Docker

- Docker, allmänt
- Postgres, problem vid uppgradering
- Docker Compose
- Användning av resurser, RAM, CPU
- Uppgradera från docker compose v1 till docker compose v2
- Docker, Ubuntu, Proxmox, IPVlan, trafik mellan containrar och brandväggsregler
- Networks

Docker, allmänt

Den här sidan bör brytas ut i flera. Det var första skrivan jag skrev om docker allmänt.

Jag kör Docker på Ubuntu-VMs.

Installera på Ubuntu-server med `sudo apt  install docker.io`.
För att kunna slå `buildx`, `sudo apt install docker-buildx`.
Docker compose är att sätt att enkelt köra docker-images. Installera med `sudo apt-get install docker-compose-v2`.

Docker compose

Det går att starta containrar enligt specifiserade värden i en Docker compose fil.

Värde	Förklaring
container_name	Specifisera ett namn för containern. Blir bättre för det mänskliga ögat när en container ex. ska stängas av
restart: always, restart: unless-stopped	Så att containers alltid startar vid boot av VM alternativt om de inte har stoppats manuellt

Networking

Ta bort nätverk

```
docker network rm NETWORK-NAME
```

Aktivera IPv6

För att Docker ska ha IPv6 stöd måste filen `/etc/docker/daemon.json` skapas. Se exempelkonfiguration nedanför.

```
{
        "userland-proxy": false,
        "ipv6": true,
        "ip6tables": true,
        "fixed-cidr-v6": "fd00:1::/64",
        "experimental": true
}
```

Starta om docker efter att filen ovan har lagts till: `sudo systemctl restart docker`

Bridge Networking

Per default används "bridge-networking". Alla Docker-containers är nåbara utifrån via virtuella maskinens egna IP adress och NAT:as till en privat IP adress på insidan. Det gör att man måste delegera portar till de olika tjänsterna. Default IPv4 nät i Docker-miljön är 172.17.0.0/16.

IPvlan

Istället för Bridge Networking kan IPvlan användas. Kan köras i antingen L2 eller L3-läge. I L2 läge är Docker-hosten en L2-switch/brygga och i L3 läge en router.

IPvlan L3-konf:

```
sudo docker network create -d ipvlan \
    --subnet=10.10.10.0/24 \
    --subnet=2001:db8::/68 \
    --ipv6 -o parent=eth0 \
    -o parent=eth0 \
    -o ipvlan_mode=l3 DOCKER_NETWORK
```

Docker Network kräver gillar inte interface namn såsom ens18. Eth0 går bra, se artikel här för hur man byter namn på ett interface.

Nätet som har skapats kan anropas manuellt vid uppstart (docker run) eller i en Docker compose fil. Se exempel längre ned.

För att tilldela en fast IP-adress kan det antingen ske via CLI vid uppstart eller genom docker-compose-filen. Då används ipam.

Konfiguration för att skapa ett nytt nät med Docker Compose:

```
# Nedan specifiseras under specifika container-konfigurationen
services:
 CONTAINERNAMN:
  networks:
   DOCKER_NETWORK:
    ipv4_address: 10.10.0.3
    ipv6_address: 2001:db8::1

# Nedan specifiseras "globlat" i compose filen
networks:
 DOCKER_NETWORK:
  driver: ipvlan
  enable_ipv6: true
  ipam:
   config:
    - subnet: 10.10.0.0/24
    - subnet: 2001:db8::/64
```

Konfiguration för att använda ett existerande nät med docker compose:

```
networks:
  DOCKER_NETWORK:
    name: DOCKER_NETWORK
    external: true
```

Det är flaggan `external: true` som gör att man kan använda ett existerande nät.

