B]
Hit:5 http://download.proxmox.com/debian/pve trixie InRelease
Get:6 http://download.proxmox.com/debian/ceph-squid trixie/no-subscription amd64 Packages [33.2 kB]
Fetched 35.9 kB in 0s (75.7 kB/s)
Reading package lists... Done
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
The following packages will be upgraded:
  pve-docs pve-qemu-kvm qemu-server
3 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Need to get 49.2 MB of archives.
After this operation, 13.3 kB of additional disk space will be used.
Do you want to continue? [Y/n] y

```

De installatie verloopt nu zonder problemen.

13.4 Abonnementswaarschuwing uitzetten

We willen onderstaande melding niet meer zien na opstarten.



Open het JavaScript-bestand dat verantwoordelijk is voor de waarschuwing:

```
root@pve:~# nano /usr/share/javascript/proxmox-widget-toolkit/proxmoxlib.js
```

Zoek volgende regel:

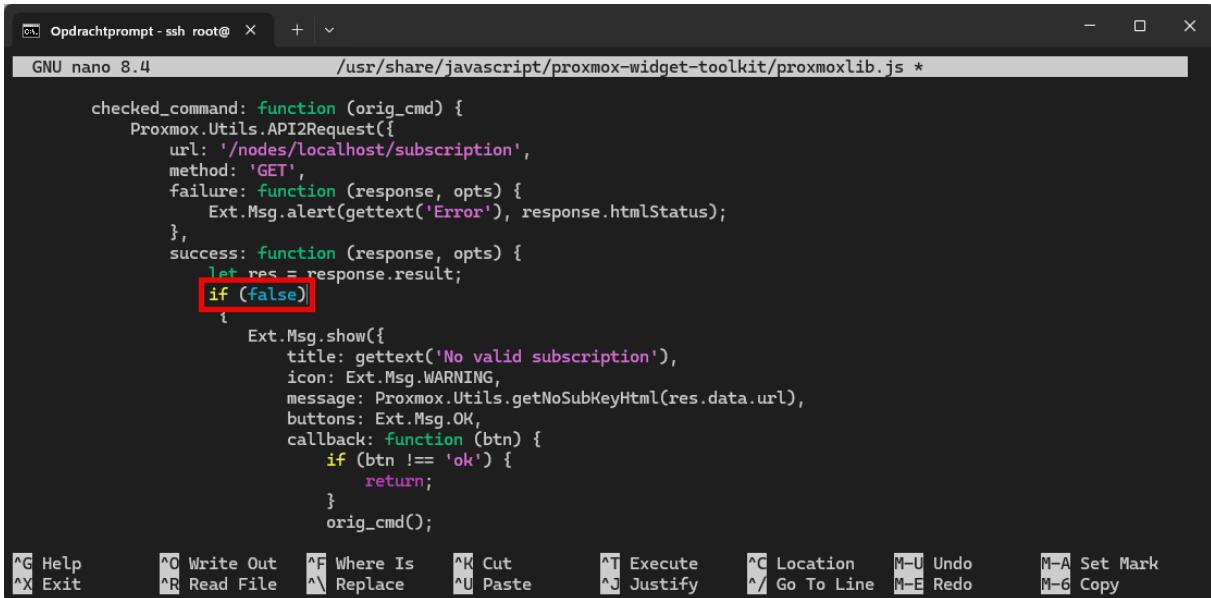
```

if (
    res === null ||
    res === undefined ||
    !res ||
    res.data.status.toLowerCase() !==
    'active'
)

```

Vervang dit door:

If (false)



```
GNU nano 8.4          /usr/share/javascript/proxmox-widget-toolkit/proxmoxlib.js *
```

```
checked_command: function (orig_cmd) {
    Proxmox.Utils.API2Request({
        url: '/nodes/localhost/subscription',
        method: 'GET',
        failure: function (response, opts) {
            Ext.Msg.alert(gettext('Error'), response.htmlStatus);
        },
        success: function (response, opts) {
            let res = response.result;
            if (false)
                Ext.Msg.show({
                    title: gettext('No valid subscription'),
                    icon: Ext.Msg.WARNING,
                    message: Proxmox.Utils.getNoSubKeyHtml(res.data.url),
                    buttons: Ext.Msg.OK,
                    callback: function (btn) {
                        if (btn !== 'ok') {
                            return;
                        }
                        orig_cmd();
                    }
                });
        }
    });
}
```

^G Help ^O Write Out ^F Where Is ^K Cut ^T Execute ^C Location M-U Undo M-A Set Mark
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^/ Go To Line M-E Redo M-G Copy

Sla het bestand uiteraard op.

De melding zal nu niet meer getoond worden na reboot

13.5 Meerdere Proxmox Servers

Het hebben van meerdere Proxmox-servers biedt meerdere voordelen:

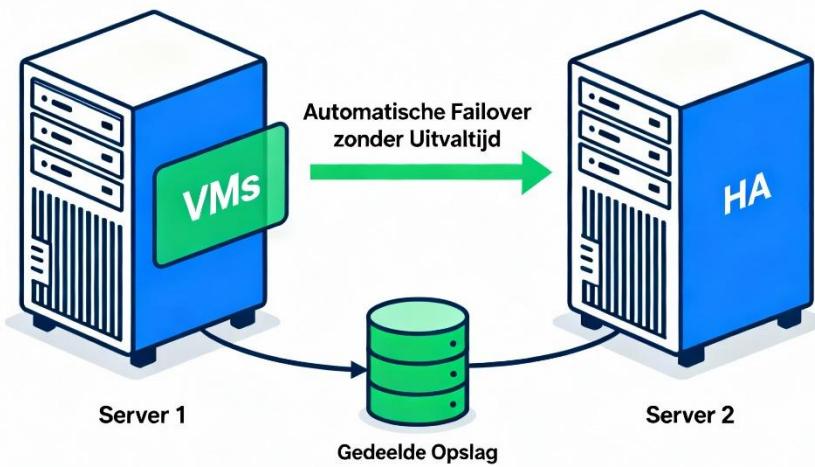
- High Availability (HA) en Failover

Een van de grootste voordelen van meerdere Proxmox-servers is het creëren van een High Availability Cluster. Dit betekent dat als één server uitvalt, de VM's of containers automatisch worden overgenomen door de andere server. Dit is essentieel voor bedrijfskritische toepassingen waarbij downtime niet acceptabel is.

Shared storage (zoals NFS, Ceph, of iSCSI) wordt gebruikt om VM-data tussen beide servers

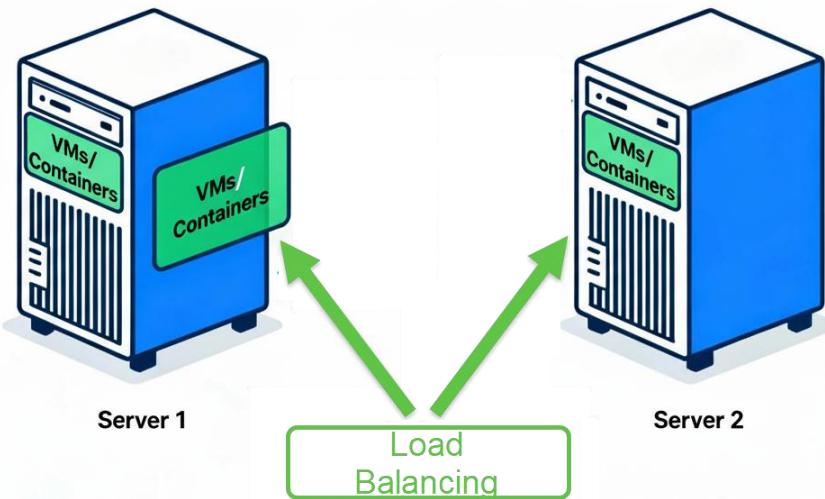
toegankelijk te maken. Als een server faalt, start de andere server automatisch de VM's op.

proxmox High Availability (HA)



- Belasting verdelen (Load Balancing)
Met twee servers kun je de belasting van de VM's en containers beter verdelen, zodat geen enkele server overbelast raakt.
Je kunt specifieke taken of VM's toewijzen aan de ene server en andere taken aan de tweede server. Dit verbetert de prestaties en voorkomt vertragingen in een enkel systeem.

proxmox Load Balancing



- Backups en Disaster Recovery
Twee servers maken het eenvoudiger om een goed backup- en herstelplan te implementeren:
 - o Backups: Je kunt snapshots of volledige backups van VM's op de ene server naar de andere repliceren. Hierdoor heb je een veilige kopie van je VM's.
 - o Disaster Recovery: In geval van een hardwarestoring kun je de VM's of containers snel herstellen op de andere server.

- Migratie en Onderhoud

Met twee servers kun je VM's of containers live migreren van de ene server naar de andere:

- o Live Migration: Verplaats VM's zonder downtime. Dit is handig voor onderhoudswerkzaamheden zoals hardware-upgrades of updates aan de Proxmox-software.
- o Je kunt een server uitschakelen voor onderhoud terwijl de andere server alle diensten blijft leveren.

- Redundantie

Het hebben van een tweede server zorgt voor redundantie, wat betekent dat je niet afhankelijk bent van één enkel systeem. Dit beschermt je tegen:

- o Hardwarestoringen: Zoals een defecte harde schijf, geheugen of moederbord.
- o Netwerkproblemen: Als één server niet meer bereikbaar is, blijven de VM's draaien op de andere server.

Het quorum-systeem in Proxmox zorgt ervoor dat alleen een meerderheid van de nodes in een cluster beslissingen mag nemen. Dit voorkomt dat gescheiden groepen (partitioned clusters) onafhankelijk van elkaar acties uitvoeren.

Daardoor dat je bijna altijd een oneven aantal servers opzet om een split-brain te voorkomen.

Split-brain is een probleem dat kan optreden in een clusteromgeving, zoals in Proxmox VE of andere gedistribueerde systemen, wanneer de communicatie tussen clusterleden wordt onderbroken. Hierdoor ontstaan er twee (of meer) onafhankelijke "groepen" binnen het cluster, die beide denken dat ze de enige actieve groep zijn. Dit leidt tot inconsistent data of conflicterende acties.

14 Proxmox Standalone Server

14.1 Inleiding

Een Proxmox standalone server is een Proxmox server die geen deel uitmaakt van een cluster.

Dit betekent dat de server zelfstandig werkt en niet gekoppeld is aan andere Proxmox-nodes. Standalone servers zijn ideaal voor kleinere omgevingen of situaties waarin clustering of high availability (HA) niet vereist is.

14.2 GUI vs CLI

14.2.1 GUI

De GUI van Proxmox is toegankelijk via een webbrowser en biedt een intuïtieve manier om virtualisatiebronnen te beheren. Het is ideaal voor gebruikers die de voorkeur geven aan een visuele benadering.

Kenmerken van de GUI:

- Overzichtelijke dashboard: Geeft statusinformatie over nodes, storage, netwerken, en resources.
- Beheer van virtuele machines en containers: Maken, starten, stoppen en configureren van VM's en containers.
- Snapshots en Back-ups: Eenvoudige toegang tot snapshotbeheer en back-upfuncties.
- Netwerkconfiguratie: Configuratie van bridges, VLAN's en netwerkinterfaces.
- Clusterbeheer: Beheer van meerdere Proxmox-nodes binnen een cluster.

U kunt de web gebaseerde beheerinterface gebruiken met elke moderne browser. Wanneer Proxmox VE detecteert dat u verbinding maakt vanaf een mobiel apparaat, wordt u doorgestuurd naar een eenvoudigere, op aanraking gebaseerde gebruikersinterface.

De webinterface is te bereiken via <https://uwipaddress:8006>

Standaardlogin is: root en het wachtwoord is gekozen tijdens het installatieproces.



Proxmox VE Login

User name: **root**

Password: *********

Realm: Linux PAM standard authentication

Language: English - English

Save User name: Login

14.2.2 CLI

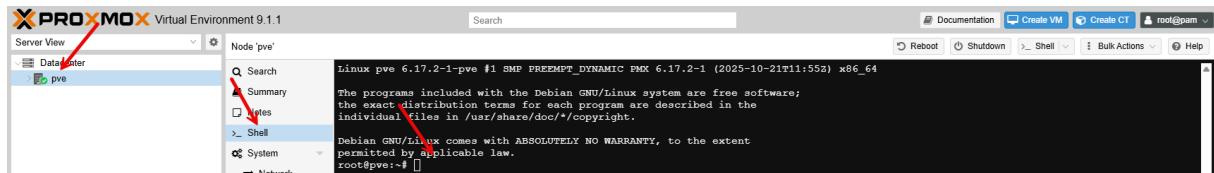
De CLI is toegankelijk via SSH of direct op de server en biedt meer flexibiliteit en controle.

Toegang gaat via ssh maar ook rechtstreeks vanuit GUI.

```
C:\Users\stijn>ssh root@192.168.112.133
root@192.168.112.133's password:
Linux pve 6.17.2-1-pve #1 SMP PREEMPT_DYNAMIC PMX 6.17.2-1 (2025-10-21T11:55Z) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Nov 20 10:59:55 2025 from 192.168.112.1
root@pve:~# |
```



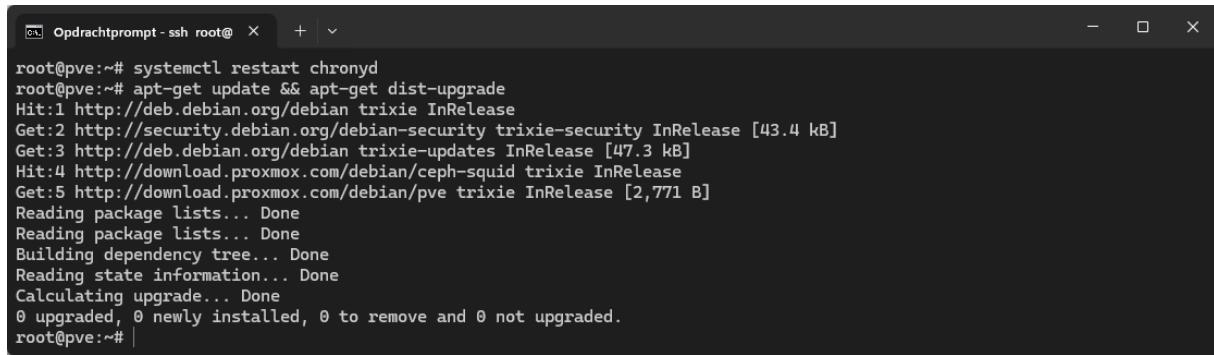
14.3 Belangrijke instellingen

14.3.1 Systeem software-updates

Proxmox biedt regelmatig updates voor alle repositories. Om updates te installeren gebruikt u de webgebaseerde GUI of de volgende CLI-opdrachten. Opgelet: de tijd moet correct staan hiervoor!

```
root@pve:~# systemctl restart chronyd
```

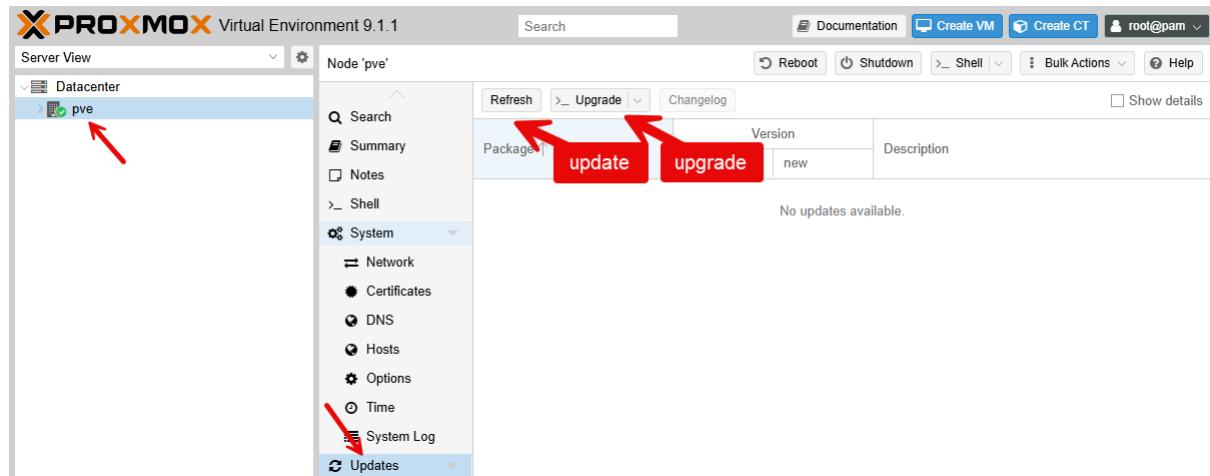
```
root@pve:~# apt-get update && apt-get dist-upgrade
```



```

root@pve:~# systemctl restart chronyd
root@pve:~# apt-get update && apt-get dist-upgrade
Hit:1 http://deb.debian.org/debian trixie InRelease
Get:2 http://security.debian.org/debian-security trixie InRelease [43.4 kB]
Get:3 http://deb.debian.org/debian-trixie-updates InRelease [47.3 kB]
Hit:4 http://download.proxmox.com/debian/ceph-squid trixie InRelease
Get:5 http://download.proxmox.com/debian/pve trixie InRelease [2,771 B]
Reading package lists... Done
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
root@pve:~#

```



14.3.2 Netwerkconfiguratie

14.3.2.1 Inleiding

De configuratie kan worden uitgevoerd via de GUI of door handmatig het bestand /etc/network/interfaces te bewerken, dat de volledige netwerkconfiguratie bevat. Het gebruik van de GUI is nog steeds de voorkeur hiervoor, omdat het u beschermt tegen fouten.

Een Linux bridge interface (vaak vmbrX genoemd) is nodig om gasten te verbinden met het onderliggende fysieke netwerk. Het kan worden gezien als een virtuele switch waarmee de gasten en fysieke interfaces zijn verbonden. Deze sectie biedt enkele voorbeelden van hoe het netwerk kan worden ingesteld.

14.3.2.2 Standaardconfiguratie met behulp van een brug

Via de CLI vind je onderstaande instellingen terug.

`nano /etc/network/interfaces`

```

GNU nano 8.4                               /etc/network/interfaces
auto lo
iface lo inet loopback

iface nic0 inet manual

auto vmbr0
iface vmbr0 inet static
    address 192.168.112.133/24
    gateway 192.168.112.2
    bridge-ports nic0
    bridge-stp off
    bridge-fd 0

source /etc/network/interfaces.d/*

```

^G Help ^O Write Out ^F Where Is ^K Cut ^T Execute ^C Location M-U Undo ^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^A Go To Line M-E Redo M-A Set Mark M-6 Copy

In de GUI zie je onderstaande.

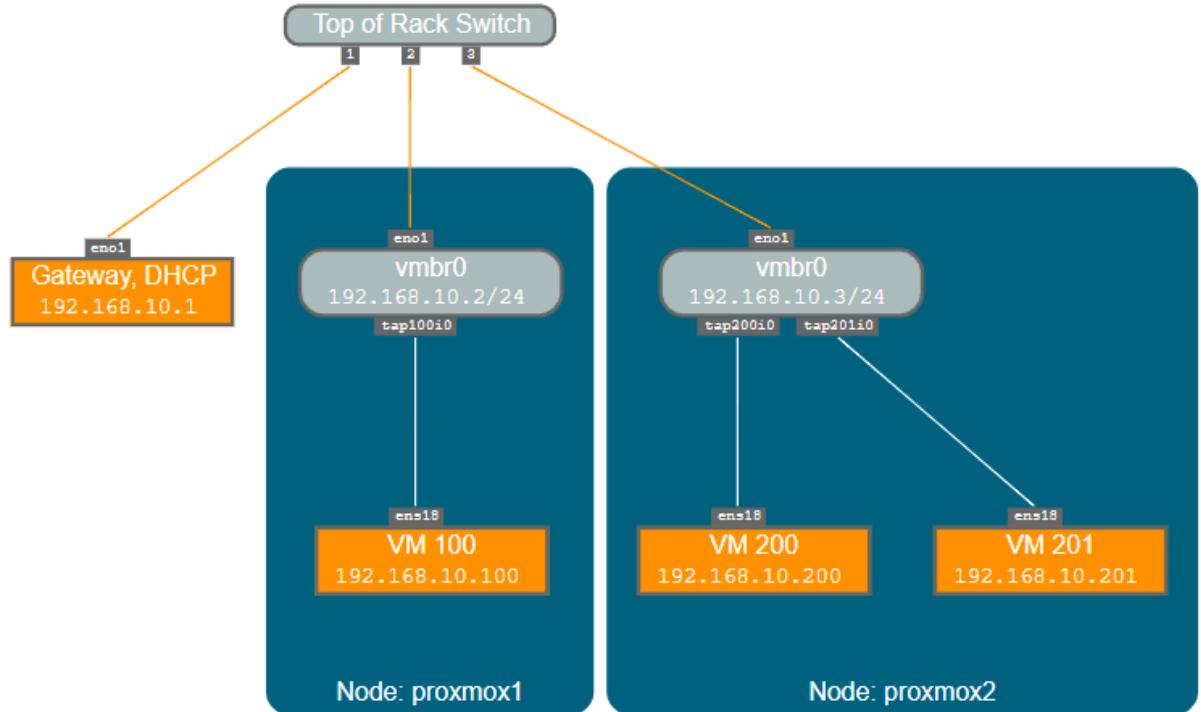
Name	Alternative Names	Type	Active	Autostart	VLAN a...	Ports/Slaves	Bond Mode	CIDR	Gateway
nic0	enp2s1 enx00c2914f161	Network Device	Yes	No	No				
vmbr0		Linux Bridge	Yes	Yes	No	nic0		192.168.112.133/24	192.168.112.2

Bridges zijn als fysieke netwerkswitches die in software zijn geïmplementeerd. Alle virtuele gasten kunnen één enkele bridge delen.

Het installatieprogramma maakt een enkele brug met de naam vmbr0 , die is verbonden met de eerste Ethernet-kaart.

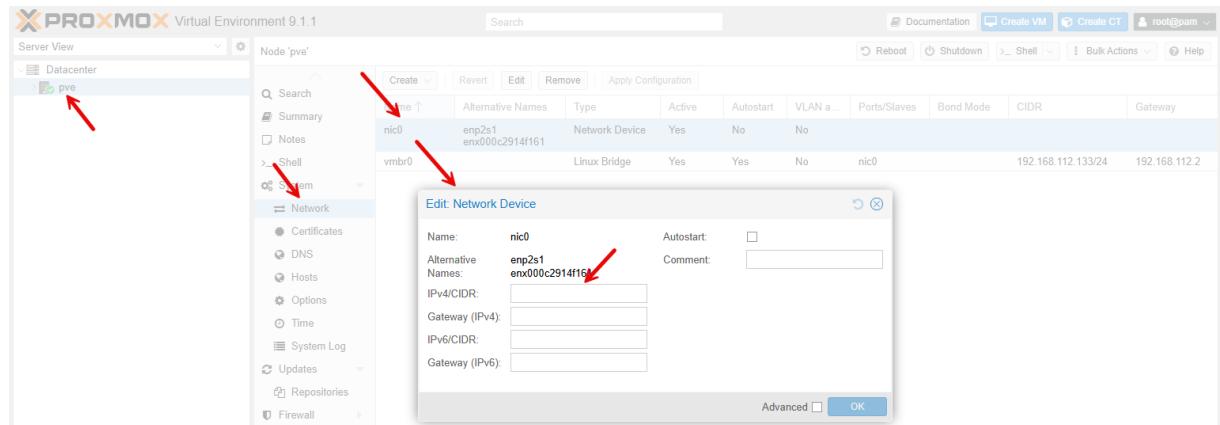
Virtuele machines gedragen zich alsof ze direct verbonden zijn met het fysieke netwerk. Het netwerk ziet op zijn beurt elke virtuele machine alsof deze zijn eigen MAC heeft, ook al is er maar één netwerkkabel die al deze VM's met het netwerk verbindt.

Hieronder zie je een voorbeeld met 2 nodes in een cluster. De VM's van de 2 nodes kunnen met elkaar communiceren.

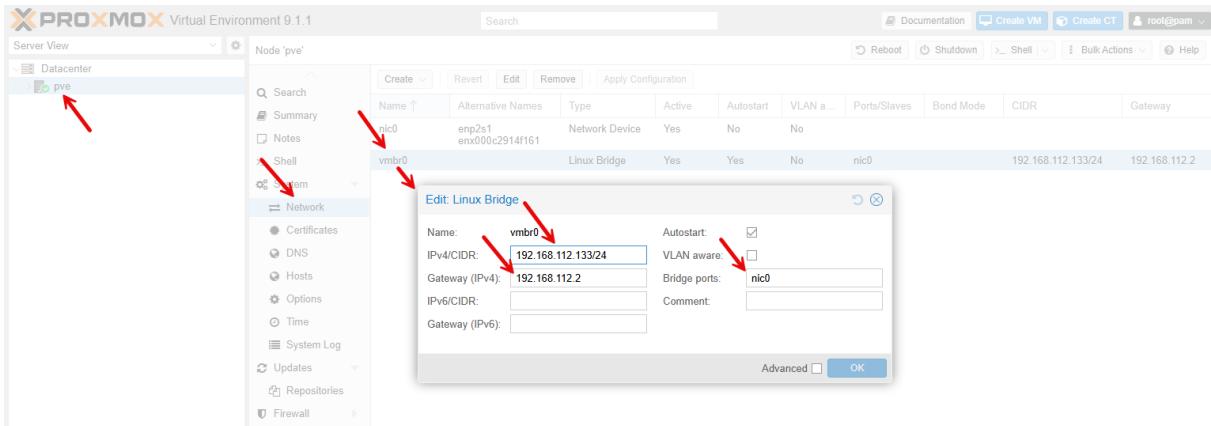


In ons geval krijgen de VM's IP-nummers in het subnet 192.168.112.0/24.

Als je naar de grafische omgeving gaat en je klikt op de netwerkkaart ens33 ga je zien dat deze geen IP-adres heeft.



Dit komt omdat je met een virtual bridge werkt. Daar vind je het IP-adres naar de buitenwereld terug.



Je ziet ook de Gateway en dat er een brug wordt gemaakt naar de netwerkkaart (nic0).

VM's en containers laat je dan verbinding maken met vmbr0.

14.3.2.3 Masquerading (NAT)

De meeste hostingproviders ondersteunen bovenstaande setup niet.

Om veiligheidsredenen schakelen ze netwerken uit zodra ze meerdere MAC-adressen op één interface detecteren. VMware doet dit gelukkig niet. Als we servers draaien in VMware kunnen we dus gebruik maken van de standaardconfiguratie.

Je kan gebruik maken van masquerading als je hostingprovider de standaard setup niet ondersteunt. VM's en containers gebruiken dan het IP van de Proxmox-host om internet te bereiken, en de host stuurt de antwoorden terug naar hen.

We komen hier later op terug.

14.3.3 Tijdsynchronisatie

De Proxmox VE cluster stack zelf is sterk afhankelijk van het feit dat alle nodes een exact gesynchroniseerde tijd hebben. Sommige andere componenten, zoals Ceph, werken ook niet goed als de lokale tijd op alle nodes niet synchroon is.

Tijdsynchronisatie tussen knooppunten kan worden bereikt met behulp van het "Network Time Protocol" (NTP). Vanaf Proxmox VE 7 wordt chrony gebruikt als de standaard NTP-daemon.

Geef in /etc/chrony/chrony.conf aan welke servers chrony moet gebruiken :

```
root@pve:~# nano /etc/chrony/chrony.conf
```

```

GNU nano 8.4                               /etc/chrony/chrony.conf
# Welcome to the chrony configuration file. See chrony.conf(5) for more
# information about usable directives.

# Use Debian vendor zone.
pool 2.debian.pool.ntp.org iburst

# Use time sources from DHCP.
sourcedir /run/chrony-dhcp

```

^G Help ^O Write Out ^F Where Is ^K Cut ^T Execute ^C Location M-U Undo
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^/ Go To Line M-E Redo M-A Set Mark
M-6 Copy

De tijdsynchronisatie verloopt via een groep NTP (Network Time Protocol)-servers.

Je kan ook specifieke NTP-servers instellen (we stellen dat wel niet in):

```

server ntp1.example.com iburst
server ntp2.example.com iburst
server ntp3.example.com iburst

```

Als de tijd niet correct staat tracht je terug verbinding te maken met een tijdserver.

```
root@pve:~# systemctl restart chronyd
```

14.4 Overzicht node management

Documentatie is lokaal opgeslagen, waardoor internettoegang niet vereist is.

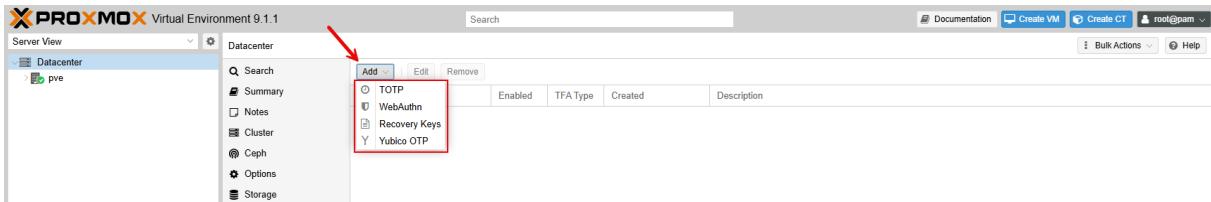
Type	Description	Disk usage...	Memory us...	CPU usage	Uptime	Host CPU ...	Host Mem...	Tags
zone	localnetwork (pve)	-	-	-	-	-	-	-
node	pve	6.5 %	41.7 %	0.2% of 12 ...	00:17:49	-	-	-
storage	local (pve)	6.5 %	-	-	-	-	-	-
storage	local-lvm (pve)	0.0 %	-	-	-	-	-	-

Je ziet ook ernaast 2 knoppen staan om een VM (virtuele machine) en een CT (container) aan te maken.

Twee-factor-authenticatie (2FA) kan worden geconfigureerd om de omgeving te beveiligen tegen ongeautoriseerde toegang.

Type	Description	Disk usage...	Memory us...	CPU usage	Uptime	Host CPU ...	Host Mem...	Tags
zone	localnetwork (pve)	-	-	-	-	-	-	-
node	pve	6.5 %	41.7 %	0.2% of 12 ...	00:19:39	-	-	-
storage	local (pve)	6.5 %	-	-	-	-	-	-
storage	local-lvm (pve)	0.0 %	-	-	-	-	-	-

Opties voor 2FA zijn onder meer webauthenticatie en herstelsleutels. Klik hiervoor op Add.



We kiezen bijvoorbeeld voor TOTP (Time-based One-Time Password). Het is een methode voor twee-factor-authenticatie waarbij een tijdelijke code wordt gegenereerd die slechts korte tijd geldig is. Je kan Google Authenticator gebruiken om de tweefactorauthenticatie te configureren.

Add a TOTP login factor

User:	<input type="text" value="root@pam"/>
Description:	<input type="text" value="Google authenticator"/>
Secret:	<input type="text" value="IXVDOH4VK4BFME27FEJG44TIQ6TESNGW"/> <button>Randomize</button>
Issuer Name:	<input type="text" value="Proxmox VE - pve"/>

Code van Google authenticator bijvoorbeeld

Verify Code:

[Help](#) [Add](#)



Zodra ik de QR-code scan in Google Authenticator, wordt deze toegevoegd.

User	Enabled	TFA Type	Created	Description
root@pam	Yes	totp	2025-11-22 12:32:18	Google authenticator

Als je terug inlogt met het wachtwoord van de gebruikers-ID zal men om de tweefactorauthenticatie vragen en moet je de code die op dat moment geldig is in Google authenticator voor deze gebruiker van Proxmox opgeven.

Second login factor required

WebAuthn TOTP App Recovery Key

Please enter your TOTP verification code: **224035**

Confirm Second Factor

Als u dat wilt uitschakelen, kunt u de tweefactorauthenticatie eenvoudig verwijderen en kunt u terug normaal inloggen.

The screenshot shows the Proxmox VE interface under the Datacenter section. In the left sidebar, 'Datacenter' is selected. In the main area, there is a table with columns: User, Enabled, TFA Type, Created, and Description. A row for 'root@pam' is selected and highlighted with a red box. The 'Enabled' column shows 'Yes', 'TFA Type' shows 'totp', 'Created' shows '2025-11-22 12:32:18', and 'Description' shows 'Google authenticator'. Above the table, there are buttons for 'Add', 'Edit', and 'Remove'. A red arrow points to the 'Remove' button.

Je kan ook herstelsleutels downloaden door te klikken op Recovery keys. Een herstelsleutel kan je maar één keer gebruiken! Als je alle herstelcodes van je 2FA hebt gebruikt, kun je nog steeds inloggen zolang je je 2FA-app (bijv. Google authenticator) hebt. Je krijgt geen nieuwe herstelcodes meer, tenzij je opnieuw inlogt en 2FA reset.

The screenshot shows the Proxmox VE interface under the Datacenter section. In the left sidebar, 'Datacenter' is selected. In the main area, there is a table with columns: Add, Edit, Remove, TOTP, WebAuthn, Recovery Keys, and Yubico OTP. The 'Recovery Keys' option is selected and highlighted with a red box. A red arrow points to this selection. Below the table, a 'Recovery Keys' dialog box is open, listing several recovery codes. A red arrow points to this dialog. To the right, a separate window titled 'Second login factor required' is shown, asking for a recovery key. A red arrow points to this window with the text 'Bij inloggen'.

Proxmox VE is georganiseerd in een boomstructuur met verschillende nodes binnen een datacenter. U kunt de details van CPU-, opslag- en geheugenverbruik in real-time bekijken van een node.

The screenshot shows the Proxmox VE 9.1.1 interface. The left sidebar lists nodes: Datacenter and pve. The 'pve' node is selected. The top navigation bar includes 'Documentation', 'Create VM', 'Create CT', 'root@pam', 'Reboot', 'Shutdown', 'Shell', 'Bulk Actions', and 'Help'. The main content area has tabs for 'Package versions', 'CPU Usage', and 'Server Load'. The 'CPU Usage' tab displays a graph from 2025-11-22 to 2025-11-22, showing CPU usage peaking around 0.5. The 'Server Load' tab shows load average spikes, with a major peak around 0.12 and several smaller peaks. A 'Tasks' section at the bottom shows a single task: 'Update package database' by 'root@pam' on Nov 22 12:30:07, status OK.

Netwerkconfiguratie in Proxmox VE omvat o.a. fysieke netwerkkaarten (bijvoorbeeld nic0) en virtuele bridges (vmbr0) zoals reeds besproken.

The screenshot shows the Proxmox VE 9.1.1 interface with the 'pve' node selected in the left sidebar. The top navigation bar is identical to the previous screenshot. The main content area shows a table of network interfaces. The table has columns: Name, Alternative Names, Type, Active, Autostart, VLAN a..., Ports/Slaves, Bond Mode, CIDR, Gateway, and Comment. It lists two entries: 'nic0' (Type: Network Device, Active: Yes, Autostart: No, VLAN a...: No, Ports/Slaves: nic0, CIDR: 192.168.112.133/24, Gateway: 192.168.112.2) and 'vmbr0' (Type: Linux Bridge, Active: Yes, Autostart: Yes, VLAN a...: No, Ports/Slaves: nic0, CIDR: 192.168.112.133/24, Gateway: 192.168.112.2).

Virtuele machines worden gekoppeld aan deze bridges om toegang te krijgen tot het netwerk. Het is belangrijk om IP-adressen toe te wijzen aan virtuele bridges in plaats van fysieke netwerkkaarten! Nieuwe bridges kunnen uiteraard worden toegevoegd. We komen hier later op terug.

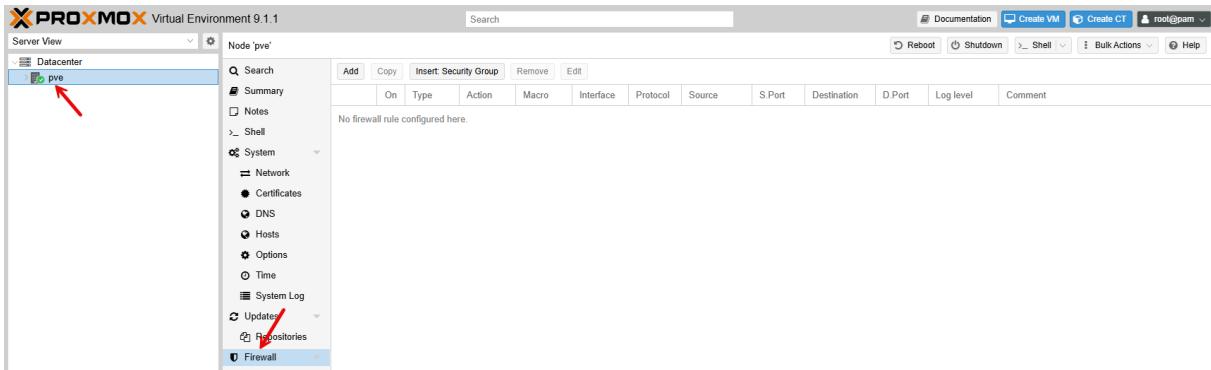
Opslag in Proxmox VE omvat fysieke schijven en logische volumes:

Device	Type	Usage	Size	GPT	Model	Serial	S.M.A.R.T	M.	Wearout
/dev/sda	unknown	partitions	214.75 GB	Yes	VMware_Virtual_S	unknown	OK	No	N/A
	partition	BIOS boot	1.03 MB	Yes				No	N/A
	partition	EFI	1.07 GB	Yes				No	N/A
	partition	LVM	213.67 GB	Yes				No	N/A

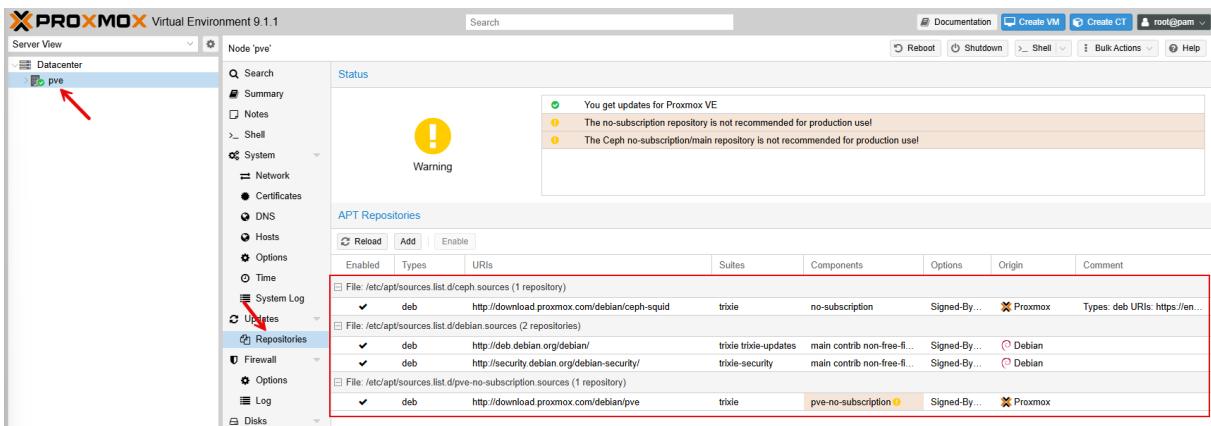
- Fysieke schijven: Schijven worden weergegeven als /dev/sdX. U kunt nieuwe schijven toevoegen, partitioneren of opnemen in een Logical Volume Manager (LVM).

Proxmox ondersteunt Ceph voor gedistribueerde opslag (high availability en replicatie). Daarop komen we later terug.

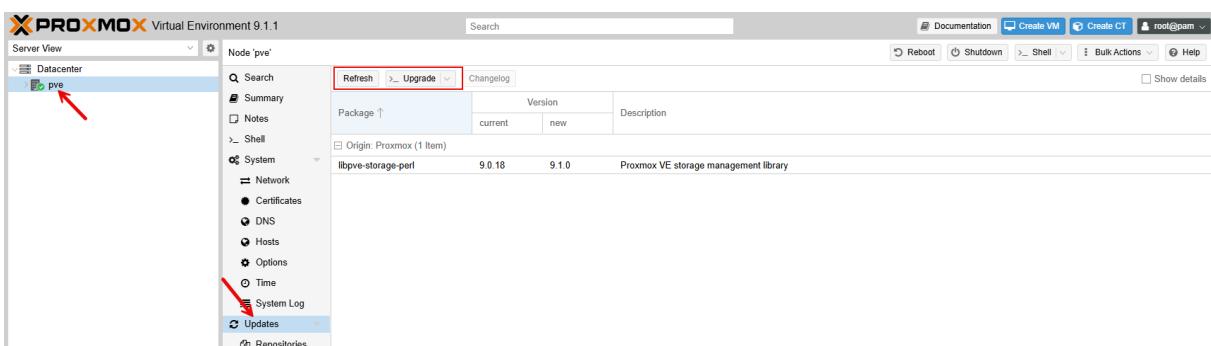
Proxmox heeft een ingebouwde firewall waarmee u regels kunt configureren om netwerktoegang te beperken. U kunt regels toepassen op het datacenter- of nodeniveau.



Updates worden beheerd via de apt-pakketbeheerder. Hoewel enterprise-repository's licenties worden aangeraden, kunnen andere repository's handmatig worden geconfigureerd zoals we reeds gedaan hebben.



Via de webinterface kunt u de status van updates controleren en uitvoeren. We hebben dit reeds besproken.



Standaard genereert Proxmox zelfondertekende TLS/HTTPS-certificaten waarmee de Proxmox-webinterface beveiligd is.

The screenshot shows the Proxmox VE 9.1.1 interface. In the left sidebar, under 'Datacenter', the 'pve' node is selected. In the main content area, the 'Certificates' tab is active. It displays a table of certificates:

File	Issuer	Subject	Valid Since	Expires	Subject Alternative Names
pve-root-ca.pem	/CN=Proxmox Virtual Environment/OU=...	/CN=Proxmox Virtual Environment/OU=...	2025-11-18 22:40:21	2035-11-16 22:40:21	127.0.0.1 0000:0000:0000:0000:0000:0000:0000:0000 localhost 192.168.112.133 pve pve.localdomain
pve-ssl.pem	/CN=Proxmox Virtual Environment/OU=...	/OU=PVE Cluster Node/0=Proxmox Virt...	2025-11-18 22:40:21	2027-11-18 22:40:21	127.0.0.1 0000:0000:0000:0000:0000:0000:0000:0000 localhost 192.168.112.133 pve pve.localdomain

Dit scherm is informatief, tenzij je geen SSL-waarschuwingen meer in je browser wil. Je kunt het pve-root-ca.pem certificaat toevoegen aan de vertrouwde root-certificaten van je besturingssysteem, zodat je browser geen waarschuwingen meer toont.

Eronder staat ACME. Een ACME-account is de registratie die Proxmox gebruikt binnen het ACME-protocol (Automatic Certificate Management Environment), een standaard voor het automatisch aanvragen en beheren van TLS/SSL-certificaten. Proxmox kan hiermee automatisch certificaten verkrijgen via bijvoorbeeld Let's Encrypt, een gratis certificaatautoriteit (let op: dit werkt alleen met een echte domeinnaam, niet met IP-adressen.). We gaan daar niet verder op in.

The screenshot shows the Proxmox VE 9.1.1 interface. In the left sidebar, under 'Datacenter', the 'pve' node is selected. In the main content area, the 'Certificates' tab is active. It displays a table of certificates and an 'ACME' button:

File	Issuer	Subject	Valid Since	Expires	Subject Alternative Names
pve-root-ca.pem	/CN=Proxmox Virtual Environment/OU=...	/CN=Proxmox Virtual Environment/OU=...	2025-11-18 22:40:21	2035-11-16 22:40:21	127.0.0.1 0000:0000:0000:0000:0000:0000:0000:0000 localhost 192.168.112.133 pve pve.localdomain
pve-ssl.pem	/CN=Proxmox Virtual Environment/OU=...	/OU=PVE Cluster Node/0=Proxmox Virt...	2025-11-18 22:40:21	2027-11-18 22:40:21	127.0.0.1 0000:0000:0000:0000:0000:0000:0000:0000 localhost 192.168.112.133 pve pve.localdomain

ACME

Add Edit Remove Order Certificates Now No Account available Add ACME Account

Domain ↑ Type Plugin

No Domains configured

Proxmox VE biedt een krachtige CLI waarmee beheerders taken kunnen uitvoeren zoals:

- Netwerkcontrole: Command ip addr toont netwerkinterfaces.
- Virtuele machines beheren: Command qm list toont een lijst van alle VM's.
- Updates: apt-get update en apt-get dist-upgrade voeren systeemupdates uit.

The screenshot shows the Proxmox VE 9.1.1 interface with a terminal window open. In the left sidebar, under 'Datacenter', the 'pve' node is selected. The terminal window displays the following output:

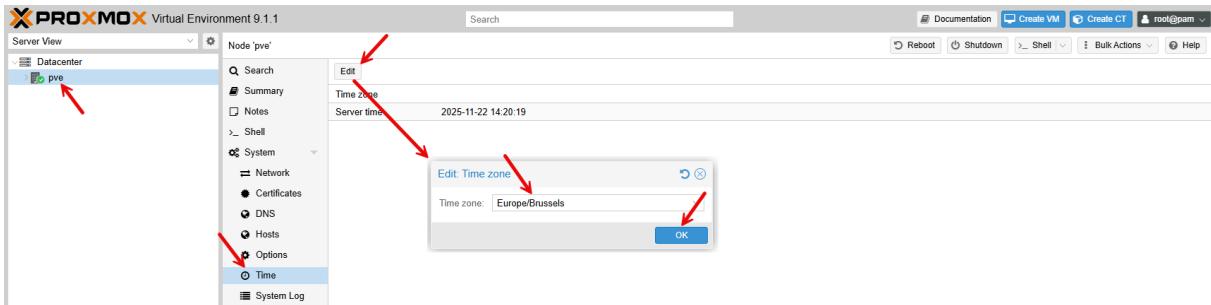
```

Linux pve 6.17.2-1-pve #1 SMP PREEMPT_DYNAMIC PMX 6.17.2-1 (2025-10-21T11:55:0) x86_64
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@pve-1:~# ip addr
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 0.0.0.0 scope host lo
        valid_lft forever preferred_lft forever
    2: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master vmbr0 state UP group default qlen 1000
        link/ether 00:0c:29:14:f1:61 brd ff:ff:ff:ff:ff:ff
        altname enp2s1
        altname enx000c2914f1c1
3: vmbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 00:0c:29:14:f1:61 brd ff:ff:ff:ff:ff:ff
    inet 192.168.112.1/24 brd 192.168.112.255 scope global vmbr0
        valid_lft forever preferred_lft forever
    4: kernel_ll: <NO-CARRIER,BROADCAST,MULTICAST,UP,LOWER_UP> mtu 64 scope link proto kernel_ll
        link/ether fe:00:20:c9:ff:fe brd ff:ff:ff:ff:ff:ff
        altname kernel_ll
        altname kernel-ll
    root@pve-1:~# apt-get update
Hit:1 http://deb.debian.org/debian trixie InRelease
Hit:2 http://security.debian.org/debian-security trixie-security InRelease
Hit:3 http://deb.debian.org/debian-trixie-updates InRelease
Hit:4 http://download.proxmox.com/debian/ceph-squid trixie InRelease
Hit:5 http://download.proxmox.com/debian/pve trixie InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
The following packages will be upgraded:
  liblibre-storage-perl
1 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Need to get 172 kB of archives.
After this operation, 2,048 B of additional disk space will be used.
Do you want to continue? [Y/n] 

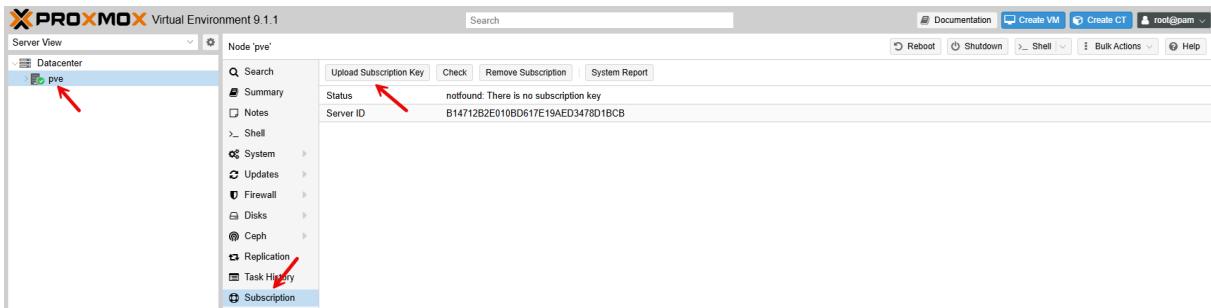
```

Tijdbeheer vind je hier terug.

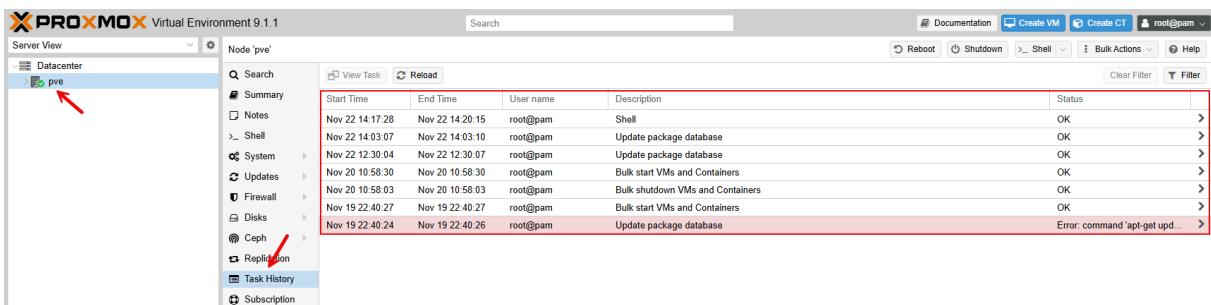


Je kan de tijdzone wijzigen door op Edit te klikken.

Bij Subscription heb je de mogelijkheid om licentiesleutels te uploaden en abonnementenstatussen te bekijken.



Je hebt ook nog Task History om logboeken te raadplegen. Kik op View/Output en View/Status om meer details te raadplegen.



14.5 Overzicht datacenter management

Op het datacenterniveau biedt Proxmox toegang tot instellingen en informatie van het cluster.

Het platform maakt het mogelijk om virtuele machines (VM's), containers en opslagapparaten binnen de pool te zoeken, beheren en configureren.

Hieronder wordt een beknopte uitleg gegeven van de belangrijkste functies en opties binnen de Proxmox Webinterface:

U kunt de status van het cluster bekijken en beheren via Cluster. Opties zoals opslag, back-ups, replicatie, toegangsrechten, High Availability, firewalls en systeemmeldingen zijn toegankelijk. Dit is nu uiteraard nog niet zichtbaar omdat we nog geen cluster hebben opgezet.

In clusterbeheer kunt u nieuwe clusters creëren en beheren.

Summary geeft een overzicht van de health van alle Proxmox-nodes:

- Aantal nodes: Online en offline status.
- Virtuele machines: Hoeveelheid actieve, inactieve en gestopte VM's.
- Containers (LXC): Actieve en gestopte containers.
- CPU-gebruik: Totaalgebruik van alle CPU's in het cluster.
- Geheugen en opslag: Gecombineerd gebruik van RAM en opslag over alle nodes.

Storage toont de gecombineerde opslag van alle nodes in het cluster.

ID	Type	Content	Path/Target	Shared	Enabled	Bandwidth Limit
local	Directory	Backup, Import, ISO image, Container template	/var/lib/vz	No	Yes	
local-lvm	LVM-Thin	Disk image, Container		No	Yes	

- Lokaal opslagbeheer:
 - o Lokale opslag, zoals /var/lib/vz, bevat o.a. geïmporteerde ISO's en templates.
 - o Disk images en containers worden beheerd in thin pools zoals LVM-thin. Bij LVM-thin wordt alleen ruimte ingenomen als ze daadwerkelijk worden gebruikt. Bij LVM niet.
- Toevoegen externe opslag: Ondersteuning voor NFS, SMB/CIFS, iSCSI, CephFS, LVM en andere systemen.

The screenshot shows the Proxmox VE 9.1.1 Datacenter interface. The left sidebar has 'Datacenter' selected under 'PVE'. The main panel shows a table of storage configurations. A red box highlights the 'Storage' section in the sidebar, and a red arrow points to it. Another red arrow points to the 'Add' button in the top right of the main panel. The table data is as follows:

Type	Content	Path/Target	Shared	Enabled	Bandwidth Limit
Directory	Backup, Import, ISO image, Container template	/var/lib/vz	No	Yes	
LVM-Thin	Disk image, Container		No	Yes	
BTRFS					
NFS					
SMB/CIFS					
iSCSI					
CephFS					
RBD					
ZFS over iSCSI					
ZFS					
Proxmox Backup Server					
ESXi					

Er is een mogelijkheid om back-ups van VM's en containers te maken.

The screenshot shows the Proxmox VE 9.1.1 Datacenter interface. The left sidebar has 'Datacenter' selected under 'PVE'. The main panel shows a table of backup configurations. A red box highlights the 'Backup' section in the sidebar, and a red arrow points to it. Another red arrow points to the 'Add' button in the top right of the main panel. The table data is as follows:

Enabled	Node	Schedule	Next Run	Storage	Comment	Retention	Selection
---------	------	----------	----------	---------	---------	-----------	-----------

Replicatie in Proxmox betekent dat een Virtuele Machine (VM) of container automatisch wordt gekopieerd naar een andere node binnen dezelfde Proxmox-cluster.

The screenshot shows the Proxmox VE 9.1.1 Datacenter interface. The left sidebar has 'Datacenter' selected under 'PVE'. The main panel shows a table of replication configurations. A red box highlights the 'Replication' section in the sidebar, and a red arrow points to it. Another red arrow points to the 'Add' button in the top right of the main panel. A tooltip at the bottom right says 'Replication needs at least two nodes'. The table data is as follows:

Enabled	Guest ↑	Job ↑	Target	Sched	Comment
---------	---------	-------	--------	-------	---------

Toegangsbeheer

- Gebruikers en groepen: Beheer gebruikersrechten en groepstoewijzingen.
- API Tokens: een digitale sleutel waarmee externe systemen of scripts veilig en zonder wachtwoord toegang krijgen tot de Proxmox API.

- Two Factor: Schakel tweefactorauthenticatie (2FA) in voor extra beveiliging.
- Groups: Groepeer gebruikers en wijs gezamenlijk permissies toe.
- Pools: Organiseer VM's/containers in logische groepen voor toegang.
- Roles: Definieer wat een gebruiker mag doen (bijv. VM beheren, opslag beheren).
- Realms: Configureer authenticatiemethoden zoals Active Directory Server of LDAP-server.

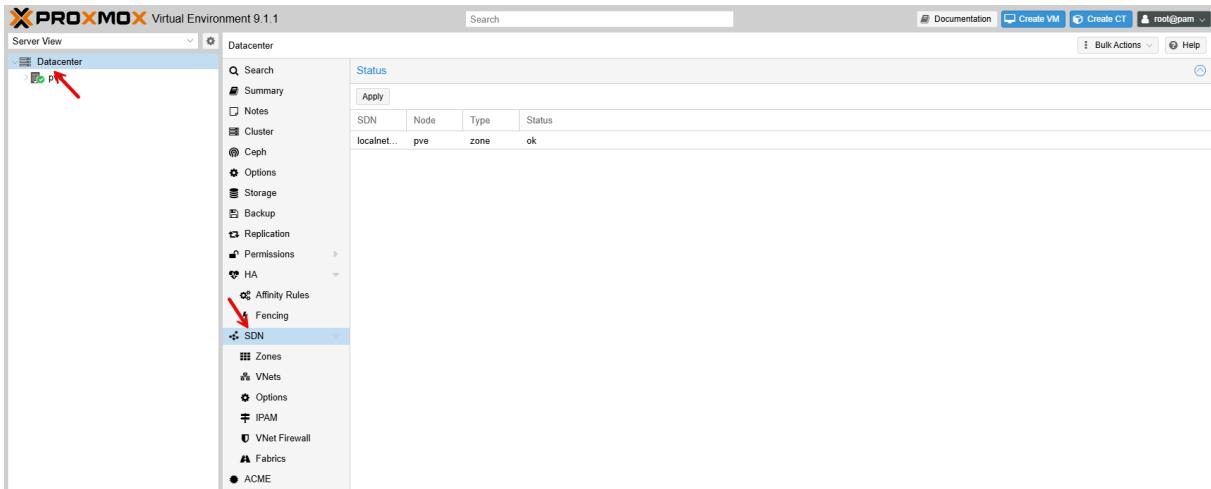
Hoge Beschikbaarheid (High Availability).

Proxmox zorgt ervoor dat containers en VM's automatisch worden verplaatst naar een andere node in het cluster als een node uitvalt. Dit vereist echter specifieke configuratie en de juiste setup van je cluster.

- Affinity rules in Proxmox zijn instellingen waarmee je bepaalt waar HA-resources (zoals VMs of containers) mogen draaien binnen een cluster, en of bepaalde resources altijd samen of juist gescheiden gehouden moeten worden.
- Fencing-opties zorgen voor het isoleren van defecte nodes in het cluster.

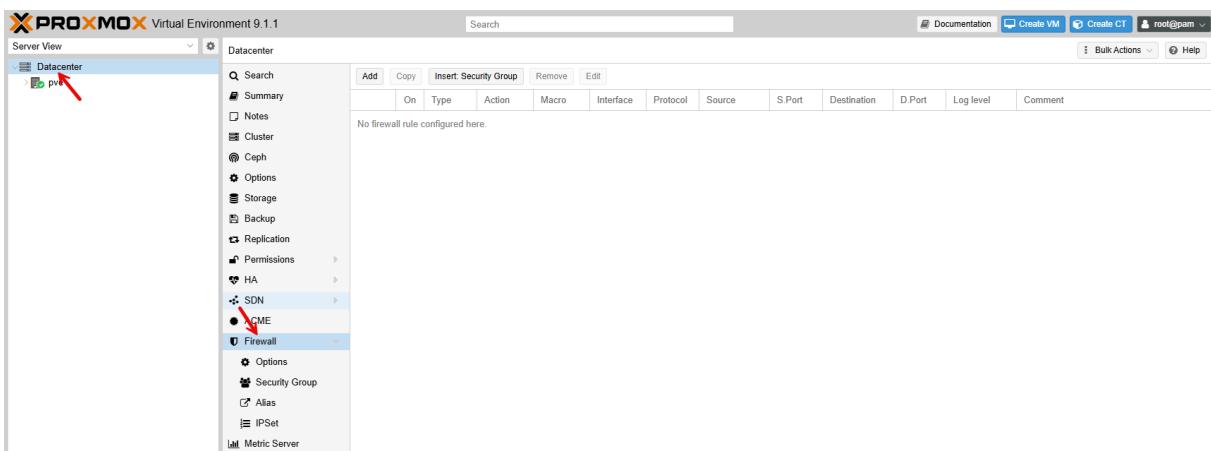
Software Defined Networking (SDN).

SDN maakt het mogelijk om één centraal punt te hebben voor netwerkbeheer, waardoor je VM's en containers altijd automatisch aan het juiste virtuele netwerk koppelt, inclusief firewall policies en automatische IP-toekenning (IPAM).



- Zones: Logische indeling van netwerken voor isolatie binnen het cluster.
- VNets: Virtuele netwerken binnen een SDN-zone voor VM's en containers.
- Options: Algemene SDN-instellingen zoals MTU-configuratie.
- IPAM: Beheer van IP-adressen en subnetten voor automatische toewijzing.
- VNet Firewall: Firewallregels om verkeer binnen VNets te beveiligen.

Je kan ook globale firewallregels instellen op datacenter niveau.



- Options: Instellingen zoals het in- of uitschakelen van de firewall.
- Security Group: Groepen van firewallregels die herbruikbaar zijn binnen verschillende VM's of containers.
- Alias: Vooraf gedefinieerde namen voor IP-adressen of netwerken om regels overzichtelijker te maken.
- IPSet: Collecties van IP-adressen en netwerken die in firewallregels gebruikt kunnen worden.

Eronder vind je ook nog:

- Metrics-server: voor statistieken en dashboards.
- Resource mapping: PCI- en USB-passthrough voor speciale hardware.
- Directory mappings: dit is een feature waarmee je directories op de Proxmox-host centraal kunt beschikbaar stellen aan virtuele machines (VM's) en containers.
- Notifications: ontvang meldingen over back-ups, systeemproblemen en andere gebeurtenissen.

- Support: voor gebruikers met een geldige licentie.

15 VM's en containers in Proxmox

15.1 Opslag

15.1.1 Inleiding

Er zijn in principe twee verschillende soorten opslagtypes:

- File level storage
Bestandsniveau-gebaseerde opslagtechnologieën bieden toegang tot een volledig uitgerust (POSIX) bestandssysteem. Deze technologieën zijn over het algemeen flexibeler dan opslag op blokniveau (zie hieronder) en maken het mogelijk om inhoud van elk type op te slaan. ZFS is waarschijnlijk het meest geavanceerde systeem en biedt volledige ondersteuning voor snapshots en klonen.
- Block level storage
Blokniveau-opslag maakt het mogelijk om grote ruwe schijfbeelden op te slaan. Het is meestal niet mogelijk om andere bestanden (zoals ISO-bestanden of back-ups) op dit type opslag te plaatsen.
De meeste moderne implementaties van block level storage ondersteunen snapshots en klonen.

Daarnaast heb je thin en thick provisioning:

- Bij thick provisioning wordt de volledige hoeveelheid toegewezen opslagruijte direct gereserveerd op het opslagmedium, ongeacht of deze ruimte daadwerkelijk wordt gebruikt.
- Bij thin provisioning wordt opslagruijte dynamisch toegewezen op basis van wat daadwerkelijk wordt gebruikt, in plaats van alles vooraf te reserveren.

Hieronder zie je een tabel van mogelijke opslagtypes in Proxmox.

Description	Shared	Level	Thick/Thin provisioning	Stable	Snapshots
Directory	no	file	beide: raw=thick, qcow2=thin	yes	no
BTRFS	no	file	thin (default), thick mogelijk	technology preview	yes
LVM	no	block	thick (default)	yes	no
LVM-thin	no	block	thin (default), thick handmatig	yes	yes
ZFS (local)	no	both	beide: afhankelijk van volume-optie	yes	yes

Description	Shared	Level	Thick/Thin provisioning	Stable	Snapshots
NFS	yes	file	beide: raw=thick, qcow2=thin	yes	no
CIFS	yes	file	beide: raw=thick, qcow2=thin	yes	no
GlusterFS	yes	file	beide: raw=thick, qcow2=thin	yes	no
CephFS	yes	file	beide: raw=thick, qcow2=thin	yes	yes
iSCSI/kernel	yes	block	thick (default), thin mogelijk met LVM-thin op target	yes	no
iSCSI/libiscsi	yes	block	thick (default), thin mogelijk met LVM-thin op target	yes	no
Ceph/RBD	yes	block	beide: standaard thin, thick mogelijk	yes	yes
ZFS over iSCSI	yes	block	beide: afhankelijk van ZFS-voloptie	yes	yes
Proxmox Backup	yes	both	n.v.t.	yes	n/a

De opslag kan verschillende content types ondersteunen, zoals virtual disk images, cdrom iso images, container templates of container root directories.. Niet alle opslagtypen ondersteunen alle content types.

We bekijken de schijfindeling.

