# Dolphin pour RomM

# RomM — Emulator Streaming avec Dolphin sur Unraid (GPU NVIDIA)

Ce tutoriel explique comment ajouter un conteneur **Dolphin** (GameCube/Wii) à une instance **RomM 5.1.0+** existante pour utiliser la fonctionnalité **Emulator Streaming** : lancer un jeu depuis l'interface RomM et y jouer dans le navigateur, avec rendu et encodage sur le GPU NVIDIA.

**Architecture :** RomM ne pilote pas Dolphin directement. Un *Docker Mod* (maintenu par LoneAngelFayt) injecte dans le conteneur Dolphin un **broker HTTP** (port 8000) que RomM contacte pour lancer/piloter les jeux. L'image et le son sont diffusés par **Selkies** (le bureau web intégré aux images LinuxServer), en H.264 NVENC.

> ⚠ Testé avec : Unraid 7, RTX 3060, driver NVIDIA 610.x, `lscr.io/linuxserver/dolphin:latest` (base Selkies/Wayland), RomM 5.1.0.

---

# 1. Prérequis Unraid

## 1.1 Plugin Nvidia Driver

- Installer le plugin **Nvidia Driver** (Community Applications).
- Driver propriétaire **580 ou supérieur** requis. LinuxServer recommande la **branche Production** pour Unraid.
- Le GPU ne doit **pas** être réservé au VFIO (il doit être disponible pour Docker).

- Sur un serveur *headless* (sans écran), la doc LinuxServer indique qu'un **dummy plug HDMI/DP** branché sur la carte est requis pour que DRM s'initialise correctement.

# 1.2 Paramètres noyau (Syslinux)

Menu **Main → Flash → Syslinux Config** (vue *Raw*), ajouter à la fin de la ligne `append` de l'entrée « Unraid OS » (et « Unraid OS GUI Mode » si utilisée) :

```
nvidia-drm.modeset=1 nvidia_drm.fbdev=1
```

Redémarrer le serveur, puis vérifier :

```
cat /proc/cmdline                                  # les 2 paramètres doivent apparaître
cat /sys/module/nvidia_drm/parameters/modeset      # doit renvoyer Y
nvidia-smi -L                                      # liste le(s) GPU + UUID
```

Notez l'**UUID du GPU** (`GPU-xxxxxxxx-...`), il servira dans le template.

# 1.3 Limite inotify (optionnel mais recommandé)

Les images Selkies consomment beaucoup d'instances inotify. Pour éviter l'avertissement « Too many open files » :

```
sysctl -w fs.inotify.max_user_instances=1024
echo 'sysctl -w fs.inotify.max_user_instances=1024' >> /boot/config/go
```

---

# 2. ⚠ Correctifs NVIDIA indispensables (Unraid)

Le runtime NVIDIA d'Unraid présente **deux trous d'injection** vers les conteneurs. Sans eux, l'émulateur tourne en rendu logiciel (très lent) ou affiche un écran noir. Les deux correctifs sont simples et persistants.

# 2.1 Composant egl-x11 (rendu OpenGL/EGL)

**C'est le point qui fait échouer la plupart des installations** (symptôme : interface Dolphin visible, mais **zone de jeu noire** avec le son, ou jeu qui rame énormément en rendu logiciel).

Le runtime NVIDIA d'Unraid injecte dans les conteneurs les plateformes EGL `wayland`, `wayland2` et `gbm`, **mais pas les deux plateformes X11** (`xlib`/`xcb`) — pourtant livrées par le driver et indispensables à Dolphin, qui crée son contexte OpenGL via **EGL sur X11**. Sans elles, l'EGL NVIDIA ne peut pas créer de surface et Dolphin retombe en rendu logiciel (ou écran noir via Zink : `MESA: error: zink: could not create swapchain`).

Vérifier que le driver de l'hôte fournit bien les fichiers :

```
ls /var/local/overlay/usr/lib64/ | grep egl-x
ls /var/local/overlay/usr/share/egl/egl_external_platform.d/ | grep -E "xlib|xcb"
```

Vous devez voir `libnvidia-egl-xlib.so.1.X.X`, `libnvidia-egl-xcb.so.1.X.X`, `20_nvidia_xlib.json` et `20_nvidia_xcb.json`. **Notez le numéro de version exact des `.so`** (ex. `1.0.5`) : il est utilisé dans les montages ci-dessous, et il **change à chaque mise à jour du driver** (voir §7).

