1. Arr Stack

The Arr stack is a stack of applications that makes it possible for you and your friends and family to request media from the internet and download them to your Jellyfin instance. It is the bread and butter behind the scenes that makes media requests streamlined and user friendly for everyone. In this chapter we will describe how to install it step by step such that you can also host a media server that you and your friends can enjoy ! We will install the following applications via a docker compose file

- Sonarr
- Radarr
- Profilarr
- Bazaarr
- Seerr
- Qbittorent
- NZBGet
- Maintainerr
- Jellydash
- Prowlarr
- Flaresolverr

After the docker compose file is up and running we will also walk you through configuring each application s.t they work together as a symphony ;).

- 1. Storage Layout
- 2. Remote Arcane host (optional)
- 3. Docker Compose File
- 3. QBittorrent
- 4. NZBGet
- 3. Radarr
- 10. Journey of a Media Request
- 8. Seerr
- 1. Create a new Docker VM
- 9. Maintainerr
- 9. Profilarr
- 9. JellyGlance

- 10. Autopulse
- 11. Wizarr
- 12. Bazarr (Subtitles)
- 14. Readarr

1. Storage Layout

Now, the installation of the Arr stack is not all that hard but it is important to choose a good storage layout. There are different methods each with its pros and cons. The one method that we will discuss in the setup is the Hardlink method, which basically means that the **downloads** and **tv** and **movie** folders are linked together instead of seen separately. That means instead of storing 2 times a movie you only store it once. For this guide we will follow the famous Trash Guides setup here that has only one **/mnt/data:/data** volume with the following structure

```
data
├── torrents
│   ├── books
│   ├── movies
│   ├── music
│   └── tv
├── usenet
│   ├── incompleted
│   └── completed
│       ├── books
│       ├── movies
│       ├── music
│       └── tv
└── media
    ├── books
    ├── movies
    ├── music
    └── tv
```

1. Create an NFS share on TrueNas scale

2. Go inside your arr stack vm via SSH

3. Then we want to make to give ownership of the /mnt folder to our user that is not root.

4. If you have not added any users then your base user and did not change your user ID it most likely will be 1000. However, you can be sure by using the command

```
id myuser
```

5. Then we make myuser owner of the mnt folder with

```
sudo chown -R myuser:myuser /mnt
sudo chmod -R 770 /mnt
```

6. Then create the data folder inside the mnt folder with

```
mkdir /mnt/data
```

7. Sweet, then now we need to install nfs tools such that we can mount the nfs share to /mnt/data

```
sudo apt install nfs-common
```

8. Then we will mount the NFS share by entering the /etc/fstab with nano

```
sudo nano /etc/fstab
```

9. There we copy paste in the nfs share credentials as follows

```
NAS-IP:/mnt/storagepool/media /mnt/data  nfs  defaults 0 0
```

10. Then to make the changes permanent we use

```
sudo systemctl daemon-reload
```

11. Then to mount the drives we use

```
sudo mount -a
```

12. You can then check if the mount occured properly by using the command

```
mount
```

13. You should then see that your /mnt/data folder is mounted to your NFS share

14. Now that that is done we will add all the folders inside the /mnt/data directory. Go inside the /mnt/directory and use the following command to create the whole structure with one command

```
mkdir -p torrents/{tv,movies,music,books} && mkdir -p
usenet/{incompleted,completed,intermediate} && mkdir -p
usenet/completed/{tv,movies,music,books} && mkdir -p media/{tv,movies,music,books}
```

15. To check if the folder structure is correct you can use the command

```
tree
```

16. If tree is not yet installed you can install it with

```
sudo apt install tree
```

Perfect! Then now in the next chapter we will start creating the docker compose file for our Arr stack ;)

2. Remote Arcane host (optional)

Now, you can of course create a docker compose file in your ARR VM, however Arcane also offers to possibility to manage other docker vm's as well. Then you directly get all the benefits of resource and update management for your ARR VM as well. This is not necessary but highly recommended for ease of use. Hence, we will show you here how to create a remote arcane host s.t we can deploy our ARR stack via the arcane Webgui instead of via the cli ;)

1. Go to your Arcane Manager host and log in

2. Next we will go to Environments > + Add Environment

3. Give it a name e.g "arr"

4. Next add the ARRVM IP address with a port you want to run it on e.g **3553** e.g 10.20.34.90:3553

5. Then click on generate agent configuration which will provide a docker compose file you can use for your Arr VM.

6. Copy the generated compose file

7. Enter your Arr VM with ssh

8. Create a new folder called arcane

```
mkdir arcane
```

9. Inside of the arcane folder create a new docker compose file and copy paste the generated compose file

```
nano compose.yml
```

10. Then as an example it looks like this

compose.yml

```
services:
  arcane-agent:
    image: ghcr.io/getarcaneapp/agent:latest
    container_name: arcane-agent
    restart: unless-stopped
    environment:
      - AGENT_MODE=true
      - EDGE_TRANSPORT=poll
      - AGENT_TOKEN=arc_f18ea27f48b812813a6d6c1099b7ad0972beb22b6c532d1d253e6e89fb8f6fbe
      - MANAGER_API_URL=https://arcane.example.org
      - PUID=1000
      - GPID=1000
    ports:
      - "3553:3553"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - /opt/docker:/app/data
```

11. Now, for the volumes you need to give it access to your docker sock as with the arcane manager and also again the volume you want to store all your projects in. I chose for /opt/docker but you can choose whatever you like of course. In that folder you specify arcane will create a projects folder.