```
root@pve:~# fdisk /dev/sda
```

```

Opdrachtprompt - ssh root@ ~ + | x
Command (m for help): p
Disk /dev/sda: 200 GiB, 214748364800 bytes, 419430400 sectors
Disk model: VMware Virtual S
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: C0C8F16F-D317-4BDC-A227-5D5EA2685438

Device Start End Sectors Size Type
/dev/sda1 34 2047 2014 1007K BIOS boot
/dev/sda2 2048 2099199 2097152 1G EFI System
/dev/sda3 2099200 419430366 417331167 199G Linux LVM

Command (m for help): |

```

Je ziet 3 partities:

/dev/sda1: BIOS boot

Deze kleine partitie wordt gebruikt door het GRUB bootloader-systeem wanneer de server in legacy BIOS-modus (in plaats van UEFI) wordt opgestart.

/dev/sda2: EFI System

Dit is een UEFI-partitie die wordt gebruikt door het systeem om op te starten in UEFI-modus. Het bevat bootloaders, zoals GRUB, en andere bestanden die nodig zijn voor het opstarten. Een grootte van 1 GB is voldoende om deze bestanden op te slaan.

/dev/sda3: Linux LVM

Deze partitie is gereserveerd voor LVM, wat flexibiliteit biedt bij het beheren van opslag.

Binnen deze LVM-partitie maakt Proxmox logische volumes aan voor verschillende doeleinden, zoals:

root (/): Voor systeem- en configuratiebestanden.

Swap: Als virtueel geheugen wanneer RAM ontoereikend is.

Data Storage: Voor het opslaan van VM's, containers, en snapshots.

We kijken wat op /dev/sda3 staat.

root@pve:~# pvdisplay /dev/sda3

```

Opdrachtprompt - ssh root@ ~ + | x
root@pve:~# pvdisplay /dev/sda3
--- Physical volume ---
PV Name           /dev/sda3
VG Name           pve
PV Size           <199.00 GiB / not usable 2.98 MiB
Allocatable       yes
PE Size           4.00 MiB
Total PE          50943
Free PE           4097
Allocated PE      46846
PV UUID           UYJti5-SDU2-ecU0-dJ5G-CehC-CHNw-euh026

root@pve:~#

```

Je ziet dat dit fysieke volume tot de volumegroep pve behoort.

We vragen nu meer info op over de volumegroep pve.

```
root@pve:~# vgdisplay pve
```



```
Windows PowerShell -> + <

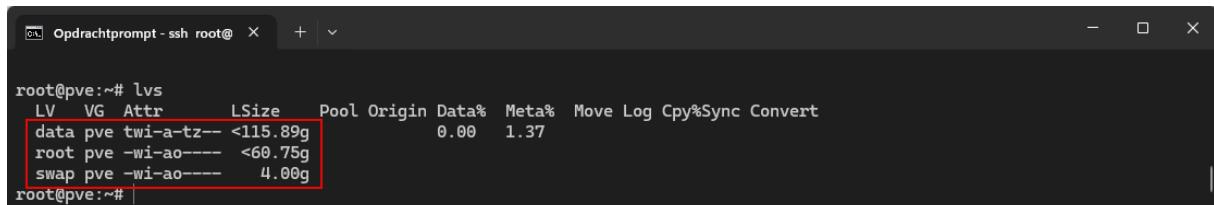
--- Volume group ---
VG Name          pve
System ID        lvm2
Format           lvm2
Metadata Areas   1
Metadata Sequence No 7
VG Access        read/write
VG Status        resizable
MAX LV           0
Cur LV           3
Open LV           2
Max PV           0
Cur PV           -
Act PV           -
VG Size          <199.00 GiB
PE Size          4.00 MiB
Total PE         50943
Alloc PE / Size  46846 / 182.99 GiB
Free  PE / Size  4097 / 16.00 GiB
VG UUID          2Zzb6m-SkMr-uJuT-Ho99-AltU-6PzS-Tptl7G

root@pve:~# |
```

Je ziet dat de volumegroep <199GiB groot is.

We vragen nu info op over het logisch volume.

```
root@pve:~# lvs
```



```
Opdrachtprompt - ssh root@ -> + <

root@pve:~# lvs
  LV   VG Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
  data pve twi-a-tz-- <115.89g            0.00   1.37
  root pve -wi-ao---- <60.75g
  swap pve -wi-ao----  4.00g
root@pve:~# |
```

- Data (< 115.89 GiB): Dit is de thin pool waarin opslag voor virtuele machines (VM's) en containers wordt bewaard.
- Root (< 60.75 GiB): Dit is het logische volume waarop het besturingssysteem van Proxmox draait.
- Swap (4.00 GiB): Dit is het logische volume dat als swapruimte (virtueel geheugen) wordt gebruikt.

We kunnen meer info opvragen over /.

```
root@pve:~# df -h | grep /dev/mapper/pve-root
```



```
Opdrachtprompt - ssh root@ -> + <

root@pve:~# df -h | grep /dev/mapper/pve-root
/dev/mapper/pve-root  60G  3.9G  53G  7% /
root@pve:~# |
```

De thin pool data wordt gebruikt voor VM-schijven en containeropslag. De thin pool wordt niet gemount omdat het geen bestandssysteem bevat zoals traditionele logische volumes.

```
root@pve:~# lvs
```

```
Opdrachtprompt - ssh root@ ~ + - X
root@pve:~# lvs
  LV   VG Attr       LSize   Pool Origin Data%  Meta% Move Log Cpy%Sync Convert
  data pve twi-a-tz-- <115.89g          0.00   1.37
  root pve -wi-ao---- <60.75g
  swap pve -wi-ao----  4.00g
root@pve:~#
```

Je ziet bij data als attr twi staan. Dat wil zeggen dat het een thin volume is.

15.1.2 Configuratie schijven en volumes

Alle opslagconfiguraties die gerelateerd zijn aan Proxmox VE worden opgeslagen in één enkel tekstbestand op /etc/pve/storage.cfg.

```
root@pve:~# nano /etc/pve/storage.cfg
```

```
Windows PowerShell - /etc/pve/storage.cfg
GNU nano 7.2
dir: local
path /var/lib/vz
content iso,vztmpl,backup

lvmthin: local-lvm
  thinpool data
  vgname pve
  content roottdir,images

^G Help      ^Q Write Out    ^W Where Is     ^K Cut        ^T Execute     ^C Location    M-U Undo
^X Exit      ^R Read File   ^\ Replace     ^U Paste      ^J Justify    ^A Go To Line  M-E Redo
                                         M-A Set Mark
                                         M-6 Copy
```

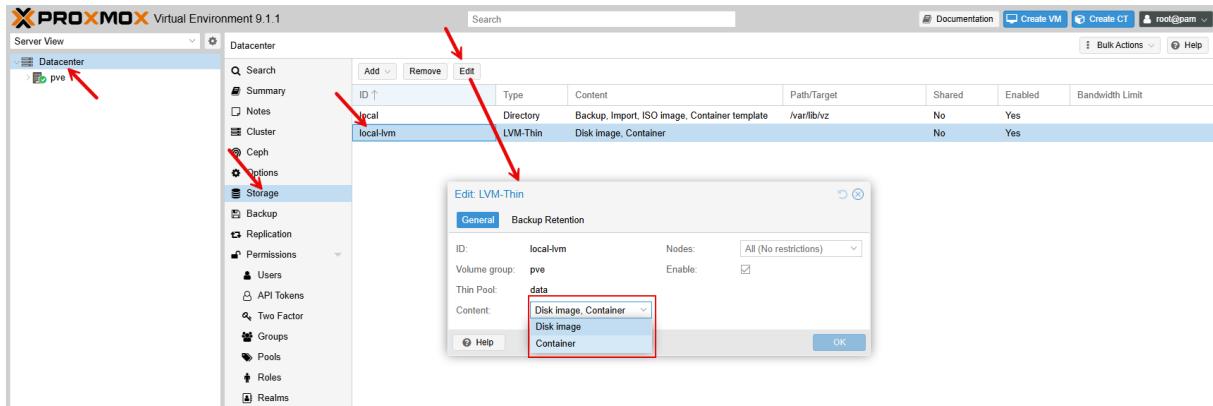
Dit bestand wordt automatisch gedistribueerd naar alle cluster-nodes (nu hebben we er wel nog maar één). Hierdoor delen alle nodes dezelfde opslagconfiguratie.

Grafisch vind je info over de opslag terug bij Datacenter, Storage.

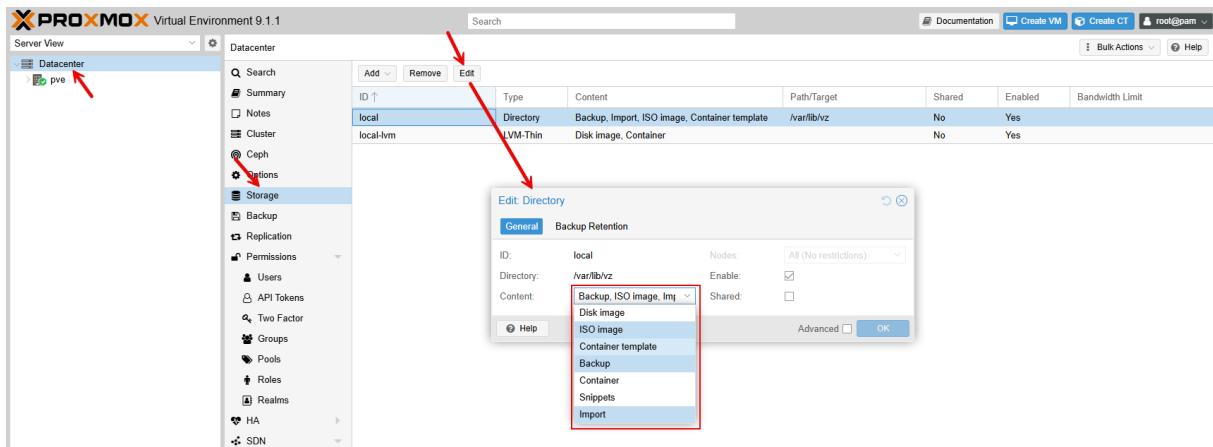
ID	Type	Content	Path/Target	Shared	Enabled	Bandwidth Limit
local	Directory	Backup, Import, ISO image, Container template	/var/lib/vz	No	Yes	
local-lvm	LVM-Thin	Disk image, Container		No	Yes	

Local is een Directory (/var/lib/vz) en local-lvm is LVM-Thin.

Als je local-lvm selecteert en klikt op Edit zal je zien dat je er enkel Disk image en container als content op kan zetten.

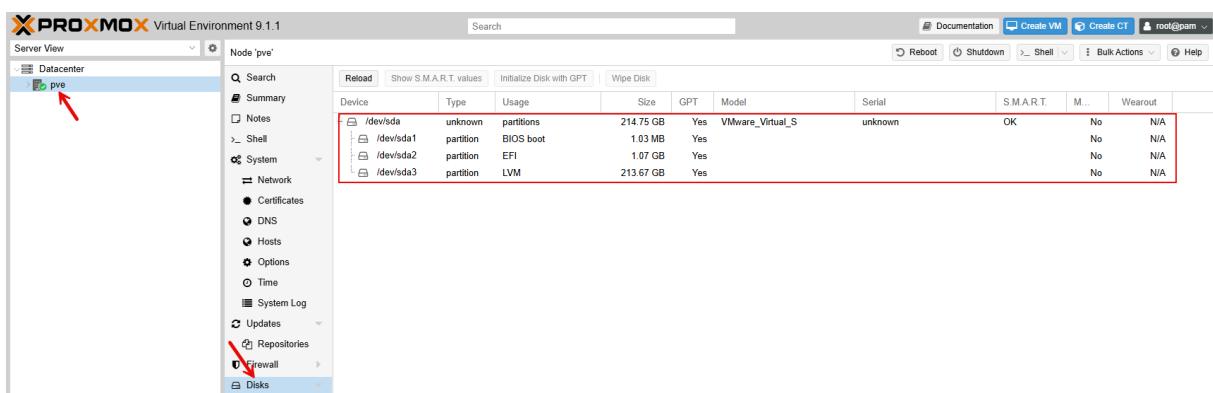


Als je local selecteert en op Edit klikt zal je zien dat je alle content erop kan zetten.



Je kan, door een item lichtblauw te maken door erop te klikken, instellen dat je dat type content wil kunnen opslaan. We passen dat nu niet aan.

Je vindt ook bij de node (pve), disks de info over de schijfindeling terug.



Als je eronder klikt op LVM zie je dat /dev/sda3 bijna volledig gebruikt wordt door de volumegroep pve.

The screenshot shows the Proxmox VE interface. The left sidebar has a tree view with 'Datacenter' expanded, showing 'pve'. Under 'pve', 'LVM' is selected. The main panel shows a table for 'Volume Group' with one entry: 'Name: pve', 'Number of LVs: 3', 'Assigned to LVs: 92%', 'Size: 213.67 GB', and 'Free: 17.18 GB'. Below the table, it says 'No volume group selected'.

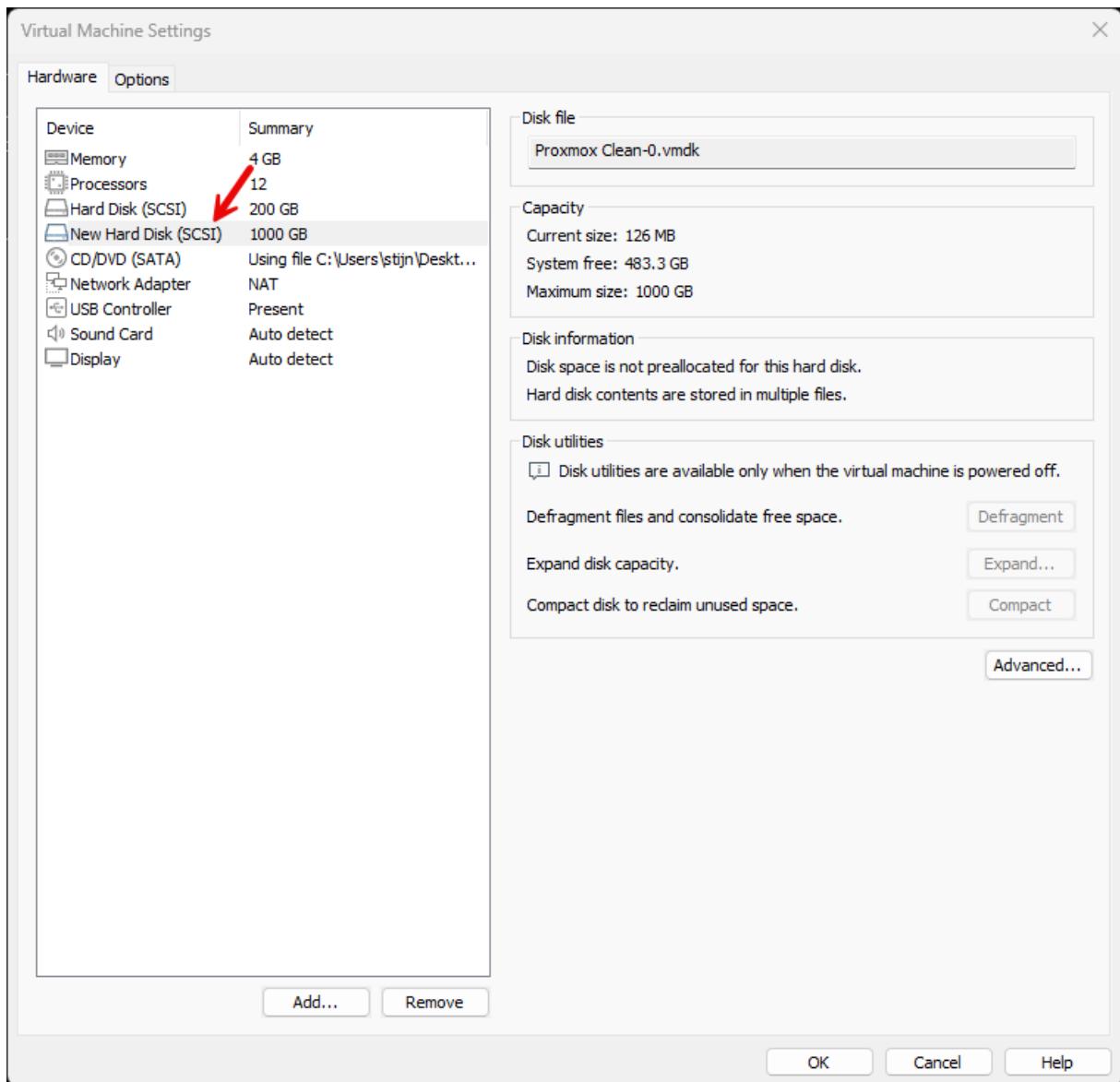
Bij LVM-Thin zie je de vrije ruimte zie je dat het logische volume data nog geen data bevat. Dat komt omdat we nog geen containers of VM's hebben aangemaakt.

The screenshot shows the Proxmox VE interface. The left sidebar has a tree view with 'Datacenter' expanded, showing 'pve'. Under 'pve', 'LVM-Thin' is selected. The main panel shows a table for 'Thinpool' with one entry: 'Name: data', 'Volume Group: pve', 'Usage: 0%', 'Size: 124.43 GB', 'Used: 0 B', 'Metadata Usage: 1%', 'Metadata Size: 1.27 GB', and 'Metadata Used: 17.35 MB'. Below the table, it says 'No thinpool selected'.

Als je klikt op Directory zie je dat dit leeg is. Je kan ook geen Directory toevoegen aangezien alle schijfruimte van /dev/sda bezet is.

The screenshot shows the Proxmox VE interface. The left sidebar has a tree view with 'Datacenter' expanded, showing 'pve'. Under 'pve', 'Directory' is selected. The main panel shows a 'Create: Directory' dialog box. It has fields for 'Disk:' (set to 'No Disks unused'), 'Filesystem:' (empty), 'Name:' (empty), and 'Add Storage:' (checkbox checked). There is a 'Create' button at the bottom right. Red arrows point from the tree view to the 'pve' node and the 'Directory' section, and another red arrow points to the 'Create: Directory' button.

We voegen een SCSI-schijf toe van 1000 GB toe aan pve.

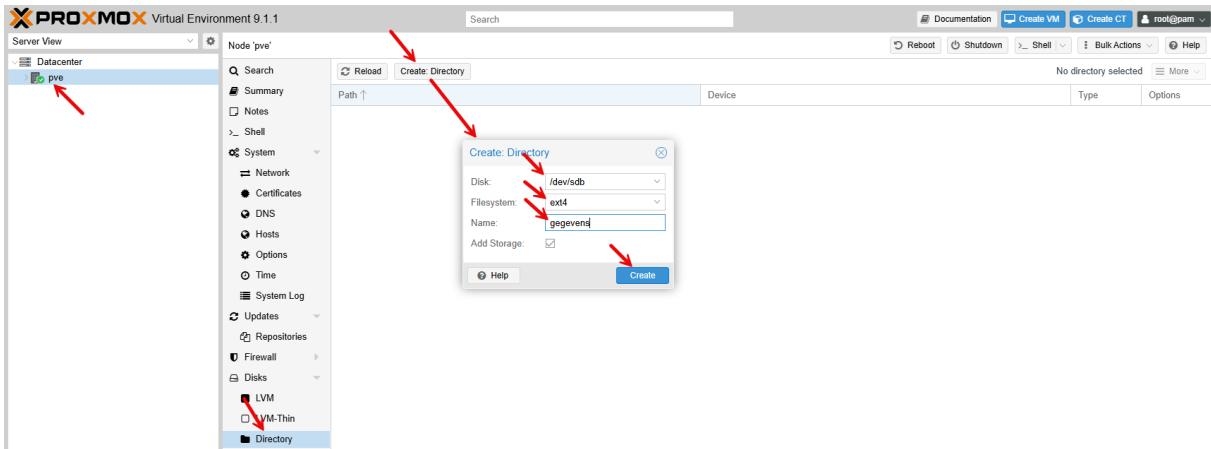


Je kan de schijf laten herkennen zonder te rebooten.

```
root@pve:~# ls /sys/class/scsi_host/ | while read host ; do echo
"--" > /sys/class/scsi_host/$host/scan ; done
```

```
Opdrachtprompt - ssh root@... + - X
root@pve:~# ls /sys/class/scsi_host/ | while read host ; do echo "--" > /sys/class/scsi_host/$host/scan ; done
root@pve:~#
```

Je kan nu een ext4 of xfs filesystem gaan aanmaken zoals hieronder staat weergegeven.



Er is nu een extra partitie aangemaakt voor het bestandsysteem.

```
root@pve:~# fdisk -l /dev/sdb

Disk /dev/sdb: 1000 GiB, 1073741824000 bytes, 2097152000 sectors

Disk model: VMware Virtual S

Units: sectors of 1 * 512 = 512 bytes

Sector size (logical/physical): 512 bytes / 512 bytes

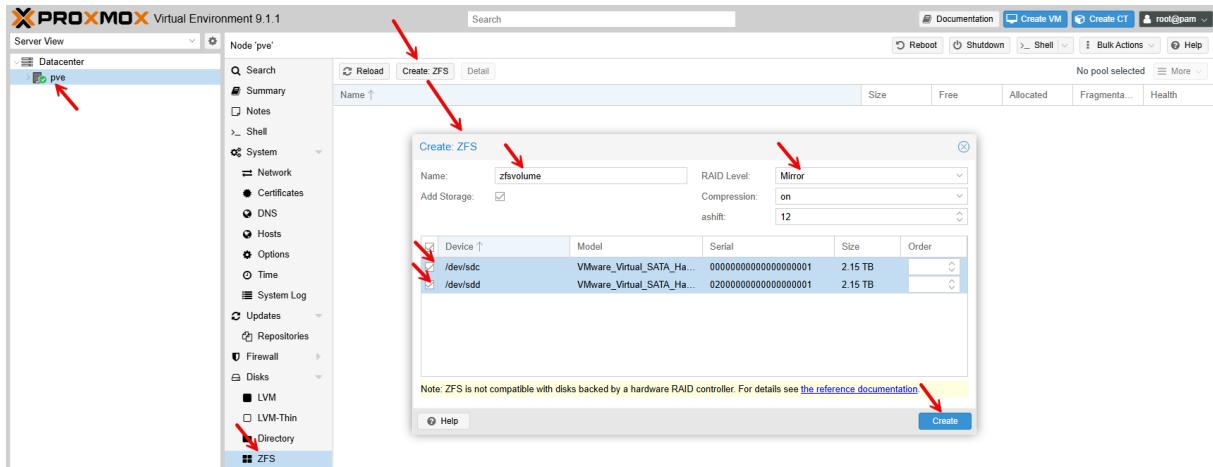
I/O size (minimum/optimal): 512 bytes / 512 bytes

Disklabel type: gpt

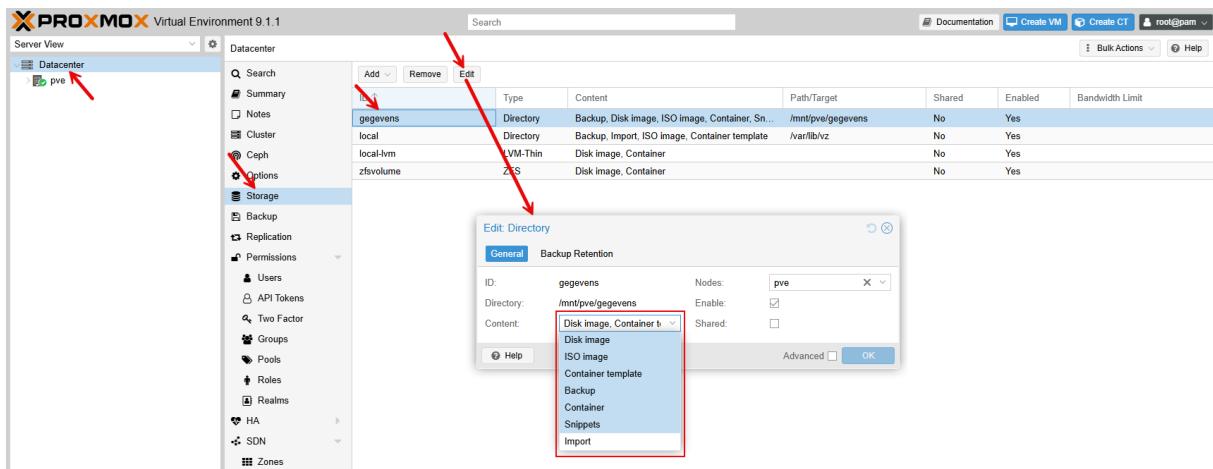
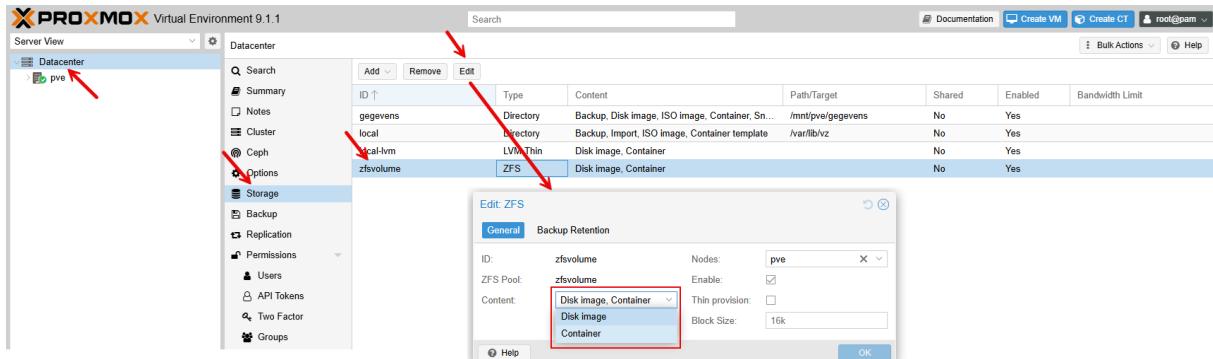
Disk identifier: 8F9960AD-3E2C-4BCB-84B2-613CE5195151
```

Device	Start	End	Sectors	Size	Type
/dev/sdb1	2048	2097151966	2097149919	1000G	Linux filesystem

Je kan ook ZFS instellen. In onderstaand voorbeeld heb ik 2 SATA-schijven van 2000 GB toegevoegd.



Je zal zien dat ZFS, met de huidige configuratie, enkel gebruik kan worden voor Disk images en containers en dat het xfs-volume gebruikt kan worden voor alle content.



We verwijderen nu de zelf toegevoegde Directory en ZFS.

The image consists of two screenshots of the Proxmox VE 9.1.1 web interface. Both screenshots show the 'Datacenter' view with the node 'pve' selected.

Screenshot 1: Directory Removal

- The left sidebar shows the navigation tree with 'Directory' selected under 'Disks'.
- The main content area shows a table with one row for a directory named '/mnt/pve/gegevens'. The table includes columns for Path, Device, Type, and Permissions.
- A red arrow points to the 'Path' column of the selected row.
- A red arrow points to the 'Destroy' button in the top right corner of the table header.
- A confirmation dialog box titled 'Confirm' is displayed, asking 'Directory gegevens - Remove' and 'Please enter the ID to confirm (gegevens)'. It contains two checkboxes: 'Cleanup Disks' and 'Cleanup Storage Configuration'. A red arrow points to the input field containing 'gegevens'.
- A red arrow points to the 'Remove' button at the bottom of the confirmation dialog.

Screenshot 2: ZFS Pool Removal

- The left sidebar shows the navigation tree with 'ZFS' selected under 'Disks'.
- The main content area shows a table with one row for a ZFS pool named 'zfsvolume'. The table includes columns for Size, Free, Allocated, Fragments, and Status (ONLINE).
- A red arrow points to the 'Name' column of the selected row.
- A red arrow points to the 'Destroy' button in the top right corner of the table header.
- A confirmation dialog box titled 'Confirm' is displayed, asking 'ZFS Pool zfsvolume - Remove' and 'Please enter the ID to confirm (zfsvolume)'. It contains two checkboxes: 'Cleanup Disks' and 'Cleanup Storage Configuration'. A red arrow points to the input field containing 'zfsvolume'.
- A red arrow points to the 'Remove' button at the bottom of the confirmation dialog.

15.2 VM

15.2.1 ISO toevoegen voor VM

15.2.1.1 Download de ISO

We downloaden Alpine Linux van <https://alpinelinux.org/downloads/> omdat Alpine Linux klein is.

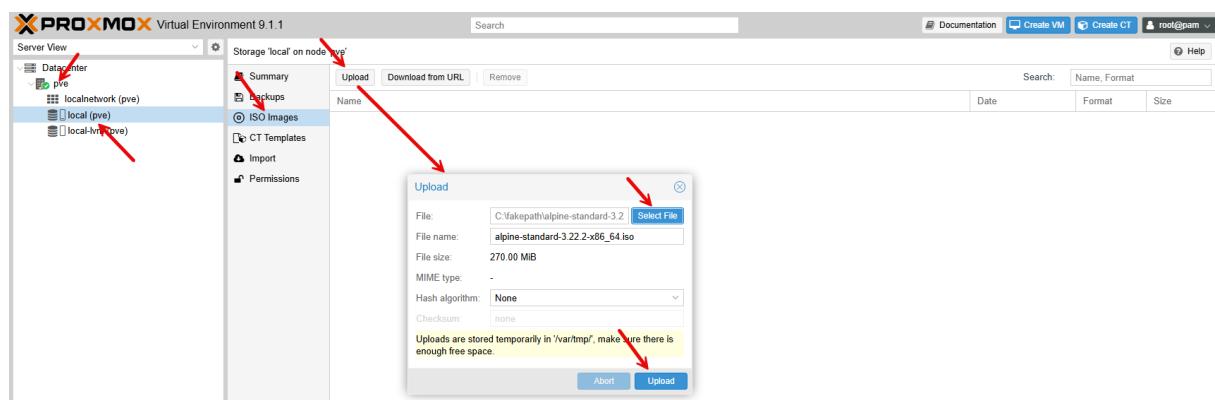
STANDARD	EXTENDED	NETBOOT
<p>Alpine as it was intended. Just enough to get you started. Network connection is required.</p> <p>aarch64 sha256 GPG armv7 sha256 GPG ppc64le sha256 GPG s390x sha256 GPG x86 sha256 GPG x86_64 sha256 GPG</p>	<p>Most common used packages included. Suitable for routers and servers. Runs from RAM. Includes AMD and Intel microcode updates.</p> <p>x86 sha256 GPG x86_64 sha256 GPG</p>	<p>Kernel, initramfs and modloop for netboot.</p> <p>aarch64 sha256 GPG armhf sha256 GPG armv7 sha256 GPG ppc64le sha256 GPG s390x sha256 GPG x86 sha256 GPG x86_64 sha256 GPG</p>

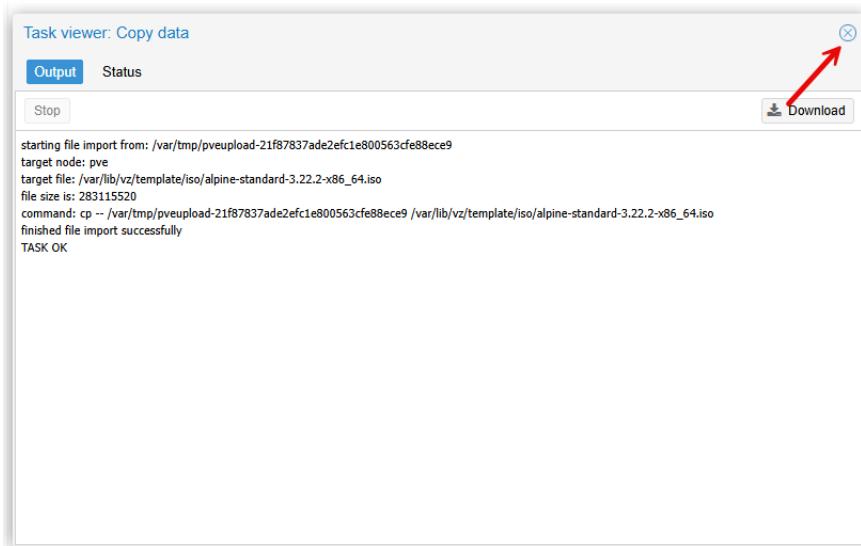
Sla de ISO op.

15.2.1.2 Upload de ISO naar Proxmox

We gaan nu naar local (pve) en kiezen voor ISO Images.

Klik op de knop "Upload": Klik op "Select File", selecteer de ISO en klik op Openen.





15.2.2 VM aanmaken

Klik op Create VM rechtsboven.

ID	Type	Content	Path/Target	Shared	Enabled	Bandwidth Limit
local	Directory	Backup, Import, ISO image, Container template	/var/lib/vz	No	Yes	
local-lvm	LVM-Thin	Disk image, Container		No	Yes	

- General

Een VM ID wordt automatisch toegewezen, maar u kunt ook een eigen ID invoeren (bijv. 555).

Geef een duidelijke naam, zoals Alpine.

Selecteer onderaan Advanced.

Voeg optioneel tags toe om het doel van de VM te identificeren. Bijvoorbeeld: LinuxOS.

- Meerdere tags kunnen worden toegevoegd of bestaande tags hergebruikt.

Create: Virtual Machine

General OS System Disks CPU Memory Network Confirm

Node: pve VM ID: 555 Name: Alpine

Add to HA:

Start at boot: Start/Shutdown order: any

Startup delay: default Shutdown timeout: default

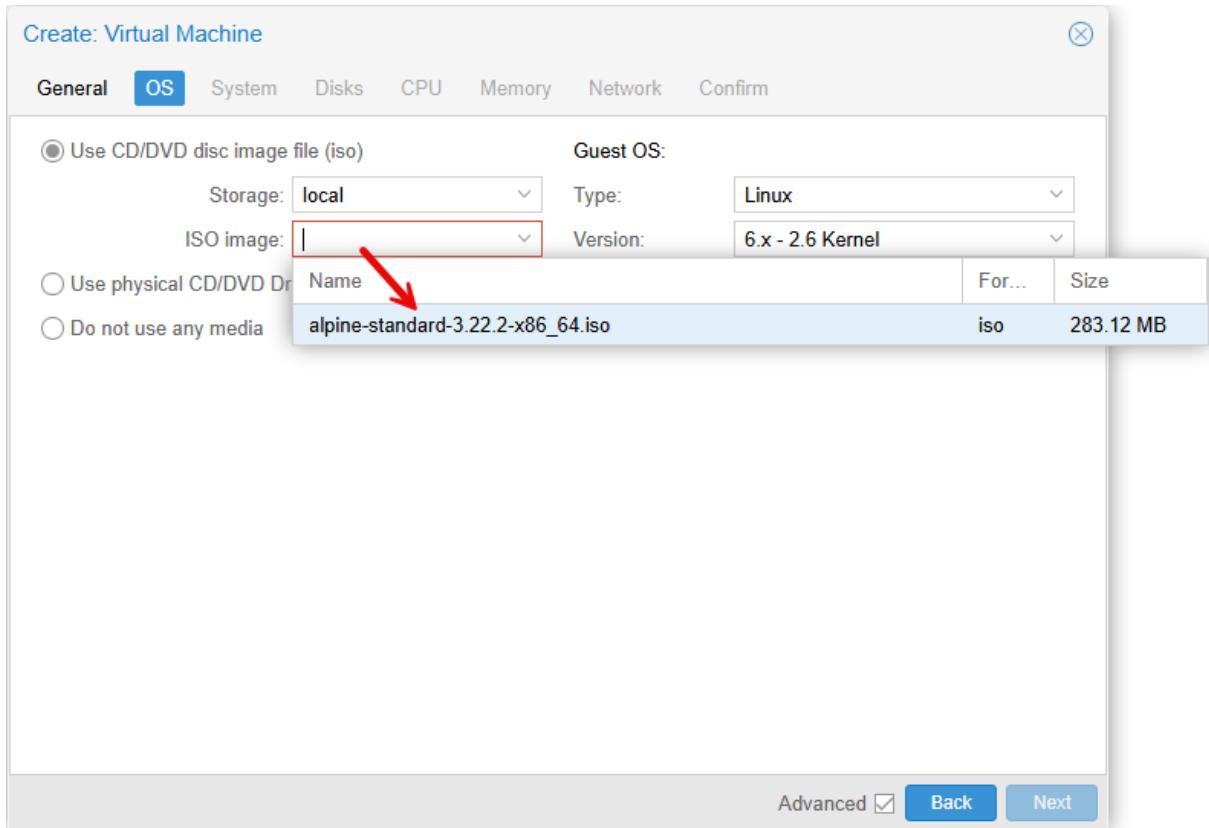
Tags: LinuxOS

Help Advanced Back Next

Het overige laten we staan en we klikken op Next.

- OS

Wij selecteren het ISO-image.



Bij versie kiezen we 6.x – 2.6 kernel.

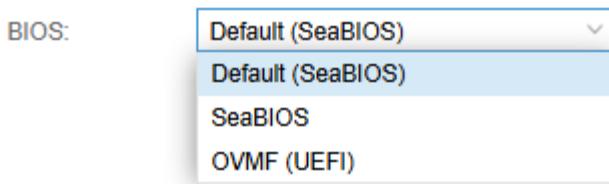
Klik erna op Next.

- System

Laat de standaardinstellingen voor SCSI-controllers en BIOS ongewijzigd.

Voor een geavanceerd BIOS kunt u UEFI selecteren, anders blijft het standaard BIOS ingesteld.

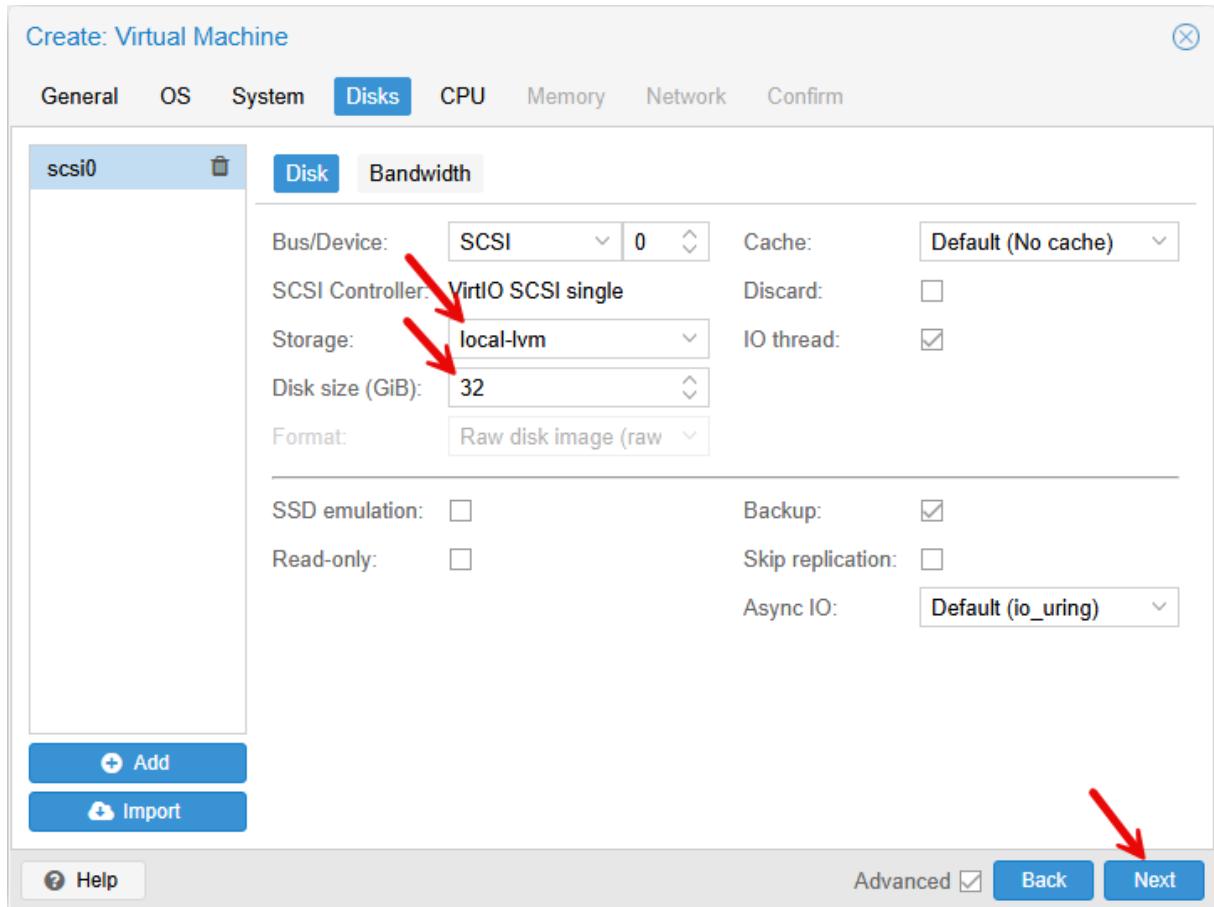
We klikken next.



- Disks

Kies de juiste opslaglocatie, hier Local LVM (enige mogelijkheid).

Stel de schijfruimte in (bijv. 32 GB).



Laat overige instellingen standaard en klik op Next.

- CPU

Kies het aantal cores voor de virtuele CPU. Standaard wordt één core toegewezen; pas dit aan naar behoefté (bijv. 2 cores). We laten dit nu staan.

Create: Virtual Machine

CPU

Sockets: 1 Type: x86-64-v2-AES

Cores: 1 Total cores: 1

VCPUs: 1 CPU units: 100

CPU limit: unlimited Enable NUMA:

CPU Affinity: All Cores

Extra CPU Flags:

Default	- ○○○ +	nested-virt	Controls nested virtualization, namely 'svm' for AMD CPUs and 'vmx' for Intel CPUs. Live migration still only works if it's the same flag on both sides. Use a CPU model similar to the host, with the same vendor, not x86-64-vX!
Default	- ○○○ +	md-clear	Required to let the guest OS know if MDS is mitigated correctly.
Default	- ○○○ +	pcid	Meltdown fix cost reduction on Westmere, Sandy-, and IvyBridge Intel CPUs.
Default	- ○○○ +	spec-ctrl	Allows improved Spectre mitigation with Intel CPUs.
Default	- ○○○ +	ssbd	Protection for 'Speculative Store Bypass' for Intel models.

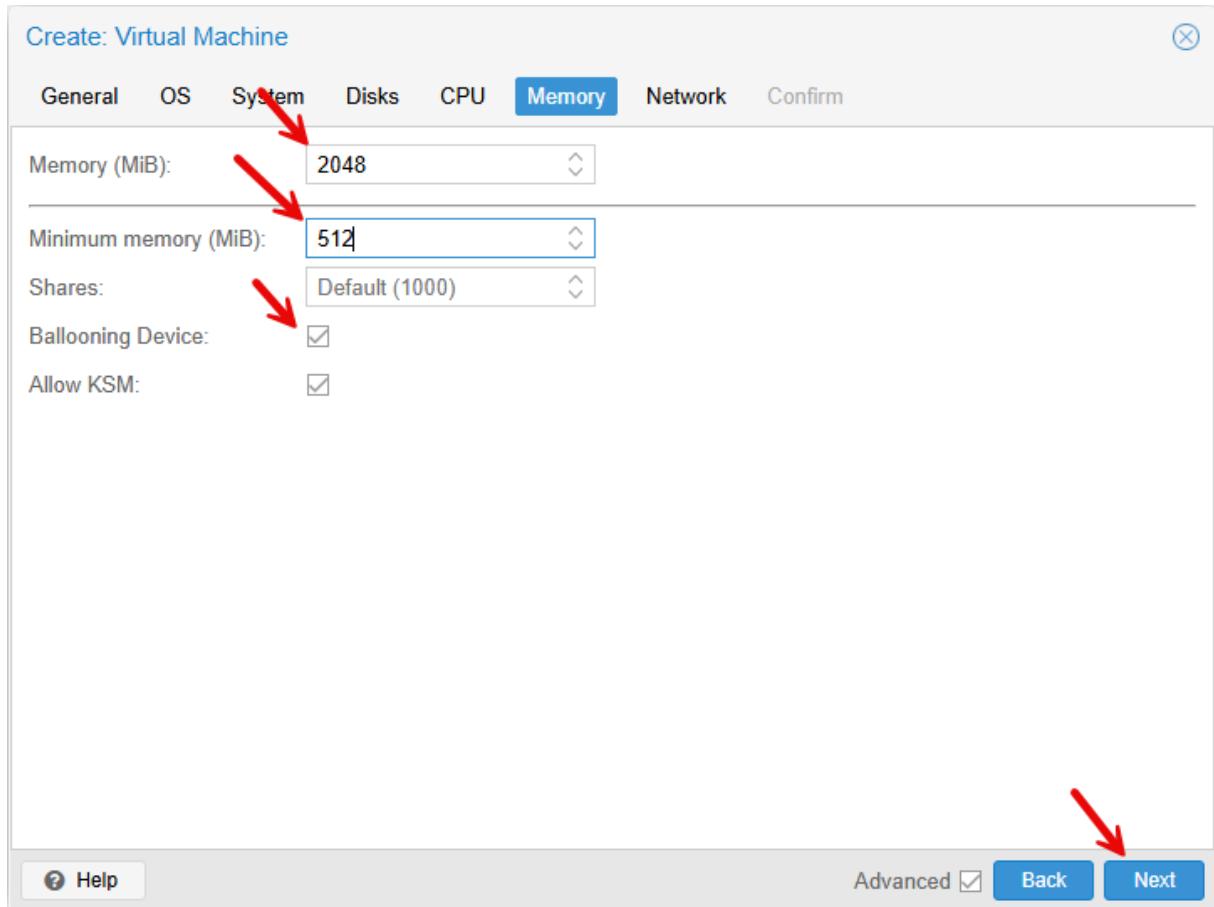
Help Advanced Back **Next**



- Memory

Wijs RAM toe aan de VM. Standaard is dit 2 GB (2048 MB).

De balloon driver wordt gebruikt om geheugen dynamisch aan een virtuele machine toe te wijzen of ervan terug te nemen, afhankelijk van de behoeften van het gast- en host-systeem. Dit proces wordt "memory ballooning" genoemd. Laat dit dus aan staan.

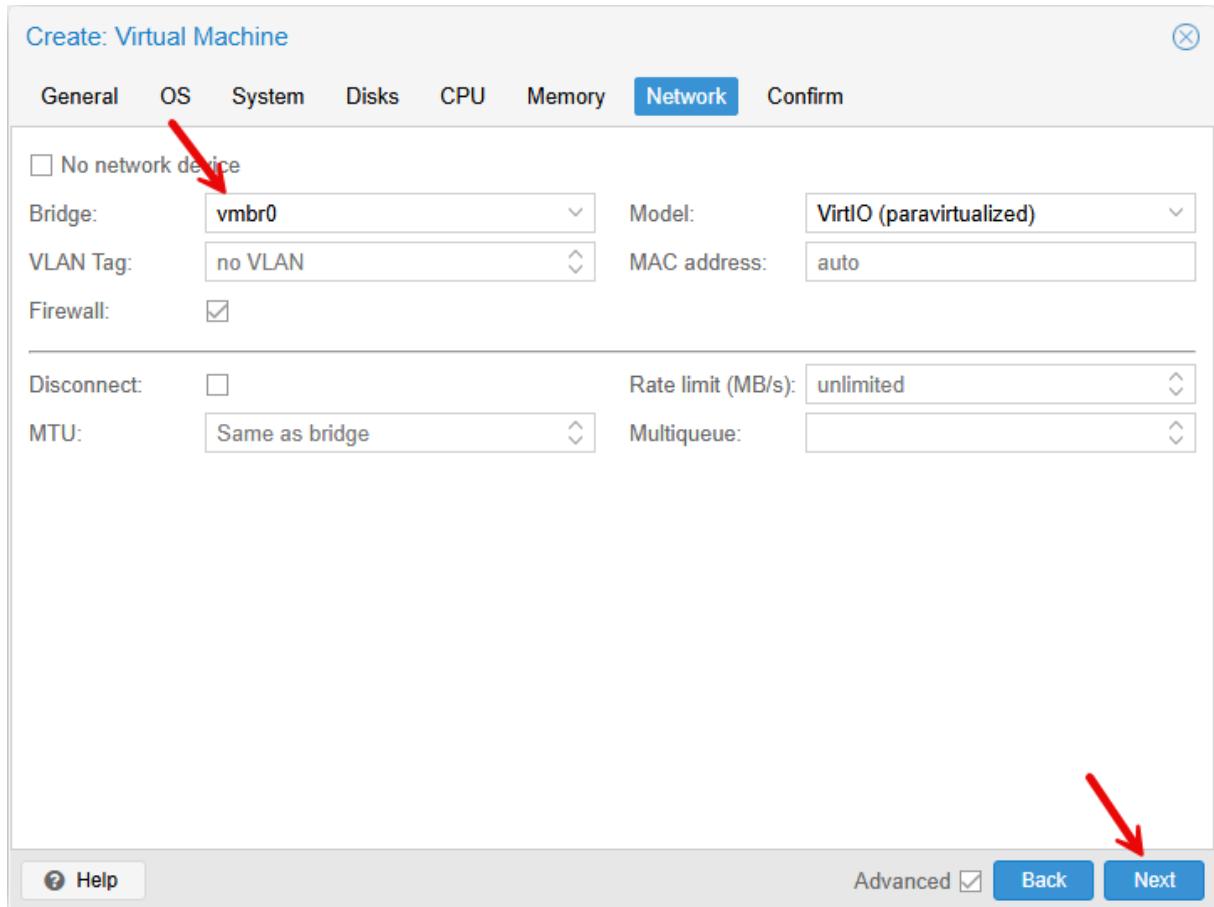


KSM staat voor Kernel Same-page Merging, een Linux-functie die geheugenoptimalisatie mogelijk maakt in virtuele machines.

We laten dit staan en klikken op Next.

- Netwerk

Selecteer de netwerkbrug (vmbr0).



We laten de overige instellingen staan en klikken op Next.

Kijk nu je instellingen na.

Create: Virtual Machine

General OS System Disks CPU Memory Network **Confirm**

Key ↑	Value
cores	1
cpu	x86-64-v2-AES
ide2	local:iso/alpine-standard-3.22.2-x86_64.iso,media=cdrom
memory	2048
name	Alpine
net0	virtio,bridge=vmbr0,firewall=1
nodename	pve
numa	0
ostype	l26
scsi0	local-lvm:32,iothread=on
scsихw	virtio-scsi-single
sockets	1
tags	LinuxOS
vmid	555

Start after created

Advanced **Back** **Finish** (arrow points to Finish button)

Klik op Finish.

De VM wordt aangemaakt en verschijnt in de lijst onder de server pve.

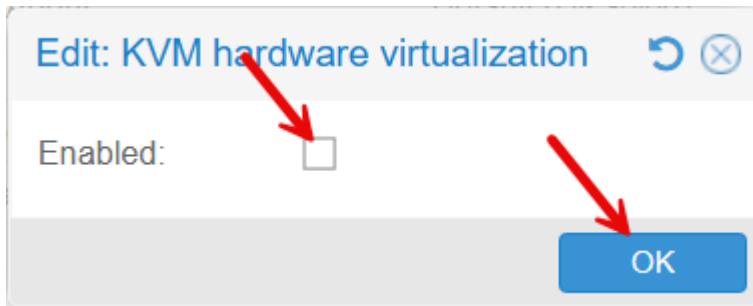
15.2.3 Nested virtualization

Indien je geen nested virtualization hebt pas je onderstaande instelling bij opties aan.

Name	Value
Name	Alpine
Start at boot	No
Start/Shutdown order	order=any
OS Type	Linux 6.x - 2.6 Kernel
Boot Order	scsi0, ide2, net0
Use tablet for pointer	Yes
Hotplug	Disk, Network, USB
ACPI support	Yes
KVM hardware virtualization	Yes
Freeze CPU at startup	No

Dubbelklik op KVM hardware virtualization.

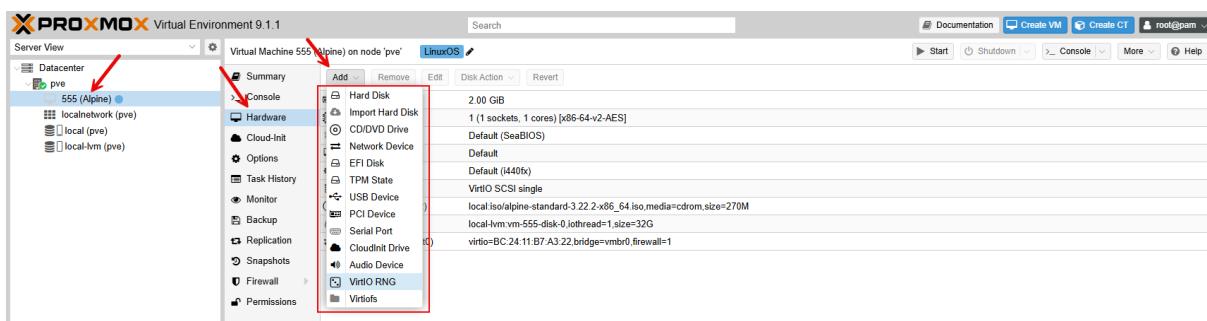
Verwijder nu het vinkje en klik uiteraard op OK.



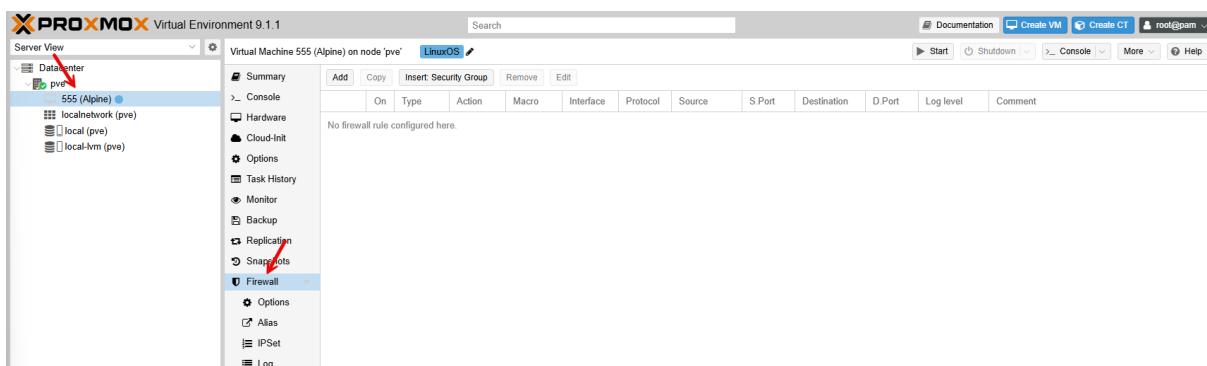
Je VM zal nu uiteraard trager draaien als je deze start t.o.v. zelfde VM met nested virtualization aan. Zonder KVM draait de VM in software-emulatie, wat trager en minder efficiënt is. Dit werkt enkel voor eenvoudige VM's.

15.2.4 Instellingen VM

Ga naar het tabblad Hardware om extra hardware toe te voegen (bijv. netwerkapparaten), hardware te wijzigen of bestaande hardware te verwijderen. We laten dat nu ongewijzigd.

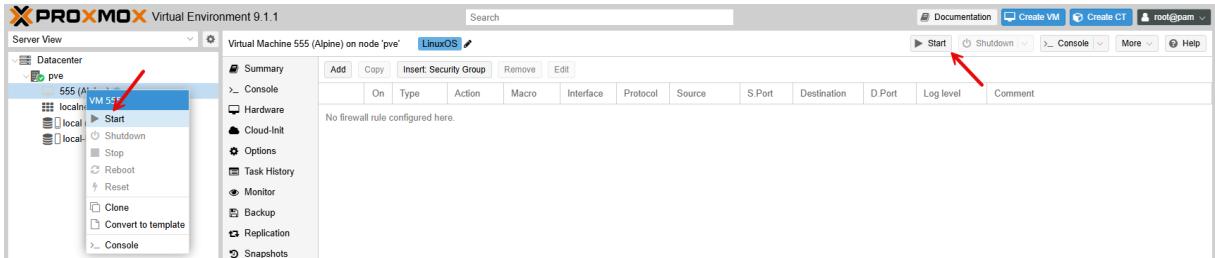


Configureer optioneel de ingebouwde firewall in Proxmox voor extra beveiliging. We laten dat nu ongewijzigd.

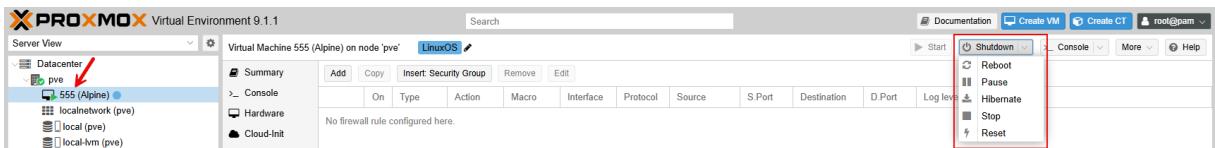


15.2.5 VM starten enz.

We starten de VM op door met de rechtermuisknop op de VM te klikken en te kiezen voor Start of door de VM te selecteren en Start te kiezen. De VM zal vanaf de opgegeven bron (bijv. een CD of ISO-image) booten.



Je kan op dezelfde plaats de VM stoppen enz.



15.2.6 Verbinding maken met VM

Klik op Console na het selecteren van de VM die je opgestart hebt.



Je kan inloggen als root zonder wachtwoord in te geven.

X QEMU (Alpine) - noVNC - Google Chrome

Niet beveiligd https://192.168.112.133:8006/?console=kvm&novnc=1&vmid=555&vmname=Alpine&node=pve&resize=off&cmd=

ISOLINUX 6.04 6.04-pre1 ETCD Copyright (C) 1994-2015 H. Peter Anvin et al

boot:

```
* /proc is already mounted
* Mounting /run ...
* /run/openrc: creating directory
* /run/lock: creating directory
* /run/lock: correcting owner
* Caching service dependencies ...
* Remounting devtmpfs on /dev ...
* Mounting /dev/queue ...
* Mounting modloop ...
* Verifying modloop
* Mounting security filesystem ...
* Mounting debug filesystem ...
* Mounting persistent storage (pstore) filesystem ...
* Mounting btrfs filesystem ...
* Starting busybox mddev ...
* Scanning hardware for mddev ...
* Loading hardware drivers ...
* Loading modules ...
* Setting system clock using the hardware clock [UTC] ...
* Checking local filesystems ...
* Remounting filesystems ...
* Mounting local filesystems ...
* Configuring kernel parameters ...
* Creating user login records ...
* Cleaning /tmp directory ...
* Setting hostname ...
* Starting busybox syslog ...
* Starting firstboot ...

