Mettre à jour les cartes d'un TCG

Mettre à jour les cartes d'un TCG

> *Dernière mise à jour : 2026-05-11*

 Page critique — workflow de référence quand un nouveau set sort.

TL;DR

Depuis 2026-05-11, la majorité des resyncs se font **directement depuis `/admin` → section "Bases de cartes"**. Plus besoin de SSH dans le container pour les usages courants.

TCG	Source	Mode admin	Workflow
Riftbound	API Riot live	"Resync. (live)" — inline	Un clic. Diff + upsert via `syncCollection`.
MTG	Scryfall bulk JSON	"Lancer le pipeline" — modal	Download → extract → ingest (~25-40 min)
Lorcana	LorcanaJSON.org	"Lancer le pipeline" — modal	Download → ingest (~5-10 min)
FAB	Package npm `@flesh-and-blood/cards` (bundlé)	"Lancer le pipeline" — modal	Ingest seul (~5-10 min). ⚠ `npm update` + redeploy d'abord pour avoir les nouvelles cartes (cf. § FAB)
Terraforming Mars	Local `${TM_CARDS_DATA_DIR}`	"Lancer le pipeline" — modal	Parse-html → ingest (~5 min)

TCG	Source	Mode admin	Workflow
Ark Nova	Local `${ARK_NOVA_CARDS_DATA_DIR}`	"Lancer le pipeline" — modal	Slice-sprites → extract → ingest (~5 min)

Le bouton "Lancer le pipeline" exécute les **commandes npm** correspondantes côté serveur via `child_process.spawn`. Les logs sont streamés en SSE dans le modal de progression. Un seul pipeline tourne à la fois (lock global pour ne pas saturer TEI).

Architecture

Service `src/services/cards/sync-jobs/`

Quatre fichiers + un index :

- **`pipelines.ts`** — **whitelist statique** `CARD_SYNC_PIPELINES: Record<collection, { label, steps: [{ label, npm }] }>`. C'est le **seul** endroit qui mappe une collection à des commandes npm. Sécurité : aucune commande shell n'est jamais construite à partir d'un paramètre client. La collection envoyée par le frontend sert uniquement de clé dans cette map.
- **`queue.ts`** — file d'attente en mémoire avec **lock global**. Un seul `activeJob: CardSyncJob | null`. Expose `isAnyRunning()`, `createJob(coll)`, `pushEvent(type, data)`, `markFinished()`, `getEvents(coll, fromIndex)`. Rétention 10 min après `finished=true` pour permettre la reprise UI.
- **`runner.ts`** — orchestrateur. Pour chaque étape :
  1. Spawn `npm run <step.npm>` avec `cwd=PROJECT_ROOT` (calculé via `fileURLToPath(import.meta.url)`).
  2. Stdout / stderr lus ligne par ligne via `readline` → events `log` typés `{ stepIndex, line, stream }`.
  3. Exit code ≠ 0 → throw → event `error` + upsert `card_collection_meta.last_status = 'error'`.
  4. Succès : event `step:done` avec durationMs.
  5. À la fin : query Qdrant pour `cardsCount`, upsert `card_collection_meta` (status='done', `last_synced_at`, `duration_ms`).
- **`types.ts`** — types partagés (`CardSyncJob`, `CardSyncEvent`, `CardSyncPipeline`, `CardSyncEventType`).
- **`index.ts`** — API publique : `startPipeline(collection)`, `isAnyRunning()`, `getActiveJob()`, `getEvents()`, `resolvePipeline()`, `listPipelineCollections()`.

Table SQLite `card_collection_meta`

Définie dans `src/schema.ts`, exposée par `src/repositories/card-collection-meta.repo.ts` (`get`, `listAll`, `upsert`).

Champ	Type	Rôle
`collection`	text PK	Nom de la collection Qdrant (`magic-cards`, `lorcana-cards`, ...)
`last_synced_at`	text (ISO)	Date du dernier succès. NULL = jamais synchronisé via l'admin
`last_status`	text enum	`idle` / `running` / `done` / `error`
`last_error`	text	Message d'erreur si `last_status='error'`, NULL sinon
`cards_count`	integer	Nombre de points Qdrant après le dernier succès
`duration_ms`	integer	Durée du dernier pipeline en ms

Migration : `migrations/0011_tiny_nemesis.sql`. Appliquée automatiquement au boot via `runMigrations()` (`src/db.ts`).

Une ligne par collection (partagée entre le jeu de base et ses extensions). Alimentée par le `runner.ts` à chaque exécution.

