BY+ · CONSOLE NETBYUS · APPLICATIONS EXTERNES

Connecter influencer-api à la console — document de mise en place

Version du 22 octobre 2026. Comment la plateforme des ambassadeurs et des partenaires (`influencer-api`) quitte l'ancienne core-api pour parler à la nouvelle (`NewByPlus/core-api`, la console) : identité, licences, catalogue, support, webhooks, administration et reprise. Public : l'équipe technique NetByUs.

En bref

- **Influencer devient une application de la console** (domaine Applications), comme Shop et ECU. Son serveur s'identifie par des requêtes signées **Ed25519** avec la clé de son instance (en-têtes `X-App-*`). Ces requêtes remplacent les clés `X-Public-Key` / `X-Secret-Key` et `X-Api-Key` / `X-Api-Secret` de l'ancienne core-api.
- **Ce qui existe déjà** dans la console : les applications, leurs instances, leurs clés et webhooks ; les licences avec titulaire et émetteur ; le support des applications (widget, tickets du serveur, délégués).
- **Ce qu'il faut ajouter** dans la console : une API signée pour que l'application lise ses offres et émette, suspende ou révoque les licences de ses distributeurs, et la validation d'une clé de licence. C'est la reprise des distributeurs prévue par la décision D7 (lot 9).
- **Ce qu'il faut changer dans influencer-api** : un client HTTP signé, à la place de `LicenceApiService` et `HelpdeskService`. Il faut aussi abandonner `contact_id` (le titulaire va dans la licence), relier les plans aux offres de la console et accepter le format de signature des webhooks de la console.
- **On ne change pas** les comptes, portefeuilles, commissions, factures et KYB des ambassadeurs et partenaires : ils restent dans influencer-api, qui garde sa propre base et son authentification (Passport).

Sommaire

1. Ce qu'on a analysé
2. Ce qu'influencer-api attend aujourd'hui de la core-api
3. Ce que la console sait déjà faire
4. L'architecture cible
5. Identité : l'application, son instance et ses clés
6. Licences des distributeurs
7. Catalogue : relier les plans aux offres
8. Support : widget, tickets transférés, délégués
9. Webhooks
10. Administration depuis la console
11. Reprise des données existantes
12. Configuration, sécurité et tests
13. Plan de mise en œuvre
14. Questions ouvertes

1. Ce qu'on a analysé

Projet	Ce qu'on en retient
`influencer-api`	Laravel 12, PostgreSQL, Passport (OAuth2), Laratrust. Modules `app/v1` : User, Ambassador, Partner, Catalogue, Admin, Helpdesk, Notification, PushNotification. Documentation : `docs/TECHNICAL_DOCUMENTATION.md`, `API_DOCUMENTATION.md`.
Ancienne `core-api`	Elle vend aujourd'hui les licences de Shop, ECU et Influencer, tient les contacts, le catalogue (applications, modules, packs, offres) et l'ancien helpdesk. influencer-api l'appelle pour ces quatre sujets.
`NewByPlus/core-api`	La console NetByUs (`packages/byplus-console`) et le socle des clients (`core/`). Domaines utiles ici : Applications, Licensing, Catalog, Support. Décision D7 : la vente des applications externes, leurs licences et les distributeurs y sont repris.

2. Ce qu'influencer-api attend aujourd'hui de la core-api

Deux sens de communication, avec trois mécanismes d'authentification différents :

Sens	Authentification	Usage
influencer → core-api	`X-Public-Key` / `X-Secret-Key` (`BY_PLUS_PUBLIC_KEY`, `BY_PLUS_SECRET_KEY`)	`App\Services\LicenceApiService` : licences, contacts, catalogue, transfert de tickets.
influencer → core-api	`X-Api-Key` / `X-Api-Secret` / `X-External-App`	`Influencer\v1\Helpdesk\Services\HelpdeskService` : tickets des ambassadeurs et partenaires.
core-api → influencer	`X-Service-Token` (`CORE_API_SERVICE_TOKEN`)	Les routes `/api/v1/service/*` : le back-office de l'ancienne console administre influencer (ambassadeurs, partenaires, dépôts, commissions, KYB, catalogue, rapports).
core-api → influencer	HMAC `X-Webhook-Signature` (`CORE_API_WEBHOOK_SECRET`)	`/api/v1/webhook/helpdesk/ticket/created` et `…/replied` : notifier l'influenceur.