Welcome to Alpine Linux 3.22
Kernel 6.12.51-0-its on x86_64 (/dev/tty1)

localhost login: root
Welcome to Alpine!
```

The Alpine Wiki contains a large amount of how-to guides and general information about administrating Alpine systems.
See <https://wiki.alpinelinux.org/>.

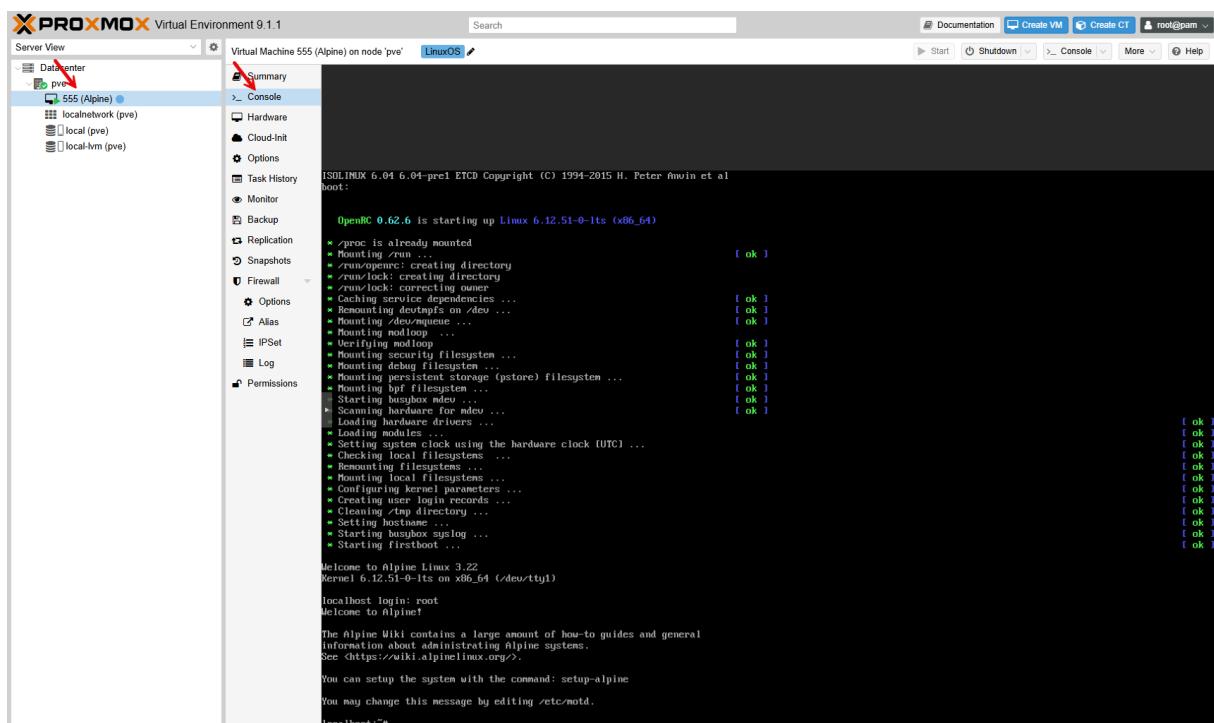
You can setup the system with the command: `setup-alpine`

You may change this message by editing `/etc/motd`.

localhost:~#

Als je dit venster sluit blijft de VM verder draaien.

Je kan ook op Console klikken zoals hieronder staat weergegeven.



15.2.7 VM instellen

15.2.7.1 Instellen toetsenbord

Dit is uiteraard specifiek voor het besturingssysteem.

Tijdelijk kan je dit als volgt instellen (even zoeken als je een querty-toetsenbord niet kent):

```
localhost:~# setup-keymap
```

Geef 2 keer be in

```
localhost:~# setup-keymap
af al an ar ar at az
ba bd be bg br
fr gb ge gh gr hr
md me nk nl mm mt
tw ua us uz vn

Select keyboard layout: [none] be
be-iso-alternate be-nodeakeys be-oss be-oss_latin9 be-wang be
Select variant (or 'abort'): be
* Caching service dependencies ...
* Setting keymap ...
localhost:~# [ ok ] [ ok ]
```

15.2.7.2 Instellen netwerk

We vragen de huidige netwerkconfiguratie op.

```
localhost:~# ip a
```

```
localhost:~# ip a
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether bc:24:11:b7:a3:22 brd ff:ff:ff:ff:ff:ff
localhost:~#
```

Je ziet dat de netwerkkaart uit (DOWN) staat. We zetten de netwerkkaart aan en vragen terug info over de netwerkkaart op.

```
localhost:~# ip link set eth0 up
```

```
localhost:~# ip a
```

```
localhost:~# ip a
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether bc:24:11:b7:a3:22 brd ff:ff:ff:ff:ff:ff
        inet 6 fe80::bc24:11ff:feb7:a322%6 link
              valid_lft forever preferred_lft forever
localhost:~#
```

Zoals je ziet heb je nog geen IPv4-adres gekregen.

Dat komt omdat er geen DHCP-client actief is. We maken de DHCP-client actief.

```
localhost:~# udhcpc -i eth0
```

```
localhost:~# udhcpc -i eth0
udhcpc: started, v1.37.0
udhcpc: broadcasting discover
udhcpc: broadcast select for 192.168.112.134, server 192.168.112.254
udhcpc: lease of 192.168.112.134 obtained from 192.168.112.254, lease time 1800
localhost:~#
```

We voeren terug ip a uit.

```
localhost:~# ip a
1: lo: <LOOPBACK,NOQUEUE> mtu 65536 qdisc noop state DOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether bc:24:11:b7:a3:22 brd ff:ff:ff:ff:ff:ff
        inet 192.168.112.134/24 brd 192.168.112.255 scope global eth0
            valid_lft forever preferred_lft forever
        inet6 fe80::be24:11ff:feb7:a322/64 scope link
            valid_lft forever preferred_lft forever
localhost:~#
```

Je ziet dat ik hier 192.168.112.134/24 heb gekomen.

We hebben nu ook verbinding met het internet.

```
localhost:~# ping -c 2 www.google.be
```

```
localhost:~# ip a
1: lo: <LOOPBACK,NOQUEUE> mtu 65536 qdisc noop state DOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether bc:24:11:b7:a3:22 brd ff:ff:ff:ff:ff:ff
        inet 192.168.112.134/24 brd 192.168.112.255 scope global eth0
            valid_lft forever preferred_lft forever
        inet6 fe80::be24:11ff:feb7:a322/64 scope link
            valid_lft forever preferred_lft forever
localhost:~# ping -c 2 www.google.be
PING www.google.be (142.250.179.163): 56 data bytes
64 bytes from 142.250.179.163: seq=0 ttl=128 time=24.442 ms
64 bytes from 142.250.179.163: seq=1 ttl=128 time=24.974 ms
--- www.google.be ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 24.442/24.700/24.974 ms
localhost:~#
```

15.3 Container

15.3.1 Vereisten

Containers in Proxmox vereisen de volgende configuratie:

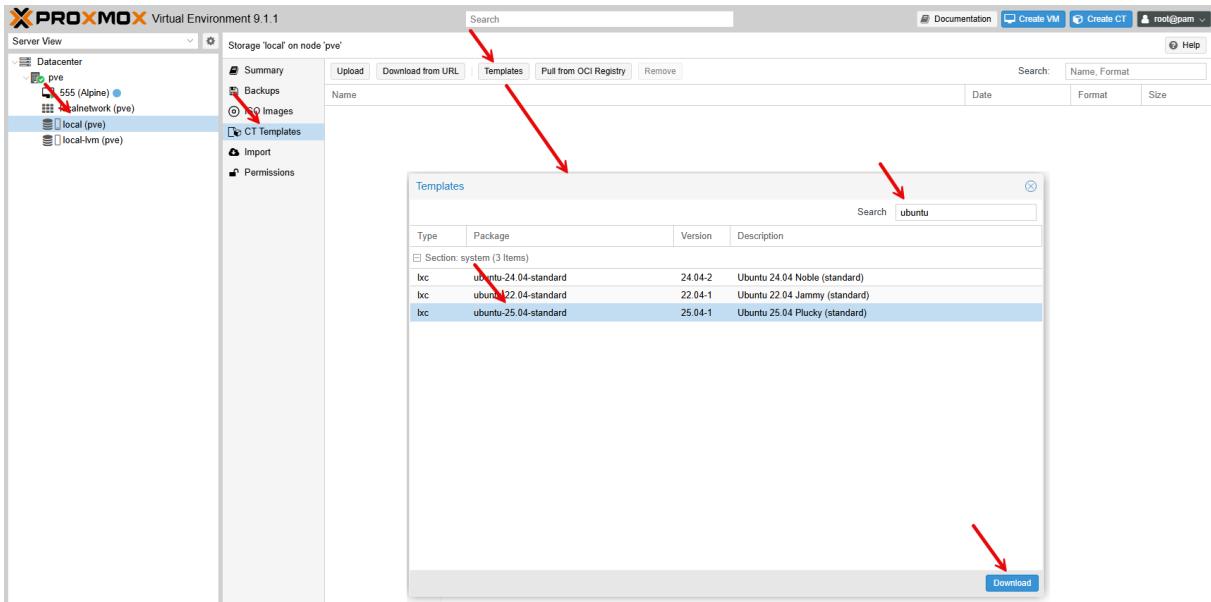
- Host OS: Een Linux-systeem (zoals Proxmox VE).
- Kernel: Ondersteuning voor LXC (standaard in Proxmox).
- Opslag: Een geschikte opslaglocatie, bijvoorbeeld ZFS, ext4, of LVM-Thin.

15.3.2 Container template downloaden

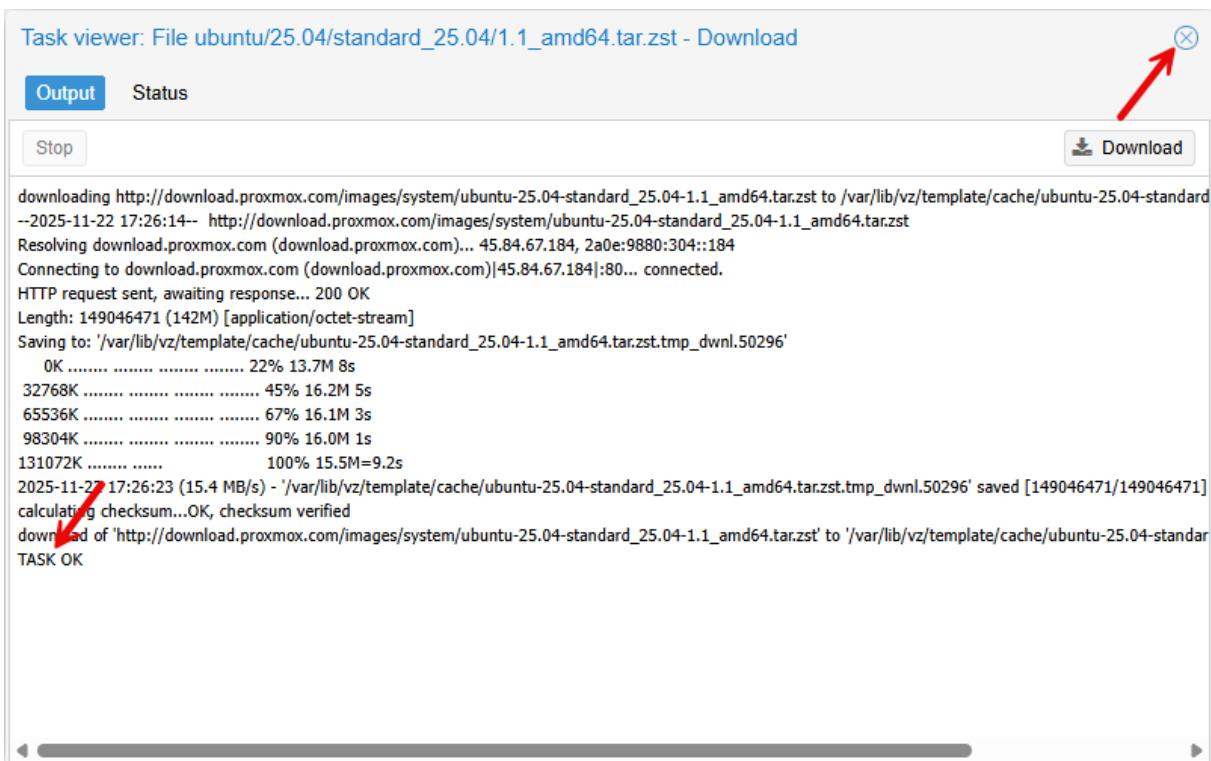
Container templates (min of meer vergelijkbaar met images in Docker) zijn vooraf geconfigureerde basisbesturingssystemen (zoals Ubuntu, Debian, Alpine, enz.) die je gebruikt om containers te maken.

Ga naar Datacenter, opslag local (pve) en klik op CT Templates, Templates.

We zoeken naar Ubuntu en selecteren ubuntu 25.10 om de template te downloaden.



Na enige tijd krijg je de melding TASK OK. Sluit het venster.



Hieronder vind je een vergelijking tussen een Docker image en een CT-template.

Aspect	CT-template (Proxmox)	Docker-image
Gebruiksscenario	Wordt gebruikt in LXC-containers binnen een hypervisor (Proxmox).	Wordt gebruikt binnen Docker en Podman om applicatiecontainers te draaien.
Type virtualisatie	OS-gebaseerde virtualisatie, waarbij het hele besturingssysteem wordt gedeeld.	Applicatiegebaseerde virtualisatie, gericht op specifieke applicaties.
Inhoud	Bevat een volledig root-bestandssysteem (zoals Debian, Alpine).	Bevat een basis-OS (bijv. alpine) met alleen de benodigde dependencies.
Besturingssysteem	Specifiek Linux-gebaseerd (geen Windows container support).	Ondersteunt zowel Linux- als Windows-images.
Opslagformaat	Comprimeert een basisbestandssysteem in een .tar.gz-bestand.	Images bestaan uit lagen (layers), opgeslagen in een union filesystem.
Configuratie	Instellingen (CPU, RAM, netwerk) via containerconfiguratie (bijv. 101.conf).	Configuratie via Dockerfile of bij het starten met docker run.
Flexibiliteit	Minder flexibel: werkt alleen met volledig root-bestandssysteem.	Zeer flexibel: < meerdere lagen mogelijk, afhankelijkheden.
Opstarttijd	Snel, maar iets langzamer dan Docker door omvang root-bestandssysteem.	Extreem snel door lichte aard van Docker-images.
Opslag-efficiëntie	Minder efficiënt: elke container heeft eigen volledige root-bestandssysteem.	Efficiënt: gedeelde lagen besparen opslagruimte.

15.3.3 Container aanmaken

Klik op Create CT rechtsboven.

The screenshot shows the Proxmox VE 9.1.1 interface. In the top right, there are several buttons: 'Documentation', 'Create VM', 'Create CT' (which has a red arrow pointing to it), 'root@pam', 'Bulk Actions', and 'Help'. Below these buttons is a table titled 'Datacenter' with columns for 'ID', 'Type', 'Content', 'Path/Target', 'Shared', 'Enabled', and 'Bandwidth Limit'. There are two entries: 'local' (Type: Directory, Content: Backup, Import, ISO image, Container template, Path/Target: /var/lib/vz) and 'local-lvm' (Type: LVM-Thin, Content: Disk image, Container, Path/Target: /var/lib/vz). On the left, a sidebar shows a tree structure with 'pve' selected, containing '555 (Alpine)', 'localnetwork (pve)', and 'local-lvm (pve)'.

- General

Een VM ID wordt automatisch toegewezen, maar u kunt ook een eigen ID invoeren (bijv. 777).

Geef een duidelijke naam, zoals Ubuntu2504.

Voeg optioneel tags toe om het doel van de VM te identificeren. Bijvoorbeeld: LinuxOS.

Je moet een wachtwoord ingeven (minimaal 5 karakters). Je ziet de mogelijkheid om een SSH Key File te laden.

Dit laten we nu ongemoeid.

The screenshot shows the 'Create: LXC Container' dialog. The 'General' tab is selected. The form contains the following fields:

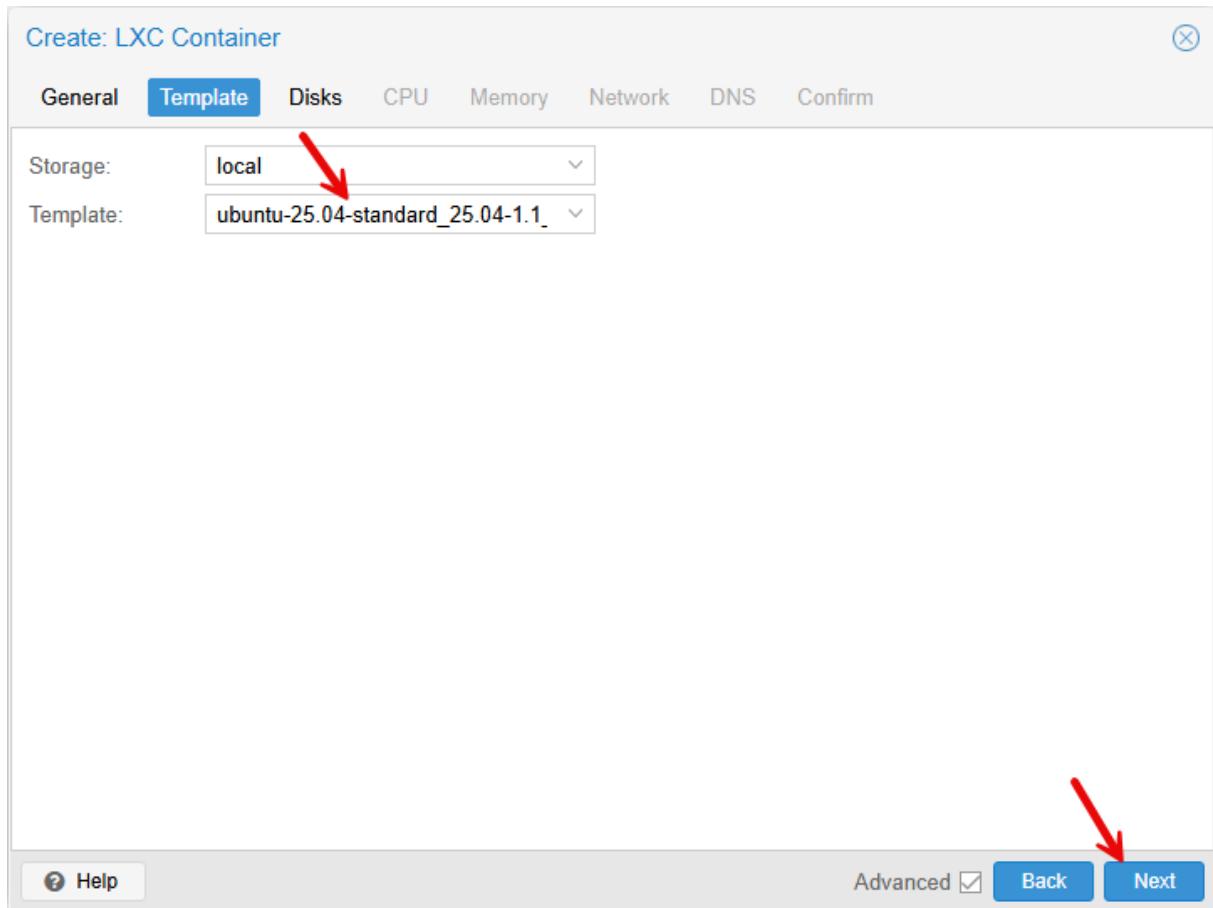
- Node: pve
- CT ID: 777
- Hostname: Ubuntu2504
- Unprivileged container:
- Nesting:
- Add to HA:
- Resource Pool: (dropdown)
- Password: (password field)
- Confirm password: (password field)
- SSH public key(s): (text area)
- Tags: LinuxOS (with a delete icon and a plus icon)

At the bottom right of the dialog, there is a 'Load SSH Key File' button and a 'Next' button.

Klik op Next.

- Template

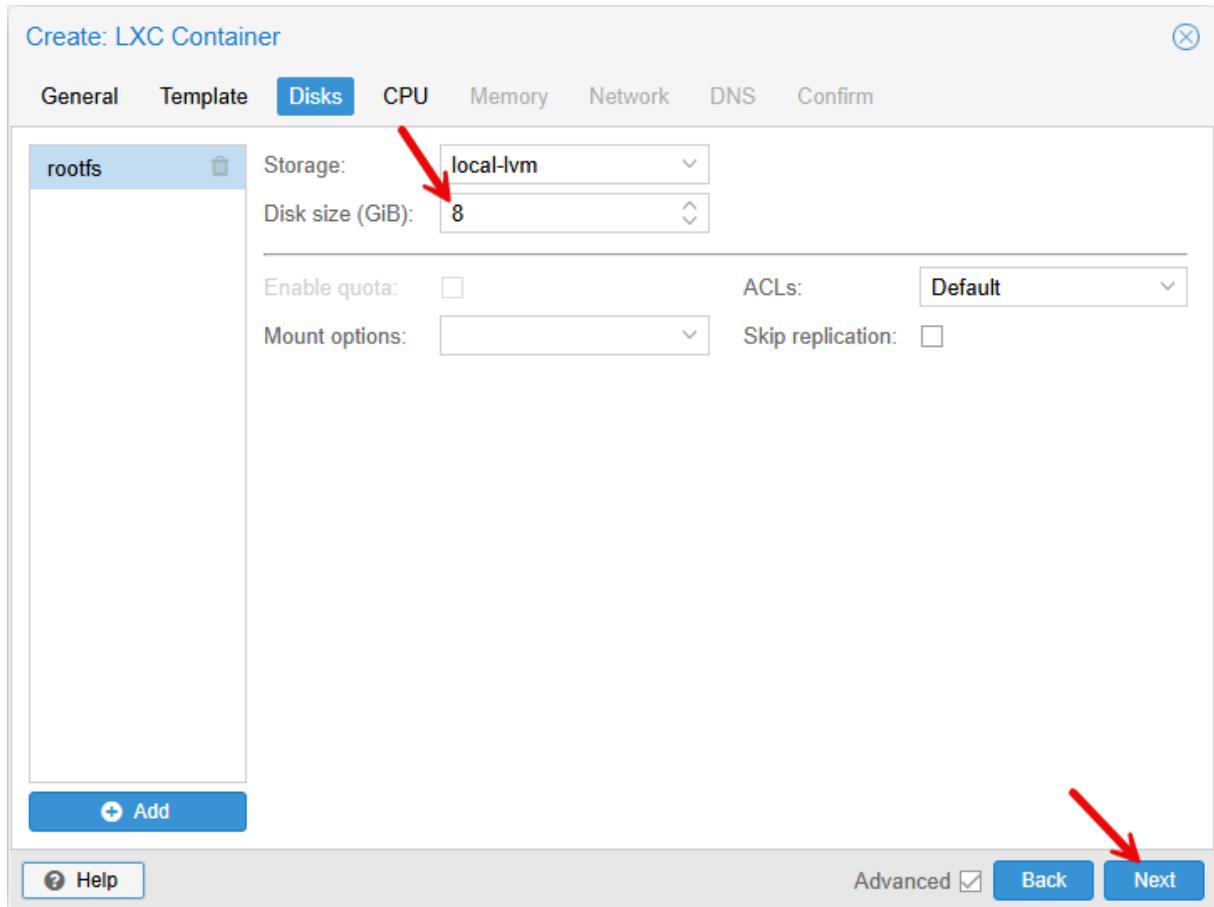
Het spreekt voor zich dat we de template van Ubuntu selecteren. We hebben geen andere.



We klikken erna op Next.

- Disks

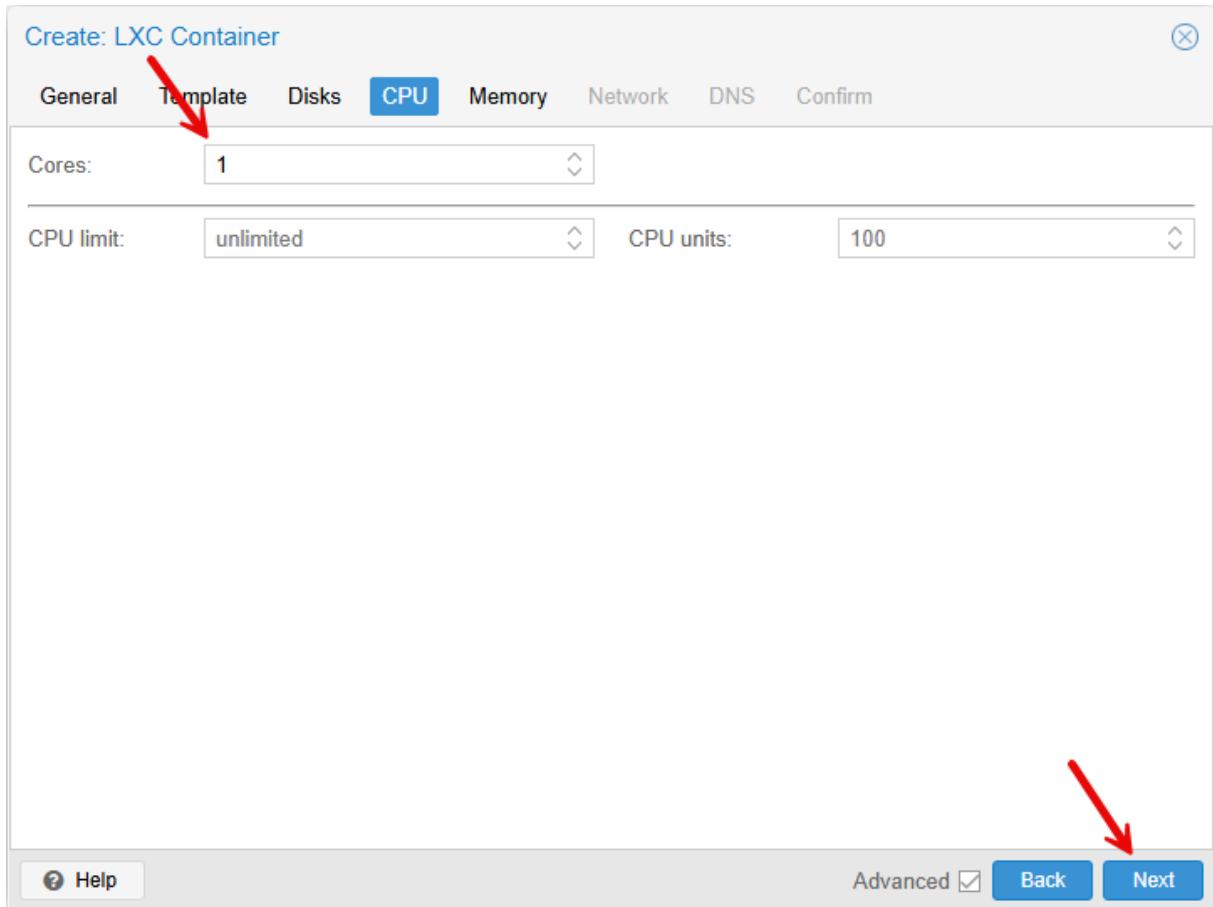
We laten een schijf van 8GiB staan. De andere instellingen laten we ongemoeid.



Klik op Next.

- CPU

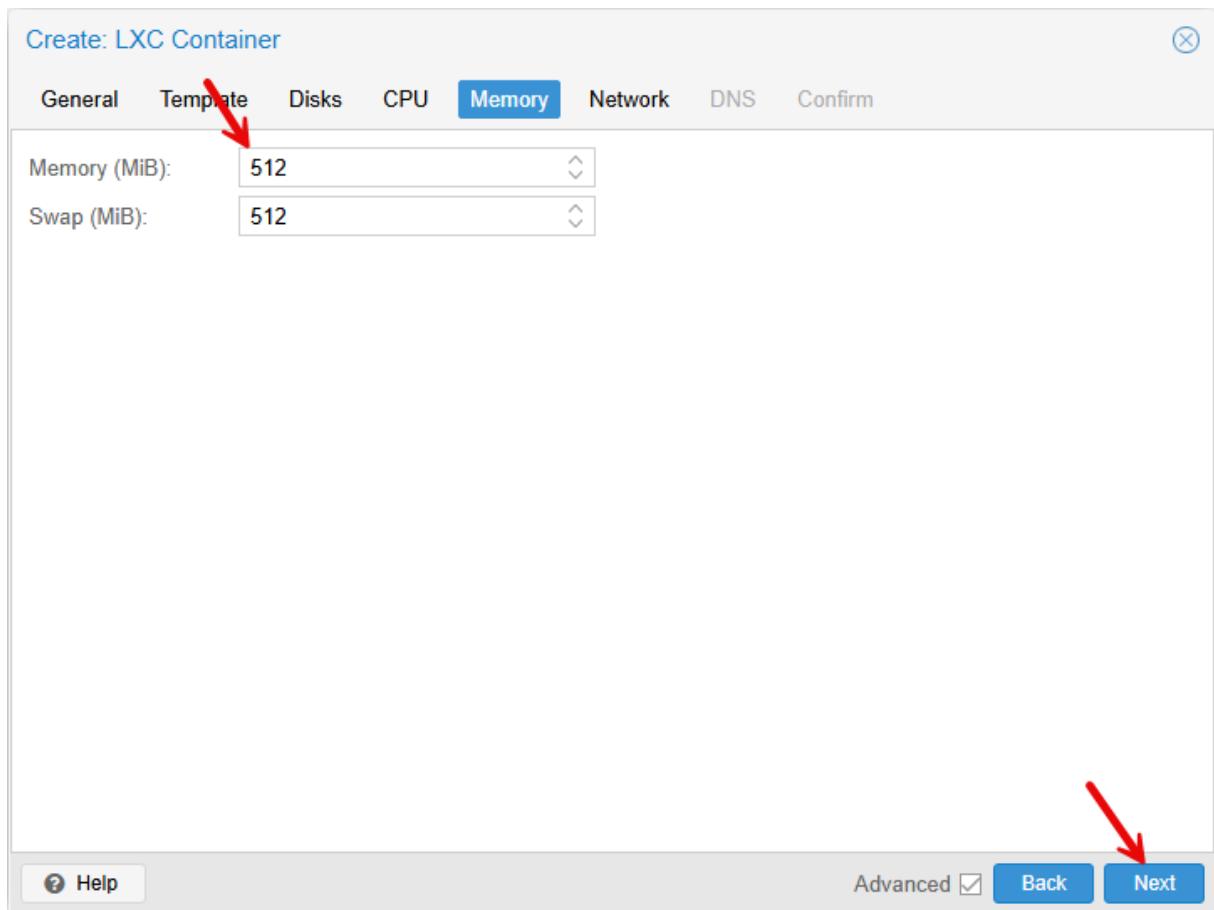
Bij CPU kan je weer het aantal cores instellen.



We laten dat nu onveranderd en klikken op Next.

- Memory

Je kan hier de grootte van geheugen instellen. We laten dit staan op 512 MB en klikken Next.



- Network

We verbinden de virtuele netwerkkaart met onze enige switch vmbr0. Stel in dat gebruik gemaakt wordt van DHCP.

Create: LXC Container

General Template Disks CPU Memory Network DNS Confirm

Name:	eth0	IPv4: <input type="radio"/> Static <input checked="" type="radio"/> DHCP
MAC address:	auto	IPv4/CIDR:
Bridge:	vmbr0	Gateway (IPv4):
VLAN Tag:	no VLAN	IPv6: <input type="radio"/> Static <input checked="" type="radio"/> DHCP <input type="radio"/> SLAAC
Firewall:	<input checked="" type="checkbox"/>	IPv6/CIDR:
Disconnect:	<input type="checkbox"/>	Rate limit (MB/s): unlimited
MTU:	Same as bridge	Host-Managed: <input type="checkbox"/>

Help Advanced Back Next

A red arrow points to the 'DHCP' radio button under 'IPv4' and another red arrow points to the 'DHCP' radio button under 'IPv6'. A final red arrow points to the 'Next' button at the bottom right.

Klik erna op Next.

- DNS

De DNS-instellingen laten we ongemoeid. Kik erna op Next.

Create: LXC Container (X)

General Template Disks CPU Memory Network **DNS** Confirm

DNS domain:

DNS servers:

Advanced **Back** **Next**



- Confirm
Je krijgt nu een overzicht. Kijk je instellingen na.

Create: LXC Container

General Template Disks CPU Memory Network DNS Confirm

Key ↑	Value
cores	1
features	nesting=1
hostname	Ubuntu2504
memory	512
net0	name=eth0,bridge=vmbr0,firewall=1,ip6=dhcp,ip=dhcp
nodename	pve
ostemplate	local:vztmp/ubuntu-25.04-standard_25.04-1.1_amd64.tar.zst
pool	
rootfs	local-lvm:8
ssh-public-keys	
swap	512
tags	LinuxOS
unprivileged	1
vmid	777

Start after created

Advanced Back **Finish**



De feature "nesting" in een LXC-container betekent dat je binnen die container andere containers kunt draaien. Het maakt dus container-in-container mogelijk, wat handig is voor ontwikkel- en testomgevingen.

Klik op Finish.

Na enige tijd zal je weer de melding TASK OK krijgen.

Task viewer: CT 777 - Create

Output Status

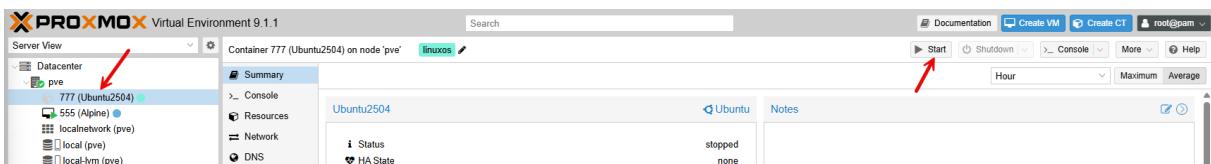
Stop Download

```
Logical volume "vm-777-disk-0" created.
Logical volume pve/vm-777-disk-0 changed.
Creating filesystem with 2097152 4k blocks and 524288 inodes
Filesystem UUID: 07965b76-e876-4df6-9757-81d372ab2ce2
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632
extracting archive '/var/lib/vz/template/cache/ubuntu-25.04-standard_25.04-1.1_amd64.tar.zst'
Total bytes read: 590848000 (564MiB, 113MiB/s)
Detected container architecture: amd64
Creating SSH host key 'ssh_host_ecdsa_key' - this may take some time ...
done: SHA256:y94SIegrjZZhPE9TVbtawfJUWxpibpiNMIGBuRetQ root@Ubuntu2504
Creating SSH host key 'ssh_host_ed25519_key' - this may take some time ...
done: SHA256:Z1rm60pSIBo1U9uRZqXRWVAFWvUCAn8ZWYLeugBexI root@Ubuntu2504
Creating SSH host key 'ssh_host_rsa_key' - this may take some time ...
done: SHA256:ZPUMHLoBU9g6+UwaLTDXuHNk9keu5fOKUC06QjTSufQ root@Ubuntu2504
TASK OK
```

Sluit dat venster.

15.3.4 Container starten

Selecteer links de container van Ubuntu2504 en klik op Start.



Je kan inloggen op Ubuntu door op Console te klikken op 1 van de 2 plaatsen zoals reeds besproken.



Log in met de user root en geef het door jouw gekozen wachtwoord in.

pve - Proxmox Console - Google Chrome

Niet beveiligd https://192.168.112.133:8006/?console=lxc&xtermjs=1&vmid=777&vmname=&node=pve&cmd=

```
login: timed out after 60 seconds
Ubuntu 25.04 Ubuntu2504 tty1

Ubuntu2504 login: root
Password:
Welcome to Ubuntu 25.04 (GNU/Linux 6.17.2-1-pve x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

root@Ubuntu2504:~#
```

16 Clusters in Proxmox

16.1 Inleiding

We zullen onderstaande instellen.

The screenshot shows the Proxmox VE 8.3.0 interface. On the left, the 'Server View' sidebar has 'Datacenter (cluster)' selected, which is highlighted with a red box. Underneath, there are three nodes listed: 'pve', 'pve2', and 'pve3'. To the right, the main panel is titled 'Datacenter' and contains a search bar and a table of nodes. The table has columns for 'Type ↑' and 'Description'. It lists three nodes: 'node pve', 'node pve2', and 'node pve3'.

Type ↑	Description
node	pve
node	pve2
node	pve3

We geven volgende IP-nummers

Hostname	IP
pve	192.168.112.110
pve2	192.168.112.120
pve3	192.168.112.130

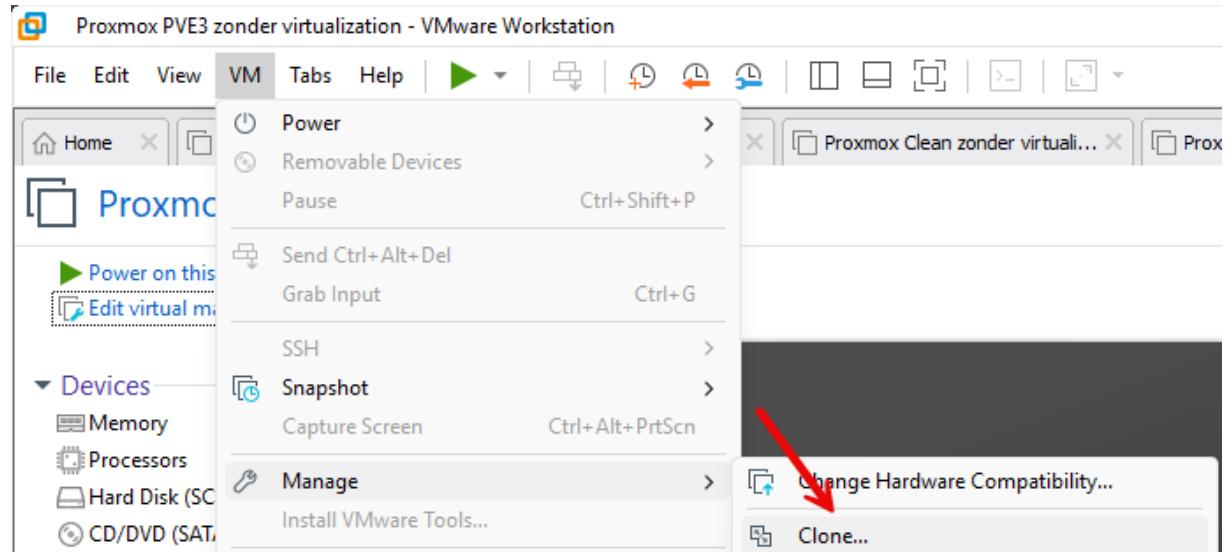
16.2 Nodes aanmaken en configureren

16.2.1 Mappen kopiëren

Kopieer je Proxmox Clean 3 keer zoals hieronder staat aangegeven.

- 📁 Proxmox PVE3
- 📁 Proxmox PVE
- 📁 Proxmox PVE2
- 📁 Proxmox Clean

Je kan uiteraard de VM ook克lonen via de menubalk van VMware maar het resultaat is hetzelfde.

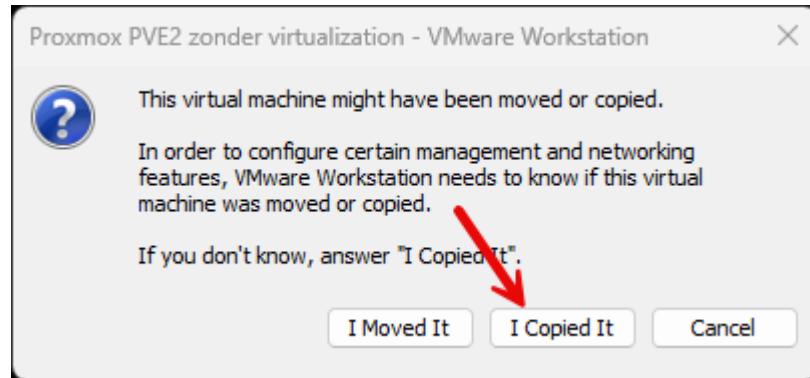


16.2.2 IP-nummer instellen

Als voorbeeld voer ik de instellingen uit op PVE2. De andere VM's moet je uiteraard ook correct instellen.

Start PVE2 op.

Belangrijk: Kies voor I Copied it al je de VM start.



- Log in op de VM zelf (niet via SSH – dat werkt (nog) niet).

```
Welcome to the Proxmox Virtual Environment. Please use your web browser to
configure this server - connect to:
https://192.168.112.138:8006/

pve login: root
Password:
Linux pve 6.17.2-1-pve #1 SMP PREEMPT_DYNAMIC PMX 6.17.2-1 (2025-10-21T11:55Z) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@pve:~#
```

- Controleer de huidige netwerkconfiguratie

Bekijk de huidige netwerkconfiguratie om te zien welke instellingen actief zijn:

```
root@pve:~# ip a
```

```
root@pve:~# 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: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 00:0c:29:3e:1a:66 brd ff:ff:ff:ff:ff:ff
        altname enp2s1
        altname enx00c293e1a66
3: vmbr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default qlen 1000
    link/ether 52:cc:71:e4:18:ae brd ff:ff:ff:ff:ff:ff
        inet 192.168.112.138/24 scope global vmbr0
            valid_lft forever preferred_lft forever
            inet6 fe80::52cc:71ff:fe41:10ae/64 scope link proto kernel ll
                valid_lft forever preferred_lft forever
root@pve:~#
```

Je ziet down staan bij apparaat ens33...

- Bewerk het netwerkconfiguratiebestand

Proxmox gebruikt standaard /etc/network/interfaces om netwerkinstellingen te beheren.

Open dit bestand met een teksteditor:

```
root@pve:~# nano /etc/network/interfaces
```

- Stel een gewijzigd IP-adres in en fix de naam van de netwerkkaart

Zoek in de configuratie naar nic0 en verander dat in ens33. Je moet dat twee keer wijzigen.

Pas ook de IP-nummer aan. Je moet onderstaand resultaat bekomen:

```
auto lo
iface lo inet loopback
```

```

iface ens33 inet manual

auto vmbr0
iface vmbr0 inet static

    address 192.168.112.120/24
    gateway 192.168.112.2
    bridge-ports ens33
    bridge-stp off
    bridge-fd 0

source /etc/network/interfaces.d/*

```

```

GNU nano 8.4                               /etc/network/interfaces *

auto lo
iface lo inet loopback

iface ens33 inet manual

auto vmbr0
iface vmbr0 inet static
    address 192.168.112.120/24
    gateway 192.168.112.2
    bridge-ports ens33
    bridge-stp off
    bridge-fd 0

source /etc/network/interfaces.d/*


[ Cancelled ]

```

Sla uiteraard dit document op.

- Configuratie controleren

Controleer het configuratiebestand op fouten voordat je het toepast:

```
root@pve:~# ifup --no-act ens33
```

- Herstart de VM:

```
root@pve:~# reboot
```

- Controleer de nieuwe configuratie

Controleer of de interface nu het ingestelde IP-adres gebruikt:

```
root@pve:~# ip a
```

```
Welcome to the Proxmox Virtual Environment. Please use your web browser to
configure this server - connect to:
https://192.168.112.138:8006/
-----
pve login: root
Password:
Linux pve 6.17.2-1-pve #1 SMP PREEMPT_DYNAMIC PMX 6.17.2-1 (2025-10-21T11:55Z) x86_64
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@pve:~# 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: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master vmbr0 state UP group default qlen 1000
    link/ether 00:0c:29:3e:1a:66 brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    altname enx000c293e1a66
3: vmbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 00:0c:29:3e:1a:66 brd ff:ff:ff:ff:ff:ff
    inet 192.168.112.129/24 brd 192.168.112.255 scope global vmbr0
        valid_lft forever preferred_lft forever
        inet6 fe80::0c29:3e!%vmbr0/64 scope link proto kernel ll
            valid_lft forever preferred_lft forever
root@pve:~# _
```

- Test de verbinding

Test of het vaste IP-adres correct werkt:

```
root@pve:~# ping www.google.be
```

```
root@pve:~# ping www.google.be
PING www.google.be (142.251.173.94) 56(84) bytes of data.
64 bytes from wi-in-f94.1e100.net (142.251.173.94): icmp_seq=1 ttl=128 time=21.9 ms
64 bytes from wi-in-f94.1e100.net (142.251.173.94): icmp_seq=2 ttl=128 time=19.3 ms
64 bytes from wi-in-f94.1e100.net (142.251.173.94): icmp_seq=3 ttl=128 time=22.2 ms
64 bytes from wi-in-f94.1e100.net (142.251.173.94): icmp_seq=4 ttl=128 time=35.1 ms
^C
--- www.google.be ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3002ms
rtt min/avg/max/mdev = 19.346/24.645/35.056/6.114 ms
root@pve:~# _
```

16.2.3 Hostname instellen

- SSH

Log u nu, via ssh, in op Proxmox PVE2.

```
root@pve:~# ssh root@192.168.112.120
```

```

C:\Users\stijn>ssh root@192.168.112.120
The authenticity of host '192.168.112.120 (192.168.112.120)' can't be established.
ED25519 key fingerprint is SHA256:NSeWqNeAqczaY3rxarNUspOpDw8h5Qz/+zdh/w3DZas.
This host key is known by the following other names/addresses:
  C:\Users\stijn/.ssh/known_hosts:19: 192.168.112.138
  C:\Users\stijn/.ssh/known_hosts:22: 192.168.112.110
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.112.120' (ED25519) to the list of known hosts.
root@192.168.112.120's password:
Linux pve 6.17.2-1-pve #1 SMP PREEMPT_DYNAMIC PMX 6.17.2-1 (2025-10-21T11:55Z) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Nov 25 11:34:53 2025 from 192.168.112.1
root@pve:~# |

```

- Controleer de huidige hostname

Voer het volgende commando uit om de huidige hostname te controleren:

root@pve:~# hostname

pve

Deze staat nog fout...

- Pas de hostname aan

Verander de hostname op de volgende locaties:

- **/etc/hostname**

Bewerk het bestand met een teksteditor, zoals nano:

root@pve:~# nano /etc/hostname

pve2

Vervang de oude hostname dus door de nieuwe hostname pve2.

```

GNU nano 8.4          /etc/hostname *
pve2

^G Help      ^O Write Out    ^F Where Is    ^K Cut        ^T Execute    ^C Location   M-U Undo
^X Exit      ^R Read File    ^\ Replace     ^U Paste      ^J Justify    ^/ Go To Line M-E Redo
                                         ^L Set Mark   M-G Copy

```

Sla het bestand op en sluit de editor.

- **/etc/hosts**

Bewerk het bestand:

root@pve:~# nano /etc/hosts

```

GNU nano 8.4                               /etc/hosts *
127.0.0.1 localhost.localdomain localhost
192.168.112.120 pve2.localdomain pve2

# The following lines are desirable for IPv6 capable hosts

::1      ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts

^G Help      ^O Write Out    ^F Where Is    ^K Cut        ^T Execute    ^C Location    M-U Undo
^X Exit      ^R Read File   ^\ Replace     ^U Paste      ^J Justify    ^/ Go To Line  M-E Redo
                                         M-A Set Mark  M-B Copy

```

Let op: pas ook de IP-nummer aan!

- Pas de hostname toe zonder herstart

Voer het volgende commando uit om de nieuwe hostname onmiddellijk toe te passen:

```
root@pve:~# hostnamectl set-hostname pve2.
```

```
root@pve:~# exec bash
```

```
pve2
```

- Herstart de server

Voor een volledige toepassing van de wijzigingen, herstart je best de server:

```
root@pve:~# reboot
```

- Check inlog

Dit zou nu correct moeten staan.

```

-----
Welcome to the Proxmox Virtual Environment. Please use your web browser to
configure this server - connect to:
https://192.168.112.120:8006/
-----
pve2 login: _

```

16.3 Installatie Cluster

16.3.1 Inleiding

We kunnen nu onze servers samenvoegen om een cluster te vormen.

Om deze taak uit te voeren, moeten we slechts drie stappen volgen.

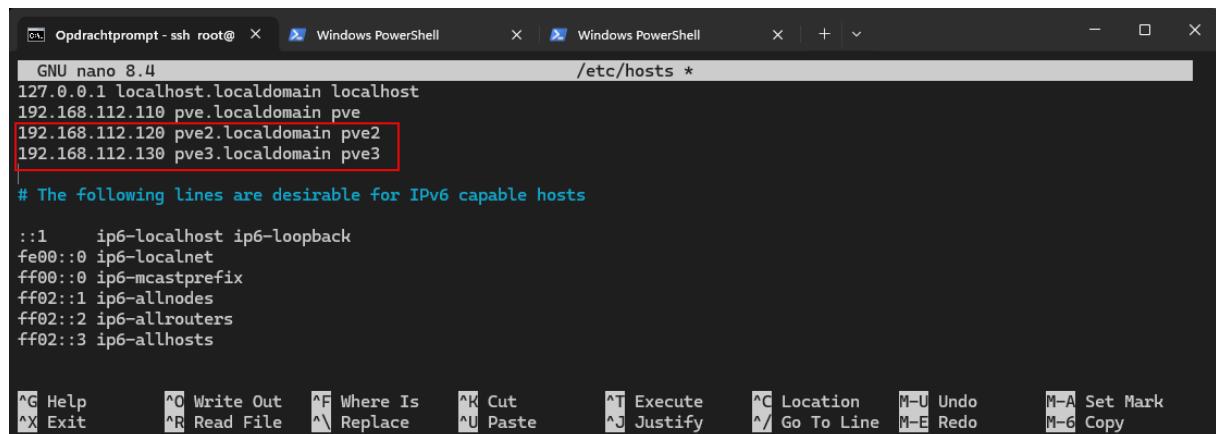
- We maken een nieuw cluster met server één met een eenvoudige opdracht.
- Voeg server twee toe aan het cluster.
- Voeg server drie toe aan het cluster.

16.3.2 Voorbereiding

We stellen via /etc/hosts in dat de servers elkaar kunnen vinden aan hand van de hostname.

Hieronder staat ingesteld hoe je dat doet op PVE.

```
root@pve:~# nano /etc/hosts
192.168.112.120 pve2.localdomain pve2
192.168.112.130 pve3.localdomain pve3
```



```
GNU nano 8.4                               /etc/hosts *
127.0.0.1 localhost.localdomain localhost
192.168.112.110 pve.localdomain pve
192.168.112.120 pve2.localdomain pve2
192.168.112.130 pve3.localdomain pve3

# The following lines are desirable for IPv6 capable hosts

::1      ip6-localhost ip6-loopback
fe00::0  ip6-localnet
ff00::0  ip6-mcastprefix
ff02::1  ip6-allnodes
ff02::2  ip6-allrouters
ff02::3  ip6-allhosts

^G Help     ^O Write Out   ^F Where Is    ^K Cut        ^T Execute    ^C Location   M-U Undo
^X Exit     ^R Read File   ^A Replace    ^U Paste      ^J Justify    ^I Go To Line M-E Redo
                                         M-A Set Mark  M-G Copy
```

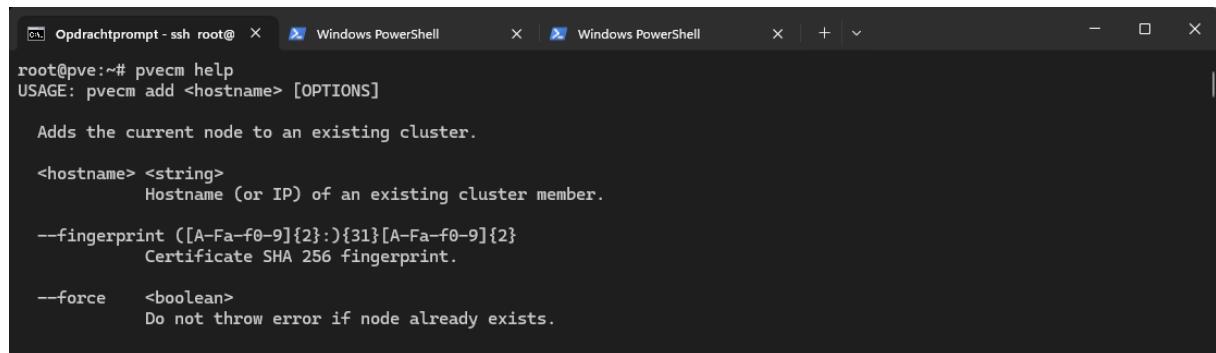
Uiteraard dien je de andere hosts toe te voegen op PVE, PVE2 en PVE3.

16.3.3 PVE

We loggen via SSH in op de eerste server (PVE).

Proxmox biedt een tool genaamd pvecm, wat staat voor Proxmox VE Cluster Manager.

