

Jellyfin

- [Installation](#)
- [Basic Configuration](#)
- [Hardware Acceleration \(GPU\)](#)
- [Metadata](#)
- [Design Tweaks](#)

Installation

1. Install Packages

```
sudo apt install apt-transport-https gnupg lsb-release
curl -fsSL https://repo.jellyfin.org/debian/jellyfin_team.gpg.key | gpg --dearmor -o /etc/apt/trusted.gpg.d/debian-jellyfin.gpg
echo "deb [arch=$( dpkg --print-architecture )] https://repo.jellyfin.org/debian $( lsb_release -c -s ) main" | sudo tee /etc/apt/sources.list.d/jellyfin.list
```

```
sudo apt update
sudo apt install jellyfin
```

```
sudo systemctl enable --now jellyfin.service
sudo systemctl restart jellyfin.service
```

2. Setup reverse proxy

In your new Jellyfin installation, head over to the `Admin Dashboard` -> `Advanced` -> `Networking` and disable HTTPS (if enabled), then add your local host (`127.0.0.1`) to the known proxies, to allow NGINX to act as a reverse proxy. After that, restart your Jellyfin server.

```
sudo systemctl restart jellyfin
```

Create a new VHOST in NGINX, `/etc/nginx/sites-enabled/streaming.example.org` and insert the following configuration (adapting the servername etc. of course):

```
upstream jellyfin {
    server 127.0.0.1:8096;
}

server {
    listen 443 ssl http2;
    listen [::]:443 ssl http2;
    server_name streaming.example.org;
    ssl_certificate /etc/ssl/...;
```

```
ssl_certificate_key /etc/ssl/...;
ssl_trusted_certificate /etc/ssl/...;

location / {
    proxy_pass http://jellyfin;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto https;
    proxy_set_header X-Nginx-Proxy true;
    proxy_redirect off;
}

}

server {
    listen 80;
    listen [::]:80;
    server_name streaming.example.org;
    return 301 https://$server_name$request_uri;
}
```

You can now reach your Jellyfin instance over streaming.example.org.

Basic Configuration

1. Creating a library

Before you can start adding media to your server, you have to create a library. You can choose between different content types. Select the one appropriate for the content you want to add. I will start by creating a library for my movies, so I choose movies from the drop-down and continue. Most of the library settings are self-explanatory. Some options, however, are worth mentioning, so here is a small overview:

Option	Recommendation
`Enable chapter image extraction`	Only enable this option, if you have a rather powerful CPU and on top of that, lots of free storage space. If you do have, this will give you preview images, when skipping through a video.
`Extract chapter images during the library scan`	You can safely disable this option, because using plugins like Fanart will provide much better images, for all the different display sizes (Banner, Poster, etc.)

2. Adding media

To add media, e.g. a movie, make sure, that you have the necessary storage capacity on the path, you specified in your library settings. After that, I would recommend creating a new folder for every movie, which you can organize in subfolders, if needed (e.g., for a movie series). Here is a small tree view of how I organize my movies:

```
|-- Movie_001
|  |-- 1.\ first\ in\ the\ series
|  |  |-- Some_Movie_Part1 - 1080p - [imdbid-tt00000000].eng.srt
|  |  |-- Some_Movie_Part1 - 1080p - [imdbid-tt00000000].ger.srt
|  |  |-- Some_Movie_Part1 - 1080p - [imdbid-tt00000000].mkv
|  |  |-- Some_Movie_Part1 - 1080p - [imdbid-tt00000000].nfo
|  |  |-- backdrop.jpg
|  |  |-- banner.jpg
|  |  |-- clearart.png
|  |  |-- disc.png
|  |  |-- folder.jpg
|  |  |-- landscape.jpg
|  |  |-- `-- logo.png
```

```
| |-- 2.\ second\ in\ the\ series
| | |-- Some_Movie_Part2 - 1080p - [imdbid-tt0000001].eng.srt
| | |-- Some_Movie_Part2 - 1080p - [imdbid-tt0000001].ger.srt
| | |-- Some_Movie_Part2 - 1080p - [imdbid-tt0000001].mkv
| | |-- Some_Movie_Part2 - 1080p - [imdbid-tt0000001].nfo
| | |-- backdrop.jpg
| | |-- banner.jpg
| | |-- clearart.png
| | |-- disc.png
| | |-- folder.jpg
| | |-- landscape.jpg
| | |-- logo.png
|-- Movie_002
| |-- Some_Other_Movie - 1080p - [imdbid-tt0000002].ger.srt
| |-- Some_Other_Movie - 1080p - [imdbid-tt0000002].mkv
| |-- Some_Other_Movie - 1080p - [imdbid-tt0000002].nfo
| |-- backdrop.jpg
| |-- disc.png
| |-- folder.jpg
| |-- logo.png
```