Le correctif consiste en **4 montages en lecture seule** + **1 variable**, intégrés au template Dolphin de la section suivante.

# 2.2 Nœud `/dev/nvidia-modeset` (présentation Vulkan)

Le toolkit injecte `nvidia0`, `nvidiactl`, `nvidia-uvm`, `nvidia-uvm-tools` et `nvidia-caps`... **mais pas `/dev/nvidia-modeset`**, qui est nécessaire à la *présentation* Vulkan (l'énumération des GPU, elle, fonctionne sans lui — d'où des diagnostics trompeurs).

Symptômes sans ce nœud : `vulkaninfo` liste bien la carte mais `vkGetPhysicalDeviceSurfacePresentModesKHR` échoue en `ERROR_UNKNOWN` ; `vkcube` sélectionne le GPU puis plante à la création de la swapchain ; Dolphin en backend Vulkan ou via Zink affiche un **écran noir avec le son**.

Vérification :

```
ls /dev/nvidia-modeset                                  # sur l'hôte : présent
docker exec dolphin ls /dev/nvidia-modeset 2>&1         # dans le conteneur : absent
```

Correctif : ajouter `--device /dev/nvidia-modeset` aux Extra Parameters (voir §3.4).

---

# 3. Conteneur Dolphin

**Ajouter un conteneur** avec l'image `lscr.io/linuxserver/dolphin` et la configuration suivante.

## 3.1 Générer un secret partagé

Ce secret authentifie les échanges RomM ↔ broker. Générez-le une fois :

```
openssl rand -hex 32
```

## 3.2 Ports (host:container)

| Port hôte | Port conteneur | Rôle |
|---|---|---|
| 3000 | 3000 | Selkies HTTP (peu utile) |
| 3001 | 3001 | **Selkies HTTPS** (le flux de jeu) |
| 8000 | 8000 | **Broker RomM** |

Adaptez les ports hôte s'ils sont déjà pris par un autre conteneur (dans ce cas, reportez le port choisi dans le bloc `streaming` du §4.2).

## 3.3 Variables d'environnement

| Variable | Valeur | Remarque |
|---|---|---|
| `DOCKER_MODS` | `ghcr.io/loneangelfayt/dolphin-romm-integration-mod:latest` | installe le broker |
| `ROM_ROOT` | `/romm/library` | racine de la bibliothèque vue par le broker |
| `BROKER_SECRET` | *(votre secret)* | même valeur que côté RomM |
| `NVIDIA_VISIBLE_DEVICES` | *(UUID du GPU)* | ou `all` si un seul GPU |
| `NVIDIA_DRIVER_CAPABILITIES` | `all` | |
| `DRINODE` | `/dev/dri/renderD128` | GPU de **rendu** (vérifier avec `ls -l /dev/dri/by-path/` que c'est bien la carte NVIDIA) |

| Variable | Valeur | Remarque |
|---|---|---|
| `DRI_NODE` | `/dev/dri/renderD128` | GPU d'**encodage** — identique = mode Zero-Copy |
| `SELKIES_MANUAL_WIDTH` | `1920` | bride la résolution du flux |
| `SELKIES_MANUAL_HEIGHT` | `1080` | sinon elle suit la fenêtre du navigateur (coûteux) |
| `__EGL_VENDOR_LIBRARY_FILENAMES` | `/usr/share/glvnd/egl_vendor.d/10_nvidia.json` | force l'EGL NVIDIA (fait partie du correctif §2) |
| `PUID` / `PGID` | `99` / `100` | standard Unraid |

# 3.4 Extra Parameters

```
--gpus all --runtime nvidia --shm-size=1gb --device /dev/nvidia-modeset --cpuset-cpus=0-5,12-17
```

- `--device /dev/nvidia-modeset` : correctif §2.2.
- `--cpuset-cpus` : épinglage CPU, **obligatoire sur tout Ryzen multi-CCD** — sans lui les performances sont fortement dégradées (voir §7). Adaptez impérativement la liste à votre processeur : la valeur ci-dessus correspond au premier CCD d'un 3900X et sera **contre-productive** sur une autre topologie. Sur un CPU mono-CCD ou Intel, retirez l'option.