```
root@pve:~# pvecm help
```



```
root@pve:~# pvecm help
USAGE: pvecm add <hostname> [OPTIONS]

Adds the current node to an existing cluster.

<hostname> <string>
    Hostname (or IP) of an existing cluster member.

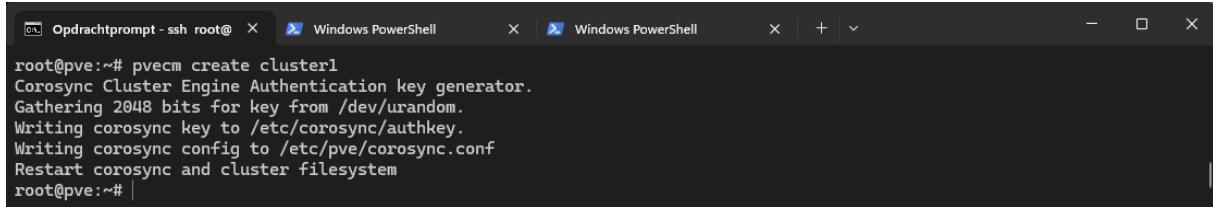
--fingerprint ([A-Fa-f0-9]{2}:){31}[A-Fa-f0-9]{2}
    Certificate SHA 256 fingerprint.

--force    <boolean>
    Do not throw error if node already exists.
```

Met deze tool kun je taken uitvoeren zoals het maken van een nieuw cluster, servers toevoegen, servers verwijderen, statusinformatie ophalen en andere gerelateerde taken uitvoeren.

Om een cluster te maken, gebruik je de opdracht pvecm create, gevolgd door de naam van het cluster.

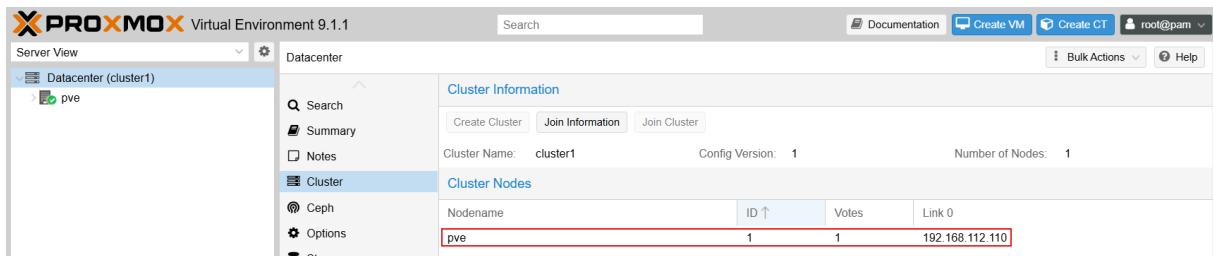
```
root@pve:~# pvecm create cluster1
```



```
root@pve:~# pvecm create cluster1
Corosync Cluster Engine Authentication key generator.
Gathering 2048 bits for key from /dev/urandom.
Writing corosync key to /etc/corosync/authkey.
Writing corosync config to /etc/pve/corosync.conf
Restart corosync and cluster filesystem
root@pve:~# |
```

En dat is het! De cluster met de naam cluster1 is aangemaakt.

Je vindt dit ook terug in de grafische omgeving.



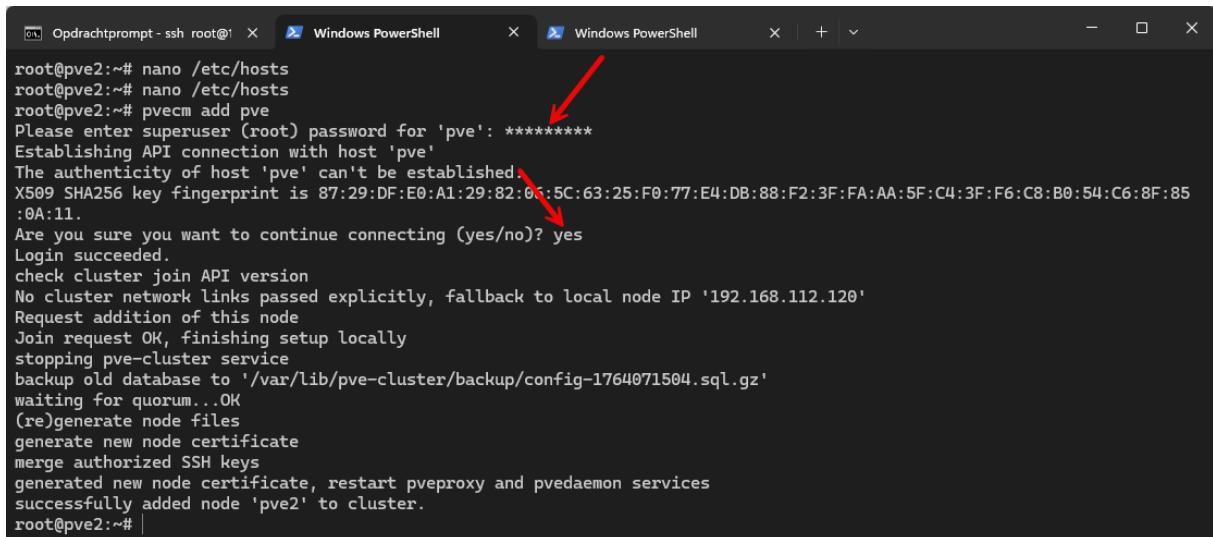
Nodename	ID ↑	Votes	Link 0
pve	1	1	192.168.112.110

16.3.4 PVE2

Laten we nu naar PVE2 gaan en deze verbinden met het cluster.

Hiervoor gebruik je opnieuw de pvecm-opdracht, maar in dit geval gebruik je pvecm add en geef je het IP-adres of de hostnaam van de server die al lid is van het cluster, in dit geval PVE.

```
root@pve2:~# pvecm add pve
```



```
root@pve2:~# nano /etc/hosts
root@pve2:~# nano /etc/hosts
root@pve2:~# pvecm add pve
Please enter superuser (root) password for 'pve': *****
Establishing API connection with host 'pve'
The authenticity of host 'pve' can't be established.
X509 SHA256 key fingerprint is 87:29:DF:E0:A1:29:82:03:5C:63:25:F0:77:E4:DB:88:F2:3F:FA:AA:5F:C4:3F:F6:C8:B0:54:C6:8F:85
:0A:11.
Are you sure you want to continue connecting (yes/no)? yes
Login succeeded.
check cluster join API version
No cluster network links passed explicitly, fallback to local node IP '192.168.112.120'
Request addition of this node
Join request OK, finishing setup locally
stopping pve-cluster service
backup old database to '/var/lib/pve-cluster/backup/config-1764071504.sql.gz'
waiting for quorum...OK
(re)generate node files
generate new node certificate
merge authorized SSH keys
generated new node certificate, restart pveproxy and pvedaemon services
successfully added node 'pve2' to cluster.
root@pve2:~# |
```

Je voert het root-wachtwoord in. Je moet ook yes klikken.

De melding "The authenticity of host 'pve' can't be established" betekent dat de Proxmox-server (in dit geval pve) waarmee je probeert te verbinden, een SSL-certificaat gebruikt waarvan de authenticiteit niet vooraf kan worden geverifieerd. Dit gebeurt vaak bij zelf-ondertekende certificaten, die standaard door Proxmox worden gebruikt.

Nu zijn twee servers verbonden met het cluster.

16.3.5 PVE3

Tot slot voegen we PVE3 toe aan het cluster. We zullen dat nu grafisch instellen.

Ga naar <http://192.168.112.110:8006> of <http://192.168.112.120:8006>.

Je zal zien dat de 2 nodes zichtbaar zijn van beide websites.

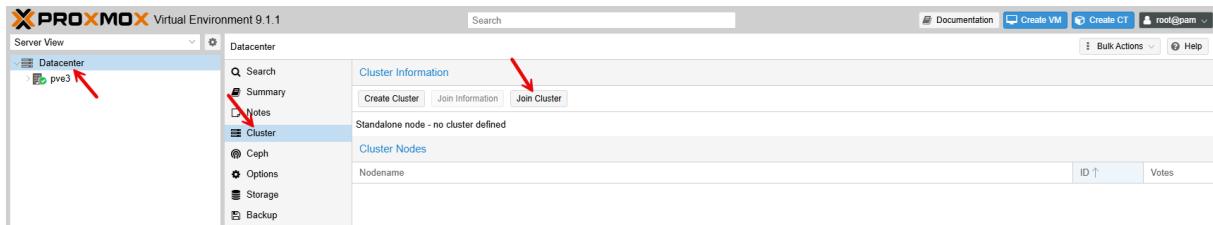
Nodename	ID ↑	Votes	Link
pve	1	1	192.168.112.110
pve2	2	1	192.168.112.120

Klik bij Datacenter op Cluster en erna op Join Information.

Klik op Copy Information.

Ga nu naar <http://192.168.112.130:8006>.

Kies daar voor Datacenter en klik op Cluster en erna op Join Cluster.



Voer <CTRL> + V uit in het veld information en voer het root-password in. Klik erna op Join 'cluster1'.

Cluster Join

Assisted join: Paste encoded cluster join information and enter password.

Information: kM2OjhGOjg1OjBBOjExliwicGVlckxpbtzJp7ljAiOixOTluMTY4LjExMi4xMTAifSwicmluZ19hZGRyljpbljE5Mi4xNjguMTEyLjExMCJdLCJ0b3RlbSI6eyJzZWNhdxR0ljoib24iLCJsaW5rX21vZGUiOijwYXNzaXZlIiwidmVyc2lvbil6ljilCJjb25maWdfdmVyc2lvbil6ljilCJpcf92ZXJzaW9uljoiaXB2NC02liwiY2x1c3RlcI9uYW1lljoIY2x1c3RlcjElLCJpbnRlcmbZhY2UiOnsiMCI6eyJsaW5rbnVtYmVyljoIMCJ9FX19

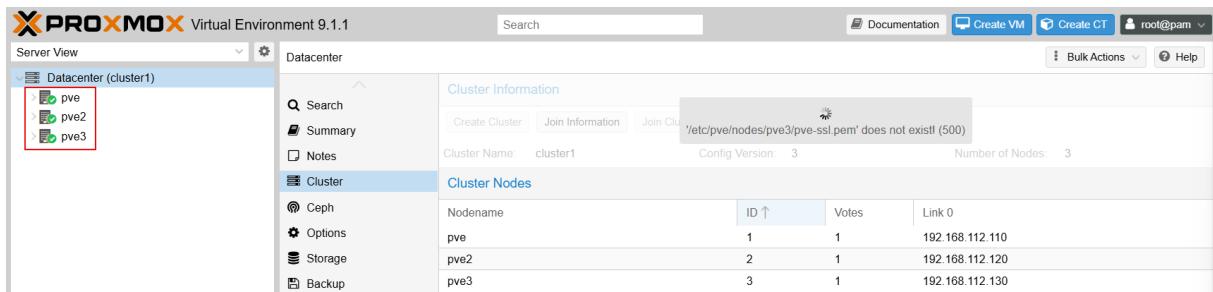
Peer Address: 192.168.112.110 Password:

Fingerprint: 87:29:DF:E0:A1:29:82:06:5C:63:25:F0:77:E4:DB:88:F2:3F:FA:AA:5F:C4:3F:F6:C8:B0:54:C6:8F:85:0A:11

Cluster Network: Link: 0 IP resolved by node's hostname peer's link address: 192.168.112.110

[Help](#) [Join 'cluster1'](#)

Nu kun je alle drie de servers beheren via dezelfde webinterface via alle IP-adressen van de nodes.



Op dit moment hebben we dus een cluster die centraal beheerd kan worden. Je kunt nu virtuele machines aanmaken en migreren, maar niet alle clustermogelijkheden zijn beschikbaar.

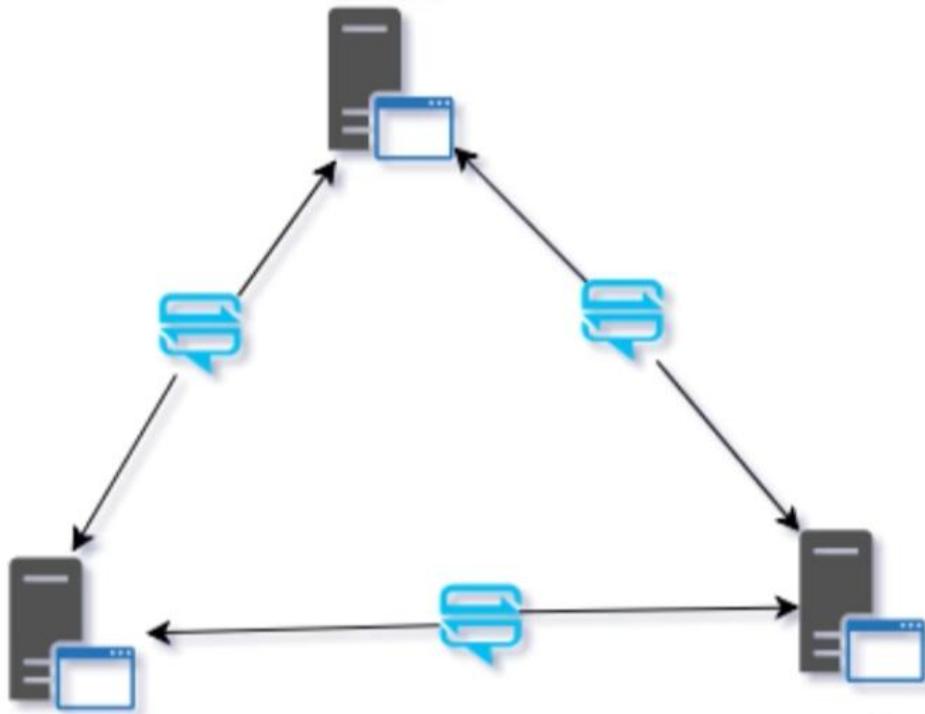
16.4 High availability en live migratie

16.4.1 Algemeen

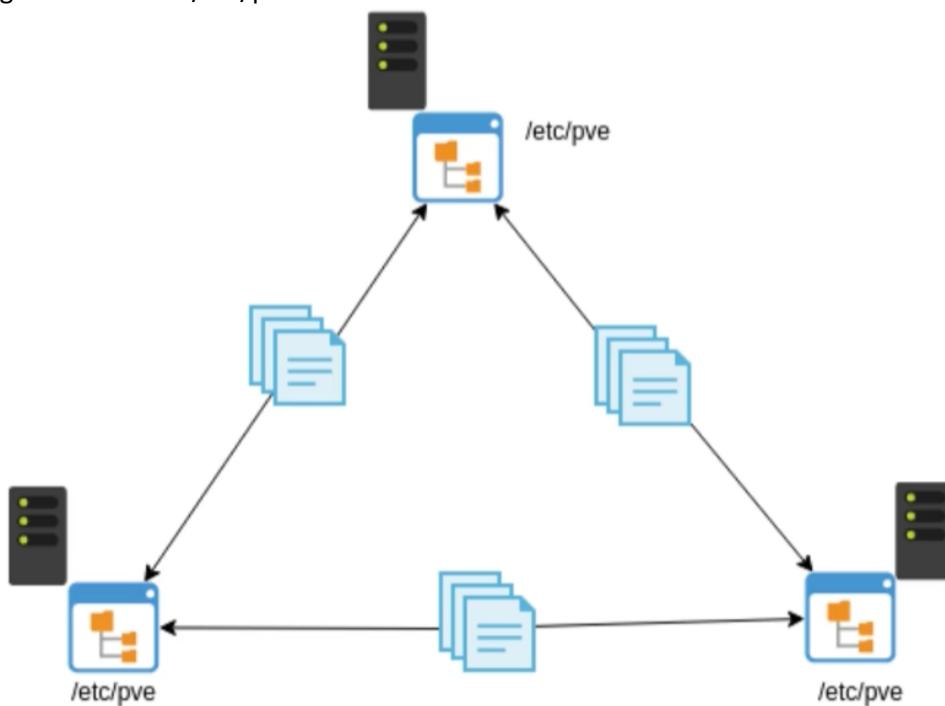
Omdat we nog geen gedeelde opslag hebben, kun je bijvoorbeeld geen high availability-groepen of live migratie gebruiken.

De belangrijkste technologieën die gebruikt worden in de cluster:

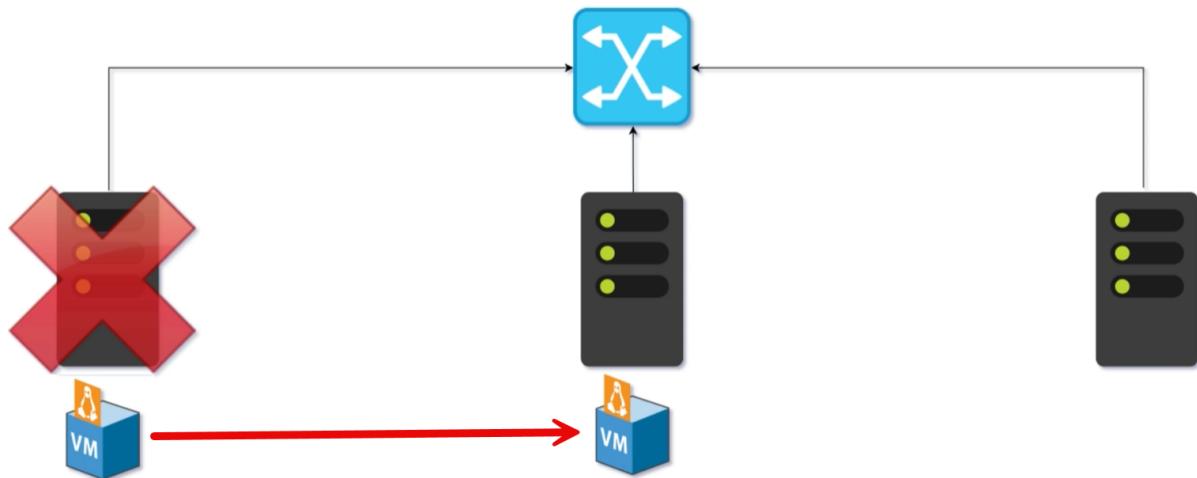
- Corosync: Een communicatiesysteem dat servers in het cluster informeert over de status van andere leden.
 - o Corosync heeft een betrouwbaar netwerk nodig met lage latency.
 - o Corosync is een kritische applicatie
 - o Corosync zorgt ervoor dat servers elkaars status kunnen zien.



- Proxmox Cluster File System (PMXCF): Een databasegestuurd bestandssysteem dat configuratiebestanden in real-time repliceert naar alle servers in het cluster. Dit wordt gemount onder /etc/pve.



Dit zorgt ervoor dat als een VM of Container op een server faalt deze VM verplaatst wordt naar een andere server.



We richten ons even op virtuele machines (VM's) en wat een VM eigenlijk is vanuit het perspectief van een besturingssysteem en een hypervisor.

Allereerst is een VM een configuratiebestand dat alle essentiële informatie over de VM bevat, zoals de CPU-configuratie, hoeveelheid geheugen, opslag, bustechnologie, netwerkkaarten, enzovoort. Om een VM te starten, leest de hypervisor dit configuratiebestand, wijst hij de benodigde resources toe en emuleert hij hardware op basis van de instructies in dit bestand.

Het tweede belangrijke onderdeel van een virtuele machine is de image, die als opslag aan de VM wordt gekoppeld.



16.4.2 Migratie zonder gedeelde opslag

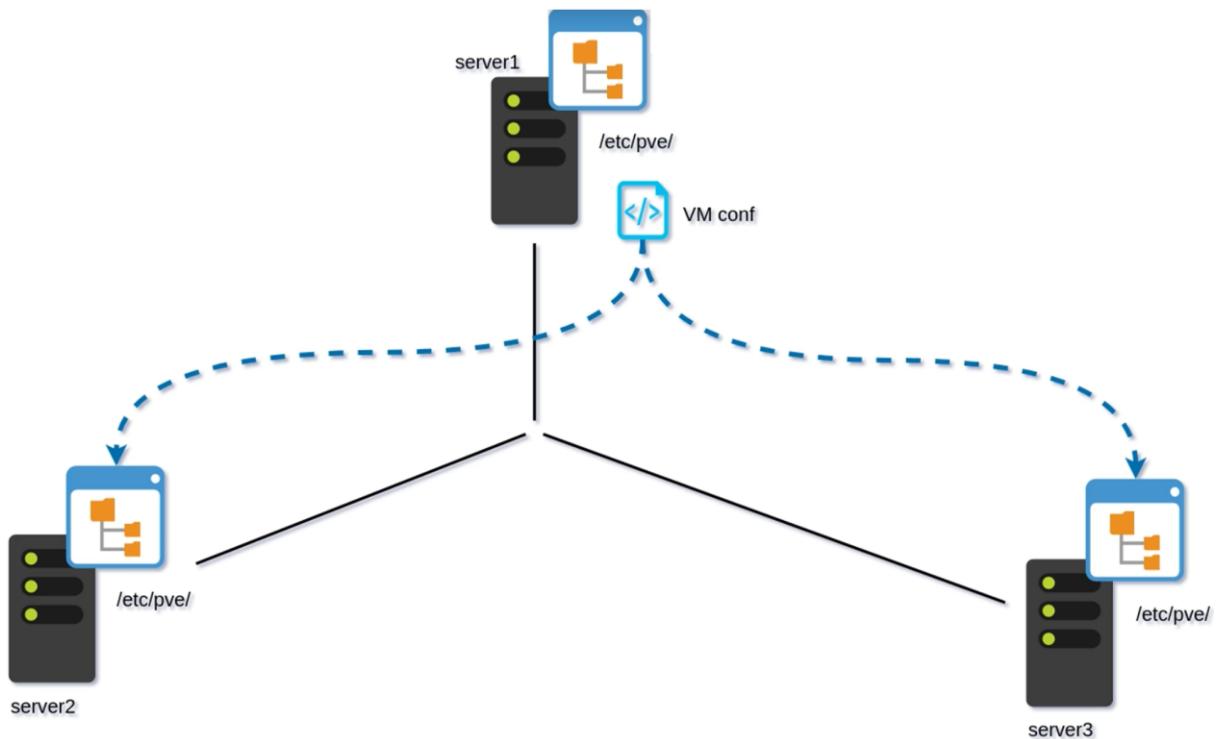
Stel je nu voor dat je een migratie moet uitvoeren zonder cluster, maar met een set losse servers.

Je moet dan het configuratiebestand van de ene server naar de andere kopiëren en vervolgens de image naar de doelsysteemserver overdragen.



16.4.3 Migratie met gedeelde opslag

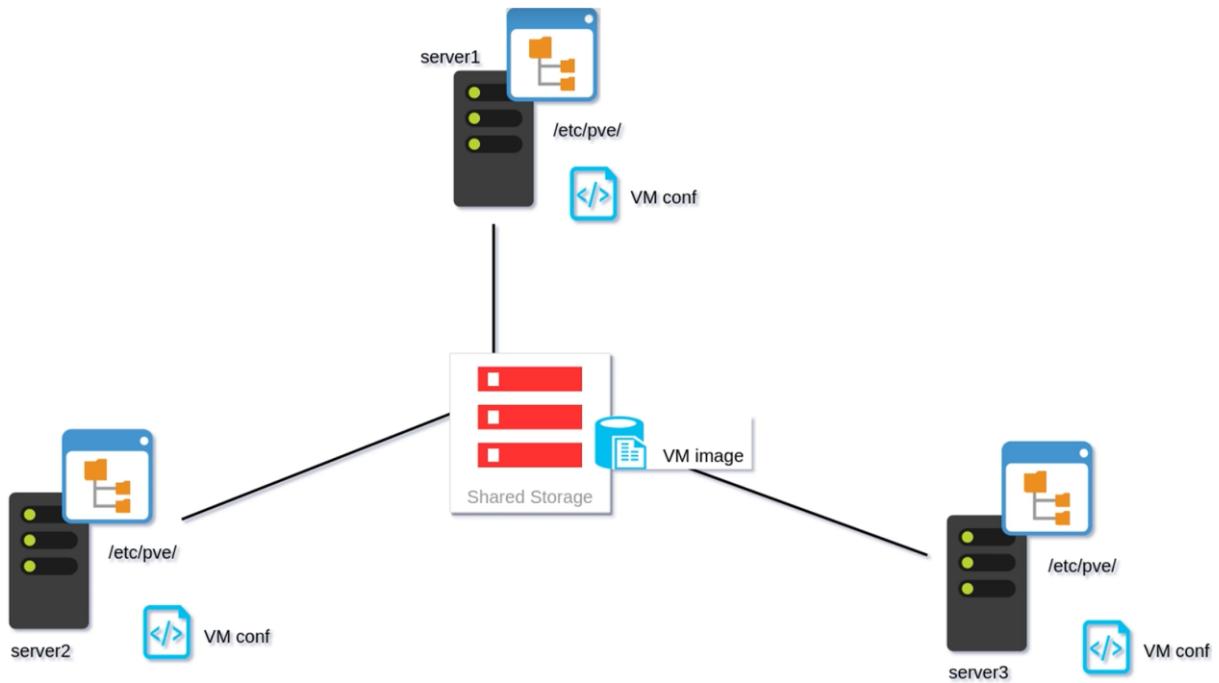
In het geval van Proxmox, waar servers in een groep samenwerken, communiceren ze met elkaar, vertrouwen ze elkaar en slaan ze configuratiebestanden op in een gerepliceerd bestandssysteem.



Dit betekent dat als je een VM aanmaakt op server één, Proxmox een configuratiebestand creëert in het Proxmox Cluster File System (PMXCF). Dit configuratiebestand wordt onmiddellijk gerepliceerd over alle servers in de groep.

Bij migratie is het dus niet nodig om bestanden van de ene server naar de andere te kopiëren, omdat elke server in de groep al de informatie bevat over alle VM's in het cluster.

Als het cluster bovendien VM-images opslaat in gedeelde opslag, zoals SAN, NAS of Ceph, dan heeft elke server in de groep toegang tot de image van de gemigreerde VM.



Zo heeft elke server in de groep toegang tot zowel het configuratiebestand als de VM-image.

- Maak een ssh-verbinding met PVE.

We bekijken de inhoud van /etc/pve.

```
root@pve:~# ls /etc/pve
```

Omdat het een gerepliceerd bestandssysteem is, is de indeling van de bestanden hetzelfde op alle servers in het cluster.

Laten we bijvoorbeeld het configuratiebestand voor opslag lezen.

```
root@pve:~# cat /etc/pve/storage.cfg
```

```
Opdrachtprompt - ssh root@ X Windows PowerShell X Windows PowerShell X + v - □ ×
root@pve:~# cat /etc/pve/storage.cfg
dir: local
    path /var/lib/vz
    content iso,vztmpl,backup,import

lvmthin: local-lvm
    thinpool data
    vgname pve
    content rootdir,images
root@pve:~# |
```

- Maak nu een ssh-verbinding met server2.

Je zal zien dat het configuratiebestand er exact hetzelfde uitziet.

```
root@pve:~# cat /etc/pve/storage.cfg
```

Opdrachtprompt - ssh root@1 | Windows PowerShell | Windows PowerShell

```
root@pve2:~# cat /etc/pve/storage.cfg
dir: local
  path /var/lib/vz
  content iso,vztmpl,backup,import

lvmthin: local-lvm
  thinpool data
  vgname pve
  content rootdir,images
root@pve2:~# |
```

Laten we nu proberen een testbestand te maken op server twee en een bericht naar dit bestand te schrijven.

```
root@pve:~# echo "message from server two" > /etc/pve/test.log
```

Opdrachtprompt - ssh root@1 | Windows PowerShell | Windows PowerShell

```
root@pve2:~# echo "message from server two" > /etc/pve/test.log
root@pve2:~# |
```

- Maak nu een ssh-verbinding met server3.

We lezen nu dit bestand vanaf server drie.

```
root@pve:~# cat /etc/pve/test.log
```

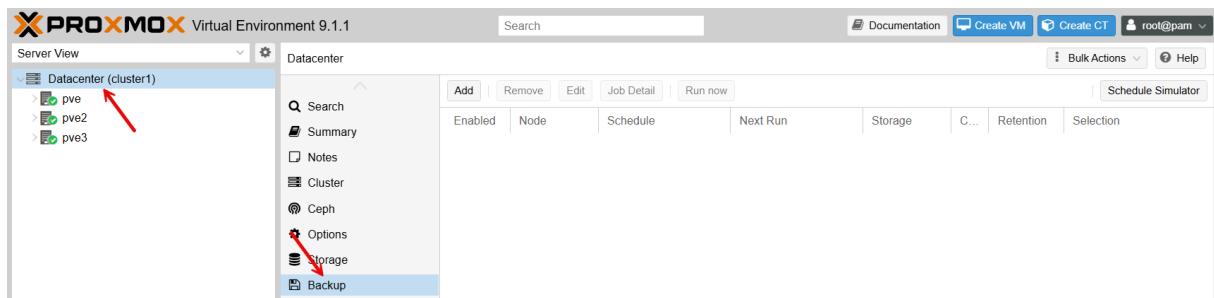
Opdrachtprompt - ssh root@1 | Windows PowerShell | Windows PowerShell

```
root@pve3:~# cat /etc/pve/test.log
message from server two
root@pve3:~# |
```

Zoals je kunt zien, slaat Proxmox configuratiebestanden op in een gerepliceerd bestandssysteem.

16.5 Voorbeeld migratie zonder gedeelde opslag

Bij het opzetten van een Proxmox-cluster kunt u vanaf dit moment de gecentraliseerde webinterface gebruiken om het cluster te beheren. Hiermee kunt u functies zoals geplande back-ups instellen, extra opslag verbinden en zelfs virtuele machines (VM's) migreren tussen servers.

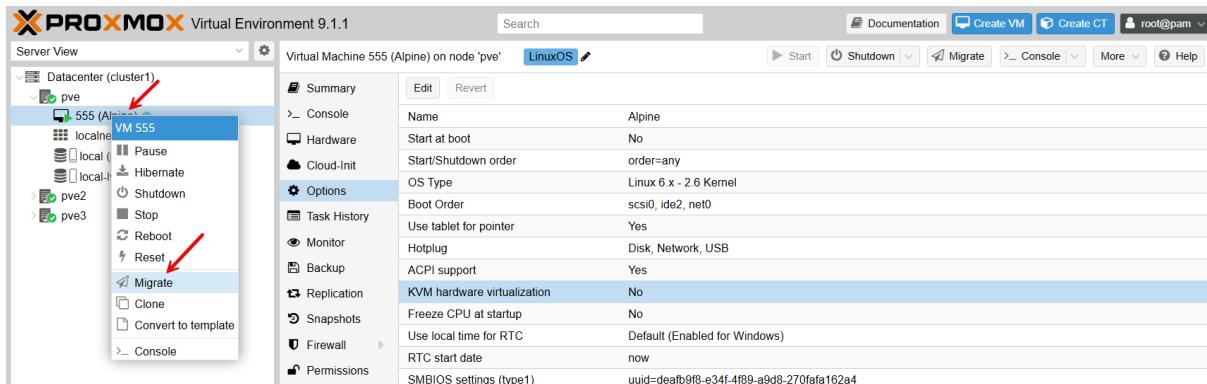


Het is echter op te merken dat migratie op dit moment alleen offline mogelijk is, aangezien het cluster nog geen gecentraliseerde opslag heeft.

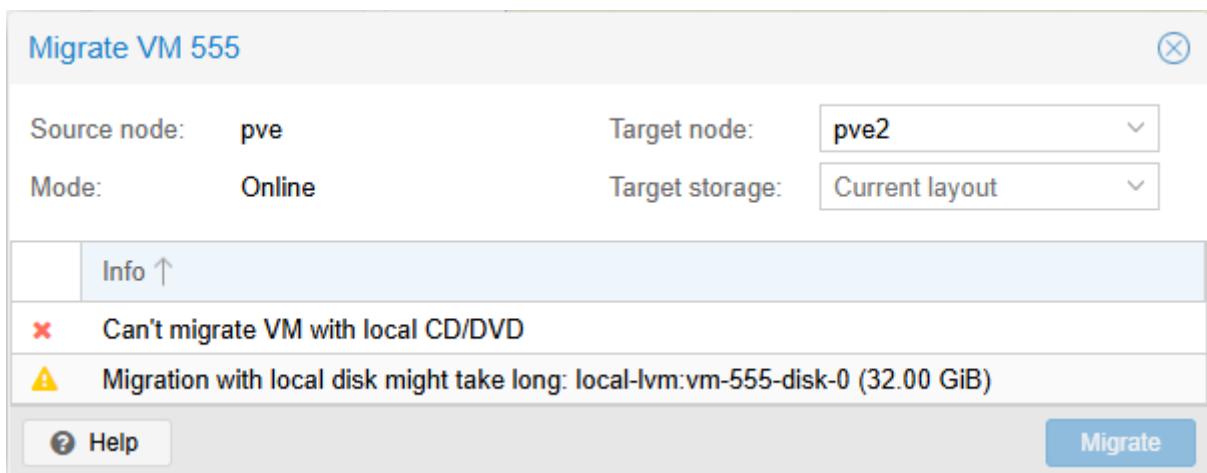
Installeer hiervoor eerst Alpine Linux terug op PVE zoals reeds besproken in het hoofdstuk VM en containers aanmaken.

Zet de VM aan. We verplaatsen nu deze VM van PVE naar PVE2.

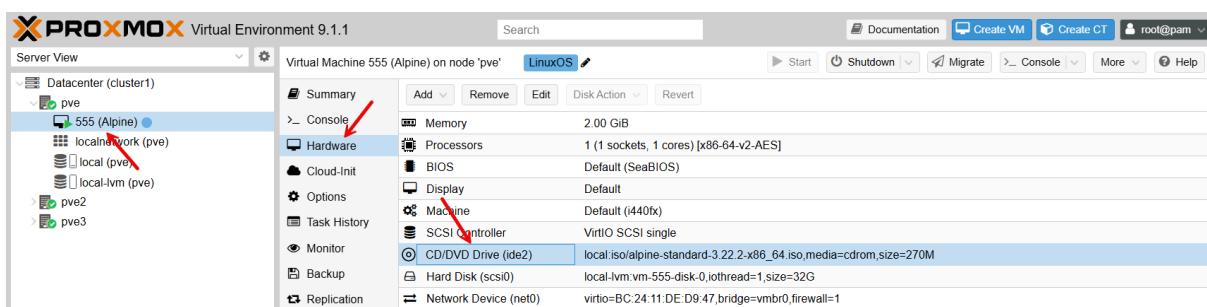
Klik met de rechtermuisknop op de VM en kies voor Migrate.



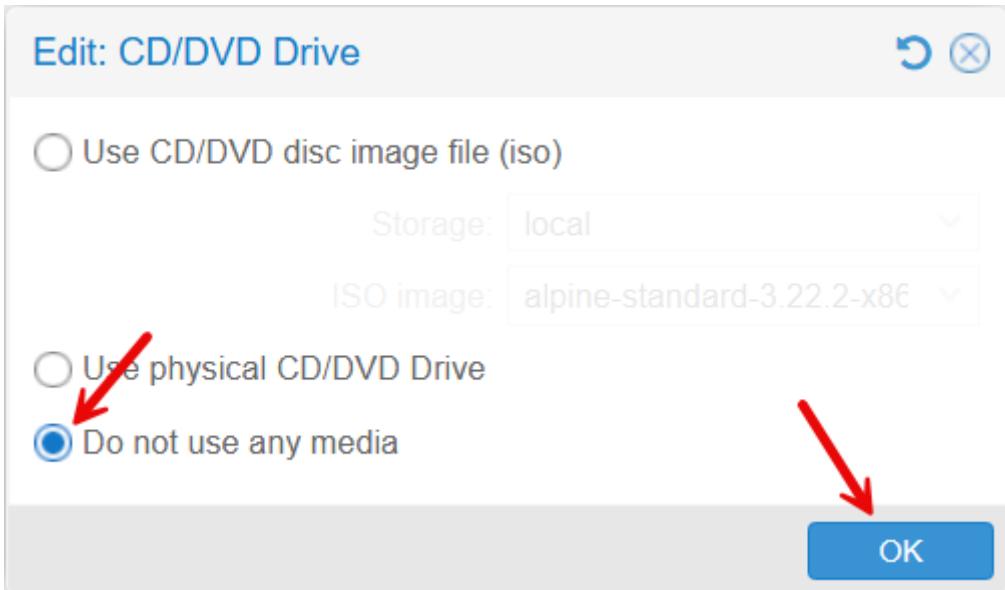
Je krijgt nu de melding dat een migratie met een lokale CD/DVD niet gaat.



We passen daarom de hardware aan voor deze VM. Dubbelklik op CD/DVD Drive bij hardware van de VM.



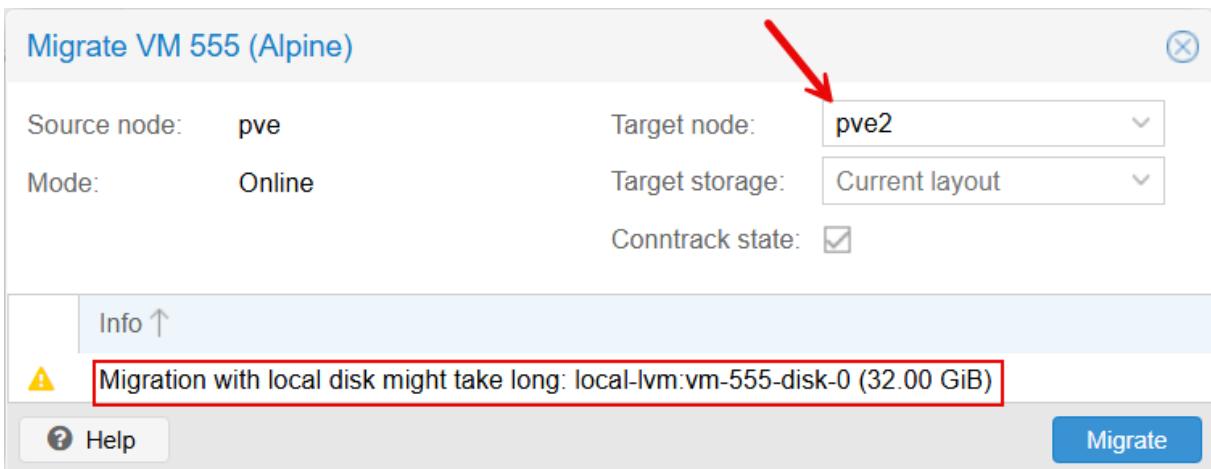
Kies dan voor Do not use any media en erna uiteraard voor OK.



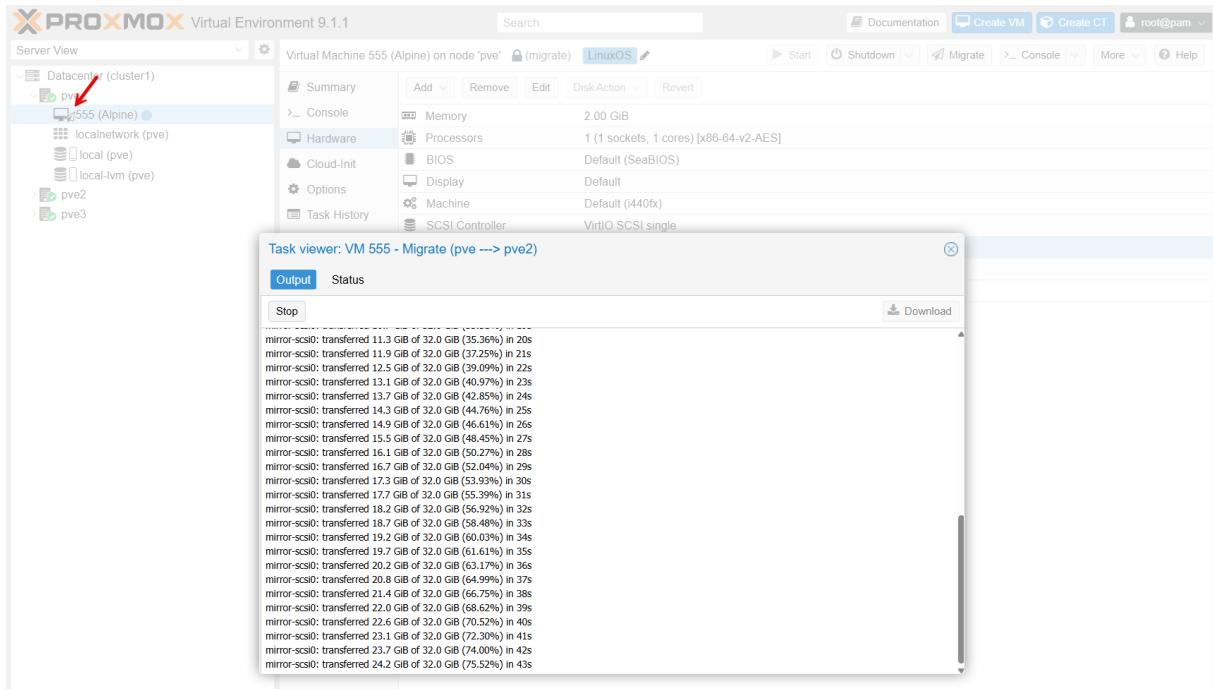
We trachten nu de VM opnieuw te verplaatsen.

Je kan aangeven naar welke node je de VM wenst te verplaatsen. We kiezen voor PVE2.

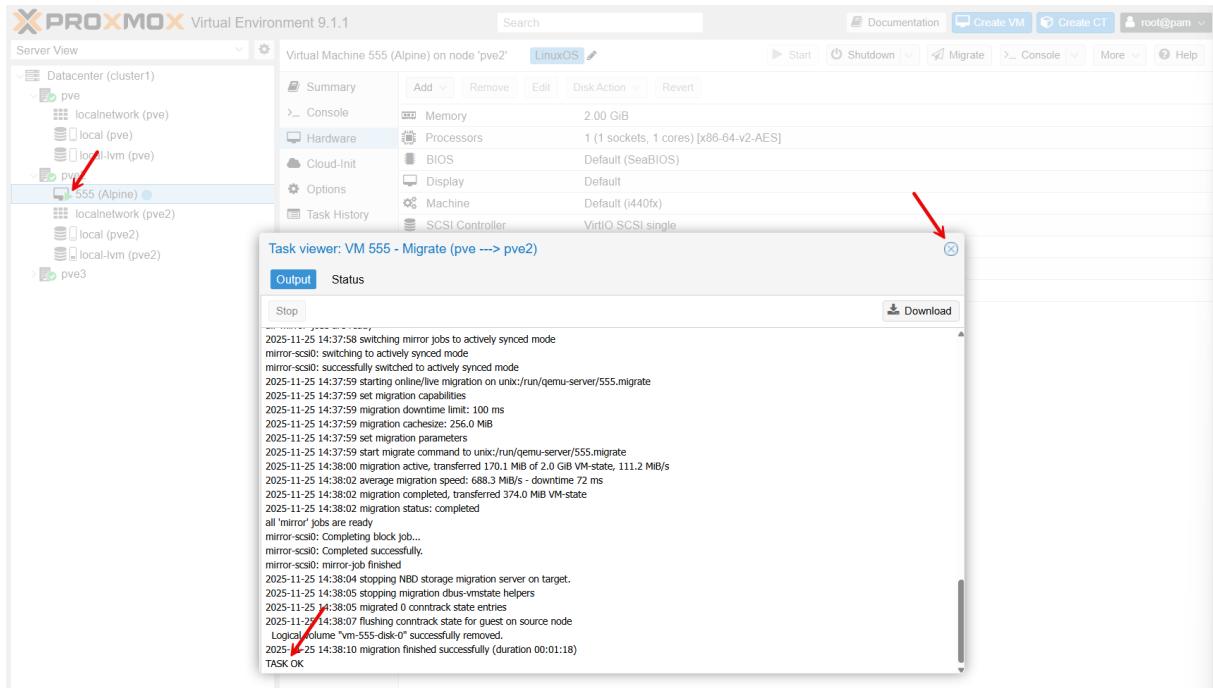
Je krijgt de melding dat het verplaatsen lang duurt. We kiezen toch voor Migrate.



De migratie wordt nu uitgevoerd. Zoals je ziet wordt de data nu getransfereerd naar PVE2.



De machine wordt uitgezet tijdens de verplaatsing (je ziet dat de VM geen groen heeft tijdens verplaatsing). Na verplaatsing zie je dat de VM draait op PVE2.

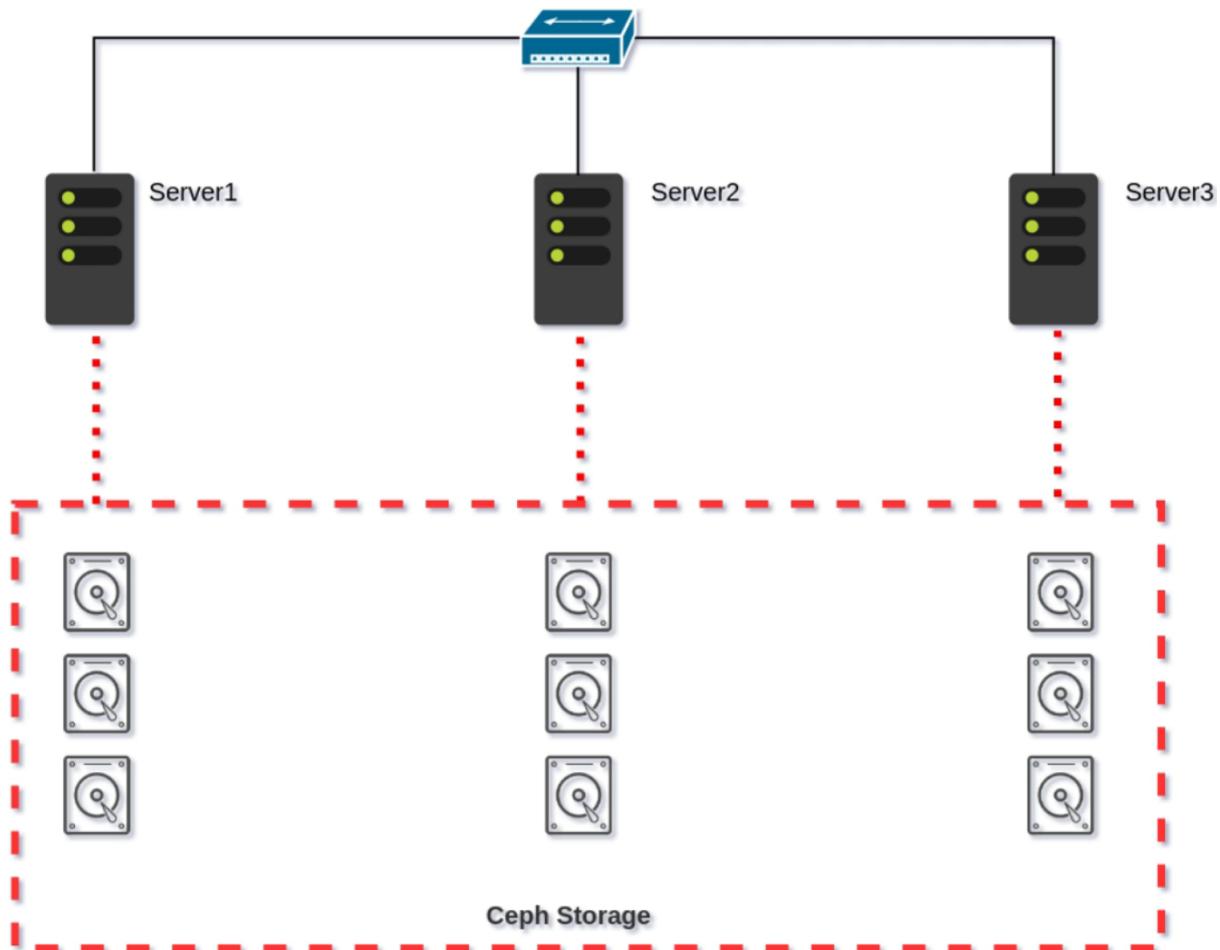


Deze situatie is natuurlijk niet wenselijk in een bedrijf die een website dag en nacht online wil hebben. Men werkt aldus met een gedeelde storage waar alle nodes toegang tot hebben.

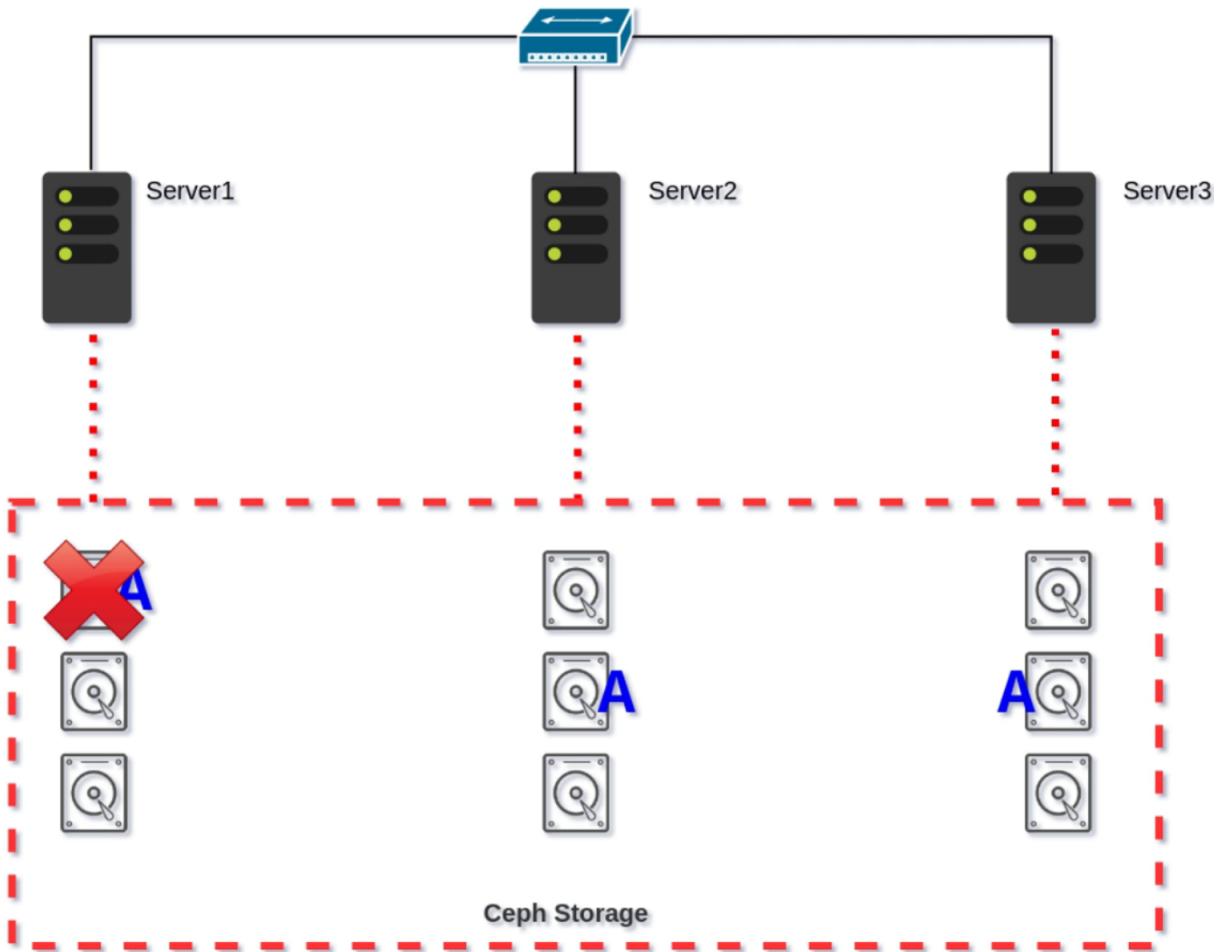
16.6 Ceph

16.6.1 Introductie tot Ceph-opslag

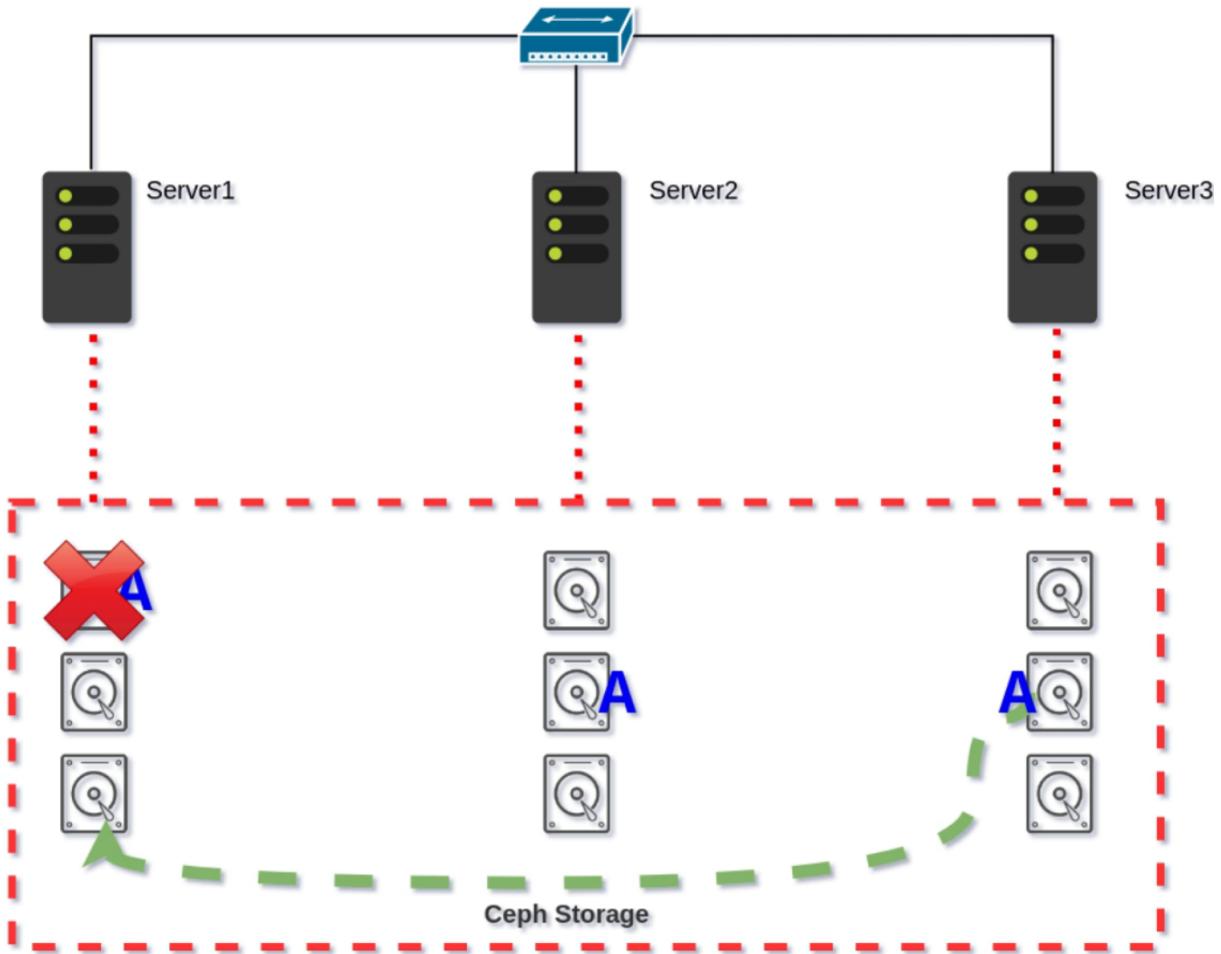
Ceph is een software-gebaseerde, redundante opslagoplossing die is ontworpen voor hoge beschikbaarheid en prestaties. Simpel gezegd, Ceph is een applicatie die harde schijven of lokale opslag van meerdere servers in een cluster samenvoegt tot één groot opslagvolume over het netwerk. Bijvoorbeeld: als u drie servers heeft met elk drie schijven, wordt dit gecombineerd tot één opslagruijte die in totaal negen schijven omvat.



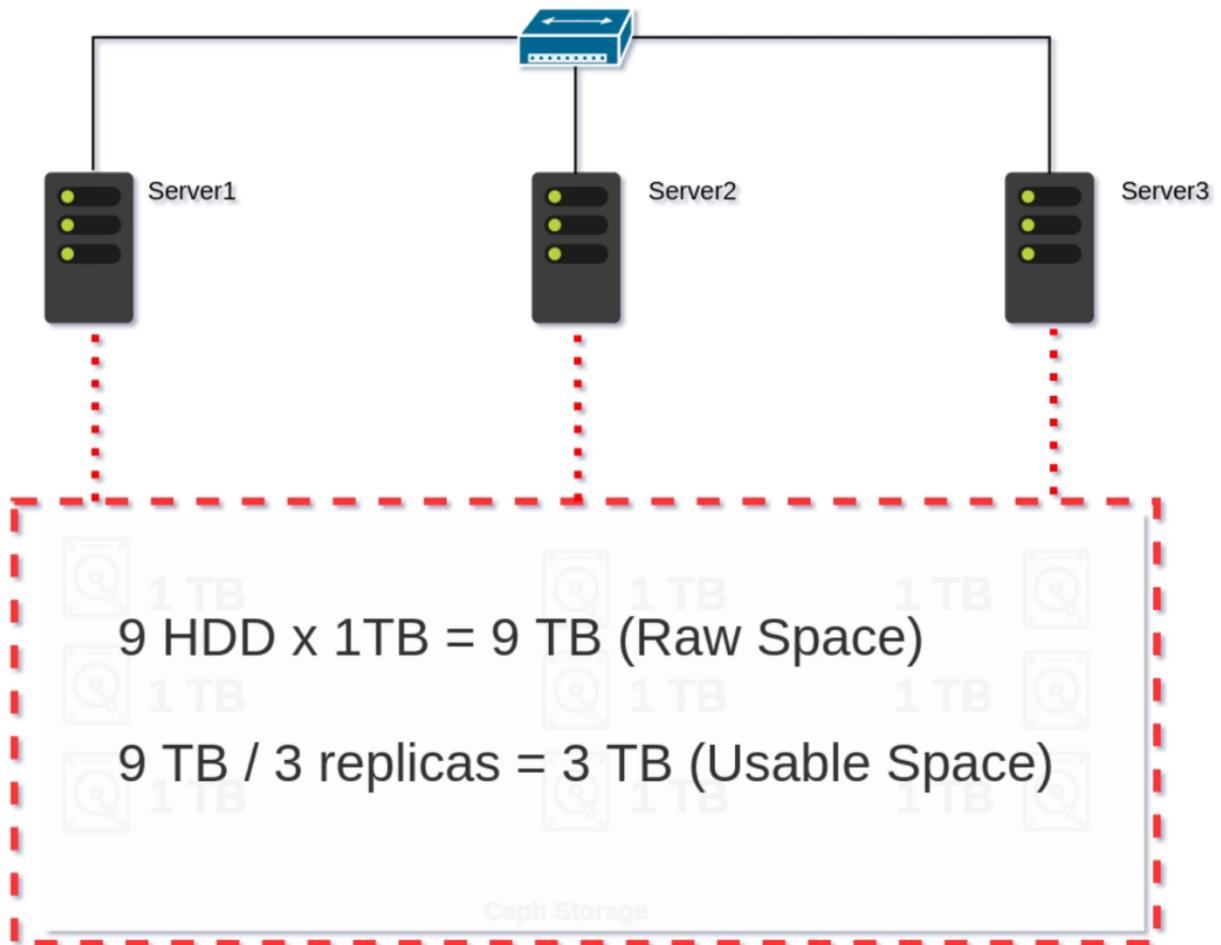
Ceph biedt redundantie en het aantal replica's is aanpasbaar op basis van uw behoeften. Voor productieomgevingen wordt aanbevolen om ten minste drie replica's te configureren. Wanneer een client (zoals een hypervisor in ons geval) gegevens opslaat in de Ceph-opslag, creëert Ceph twee kopieën van dezelfde gegevens op andere servers en schijven. Dit zorgt ervoor dat zelfs als een schijf of een hele server uitvalt, er altijd nog twee kopieën beschikbaar zijn.



Daarnaast is Ceph een zelfherstellend systeem. Als een kopie van de gegevens ontbreekt of inconsistent wordt, zal Ceph automatisch een nieuwe kopie maken om de replicatiefactor van de opslagpool te waarborgen. Hierdoor kan het cluster meerdere schijven of zelfs servers verliezen zonder gegevensverlies.



Uiteraard heeft dat natuurlijk een invloed op de hoeveelheid ruimte die je kan opslaan.

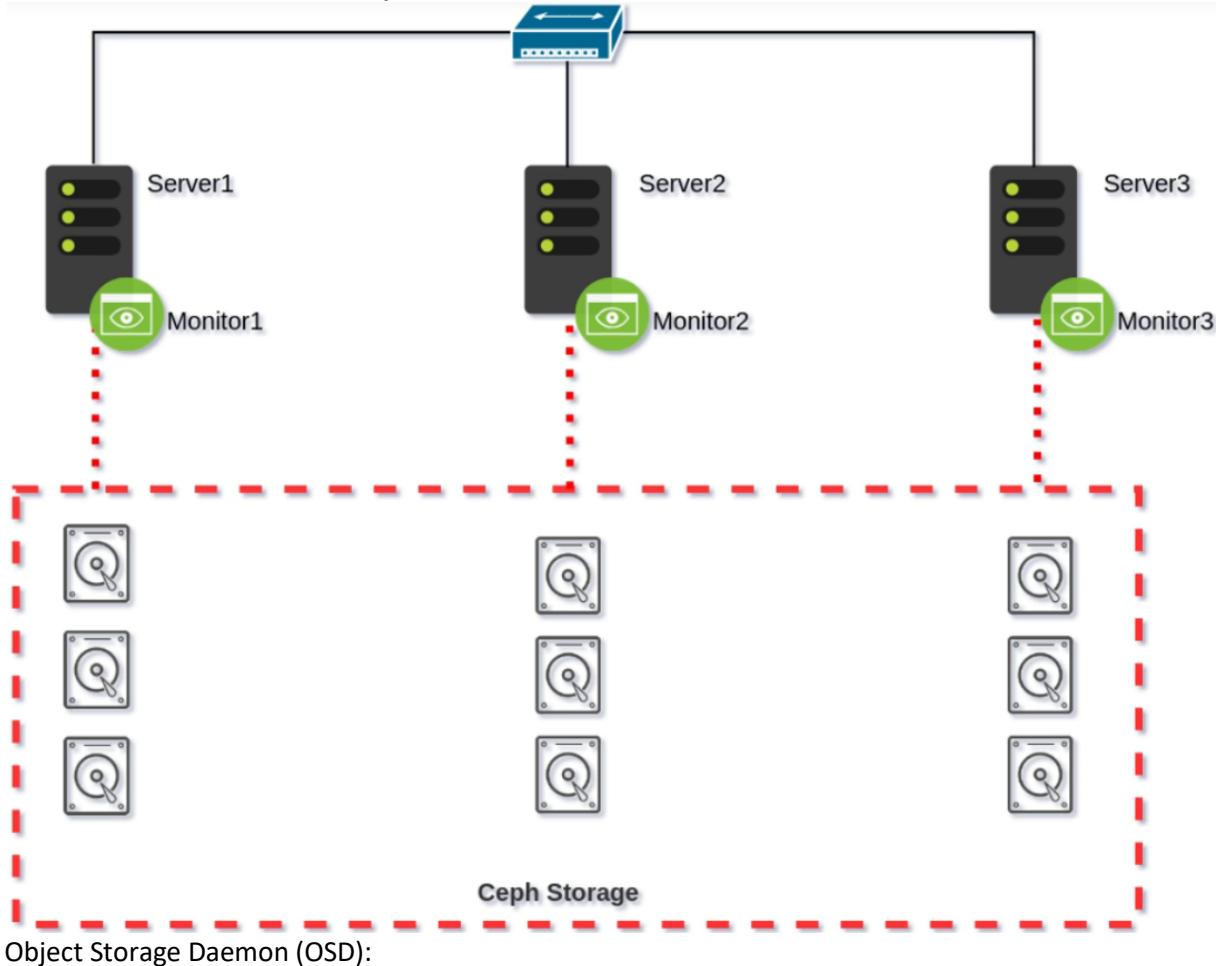


16.6.2 Belangrijke componenten van Ceph

- Monitor (MON):

Een monitor bewaart een masterkopie van de clustermap en houdt de algemene status van het Ceph-cluster bij. Voor productieomgevingen zijn minstens drie monitors vereist, en het

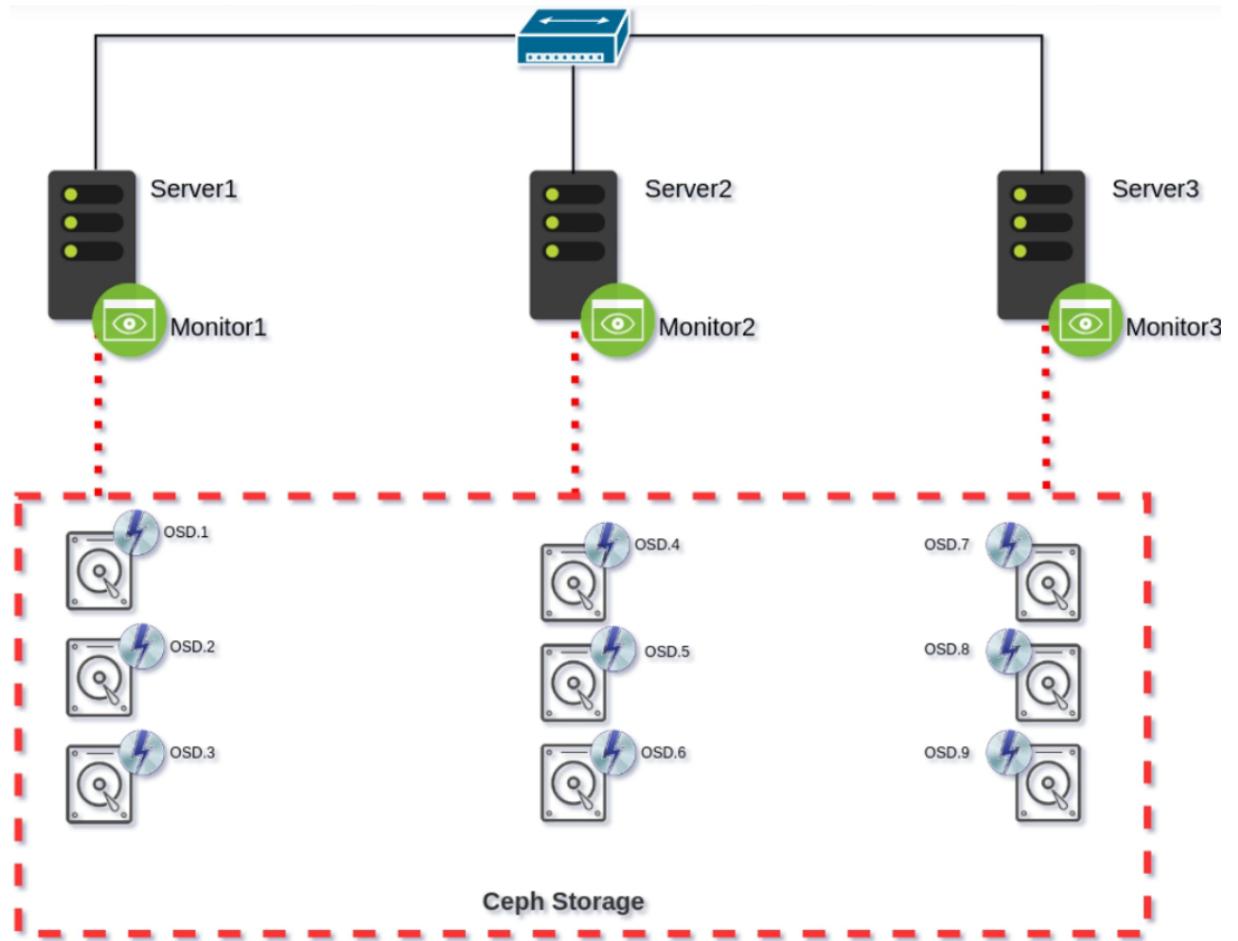
aantal monitors moet oneven zijn.



Object Storage Daemon (OSD):

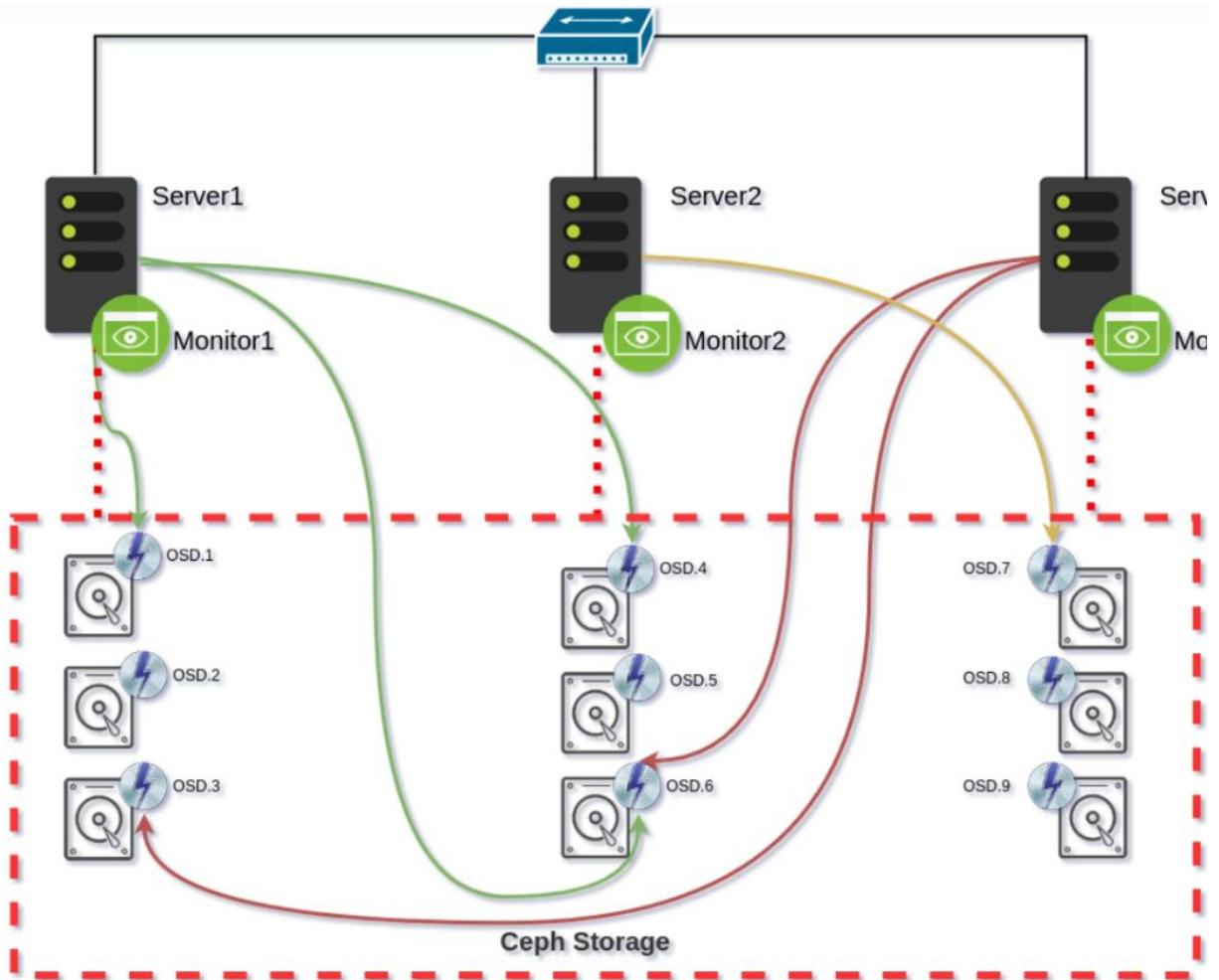
Een OSD is een proces dat communicatie met individuele schijven mogelijk maakt. Voor elke fysieke schijf in het cluster moet een aparte OSD draaien. Als een server bijvoorbeeld drie

schijven heeft, zijn er drie OSD-processen nodig.



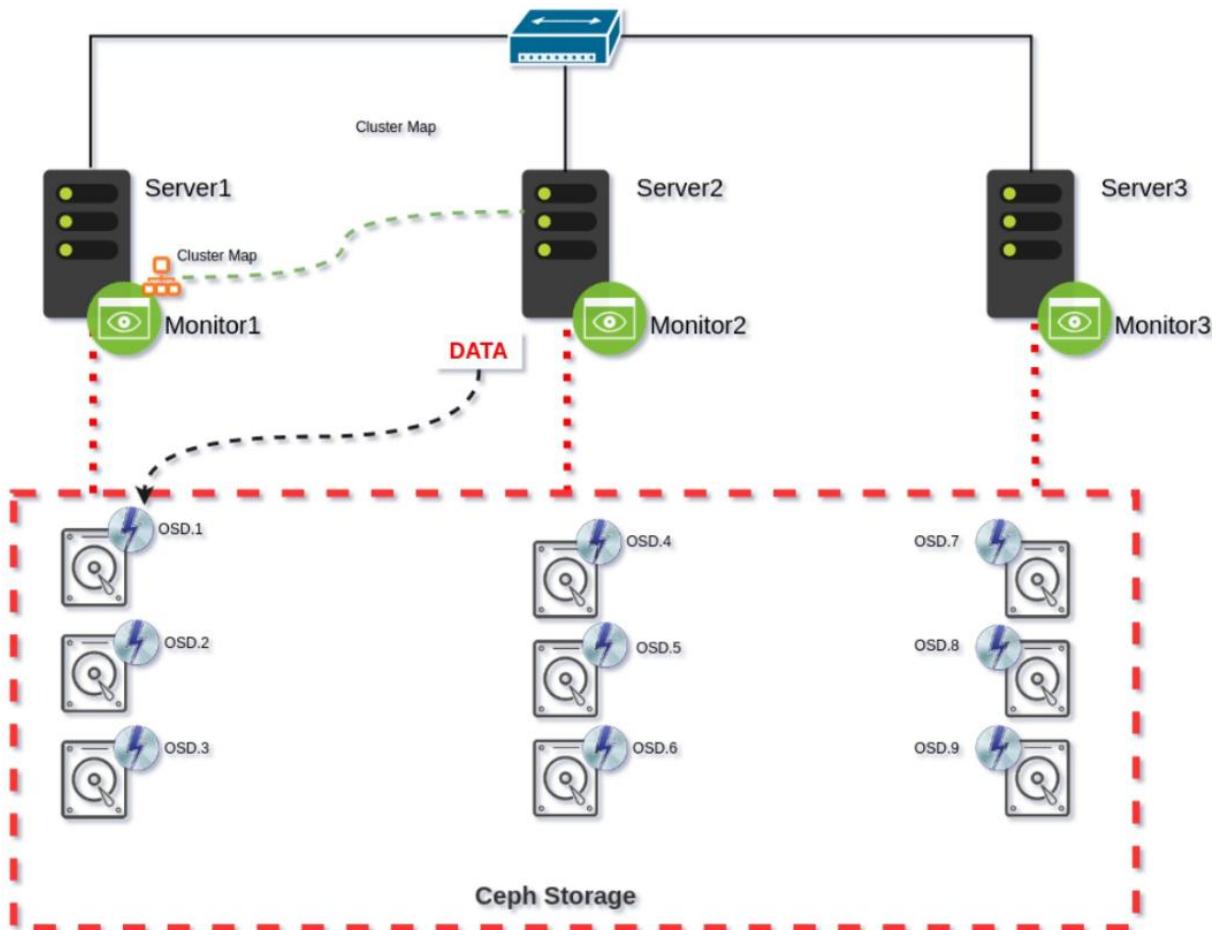
Ceph heeft geen single point of failure. De client communiceert direct met alle schijven in het cluster zonder tussenliggende gateways of bottlenecks. Dit zorgt voor een robuuste en schaalbare

opslagoplossing.



Dus als een client gegevens in een cluster wil lezen of schrijven, zijn er twee stappen nodig:

- Stap één: De client moet beschikken over de nieuwste clustermap, aangezien de clustermap informatie bevat over alle OSD's in het cluster.
- Stap twee: Aan de hand van de informatie uit de clustermap past de client het Crush-algoritme toe om te bepalen met welke OSD het moet communiceren om een specifiek object te lezen of te schrijven.



Samenvatting:

- Ceph is een softwaregedefinieerde opslagoplossing (toepassing).
- Het biedt redundantie en is zelfherstellend
- Er zijn geen SPOF (Single Point of Failure).
- Voor het uitvoeren van de clusterbeheerder is minimaal één monitor nodig om de clustermap en -status bij te houden.
- OSD (Object Storage Daemon) maakt directe communicatie met een opslagstation via het netwerk mogelijk, waarbij clients gebruikmaken van het Crush-algoritme om rechtstreeks met elke OSD te communiceren.

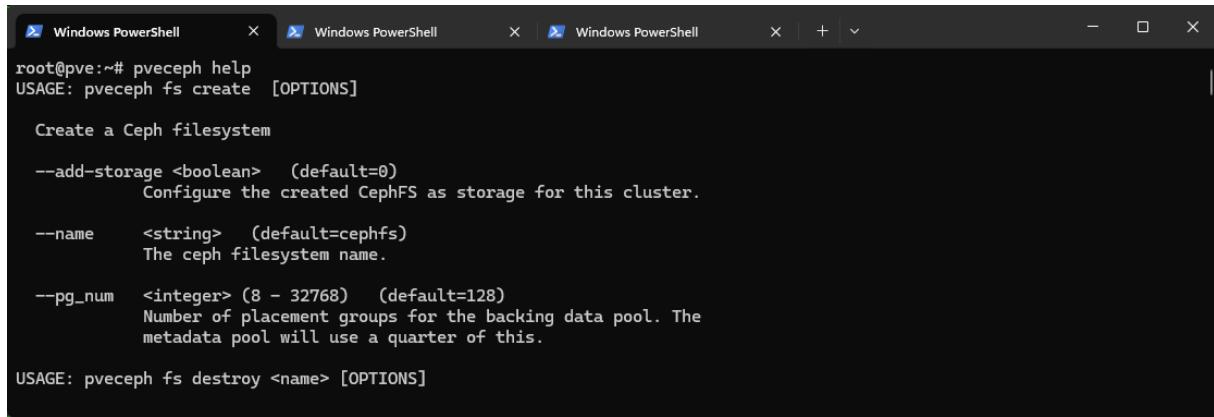
16.6.3 Installatie

16.6.3.1 Vooraf

Hoewel de installatieprocedure wellicht technisch lijkt, is deze rechtlijnig en eenvoudig uit te voeren.

De implementatietool van Proxmox voor ceph is genaamd pveceph.

`pveceph help`