Routes admin

Toutes sous `/api/admin/cards/*`, protégées par `requireAuth` + `requireAdmin`.

Route	Méthode	Rôle
`/admin/cards`	GET	Liste enrichie (dédoublonnée par collection) avec `chunksCount`, `supportsLiveResync`, `hasPipeline`, `pipelineSteps`, `lastSyncedAt`, `lastStatus`, `lastError`, `durationMs`
`/admin/cards/:collection/resync`	POST → SSE	Mode "live" (Riftbound) : `syncCollection()` qui diffe contre Qdrant. 409 pour les sources sans `supportsLiveResync`
`/admin/cards/:collection/pipeline/start`	POST	Lance le pipeline npm en arrière-plan. 202 si OK, 404 si pas de pipeline, 409 si un autre tourne
`/admin/cards/:collection/pipeline/stream`	GET → SSE	Replay historique + events temps réel (`pipeline:start`, `step:start`, `log`, `step:done`, `pipeline:done`, `error`, `heartbeat`)

Route	Méthode	Rôle
`/admin/cards/pipeline/active`	GET	État du lock global. Renvoie `{ collection, startedAt, finished } | null`. Utilisé par l'UI pour rouvrir le modal au refresh

Frontend

- `frontend/src/components/admin/AdminCardDecksSection.vue` — section "Bases de cartes" enrichie (badges fraîcheur, 3 modes de bouton selon `supportsLiveResync` / `hasPipeline`).
- `frontend/src/components/admin/CardSyncPipelineModal.vue` — modal de progression. Ouvre un EventSource sur `/admin/cards/:collection/pipeline/stream`, accumule les events, rend les étapes + console + chrono. Garde les 500 dernières lignes de logs côté client (les scripts MTG produisent ~30 000 lignes).
- `frontend/src/views/AdminView.vue` — orchestration : appelle `pipelineActive()` au mount pour la reprise auto, gère l'ouverture/fermeture du modal, émet `start-pipeline` / `resync` vers la section.

Workflow par TCG (en ligne de commande, si besoin de debug)

Tous les pipelines admin se basent sur ces commandes — elles restent disponibles en CLI si tu veux faire un dry-run, un debug, ou si l'admin est cassé.

FAB ⚠ cas spécial (data bundlée)

La data FAB est dans le package npm `@flesh-and-blood/cards` embarqué dans l'image Docker. **Cliquer "Lancer le pipeline" depuis `/admin` ne ramène PAS de nouvelles cartes** tant que l'image n'est pas rebuilt avec un package à jour. Pour vraiment ajouter de nouvelles cartes :

```
# 1. Mettre à jour le package localement (machine dev)
npm update @flesh-and-blood/cards @flesh-and-blood/types

# 2. Commit + push
git add package.json package-lock.json
git commit -m "chore(deps): update FAB cards to <version>"
git push
```

```
# 3. Attendre que la CI Gitea build/push l'image
# (jobs `test` + `build` dans .gitea/workflows/build.yml)

# 4. Sur Unraid : pull + restart le container
docker compose -f /mnt/user/appdata/boardgame-referee/docker-compose.yml pull app
docker compose -f /mnt/user/appdata/boardgame-referee/docker-compose.yml up -d --force-recreate app

# 5. /admin → "Lancer le pipeline" sur Flesh and Blood
```

MTG

Pipeline admin = `cards:mtg:download` → `cards:mtg:extract` → `cards:mtg:ingest` enchaînés. Si tu veux les lancer manuellement :

```
docker exec boardgame-referee npm run cards:mtg:download
docker exec boardgame-referee npm run cards:mtg:extract
docker exec boardgame-referee npm run cards:mtg:ingest

# Traduction des cartes non encore traduites (Haiku batch, hors pipeline admin)
docker exec boardgame-referee npm run cards:mtg:translate-missing

# Si Standard a tourné (rotation), refixer les légalités
docker exec boardgame-referee npm run meta:mtg:fix-legalities
```

Note : `cards:mtg:translate-missing` peut prendre 2-3h et n'est **pas** dans le pipeline admin (volontairement — c'est une étape lente et batch-bornée). À lancer en CLI quand nécessaire.

Lorcana

Pipeline admin = `cards:lorcana:download` → `cards:lorcana:ingest`. Manuellement :

```
docker exec boardgame-referee npm run cards:lorcana:download
docker exec boardgame-referee npm run cards:lorcana:ingest

# Symboles inline (1 seule fois, ne change pas entre sets)
docker exec boardgame-referee npm run cards:lorcana:ingest-symbols
```

⚠ **Pas relancer la sync méta DotGG tant qu'elle ne reprend pas** (figée 21 nov 2025).