12. Also I added my user and group ID that I use with **PUID** and **GPID** both setting to a 1000. This is necessary to make sure you don't have permissions issues.

13. Then simply start the arcane agent container with

```
docker compose up -d
```

14. Then go back to your Arcane manager and click on test connection. It should then show online if everything went well ;)

15. Then the last step to actually start working inside your arr vm you need to go back to your dashboard and select the arr Environment ;)

16. Alright! Then let's make a new project on the next page

3. Docker Compose File

Okey then! So you have created an NFS share, created the Trash Guides folder structure inside of that NFS share then now it is time to use that to setup the docker compose file. The compose file comes from different websites and it is surely not the only way to do this. However, it is a simple one that creates one docker network called **internal proxy** and uses **gluetun** as a VPN container. Furthermore, we will use the open ports 4001 - 4012 for the applications that we are running. You can keep the original official port numbers but I like to keep it structured instead of random port numbers. With all of that said the compose file looks like this

compose.yml

```
services:
# Sonarr - TV Shows
  sonarr:
    image: lscr.io/linuxserver/sonarr:latest
    container_name: sonarr
    environment:
      - PUID=${PUID}
      - PGID=${PGID}
      - TZ=${TZ}
    volumes:
      - ./sonarr/config:/config
      - ${ROOT_MEDIA_PATH}:/data # Access to /data/downloads and /data/media/tv
    ports:
      - 4001:8989
    restart: unless-stopped
    networks:
      - internal-proxy

  # Radarr - Movies
  radarr:
    image: lscr.io/linuxserver/radarr:latest
    container_name: radarr
    environment:
      - PUID=${PUID}
      - PGID=${PGID}
```

```
      - TZ=${TZ}
    volumes:
      - ./radarr/config:/config
      - ${ROOT_MEDIA_PATH}:/data # Access to /data/downloads and /data/media/movies
    ports:
      - 4002:7878
    restart: unless-stopped
    networks:
      - internal-proxy

# Prowlarr - Indexer centralized management
  prowlarr:
    image: lscr.io/linuxserver/prowlarr:latest
    container_name: prowlarr
    environment:
      - PUID=${PUID}
      - PGID=${PGID}
      - TZ=${TZ}
    volumes:
      - ./prowlarr/config:/config
    ports:
      - 4003:9696
    restart: unless-stopped
    networks:
      - internal-proxy

# Profilarr
  profilarr:
    image: ghcr.io/dictionarry-hub/profilarr:latest
    container_name: profilarr
    restart: unless-stopped
    ports:
      - "4004:6868"
    volumes:
      - ./config:/config
    environment:
      - PUID=${PUID}
      - PGID=${PGID}
      - UMASK=022
```

```
      - TZ=${TZ}
    depends_on:
      - radarr
      - sonarr

# FlareSolverr - Bypasses Cloudflare protection for Indexers
  flaresolverr:
    image: ghcr.io/flaresolverr/flaresolverr:latest
    container_name: flaresolverr
    environment:
      - LOG_LEVEL=${LOG_LEVEL:-info}
      - TZ=${TZ}
    ports:
      - 4005:8191
    restart: unless-stopped
    networks:
      - internal-proxy

# bazarr for subtitles
  bazarr:
    image: lscr.io/linuxserver/bazarr:latest
    container_name: bazarr
    environment:
      - PUID=${PUID}
      - PGID=${PGID}
      - TZ=${TZ}
    volumes:
      - ./bazarr/config:/config
      - ${ROOT_MEDIA_PATH}:/data
    ports:
      - 4006:6767
    restart: unless-stopped
    networks:
      - internal-proxy

  maintainerr:
    image: ghcr.io/maintainerr/maintainerr:latest # or
ghcr.io/maintainerr/maintainerr:development (to test unreleased changes)
    container_name: maintainerr
```

```
    user: 1000:1000
    volumes:
      - type: bind
        source: /mnt/data
        target: /opt/data
#          - type: bind # uncomment for the leftover-folder cleanup: your library, at the same path Radarr/Sonarr report it at
#            source: /path/to/media
#            target: /path/to/media
    environment:
      - TZ=Europe/Amterdam
#          - BASE_PATH=/maintainerr # uncomment if you're serving maintainerr from a subdirectory
#          - UI_HOSTNAME=:: # uncomment if you want to listen on IPv6 instead (default 0.0.0.0)
#          - UI_PORT=4007 # uncomment to change the UI port (default 6246)
#          - GITHUB_TOKEN=ghp_yourtoken # Optional: GitHub Personal Access Token for higher API rate limits (60/hr without, 5000/hr with token)
    ports:
      - 4007:6246
    restart: unless-stopped
    healthcheck: # already baked into the image; included here so you can tune it
      test: ['CMD', '/opt/app/healthcheck.sh']
      interval: 30s
      timeout: 5s
      start_period: 40s
      retries: 3
    networks:
      - internal-proxy

  seerr:
    image: ghcr.io/seerr-team/seerr:latest
    init: true
    container_name: seerr
    environment:
      - PUID=${PUID}
      - PGID=${PGID}
      - LOG_LEVEL=debug
      - TZ=${TZ}
```

```
      - PORT=5055 #optional
    ports:
      - 4011:5055
    volumes:
      - ./seerrconfig:/app/config
    healthcheck:
      test: wget --no-verbose --tries=1 --spider
http://10.69.1.103:4011/api/v1/settings/public || exit 1
      start_period: 20s