```
root@pve:~# pveceph help
USAGE: pveceph fs create [OPTIONS]

Create a Ceph filesystem

--add-storage <boolean> (default=0)
    Configure the created CephFS as storage for this cluster.

--name <string> (default=cephfs)
    The ceph filesystem name.

--pg_num <integer> (8 - 32768) (default=128)
    Number of placement groups for the backing data pool. The
    metadata pool will use a quarter of this.

USAGE: pveceph fs destroy <name> [OPTIONS]
```

Het is echter belangrijk dat de voorbereiding van het besturingssysteem correct is uitgevoerd. Dit omvat:

- Correct geconfigureerde hostbestanden.
- Geconfigureerde repositories.
- De servers moeten zich in hetzelfde subnet bevinden.
- De tijdservers moeten goed zijn ingesteld.

Wij hebben dit reeds ingesteld 😊.

Voer onderstaande uit om nog eens te synchroniseren met de tijdservers.

```
root@pve:~# chronyc makestep
```

```
200 OK
```

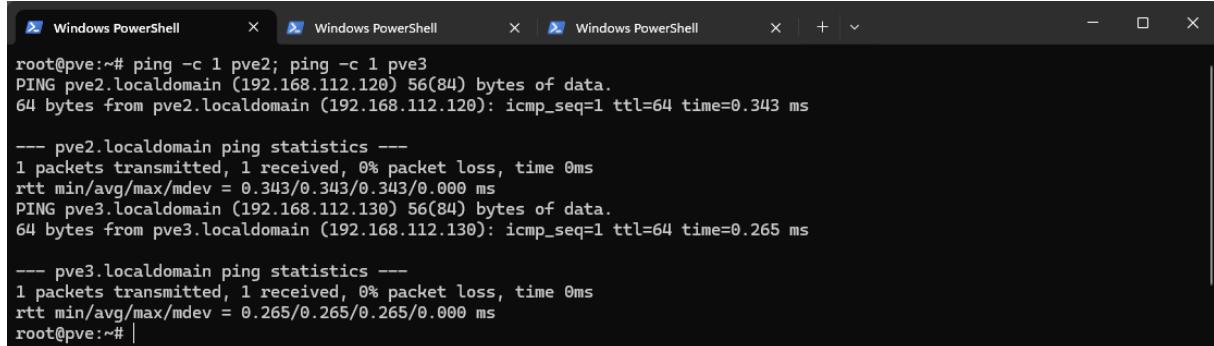
Check de tijd.

```
root@pve:~# date
```

```
Tue Nov 25 03:36:22 PM CET 2025
```

Check dat de servers naar elkaar kunnen pingen. Voer uit op pve.

```
root@pve:~# ping -c 1 pve2; ping -c 1 pve3
```



```
root@pve:~# ping -c 1 pve2; ping -c 1 pve3
PING pve2.localdomain (192.168.112.120) 56(84) bytes of data.
64 bytes from pve2.localdomain (192.168.112.120): icmp_seq=1 ttl=64 time=0.343 ms

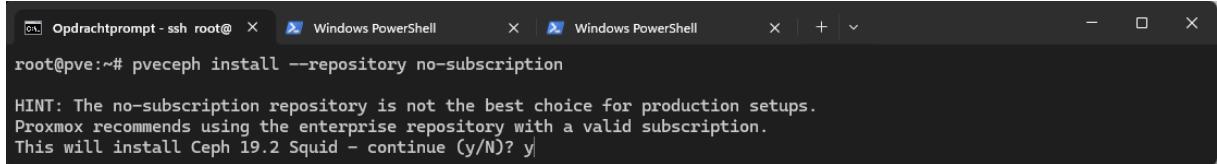
--- pve2.localdomain ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.343/0.343/0.343/0.000 ms
PING pve3.localdomain (192.168.112.130) 56(84) bytes of data.
64 bytes from pve3.localdomain (192.168.112.130): icmp_seq=1 ttl=64 time=0.265 ms

--- pve3.localdomain ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.265/0.265/0.265/0.000 ms
root@pve:~# |
```

Voer deze test ook uit op de 2 andere servers.

Gebruik pveceph om de software te installeren. Doe dit op alle servers.

```
root@pve:~# pveceph install --repository no-subscription
```



```
Opdrachtprompt - ssh root@ X Windows PowerShell X | Windows PowerShell X + - X
root@pve:~# pveceph install --repository no-subscription
HINT: The no-subscription repository is not the best choice for production setups.
Proxmox recommends using the enterprise repository with a valid subscription.
This will install Ceph 19.2 Squid - continue (y/N)? y|
```

Voer nog een keer Y uit.

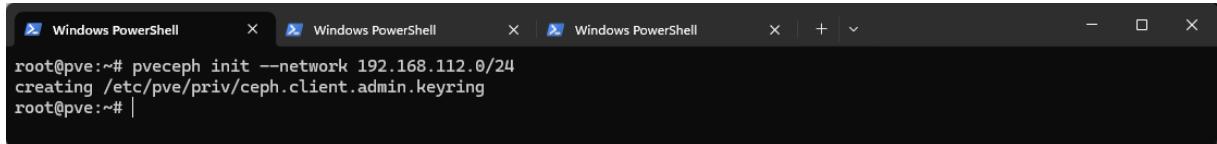
Na enige tijd is Ceph geïnstalleerd.

Nogmaals: voer dit uit op alle servers in het cluster.

16.6.3.2 Initialisatie van configuratie

Genereer het initiële configuratiebestand op slechts één server met het commando:

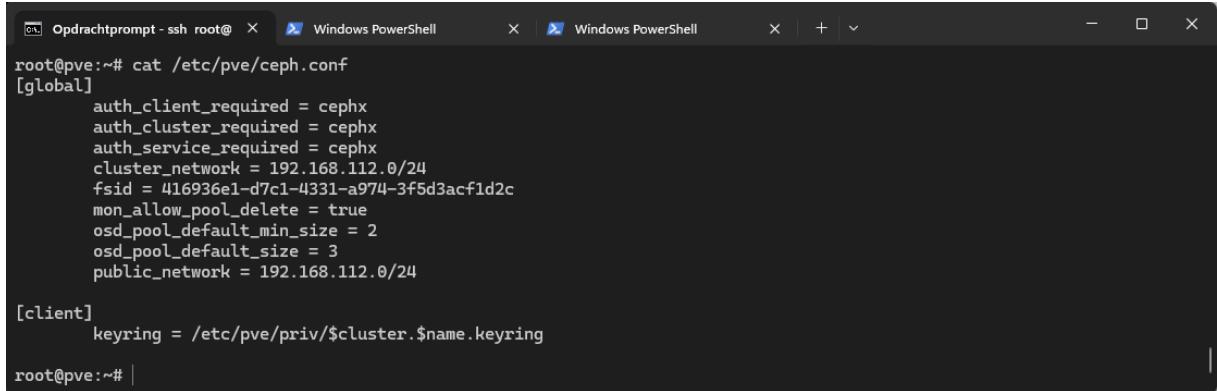
```
root@pve:~# pveceph init --network 192.168.112.0/24
```



```
Windows PowerShell X | Windows PowerShell X | Windows PowerShell X + - X
root@pve:~# pveceph init --network 192.168.112.0/24
creating /etc/pve/priv/ceph.client.admin.keyring
root@pve:~# |
```

Bij het uitvoeren van dit commando wordt /etc/pve/ceph.conf aangemaakt.

```
root@pve:~# cat /etc/pve/ceph.conf
```

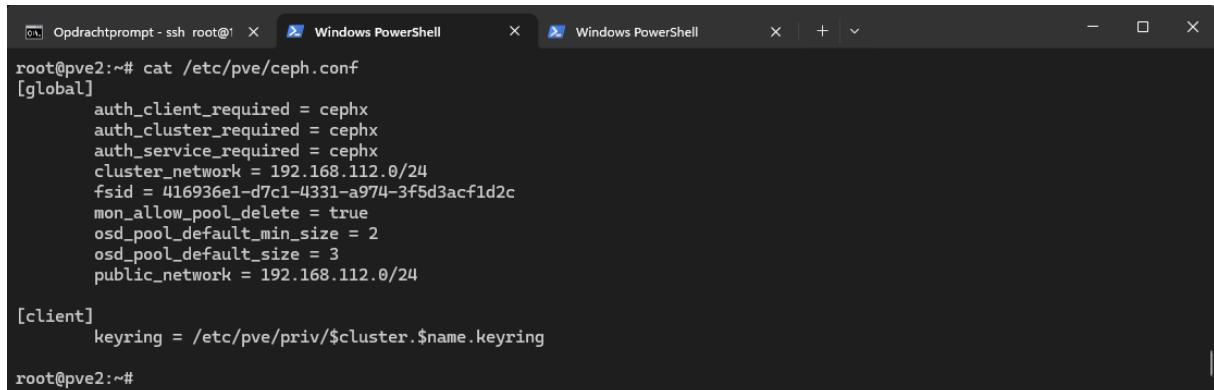


```
Opdrachtprompt - ssh root@ X Windows PowerShell X | Windows PowerShell X + - X
root@pve:~# cat /etc/pve/ceph.conf
[global]
auth_client_required = cephx
auth_cluster_required = cephx
auth_service_required = cephx
cluster_network = 192.168.112.0/24
fsid = 416936e1-d7c1-4331-a974-3f5d3acf1d2c
mon_allow_pool_delete = true
osd_pool_default_min_size = 2
osd_pool_default_size = 3
public_network = 192.168.112.0/24

[client]
keyring = /etc/pve/priv/$cluster.$name.keyring
root@pve:~# |
```

Dit bestand wordt automatisch gerepliceerd naar alle servers in het cluster. Voor onderstaande uit op een andere server.

```
root@pve2:~# cat /etc/pve/ceph.conf
```



```
root@pve2:~# cat /etc/pve/ceph.conf
[global]
    auth_client_required = cephx
    auth_cluster_required = cephx
    auth_service_required = cephx
    cluster_network = 192.168.112.0/24
    fsid = 416936e1-d7c1-4331-a974-3f5d3acf1d2c
    mon_allow_pool_delete = true
    osd_pool_default_min_size = 2
    osd_pool_default_size = 3
    public_network = 192.168.112.0/24

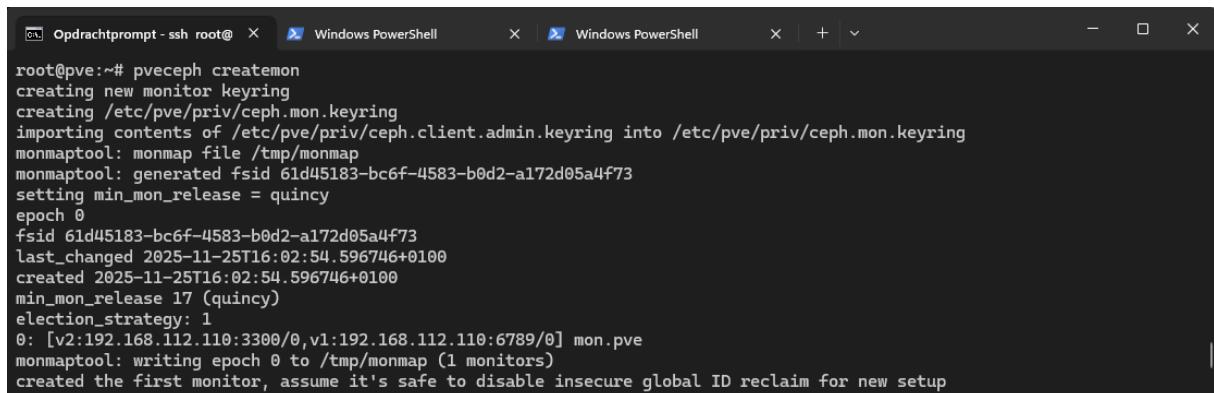
[client]
    keyring = /etc/pve/priv/$cluster.$name.keyring

root@pve2:~#
```

16.6.3.3 Creëren van monitors

Maak een monitor op PVE:

```
root@pve:~# pveceph createmon
```

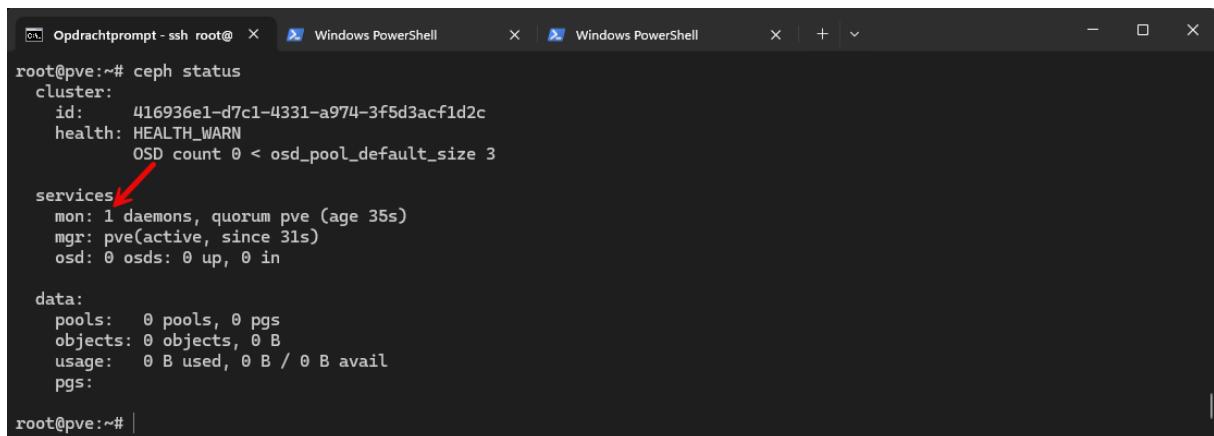


```
root@pve:~# pveceph createmon
creating new monitor keyring
creating /etc/pve/priv/ceph.mon.keyring
importing contents of /etc/pve/priv/ceph.client.admin.keyring into /etc/pve/priv/ceph.mon.keyring
monmappool: monmap file /tmp/monmap
monmappool: generated fsid 61d45183-bc6f-4583-b0d2-a172d05a4f73
setting min_mon_release = quincy
epoch 0
fsid 61d45183-bc6f-4583-b0d2-a172d05a4f73
last_changed 2025-11-25T16:02:54.596746+0100
created 2025-11-25T16:02:54.596746+0100
min_mon_release 17 (quincy)
election_strategy: 1
0: [v2:192.168.112.110:3300/0,v1:192.168.112.110:6789/0] mon.pve
monmappool: writing epoch 0 to /tmp/monmap (1 monitors)
created the first monitor, assume it's safe to disable insecure global ID reclaim for new setup
```

Met 1 monitor kunnen we interageren met de array.

We voeren nu volgende uit:

```
root@pve:~# ceph status
```



```
root@pve:~# ceph status
cluster:
  id:      416936e1-d7c1-4331-a974-3f5d3acf1d2c
  health:  HEALTH_WARN
          OSD count 0 < osd_pool_default_size 3

  services
    mon: 1 daemons, quorum pve (age 35s)
    mgr: pve(active, since 31s)
    osd: 0 osds: 0 up, 0 in

  data:
    pools:  0 pools, 0 pgs
    objects: 0 objects, 0 B
    usage:   0 B used, 0 B / 0 B avail
    pgs:

root@pve:~#
```

Ceph heeft drie monitoren nodig voor een high availability quorum.

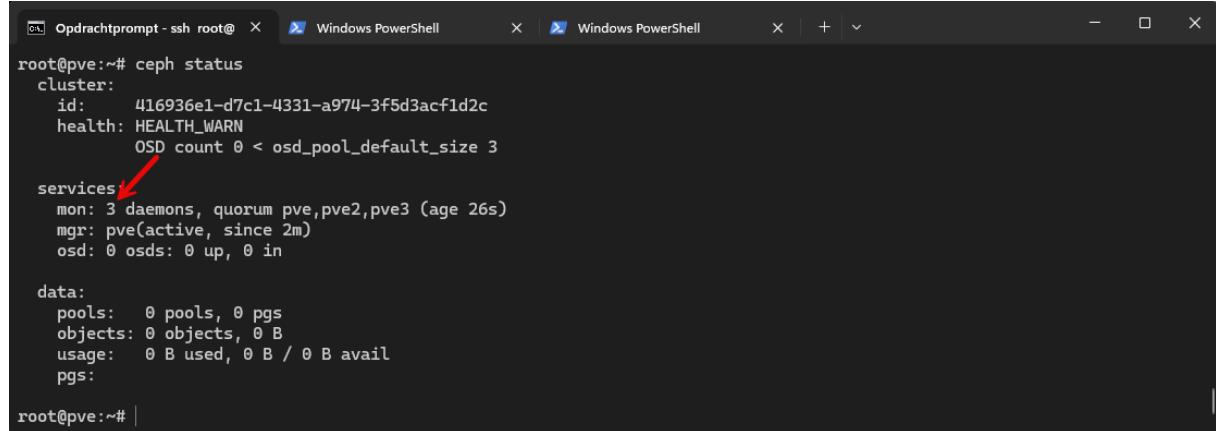
We maken dus ook op PVE2 en PVE3 een monitor aan.

```
root@pve2:~# pveceph createmon
```

```
root@pve3:~# pveceph createmon
```

Daarna voeren we terug ceph status uit.

```
root@pve:~# ceph status
```



```
Opdrachtprompt - ssh root@ X Windows PowerShell X | Windows PowerShell X - + - 
root@pve:~# ceph status
cluster:
  id: 416936e1-d7c1-4331-a974-3f5d3acf1d2c
  health: HEALTH_WARN
    OSD count 0 < osd_pool_default_size 3

  services
    mon: 3 daemons, quorum pve,pve2,pve3 (age 26s)
      mgr: pve(active, since 2m)
      osd: 0 osds: 0 up, 0 in

  data:
    pools: 0 pools, 0 pgs
    objects: 0 objects, 0 B
    usage: 0 B used, 0 B / 0 B avail
    pgs:
```

16.6.3.4 Aanmaken van Object Storage Daemons

In het geval van Ceph is een OSD een proces dat verantwoordelijk is voor het beheren van een fysieke schijf (of een deel ervan) in het cluster. Elke OSD zorgt voor:

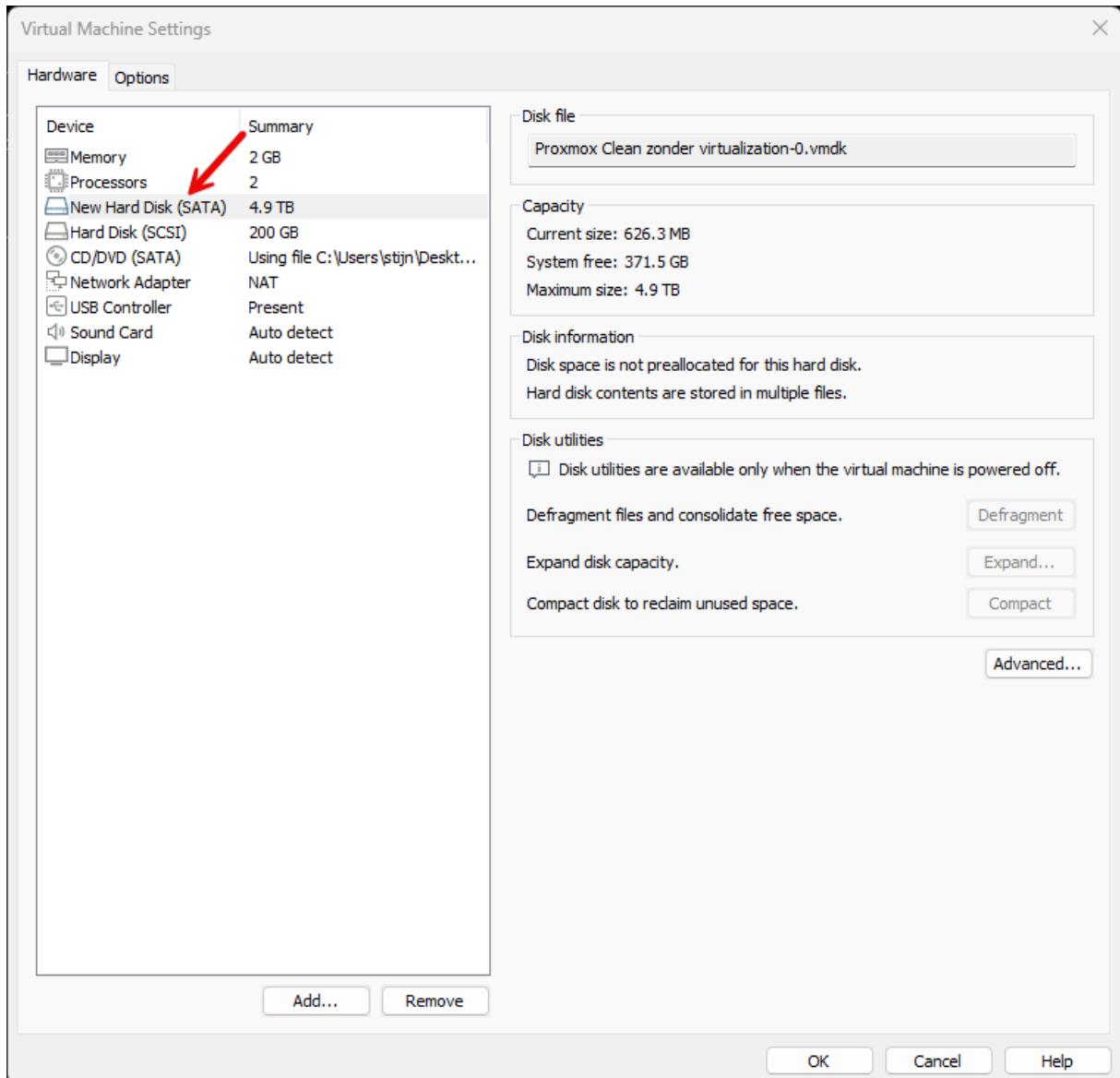
- Opslaan van objecten: Het slaat gegevens op in de vorm van objecten op de toegewezen schijf.
- Replicatie: Het zorgt ervoor dat de gegevens worden gerepliceerd naar andere OSD's volgens de ingestelde replicatiefactor.
- Communicatie: Het communiceert met andere OSD's en clients (zoals monitors of hypervisors) binnen het cluster.
- Zelfherstel: In het geval van een storing zorgt de OSD voor het herstellen van replicaties om de consistentie in het cluster te waarborgen.

Je hebt per schijf een OSD nodig.

Ceph ondersteunt een breed scala aan schijfcontrollers, omdat het ontworpen is om op standaard hardware te draaien. Hier is een overzicht van de meest gebruikte controllers en hun ondersteuning binnen Ceph:

- SATA
- SCSI
- SAS
- NVMe
- RAID
- USB

We zullen op elke node een SATA-schijf toevoegen van 5000 GB. Deze worden immers onmiddellijk herkend.



Na installatie checken we of de schijf herkend is.

```
root@pve:~# lsblk
```

```

root@pve:~# lsblk
NAME      MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
sda        8:0    0  200G  0 disk
└─sda1     8:1    0 100T  0 part
└─sda2     8:2    0   1G  0 part
└─sda3     8:3    0 199G  0 part
  ├─pve-swap 252:0  0   4G  0 lvm  [SWAP]
  ├─pve-root  252:1  0 60.7G 0 lvm /
  ├─pve-data_tmeta 252:2  0  1.2G 0 lvm
  └─pve-data_tpool 252:4  0 115.9G 0 lvm
    ├─pve-data 252:5  0 115.9G 1 lvm
    └─pve-data_tdata 252:3  0 115.9G 0 lvm
      └─pve-data_tpool 252:4  0 115.9G 0 lvm
        └─pve-data 252:5  0 115.9G 1 lvm
sdb        8:16   0  4.9T  0 disk
sr0       11:0   1  1.7G  0 rom
root@pve:~#

```

Maak een keyring aan op elke server.

```

root@pve:~# ceph auth get-or-create client.bootstrap-osd mon
'allow profile bootstrap-osd' > /var/lib/ceph/bootstrap-
osd/ceph.keyring

root@pve2:~# ceph auth get-or-create client.bootstrap-osd mon
'allow profile bootstrap-osd' > /var/lib/ceph/bootstrap-
osd/ceph.keyring

root@pve3:~# ceph auth get-or-create client.bootstrap-osd mon
'allow profile bootstrap-osd' > /var/lib/ceph/bootstrap-
osd/ceph.keyring

```

Een keyring in Ceph is een bestand dat beveiligingssleutels bevat waarmee een node of een service zich kan authentiseren bij de Ceph-cluster.

We maken nu een ceph-volume aan. Voer dat uiteraard uit op de 3 servers.

```
root@pve:~# ceph-volume lvm create --data /dev/sdb
```

We voeren nu onderstaand commando uit op de 3 servers om een OSD aan te maken voor /dev/sdb.

```

root@pve:~# pveceph createosd /dev/sdb
root@pve2:~# pveceph createosd /dev/sdb
root@pve3:~# pveceph createosd /dev/sdb

```

```

root@pve:~# pveceph createosd /dev/sdb
create OSD on /dev/sdb (bluestore)
wiping block device /dev/sdb
200+0 records in
200+0 records out

service.
Running command: /bin/systemctl start ceph-osd@0
--> ceph-volume lvm activate successful for osd ID: 0
--> ceph-volume lvm create successful for: /dev/sdb
root@pve:~#

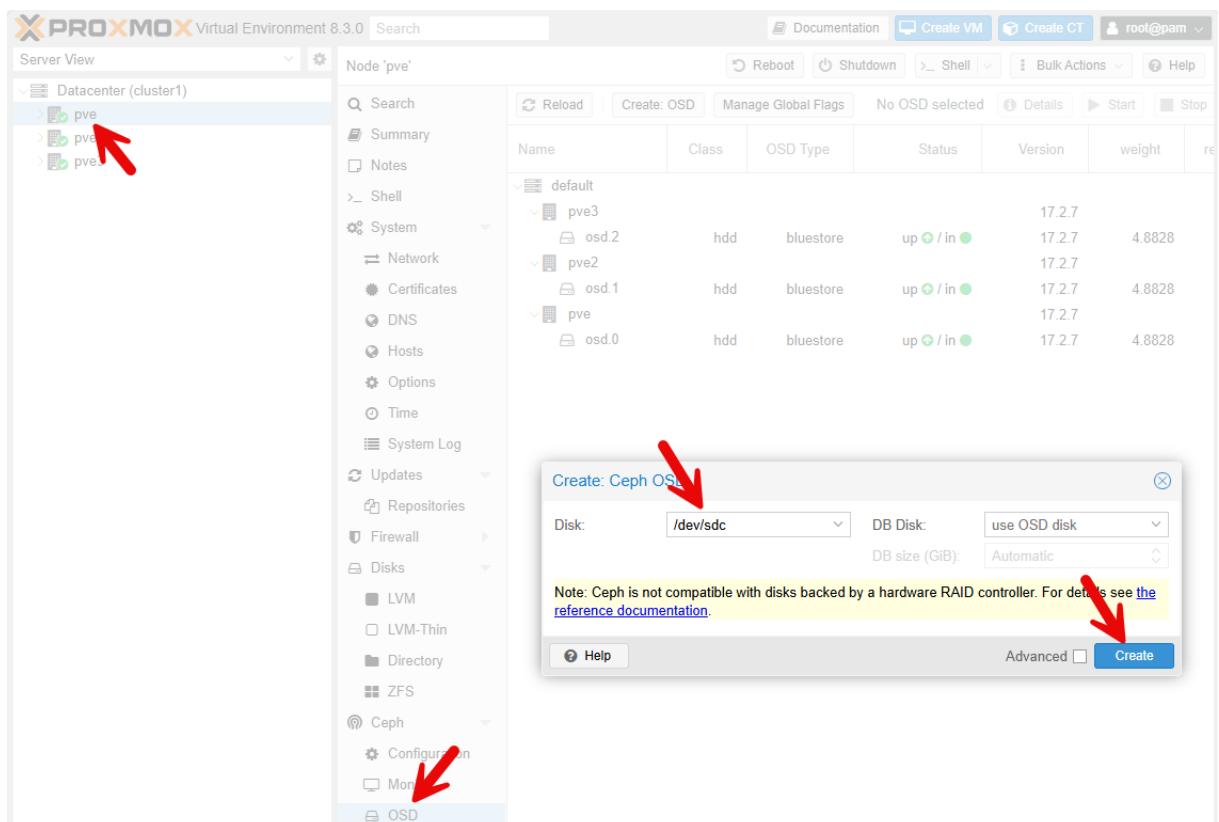
```

Opgelet: de schijf moet leeg zijn voordat je een OSD kan aanmaken op een schijf. Als dat niet zo is moeten alle bestaande gegevens, partitietabellen en metadata worden gewist.

- Eerste gedeelde schijf wissen met dd
`dd if=/dev/zero of=/dev/sdb bs=1M count=200`
- Wis nu de bestandssysteemhandtekeningen (in “superblocken”)
`wipefs --all /dev/sdb`

PS Je kan uiteraard ook in de GUI ook OSD's toevoegen. Hieronder staat aangegeven hoe je dat zou kunnen doen (we stellen dit nu wel niet in).

De OSD's voeg je via de GUI toe door de server te selecteren en te kiezen voor OSD en dan te kiezen voor Create: OSD.



De Ceph Manager daemon is verantwoordelijk voor het beheren en monitoren van het Ceph-cluster.

In de grafische omgeving zal je ook zien dat er nu 3 monitors zijn (1 actie en 2 standby).

Name	Host	Status	Address	Version	Quorum
mon.pve	pve	running	192.168.112.110:6789/0	19.2.3	Yes
mon.pve2	pve2	running	192.168.112.120:6789/0	19.2.3	Yes
mon.pve3	pve3	running	192.168.112.130:6789/0	19.2.3	Yes

Name	Host	Status	Address	Version
mgr.pve	pve	active	192.168.112.110	19.2.3

16.6.3.5 Managers

Er is één manager: PVE. In Ceph kan er slechts één actieve manager tegelijk zijn om te voorkomen dat meerdere processen elkaar overlappen. Dit wordt de "Active Manager" genoemd. Eventuele andere geïnstalleerde managers blijven in standby-modus. Als de actieve manager uitvalt, neemt een van de standby-managers de rol over.

Daarom zullen we ook van PVE2 en PVE3 een manager maken. Voer onderstaande dan ook uit op de twee andere servers.

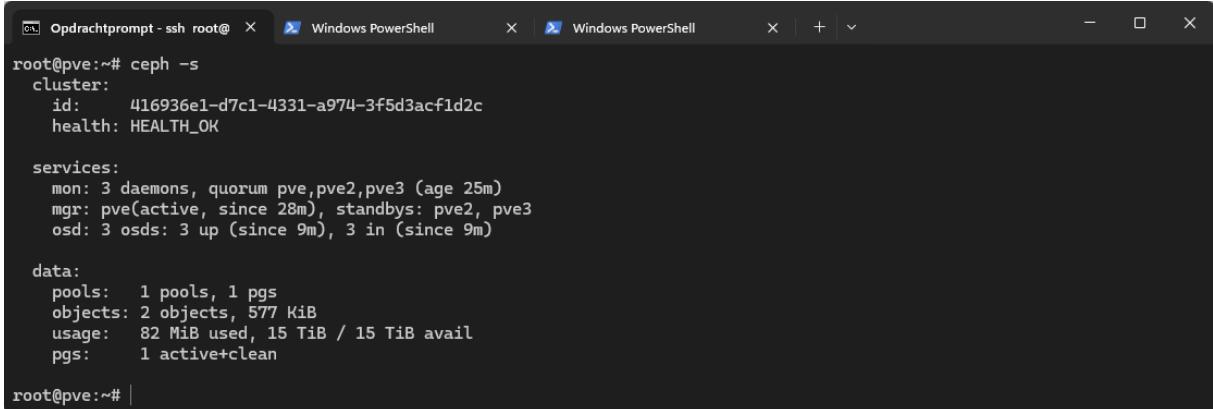
```
root@pve2:~# pveceph createmgr
```

```
root@pve3:~# pveceph createmgr
```

```
root@pve2:~# pveceph createmgr
creating manager directory '/var/lib/ceph/mgr/ceph-pve2'
creating keys for 'mgr.pve2'
setting owner for directory
enabling service 'ceph-mgr@pve2.service'
Created symlink '/etc/systemd/system/ceph-mgr.target.wants/ceph-mgr@pve2.service' -> '/usr/lib/systemd/system/ceph-mgr@.service'.
starting service 'ceph-mgr@pve2.service'
root@pve2:~# |
```

We controleren de huidige situatie op eender welke server.