Riftbound

Mode "live" depuis l'admin = un clic. En CLI pour debug :

```
docker exec boardgame-referee npm run cards:riftbound:fetch
docker exec boardgame-referee npm run cards:riftbound:ingest
```

Pas de rebuild image nécessaire (sauf si tu changes le code de normalisation).

Terraforming Mars

Pipeline admin = `cards:tm:parse` → `cards:tm:ingest`. Manuellement :

```
docker exec boardgame-referee npm run cards:tm:parse
docker exec boardgame-referee npm run cards:tm:ingest
```

Ark Nova

Pipeline admin = `cards:ark-nova:slice-sprites` → `cards:ark-nova:extract` → `cards:ark-nova:ingest`. Manuellement :

```
docker exec boardgame-referee npm run cards:ark-nova:slice-sprites
docker exec boardgame-referee npm run cards:ark-nova:extract
docker exec boardgame-referee npm run cards:ark-nova:ingest
```

Diff incrémental sans réembed (`cards:sync`)

Pour les mises à jour mineures (correction d'effet erraté, fix d'une traduction) sans repasser par le pipeline complet :

```
docker exec -e COLLECTION=flesh-and-blood-cards boardgame-referee npm run cards:sync
```

`scripts/cards/sync.ts` appelle `syncCollection()` qui diffe par `card_id + hash(card_text)` et n'embed que ce qui a changé. Idempotent.

C'est aussi ce que fait le mode "Resync. (live)" Riftbound côté admin.

Sécurité — pourquoi la whitelist statique

Le service `sync-jobs` utilise `child_process.spawn` pour exécuter des commandes npm. **Aucune partie de la commande n'est construite à partir d'input client** :

- `scriptName` vient uniquement de `CARD_SYNC_PIPELINES` (whitelist en dur dans le code)
- La `collection` envoyée par le client sert juste de clé dans cette map (un nom non whitelisté → 404)
- Pas de `shell: true`, pas d'argument supplémentaire, pas de `env: { ...userInput }`

Pour ajouter une nouvelle pipeline :

1. Ajouter les scripts npm dans `package.json`
2. Ajouter une entrée dans `CARD_SYNC_PIPELINES` (`src/services/cards/sync-jobs/pipelines.ts`)
3. Build + redeploy (aucun changement frontend nécessaire, la liste est servie dynamiquement par `GET /admin/cards`)

Vérifier que ça a marché

1. **Badge fraîcheur vert** dans `/admin` (≤ 7 jours) avec compteur de cartes mis à jour.
2. **Autocomplete** : sur `/play` du jeu concerné, taper `@<carte récente>` → la carte ressort.
3. **Question synergy** : "*Quelles sont les nouvelles cartes du set X ?*" → l'oracle doit pouvoir les citer.
4. **Si 0 résultat** : vérifier que `games.has_card_database` (en BDD SQLite) pointe sur la collection Qdrant exacte. Reconnect via `npm run cards:<tcg>:link-game` si le lien manque.

Restart cache mémoire après gros update

`services/cards-cache.ts` charge les collections en mémoire au boot. Après un gros update (>1000 cartes), restart le container pour s'assurer que le cache est warm avec la nouvelle data :

```
docker compose -f /mnt/user/appdata/boardgame-referee/docker-compose.yml restart app
```

⚡ **Note** : `syncCollection()` (mode "live" Riftbound) et le runner pipeline n'invalident **pas automatiquement** ce cache — le hot reload des cartes en mémoire est un TODO connu. En attendant, restart après gros update.

Debug

Symptôme	Piste
Bouton désactivé sans raison apparente	Un autre pipeline tourne — `GET /admin/cards/pipeline/active` ou regarder le log container
Modal ne s'ouvre pas au refresh	`pipelineActive()` n'a pas trouvé le job → expiré (>10 min après fin) ou backend redémarré
Logs vides dans la console	EventSource déconnecté (vérifie network tab, status code) ou heartbeat manquant côté serveur
`last_status=error` permanent	Lire `last_error` (truncated dans l'UI, complet en BDD) ; inspecter les logs container pour le traceback complet
Pipeline qui finit en 0s avec exit code 0	Script npm introuvable — vérifier que la commande dans `CARD_SYNC_PIPELINES` existe bien dans `package.json`

Revision #2
Created 2026-05-10 15:20:25 UTC by thymon
Updated 2026-05-11 15:48:40 UTC by thymon