Kommandon

Kommando	Förklaring
docker images	Se vilka images som finns lokalt
docker ps -a	Se containrar som är igång
docker network inspect xxxx	Kontrollera networking
docker compose up	Startar container från konfigurationsfil i lokal katalog, -d eller --detach för att starta i bakgrunden
docker compose stop	Stoppa alla containrar från konfigurationsfil i lokal katalog
docker compose start	Starta alla containrar från konfigurationsfil i lokal katalog

docker compose logs -f	Se loggar för den docker compose container vars mapp du står i. Släng på -f för att följa
docker system prune --....	Ta bort oanvända resurser
docker attach xxx	Anslut mot en container
docker rm	Ta bort container, ange ID
docker exec xxx uptime	Slå ex. uptime kommandot i en container
docker exec --user="root" -it xxxx nsenter	Hamna i containerns cli
docker container logs xxx	Se logg på en container, ange ID

docker run, flaggor	Förklaring
-d	Detach, kör i bakgrunden

Saker man kan installera med docker

LibreNMS (fork med docker-compose här)

Oxidized

Netbox (på denna wiki)

Technitium

NginxProxyManager

Avkorado

Postgres, problem vid uppgradering

När jag uppgraderade containern för Paperless så uppgraderades postgres.

Jag fick då följande felmeddelande i loggen:

```
db-1 | 2025-12-07 15:36:37.841 UTC [39] WARNING: database "paperless" has a collation version mismatch
db-1 | 2025-12-07 15:36:37.841 UTC [39] DETAIL: The database was created using collation version 2.36, but the operating system provides version 2.41.
db-1 | 2025-12-07 15:36:37.841 UTC [39] HINT: Rebuild all objects in this database that use the default collation and run ALTER DATABASE paperless REFRESH COLLATION VERSION, or build PostgreSQL with the right library version.
```

Lösningen på det var enligt nedan:

```
1) Hitta namnet på containern


jehrlander@docker-mgmt:~/docker/paperless-ngx/docker/compose$ sudo docker compose ps -a
NAME IMAGE COMMAND SERVICE CREATED STATUS PORTS
paperless-broker-1 docker.io/library/redis:7 "docker-entrypoint.s…" broker 2 minutes ago Up 2 minutes
paperless-db-1 docker.io/library/postgres:16 "docker-entrypoint.s…" db 2 minutes ago Up 2 minutes
paperless-webserver-1 ghcr.io/paperless-ngx/paperless-ngx:latest "/init" webserver 2 minutes ago Up 2 minutes (healthy)

2) Hoppa in på containern
jehrlander@docker-mgmt:~/docker/paperless-ngx/docker/compose$ sudo docker exec -it paperless-db-1 bash

3) Öppna postgresdatabasen

root@41be9e1eb574:/# psql -U paperless
WARNING: database "paperless" has a collation version mismatch
DETAIL: The database was created using collation version 2.36, but the operating system provides version 2.41.
HINT: Rebuild all objects in this database that use the default collation and run ALTER DATABASE paperless REFRESH COLLATION VERSION, or build PostgreSQL with the right library version.
psql (16.11 (Debian 16.11-1.pgdg13+1))
```

Type "help" for help.

4) Gör en reindex

paperless=# REINDEX DATABASE paperless;
REINDEX

5) Byt databasversion

paperless=# ALTER DATABASE paperless REFRESH COLLATION VERSION;
NOTICE: changing version from 2.36 to 2.41
ALTER DATABASE

6) Gå ut från DB, container och starta om containern

paperless=# exit
root@41be9e1eb574:/# exit
exit
jehrlander@docker-mgmt:~/docker/paperless-ngx/docker/compose$ sudo docker compose restart
WARN[0000] /home/jehrlander/docker/paperless-ngx/docker/compose/docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion
[+] Restarting 3/3
 ✔ Container paperless-broker-1 Started 0.3s
 ✔ Container paperless-db-1 Started 0.9s
 ✔ Container paperless-webserver-1 Started

Docker Compose

Jag använder mig av docker compose på Ubuntu.

Patchning

Patcha en container går väldigt fort och kan ske med minimal nedtid.

Slå först `sudo docker compose pull` för att hämta senaste avbilderna. Sedan `sudo docker container up` för att återskapa containern med senaste avbilden. Klart!