# 3.5 Volumes

| Host Path | Container Path | Mode |
|---|---|---|
| *(votre dossier de ROMs — le même que celui monté dans RomM)* | `/romm/library` | **ro** |
| *(votre appdata, ex. `/mnt/user/appdata/dolphin`)* | `/config` | rw |
| `/var/local/overlay/usr/lib64/libnvidia-egl-xlib.so.1.X.X` | `/usr/lib64/libnvidia-egl-xlib.so.1` | **ro** |
| `/var/local/overlay/usr/lib64/libnvidia-egl-xcb.so.1.X.X` | `/usr/lib64/libnvidia-egl-xcb.so.1` | **ro** |
| `/var/local/overlay/usr/share/egl/egl_external_platform.d/20_nvidia_xlib.json` | `/usr/share/egl/egl_external_platform.d/20_nvidia_xlib.json` | **ro** |
| `/var/local/overlay/usr/share/egl/egl_external_platform.d/20_nvidia_xcb.json` | `/usr/share/egl/egl_external_platform.d/20_nvidia_xcb.json` | **ro** |

> Remplacez `1.X.X` par la version relevée au §2. Le Container Path des libs est volontairement le nom **court** `.so.1` : c'est celui que le loader EGL recherche

> (déclaré dans les JSON), ce qui évite d'avoir à créer des symlinks.

> ⚠️ **Important :** le chemin de la bibliothèque doit être **identique** dans RomM et dans Dolphin (`/romm/library` des deux côtés), sinon le broker ne retrouvera pas les fichiers que RomM lui demande de lancer.

## 3.6 Vérifications après démarrage

```
docker logs dolphin 2>&1 | grep -iE "broker|NVENC|Zero-Copy"
```

Attendu : `ROM broker listening on port 8000`, `Shared secret auth enabled`, `NVENC ... Initialized successfully`, `Decision: Zero-Copy path active`.

```
curl -s http://localhost:8000/health        # → {"status":"ok"}
docker exec dolphin sh -c 'ldconfig -p | grep egl-x'   # → les 2 libs xlib/xcb
```

---

# 4. Côté RomM

## 4.1 Variable d'environnement

Ajouter au conteneur RomM :

| Variable | Valeur |
|---|---|
| `STREAMING_BROKER_SECRET` | *(le même secret qu'au §3.1)* |

## 4.2 config.yml

Dans le `config.yml` de RomM (dossier config), ajouter le bloc `streaming` (adapter l'IP du serveur et les ports choisis) :

```
streaming:
  enabled: true
  containers:
```

```
  - platform: ngc
    host: https://IP_DU_SERVEUR:3001
    broker_host: http://IP_DU_SERVEUR:8000
    label: Dolphin
    memory_card_sync: true
  - platform: wii
    host: https://IP_DU_SERVEUR:3001
    broker_host: http://IP_DU_SERVEUR:8000
    label: Dolphin
```

Redémarrer RomM après modification.

# 4.3 Structure de la bibliothèque

- Le dossier de plateforme doit correspondre au **fs_slug** de RomM : `ngc` pour la GameCube, `wii` pour la Wii (pas `gc`, pas `gamecube`) — sinon prévoir un mappage dans `config.yml`.
- Les ROMs doivent être des **fichiers directs** (`.rvz`, `.iso`) — **pas d'archives** `.zip`/`.7z`, Dolphin ne peut pas les lancer via le broker.
- ⚠ RomM supporte deux structures (`library/roms/<plateforme>/` ou `library/<plateforme>/roms/`) mais **la simple présence d'un dossier `library/roms/` fait basculer toute la bibliothèque sur la première** : ne mélangez pas les deux, ou toutes vos autres plateformes deviendront invisibles au scan.

Rescanner la plateforme dans RomM après tout changement.