```
ceph -s
```



```
root@pve:~# ceph -s
cluster:
  id: 416936e1-d7c1-4331-a974-3f5d3acf1d2c
  health: HEALTH_OK

  services:
    mon: 3 daemons, quorum pve,pve2,pve3 (age 25m)
    mgr: pve(active, since 28m), standbys: pve2, pve3
    osd: 3 osds: 3 up (since 9m), 3 in (since 9m)

  data:
    pools: 1 pools, 1 pgs
    objects: 2 objects, 577 KiB
    usage: 82 MiB used, 15 TiB / 15 TiB avail
    pgs: 1 active+clean

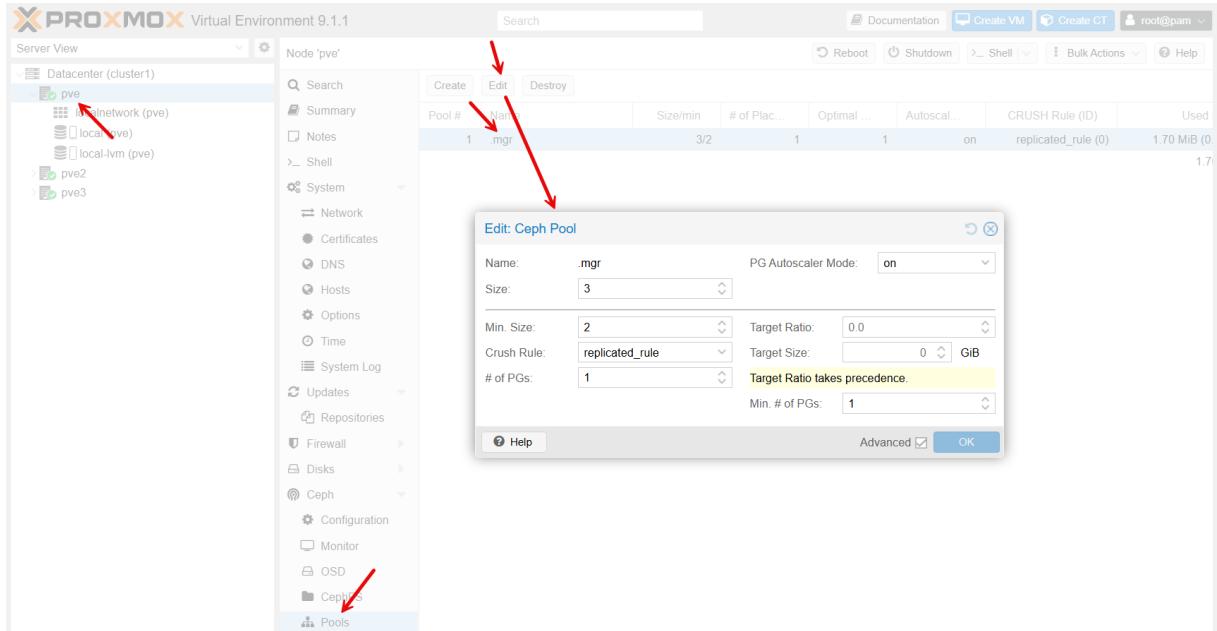
root@pve:~#
```

16.6.3.6 Pool configureren

Een Ceph pool is een logische groep voor het opslaan van objecten. Essentiële parameters voor een pool zijn:

- Het aantal placement groups.
- De replicatiefactor.

Navigeer naar de Ceph-sectie en selecteer Pools, selecteer Pool #1 en klik op Edit.



Je ziet hier een aantal velden. We bespreken de belangrijkste.

- Size

Het aantal replicaties van elk object in de pool.

Elk object in de pool wordt drie keer gerepliceerd, op drie verschillende OSD's.

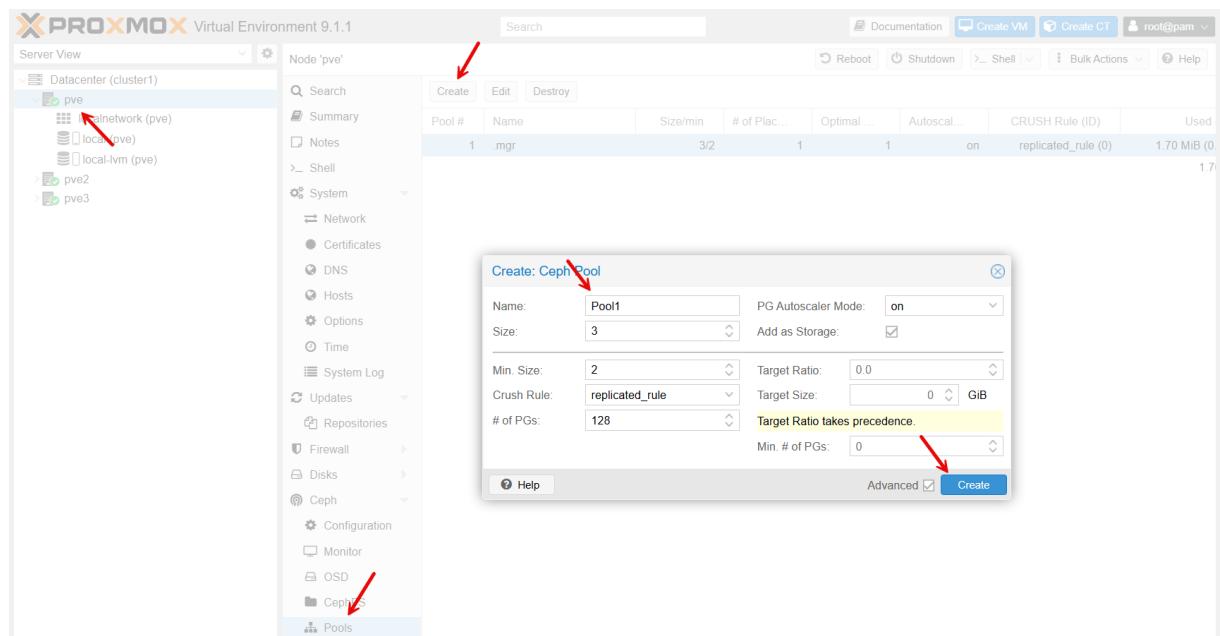
Dit zorgt voor gegevensredundantie en fouttolerantie.

Voor productieomgevingen wordt minimaal een replicatiefactor van 3 aanbevolen.

- Min.Size
Het minimale aantal replicas dat beschikbaar moet zijn voordat een object wordt beschouwd als veilig toegankelijk.
Het cluster blijft operationeel zolang minstens 2 van de 3 replicas beschikbaar zijn.
Min. Size moet meestal 2 zijn bij een replicatiefactor van 3.
- Crush Rule
De CRUSH (Controlled Replication Under Scalable Hashing)-regel bepaalt hoe objecten worden verdeeld over de OSD's in het cluster.
- # of PGs (Placement Groups)
Het aantal Placement Groups (PG's) bepaalt hoe objecten worden verdeeld over de OSD's. Hoe meer PG's een pool heeft, hoe fijner de objecten worden verdeeld over de beschikbare OSD's. Dit resulteert in een betere balans in termen van opslagruimte en I/O-belasting. Meer PG's verbruiken wel meer CPU, geheugen en netwerkbandbreedte op de OSD's en monitors.
- PG Autoscaler Mode
Dit regelt of het aantal PG's automatisch wordt aangepast.
- Target Ratio
Het relatieve deel van de totale clusteropslag dat deze pool mag gebruiken. Hier is geen limiet ingesteld (0.0).
- Target Size
De maximale grootte van de pool. Ook hier is geen limiet ingesteld voor de grootte van de pool (0 GiB).
- Min. # of PGs
Het minimum aantal Placement Groups dat voor deze pool mag worden ingesteld.

In Ceph is de .mgr pool een speciale pool die wordt gebruikt door de Ceph Manager (Mgr) daemon. Deze pool bevat metadata die nodig is voor de werking van de Manager, zoals monitoringdata en clusterstatistieken. Het verwijderen van deze pool kan ernstige gevolgen hebben voor de functionaliteit van de Ceph-cluster.

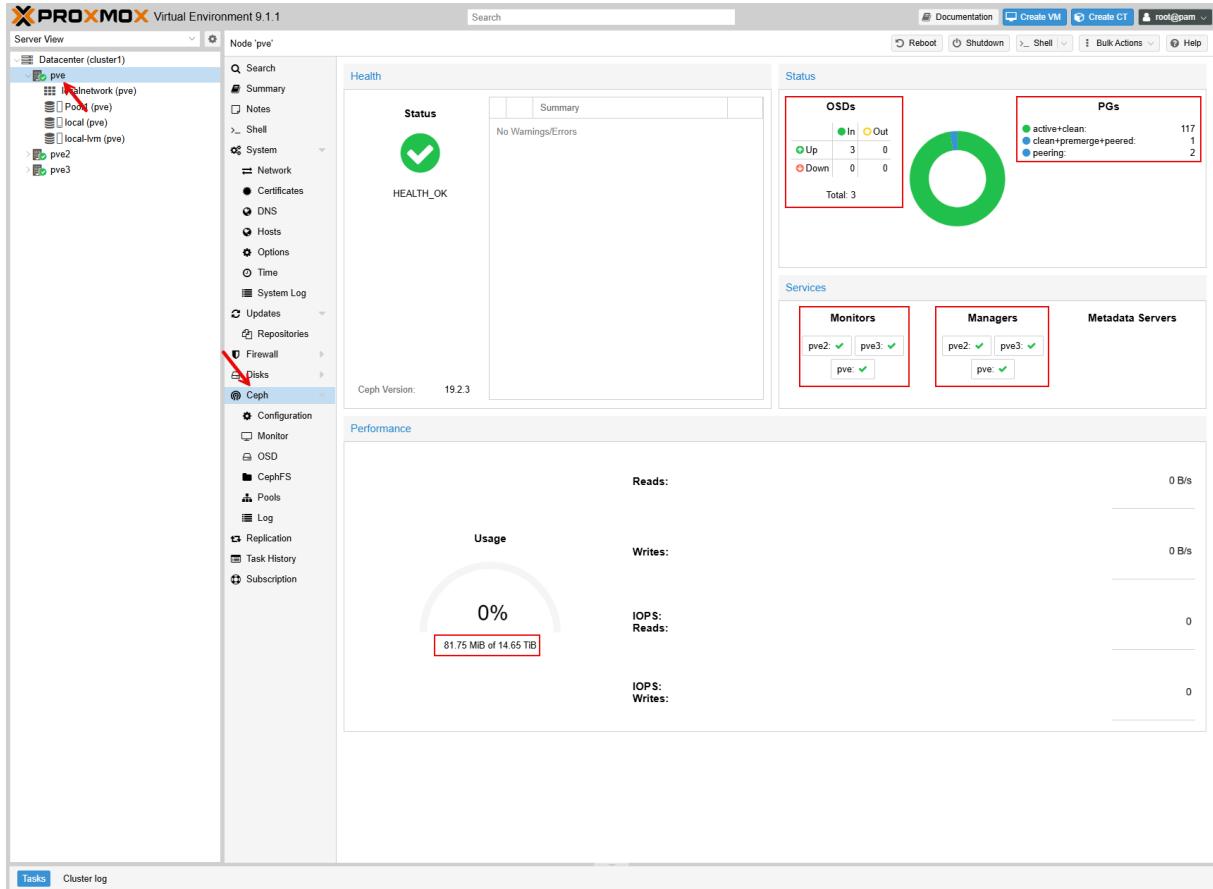
We maken een nieuwe pool bij door op Create te klikken. Geef als naam Pool1 en klik op Create.



16.6.4 Check

Ga naar de grafische omgeving van PVE en kijk naar de status van Ceph.

Je ziet ook de OSD's en de Monitors.



Je ziet dat er 3 OSD's, 3 monitors en 3 managers zijn zoals we hebben ingesteld.

Eronder zie je ook de totale hoeveelheid opslagruimte.

Je kan bijna dezelfde inhoud tonen via CLI zoals we reeds gezien hebben.

```
root@pve:~# ceph -s
```

```

root@pve:~# ceph -s
cluster:
  id:        416936e1-d7c1-4331-a974-3f5d3acf1d2c
  health:    HEALTH_OK

  services:
    mon: 3 daemons, quorum pve,pve2,pve3 (age 32m)
    mgr: pve(active, since 34m), standbys: pve2, pve3
    osd: 3 osds: 3 up (since 15m), 3 in (since 16m)

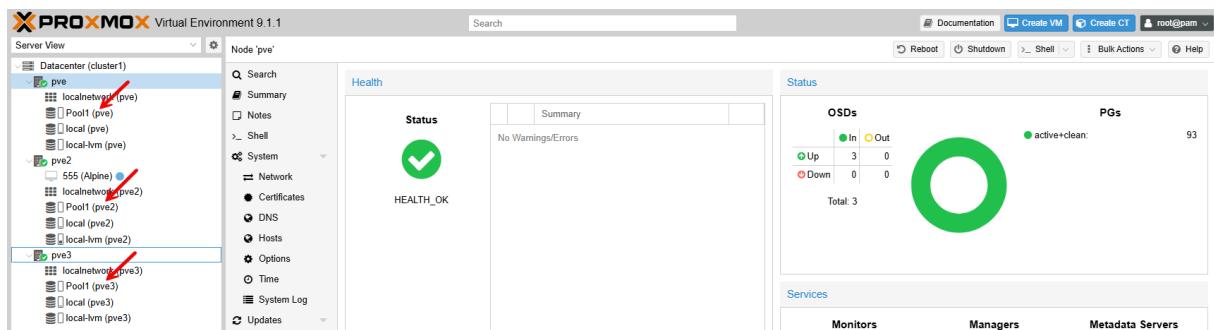
  data:
    pools:   2 pools, 108 pgs
    objects: 2 objects, 577 KiB
    usage:   136 MiB used, 15 TiB / 15 TiB avail
    pgs:     108 active+clean

root@pve:~#

```

16.7 Voorbeeld Live migratie

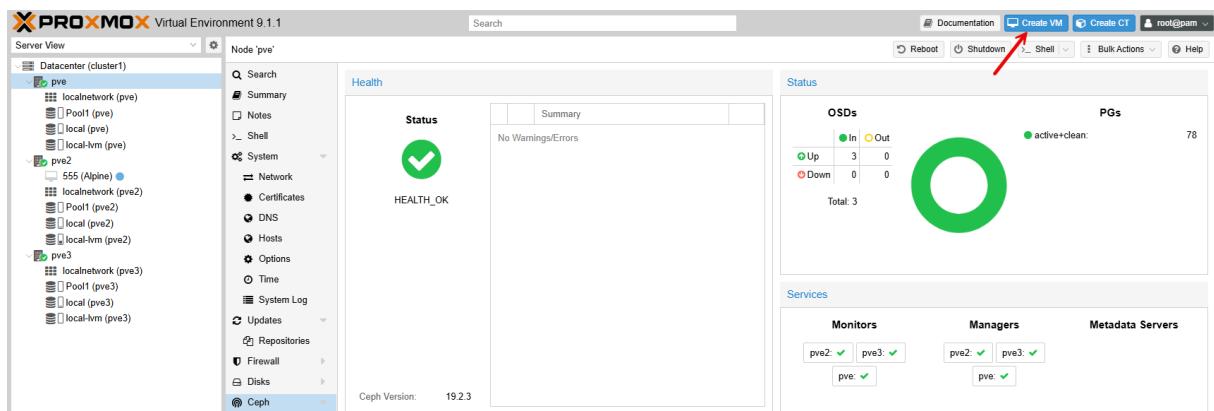
Als je kijkt in Server View aan de linkerkant zal je ook zien dat Pool1 nu beschikbaar is voor de 3 servers.



Wanneer je een VM aanmaakt in Ceph pool zal tijdens een migratie alleen het RAM-geheugen worden gemigreerd. De opslag wordt in realtime gesynchroniseerd tussen alle drie de schijven die beschikbaar zijn op de drie nodes. Dit is het voordeel van self-storage: het biedt high availability. Aangezien er drie nodes zijn, kunnen virtuele machines (VM's) naar elk van deze nodes worden gemigreerd.

We kunnen beginnen met het aanmaken van een virtuele machine die gebruik maakt van Pool1.

Stel dat ik naar PVE ga en daar een VM aanmaak.



Voor VM ID kiezen we 113. Ik geef de VM de naam "VMx". Klik Next.

Create: Virtual Machine

General OS System Disks CPU Memory Network Confirm

Node: pve
VM ID: 113
Name: VMx

Add to HA:

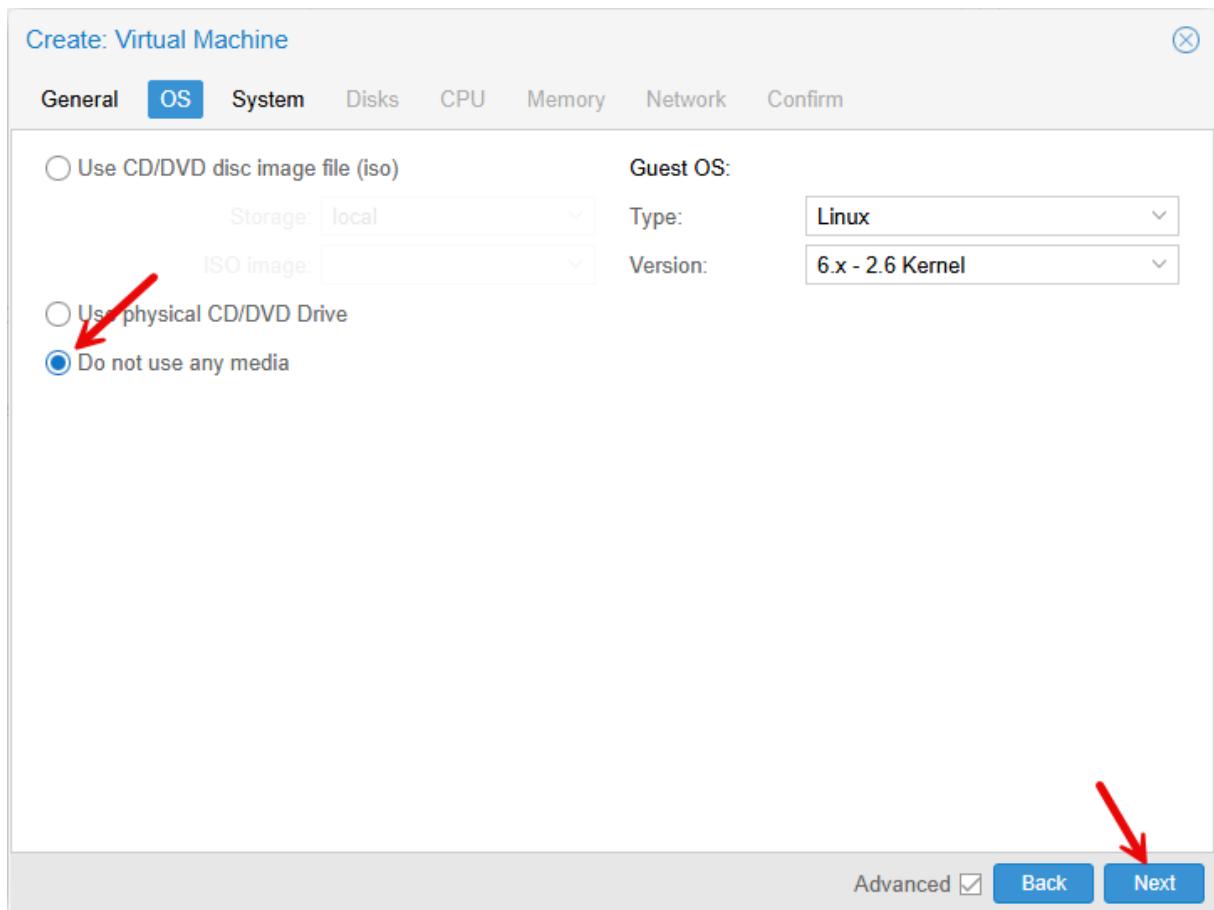
Start at boot: Start/Shutdown order: any

Startup delay: default
Shutdown timeout: default

Tags
No Tags

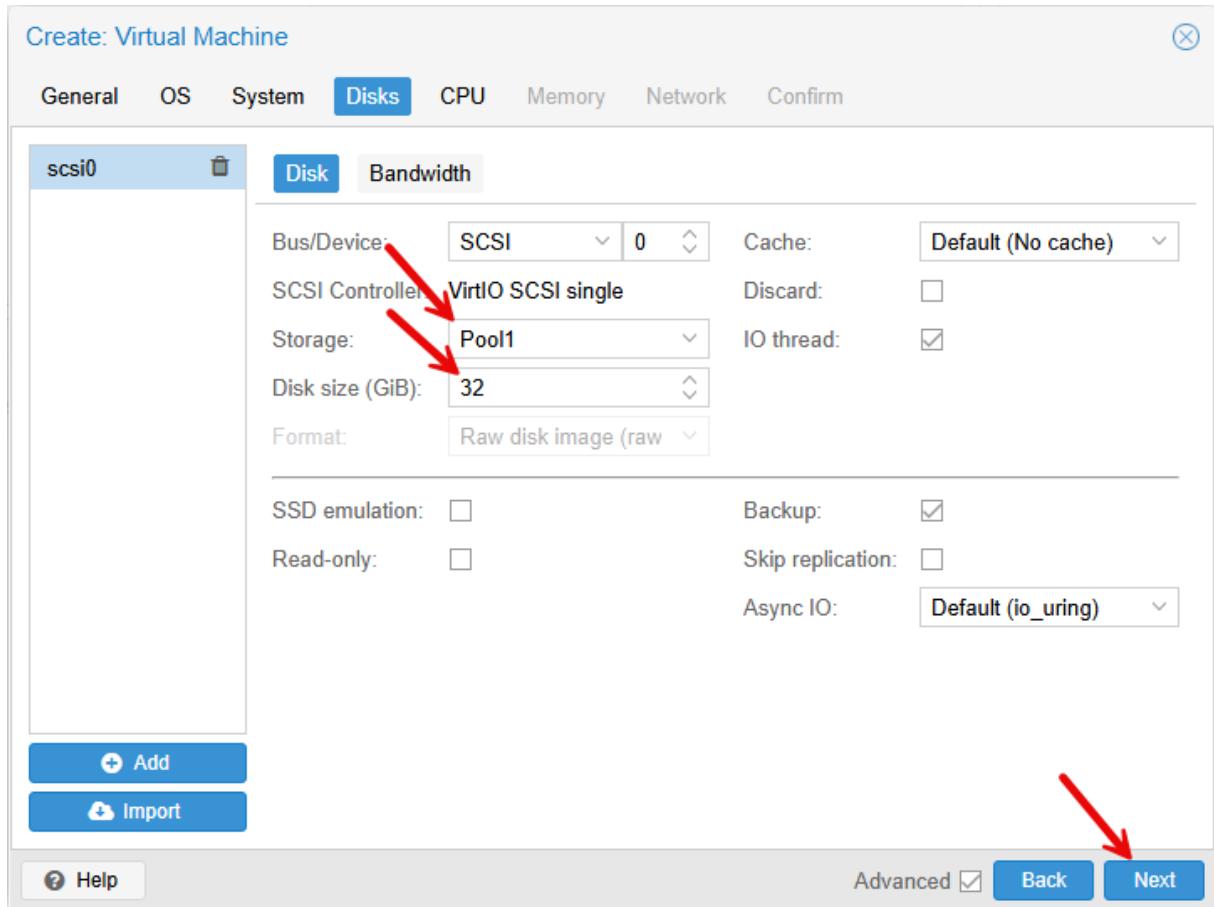
Help Advanced Back

Het besturingssysteem (OS) laat ik leeg; ik installeer dit nu niet. Klik Next.



De systeeminstellingen laat ik standaard. Klik Next.

Ik maak een virtuele schijf aan van 32 GB. Deze sla ik nu uiteraard op in Pool1. Klik Next.



De CPU-, geheugen- en netwerkinstellingen laat ik standaard. Klik telkens op Next.

Klik in het laatste venster op Finish.

Create: Virtual Machine

General OS System Disks CPU Memory Network **Confirm**

Key ↑	Value
cores	1
cpu	x86-64-v2-AES
ide2	none,media=cdrom
memory	2048
name	VMx
net0	virtio,bridge=vmbr0,firewall=1
nodename	pve
numa	0
ostype	l26
scsi0	Pool1:32,iothread=on
scsихw	virtio-scsi-single
sockets	1
vmid	113

Start after created

Advanced **Back** **Finish**

De VM wordt aangemaakt en kan worden ingeschakeld.

Let op: studenten die geen nested virtualization gebruiken moeten uiteraard KVM hardware virtualization uitschakelen.

Virtual Machine 113 (VMx) on node 'pve'

Summary Edit Revert

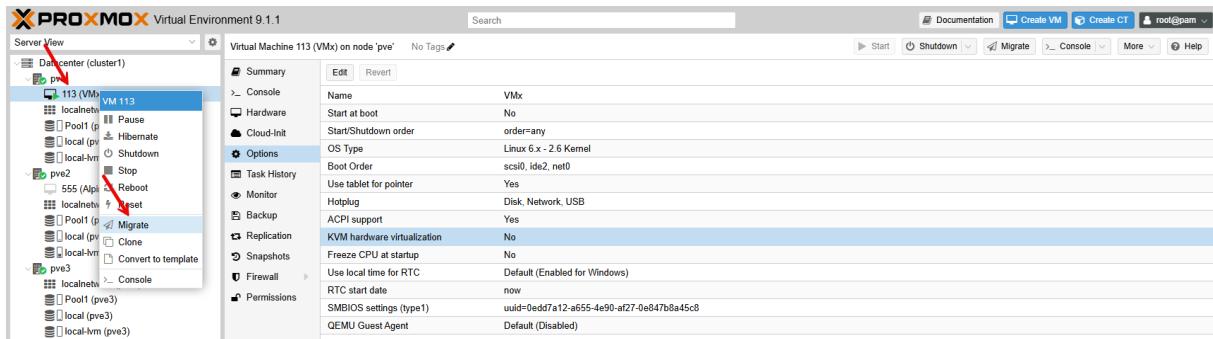
Name: VMx
Start at boot: No
OS Type: Linux 6.x - 2.6 Kernel
Boot Order: scsi0,ide2,net0
ACPI support: Yes
Hotplug: Yes
KVM hardware virtualization: Enabled:

Enabled: OK

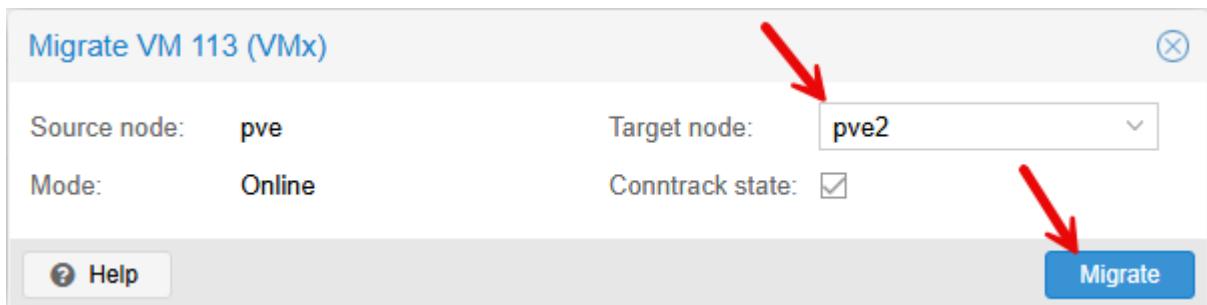


De VM draait nu en ik wil deze in realtime migreren naar een andere node. De VM bevindt zich op dit moment op PVE en ik migreer deze naar PVE2.

Klik hiervoor met de rechtermuisknop op 113 en klik voor Migrate.



Selecteer PVE2 en kies voor Migrate.



De migratie wordt gestart en voltooid. Het proces verloopt snel, omdat alleen het RAM-geheugen wordt gemigreerd. Het RAM-geheugen is het enige dat gekopieerd hoeft te worden van de ene PV naar de andere, aangezien de opslag gemeenschappelijk is. Hierdoor verloopt de migratie efficiënt en is er amper downtime.

Task viewer: VM 113 - Migrate (pve --> pve2) X

Output Status

Stop Download

```
2025-11-25 16:54:12 starting migration of VM 113 to node 'pve2' (192.168.112.120)
2025-11-25 16:54:12 starting VM 113 on remote node 'pve2'
2025-11-25 16:54:14 start remote tunnel
2025-11-25 16:54:15 ssh tunnel ver 1
2025-11-25 16:54:15 starting online/live migration on unix:/run/qemu-server/113.migrate
2025-11-25 16:54:15 set migration capabilities
2025-11-25 16:54:15 migration downtime limit: 100 ms
2025-11-25 16:54:15 migration cachesize: 256.0 MiB
2025-11-25 16:54:15 set migration parameters
2025-11-25 16:54:15 start migrate command to unix:/run/qemu-server/113.migrate
2025-11-25 16:54:16 average migration speed: 2.0 GiB/s - downtime 12 ms
2025-11-25 16:54:16 migration completed, transferred 7.1 MiB VM-state
2025-11-25 16:54:16 migration status: completed
2025-11-25 16:54:16 stopping migration dbus-vmstate helpers
2025-11-25 16:54:16 migrated 0 conntrack state entries
2025-11-25 16:54:18 flushing conntrack state for guest on source node
2025-11-25 16:54:21 migration finished successfully (duration 00:00:09)
TASK OK
```

Na de migratie is VM 113 uiteraard niet langer zichtbaar op PVE, maar wel op PVE2. Op deze manier kan eenvoudig een VM van de ene naar de andere PV worden gemigreerd.

16.8 High availability

16.8.1 Algemeen

We kunnen de beschikbaarheid uitdrukken als een percentage van de uptime in een bepaald jaar.

Beschikbaarheid %	Uitvaltijd per jaar
99	3,65 dagen
99,9	8,76 uur
99,99	52.56 minuten
99.999	5.26 minuten
99,9999	31,5 seconden
99,99999	3,15 seconden

Uiteraard willen we de beschikbaarheid zo hoog mogelijk.

De volgende oplossingen worden hiervoor toegepast:

- Gebruik betrouwbare “server”-componenten
Computercomponenten met dezelfde functionaliteit kunnen verschillende betrouwbaarheidsgestallen hebben, afhankelijk van de componentkwaliteit. De meeste

leveranciers verkopen componenten met een hogere betrouwbaarheid als "server"-componenten - meestal tegen een hogere prijs.

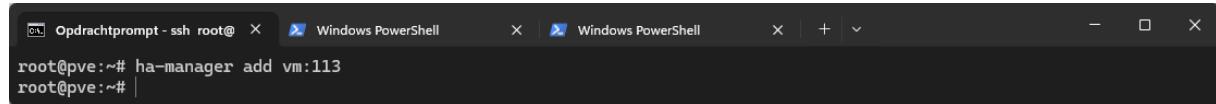
- Elimineer single point of failure (redundante componenten)
 - o Gebruik een onderbrekingsvrije voeding (UPS)
 - o Gebruik redundante voedingen in uw servers
 - o Gebruik ECC-RAM
 - o Gebruik redundante netwerkhardware
 - o RAID gebruiken voor lokale opslag
 - o Gebruik gedistribueerde, redundante opslag voor VM-gegevens
- Verminder de uitvaltijd verder door
 - o snel bereikbare beheerders (24/7)
 - o beschikbaarheid van reserveonderdelen (andere knooppunten in een Proxmox VE-cluster)
 - o automatische foutdetectie (geleverd door ha-manager)
 - o automatische failover (geleverd door ha-manager)

Beschikbaarheid verhogen van 99% naar 99,9% is relatief eenvoudig. Maar beschikbaarheid verhogen van 99,9999% naar 99,99999% is erg moeilijk en kostbaar. ha-manager heeft typische foutdetectie- en failovertijden van ongeveer 2 minuten, dus u kunt niet meer dan 99,999% beschikbaarheid krijgen.

16.8.2 Management Tasks

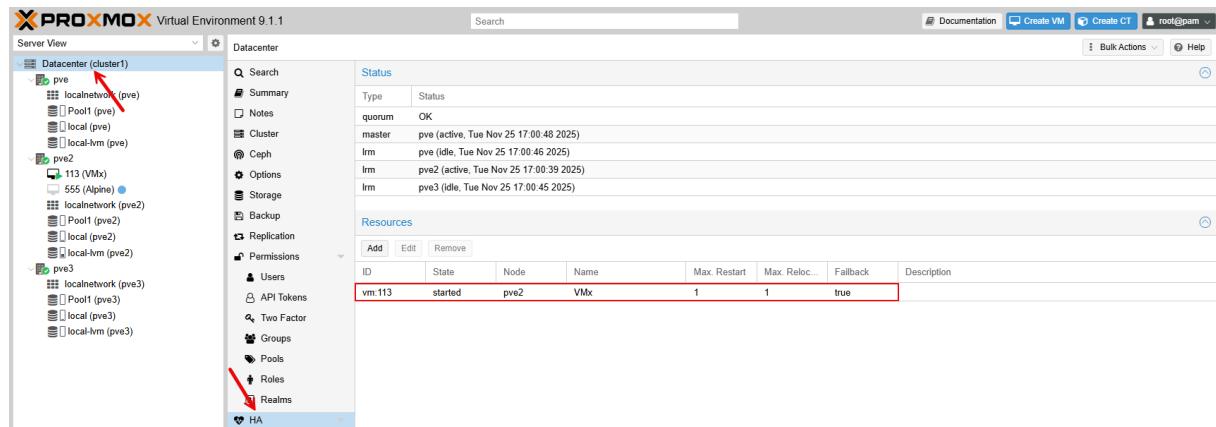
De eerste stap is om HA voor een resource in te schakelen. Dit doet u door de resource toe te voegen aan de HA-resourceconfiguratie. U kunt dit doen met behulp van de GUI of gewoon de opdrachtregeltool gebruiken, bijvoorbeeld:

```
root@pve:~# ha-manager add vm:113
```



```
Opdrachtprompt - ssh root@ X Windows PowerShell X Windows PowerShell X + - □ X
root@pve:~# ha-manager add vm:113
root@pve:~# |
```

In de grafische omgeving zie je dit nu terug bij Datacenter, HA.



De HA-stack probeert nu de resources te starten en draaiende te houden.

U kunt de status van de 'aangevraagde' resources configureren. U wilt bijvoorbeeld dat de HA-stack de resource stopt. Deze opdracht verandert de HA-status van de VM naar stopped, wat betekent dat de HA-manager de VM niet opnieuw zal starten na een failover of herstart:

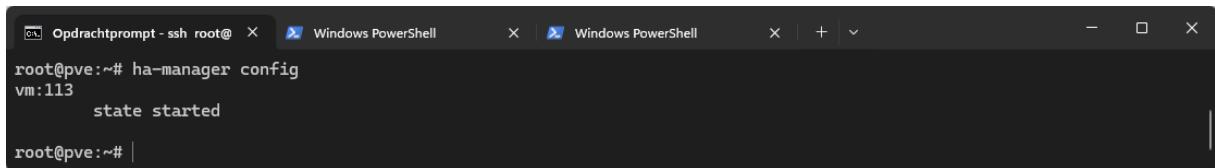
```
root@pve:~# ha-manager set vm:113 --state stopped
```

We veranderen de status terug naar start.

```
root@pve:~# ha-manager set vm:113 --state started
```

Om de huidige HA-resourceconfiguratie te bekijken, gebruikt u:

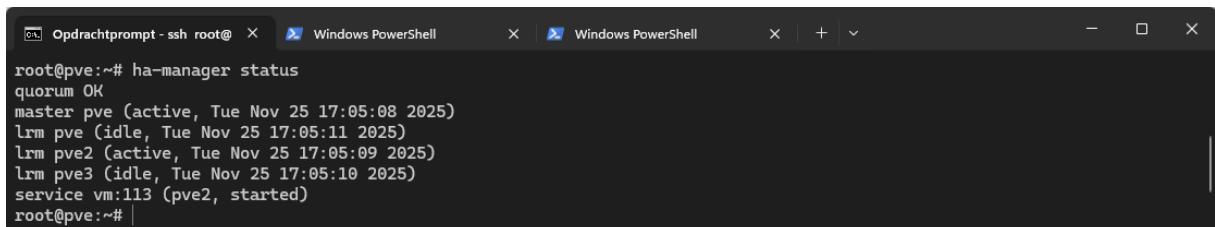
```
root@pve:~# ha-manager config
```



```
Opdrachtprompt - ssh root@ X Windows PowerShell X | Windows PowerShell X - + - 
root@pve:~# ha-manager config
vm:113
    state started
root@pve:~# |
```

En u kunt de actuele HA-manager en resourcestatus bekijken met:

```
root@pve:~# ha-manager status
```

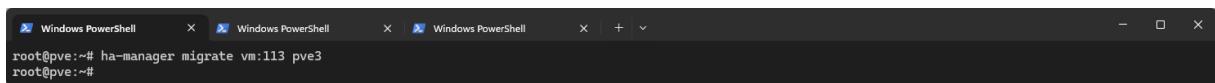


```
Opdrachtprompt - ssh root@ X Windows PowerShell X | Windows PowerShell X - + - 
root@pve:~# ha-manager status
quorum OK
master pve (active, Tue Nov 25 17:05:08 2025)
lrm pve (idle, Tue Nov 25 17:05:11 2025)
lrm pve2 (active, Tue Nov 25 17:05:09 2025)
lrm pve3 (idle, Tue Nov 25 17:05:10 2025)
service vm:113 (pve2, started)
root@pve:~# |
```

De Local Resource Manager (LRM) draait op elke node in het cluster en is verantwoordelijk voor het lokale beheer van HA-taken. Het werkt samen met de CRM (Cluster Resource Manager), die het overzicht heeft over het hele cluster.

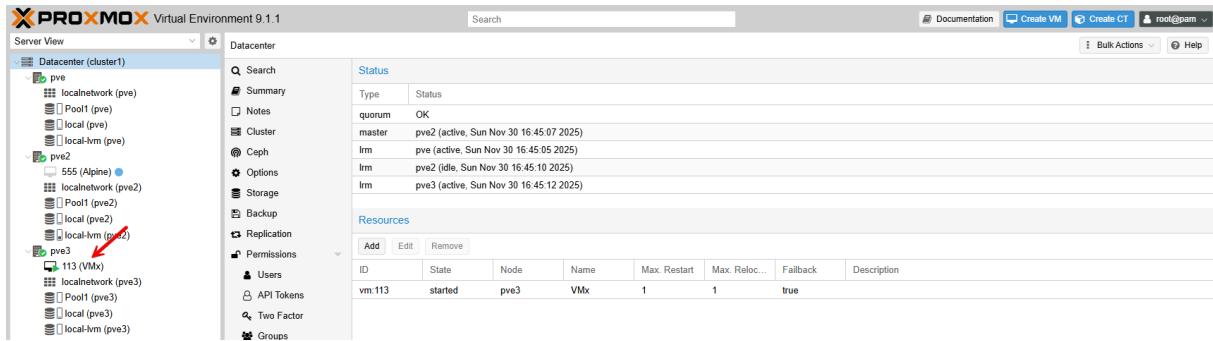
U kunt ook resourcemigratie naar andere knooppunten initiëren:

```
root@pve:~# ha-manager migrate vm:113 pve3
```



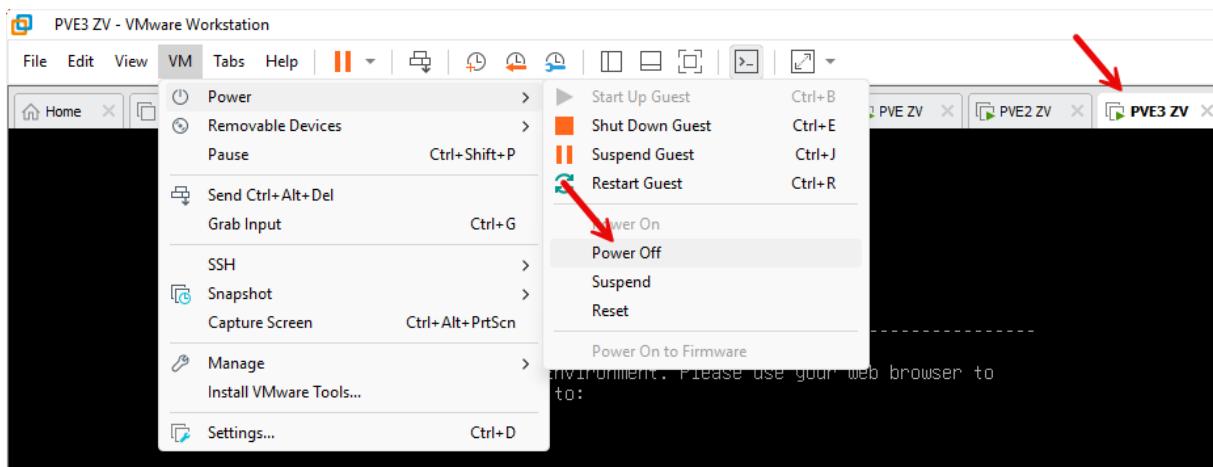
```
Windows PowerShell X | Windows PowerShell X | Windows PowerShell X - + - 
root@pve:~# ha-manager migrate vm:113 pve3
root@pve:~# |
```

Je zal in de GUI zien dat VM 113 na enkele ogenblikken op PVE3 wordt uitgevoerd.

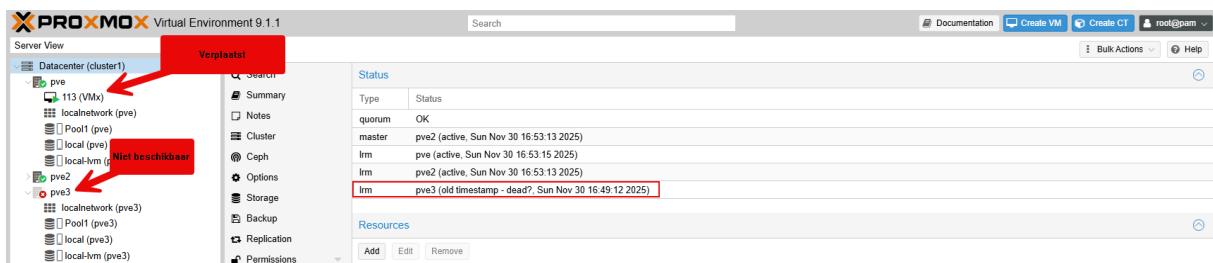


16.8.3 Check

We bootsen een stroomuitval van PVE3 na door Power Off op deze VM uit te voeren.



Je zal zien dat na enige tijd VM 113 draait op een andere server.



Zet erna de VMware VM PVE3 terug aan.

16.8.4 Affinity Rules

16.8.5 Inleiding

In Proxmox 9 bepaalt "affinity" waar een HA-resource (zoals een VM) mag draaien en hoe deze zich gedraagt binnen het HA-cluster.

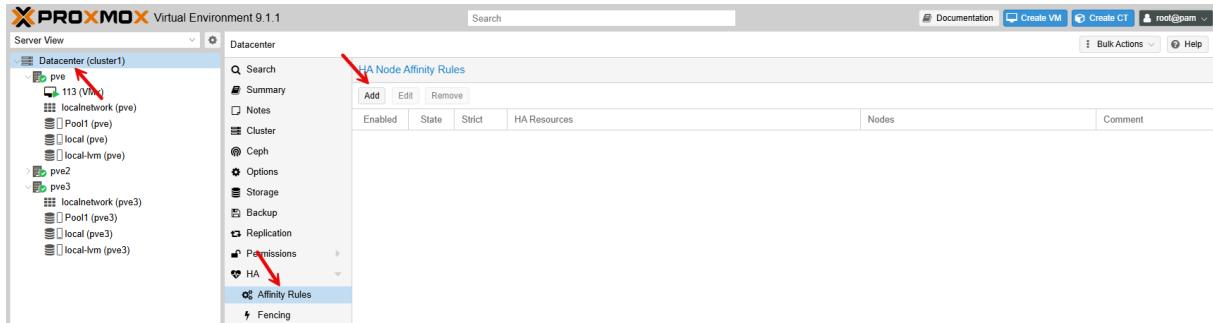
Er zijn 2 soorten affinity rules:

- HA Node Affinity bepaalt op welke nodes een VM mag draaien.
- HA Resource Affinity bepaalt de relatie tussen meerdere HA-resources (VM-VM). Je kan bepalen of je wil dat bepaalde VM's liever bij elkaar draaien of liever uit elkaar.

16.8.6 HA Node Affinity Rules

Open HA in datacenter en selecteer vervolgens Affinity Rules.

Klik op Add bij HA Node Affinity Rules.

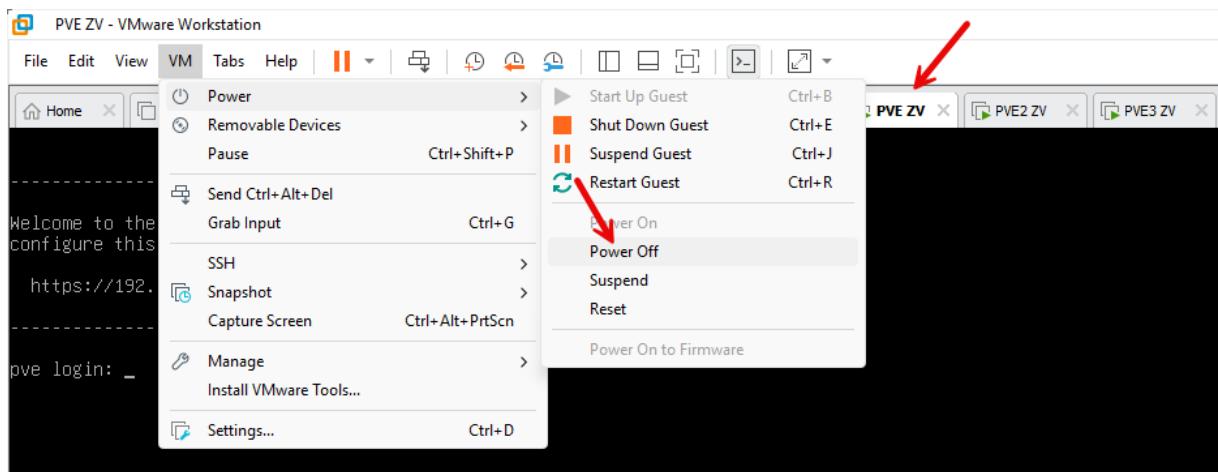


In het venster dat verschijnt vink je Enable aan en kies je bij HA Resources voor 113. Selecteer daarna de nodes pve en pve2 waarop deze VM mag draaien. Activeer vervolgens ook Strict: hierdoor wordt de VM beperkt tot de aangevinkte nodes. Klik op Add om dit toe te passen.

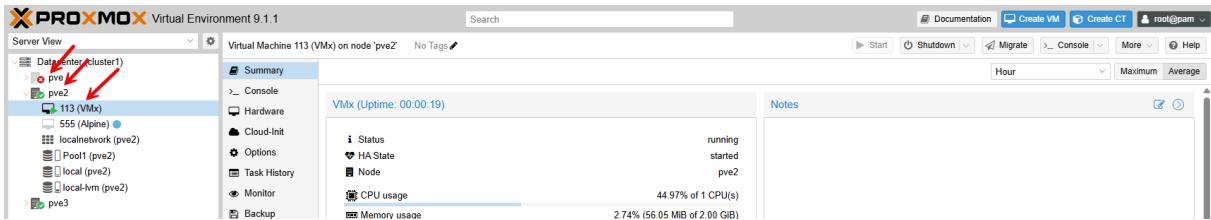
Add: HA Node Affinity

Enable:	<input checked="" type="checkbox"/>	Strict:	<input checked="" type="checkbox"/>
HA Resources:	113		
Comment:	<input type="text"/>		
Node ↑	Memory usage %	CPU usage	Priority
<input checked="" type="checkbox"/> pve	88.1 %	6.9% of 2 CPUs	<input type="button" value="^"/>
<input checked="" type="checkbox"/> pve2	67.3 %	3.4% of 2 CPUs	<input type="button" value="^"/>
<input type="checkbox"/> pve3	82.0 %	3.3% of 2 CPUs	<input type="button" value="^"/>

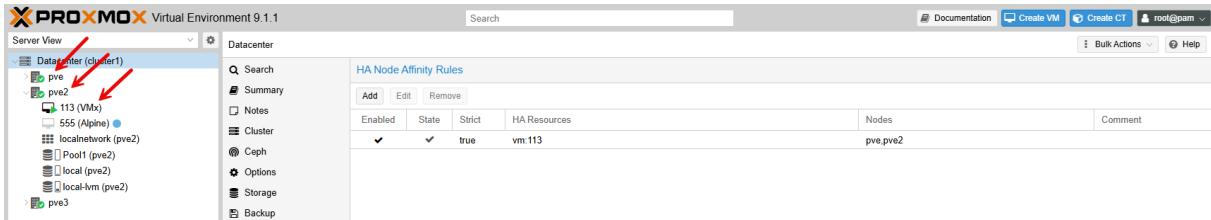
Schakel nu pve uit.



VM 113 zal nu naar pve2 verhuizen, omdat deze node binnen de toegestane selectie valt.

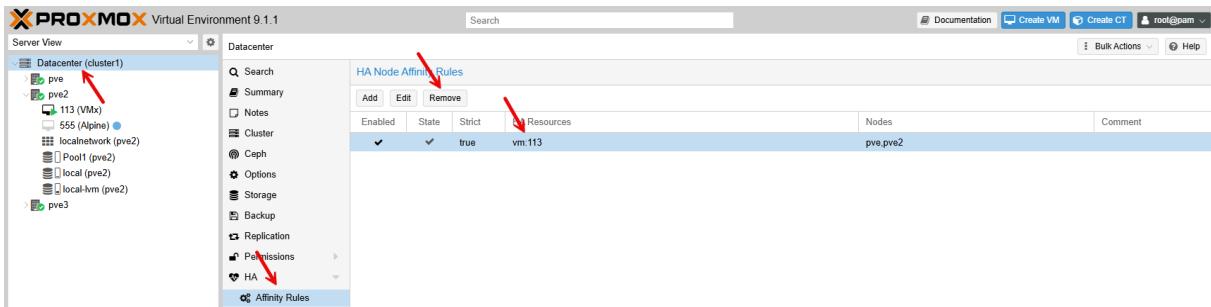


Als je pve daarna opnieuw inschakelt, blijft VM 113 gewoon op pve2 draaien, dankzij het standaard no-fallback gedrag.

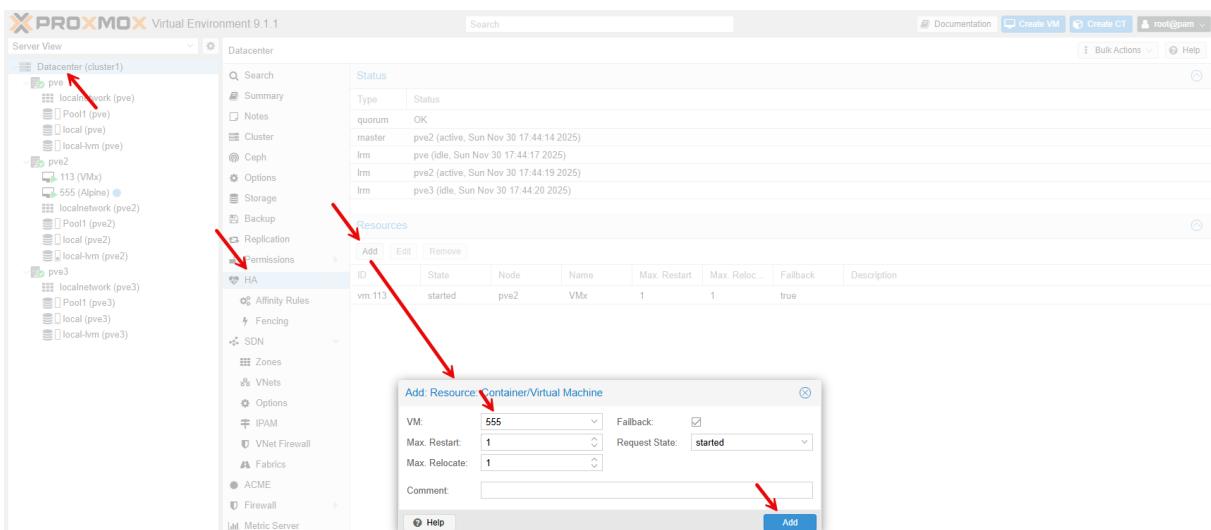


16.8.6.1 HA Resource Affinity Rules

We verwijderen eerst de HA Node Affinity Rule die we hebben ingesteld.

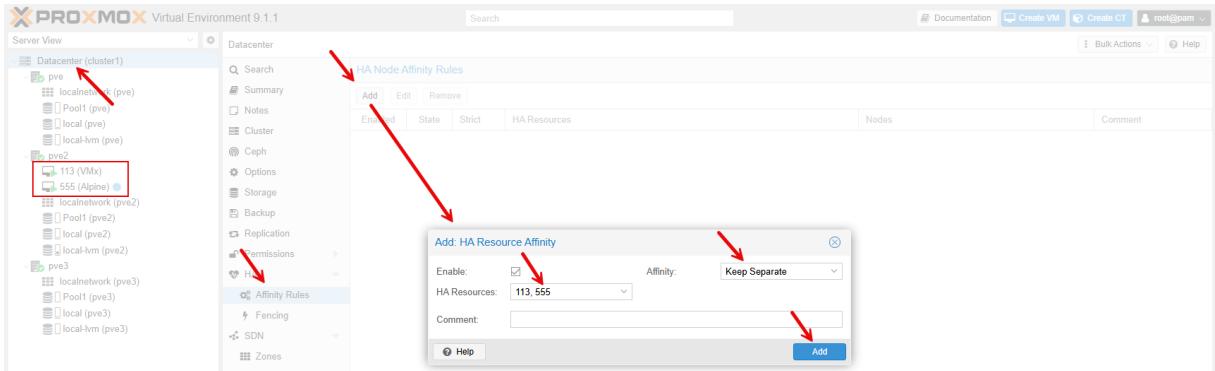


Voor HA resource affinity rule moet je eerst een tweede VM toevoegen aan HA. Voeg Alpine (555) toe aan HA.

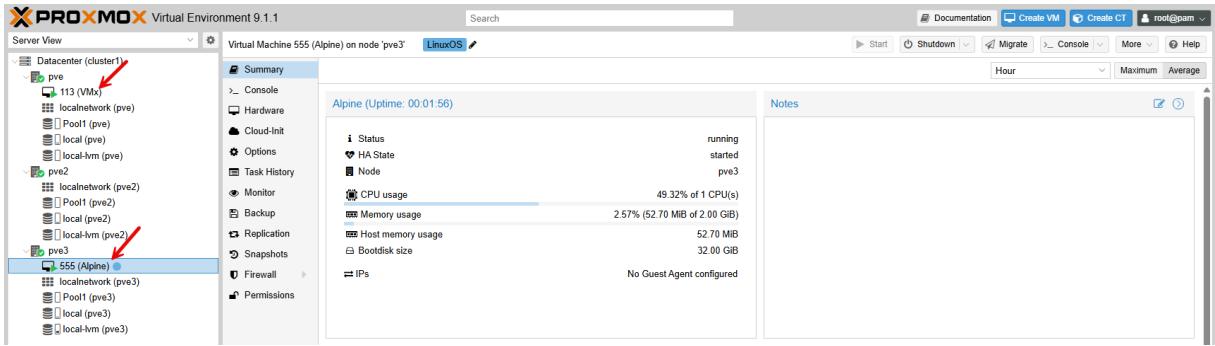


Ga nu naar Affinity Rules onder HA.

Klik op Add bij HA Resource Affinity Rules. Je kan nu kiezen om bepaalde VM's apart of samen te houden. In onderstaan voorbeeld draaien 113 en 555 op pve2. Ik stel nu in om deze apart te zetten.



Je zal constateren dat na enige tijd de VM's niet meer draaien op dezelfde node.



Verwijder nu de Resource Affinity Rule.

16.8.7 Fencing

In Proxmox wordt fencing gebruikt om een onbetrouwbare node "uit te schakelen" of ontoegankelijk te maken, zodat andere nodes in het cluster de resources (zoals VM's) veilig kunnen overnemen.

Fencing wordt meestal uitgevoerd door hardware device en wordt niet verder behandeld.

17 Proxmox CLI

17.1 Inleiding

Nu we de basis van Proxmox onder de knie hebben zullen we bestuderen hoe we via de CLI vm's, containers en netwerken kunnen aanmaken. We gaan ook een stapje verder dan hetgeen we tot nu toe gezien hebben.

De belangrijkste tools zijn:

- Qm: beheer van QEMU Virtual Machines
- Pct: beheer van containers (LXC)

17.2 Installatie VM

We maken een VM met Debian aan via de Proxmox CLI met volgende instellingen:

- VM ID: 110
- Naam: DebianLinux
- RAM: 1024 MB
- Cores: 2
- Disk: 15 GB
- ISO: Alma Linux Minimal 10
- Storage: local-lvm
- Bridge: vmbr0

We checken eerst de locatie waar we ISO's opslaan.

```
root@pve:~# pvesm status
```

Name	Type	Status	Total (KiB)	...
Pool1	rbd	active	4980692679	...
local	dir	active	62381820	...
local-lvm	lvmthin	active	121516032	...

Hieruit kunnen we volgende afleiden:

- local (dir): directory storage, geschikt voor ISO's
- local-lvm (lvmthin): block storage, kan géén ISO's opslaan
- Pool1 (rbd): Ceph RBD blockstorage, ook géén ISO's

Je kan checken waar de locatie van local zich bevindt.

```
root@pve:~# grep -A2 "^dir: local$" /etc/pve/storage.cfg
```

```
dir: local
```

```
path /var/lib/vz  
content iso,vztmpl,import,backup
```

We zoeken verder...

```
root@pve:~# find /var/lib/vz -name iso  
/var/lib/vz/template/iso
```

De standaarddirectory voor ISO's is dus /var/lib/vz/template/iso/.

We gaan naar deze directory om de ISO daar te downloaden.

```
root@pve:~# cd /var/lib/vz/template/iso/
```

De locatie waar je Debian Linux kan downloaden is <https://www.debian.org/>.

Klik op Andere downloads.

HET BESTURINGSSYSTEEM

Debian is een volledig vrij besturingssysteem!



We kopiëren het linkadres van 64-bits pc netinst iso.

Een installatie-image downloaden

- Een klein installatie-image: kan snel gedownload worden en moet op een verwijderbare schijf geschreven worden. Om dit te kunnen gebruiken, heeft uw machine een internetverbinding nodig.



Gebruik deze link om de ISO te downloaden.

```
root@pve:/var/lib/vz/template/iso# wget
https://cdimage.debian.org/debian-cd/current/amd64/iso-
cd/debian-13.2.0-amd64-netinst.iso
```

...

We gaan terug naar de homedirectory (niet nodig maar zo verliezen we hier minder ruimte 😊).

```
root@pve:/var/lib/vz/template/iso# cd
```

We maken nu een nieuwe VM aan door gebruik te maken van het commando qm.

```
root@pve:~# qm create 110 --name debianLinux --memory 1024
--cores 2 --net0 virtio,bridge=vmbr0
```

We voegen nu een virtuele SCSI-schijf van 15 GB toe aan deze VM.

```
root@pve:~# qm set 110 --scsihw virtio-scsi-pci --scsi0 local-
lvm:15
```

...

We koppelen de ISO aan deze VM

```
root@pve:~# qm set 110 --cdrom local:iso/debian-13.2.0-amd64-
netinst.iso
```

We zorgen ervoor dat we opstarten van de ISO.

```
root@pve:~# qm set 110 --boot order=ide2
```

PS Ide0 en 1 worden gebruikt voor de eerste en tweede harde schijf. Ide2 wordt gebruikt voor voor de ISO.

Aangezien Debian Linux UEFI ondersteunt zullen we UEFI instellen.

- Eerst stellen we BIOS-type of UEFI in.

```
root@pve:~# qm set 110 --bios ovmf
```

```
update VM 110: --bios ovmf
```

Ovmf is Open Virtual Machine Firmware.

- Met onderstaande maak je een EFI-disk aan. Dat is een kleine virtuele disk die gebruikt wordt door het UEFI-firmware systeem. Zonder deze disk kan je niet opstarten.

```
root@pve:~# qm set 110 --efidisk0 local-lvm:1
```

...

We zetten KVM uit als we geen nested virtualization hebben.

```
root@pve:~# qm set 110 --kvm 0
```

```
update VM 110: --kvm 0
```

We vragen de VM's op.

```
root@pve:~# qm list
```

VMID	NAME	STATUS	MEM(MB)	BOOTDISK(GB)	PID
110	DebianLinux	stopped	1024	0.00	0

Er staat geen grootte bij bootdisk (GB) omdat je nu opstart van CD-ROM.

Uiteraard is DebianLinux nu zichtbaar in de GUI.



We starten de VM op.