Container name

Ge containern ett specifikt namn med variabeln `container_name`. Ex:

```
grampsweb_redis:
  container_name: grampsweb_redis
```

User

Specifisera vilken användare som ska användas. Denna måste komma åt filtjänsterna. Specifiserar man inte en användare så används root. Det är alltså bra att specifisera användare. Om man kör numrerar UID/GID, ex 1000:1000, så behöver man inte skapa dem i förväg. Det kan vara bra att ha olika UID/GID per compose-fil.

Ex.

```
services:
  navidrome:
    image: deluan/navidrome:latest
    user: "1000:1000"
    volumes:
      - "./data:/data"
```

Restart

Man kan ställa in så att ens contianer alltid startar när hosten startar genom `restart: always`. Jag gillar `restart: unless-stopped`, vilket gör att containern inte startar automatiskt om den manuellt har stängts av.

Ex:

```
services:
  server:
    restart: unless-stopped
```

Volumes

Går att konfigurera på lika olika sätt. Ex. `config:/etc/dns` placerar volymen i `/var/lib/docker/volumes`. `./data:/data placerar` filerna i en undermapp till mappen som `docker-compose.yml` filen är i. Här är ett exempel på en NFS-monterad mapp:
`/data/docker/audiobookshelf/audiobooks:/audiobooks`

`sudo docker volume ls` listar alla volymer som docker känner till. inspect ger bra information om en specifik volym, ex:

```
jehrlander@docker-mgmt:~/docker/DnsServer$ sudo docker volume inspect dnsserver_config
[
    {
        "CreatedAt": "2024-04-18T13:54:19Z",
        "Driver": "local",
        "Labels": {
            "com.docker.compose.project": "dnsserver",
            "com.docker.compose.version": "2.20.2",
            "com.docker.compose.volume": "config"
        },
        "Mountpoint": "/var/lib/docker/volumes/dnsserver_config/_data",
        "Name": "dnsserver_config",
        "Options": null,
        "Scope": "local"
```

```
}
```

För att rensa oanvända volymer, som ingen container använder, använd `sudo docker volume prune -a`.

NFS-montera i container

En NFS montering sker från hosten. Man kan antingen montera NFS direkt på hosten via fstab (se den här artikeln för Ubuntu) eller så kan man montera per compose fil. Gör man det i compose-filen så blir filen mer portabel. Å andra sidan tycker jag att det skalar trevligt med att montera direkt från hosten . Jag vet inte vad som är bäst, säkerheten har jag inte rätt ut ännu.

Montera från host

Följ artikeln för Ubuntu länkad ovan. I exemplet nedanför är då mappen monterad som /data/music. Mer konfiguration behövs ej.

```
services:
  metadata-remote:
    image: ghcr.io/wow-signal-dev/metadata-remote:latest
    container_name: metadata-remote
    ports:
      - "8338:8338"
    volumes:
      - /data/music:/music
    environment:
      - PUID=1000
      - PGID=1000
    restart: unless-stopped
    networks:
      DOCKER_NETWORK:
        ipv4_address: 10.10.0.63

networks:
  DOCKER_NETWORK:
    name: DOCKER_NETWORK
    external: true
```

Ibland kan man få behörighetsproblem:

```
jehrlander@docker-mgmt:~/docker/kiwix$ sudo docker compose up
Attaching to kiwix-serve-1
kiwix-serve-1  | '/data' directory is not writable by 'user:user' (1001:1001). ZIM file(s) can not be written.
kiwix-serve-1 exited with code 1
```

Här syns användaren som används. Då är det lätt att fixa med:

```
sudo chown -R 1001:1001 /data/docker/kiwix
```