---

# 5. Premier lancement

1. Ouvrir une première fois `https://IP_DU_SERVEUR:3001` et **accepter le certificat auto-signé** (sinon le flux ne se chargera pas dans RomM).
2. Dans RomM, ouvrir la fiche d'un jeu GameCube → bouton **Play on Dolphin**.
3. Le jeu doit s'afficher, fluide, avec le son.

**Vérifier que le GPU est bien utilisé** (pendant qu'un jeu tourne) :

```
docker exec -u abc dolphin sh -c 'for p in /proc/[0-9]*; do ls -l $p/fd 2>/dev/null | grep -q nvidia && echo "$(basename $p) $(tr "\0" " " < $p/cmdline | cut -c1-60)"; done'
```

`dolphin-emu` **doit** apparaître dans la liste. S'il n'y a que `selkies`, `labwc` et `Xwayland`, Dolphin est en rendu logiciel → revoir le §2.

> Note : lancez bien ce test avec `-u abc` — en root, la lecture de `/proc/<pid>/fd` des processus de l'utilisateur `abc` échoue silencieusement.

---

# 6. Réglages Dolphin utiles

Le fichier de configuration est `<appdata>/.config/dolphin-emu/Dolphin.ini`. **Toujours éditer conteneur arrêté** (Dolphin réécrit ses fichiers en se fermant), et savoir que **le mod réimpose `GFXBackend = OpenGL` à chaque recréation du conteneur** — c'est le bon backend ici (c'est le chemin EGL corrigé au §2), ne pas chercher à le changer.

> ⚠ **Après toute édition depuis l'hôte** (`sed -i`, `cat >`, redirection…), rétablissez le propriétaire : `chown -R 99:100 <appdata>`. Ces commandes peuvent recréer le fichier en `root:root`, et l'émulateur ne peut alors plus enregistrer sa configuration — silencieusement, à part une ligne d'erreur dans son log.

## 6.1 Performances

Réglage recommandé (section `[Core]`) :

```
CPUThread = True
```

C'est le mode « Dual Core » de Dolphin ; selon les versions il peut être désactivé par défaut, et son absence coûte très cher en performances.

## 6.2 Jeux en français

La langue de la **console émulée** (celle que les jeux GameCube utilisent) se règle dans `[Core]` :

```
docker stop dolphin
F=<appdata>/.config/dolphin-emu/Dolphin.ini

sed -i '/^SelectedLanguage *=/d' "$F"
sed -i '/^\[Core\]/a SelectedLanguage = 2' "$F"

chown -R 99:100 <appdata>
```

```
docker start dolphin
grep -n "^SelectedLanguage" "$F"
```

Valeurs : `0` anglais, `1` allemand, `2` **français**, `3` espagnol, `4` italien, `5` néerlandais.

> Cette clé survit aux redémarrages **et** aux recréations du conteneur : le mod ne la touche pas.

> ⚠ **N'ajoutez pas** `[Interface] LanguageCode` pour traduire l'interface de Dolphin : dans cette image, la clé provoque une boîte de dialogue d'erreur de langue à chaque démarrage. Les jeux sont de toute façon en français grâce à `SelectedLanguage`, ce qui est l'essentiel — l'interface de l'émulateur n'est presque jamais visible pour les joueurs (le broker lance directement le jeu en plein écran).

> **Jeux Wii :** la langue ne vient pas de `Dolphin.ini` mais de la NAND Wii émulée. Elle se règle dans l'interface (Options → Configuration → Wii → Langue du système) et persiste dans l'appdata.

---

# 7. ⚠ Épinglage CPU — obligatoire sur Ryzen multi-CCD

Sur les Ryzen à plusieurs CCD (3900X, 3950X, 5900X, 5950X, 7900X, 7950X…), les cœurs sont répartis en groupes ayant chacun **son propre cache L3**. Un émulateur dont les threads dialoguent en permanence perd beaucoup quand l'ordonnanceur les fait migrer d'un groupe à l'autre : les échanges passent alors par l'Infinity Fabric, bien plus lent.

**Ce n'est pas une optimisation facultative :** sans épinglage, comptez une perte de l'ordre d'un tiers des performances, suffisante pour rendre injouables les titres exigeants. Le symptôme est déroutant — **performances médiocres alors que rien ne sature** (CPU global bas, GPU à 10 %, aucun thread à 100 %) — parce que les threads n'attendent pas du calcul, mais la mémoire.