As you can see, you can specify many options in the filename, to make Jellyfin aware of certain properties. Here are a few:

Option	What it does
`[tmdbid=xyz]` / `[imdbid=xyz]`	Specify which IMDB/TMDBID the movie has, so the according metadata gets fetched automatically.
`.sample` `.trailer` `.theme` `.interview` `.featurette`	Let Jellyfin know, that the file is a sample/trailer/theme/interview/featurette file.
`[1080p]` `[720p]` `[480p]`	Usually unnecessary, but in case, it's missing in the files metadata, let Jellyfin know, what resolution the video has.

FYI, instead of giving your videos file extensions like `.trailer` you can also place the files in subfolders, to keep it more structured. Supported names for subfolders are:

- `behind the scenes`
- `deleted scenes`
- `interviews`
- `scenes`
- `samples`
- `shorts`

- `featurettes`
- `extras` - Generic catch-all for extras of an unknown type.
- `trailers`

You can also find information on the above mentioned topics in the Jellyfin documentation:

<https://jellyfin.org/docs/general/server/media/movies.html>

Hardware Acceleration (GPU)

This section only covers setup for Nvidia GPUs

1. Finding a suitable GPU

The recommended way for hardware acceleration is to use a dedicated GPU. The usual manufacturers are AMD and Nvidia. However, experience has shown, that despite Nvidia's flawed Linux support regarding drivers, it is still the more performant and stable option, for transcoding media files (specifically HEVC). If you are wondering, which GPU is affordable and also supports all the common codecs, I can recommend you to take a look at the following matrix provided by Nvidia: <https://developer.nvidia.com/video-encode-and-decode-gpu-support-matrix-new>. Make sure, that your GPU supports at least H.265 (4K YUV 4:2:0), since there are still a lot of clients, that don't support the rather new codec. Keep in mind, that using an old "Gaming" GPU might not be the best option for you, because they consume a lot of power and furthermore are locked to a maximum of 3 concurrent sessions. So if you have more than three people streaming video at any given time, you'll run into problems. This leaves Nvidia's Professional/Datacenter GPU's. My recommendation for a cheap, low power consuming and free of restrictions GPU is the [Nvidia Quadro P2000](#). With its 5GB of GDDR5 VRAM, it has enough power, to allow a whole family to stream HQ content. If you know, that you don't exceed the session limit, the cheaper [Nvidia Quadro P400](#) might be of interest to you. With that out of the way, you can continue to install the Nvidia drivers.

2. Installing NVIDIA drivers

Before installing the drivers, make sure, that you meet all requirements:

- ☒ Supported kernel version (<https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html#system-requirements>)
- ☒ CUDA capable GPU
- ☒ NVENC and NVDEC capable GPU

To start off, make sure, that your `/etc/apt/sources.list` file includes the `contrib` and `non-free` repositories. E.g. the following:

```
deb http://deb.debian.org/debian/ bullseye main contrib non-free
deb-src http://deb.debian.org/debian/ bullseye main contrib non-free
deb http://deb.debian.org/debian/ bullseye-updates main contrib non-free
deb-src http://deb.debian.org/debian/ bullseye-updates main contrib non-free
```

Next, update all packages and install the `nvidia-driver` package.

```
apt update
apt -y install nvidia-driver firmware-misc-nonfree
```

Make sure, that the default nouveau drivers are blacklisted, by running `cat /etc/modprobe.d/nvidia-blacklists-nouveau.conf`

```
# You need to run "update-initramfs -u" after editing this file.

# see #580894
blacklist nouveau
```

If all is set, reboot your machine. You should be able, to verify the loaded drivers now, by running `lsmod | grep nvidia`:

```
nvidia_uvm      1273856  0
nvidia_drm      73728  0
drm_kms_helper  278528  1 nvidia_drm
nvidia_modeset  1146880  1 nvidia_drm
nvidia          40828928  2 nvidia_uvm,nvidia_modeset
drm             618496  4 drm_kms_helper,nvidia,nvidia_drm
```

Additionally, for monitoring purposes, you can install `nvidia-smi`. Simply run the following command:

```
apt -y install nvidia-smi
```

This allows you, to see your GPU's current status:

```
~# nvidia-smi
Mon Jul 11 13:33:22 2022

+-----+
| NVIDIA-SMI 515.48.07    Driver Version: 515.48.07    CUDA Version: 11.7    |
+-----+-----+-----+
| GPU  Name      Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|                               |              MIG M. |
+=====+
| 0  Quadro P400      Off | 00000000:03:00.0 Off |          N/A |
| 45%  49C   P0   N/A /  N/A |    0MiB / 2048MiB |      0%      Default |
```