Les appels sortants, un par un

Appel (ancienne core-api)	Rôle dans influencer
`POST /distributor/licenses`	Générer (ou renouveler) la licence d'un client d'ambassadeur ou de partenaire : `offer_id`, `contact_id`, `name`, `duration`, `duration_type`, `starts_at`. Retour : `license_key`, `id`.
`POST /licenses/{id}/suspend|activate|revoke`	Suspendre, réactiver, révoquer.
`POST /licenses/send/{id}/by/mail`	Envoyer la licence au client final.
`POST /public-licences/licenses/validate`	Vérifier une clé.
`POST /store/contact`, `GET /contact/by-mail`	Créer ou retrouver le contact core-api de l'ambassadeur (`ambassadors.contact_id`).
`GET /applications/collection`, `…/modules`, `…/packs`, `/license-types`, `/get/offers/license/collection`	Lire le catalogue pour relier un plan influencer à une offre core-api (`plans.core_api_offer_id`, `core_api_frequency`, `core_api_server`).

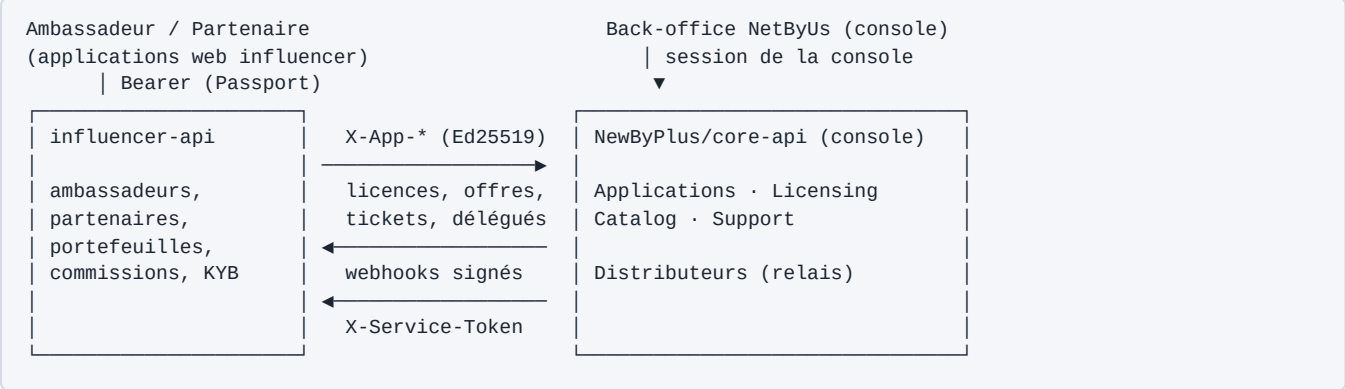
Appel (ancienne core-api)	Rôle dans influencer
`POST /distributor/influencers/assign` (et retrait)	Déclarer un influenceur distributeur d'une application.
`POST /helpdesk/receive-transferred-ticket`, `POST /helpdesk/sync-response`	Transférer un ticket au support NetByUs, synchroniser une réponse.
`GET get/application/by/key`	Retrouver une application par sa clé publique.