      timeout: 3s
      interval: 15s
      retries: 3
    restart: unless-stopped
    networks:
      - internal-proxy


  qbittorrent:
    image: lscr.io/linuxserver/qbittorrent
    container_name: qbittorrent
    network_mode: "service:gluetun"  # Routes all traffic through Gluetun
    depends_on:
      gluetun:
        condition: service_healthy # Wait for VPN to be up first
    environment:
      - PUID=1000
      - PGID=1000
      - TZ=${TZ}
      - WEBUI_PORT=4009
      - WEBUI_ADDRESS=0.0.0.0
      - WEBUI_EXTERNAL_ACCESS=true
    volumes:
      - ./qbittorrent:/config
      - ${ROOT_MEDIA_PATH}:/data
    restart: unless-stopped

  nzbget:
    image: lscr.io/linuxserver/nzbget:latest
    container_name: nzbget
```

```
    environment:
      - PUID=${PUID}
      - PGID=${PGID}
      - TZ=${TZ}
    volumes:
      - /etc/localtime:/etc/localtime:ro
      - ./nzbget:/config
      - /mnt/data:/data
    ports:
      - 4008:6789
    restart: unless-stopped
    networks:
      - internal-proxy

  jellydash:
    image: ghcr.io/themartz90/jellydash:latest
    restart: unless-stopped
    ports:
      - "${APP_PORT:-8080}:80"
    environment:
      PUID: "${PUID}"
      PGID: "${PGID}"
      TZ: "${TZ}"
      APP_ENV: "${APP_ENV:-production}"
      APP_DEBUG: "${APP_DEBUG:-false}"
      APP_TIMEZONE: "${APP_TIMEZONE:-UTC}"
      # Bundled MariaDB below. Using your own database server instead? Point
      # DB_HOST at it and drop the db service in docker-compose.override.yml.
      DB_HOST: "${DB_HOST:-db}"
      DB_PORT: "${DB_PORT:-3306}"
      DB_DRIVER: mysqli
      DB_NAME: "${DB_NAME:-jellydash}"
      DB_USER: "${DB_USER:-jellydash}"
      DB_PASS: "${DB_PASS:-change-me}"
      JELLYFIN_URL: "${JELLYFIN_URL}"
      JELLYFIN_API_TOKEN: "${JELLYFIN_API_TOKEN}"
      JELLYFIN_VERIFY_SSL: "${JELLYFIN_VERIFY_SSL:-true}"
      LIBRARIES_CACHE_TTL: "${LIBRARIES_CACHE_TTL:-300}"
      # Jellyseerr (optional; the page hides until these are set)
```

```
      JELLYSEER_URL: "${JELLYSEER_URL:-}"
      JELLYSEER_API_TOKEN: "${JELLYSEER_API_TOKEN:-}"
      JELLYSEER_VERIFY_SSL: "${JELLYSEER_VERIFY_SSL:-true}"
      SEERR_POLL_INTERVAL: "${SEERR_POLL_INTERVAL:-120}"
      SEERR_NOTIFY_ENABLED: "${SEERR_NOTIFY_ENABLED:-true}"
      # Background poller (history recording, cache warming, alerts)
      POLLER_ENABLED: "${POLLER_ENABLED:-true}"
      POLL_INTERVAL: "${POLL_INTERVAL:-30}"
      # Web Push notifications (optional; generate keys once, see README)
      PUSH_ENABLED: "${PUSH_ENABLED:-true}"
      VAPID_PUBLIC_KEY: "${VAPID_PUBLIC_KEY:-}"
      VAPID_PRIVATE_KEY: "${VAPID_PRIVATE_KEY:-}"
      VAPID_SUBJECT: "${VAPID_SUBJECT:-mailto:admin@example.com}"
      PUSH_IGNORE_USERS: "${PUSH_IGNORE_USERS:-}"
      # Public base URL (optional): external alert links point back here.
      APP_URL: "${APP_URL:-}"
      # Extra notification channels; each is active once its config is set.
      TELEGRAM_BOT_TOKEN: "${TELEGRAM_BOT_TOKEN:-}"
      TELEGRAM_CHAT_ID: "${TELEGRAM_CHAT_ID:-}"
      PUSHOVER_APP_TOKEN: "${PUSHOVER_APP_TOKEN:-}"
      PUSHOVER_USER_KEY: "${PUSHOVER_USER_KEY:-}"
      DISCORD_WEBHOOK_URL: "${DISCORD_WEBHOOK_URL:-}"
      # Libraries hidden from Trending/Most Watched (also editable in Settings)
      TRENDING_EXCLUDE_LIBRARIES: "${TRENDING_EXCLUDE_LIBRARIES:-}"
      # Optional login gate (recommended when internet-facing)
      AUTH_ENABLED: "${AUTH_ENABLED:-false}"
      AUTH_ADMIN_USER: "${AUTH_ADMIN_USER:-}"
      AUTH_ADMIN_PASSWORD: "${AUTH_ADMIN_PASSWORD:-}"
    depends_on:
      db:
        condition: service_healthy
    volumes:
      - ./jellydash_cache:/var/www/html/cache
      - ./jellydash_runtime_cache:/var/www/html/var/cache
      - ./jellydash_logs:/var/www/html/var/log
      - ./jellydash_uploads:/var/www/html/public/uploads
      # Drop-in feature modules (see docs/MODULES.md), e.g.:
      # - ./my-modules/downloads:/var/www/html/modules/downloads:ro
    networks:
```

```
    - internal-proxy

db:
  image: mariadb:11
  restart: unless-stopped
  environment:
    MARIADB_RANDOM_ROOT_PASSWORD: "yes"
    MARIADB_DATABASE: "${DB_NAME:-jellydash}"
    MARIADB_USER: "${DB_USER:-jellydash}"
    MARIADB_PASSWORD: "${DB_PASS:-change-me}"
    PUID: "${PUID}"
    PGID: "${PGID}"
    TZ: "${TZ}"
  healthcheck:
    test: ["CMD", "healthcheck.sh", "--connect", "--innodb_initialized"]
    interval: 5s
    timeout: 5s
    retries: 12
  volumes:
    - ./jellydash_data:/var/lib/mysql
  networks:
    - internal-proxy

gluetun:
  image: qmcgaw/gluetun:latest
  container_name: gluetun
  cap_add:
    - NET_ADMIN
  environment:
    - PUID=${PUID}
    - PGID=${PGID}
    - TZ=${TZ}
    - VPN_SERVICE_PROVIDER=${VPN_SERVICE_PROVIDER}
    - VPN_TYPE=${VPN_TYPE}
    - WIREGUARD_PRIVATE_KEY=${WIREGUARD_PRIVATE_KEY}
    - WIREGUARD_ADDRESSES=${WIREGUARD_ADDRESSES}
    - VPN_PORT_FORWARDING=${VPN_PORT_FORWARDING}
  ports:
    - 8888:8888 # Health check and control
```

```
      - 4009:4009
#      - 4009:6881 # qBittorrent P2P port
#      - 4009:6881/udp
    restart: unless-stopped
    networks:
      - internal-proxy

networks:
  internal-proxy:
    driver: bridge
```