								N/A	
+-----+-----+-----+									
+-----+-----+-----+									
Processes:									
GPU	GI	CI	PID	Type	Process name	GPU Memory			
	ID	ID				Usage			
=====									
=====									
No running processes found									
+-----+-----+-----+									

2.1 Installing CUDA Toolkit

This only works on systems with x86_64 architecture

First, remove outdated signing keys, by running

```
apt-key del 7fa2af80
```

After that, setup the CUDA repository:

```
wget https://developer.download.nvidia.com/compute/cuda/repos/<distro>/<arch>/cuda-keyring_1.0-1_all.deb
```

```
dpkg -i cuda-keyring_1.0-1_all.deb
```

Update your repositories and install CUDA:

```
apt update
apt -y install cuda
```

After a successful installation, reboot your system.

2.2 Compiling FFmpeg

Before you start, install the following packages, if missing:

```
apt -y install build-essential yasm cmake libtool libc6 libc6-dev unzip wget libnuma1 libnuma-dev pkg-config
```

After that, clone the ffnvcodec and FFmpeg repository.

```
git clone https://git.videolan.org/git/ffmpeg/nv-codec-headers.git
git clone https://git.ffmpeg.org/ffmpeg.git ffmpeg/
```

Install ffnvcode:

```
cd nv-codec-headers && sudo make install && cd ../ffmpeg
```

Configure FFmpeg:

```
./configure --enable-nonfree --enable-cuda-nvcc --enable-libnpp --extra-cflags=-I/usr/local/cuda/include --extra-ldflags=-L/usr/local/cuda/lib64 --disable-static --enable-shared
```

Compile FFmpeg:

```
make -j 8 # Change according to available Threads!
```

Install libraries

```
make install
```

Check your installation, by running `ffmpeg`. If you run into the following error

```
ffmpeg: error while loading shared libraries: libavdevice.so.52: cannot open shared object file: No such file or directory
```

you need to make a change to the `/etc/ld.so.conf` config file. First, find out, where the library is located:

```
find / -name "libavdevice.so.52"
```

This returns `/usr/local/lib` as the location. Simply add this information to the above config file and run

```
ldconfig
```

You should now be able to enable Hardware Acceleration in Jellyfin. Head over to your Jellyfin Web GUI and navigate to `Administration` -> `Dashboard` -> `Playback` and set `Hardware Acceleration` to `Nvidia NVENC`. Depending on your GPU, you can enable hardware decoding for the supported Codecs. To test, if the GPU is actually doing the work, run a movie, and check `nvidia-smi` for running processes:

```
~# nvidia-smi
Mon Jul 11 13:50:25 2022
```

```
+-----+
| NVIDIA-SMI 515.48.07   Driver Version: 515.48.07   CUDA Version: 11.7   |
+-----+-----+-----+
| GPU  Name      Persistence-M| Bus-Id      Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|                               |              MIG M. |
+=====+=====+=====+
=====|
|  0  Quadro P400      Off | 00000000:03:00.0 Off |          N/A |
| 37%   50C   P0   N/A /  N/A |  823MiB /  2048MiB |   66%    Default |
|                               |              N/A |
+-----+-----+-----+

+-----+
| Processes:                                |
| GPU  GI  CI       PID   Type   Process name          GPU Memory |
|      ID  ID                                   Usage      |
+=====+=====+=====+
=====|
|  0  N/A  N/A   61181    C   ...ib/jellyfin-ffmpeg/ffmpeg    821MiB |
+-----+-----+-----+
```

Metadata

1. Metadata Providers

Every so often, Jellyfin does not automatically detect the movie/show you just added to your library. In that case, you can use the Internet Movie Database (IMDB), as well as The Movie Database (TMDb) as metadata providers. Simply head over to each site, search for your movie and in the URL you will find the `movie ID` string, you need to specify in Jellyfin.

E.g. for 2001: A Space Odyssey the URLs look like this:

https://www.imdb.com/title/tt0062622/?ref_=fn_al_tt_1

<https://www.themoviedb.org/movie/62-2001-a-space-odyssey>

The highlighted strings are the IDs of the movie, which are unique identifiers, allowing Jellyfin to lookup all the missing metadata on them. If you are adding a movie series, e.g. Harry Potter, you can also specify a `TheMovieDb Box Set Id`. Luckily, you don't have to do that manually. Simply provide the TMDb movie ID as usual and the Box Set ID will be added automatically (if the movie is part of a series).