3. Ce que la console sait déjà faire

Domaine	Ce qui sert à influencer
Applications	Une application (`applications` : `key`, `kind` = `product`, `scopes`) a une instance par client (`app_instances`), des clés Ed25519 (`POST /api/v1/applications/instances/{id}/keys` : la clé privée n'est rendue qu'une fois) et des webhooks signés (`…/webhooks`, test compris). `ApplicationInstances::signedBy($request)` vérifie une requête signée et rend l'instance.
Signature T27	`Core\Integrations\Public\AppSignature` : en-têtes `X-App-Key`, `X-App-Timestamp`, `X-App-Nonce`, `X-App-Signature`. Message signé (`LinkSignature::canonical`) : méthode, chemin, heure, nonce, SHA-256 du corps, une ligne chacun.
Licensing	Une licence (`lic_licenses`) porte `application_key`, son offre, un titulaire `contact` (JSON, sans fiche à créer ailleurs), `frequency`, `duration`, ses dates, son statut, `metadata` et son émetteur (`issued_by_type`, `issued_by_id`). Les actions existent : émettre, suspendre, réactiver, révoquer, renouveler (routes de la console, pour ses agents).
Catalog	Les offres (`Offer`, `OfferPrice`, listes de prix) de chaque application.
Support	Le widget (package `supportwidget`, mode `app`, identité signée par le serveur de l'application), les routes du serveur `/api/v1/store/support/app/*` (ouvrir un ticket, tickets et réponses des délégués) et l'auteur `INFLUENCER`.
Webhooks	`WebhookEventEnum` : `helpdesk_ticket_created`, `ticket_replied`, `ticket_status_changed`, `licence_updated`, `ping` … Envoi signé `X-Webhook-Signature: sha256=…` (HMAC-SHA256 du corps), rejoué en cas d'échec.

> **Ce qui manque**
>
> La console ne propose pas encore d'API signée pour qu'une *application* émette des licences au nom de ses distributeurs. Ses routes de licences sont celles des agents NetByUs. C'est le cœur du travail côté console (section 6). Le reste existe et se branche.

4. L'architecture cible



Chacun garde son métier : influencer-api tient les distributeurs, la console tient le catalogue, les licences et le support. Les deux ne partagent aucune base. Toute écriture qui touche une licence passe par la console, seule à signer les licences.

5. Identité : l'application, son instance et ses clés

1. Dans la console, **Applications** : créer l'application `influencer` (`kind = product`, éditeur NetByUs), si la reprise (`Applications\Legacy\ApplicationsImporter`) ne l'a pas déjà fait.
2. Créer son **instance** pour le client `netbyus`. Influencer est exploité par NetByUs : c'est son instance qui vend aux distributeurs.
3. Créer une **clé** (« Serveur influencer-api ») : la console rend la clé privée (base64) une seule fois. On la range dans le `.env` d'influencer-api avec l'identifiant de l'instance, jamais dans le dépôt.
4. Déclarer le **webhook** de l'instance : `https://{influencer}/api/v1/webhooks/byplus`, avec les événements utiles (section 9).

Signer une requête depuis influencer-api

Un client unique remplace `LicenceApiService::http()` et `HelpdeskService`. Il signe chaque requête avec la clé privée (extension `sodium` de PHP, déjà présente avec Laravel) :

```
// app/Services/ByPlusConsole.php (influencer-api)
public function request(string $method, string $path, array $body = []): Response
{
    $json = $body === [] ? '' : json_encode($body, JSON_UNESCAPED_SLASHES |
JSON_UNESCAPED_UNICODE);
    $timestamp = time();
    $nonce = bin2hex(random_bytes(16));
    $message = implode("\n", [strtoupper($method), '/'.ltrim($path, '/'), $timestamp, $nonce,
hash('sha256', $json)]);
    $signature = base64_encode(sodium_crypto_sign_detached($message,
base64_decode(config('services.byplus.private_key'))));

    return Http::baseUrl(config('services.byplus.url'))->timeout(15)
        ->withHeaders([
            'X-App-Key' => config('services.byplus.instance_id'),
            'X-App-Timestamp' => (string) $timestamp,
            'X-App-Nonce' => $nonce,
            'X-App-Signature' => $signature,
            'Accept' => 'application/json',
        ])
        ->withBody($json, 'application/json')
        ->send($method, $path);
}
```

