Guides

How-to, step-by-step instructions for things we run and maintain in the home lab.

Introduction

Step-by-step, how-to instructions for the things we run and maintain in the home lab. Each guide is its own page — or its own chapter when it grows long enough to need one.

Pick a guide from the sidebar to get started.

Updating the Arr Stack

This guide covers how to update the services running in the arr-stack: **Prowlarr**, **Radarr**, **Sonarr (TV)**, **Sonarr (Anime)**, **qBittorrent**, and **Seerr**. All services run as Docker containers via Docker Compose on the arr-server VM.

The same pattern applies to every container in the stack — pull the new image, then recreate the container. This is generally safe: pulling doesn't touch anything running, and recreating only replaces containers whose image actually changed.

Step by step

1. Log in to the arr-server

```
ssh <user>@<arr-server>
```

2. Go to the compose directory (where `docker-compose.yml` lives)

```
cd ~/arr-stack
```

3. Pull the new images — this is *safe*, it only downloads, it doesn't touch the running containers

```
docker compose pull
```

To update only specific services, name them:

```
docker compose pull radarr sonarr-tv
```

4. Recreate the containers with the new image — this is where the brief downtime happens (usually well under a minute per service). Docker only recreates containers whose image actually changed:

```
docker compose up -d
```

Or for specific services only:

```
docker compose up -d radarr sonarr-tv
```

5. Verify everything is up and on the new version

```
docker compose ps
```

All services should show `Up`. Give them 30–60 seconds to fully initialize before checking further.

Check the version in each service's web UI (usually under **Settings → General** or the bottom of the sidebar), or via logs:

```
docker compose logs -f --tail=50
```

Updating everything at once

To update all six services in one go:

```
cd ~/arr-stack
docker compose pull
docker compose up -d
```

Service-specific notes

Service	Notes after updating
Prowlarr	Feeds indexers to Radarr and both Sonarr instances. After updating, check **Settings → Apps** to confirm all three still show as synced.
Radarr	Check **Settings → Indexers** still shows all indexers active.
Sonarr (TV) / **Sonarr (Anime)**	Two independent instances — update and verify separately. Confirm episode/series detection still works.
qBittorrent	Active torrents and settings persist across updates (stored in the config volume). Verify downloads/seeding resume normally.
Seerr	Stateless request frontend. Verify it can still reach Radarr/Sonarr and that a test request goes through.

Rolling back

If an update causes problems, you can pin back to the previous image.

1. Find the image ID/digest that was running before the update:

   ```
   docker image ls --digests | grep <service-name>
   ```

   (Do this *before* pulling if you think you might need to roll back — note the current digest first.)
2. Stop the affected service:

   ```
   docker compose stop <service-name>
   ```

3. Temporarily pin the image in `docker-compose.yml` to the old digest:

   ```
   radarr:
     image: lscr.io/linuxserver/radarr@sha256:<previous_digest>
   ```

4. Recreate with the pinned image:

   ```
   docker compose up -d <service-name>
   ```

5. Once confirmed stable, either leave it pinned or revert the compose file to `:latest` and re-test later.

If config itself got corrupted by the update (not just the binary), restore the relevant folder from a backup of `~/arr-stack/config/`.

Backing up before updates

Config for every service lives under `~/arr-stack/config/<service>/`. A quick manual backup before a risky update:

```
tar -czf ~/arr-stack-backup-$(date +%Y%m%d-%H%M%S).tar.gz -C ~/arr-stack config/
```

If something won't come back up

1. Check the logs for that specific service:

   ```
   docker compose logs <service-name>
   ```

2. Check disk space — a full disk is a common cause of failed startups:

```
df -h ~/arr-stack
```

3. Check config folder permissions look sane:

```
ls -la ~/arr-stack/config/
```

4. Try a plain restart before anything more drastic:

```
docker compose restart <service-name>
```

Update frequency

- All images here track `:latest`, so there's no fixed release cadence to wait for — updates just mean whatever the maintainers shipped since your last pull.
- For routine point releases, it's fine to just update.
- For anything that looks like a major version bump in the release notes, skim the changelog first — Prowlarr/Radarr/Sonarr occasionally have schema or config-format changes worth knowing about ahead of time.

Further reading

- LinuxServer.io image docs: https://docs.linuxserver.io/
- Radarr releases: https://github.com/Radarr/Radarr/releases
- Sonarr releases: https://github.com/Sonarr/Sonarr/releases
- Prowlarr releases: https://github.com/Prowlarr/Prowlarr/releases
- qBittorrent releases: https://github.com/qbittorrent/qBittorrent/releases
- Seerr releases: https://github.com/seerr-team/seerr/releases

qBittorrent: migrating seeding torrents from Windows to Linux

A guide for moving a large set of seeding torrents from a qBittorrent client on **Windows** to **qBittorrent in Docker on Linux** — without breaking the seeding (which matters for private trackers like TorrentLeech and their Hit-and-Run rules).

> Done 2026-06-29: 193 torrents (movies, anime, TV, PC games, OS ISOs) migrated from a Windows server to Docker qBittorrent on the arr-server. All 193 are seeding, verified to survive a reboot.