2. Manually editing a files Metadata

Sometimes when you acquire a video source, it can happen, that the metadata tags for it's video etc. are packed with unnecessary comments and ads. To remove them, you can use mkvtoolnix for `MKV` files. Simply run the following command, replacing the movie name with the one you want to query:

```
mkvinfo 'Filmname'
```

The output for then looks something like this:

```
+ EBML head
|+ EBML version: 1
|+ EBML read version: 1
|+ Maximum EBML ID length: 4
|+ Maximum EBML size length: 8
|+ Document type: matroska
|+ Document type version: 4
```

```
|+ Document type read version: 2
+ Segment: size 1820877305
|+ Seek head (subentries will be skipped)
|+ EBML void: size 4044
|+ Segment information
| + Timestamp scale: 1000000
| + Multiplexing application: libebml v1.3.0 + libmatroska v1.4.1
| + Writing application: mkvmerge v6.5.0 ('Some Text') built on Dec 31 0000 00:00:00
| + Duration: 01:30:00.000000000
| + Date: Thu Dec 31 00:00:00 0000 UTC
| + Segment UID: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
|+ Tracks
| + Track
|   + Track number: 1 (track ID for mkvmerge & mkvextract: 0)
|   + Track UID: 00000000000000000000
|   + Track type: video
|   + Lacing flag: 0
|   + Minimum cache: 1
|   + Codec ID: V_MPEG4/ISO/AVC
|   + Codec's private data: size 50 (h.264 profile: High @L4.1)
|   + Default duration: 00:00:00.041708333 (23.976 frames/fields per second for a video track)
|   + Language: und
|   + Name: Some unwanted information
|   + Video track
|     + Pixel width: 1920
|     + Pixel height: 1080
|     + Display width: 1920
|     + Display height: 1080
|   + Track
|     + Track number: 2 (track ID for mkvmerge & mkvextract: 1)
|     + Track UID: 1353746707212659340
|     + Track type: audio
|     + Codec ID: A_AC3
|     + Default duration: 00:00:00.000000000 (31.250 frames/fields per second for a video track)
|     + Name: Some unwanted information
|     + Audio track
|       + Sampling frequency: 48000
|       + Channels: 2
|+ EBML void: size 1099
|+ Cluster
```

Now to remove the 'Some unwanted information' metadata, simply run the following commands. Please keep in mind, that you might have more or less tracks containing the metadata you want to remove.

```
mkvpropedit "Dateiname" --edit track:1 --edit track:2 --set name=""
```

If you want to remove known metadata from a series of files, e.g. episodes of a show, you can also make a small script:

```
#!/bin/bash  
  
for i in *.mkv; do  
    [ -f "$i" ] || break  
    mkvpropedit "$i"  
    --edit track:1 --edit track:2 --set name=""  
done
```

If you want to edit the metadata of e.g., an MP4 file, you can use the version of `ffmpeg` Jellyfin installed in `/usr/lib/jellyfin-ffmpeg/ffmpeg`

Design Tweaks

1. Custom Theme/Skin

You can either install the [SkinManager](#) Plugin to try out a new look for Jellyfin, or directly edit CSS in the `Custom CSS` section under the `General` settings. I experienced the SkinManager as pretty unstable, so I chose to use the second option.

```
@import url("https://cdn.jsdelivr.net/gh/prayag17/JellySkin@latest/default.css");
@import url("https://cdn.jsdelivr.net/gh/prayag17/JellySkin/default.css");
@import url("https://cdn.jsdelivr.net/gh/prayag17/JellySkin@latest/addons/Logo.css");
@import url("https://cdn.jsdelivr.net/gh/prayag17/JellySkin/addons/Logo.css");
@import url("https://cdn.jsdelivr.net/gh/prayag17/JellySkin/addons/imp-per.css");
@import url("https://cdn.jsdelivr.net/gh/prayag17/JellySkin/addons/progress-bar.css");

.btnForgotPassword,
.btnQuick {
  display: none !important;
}
```

You can find the theme here: <https://github.com/prayag17/JellySkin>

2. Enable Backdrops for all users

Sadly, you cannot enable backdrops for all users from the settings. However, you can do this, by editing the `bundle.js` file. To locate the file, run

```
find / -name *.bundle.js | grep jellyfin
=> /usr/share/jellyfin/web/main.jellyfin.bundle.js
```

Next, edit the file (here `/usr/share/jellyfin/web/main.jellyfin.bundle.js`) and replace this line

```
enableBackdrops:function(){return P}
```

with the following one:

```
enableBackdrops:function(){return x}
```

After that, restart Jellyfin and clear it's cache.