and the associated environment file looks like this

.env

```
# User and Group IDs for Permissions
PUID=1000
PGID=1000
TZ=Europe/Amsterdam

# Global Paths for Download Client and *Arr Apps
DOWNLOADS_PATH=/mnt/data
MEDIA_PATH=/mnt/data
ROOT_MEDIA_PATH=/mnt/data

# Network Settings
PROXY_NETWORK=internal-proxy

VPN_SERVICE_PROVIDER=protonvpn
VPN_TYPE=wireguard
WIREGUARD_PRIVATE_KEY=private-key
WIREGUARD_ADDRESSES=x.x.x.x/32
VPN_PORT_FORWARDING=yes

# Jellydash configuration.
# ------------------------------------------------------------------
```

```
# Basics
# ------------------------------------------------------------------

# Leave as production. The local mode is only for development.
APP_ENV=production
APP_DEBUG=false

# The port Jellydash runs on, so http://your-host:8080
APP_PORT=4010

# Your timezone (IANA name, e.g. Europe/Prague). Used for timestamps
# and the daily statistics boundaries.
APP_TIMEZONE=Europe/Amsterdam

# Public address of your dashboard. Optional, can stay empty.
# When set, notification alerts include a link back to the app.
APP_URL=

#Database

DB_HOST=db
DB_PORT=3306
DB_NAME=jellydash
DB_USER=jellydash
# Required. Pick any password, the bundled database is created with it.
DB_PASS=9u2983fjosdjhf9823jf9jisodf

# ------------------------------------------------------------------
# Jellyfin (required)
# ------------------------------------------------------------------

# Required. The address of your Jellyfin server.
JELLYFIN_URL=https://jellyfin.example.org

# Required. An API key from Jellyfin > Dashboard > API Keys. Use an admin
# key, some statistics need to read library paths.
JELLYFIN_API_TOKEN=jellyfin-api-token

# Set false only when your Jellyfin runs on self-signed HTTPS.
```

```
JELLYFIN_VERIFY_SSL=true

# How long the Libraries page cache lives, in seconds. The default is fine.
LIBRARIES_CACHE_TTL=300

# ------------------------------------------------------------------
# Jellyseerr (optional, can stay empty)
# ------------------------------------------------------------------
# Fill these and the Jellyseerr page appears in the menu, including alerts
# for new requests. API key: Jellyseerr > Settings > General > API Key.

JELLYSEER_URL=
JELLYSEER_API_TOKEN=
JELLYSEER_VERIFY_SSL=true

# Seconds between request syncs, and whether new requests send an alert.
SEERR_POLL_INTERVAL=120
SEERR_NOTIFY_ENABLED=true

# ------------------------------------------------------------------
# Background poller
# ------------------------------------------------------------------
# Records play history even when nobody has the dashboard open, and sends
# the notifications. You want this on.

POLLER_ENABLED=true

# Seconds between Jellyfin checks. 30 is a good balance.
POLL_INTERVAL=30

# ------------------------------------------------------------------
# Notifications (all optional, can stay empty)
# ------------------------------------------------------------------
# Alerts when someone starts playing and when a new Jellyseerr request
# comes in. A channel is on as soon as its values are filled. Setup steps
# for every channel are in the README.

# Master switch for all notification channels.
PUSH_ENABLED=true
```

```
# Jellyfin usernames that never trigger an alert, comma-separated.
# Typically your own. Also editable later in Settings inside the app.
PUSH_IGNORE_USERS=

# ------------------------------------------------------------------
# Login (optional)
# ------------------------------------------------------------------
# Off by default. Turn it on when the dashboard is reachable from the
# internet. The admin user is created automatically on the next start.

AUTH_ENABLED=false
AUTH_ADMIN_USER=jkodfk
# The password needs at least 8 characters, shorter ones are rejected.
AUTH_ADMIN_PASSWORD=tJT9off2KfCN2Ug0x8CI

# ------------------------------------------------------------------
# Statistics
# ------------------------------------------------------------------

# Jellyfin libraries to hide from Trending and Most Watched, comma-separated
# names. Also editable later in Settings inside the app.
TRENDING_EXCLUDE_LIBRARIES=
```