Prerequisites

- The source files already live on the NAS (the same physical files the Linux client will seed).
- Both the Windows and Linux qBittorrent run a **standard release** (not a beta or fork) — otherwise the tracker may ban the client. qBittorrent 5.2.2 is fine for TorrentLeech.
- The torrents have already met the tracker's minimum seed time (if not: migrate anyway, but be aware of the Hit-and-Run risk).

NAS layout (NFS on the arr-server)

NAS	Export	Mount point
NAS01	`/volume1/Movies`	`/mnt/media/movies`
NAS01	`/volume1/Apps`	`/mnt/media/apps`
NAS02	`/volume1/TV`	`/mnt/media/tv`
NAS02	`/volume1/Anime`	`/mnt/media/anime`
NAS02	`/volume1/Games-PC`	`/mnt/media/games-pc`

Each share is in `/etc/fstab` with `_netdev` and bind-mounted into the qBittorrent container (`/mnt/media/x:/mnt/media/x`).

Step 1 — Export from Windows

The `.torrent` and `.fastresume` files live in:

```
C:\Users\<user>\AppData\Local\qBittorrent\BT_backup\
```

Copy the **entire** folder (both the `.torrent` AND the `.fastresume` files) to somewhere Linux can reach, e.g. a temp folder on the NAS.

> **Note:** always keep an untouched copy as a "pristine source" — never touch it, always recreate from it.

Step 2 — Docker volumes

The qBittorrent container must see the files at the **same path** the `.fastresume` points to. Add the NAS mounts to `docker-compose.yml`:

```
  qbittorrent:
    volumes:
      - ./config/qbittorrent:/config
      - /mnt/media/movies:/mnt/media/movies
      - /mnt/media/tv:/mnt/media/tv
      # ...and so on for each share
```

Step 3 — Rewrite the paths in .fastresume

`.fastresume` is **bencode-encoded** (binary, with length prefixes). A plain `sed` will NOT work — you have to decode, swap the fields, and re-encode. Two fields hold paths:

- `save_path` (libtorrent's — the one that actually counts)

- `qBt-savePath` (qBittorrent's own)

Mapping rule (Windows → Linux), prefix-preserving:

Windows save_path	Linux
`//nas01/Movies` (movie in a subfolder)	`/mnt/media/movies`
`//nas01/Movies/<movie>` (file directly)	`/mnt/media/movies/<movie>`
`//nas02/Anime/<x>`	`/mnt/media/anime/<x>`
`//nas02/TV/<x>`	`/mnt/media/tv/<x>`

> **Important:** don't blanket-swap everything to the same folder. If the `save_path` includes the movie name, that means the content is the file directly — in which case the subfolder has to be preserved.

The Python script uses its own bencode decode/encode with a **safety check**: `encode(decode(file)) == file` must be byte-identical before anything is written (otherwise the encoder isn't canonical → abort). Stop qBittorrent before editing the files (it overwrites `.fastresume` on shutdown).

Step 4 — Load them in and recheck

1. Copy the `.torrent` files + the rewritten `.fastresume` files into the Linux qBittorrent's `BT_backup`.
2. `docker compose up -d qbittorrent` (recreate if volumes changed).
3. qBittorrent rechecks and starts seeding. Torrents that find their files go to `stalledUP` / `uploading`.

Common errors:

- `missingFiles` = qBittorrent can't find the files → wrong path, a missing docker volume, or a folder name that doesn't match (e.g. a dedup that renamed it).
- **Trailing space in a folder name:** Windows allows/strips trailing spaces differently from Linux. Fix it qBittorrent-native with the `torrents/renameFolder` API (do NOT rename the folder on disk — Radarr/Jellyfin expect it without the space).

Step 5 — Turn off the seeding queue

For a seedbox that should seed everything forever: turn off queueing so **all** torrents announce at once.

```
POST /api/v2/app/setPreferences   json={"queueing_enabled":false}
```

Otherwise `max_active_uploads` / `max_active_torrents` limit how many actually seed.

Step 6 — Make it reboot-safe (critical!)

An fstab entry with `_netdev` survives a reboot — **but Docker doesn't wait for the NFS mounts by default.** If Docker starts qBittorrent before NFS is ready, empty folders get bind-mounted → every torrent becomes `missingFiles`.

Fix — a systemd drop-in at `/etc/systemd/system/docker.service.d/wait-for-nfs.conf`:

```
[Unit]
After=remote-fs.target
Wants=remote-fs.target
```

Then `systemctl daemon-reload`. This orders Docker after all `_netdev` mounts are ready.

Verify with an actual reboot: after restarting, all mounts should be active, qBittorrent up, and every torrent seeding (0 `missingFiles`).

Verification commands

```
# Mounts active?
findmnt -t nfs4 -o TARGET,SOURCE

# fstab valid?
sudo findmnt --verify
```

```
# Can qBittorrent see the files?
docker exec qbittorrent ls /mnt/media/movies

# Torrent states (via the API, after getting a login cookie)
curl -s -b cookie 'http://localhost:8080/api/v2/torrents/info' \
  | grep -o '"state":"[^"]*"' | sort | uniq -c

# Tracker announcing? (status 2 = working)
curl -s -b cookie 'http://localhost:8080/api/v2/torrents/trackers?hash=<HASH>'
```