Det går även att definiera volymer längst ned i en compose-fil och sedan hänvisa till dem längre upp. Gör man såhär så måste man dock skapa alla mappar själva, det kan inte Docker göra via den här typen av volymer.

```
services:
  # MongoDB: https://hub.docker.com/_/mongo/
  mongodb:
    image: "mongo:7.0"
    restart: "on-failure"
    networks:
      - GRAYLOG_INTERNAL
    volumes:
      - "mongodb_data:/data/db"
      - "mongodb_config:/data/configdb"

volumes:
  mongodb_data:
    driver: local
    driver_opts:
      type: none
      o: bind
      device: /data/docker/graylog/mongodb_data
  mongodb_config:
    driver: local
    driver_opts:
      type: none
      o: bind
      device: /data/docker/graylog/mongodb_config
```

Montera direkt från container

Här är ett exempel på en compose fil (för applikation Kiwix) där monteringen sker via compose-filen. Då är det docker-containern själv som pratar NFS, vilket syns definierat i driver:

```
services:
  kiwix-serve:
    ports:
      - 8080:8080
    image: ghcr.io/kiwix/kiwix-serve:latest
    # uncomment next 4 lines to use it with local zim file in /tmp/zim
    volumes:
     - kiwix-nfs:/data
    command:
     - '*.zim'
    # uncomment next 2 lines to use it with remote zim file
    environment:
    - 'DOWNLOAD=https://download.kiwix.org/zim/wikipedia/wikipedia_en_all_maxi_2025-08.zim'
    networks:
      DOCKER_NETWORK:
        ipv4_address: 10.10.0.66

networks:
  DOCKER_NETWORK:
    name: DOCKER_NETWORK
    external: true

volumes:
  kiwix-nfs:
    driver: local
    driver_opts:
      type: nfs
      o: addr=10.0.7.50,rw,soft,nfsvers=4
      device: ":/volume1/kiwix"
```

Användning av resurser, RAM, CPU

För att se hur mycket resurser en container använder slå `sudo docker stats`.

Uppgradera från docker compose v1 till docker compose v2

Nedan är saxat från ChatGPT. Jag hade en gammal Ubuntu-VM som körde gamla docker compose versionen. Här är en guide för att uppgradera:

1️⃣ Ta bort gamla v1 (valfritt men städat)

```
sudo apt remove docker-compose
```

2️⃣ Lägg till Dockers APT-repo

```
sudo apt update
sudo apt install ca-certificates curl gnupg

sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg

echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] \
  https://download.docker.com/linux/ubuntu jammy stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt update
```

---

3 Installera Compose v2-plugin

`sudo apt install docker-compose-plugin`

Verifiera:

`docker compose version`

 Nu ska du få `v2.x.x`

Docker, Ubuntu, Proxmox, IPVlan, trafik mellan containrar och brandväggsregler

Min uppsättning är sådan att jag använder docker (med compose) på Ubuntu-VMar som i sin tur är hostade på Proxmox-hypervisor. Jag tilldelar containrar IPv4-adresser från ett dockernätverk som är av typen IPVlan. Det är ett eget nätverk som finns bakom Ubuntu-hosten. Jag har host-brandvägg aktiverat på VM i Proxmox.

Den här kombinationen av tekniker gör att jag ser trafiken mellan containrarna i Proxmox på host-brandväggen och kan bygga regler där. Jag är inte superbra på hur det funkar, jag har kopierat in ett svar från ChatGPT här:

Svar från ChatGPT

IPVlan *logiskt* kopplar containrarna direkt mot varandra, men i din **Proxmox-miljö** finns ett extra lager som gör att trafiken **ser ut som om den går via hosten**. Låt oss reda ut exakt varför.

Kort svar

Containrarna pratar **inte via Linux-hostens IP-stack**, men deras trafik går ändå **via hostens NIC → Proxmox vSwitch → Proxmox brandvägg**.
Därför syns den i hypervisorns firewall – och det är helt korrekt beteende.