I keep every tunable value in a `.env` file rather than hardcoding it in the Compose file. Ports, paths, and the user IDs all live in one place, which makes the Compose file readable and easy to reuse. I also run every service on an **external** network called `media_network`, which means the network is created once, outside of Compose, and the stack attaches to it. That keeps the network stable even when I tear the stack down and bring it back up.

Sweet! So you want to install the arr stack on your self hosted device. Well, as with any app you need decide which app workflow deployment you want to use. Here, we focus on the docker compose deployment using a dedicated VM. Now, personally I like to run the arr stack in a dedicated VM and Jellyfin in another Docker VM. This gives you the possibility to move your Jellyfin

instance around easily to different hosts and if something is wrong with your arr stack then your users can still enjoy the already downloaded media. Eventually, it is up to you and the amount of resources you have at your disposal what your decision will be.

1. Create a new docker vm by using the proxmox helper scripts ;) Or if you want to continue using your existing docker vm then you need

2.

Create a project directory and the `.env` file:

Hottio Decluttar

Whisparr is the Sonarr/Radarr equivalent for adult content

Whisper-AI generates subtitles on the fly for your media

compose.yml

```
services:
# Sonarr - TV Shows
  sonarr:
    image: lscr.io/linuxserver/sonarr:latest
    container_name: sonarr
    environment:
      - PUID=${PUID}
      - PGID=${PGID}
      - TZ=${TZ}
    volumes:
      - ./sonarr/config:/config
      - ${ROOT_MEDIA_PATH}:/data # Access to /data/downloads and /data/media/tv
    ports:
      - 8989:8989
    restart: unless-stopped
    networks:
      - internal-proxy

  # Radarr - Movies
```

```
  radarr:
    image: lscr.io/linuxserver/radarr:latest
    container_name: radarr
    environment:
      - PUID=${PUID}
      - PGID=${PGID}
      - TZ=${TZ}
    volumes:
      - ./radarr/config:/config
      - ${ROOT_MEDIA_PATH}:/data # Access to /data/downloads and /data/media/movies
    ports:
      - 7878:7878
    restart: unless-stopped
    networks:
      - internal-proxy

networks:
  internal-proxy:
    driver: bridge

# Prowlarr - Indexer centralized management
  prowlarr:
    image: lscr.io/linuxserver/prowlarr:latest
    container_name: prowlarr
    environment:
      - PUID=${PUID}
      - PGID=${PGID}
      - TZ=${TZ}
    volumes:
      - ./prowlarr/config:/config
    ports:
      - 9696:9696
    restart: unless-stopped
    networks:
      - internal-proxy

# Profilarr & Decluttarr (Config only, no media access needed usually)
  profilarr:
    image: santiagosayshey/profilarr:latest
```

```
    container_name: profilarr
    environment:
      - TZ=${TZ}
    volumes:
      - ./profilarr/config:/config
    ports:
      - 6868:6868
    restart: unless-stopped
    networks:
      - internal-proxy
    depends_on:
      - sonarr
      - radarr

# FlareSolverr - Bypasses Cloudflare protection for Indexers
  flaresolverr:
    image: ghcr.io/flaresolverr/flaresolverr:latest
    container_name: flaresolverr
    environment:
      - LOG_LEVEL=${LOG_LEVEL:-info}
      - TZ=${TZ}
    ports:
      - 8191:8191
    restart: unless-stopped
    networks:
      - internal-proxy

# Gluetun - VPN Client
  gluetun:
    image: qmcgaw/gluetun:latest
    container_name: gluetun
    cap_add:
      - NET_ADMIN
    environment:
      - PUID=${PUID}
      - PGID=${PGID}
      - TZ=${TZ}
      - VPN_SERVICE_PROVIDER=${VPN_SERVICE_PROVIDER}
      - VPN_TYPE=${VPN_TYPE}
```

```
      - WIREGUARD_PRIVATE_KEY=${WIREGUARD_PRIVATE_KEY}
      - WIREGUARD_ADDRESSES=${WIREGUARD_ADDRESSES}
      - VPN_PORT_FORWARDING=${VPN_PORT_FORWARDING}
    ports:
      - 8888:8888 # Health check and control
      - 6881:6881 # qBittorrent P2P port
      - 6881:6881/udp
    restart: unless-stopped
    networks:
      - internal-proxy

# If using Gluetun, use qbittorrent from linuxserver io, NOT the hotio image below!
  qbittorrent:
    image: lscr.io/linuxserver/qbittorrent
    network_mode: "service:gluetun" # The "Magic" line
    depends_on:
      gluetun:
        condition: service_healthy # Wait for VPN to be up first

# Lidarr - Music
  lidarr:
    image: lscr.io/linuxserver/lidarr:latest
    container_name: lidarr
    environment:
      - PUID=${PUID}
      - PGID=${PGID}
      - TZ=${TZ}
    volumes:
      - ./lidarr/config:/config
      - ${ROOT_MEDIA_PATH}:/data
    ports:
      - 8686:8686
    restart: unless-stopped
    networks:
      - internal-proxy

  # Bazarr - Subtitles
  bazarr:
    image: lscr.io/linuxserver/bazarr:latest
```

```
    container_name: bazarr
    environment:
      - PUID=${PUID}
      - PGID=${PGID}
      - TZ=${TZ}
    volumes:
      - ./bazarr/config:/config
      - ${ROOT_MEDIA_PATH}:/data
    ports:
      - 6767:6767
    restart: unless-stopped
    networks:
      - internal-proxy

  # whisper-ai - Local Subtitle Generation (Requires GPU)
  whisper-ai:
    image: ${WHISPER_IMAGE_URL} # Use a pre-built image like "jellyfin-whisper-
lab/whisper-container"
    container_name: whisper-ai
    environment:
      - PUID=${PUID}
      - PGID=${PGID}
      - TZ=${TZ}
    volumes:
      - ./whisper/config:/config
      - ${ROOT_MEDIA_PATH}:/data
    # Enable GPU access if you have one, or comment out the 'deploy' section for CPU-only:
    # deploy:
    #   resources:
    #     reservations:
    #       devices:
    #         - driver: nvidia
    #           count: all
    #           capabilities: [gpu]
    restart: unless-stopped
    networks:
      - internal-proxy

  # Decluttarr - Queue Cleaner
```

```
  decluttarr:
    image: hotio/decluttarr:latest
    container_name: decluttarr
    environment:
      - PUID=${PUID}
      - PGID=${PGID}
      - TZ=${TZ}
      - DECLUTTARR_CRON="*/15 * * * *"
      # API Keys/URLs for Sonarr, Radarr, etc. go here post-setup!
    volumes:
      - ./decluttarr/config:/config
    restart: unless-stopped
    networks:
      - internal-proxy

  nzbget:
    image: lscr.io/linuxserver/nzbget:latest
    container_name: nzbget
    environment:
      - PUID=${PUID}
      - PGID=${PGID}
      - TZ=${TZ}
    volumes:
      - /etc/localtime:/etc/localtime:ro
      - ./nzbget:/config
      - /mnt/data:/data
    restart: unless-stopped
    networks:
      - internal-proxy

  jellydash:
    image: ghcr.io/themartz90/jellydash:latest
    restart: unless-stopped
    ports:
      - "${APP_PORT:-8080}:80"
    environment:
      APP_ENV: "${APP_ENV:-production}"
      APP_DEBUG: "${APP_DEBUG:-false}"
      APP_TIMEZONE: "${APP_TIMEZONE:-UTC}"
```

```
# Bundled MariaDB below. Using your own database server instead? Point
# DB_HOST at it and drop the db service in docker-compose.override.yml.
DB_HOST: "${DB_HOST:-db}"
DB_PORT: "${DB_PORT:-3306}"
DB_DRIVER: mysqli
DB_NAME: "${DB_NAME:-jellydash}"
DB_USER: "${DB_USER:-jellydash}"
DB_PASS: "${DB_PASS:-change-me}"
JELLYFIN_URL: "${JELLYFIN_URL}"
JELLYFIN_API_TOKEN: "${JELLYFIN_API_TOKEN}"
JELLYFIN_VERIFY_SSL: "${JELLYFIN_VERIFY_SSL:-true}"
LIBRARIES_CACHE_TTL: "${LIBRARIES_CACHE_TTL:-300}"
# Jellyseerr (optional; the page hides until these are set)
JELLYSEER_URL: "${JELLYSEER_URL:-}"
JELLYSEER_API_TOKEN: "${JELLYSEER_API_TOKEN:-}"
JELLYSEER_VERIFY_SSL: "${JELLYSEER_VERIFY_SSL:-true}"
SEERR_POLL_INTERVAL: "${SEERR_POLL_INTERVAL:-120}"
SEERR_NOTIFY_ENABLED: "${SEERR_NOTIFY_ENABLED:-true}"
# Background poller (history recording, cache warming, alerts)
POLLER_ENABLED: "${POLLER_ENABLED:-true}"
POLL_INTERVAL: "${POLL_INTERVAL:-30}"
# Web Push notifications (optional; generate keys once, see README)
PUSH_ENABLED: "${PUSH_ENABLED:-true}"
VAPID_PUBLIC_KEY: "${VAPID_PUBLIC_KEY:-}"
VAPID_PRIVATE_KEY: "${VAPID_PRIVATE_KEY:-}"
VAPID_SUBJECT: "${VAPID_SUBJECT:-mailto:admin@example.com}"
PUSH_IGNORE_USERS: "${PUSH_IGNORE_USERS:-}"
# Public base URL (optional): external alert links point back here.
APP_URL: "${APP_URL:-}"
# Extra notification channels; each is active once its config is set.
TELEGRAM_BOT_TOKEN: "${TELEGRAM_BOT_TOKEN:-}"
TELEGRAM_CHAT_ID: "${TELEGRAM_CHAT_ID:-}"
PUSHOVER_APP_TOKEN: "${PUSHOVER_APP_TOKEN:-}"
PUSHOVER_USER_KEY: "${PUSHOVER_USER_KEY:-}"
DISCORD_WEBHOOK_URL: "${DISCORD_WEBHOOK_URL:-}"
# Libraries hidden from Trending/Most Watched (also editable in Settings)
TRENDING_EXCLUDE_LIBRARIES: "${TRENDING_EXCLUDE_LIBRARIES:-}"
# Optional login gate (recommended when internet-facing)
AUTH_ENABLED: "${AUTH_ENABLED:-false}"
```

```
      AUTH_ADMIN_USER: "${AUTH_ADMIN_USER:-}"
      AUTH_ADMIN_PASSWORD: "${AUTH_ADMIN_PASSWORD:-}"
    depends_on:
      db:
        condition: service_healthy
    volumes:
      - ./jellydash_cache:/var/www/html/cache
      - ./jellydash_runtime_cache:/var/www/html/var/cache
      - ./jellydash_logs:/var/www/html/var/log
      - ./jellydash_uploads:/var/www/html/public/uploads
      # Drop-in feature modules (see docs/MODULES.md), e.g.:
      # - ./my-modules/downloads:/var/www/html/modules/downloads:ro

  db:
    image: mariadb:11
    restart: unless-stopped
    environment:
      MARIADB_RANDOM_ROOT_PASSWORD: "yes"
      MARIADB_DATABASE: "${DB_NAME:-jellydash}"
      MARIADB_USER: "${DB_USER:-jellydash}"
      MARIADB_PASSWORD: "${DB_PASS:-change-me}"
    healthcheck:
      test: ["CMD", "healthcheck.sh", "--connect", "--innodb_initialized"]
      interval: 5s
      timeout: 5s
      retries: 12
    volumes:
      - ./jellydash_data:/var/lib/mysql
```