```
root@pve:~# qm start 110
```

Ga naar de console via de GUI. Je krijgt nu onderstaand scherm. Kies voor Install (niet grafische installatie).



Je zal de tab-toets moeten gebruiken omdat de muis niet werkt.

Gebruik onderstaande instellingen in de schermen die erna volgen:

- Kies nu de taal Dutch - Nederlands.
- Kies nu voor België.
- Kies je toetsenbordindeling, bijvoorbeeld Belgisch.
- Debian laadt nu aanvullende installatiemodules, herkent hardware, stelt netwerk in enz.
- Wij laten Debian staan voor de computernaam.
- In het venster van het domein typ je lan in.
- Kies erna het wachtwoord voor de root. We kiezen voor 123.
- Geef het wachtwoord moet je erna bevestigen.
- Maak erna een gebruiker student aan.
 - o We kiezen als volledige naam student.
 - o De gebruikersnaam is student.
- Stel het wachtwoord voor de gebruiker student in. Kies weer voor 123.
- Ter controle geef je het wachtwoord voor student nogmaals in.
- Bij schijfindeling kies je voor: Begeleid - benut gehele schijf.
- Je krijgt nog de melding dat alle gegevens van de schijf gewist zullen worden. Bevestig dit.
- Erna kan je instellen hoe de schijf wordt ingedeeld. Kies voor Alles in één partitie (aanbevolen voor nieuwe gebruikers).

- Erna kies je voor "Schijfindeling afsluiten & veranderingen naar schijf schrijven."
- In het venster over de aanpassing van de schijf kies je Ja.
- Het basissysteem wordt nu geïnstalleerd.
- Je kan nu de installatiemedia laten scannen. Doe dit niet. Kies dus voor Nee.
- Kies erna voor België voor spiegelserver.
- Kies dan voor <ftp.belnet.be>.
- Het veld van de proxyserver laat je leeg.
- Laat Debian verder installeren en configureren.
- Erna krijgt je de melding of je wil meewerken aan het onderzoek naar het gebruik van pakketten. Kies Nee.
- Kies nu voor Standaard systeemhulpmiddelen. Meer installeren is overkill voor ons systeem.
- Het kan zijn dat het scherm blauw wordt maar de installatie gaat verder.
- Erna krijgt je de melding dat de installatie voltooid is. Klik dan op Verdergaan.
- Zet nu de VM uit aangezien je terug in de installer terechtkomt.

Zet nu de bootorder op de SCSI-disk

```
root@pve:~# qm set 110 --boot order=scsi0
update VM 110: --boot order=scsi0
```

Start de VM terug op.

```
root@pve:~# qm start 110
```

Je zal zien dat je nu kan inloggen op het Debian-systeem. Je hebt verbinding met het internet enz.

```
QEMU (debianlinux) - noVNC - Google Chrome
Niet beveiligd https://192.168.112.110:8006/?console=kvm&novnc=1&vmid=110&vmname=debianlinux&node=pve&resize=off&cmd=
Debian GNU/Linux 13 debian tty1
debian login: student
Password:
Linux debian 6.12.57+deb13-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.12.57-1 (2025-11-05) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
student@debian:~$ halt
-bash: halt: opdracht niet gevonden
student@debian:~$ 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 ::/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: ens18: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether bc:24:11:5c:65:dd brd ff:ff:ff:ff:ff:ff
    altname enp0s18
    altname enxbc24115c65dd
    inet 192.168.112.149/24 brd 192.168.112.255 scope global dynamic noprefixroute ens18
        valid_lft 1602sec preferred_lft 1377sec
    inet6 fe80::85ac:7f37:4d9ff:fd15/64 scope link
        valid_lft forever preferred_lft forever
student@debian:~$ ping www.google.be
PING www.google.be (74.125.133.94) 56(84) bytes of data.
64 bytes from w0-in-f94.1e100.net (74.125.133.94): icmp_seq=1 ttl=128 time=21.6 ms
64 bytes from w0-in-f94.1e100.net (74.125.133.94): icmp_seq=2 ttl=128 time=22.3 ms
64 bytes from w0-in-f94.1e100.net (74.125.133.94): icmp_seq=3 ttl=128 time=22.3 ms
^C
--- www.google.be ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 21.564/22.058/22.322/0.349 ms
student@debian:~$ _
```

17.3 Installatie container

Zoals je weet gebruiken LXC containers templates? Deze hebben meestal de extensie .tar.gz. Controleer eerst de beschikbare templates.

```
root@pve:~# pveam available
```

...

Je krijgt nu alle beschikbare templates.

Je kan de aanwezige templates ook updaten.

```
root@pve:~# pveam update
update successful
```

We zullen zoeken naar containers van Rocky Linux.

```
root@pve:~# pveam available | grep rocky
system          rockyLinux-10-default_20251001_amd64.tar.xz
system          rockyLinux-9-default_20240912_amd64.tar.xz
```

We downloaden nu de template van Rocky Linux 10 naar local (standaardlocatie).

```
root@pve:~# pveam download local rockylinux-10-
default_20251001_amd64.tar.xz
```

We kijken na of de template aanwezig is

```
root@pve:~# pveam list local
NAME                                     SIZE
local:vztmpl/rockylinux-10-default_20251001_amd64.tar.xz    80.17MB
```

We maken nu een container aan de hand van dit LXC-template.

```
root@pve:~# pct create 200 local:vztmpl/rockylinux-10-
default_20251001_amd64.tar.xz -rootfs local-lvm:8 -net0
name=eth0,bridge=vmbr0,ip=dhcp -hostname rockyct -password
PXLPXL
```

We starten nu de container.

```
root@pve:~# pct start 200
```

We loggen nu in op de container.

```
root@pve:~# pct console 200

Connected to tty 1

Type <Ctrl+a q> to exit the console, <Ctrl+a Ctrl+a> to enter
Ctrl+a itself

Rocky Linux 10.0 (Red Quartz)

Kernel 6.17.2-1-pve on x86_64
```

```
rockyct login: root
```

```
Password:
```

```
[root@rockyct ~]#
```

17.4 Netwerken

17.4.1 Inleiding

Proxmox geeft je meerdere netwerk-opties, afhankelijk van wat je wilt bereiken. We bespreken een Linux Bridge, NAT-bridge en Linux Bond. We testen alles uit op PVE.

17.4.2 Back-up netwerkconfiguratie

Maak vooraf best een back-up van /etc/network/interfaces.

```
root@pve:~# cp /etc/network/interfaces interfaces~
```

17.4.3 NAT-bridge

17.4.3.1 NAT-bridge aanmaken via CLI

Open /etc/network/interfaces:

```
root@pve:~# nano /etc/network/interfaces
```

Voeg dit onderaan toe:

```
auto vmbr1
iface vmbr1 inet static
    address 192.168.100.1/24
    bridge_ports none
    bridge_stp off
```

```
bridge_fd 0

# NAT via vmbr0 (hoofd-bridge)
post-up iptables -t nat -A POSTROUTING -s 192.168.100.0/24 -o vmbr0 -j MASQUERADE
post-down iptables -t nat -D POSTROUTING -s 192.168.100.0/24 -o vmbr0 -j MASQUERADE
```

Pas vmbr0 aan als jouw internet-bridge anders heet.

Netwerk herstarten:

```
root@pve:~# systemctl restart networking
```

17.4.3.2 DHCP-server via CLI

We installeren dnsmasq voor de DNS-service.

```
root@pve:~# apt install dnsmasq
```

Nieuwe config maken.

```
root@pve:~# nano /etc/dnsmasq.d/vmbr1.conf
interface=vmbr1
dhcp-range=192.168.100.100,192.168.100.200,12h
```

(Her)start nu de DNS-service.

```
root@pve:~# systemctl restart dnsmasq
```

17.4.3.3 VM netwerkadapter via CLI koppelen

Je kan voor een bestaande VM de instellingen van de NIC wijzigen.

```
root@pve:~# qm set 200 -net0 virtio,bridge=vmbr1
```

Je kan voor een bestaande CT de instellingen van de NIC wijzigen.

```
root@pve:~# pct set 200 -net0 name=eth0,bridge=vmbr1,ip=dhcp
```

Start VM of container en test IP en of je verbinding hebt met internet.

```
[root@rockyct ~]# 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: eth0@if6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether bc:24:11:9f:d0:40 brd ff:ff:ff:ff:ff:ff link-netnsid 0
        inet 192.168.100.185/24 brd 192.168.100.255 scope global dynamic noprefixroute eth0
            valid_lft 43170sec preferred_lft 43170sec
[root@rockyct ~]# ping -c 1 www.google.be
PING www.google.be (64.233.184.94) 56(84) bytes of data.
64 bytes from wa-in-f94.1e100.net (64.233.184.94): icmp_seq=1 ttl=127 time=23.6 ms
--- www.google.be ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 23.649/23.649/23.649/0.000 ms
[root@rockyct ~]#
```

17.4.4 Bonding

17.4.4.1 Inleiding

Bonding betekent dat je twee fysieke netwerkkaarten samenvoegt tot één logische interface voor high availability (en eventueel meer bandbreedte).

Zo ziet dat er conceptueel uit:

ens33 ↘

ens37 ↘ → bond0 → vmbr0 → LAN/switch

De belangrijkste bond-modes:

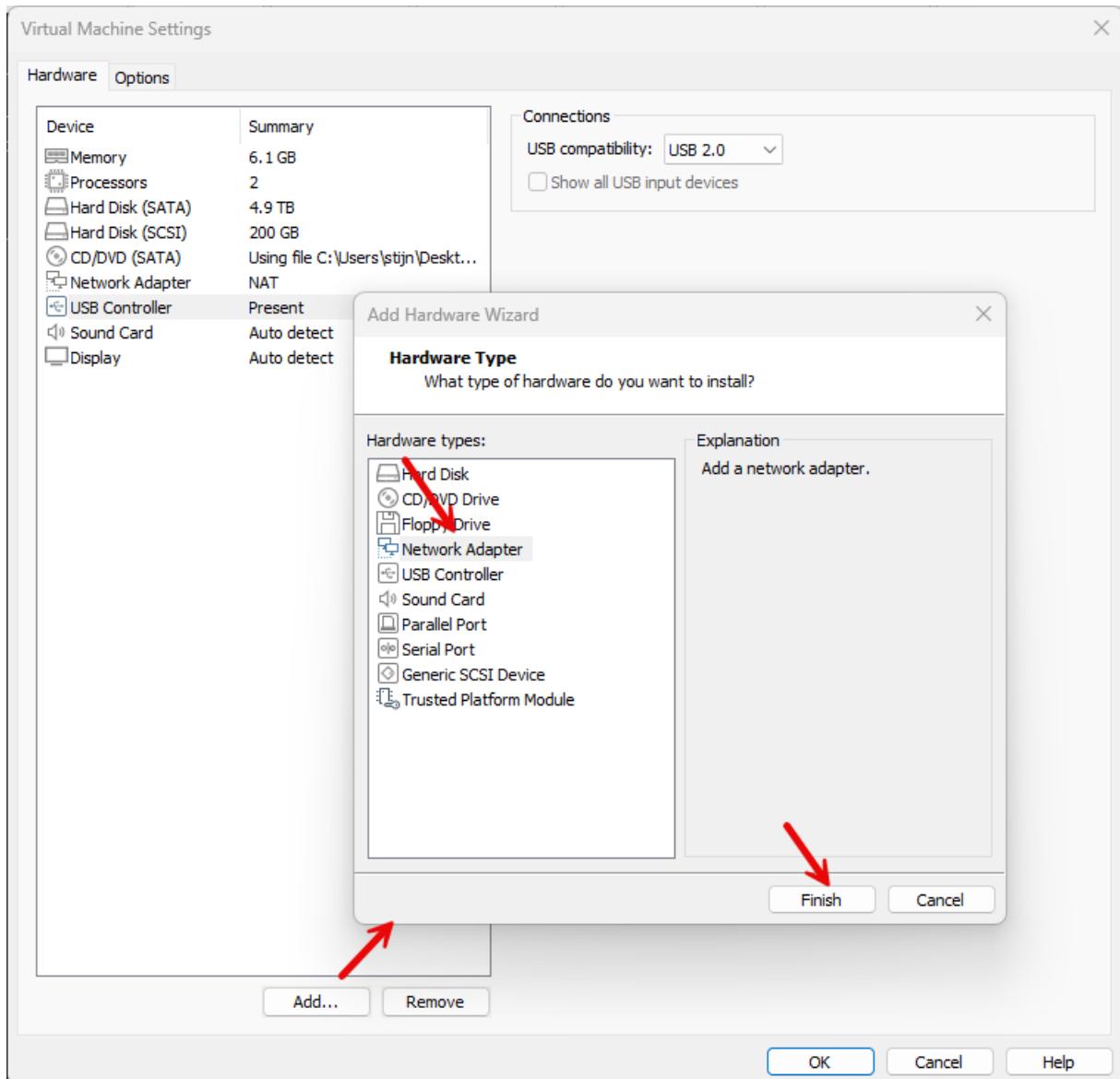
- active-backup (mode 1)
 - Slechts één NIC tegelijk actief; valt die uit, neemt de andere het naadloos over.
 - Ideaal voor redundantie, werkt ook op “domme” switches zonder LACP.smartable+1
- 802.3ad / LACP (mode 4)
 - Beide NIC's actief; biedt één redundantie én load-balancing.
 - Vereist dat de switchpoorten in een LACP-trunk staan.

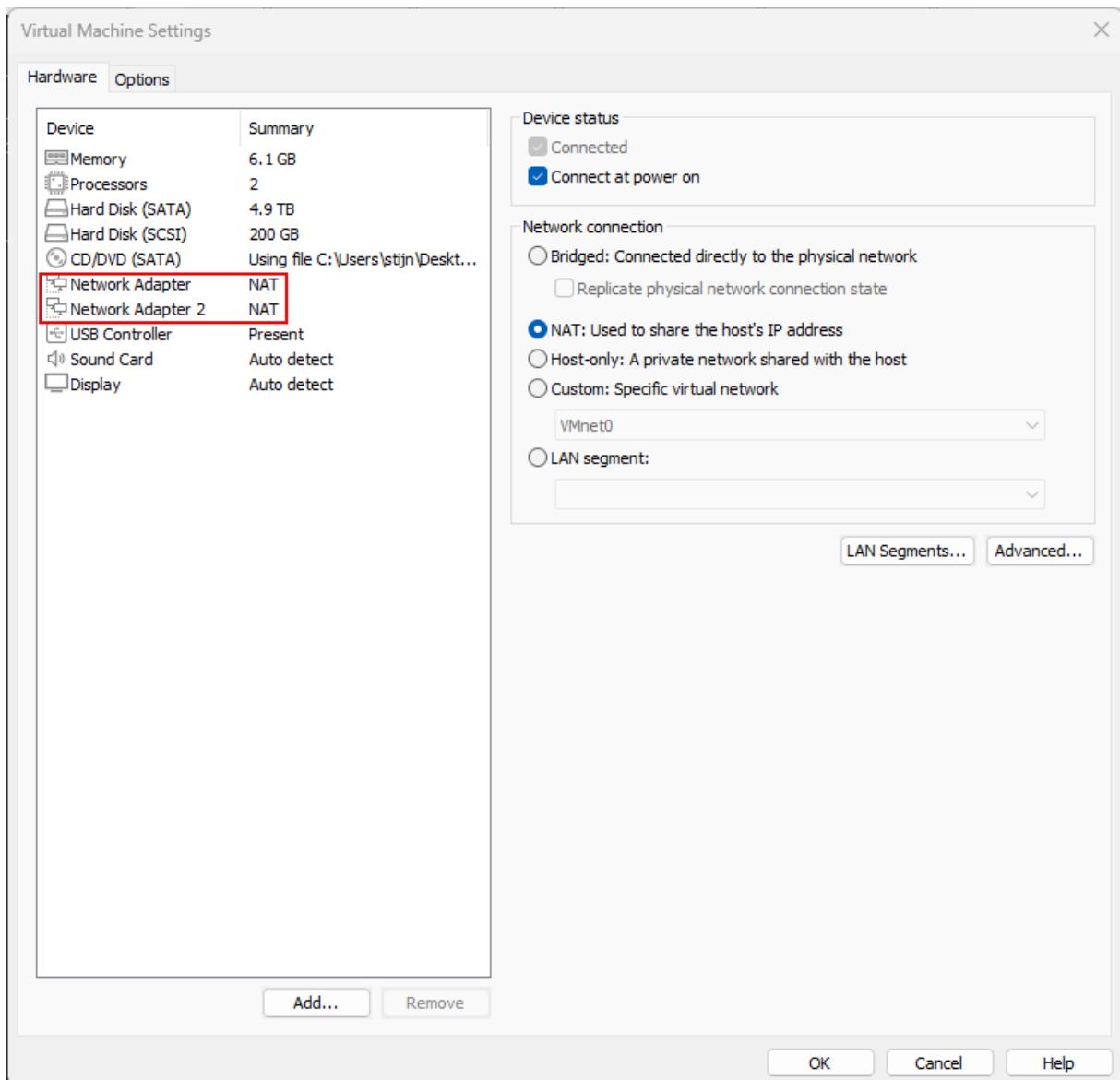
Wij implementeren active-backup.

17.4.4.2 Netwerkkaart toevoegen

Voor bonding heb je een tweede netwerkkaart nodig.

Voeg een tweede netwerkkaart toe die verbonden is met NAT aan de VM PVE zoals hieronder staat weergegeven.





17.4.4.3 Configuratie

Bekijk nu eerst de toegewezen naam van de nieuwe netwerkkaart op je systeem.

```
root@pve:~# ip a
...
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel
    master vmbr0 state UP group default qlen 1000
        link/ether 00:0c:29:c1:bb:43 brd ff:ff:ff:ff:ff:ff
        altname enp2s1
        altname enx000c29c1bb43
```

```

...
177: ens37: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN
group default qlen 1000
    link/ether 00:0c:29:c1:bb:4d brd ff:ff:ff:ff:ff:ff
    altname enp2s5
    altname enx000c29c1bb4d

```

Zoals je ziet is er hier een tweede netwerkkaart toegevoegd. De status is down. Logisch aangezien de netwerkkaart nog niet geconfigureerd is in /etc/network/interfaces.

We stellen nu bond in voor ens33 en ens37 in /etc/network/interfaces.

```

root@pve:~# nano /etc/network/interfaces

# Loopback
auto lo
iface lo inet loopback

# Bond0: active-backup van twee NICs in hetzelfde NAT-subnet
auto bond0
iface bond0 inet manual
    bond-slaves ens33 ens37
    bond-mode active-backup
    bond-miimon 100

# LAN bridge via bond0
auto vmbr0
iface vmbr0 inet static
    address 192.168.112.110/24
    gateway 192.168.112.2
    bridge_ports bond0

```

```
bridge_stp off
```

```
bridge_fd 0
```

Herstart nu de netwerkinterfaces.

```
root@pve:~# ifreload -a
```

Controle interfaces.

```
root@pve:~# ip a
```

```
2: ens33: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc fq_codel master bond0 state UP group default qlen 1000
    link/ether 00:0c:29:c1:bb:43 brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    altname enx000c29c1bb43

172: vmbr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 00:0c:29:c1:bb:43 brd ff:ff:ff:ff:ff:ff
    inet 192.168.112.110/24 scope global vmbr0
        valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fec1:bb43/64 scope link proto kernel_ll
        valid_lft forever preferred_lft forever

177: ens37: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc fq_codel master bond0 state UP group default qlen 1000
    link/ether 00:0c:29:c1:bb:43 brd ff:ff:ff:ff:ff:ff permaddr 00:0c:29:c1:bb:4d
    altname enp2s5
    altname enx000c29c1bb4d

178: bond0: <BROADCAST,MULTICAST,MASTER,UP,LOWER_UP> mtu 1500 qdisc noqueue master vmbr0 state UP group default qlen 1000
    link/ether 00:0c:29:c1:bb:43 brd ff:ff:ff:ff:ff:ff
```

Je ziet:

- ens33 en ens37 zijn slave van bond0.
- bond0 is master van vmbr0
- vmbr0 heeft IP 192.168.112.110/24
- Alle interfaces zijn UP / LOWER_UP

We checken de bond status.

```
root@pve:~# cat /proc/net/bonding/bond0
```

...

Currently Active Slave: ens33

...

We testen connectie met NAT-gateway en met internet

```
root@pve:~# ping -c 1 192.168.112.2
```

```
PING 192.168.112.2 (192.168.112.2) 56(84) bytes of data.
```

```
64 bytes from 192.168.112.2: icmp_seq=1 ttl=128 time=0.107 ms
```

```
--- 192.168.112.2 ping statistics ---
```

```
1 packets transmitted, 1 received, 0% packet loss, time 0ms
```

```
rtt min/avg/max/mdev = 0.107/0.107/0.107/0.000 ms
```

```
root@pve:~# ping -c 1 www.google.be
```

```
PING www.google.be (64.233.184.94) 56(84) bytes of data.
```

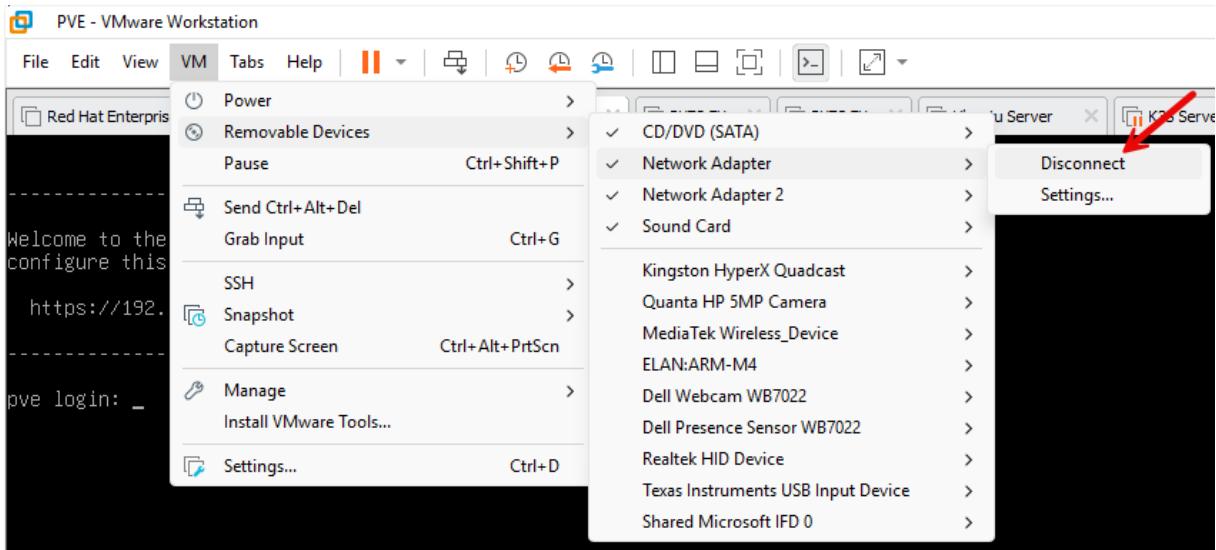
```
64 bytes from wa-in-f94.1e100.net (64.233.184.94): icmp_seq=1  
ttl=128 time=24.6 ms
```

```
--- www.google.be ping statistics ---
```

```
1 packets transmitted, 1 received, 0% packet loss, time 0ms
```

```
rtt min/avg/max/mdev = 24.576/24.576/24.576/0.000 ms
```

We testen nu de failover. We disconnecteren ens33.



We checken terug de bond status.

```
root@pve:~# cat /proc/net/bonding/bond0
```

...

Currently Active Slave: ens37

...

Dat is correct 😊.

We testen terug connectie met NAT-gateway en met internet

```
root@pve:~# ping -c 1 192.168.112.2
```

```
PING 192.168.112.2 (192.168.112.2) 56(84) bytes of data.
```

```
64 bytes from 192.168.112.2: icmp_seq=1 ttl=128 time=0.104 ms
```

```
--- 192.168.112.2 ping statistics ---
```

```
1 packets transmitted, 1 received, 0% packet loss, time 0ms
```

```
rtt min/avg/max/mdev = 0.104/0.104/0.104/0.000 ms
```

```
root@pve:~# ping -c 1 www.google.be
```

```
PING www.google.be (64.233.184.94) 56(84) bytes of data.
```

```
64 bytes from wa-in-f94.1e100.net (64.233.184.94): icmp_seq=1
ttl=128 time=22.4 ms
```

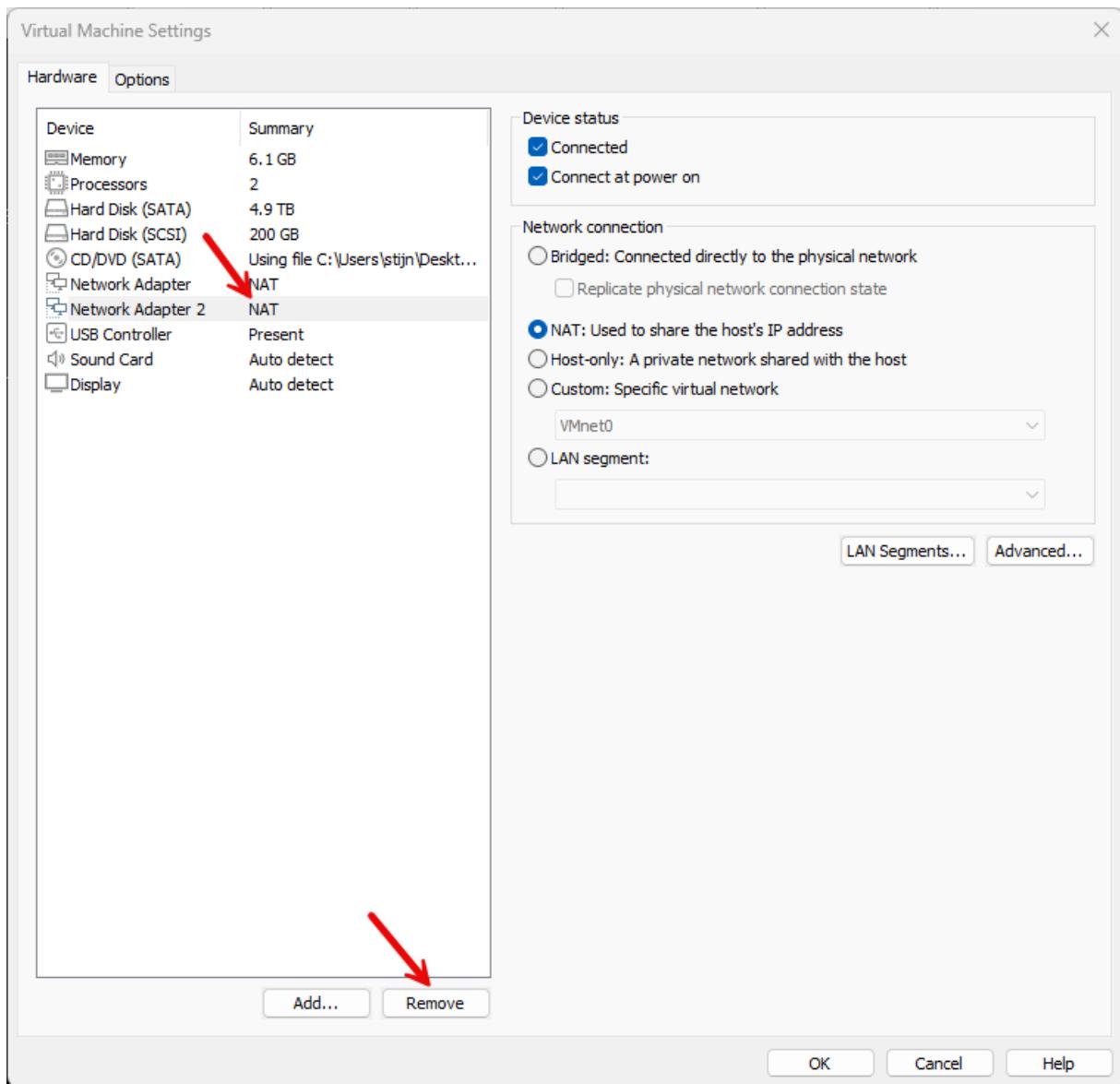
```
--- www.google.be ping statistics ---
```

```
1 packets transmitted, 1 received, 0% packet loss, time 0ms
```

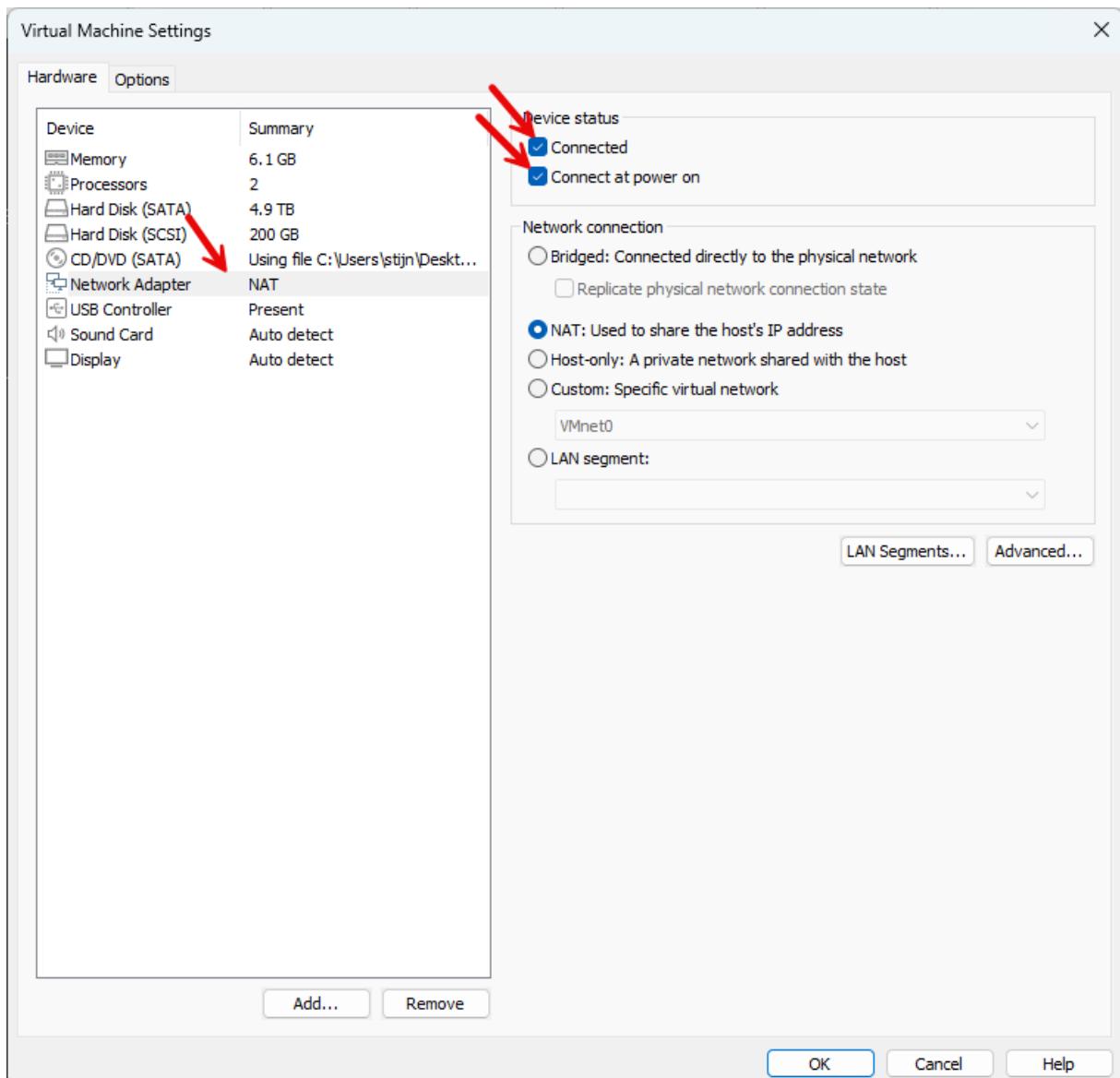
```
rtt min/avg/max/mdev = 22.351/22.351/22.351/0.000 ms
```

17.4.5 Restore netwerkconfiguratie

Verwijder Netwerk Adapter 2 van PVE.



Zet de eerste netwerkadapter ook aan.



Zet je back-up van /etc/network/interfaces terug.