À respecter

- Le corps signé doit être **exactement** celui envoyé : on encode une fois et on l'envoie tel quel (`withBody`).
- Le chemin signé est le chemin complet, sans le domaine : `/api/v1/store/…`.
- `X-App-Key` porte l'**identifiant de l'instance** (un UUID), pas celui d'une clé : la console essaie chacune des clés valides de l'instance (`InstanceBook::signedBy()`). Deux clés peuvent donc coexister le temps d'une rotation.
- La signature et la clé privée sont en base64. Les horloges doivent être à l'heure (NTP) : la console refuse une requête à plus de 5 minutes de son heure, ou un nonce déjà vu.
- On compare une signature avec celle de `php artisan byplus:support:sign-request`, qui signe une requête d'essai.

6. Licences des distributeurs

Côté console à créer

Un groupe de routes signées dans Licensing, sur le modèle de `support.app` (middleware qui pose l'instance comme acteur : `Actor::apiKey($instanceId)`). L'instance n'agit que sur les licences de **son** application, et seulement celles qu'elle a émises.

Route (préfixe `/api/v1/store/licensing/app`)	Rôle
`GET /offers`	Les offres vendables de l'application de l'instance, avec leurs prix (`offer_price_id`), fréquences et durées permises.
`POST /licenses`	Émettre : `offer_price_id`, `duration`, `starts_at`, `name`, `contact` {name, email, phone}, `distributor` {type: ambassador|partner, id, name}, `reference` (l'identifiant de la licence côté influencer, pour l'idempotence). Retour : `id`, `license_key`, dates, statut.

Route (préfixe `/api/v1/store/licensing/app`)	Rôle
`GET /licenses/{id}`	L'état d'une licence émise par l'instance.
`POST /licenses/{id}/suspend`, `/unsuspend`, `/revoke`, `/renew`	Les actions existantes, ouvertes à l'instance émettrice.
`POST /licenses/{id}/send`	Envoyer la licence par e-mail au titulaire (gabarit de la console, ou sujet et texte fournis).
`POST /licenses/validate`	Vérifier une clé (statut, application, échéance), sans rien révéler d'autre.

À l'émission, la console range le distributeur dans `metadata.distributor`, l'émetteur dans `issued_by_type = api_key` et `issued_by_id = {instance}`, et la référence dans `metadata.reference`. Une même référence rend la même licence : un nouvel essai après une coupure ne crée pas de doublon. Le journal de la console (`ConsoleJournal`) trace chaque action.

Côté influencer-api à changer

Aujourd'hui	Demain
`LicenceApiService::create()` → `/distributor/licenses`	`POST /licenses` signé, avec `offer_price_id` (au lieu de `offer_id`), le `contact` du client final et le `distributor`. On garde `licenses.external_license_id` = l'identifiant de la console.
`suspend`, `unsuspend`, `revoke`, `sendByMail`, `validate`	Les routes équivalentes ci-dessus.
`storeContact`, `findContactByEmail`, `ambassadors.contact_id`	Supprimés : la console n'a plus besoin d'une fiche contact. Le titulaire est dans la licence ; le distributeur, dans `metadata.distributor`. La vérification « ambassadeur non relié à la core-api » disparaît.
`assignInfluencerToApplication` / retrait	Supprimés : c'est l'instance qui émet ; les droits d'un distributeur sur un plan restent dans influencer (`ambassador_products`, `partner_products`).

La logique de solde ne change pas. On vérifie le solde, puis on appelle la console. On ne débite le portefeuille que si la licence est émise, dans la même transaction que la licence locale : c'est déjà le cas dans `LicenseController`.

7. Catalogue : relier les plans aux offres

- Les plans d'influencer (`plans`, `plan_prices`) restent les prix de vente aux distributeurs (prix public, prix de cession, par pays).
- Le lien vers la console change : `core_api_offer_id` devient `byplus_offer_price_id` (le prix d'offre à émettre). `core_api_frequency` devient la fréquence de cette offre, et `core_api_server` n'a plus d'usage.
- `CoreApiCatalogueController` lit `GET /offers` (section 6) au lieu des cinq routes de l'ancien catalogue. L'écran d'administration qui associe un plan à une offre garde son principe.
- `applications.core_api_id` devient la clé de l'application dans la console (`influencer`, `shop`, `ecu`…).