Gluetun

Source: [ThomasWildeTech Arr stack docker compose file]

Media Server Janitors

https://corelab.tech/maintainerr-declutarr-media-automation-guide/

3. QBittorrent

4. NZBGet

3. Radarr

1. Create Account

Now, that we have setup our Download clients i.e qBittorent and NZBGet we will configure Radarr, Sonarr, Lidarr, Bazaarr. However, we will only demonstrate one of the 4 since the configuration is exactly the same

1. Go to your ARRVM-IP :4001
2. Once, there you will start making an account. For this you can again use Bitwarden to create random login credentials for your instance

2. Media Management

1. in this step we will assign the folders in which the radarr files will reside
2. Go to Settings > Media Management > Add root folder
3. Here you will choose the movie folder you have created in your docker compose file. If you follow this guide this would be at **/data/media/Movies**
4. Congratulations, now you have told Radarr where to store all the movies.

3. Downloading Clients

Now we will connect the downloading clients qBittorent and nzbget to Radarr for it to use.

1. Go to Settings > Downloading Clients > + add
2. Select qBittorent and fill in the credentials of your qBittorent downloading client
3. Once done, add another client but this time choose for nzbget.
4. Also fill in here the credentials of your nzbget client with the category movies
5. Awesome you are all set !

4. Indexers

The final thing we need to setup for our Radarr instance are indexers. However, we will use Prowlarr for this as a centralised indexer for all ARR apps. The only thing we need for Prowlarr to connect to Radarr is it's API key.

1. Go to Settings > General
2. Copy paste the API key

10. Journey of a Media Request

Understanding how these apps communicate is the secret to a stable "Media & Automation Fortress." Here is the logical path every file takes from the moment you click "Request" to the moment it hits your screen:

1. **Media Request:** A user browses the beautiful **Seerr** (Overseerr/Jellyseerr) interface on their phone or web browser and hits "Request".
2. **Route Request:** Seerr checks its logic and sends the "Request" to **Sonarr** (if it's a TV show) or **Radarr** (if it's a movie).
3. **Search:** The *Arr app asks **Prowlarr** to find the best quality file across all your configured indexers and trackers.
4. **Cloudflare Bypass (Optional):** If a tracker is being stubborn, **FlareSolverr** steps in to solve the "human verification" challenge and pass the data back to Prowlarr.
5. **Send Grab:** Once the best file is found, the *Arr app sends a "Grab" command to your download client.
6. **Add Torrent: qBittorrent** receives the file and begins downloading it securely behind its integrated VPN.
7. **Download & Atomic Move:** Once the download is 100% complete, the *Arr app sees it and performs an **Atomic Move** (instant move) from your download folder to your library folder.
8. **Library Scan & Stream: Plex or Jellyfin** detects the new file, grabs the metadata (posters, descriptions), and notifies you that it's ready to stream!

8. Seerr

Installation

Sweet now we have our indexers setups, our downloading clients and we created quality profiles. Then now it is time to bring it all together with Seerr. Seerr allows users to request movies and tv shows from a beautiful webgui that even your grandmother can use. It is so simple you just search for a piece of media, click on request and the whole process happens in the background ;)

Note! Make sure you set the owner of the seerconfig folder to the PUID and PGID you specified in your docker compose file ;)

1. So to configure Seer we simply go to ARRVM-IP:4011

2. Then it will straight up ask you to login with Jellyfin, Emby or Plex. Choose Jellyfin because duhhh,

3. Log in with your public Jellyfin SSL address and port 443

4. Then sync your libraries e.g movies and tv shows libraries

5. Then click on run manual library scan

6. Then continue

7. Then add your Radarr credentials by filling in the Radarr API key. You can find this by going to your Radarr instance > Settings > General and scroll down where you will find the API key.