```
root@pve:~# cp interfaces~ /etc/network/interfaces
```

```
root@pve:~# reboot
```

18 Proxmox cloud-images

18.1 Inleiding

Een Cloud-Image is een kant-en-klare virtuele schijf met een basisinstallatie van een besturingssysteem. Het idee is dat je hiermee snel en efficiënt meerdere virtuele machines kunt opzetten zonder telkens handmatig het OS te installeren. Cloud-Images zijn vaak generiek en hardware-onafhankelijk, zodat ze op verschillende virtuele omgevingen kunnen draaien. Belangrijk is dat ze meestal al Cloud-Init bevatten, een systeem dat automatisch een VM configureert bij de eerste keer opstarten.

Cloud-Init zorgt ervoor dat elke virtuele machine unieke instellingen krijgt, ondanks dat ze allemaal dezelfde basisimage gebruiken. Denk aan het aanmaken van gebruikers, instellen van wachtwoorden, configureren van netwerkinterfaces (zoals DHCP of een statisch IP), het installeren van pakketten of het aanpassen van toetsenbordinstellingen. Kortom, Cloud-Init personaliseert de VM op basis van instructies die je vooraf opgeeft.

In Proxmox en vergelijkbare virtualisatieplatforms werkt dit vaak via een speciale Cloud-Init "CD". Dit is een virtuele ISO die de configuratie bevat: gebruikersgegevens, netwerkconfiguratie en eventuele extra commando's die bij de eerste boot moeten worden uitgevoerd. Wanneer de VM voor het eerst opstart, leest Cloud-Init deze ISO en past alle instellingen toe. De basisdisk van de VM blijft onaangetast; de Cloud-Init ISO is puur bedoeld om de configuratie door te voeren.

Het voordeel van deze aanpak is duidelijk: je kunt één enkele basisimage gebruiken om meerdere VM's te maken, terwijl elke VM toch zijn eigen unieke instellingen krijgt. Je hoeft dus nooit een volledige installatie van het besturingssysteem uit te voeren voor elke nieuwe VM. De combinatie van een read-only basisdisk en een dynamische Cloud-Init configuratie maakt het beheer van virtuele machines sneller, consistent en efficiënter.

18.2 Formaat

Cloud-images kunnen in verschillende bestandsformaten worden aangeboden, en elk formaat heeft zijn eigen eigenschappen en voordelen.

Het meest eenvoudige formaat is RAW, een directe en onbewerkte kopie van een schijf. Dit type image biedt de hoogste prestaties omdat er geen extra lagen of optimalisaties worden toegepast, maar het neemt wel meteen de volledige schijfgrootte in beslag.

Een ander veelvoorkomend formaat is IMG. Dit is eigenlijk geen vast schijfformaat, maar eerder een container die afhankelijk van de aanbieder verschillende types kan bevatten. In de praktijk is een IMG-bestand meestal vergelijkbaar met RAW, al kan het soms ook een ander intern formaat bevatten. Virtualisatiesoftware zoals Proxmox behandelt een IMG doorgaans alsof het een RAW-image is.

Het meest geavanceerde formaat is QCOW2, dat speciaal ontworpen werd voor virtuele omgevingen. QCOW2-bestanden groeien dynamisch mee met de gebruikte data en bevatten functies zoals snapshots, compressie en encryptie. Dat maakt ze bijzonder geschikt voor cloud-images en moderne virtuele infrastructuren, al leveren ze door die extra functionaliteit een heel klein beetje in op pure snelheid vergeleken met RAW.

Kort samengevat: RAW biedt maximale prestaties maar is groot in opslag, IMG is flexibel en meestal RAW-achtig, en QCOW2 is het slimste en meest veelzijdige formaat dankzij ondersteuning voor functies die in cloud-omgevingen erg handig zijn.

Formaat	Grootte	Performance	Snapshots	Wat het is
RAW	Altijd groot	snelst	Nee	Kale schijf
IMG	Verschilt	Verschilt	Verschilt	Container voor RAW/QCOW2 (afhankelijk van maker)
QCOW2	Klein (dynamisch)	Relatief snel	Ja	Slim virtueel schijfformaat

18.3 Dowloadlinks

Hieronder vind je de rechtstreekse downloadlinks van cloud-images voor veelgebruikte distro's.

- RedHat based
 - o Oracle Linux 10
https://yum.oracle.com/templates/OracleLinux/OL10/u0/x86_64/OL10U0_x86_64-kvm-b266.qcow2
 - o Rocky Linux 10
https://dl.rockylinux.org/pub/rocky/10/images/x86_64/Rocky-10-GenericCloud-Base.latest.x86_64.qcow2
 - o AlmaLinux 10
https://repo.almalinux.org/almalinux/10/cloud/x86_64/images/AlmaLinux-10-GenericCloud-latest.x86_64.qcow2
- Debian based
 - o Debian 12
<https://cloud.debian.org/images/cloud/bookworm/latest/debian-12-genericcloud-amd64.qcow2>
 - o Ubuntu 25.04
[Ubuntu 25.04 \(Plucky Puffin\) release \[20251122\]](https://ubuntu.com/download/desktop/archives/25.04)

18.4 Voorbeelden

18.4.1 Via GUI

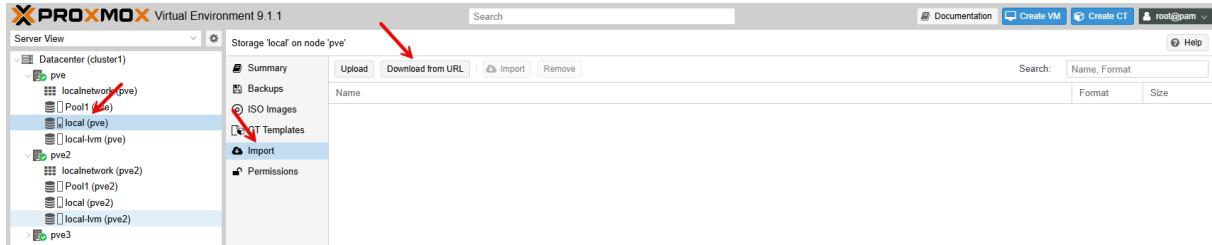
18.4.1.1 Importeren cloud based image

Voor een Cloud-Image in Proxmox kan je best gebruik maken van Import en niet van ISO Images om de cloud image op te slaan.

In Proxmox is “Import” een onderdeel dat bij bepaalde storage-types automatisch verschijnt. Zoals je weet zijn er in Proxmox standaard 2 soorten opslag geconfigureerd: local en local-lvm.

Import is aanwezig bij een opslag van het type directory: dat is dus local.

We zullen een image uploaden naar de standaard lokale opslag op pve.



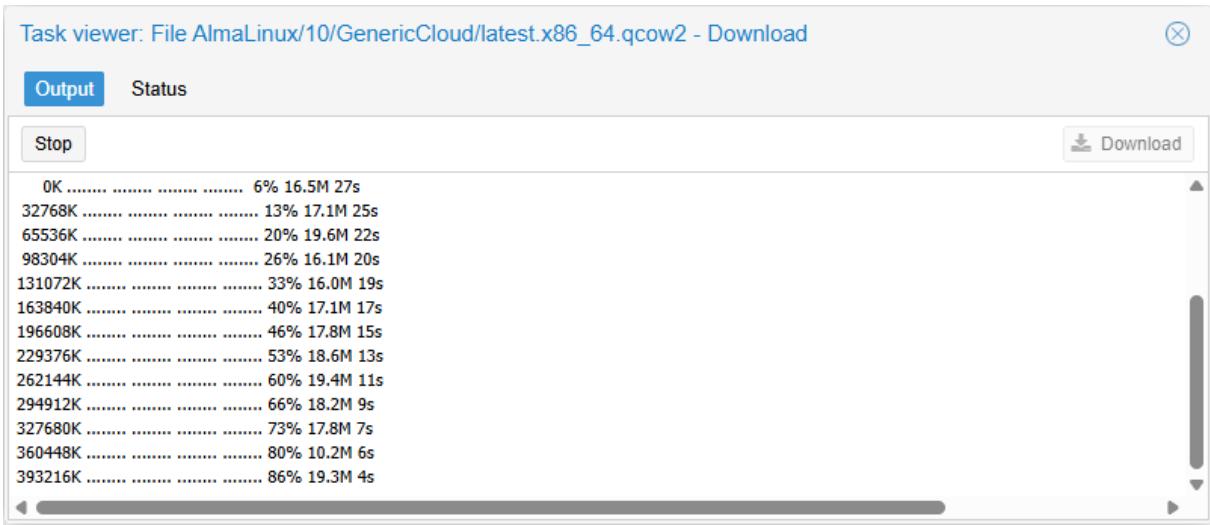
We zullen gebruik maken van de cloudimage van AlmaLinux.

Vul https://repo.almalinux.org/almalinux/10/cloud/x86_64/images/AlmaLinux-10-GenericCloud-latest.x86_64.qcow2 in bij het venster om te downloaden. Klik erna op Query URL.

The screenshot shows a 'Download from URL' dialog box. It has fields for 'URL' (containing 'https://repo.almalinux.org/almalinux/10/cloud/x86_64/images/Alm...'), 'File name:' (containing 'AlmaLinux-10-GenericCloud-latest.x86_64.qcow2'), and other options like 'Hash algorithm: None', 'Checksum: none', 'Verify certificates: checked', and 'Advanced' with a checked checkbox. A red arrow points from the 'Query URL' button to the 'File name:' field. Another red arrow points from the 'File name:' field to the 'Download' button.

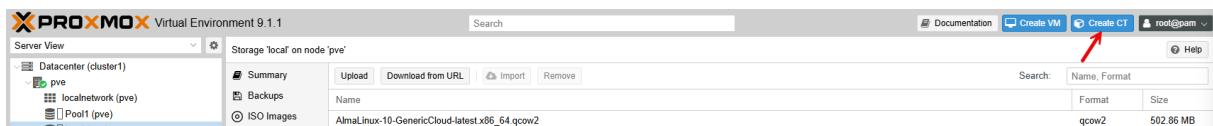
Je ziet de naam bij “File name:” verschijnen na het klikken op Query URL. Klik erna op Download.

Het image wordt dan gedownload.



18.4.1.2 VM aanmaken en instellen

Klik rechtsboven nu op Create VM.



Maak nu een VM aan met de instellingen die je hieronder ziet. We zullen de VM uitvoeren op PVE.

Create: Virtual Machine

General OS System Disks CPU Memory Network Confirm

Node: pve
VM ID: 114
Name: Alma

Add to HA:

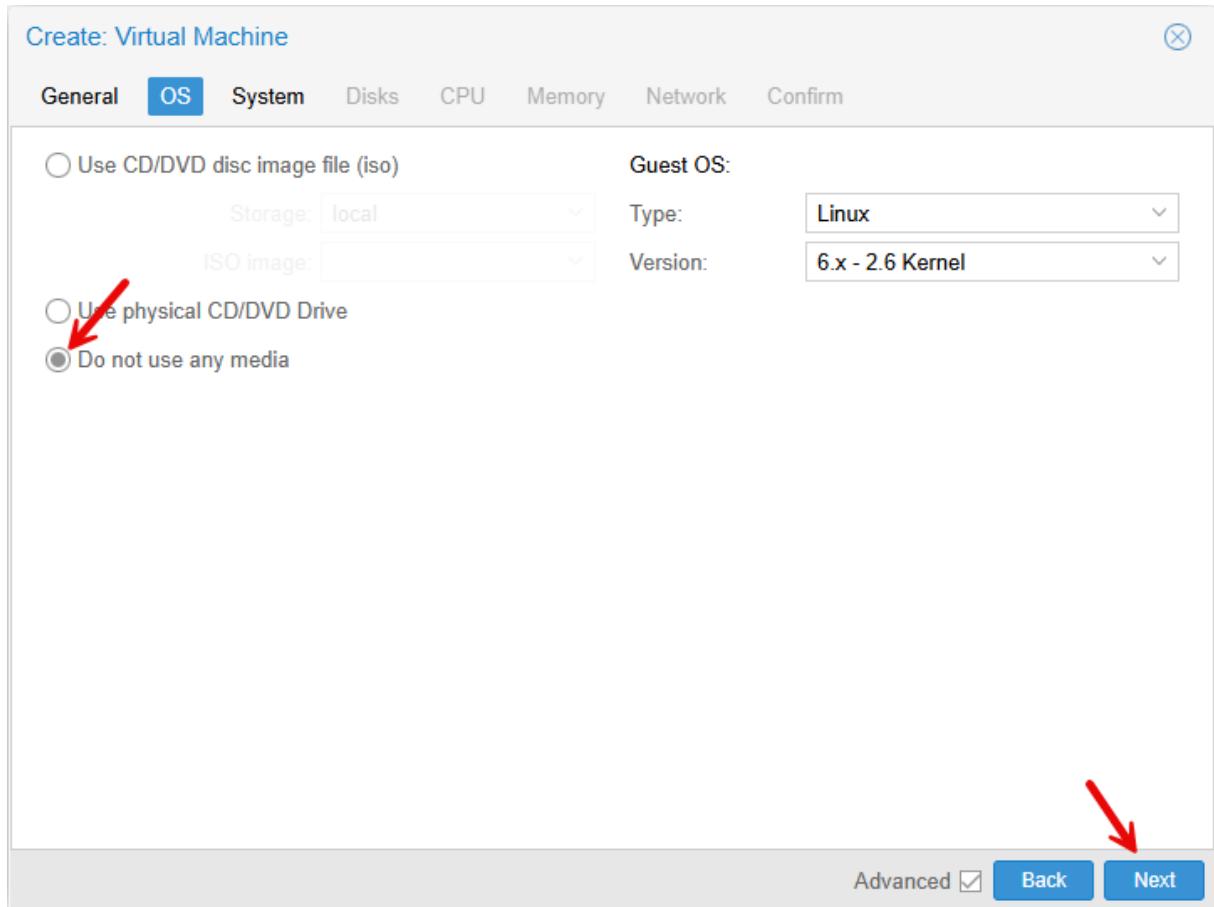
Start at boot: Start/Shutdown order: any

Startup delay: default
Shutdown timeout: default

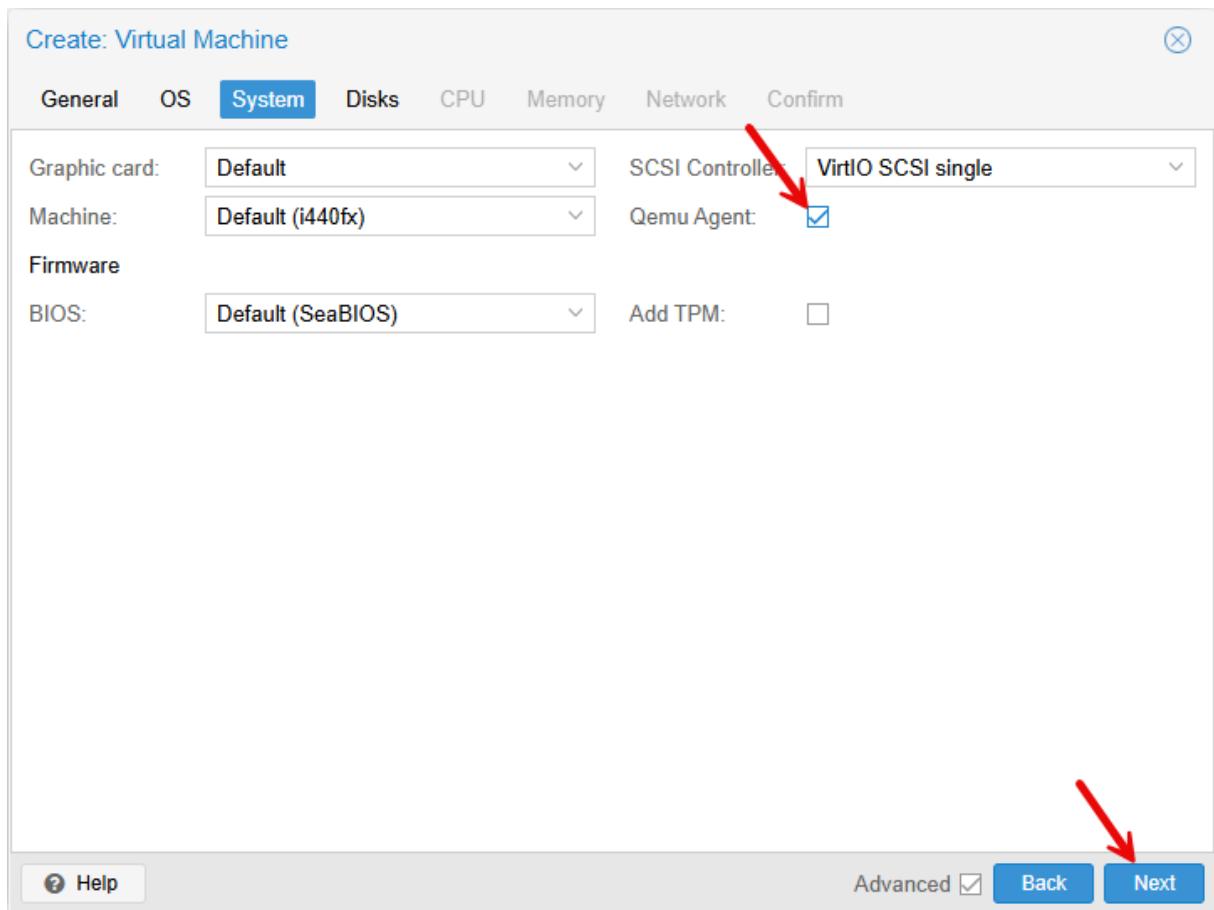
Tags
No Tags +

Help Advanced Back Next

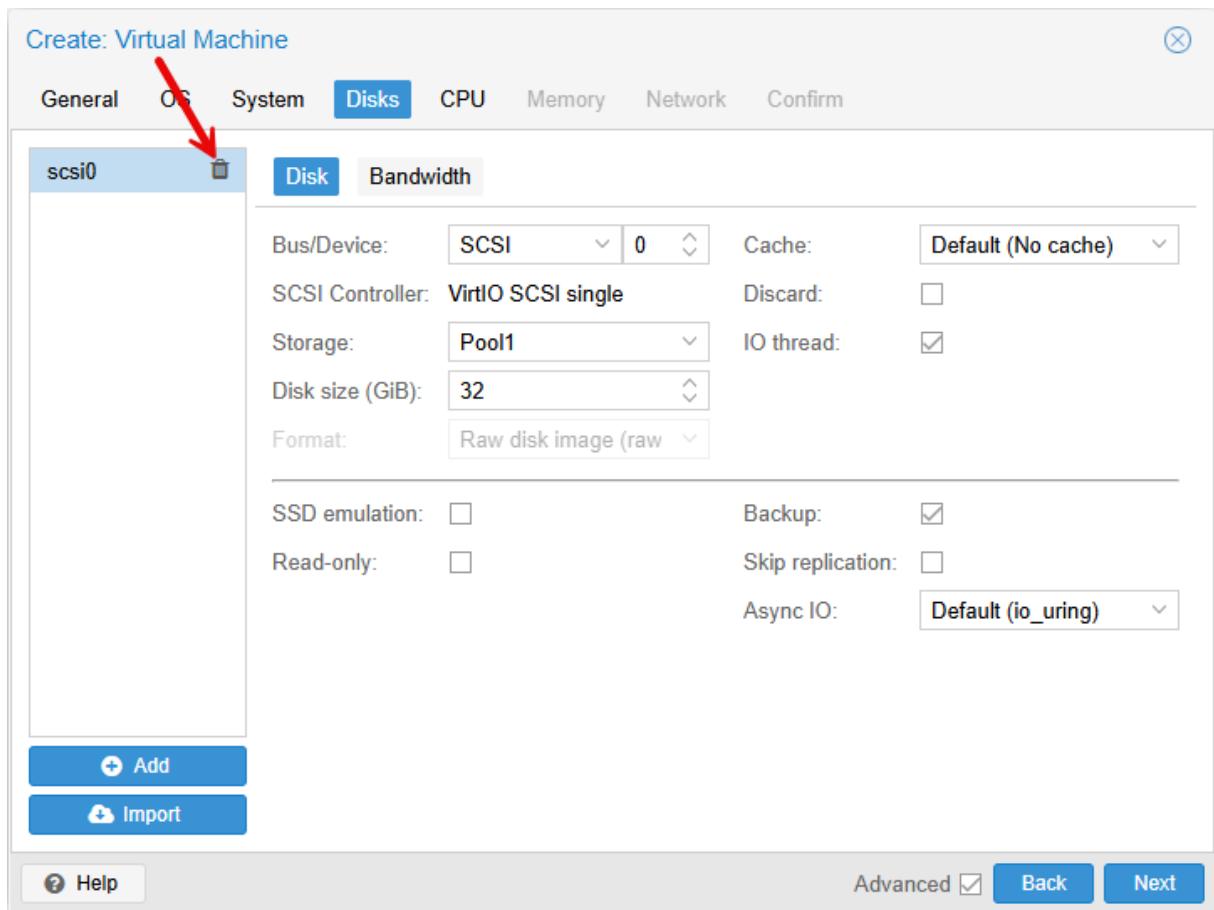
In een volgend venster kies je ervoor om geen media toe te voegen aangezien je gaat werken met een cloud image. Bij Guest OS kies je voor recente kernelversie van Linux.



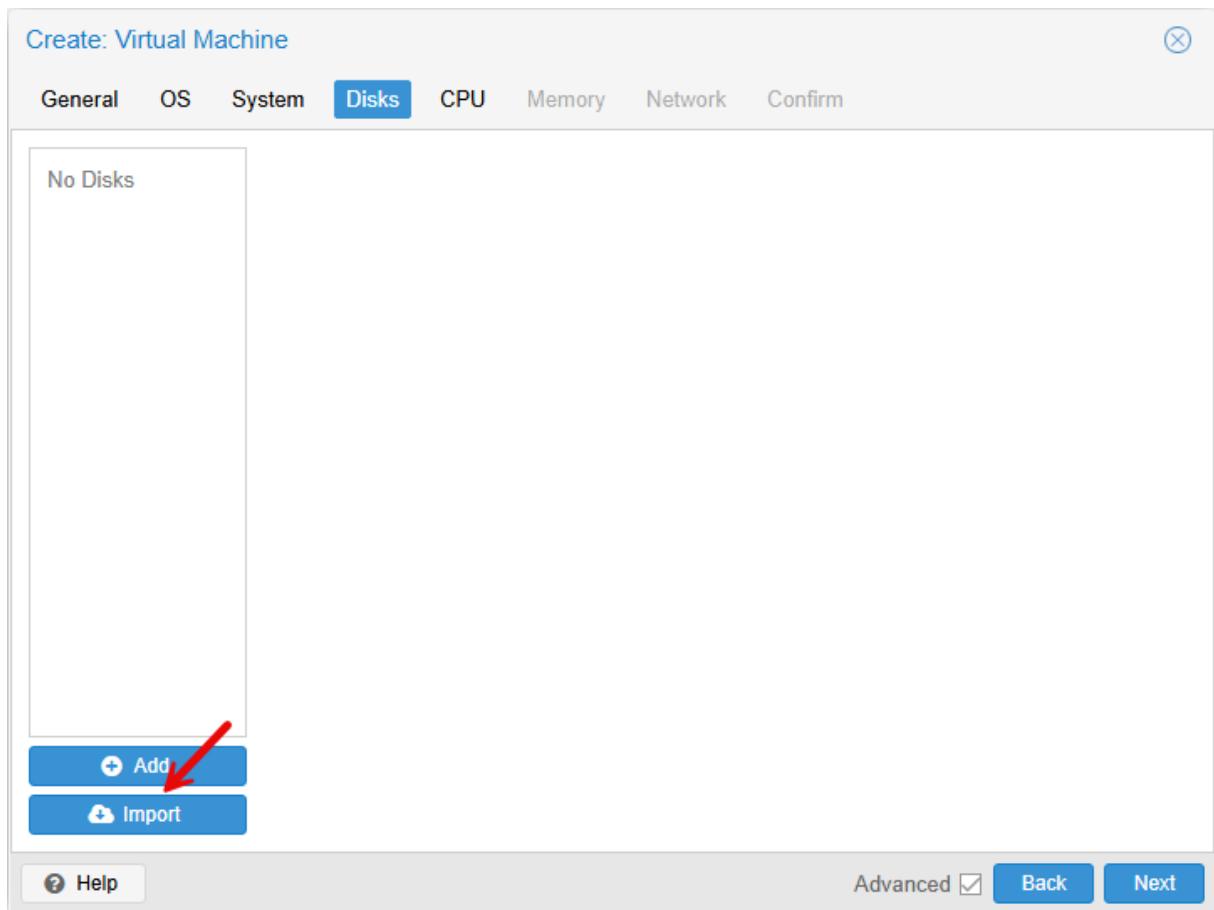
In het volgende venter zet je een vinkje voor QEMU guest agent. De QEMU Guest Agent een klein programma dat binnenin een virtuele machine draait en communiceert met de hypervisor. Het is een "brug" tussen de VM en de host: het laat de host informatie opvragen en commando's uitvoeren binnen de VM op een veilige manier. Uiteraard moet daarvoor ook software in de VM geïnstalleerd zijn! We komen daar later op terug. De andere instellingen laat je ongemoeid.



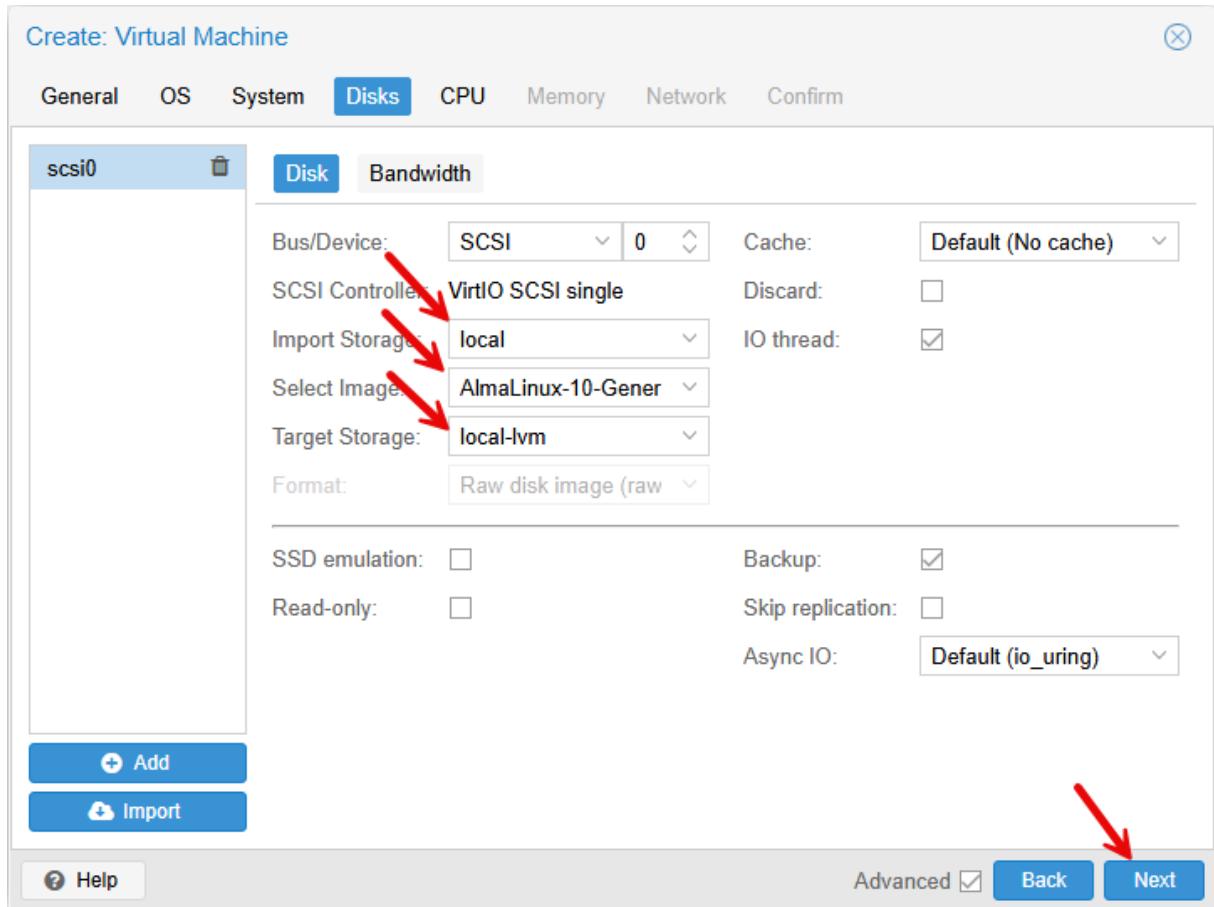
De schijf die nu wordt voorgesteld verwijder je.



Klik nu op Import.



Selecteer het image dat we geüpload hebben van de storage local. We gaan alles lokaal opslaan op PVE. Kies dan ook voor local-lvm bij Target Storage.



Het tabblad CPU kan nogal tricky zijn om in te stellen, afhankelijk van het cloud image.

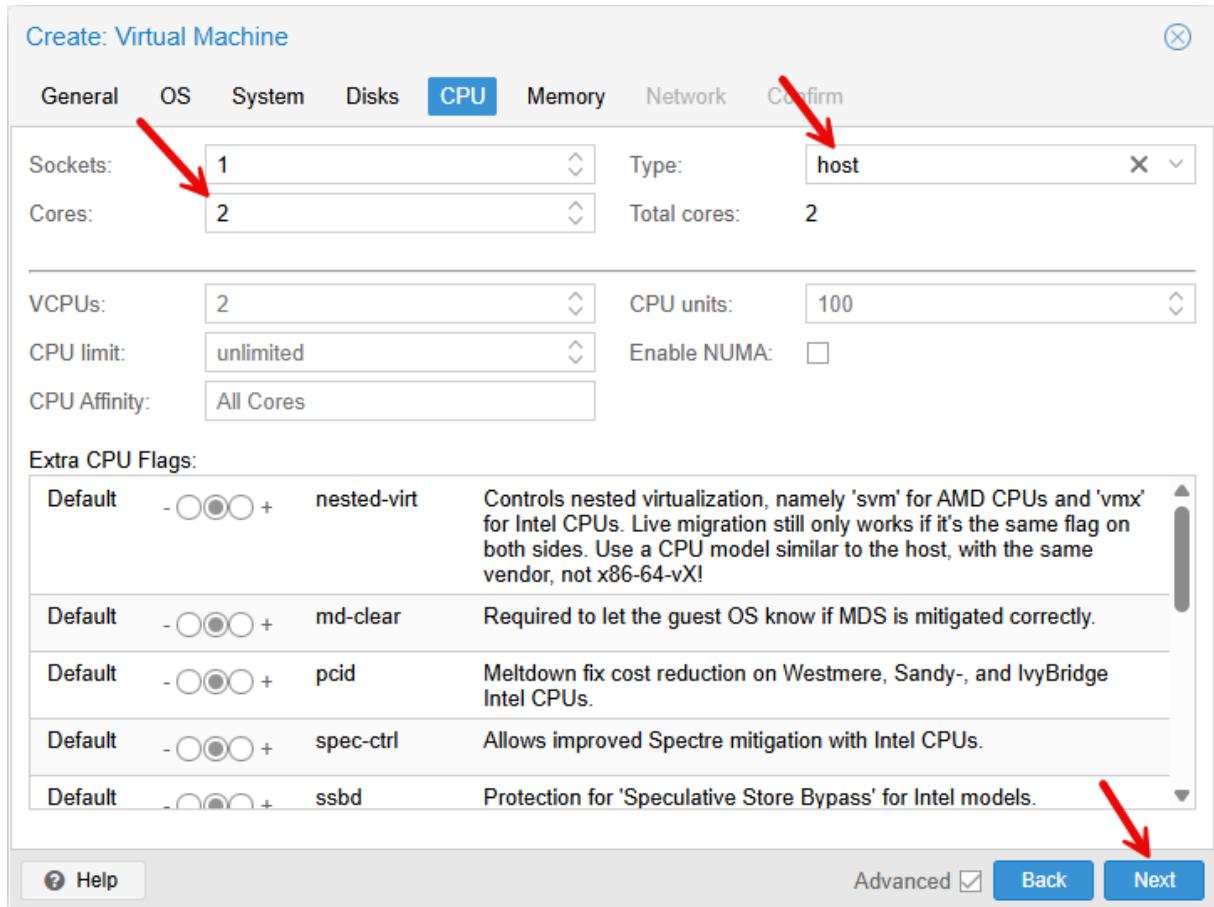
Het CPU-type bepaalt welk virtueel CPU-model de VM te zien krijgt. Dit beïnvloedt:

- Welke instructiesets beschikbaar zijn (zoals AES, AVX, SSE)
 - o AES is een encryptiestandaard, SSE en AVX zijn CPU-instructiesets voor snelle vector- en floating-point berekeningen.
- Hoe compatibel de VM is met migratie tussen hosts.
- Welke beveiligingen tegen CPU-fouten zoals Spectre en Meltdown aanstaan.

Voorbeelden:

- X86-64-v2-AES: modern, met AES-instructies
- Host: exact zoals de fysieke CPU van je Proxmox-node
- Kvm64: zeer compatibel, maar beperkt qua features

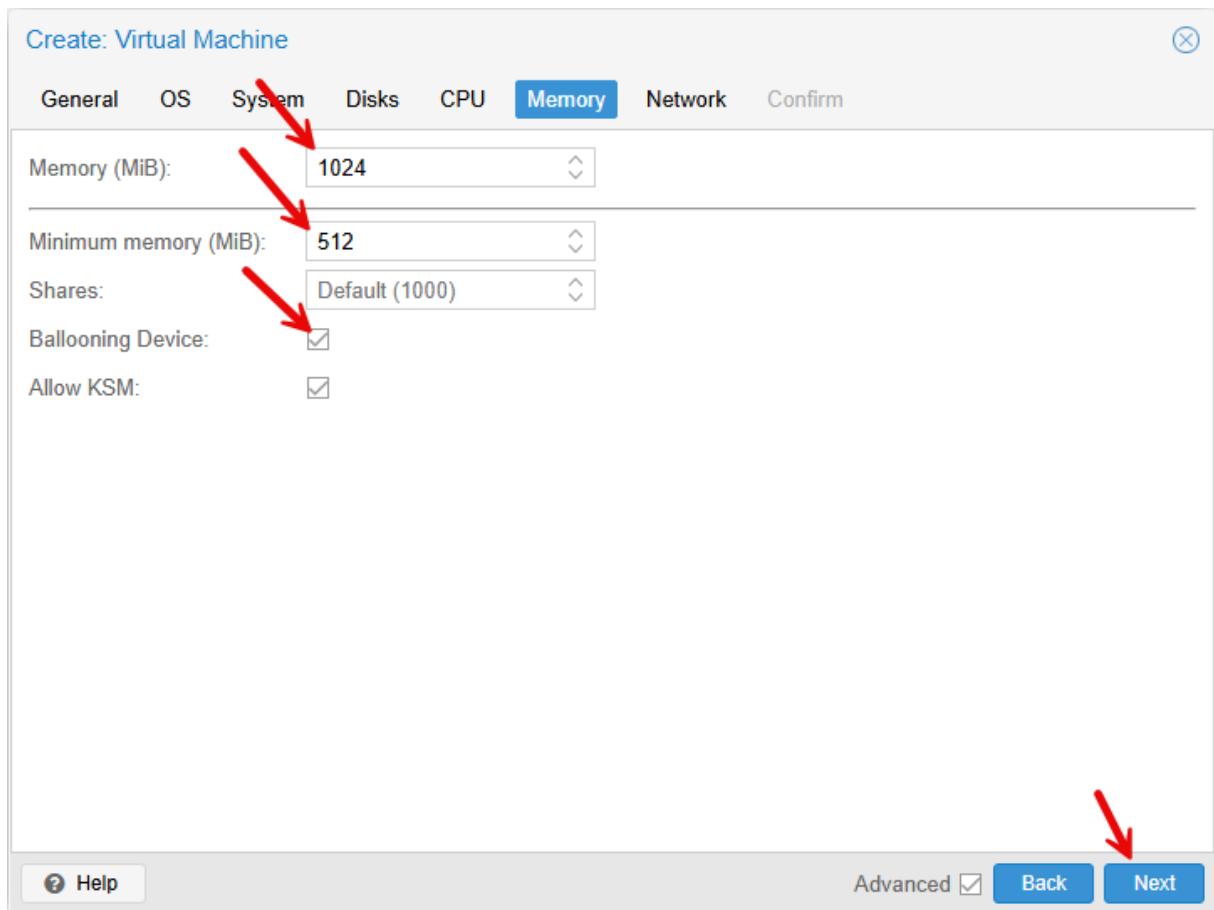
Indien mogelijk: kies host. Hier lukt dat met deze cloud image 😊.



In volgende venster stel je Ballooning in. Dit is een techniek waarmee je het geheugengebruik van een VM dynamisch kunt aanpassen — zonder de VM te herstarten. Het is vooral handig in omgevingen waar je meerdere VM's draait en het RAM slim wilt verdelen.

Stel onderstaande in:

- Memory: 1024 MiB: dit is het maximum dat de VM mag gebruiken.
- Minimum memory: 512 MiB: dit is het laagste dat de host mag terugvragen via ballooning.
- Ballooning device: aangevinkt
- Allow KSM: aangevinkt: Kernel Samepage Merging, bespaart extra RAM door identieke geheugenpagina's te combineren.



In het volgende venster stel je je netwerk in.

Model "VirtIO" is snel en geoptimaliseerd voor virtualisatie. We laten de standaard staan.

Create: Virtual Machine X

General OS System Disks CPU Memory Network **Network** Confirm

No network device

Bridge: Model:

VLAN Tag: MAC address:

Firewall:

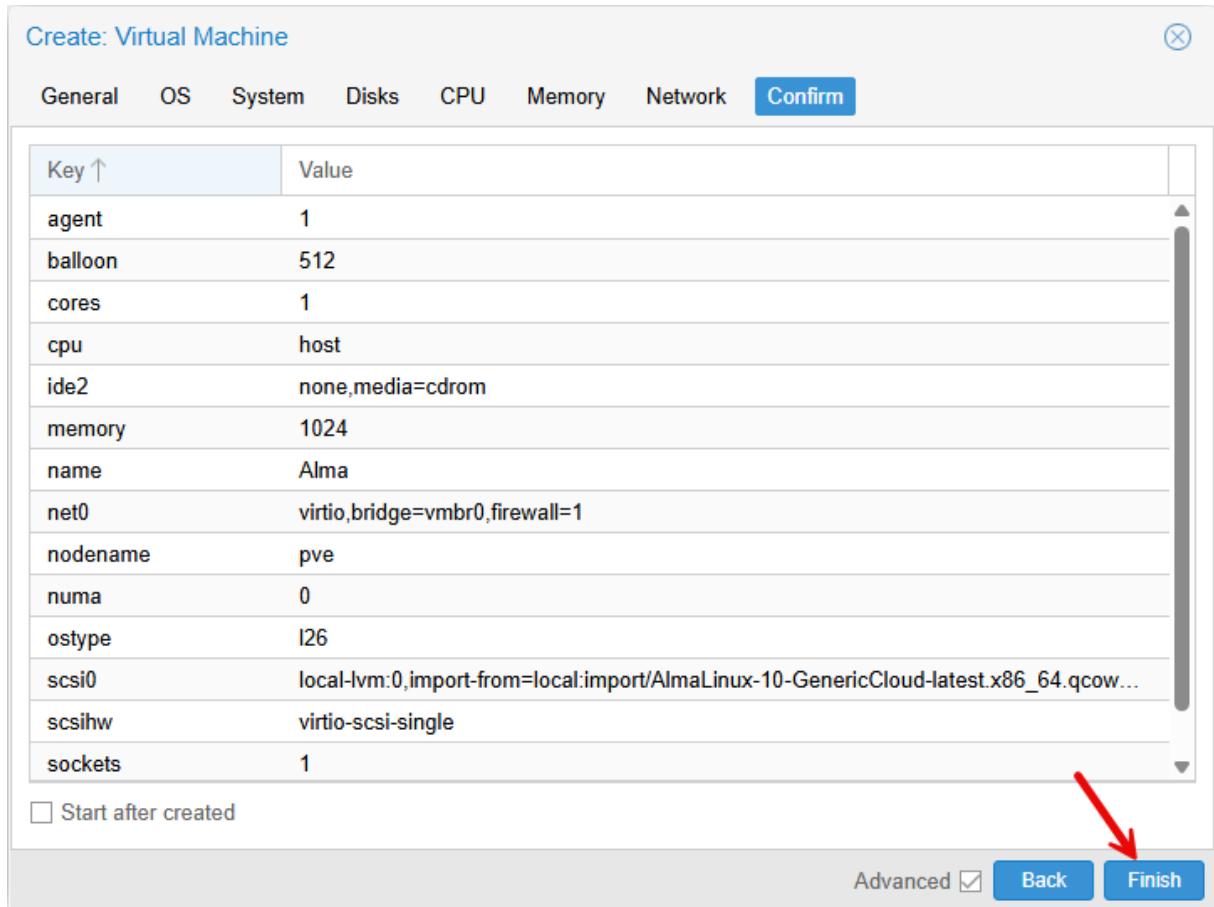
Disconnect: Rate limit (MB/s):

MTU: Multiqueue:

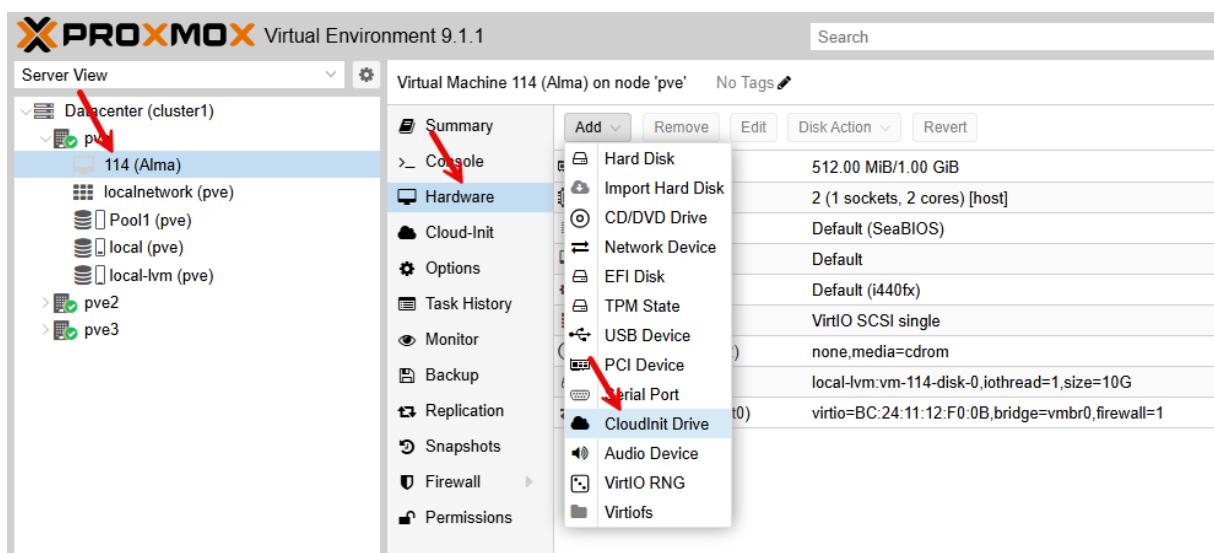
Help Advanced Back Next

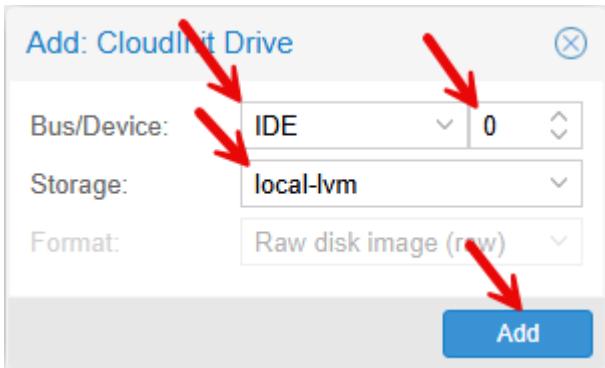


Kijk nu je instellingen na.



We voegen nu een CloudInit Drive toe voor de bijkomende instellingen van het cloud image.





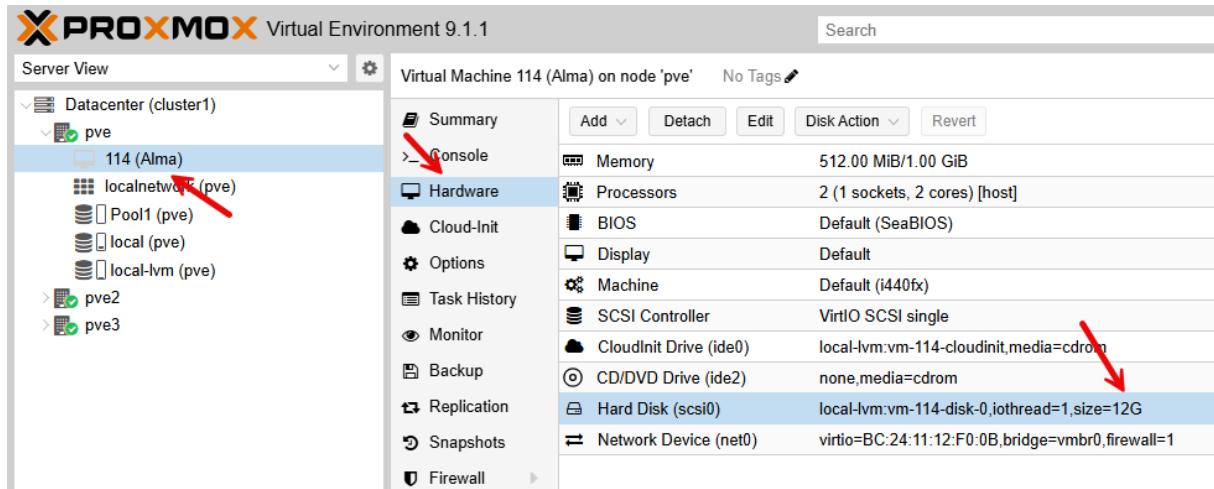
We resizen nu de harde schijf.

We maken de virtuele schijf nu 2 GB groter als voorbeeld.

Opgelet: dat is een structurele wijziging aan de virtuele disk, en dus hard provisioning!



Hieronder zie je dat de virtuele schijf groter is geworden.



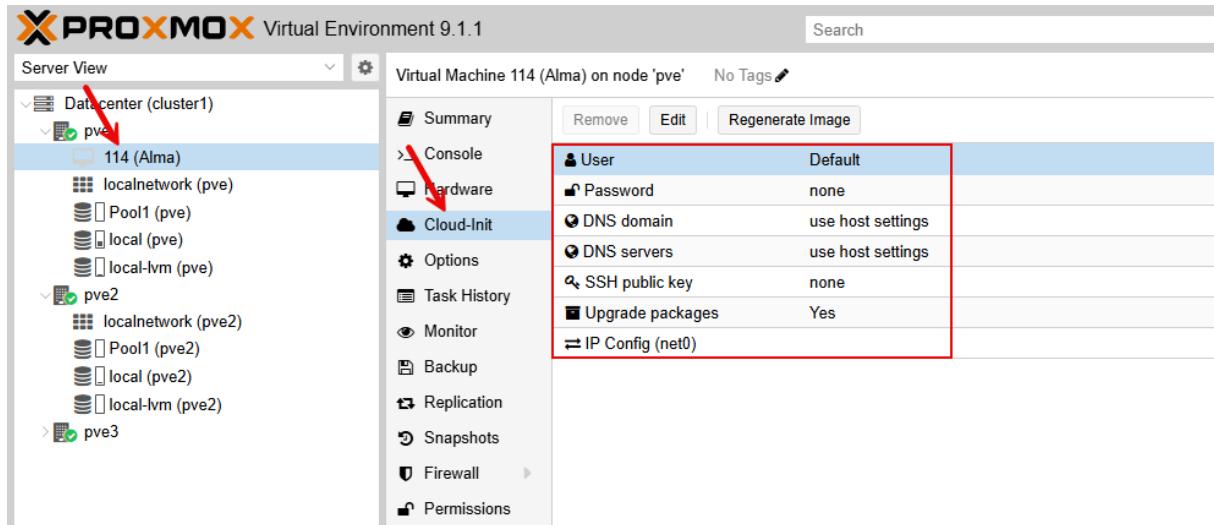
We hebben zo-even een Cloud-Init drive toegevoegd.

Zodra je in het Cloud-Init tabblad van de VM een gebruiker, wachtwoord, IP of DNS invult, schrijft Proxmox die waarden weg naar drie bestanden op die drive:

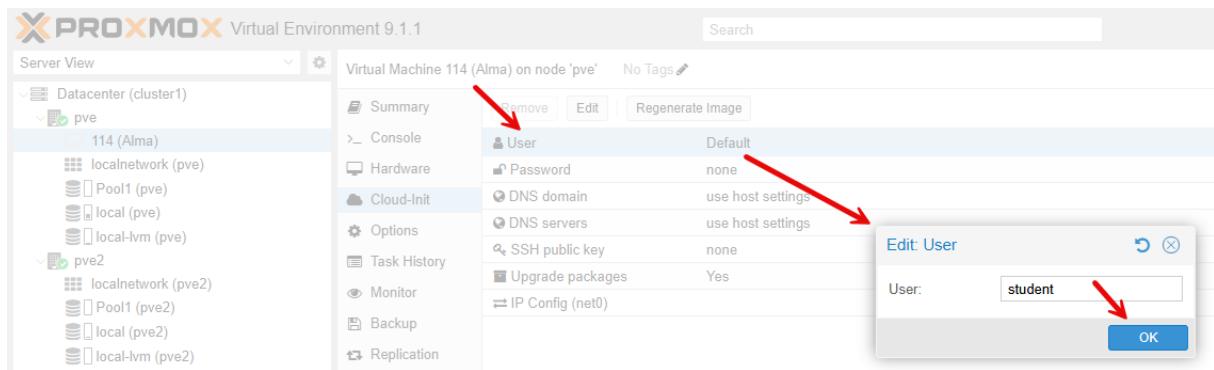
- Gebruikers, wachtwoord, SSH-sleutels, packages
- DHCP of statische IP-instellingen
- VM-ID, hostname, enz.

Opgelat: niet alle instellingen zijn beschikbaar via de GUI.

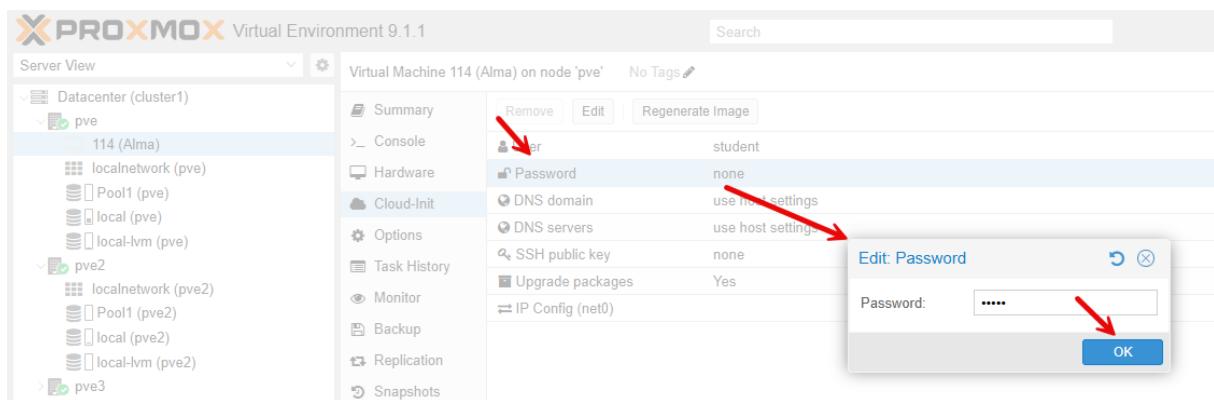
Instelling	Beschrijving
User	De standaard Cloud-Init gebruiker (ciuser)
Password	Het wachtwoord voor die gebruiker (cipassword)
DNS Domain	Domeinnaam voor de VM
DNS Servers	Lijst van DNS-servers die de VM gebruikt
SSH Public Key	OpenSSH sleutel voor de gebruiker
Upgrade Packages	Automatische OS-upgrades aan/uit
IP Config	DHCP of statisch IP instellen (ipconfig0)



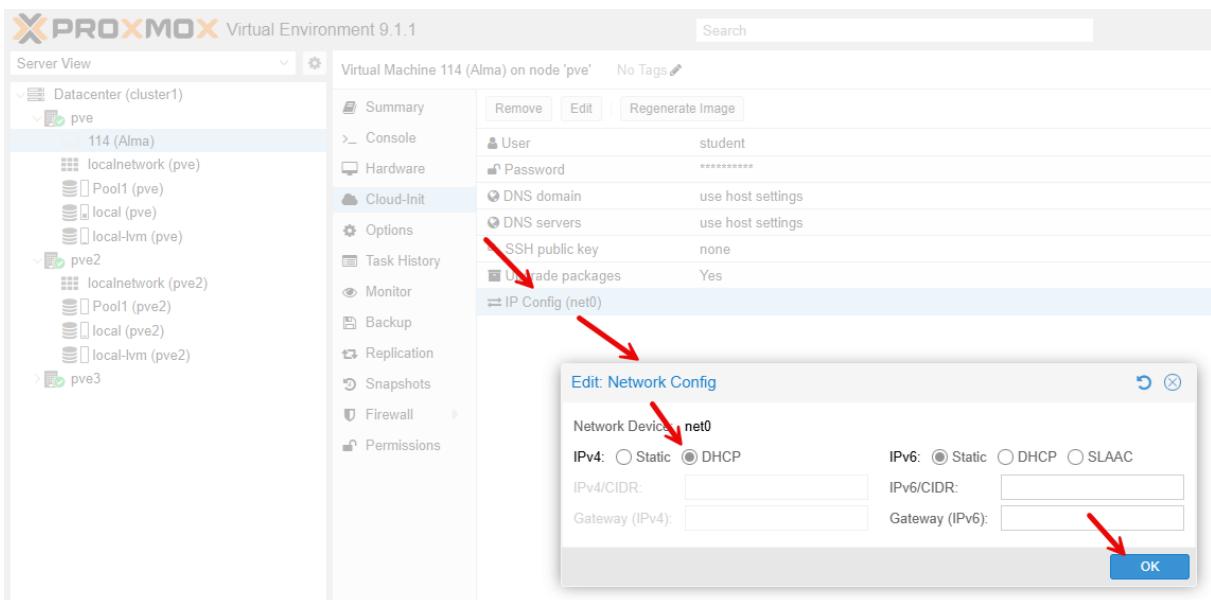
We passen de gebruiker aan.



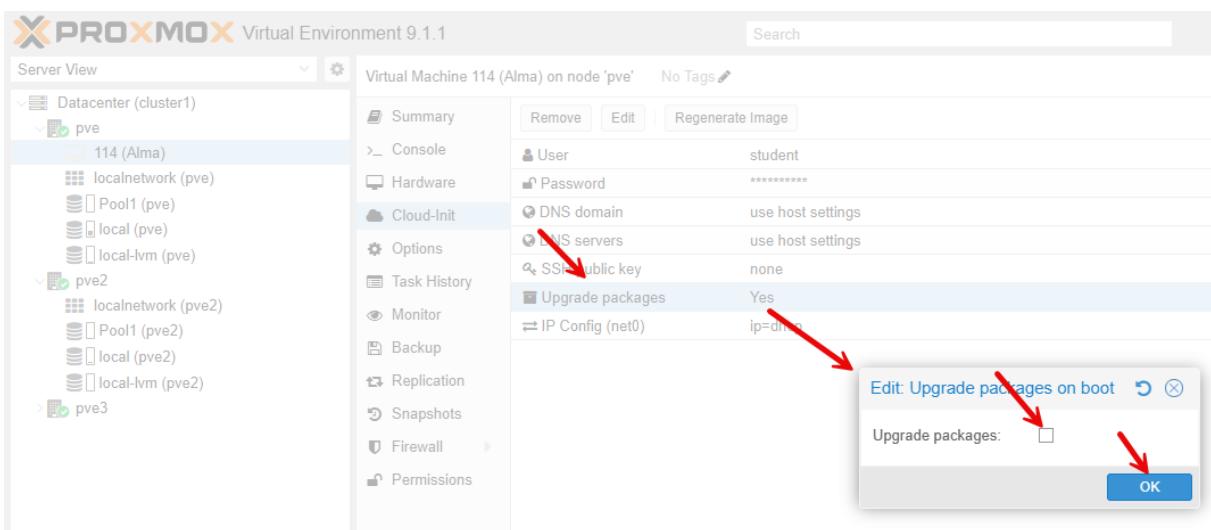
We stellen het wachtwoord in: PXL.



We kiezen ook voor verkrijgen IPv4-nummer via DHCP.



Om tijd te winnen zullen we ervoor zorgen dat er geen packages worden geüpdated.



De knop "Regenerate Image" in de Cloud-Init context van Proxmox betekent eigenlijk: maak de Cloud-Init drive opnieuw aan met de laatste instellingen die je in de GUI hebt ingevuld. We doen dat nu dus ook.

Virtual Machine 114 (Alma) on node 'pve' No Tags

Summary	Remove	Edit	Regenerate Image
Console	User Student		
Hardware	Password *****		
Cloud-Init	DNS domain	use host settings	
Options	DNS servers	use host settings	
Task History	SSH public key	none	
Monitor	Upgrade packages	No	
Backup	IP Config (net0)	ip=dhcp	

18.4.1.3 VM starten

Virtual Machine 114 (Alma) on node 'pve' No Tags

Start Now

Guest not running

18.4.1.4 Inloggen

Log nu in met je naam en het wachtwoord dat je gekozen hebt.

Opgelet: je toetsenbord staat op QWERTY! Gelukkig hebben we voor het wachtwoord "PXL" gekozen, want op zowel QWERTY- als AZERTY-toetsenborden liggen deze letters op dezelfde plek.

AlmaLinux 10.1 (Heliotrope Lion)
Kernel 6.12.0-124.8.1.el10_1.x86_64 on an x86_64
Alma login: student
Password:
[student@Alma ~]\$ _

```
[student@Alma ~]sudo localectl set-keymap be
```

Het package qemu-guest-agent is standard al geïnstalleerd in deze image.

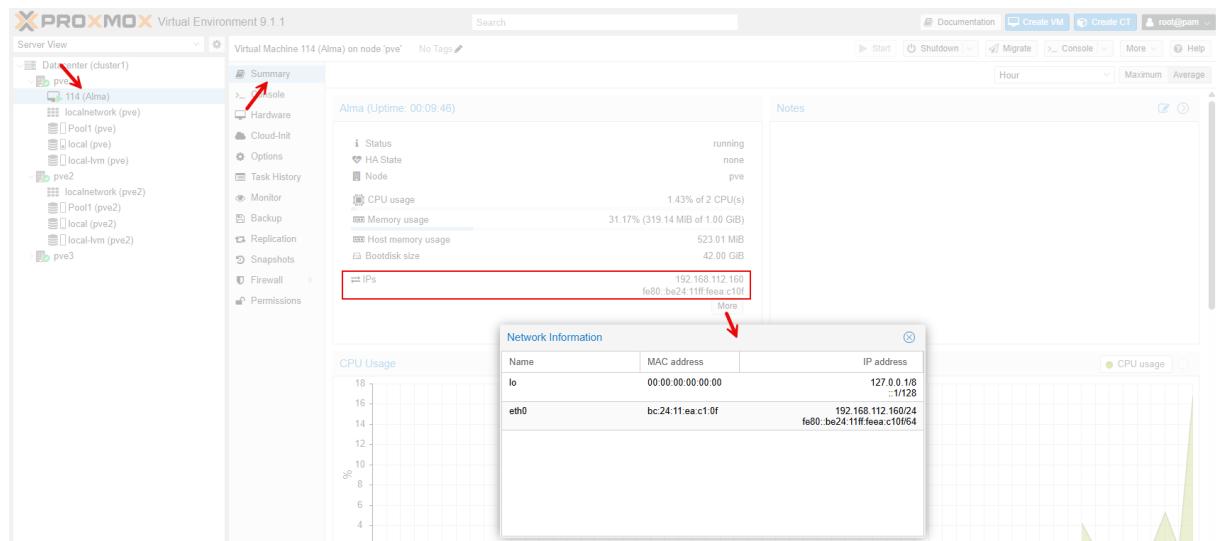
We updaten het package.

```
[student@Alma ~]sudo dnf install qemu-guest-agent -y
```

```
AlmaLinux 10.1 (Heliotrope Lion)
Kernel 6.12.0-124.8.1.el10_1.x86_64 on an x86_64

Alma login: student
Password:
[student@Alma ~]$ sudo localectl set-keymap be
[student@Alma ~]$ sudo dnf install qemu-guest-agent -y
Almalinux 10 - AppStream
Almalinux 10 - BaseOS
Almalinux 10 - CRB
Almalinux 10 - Extras
Package qemu-guest-agent-10:10.0.0-12.el10.alma.1.x86_64 is already installed.
Dependencies resolved.
=====
| Package           | Architecture | Version          | Repository | Size |
| Upgrading:       |              |                  |            |       |
| qemu-guest-agent | x86_64      | 10:10.0.0-14.el10_1.alma.1 | appstream | 449 k |
|                   |              |                  |            |       |
| Transaction Summary |              |                  |            |       |
| Upgrading 1 Package |              |                  |            |       |
| Total download size: 449 k |              |                  |            |       |
| Downloading Packages: |              |                  |            |       |
| qemu-guest-agent-10.0.0-14.el10_1.alma.1.x86_64.rpm | 2.0 MB/s | 449 kB | 00:00 |
| Total |              |                  |            |       |
| Running transaction check |              |                  |            |       |
| Transaction check succeeded. |              |                  |            |       |
| Running transaction test |              |                  |            |       |
| Transaction test succeeded. |              |                  |            |       |
| Running transaction |              |                  |            |       |
|   Preparing : |              |                  |            |       |
|     Upgrading : qemu-guest-agent-10:10.0.0-14.el10_1.alma.1.x86_64 | 1/1 |
|     Running scriptlet: qemu-guest-agent-10:10.0.0-14.el10_1.alma.1.x86_64 | 1/2 |
|     Running scriptlet: qemu-guest-agent-10:10.0.0-12.el10.alma.1.x86_64 | 1/2 |
|     Cleanup : qemu-guest-agent-10:10.0.0-12.el10.alma.1.x86_64 | 2/2 |
|     Running scriptlet: qemu-guest-agent-10:10.0.0-12.el10.alma.1.x86_64 | 2/2 |
|     [ 420.514690] systemd-rc-local-generator[1425]: /etc/rc.d/rc.local is not marked executable, skipping. | 2/2 |
| Upgraded: |              |                  |            |       |
|   qemu-guest-agent-10:10.0.0-14.el10_1.alma.1.x86_64 | 1/1 |
| Complete! |              |                  |            |       |
[student@Alma ~]$ _
```

Zonder QEMU Guest Agent ziet de GUI IP vaak niet. Bij ons zie je die nu uiteraard wel.



18.4.2 Via CLI

18.4.2.1 VM aanmaken en instellen

Verwijder de VM van Alma Linux om ruimte vrij te maken!

We zullen gebruik maken van het cloud image van Oracle Linux voor dit voorbeeld.

Ga naar de import-map en download het image .

```
root@pve:~# cd /var/lib/vz/import/  
root@pve:/var/lib/vz/import# wget  
https://yum.oracle.com/templates/OracleLinux/OL10/u0/x86_64/OL10  
U0_x86_64-kvm-b266.qcow2  
root@pve:/var/lib/vz/import# cd
```

Maak de VM aan (Naam: Oracle, ID: 115, 1GB Ram, Qemu Agent aan)

```
root@pve:~# qm create 115 --name "Oracle" --memory 1024 --ostype  
l26 --net0 virtio,bridge=vmbr0,firewall=1 --agent 1 --scsihw  
virtio-scsi-single --sockets 1 --cores 1 --cpu max --vcpus 1 --  
cpuunits 100 --cpulimit 0
```

Zoals je ziet heb ik hier cpu op max gezet i.p.v. host. Het werkte op mijn systeem niet met host....

Voeg de Cloud-Init drive toe op ide2 bijvoorbeeld.

```
root@pve:~# qm set 115 --ide2 local-lvm:cloudinit,media=cdrom
```

Importeer de gedownloade disk naar local-lvm.

```
root@pve:~# qm importdisk 115 /var/lib/vz/import/OL10U0_x86_64-  
kvm-b266.qcow2 local-lvm
```

Koppel de disk aan de VM als scsi0 en zet Discard aan

```
root@pve:~# qm set 115 --scsi0 local-lvm:vm-115-disk-  
0,discard=on
```

Vergroot de disk met 2GB

```
root@pve:~# qm resize 115 scsi0 +2G  
Size of logical volume pve/vm-115-disk-0 changed from 37.00 GiB  
(9472 extents) to 39.00 GiB (9984 extents).
```

```
Logical volume pve/vm-115-disk-0 successfully resized.
```

Configureer Cloud-Init (User: student, DHCP, geen upgrade).

```
root@pve:~# qm set 115 --ciuser student --cipassword "PXL" --
ipconfig0 ip=dhcp --ciupgrade 0

update VM 115: --cipassword <hidden> --ciupgrade 0 --ciuser student
--ipconfig0 ip=dhcp
```

Genereer nu opnieuw de Cloud-Init ISO voor VM 115 op basis van de huidige Cloud-Init-instellingen.

```
root@pve:~# qm cloudinit update 115
generating cloud-init ISO
```

Stel de boot volgorde in op scsi0.

```
root@pve:~# qm set 115 --boot c --bootdisk scsi0

update VM 115: --boot c --bootdisk scsi0
```

Start nu de VM.

```
root@pve:~# qm start 115
```

18.4.2.2 Inloggen

Log in met student en het wachtwoord PXL.

Zoals je zal merken staat het toetsenbord weer op querky.

Je kan dat uiteraard weer aanpassen door onderstaande uit te voeren.

```
[student@Oracle ~]$ sudo localectl set-keymap be
```

Na reboot staat het toetsenbord nog steeds correct 😊.

18.4.2.3 SSH

SSH met een wachtwoord werkt vaak niet out of the box als je een VM aan de hand van een cloud-image aanmaakt. Zo ook bij de Oracle VM die we juist hebben aangemaakt.

Installeer nano als je geen ervaring hebt met de vi-editor om dit toch mogelijk te maken in de Oracle VM.

```
[student@Oracle ~]$ sudo dnf install nano
```

We passen nu het configuratiebestand van sshd aan.

```
[student@Oracle ~]$ sudo nano /etc/ssh/sshd_config
```

```
#PasswordAuthentication yes => PasswordAuthentication yes
```

```
GNU nano 8.1                               /etc/ssh/sshd_config                                Modified
#MaxAuthTries 6
#MaxSessions 10
#PubkeyAuthentication yes
# The default is to check both .ssh/authorized_keys and .ssh/authorized_keys2
# but this is overridden so installations will only check .ssh/authorized_keys
AuthorizedKeysFile      .ssh/authorized_keys
#AuthorizedPrincipalsFile none
#AuthorizedKeysCommand none
#AuthorizedKeysCommandUser nobody
# For this to work you will also need host keys in /etc/ssh/ssh_known_hosts
#HostbasedAuthentication no
# Change to yes if you don't trust ~/.ssh/known_hosts for
# HostbasedAuthentication
#IgnoreUserKnownHosts no
# Don't read the user's ~/.rhosts and ~/.shosts files
#moreRhosts yes
# To disable tunneled clear text passwords, change to no here!
#PasswordAuthentication yes
#PermitEmptyPasswords no
# Change to no to disable s/key passwords
#RbdInteractiveAuthentication yes
# Kerberos options
#KerberosAuthentication no
#KerberosOrLocalPasswd yes
#KerberosTicketCleanup yes
#KerberosGetfSToken no
#KerberosUseKuserok yes
# GSSAPI options
#GSSAPIAuthentication no
#GSSAPICleanupCredentials yes
#GSSAPIStrictAcceptorCheck yes
#GSSAPIKeyExchange no
#GSSAPINablekusers no
# Set this to 'yes' to enable PAM authentication, account processing,
# and session processing. If this is enabled, PAM authentication will
# be allowed through the RbdInteractiveAuthentication and
#RbdInteractivePAMAuthentication options
#
```

Opgelet: je moet onderstaande bestand ook wijzigen.

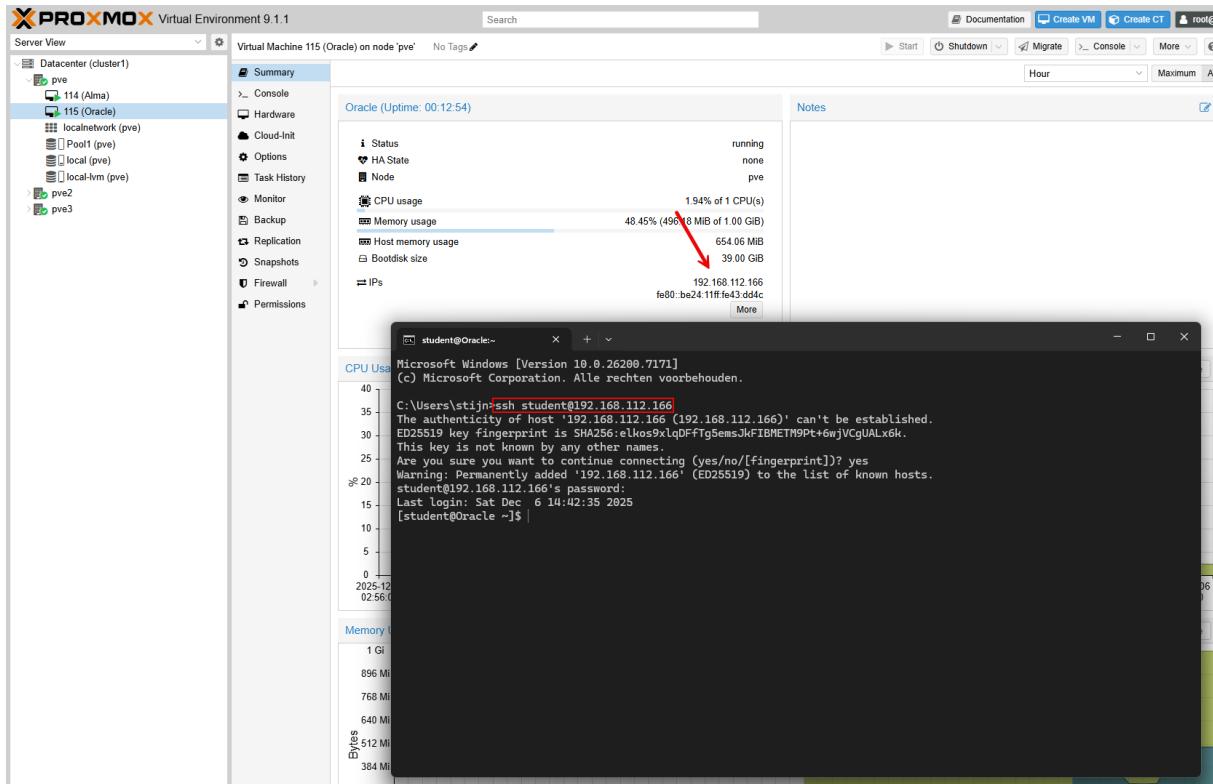
```
[student@Oracle ~] sudo nano /etc/ssh/sshd_config.d/50-cloud-init.conf
```

```
# PasswordAuthentication no      # <= zet hier du een # voor
```

Herstart sshd om de wijzigingen door te voeren.

```
[student@Oracle ~] sudo systemctl restart sshd
```

Je kan nu verbinding maken met de VM via SSH.



18.4.3 Cloud-Init YAML-bestand

Verwijder de VM van Oracle Linux om ruimte vrij te maken!

Zoals je juist gezien hebt moet je de keyboard-layout nog aanpassen in de VM. Je kan dit vermijden door gebruik te maken van YAML-bestanden.

Het voordeel van YAML-bestanden bij Cloud-Init in Proxmox is dat je veel meer instellingen kunt configureren dan via de GUI, zoals:

- meerdere users en groepen
- keyboard-layout, locale, timezone
- packages installeren
- services starten
- bestanden aanmaken (write_files)
- custom scripts uitvoeren (runcmd, bootcmd)
- geavanceerde netwerkconfiguratie

We zullen dit uittesten met een nieuwe VM.

```
root@pve:~# qm create 116 --name "Oracle" --memory 1024 --ostype l26 --net0 virtio,bridge=vmbr0,firewall=1 --agent 1 --scsihw virtio-scsi-single --sockets 1 --cores 1 --cpu max --vcpus 1 --cpuunits 100 --cpulimit 0
```

Uiteraard voegen we nu ook de Cloud-Init drive toe (op ide2).

```
root@pve:~# qm set 116 --ide2 local-lvm:cloudinit,media=cdrom
```

Importeer nu uiteraard de gedownloade disk naar local-lvm.

```
root@pve:~# qm importdisk 116 /var/lib/vz/import/0L10U0_x86_64-kvm-b266.qcow2 local-lvm
```

Koppel de disk aan de VM als scsi0 en zet Discard (TRIM/UNMAP) aan.

```
root@pve:~# qm set 116 --scsi0 local-lvm:vm-116-disk-0,discard=on
```

Vergroot de disk met 2GB

```
root@pve:~# qm resize 116 scsi0 +2G
Size of logical volume pve/vm-116-disk-0 changed from 37.00 GiB (9472 extents) to 39.00 GiB (9984 extents).
Logical volume pve/vm-116-disk-0 successfully resized.
```

We willen nu via een YAML-bestand een gebruiker met een wachtwoord aanmaken. Het wachtwoord dat je in het YAML-bestand zet moet geïncrypt zijn. Je kan een geïncrypt wachtwoord, na het installeren van whois, als volgt aanmaken.

```
root@pve:~# apt install whois
root@pve:~# mkpasswd --method=SHA-512 "PXL"
$6$aK7urudmAGGImhVY$gvFmZ3tf1xjz2qYdomnHDFCpw0QGIp423suRI41/awGP
2kIkBDZm7ncg2Ye70Mg92/22aiQx4wrZv.SZY67G1
```

het is normaal dat je telkens een ander SHA-512-wachtwoordhash krijgt, zelfs voor hetzelfde wachtwoord. Ze worden allemaal correct aanvaard.

Als je een YAML-bestand gebruikt voor Cloud-Init in Proxmox, moet dat bestand in een opslag staan die het content-type Snippets ondersteunt. We moeten het bestand dan ook aanmaken in de juiste directory.

```
root@pve:~# nano /var/lib/vz/snippets/116-user-data.yaml
#cloud-config
users:
  - name: student
    gecos: Student User
```

```
sudo: ALL=(ALL) NOPASSWD:ALL
shell: /bin/bash
lock_passwd: false
passwd:
$6$ak7urudmAGGImhVY$gvFmZ3tf1xjz2qYdomnHDFCpw0QGIp423suRI41/awGP
2kIkBZm7ncg2Ye70Mg92/22aiQx4wrZv.SZY67G1

ssh_pwauth: true

ssh:
allow-pw: true

disable_root: true

package_update: false
package_upgrade: false

chpasswd:
expire: false

network:
version: 2
ethernets:
eth0:
dhcp4: true

# Oracle Linux correct keyboard layout
```

```

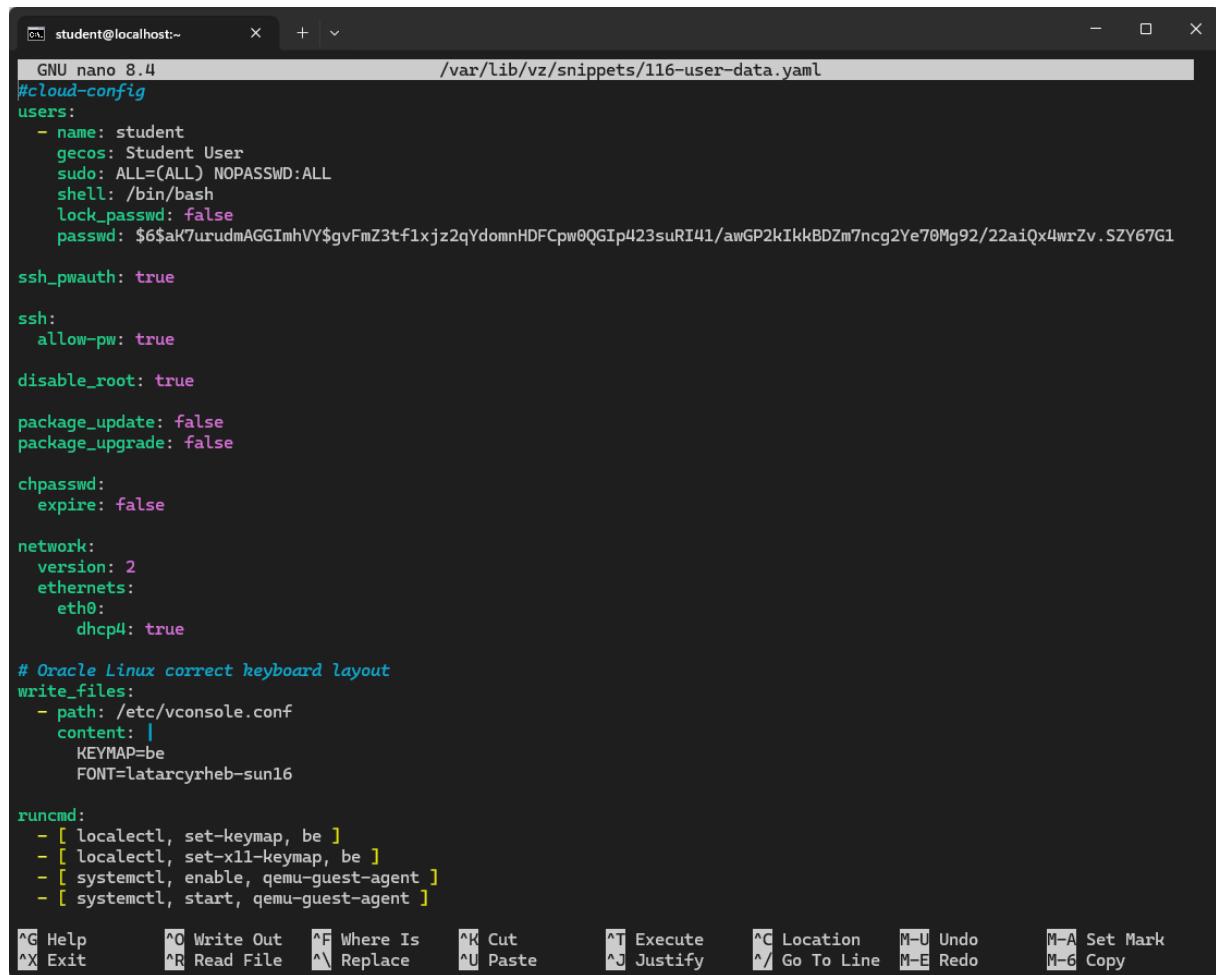
write_files:
  - path: /etc/vconsole.conf
    content: |
      KEYMAP=be
      FONT=latarcyrheb-sun16

```

```

runcmd:
  - [ localectl, set-keymap, be ]
  - [ localectl, set-x11-keymap, be ]
  - [ systemctl, enable, qemu-guest-agent ]
  - [ systemctl, start, qemu-guest-agent ]

```



```

student@localhost:~          + | 
GNU nano 8.4                  /var/lib/vz/snippets/116-user-data.yaml
#cloud-config
users:
  - name: student
    gecos: Student User
    sudo: ALL=(ALL) NOPASSWD:ALL
    shell: /bin/bash
    lock_passwd: false
    passwd: $6$ak7urudmAGGImhVY$gvFmZ3tf1xjz2qYdomnHDFCpw0QGIp423suRI41/awGP2kIkBDZm7ncg2Ye70Mg92/22aiQx4wrZv.SZY67G1
  ssh_pwauth: true
  ssh:
    allow-pw: true
  disable_root: true
  package_update: false
  package_upgrade: false
  chpasswd:
    expire: false
network:
  version: 2
  ethernets:
    eth0:
      dhcp4: true
# Oracle Linux correct keyboard layout
write_files:
  - path: /etc/vconsole.conf
    content: |
      KEYMAP=be
      FONT=latarcyrheb-sun16
runcmd:
  - [ localectl, set-keymap, be ]
  - [ localectl, set-x11-keymap, be ]
  - [ systemctl, enable, qemu-guest-agent ]
  - [ systemctl, start, qemu-guest-agent ]

```

The terminal window shows the configuration file being edited. At the bottom, there is a status bar with various keyboard shortcuts for nano editor:

- ^G Help
- ^O Write Out
- ^F Where Is
- ^K Cut
- ^T Execute
- ^C Location
- M-U Undo
- M-A Set Mark
- ^X Exit
- ^R Read File
- ^N Replace
- ^U Paste
- ^J Justify
- ^Y Go To Line
- M-E Redo
- M-G Copy

Beknopt uitgelegd staat hier het volgende in:

- Gebruiker en SSH
 - o Maakt gebruiker student aan met sudo-rechten zonder wachtwoord.
 - o Wachtwoord is vooraf ingesteld (geëncrypteerd hash).
 - o SSH wachtwoordlogin toegestaan (ssh_pauth: true).
- Beveiliging
 - o Rootlogin uitgeschakeld (disable_root: true).
 - o Geen automatische updates (package_update: false, package_upgrade: false).
 - o Wachtwoord verloopt niet (chpasswd.expire: false).
- Netwerk
 - o Interface eth0 gebruikt DHCP (dhcp4: true).
- Keyboard layout
 - o Schrijft Belgische toetsenbordindeling naar /etc/vconsole.conf:
 - KEYMAP=be
 - FONT=latacyrheb-sun16
 - o Zet keymap en X11-keymap via localectl naar be.
- System commands
 - o Activeert en start qemu-guest-agent bij boot.

Koppel een aangepaste Cloud-Init configuratie aan VM 116, zodat die bij boot de instellingen uit het YAML-bestand toepast.

```
root@pve:~# qm set 116 --cicustom "user=local:snippets/116-user-data.yaml"
```

```
update VM 116: --cicustom user=local:snippets/116-user-data.yaml
```

Proxmox leest bij boot rechtstreeks dat bestand in en schrijft het naar de Cloud-Init drive. Je moet cloudinit dan ook niet meer updaten tenzij je velden in de GUI terug wilt schrijven.

We stellen de boodisk enkel in op scsi0.

```
root@pve:~# qm set 116 --boot c --bootdisk scsi0
```

We starten nu de VM op.

```
root@pve:~# qm start 116
generating cloud-init ISO
```

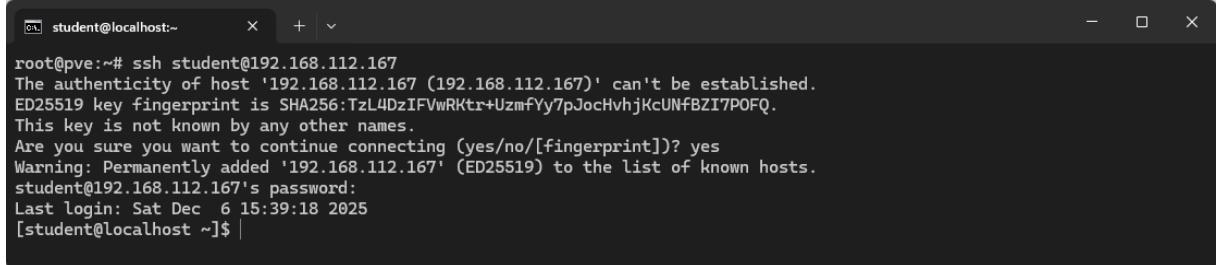
Je kan nu inloggen op de VM.

- Je toetsenbord staat onmiddellijk correct ingesteld.

```
Oracle Linux Server 10.0
Kernel 6.12.0-102.36.5.2.el11uek.x86_64 on an x86_64

localhost login: student
Password:
Last login: Sat Dec  6 15:39:06 on ttv1
[student@localhost ~]$ azerty
```

- Je kan ook verbinding maken met SSH.



```
student@localhost:~# ssh student@192.168.112.167
The authenticity of host '192.168.112.167 (192.168.112.167)' can't be established.
ED25519 key fingerprint is SHA256:TzL4DzIFVwRKtr+UzmfYy7pJocHvhjKcUNfBZI7POFQ.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.112.167' (ED25519) to the list of known hosts.
student@192.168.112.167's password:
Last login: Sat Dec  6 15:39:18 2025
[student@localhost ~]$
```