8. Support : widget, tickets transférés, délégués

Le support des applications est déjà prêt dans la console (document « Le support et ses widgets »). Influencer s'y branche en trois points.

Besoin	Avec la console
Un ambassadeur ou un partenaire demande de l'aide	Le widget `@netbyus/support-widget` en mode `app` dans leurs applications web. Le serveur influencer signe l'identité de l'utilisateur (`sub` , `name` , `email` , `iat` , `exp` sur 5 minutes, préfixe `byplus-support-widget`) avec la clé de l'instance. Le widget l'échange contre une session (`POST /api/v1/store/support/session`). Le temps réel passe par Reverb. Cela remplace les écrans de tickets faits à la main et `HelpdeskController` .
Un traitement ouvre un ticket (l'ancien transfert)	`POST /api/v1/store/support/app/tickets` signé (remplace `/helpdesk/receive-transferred-ticket`).
Un distributeur répond à un ticket qui lui est confié	Les routes des délégués : `GET …/app/delegates/{délégué}/tickets` , `POST …/tickets/{ticket}/responses` , `PATCH …/tickets/{ticket}/status` (remplace `/helpdesk/sync-response` , `indexInfluencer` , `updateTicketStatus`). La réponse porte l'auteur `INFLUENCER` . Il faut activer les délégués pour l'application (`helpdesk_application_settings.broadcast_to_influencer`).

Le module Helpdesk d'influencer-api se réduit alors à : signer l'identité du widget, proxifier les routes des délégués pour ses utilisateurs, recevoir les webhooks du support.

9. Webhooks

Événement de la console	Ce que fait influencer
`helpdesk_ticket_created` , `ticket_replied` , `ticket_status_changed`	Notifier l'ambassadeur ou le partenaire (notification, push, e-mail), comme les routes `webhook/helpdesk/ticket/*` actuelles.
`licence_updated`	Mettre à jour la licence locale (statut, échéance) quand un agent NetByUs la change dans la console.
`ping`	Répondre 200 : c'est le bouton « Tester » de la console.

> **Le format de signature diffère**
>
> La console envoie `X-Webhook-Signature: sha256=<hex>` . `App\Services\WebhookService` compare aujourd'hui l'empreinte seule. Il faut retirer le préfixe `sha256=` avant `hash_equals` , avec le secret du webhook de l'instance (`BYPLUS_WEBHOOK_SECRET`). Un seul point d'entrée, `POST /api/v1/webhooks/byplus` , aiguille ensuite selon l'événement.

10. Administration depuis la console

Aujourd'hui, le back-office de l'ancienne console administre influencer par `/api/v1/service/*` (`X-Service-Token`). Deux options :

Option	Principe	Avis
A. Domaine « Distributeurs » dans la console	Un domaine `Platform\Distributors` relaie les routes `/service/*` côté serveur, avec `X-Service-Token` gardé dans la configuration de la console. Il ajoute ses droits (`distributors.view` , `distributors.manage` …) et le journal de la console. Le front ajoute un menu Distributeurs à la console : ambassadeurs, partenaires, dépôts, commissions, KYB, rapports.	**Recommandé** : un seul back-office, les droits et la traçabilité de la console, le jeton jamais dans un navigateur.
B. Back-office propre à influencer	Les admins utilisent l'application web d'influencer (rôle `admin` , Passport).	Possible en transition ; deux comptes et deux journaux.

Avec l'option A, influencer-api ne change rien à ses routes `/service/*`. On génère seulement un nouveau jeton (`php artisan service:token:generate`) et on le place dans le `.env` de la console (`INFLUENCER_SERVICE_TOKEN`). Une restriction par adresse IP de la console reste conseillée.