Vad som faktiskt händer i ditt setup

Du har i praktiken detta:

```
[Container A]
     |
   IPVlan
     |
[Ubuntu VM kernel]
     |
  virtio-net
     |
[Proxmox bridge / vSwitch]
     |
[Proxmox firewall]  □□ du ser trafiken här
     |
[Ubuntu VM NIC]
     |
[Container B]
```

Viktig poäng

- **IPVlan gör switching i kernel**
- Men **kernel är i en VM**
- VM:ns nätverksinterface är **virtuellt (virtio)**
- All trafik måste därför passera **Proxmox bridge**

Varför detta inte motsäger hur IPVlan fungerar

“Men IPVlan ska ju vara host-local?”

Ja — på **bare metal**.

I en VM:

- Ubuntu-hostens “fysiska interface” är egentligen **ett virtuellt NIC**
- IPVlan använder detta NIC som *parent*
- All egress/ingress → **Proxmox vSwitch**

Så även:

```
containerA → containerB
```

blir i praktiken:

```
containerA
 → ipvlan
 → VM kernel
 → virtio-net
 → Proxmox bridge (firewall!)
 → virtio-net
 → VM kernel
 → ipvlan
 → containerB
```

Ingen NAT. Ingen routing.
Men **hypervisorn ser L2/L3-trafiken**.

Varför Proxmox brandvägg ser trafiken

Proxmox brandvägg:

- Körs på **bridge-nivå**
- Inspekterar **all VM-trafik**
- Spelar ingen roll om IP-källan och IP-destinationen är “samma VM”

Så:

- Samma MAC ✔
- Olika IP ✔

- Bridge-forwarding ✔
  → Firewall triggeras

Detta är **precis som om två processer i samma VM pratar via nätet**.

Varför detta är *bra* (och varför du gillar det □□)

Du får:

- □ Riktiga IP-adresser
- □ Ingen Docker NAT
- □ Full synlighet i Proxmox FW
- □ Central policy enforcement
- □ Mikrosegmentering utan overlay-nät

I praktiken beter sig containrarna som **mini-VMs bakom samma NIC**.

Detta är en av de *stora* fördelarna med:

> **IPVlan + VM + Proxmox**

Skillnad mot Docker bridge i VM

Med `bridge` hade du ofta sett:

- NAT inuti VM
- Mindre synlig L3-trafik i Proxmox
- Svårare policy enforcement

IPVlan ger dig **ren trafik upp till hypervisorn**.

Mental modell (bra att komma ihåg)

> IPVlan = *”direct attach” till **det interface du har***
> I en VM → det interface ägs av hypervisorn

Så:

- Bare metal → stannar lokalt
- VM → går via vSwitch

Networks

När det gäller docker-nätverk så kan man dela upp dem i två kategorier:

1. De som måste vara publikt nåbara (external)
2. De som inte måste vara publikt nåbara (internal)

Exempelvis databaser behöver inte vara publikt nåbara om de enbart ska serva containern i fråga. Då kan de klassas som internal. Det är också en fördel att ha ex. postgresdatabaser på interna nätverk, då om man har flera databaser externt nåbara kommer andra containrar som använder sig av postgres att försöka använda en annan en sin egen postgrescontainer. Webbsidor som körs på samma docker-host som en reverse proxy behöver inte heller vara extern nåbara, det räcker att reverse proxyn når containern.

Här är ett docker-compose exempel på hur det kan se ut. Notera att webservern `web_recipes` har två nätverk tilldelade, både det interna och det externa. Nätverket `TANDOOR_INTERNAL` skapas automatiskt när containrarna skapas. Nätverket `DOCKER_NETWORKS` är ett externt skapat nätverk, det har alltså skapats långt innan den här compose-filen skapades:

```
services:
  db_recipes:
    networks:
      TANDOOR_INTERNAL:

  web_recipes:
    depends_on:
      - db_recipes
    networks:
      DOCKER_NETWORK:
        ipv4_address: 10.10.0.36
      TANDOOR_INTERNAL:

networks:
  DOCKER_NETWORK:
    name: DOCKER_NETWORK
    external: true
  TANDOOR_INTERNAL:
    name: TANDOOR_INTERNAL
    driver: bridge
```

```
internal: true
```