8. Choose a quality profile. For me 1080p is more than enough but if you want SD you can of course do this

9. Choose the root folder which will pop down automatically after you have clicked on test

10. Minimum availability: You can leave this to announced ;) Then people can already request a movie that is not released yet or in cinemas. Obviously they will have to wait until that movie comes available which can take some time. But the users will get a notification once the movie finally came available.

11. Also Enable scan: Scan Radarr for existing media and request status so users cannot request content already available

12. Then do the exact same steps for Sonarr.

13. Perfect then from now on your Jellyfin users can also log into Seerr and request movies and tv shows. Once, they request a movie/show it will automatically trigger radarr/sonarr and download the media according to your quality preferences.

User Request Limitations

14. Now, it is wise to let your users only request 1 movie a day and maybe one show a week. In this way you can keep your resources controlled and predictable.

1. Create a new Docker VM

We recommend a

9. Maintainerr

Maintainerr makes sure that your

Configure Jellyfin

1. Go to your Maintainerr dashboard at ArrVM-IP:4007

2. Then the first thing we need to do is to connect to your Jellyfin server with the server URL and API key

3. To obtain the Jellyfin API key go to your Jellyfin instance -> Dashboard -> API-keys and make a new API key for maintainerr

4. Then you can fill this information inside your Maintainerr Jellyfin configuration

Configure Radarr

1. To connect Radarr you need to know the IP address with port number (4002) and again an API key

2. Go to Radarr instance > Settings > General > and copy the API key

3. Then fill it into the Maintainerr Radarr configuration

Configure Sonarr

1. To connect Sonarr you need to know the IP address with port number (4001) and again an API key

2. Go to Sonarr instance > Settings > General > and copy the API key

3. Then fill it into the Maintainerr Sonarr configuration

Configure Seerr

1. To connect Seerr you need to know the IP address with port number (4011) and again an API key

2. Go to Serr instance > Settings > General > and copy the API key

3. Then fill it into the Maintainerr Sonarr configuration

9. Profilarr

let sgoo

9. JellyGlance

- **E2EE = NO**
- **Authentication:** Jellyfin Quick Connect, Local Account
- **Admin Metadata Access:** Access to everything
- **Website:** jellyglance.com
- **Repository:** Github
- **Installation Documentation:** Docker Deployment
- **Installation Videos**:
- **Community:**
- **Donations:**

Jellyglance js self-hosted dashboard for live sessions, libraries, users, requests, downloads, transcodes, invites, cleanup, Jellyfin jobs, health checks, backups, imports, newsletters, and webhooks. This means you can integrate Radarr, Sonarr, Seerr etcetera

Installation

1. The part that is important for Jellyglance in the docker compose file is the following

jellyglance_compose.yml

```
  jellyglance-db:
    image: postgres:16-alpine
    container_name: jellyglance-db
    restart: unless-stopped
    shm_size: "1gb"
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: LONGASSCOMPLICATED_DATABASE_SECRET
      POSTGRES_DB: jellyglance
    volumes:
      - ./postgres-data:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready --dbname=jellyglance --username=postgres"]
      interval: 10s
      timeout: 5s
```

```
      retries: 5
    networks:
      - internal-proxy

  jellyglance:
    image: ghcr.io/nerdy-technician/jellyglance:latest
    container_name: jellyglance
    restart: unless-stopped
    depends_on:
      jellyglance-db:
        condition: service_healthy
    ports:
      - "4013:3000"
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: ALONGASSCOMPLICATED_DATABASE_SECRET
      POSTGRES_IP: jellyglance-db
      POSTGRES_PORT: 5432
      POSTGRES_DB: jellyglance
      JWT_SECRET: ALONGASSCOMPLICATED_JWT_SECRET
      TZ: Europe/Amsterdam
      CONFIG_DIR: /app/config
      BACKUP_DIR: /app/backups
    volumes:
      - ./config:/app/config
      - ./backups:/app/backups
    networks:
      - internal-proxy
```

2. Once you have it up and running you can access it on the port you have set for Jellyglance Web.

3. Then the first thing to do is to login and I would just recommend to login with Jellyfin Quick connect

4. Then you can start adding the integrations in much the same way that you added Sonarr, Radarr, etcetera to Prowlarr for instance.

10. Autopulse

```
  autopulse:
    image: ghcr.io/dan-online/autopulse:latest
    container_name: autopulse
    environment:
      - PUID=${PUID}
      - PGID=${PGID}
      - TZ=${TZ}
      - JELLYFIN_URL=https://pinx.rfeyn.org
      - JELLYFIN_API_KEY=3108e1a5ebfc4f2fb4c9b283be825441
      - WATCH_PATHS=/mnt/data/media/movies,/mnt/data/media/tv,/mnt/data/media/music
      - DEBOUNCE_SECONDS=10
      - TZ=Europe/Amsterdam
    volumes:
      - /mnt/data/media/movies:/media/movies:ro
      - /mnt/data/media/tv:/media/tv:ro
      - /mnt/data/media/music:/media/music:ro
    restart: unless-stopped
    networks:
      - internal-proxy
```

11. Wizarr

12. Bazarr (Subtitles)

Now, one of the most important plugins for your Jellyfin server will be subtitles as well. It is great if you have a movie or show but if you cannot understand what is said then well...;)

So, in this section we will explain the methods you can add subtitles to your movies and shows automatically and manually.

1. The first method will be the manual method by using the opensubtitles.com plugin
2. To install this plugin go to Dashboard > Settings >

Problems with subtitles like PGSSUB

jellyfin.org

14. Readarr

Repository: Github