Identifiez vos groupes de cache :

```
lscpu -e=CPU,CORE,L3
```

Les CPU partageant le même numéro de L3 (ou deux L3 voisins d'un même CCD) forment un groupe. Exemple sur un 3900X : le premier CCD regroupe les CPU `0-5` et leurs hyperthreads `12-17` .

Épinglez ensuite le conteneur sur un seul CCD via `--cpuset-cpus=0-5,12-17` (Extra Parameters), ou l'onglet **CPU Pinning** du template Unraid.

> Le gain observé sur ce type de configuration va de quelques pourcents à **+50 %** . C'est aussi ce qui explique qu'une VM avec vCPU épinglés paraisse bien plus rapide que le même émulateur en conteneur non épinglé, sur la même machine.

> **Multi-conteneurs :** si Dolphin et Eden tournent simultanément, répartissez-les sur des groupes différents plutôt que de les faire cohabiter sur le même CCD.

# 8. Maintenance et limites connues

- **À chaque mise à jour du driver NVIDIA de l'hôte**, le nom des libs (`libnvidia-egl-*.so.1.X.X`) change : mettre à jour les 2 Host Path correspondants dans le template Dolphin, sinon écran noir / rendu logiciel au retour.
- **Une seule session à la fois** par conteneur : un conteneur = une instance Dolphin = un flux. Pour du multi-joueurs simultané, dupliquer le conteneur (autres ports, autre secret possible) et ajouter une entrée dans `streaming.containers`.
- **Sessions fantômes** : fermer l'onglet du navigateur ne tue pas Dolphin ; la session peut rester occupée. En attendant mieux : `docker exec dolphin pkill dolphin-emu`.
- Les paquets installés à la main dans le conteneur (`apt-get`) **ne survivent pas** à une recréation. Pour des outils de diagnostic permanents, chaîner le mod `universal-package-install` : `DOCKER_MODS=ghcr.io/loneangelfayt/dolphin-romm-integration-mod:latest|linuxserver/mods:universal-package-install` + `INSTALL_PACKAGES=vulkan-tools|mesa-utils|x11-utils`.

# 9. Dépannage rapide

| Symptôme | Cause probable | Solution |
|---|---|---|
| « ROMs not found » au scan | Nom de dossier ≠ fs_slug, ou structures mélangées | §4.3 |

| Symptôme | Cause probable | Solution |
|---|---|---|
| Pas de bouton « Play on Dolphin » | Bloc `streaming` absent/invalide, ou secret différent | §4.1–4.2 |
| Flux ne se charge pas dans RomM | Certificat auto-signé jamais accepté | §5.1 |
| Jeu très lent, `dolphin-emu` absent du test des fd | Rendu logiciel (egl-x11 manquant) | §2.1 |
| Zone de jeu **noire**, interface et son OK | Présentation Vulkan impossible ( `/dev/nvidia-modeset` manquant) | §2.2 |
| Jeu lent alors que rien ne sature (CPU bas, GPU ~10 %) | Threads migrant entre CCD | §7 |
| Écran noir après une mise à jour du driver | Version des libs egl-x11 changée | §8 |
| RomM renvoie **502** au lancement | L'émulateur crashe au démarrage : le broker abandonne après 5 essais. Voir `<appdata>/dolphin.log` | ci-dessous |
| `qt.qpa.xcb: could not connect to display :0` dans `dolphin.log` | Sockets X périmées accumulées dans `/tmp/.X11-unix` (le broker détecte un affichage mort) | **Recréer** le conteneur — un simple redémarrage conserve `/tmp` |
| Boîte d'erreur de langue au démarrage | Clé `[Interface] LanguageCode` présente | La supprimer (§6.2) |
| « Failed to initialize video backend » | Variables graphiques parasites ou backend forcé à Vulkan | Revenir à la config §3, `GFXBackend = OpenGL` |

---

Revision #3
Created 2026-07-30 12:34:05 UTC by thymon
Updated 2026-07-31 22:19:53 UTC by thymon