11. Reprise des données existantes

1. **Licences** : la reprise de la console (`Licensing\Legacy\LicensesImporter`) recopie les licences de l'ancienne core-api. Elle garde l'ancien identifiant dans une table de correspondance (ancien → nouveau). Les licences émises par des distributeurs y reçoivent `metadata.distributor` d'après la table de l'ancienne core-api (`application_influencers` , `contact_id`).
2. **Influencer** : une commande `byplus:remap` lit la correspondance et met à jour `licenses.external_license_id` ainsi que les plans (`core_api_offer_id` → `byplus_offer_price_id`). Elle est idempotente et produit un rapport des lignes non trouvées.
3. **Bascule** : on fige les émissions côté influencer (mode maintenance du module Ambassador/Partner), on lance les deux reprises, on bascule la configuration (`.env`), puis on rouvre. Le retour arrière consiste à remettre l'ancien `.env` : l'ancienne core-api reste en lecture tant que la reprise n'est pas validée.

12. Configuration, sécurité et tests

Variables

Où	Variables
influencer-api **ajout**	`BYPLUS_URL` (ex. `https://byplus.io`), `BYPLUS_APP_INSTANCE_ID` , `BYPLUS_APP_PRIVATE_KEY` (base64), `BYPLUS_WEBHOOK_SECRET` , `BYPLUS_WIDGET_KEY` (clé publique du widget de l'instance).
influencer-api **retrait**	`BY_PLUS_API_URL` , `BY_PLUS_PUBLIC_KEY` , `BY_PLUS_SECRET_KEY` , `BY_PLUS_COMPANY_ID` , `CORE_API_WEBHOOK_SECRET` (remplacé). `CORE_API_SERVICE_TOKEN` reste si l'option A est retenue.
console (NewByPlus/core-api) **ajout**	`INFLUENCER_API_URL` , `INFLUENCER_SERVICE_TOKEN` (option A seulement).

Sécurité

- La clé privée d'instance ne quitte jamais le serveur d'influencer. En cas de fuite : révoquer la clé dans la console et en créer une autre, sans interruption si deux clés coexistent le temps de la rotation.
- Une instance ne voit et n'agit que sur son application et les licences qu'elle a émises. Un délégué ne voit que les tickets qui lui sont confiés.

- Les règles du document « Règles de sécurité » s'appliquent : rien de secret dans les journaux, limite de requêtes sur les routes signées, HTTPS partout.

Tests

- **Console** : tests de fonctionnalités des routes signées (signature valide, refusée, rejouée ; licence d'une autre application refusée ; idempotence par référence ; webhook `licence_updated` émis). Vérifier aussi l'analyse statique, deptrac (le domaine Licensing ne lit Applications que par son contrat public), bref `make check`.
- **influencer-api** : le client signé, testé contre un faux serveur (`Http::fake`) qui vérifie la signature avec la clé publique, et l'endpoint de webhook avec le préfixe `sha256=`.
- **De bout en bout**, en local : générer une licence depuis un compte ambassadeur, la voir dans la console, la suspendre depuis la console, voir le statut revenir dans influencer par webhook.

13. Plan de mise en œuvre

Lot	Contenu	Où	Dépend de
I1	Application `influencer`, instance `netbyus`, clé, webhook	Console (paramétrage, sans code)	—
I2	Client signé `ByPlusConsole` et son test	influencer-api	I1
I3	API signée des licences : offres, émission, actions, envoi, validation	Console, domaine Licensing	—
I4	Licences et catalogue branchés sur I3 ; fin de `contact_id`	influencer-api	I2, I3
I5	Webhooks : point d'entrée unique, préfixe `sha256=`	influencer-api	I1
I6	Support : widget en mode app, tickets du serveur, délégués	influencer-api (+ fronts)	I2
I7	Domaine Distributeurs (option A) et ses écrans	Console + core-app	—
I8	Reprise et bascule	Les deux	I3 à I6, lot 9 de la console

14. Questions ouvertes

- **Plusieurs applications vendues** : un ambassadeur vend Shop, ECU et d'autres. Faut-il une instance « influencer » qui émet pour plusieurs applications (droit à déclarer dans ses `scopes`), ou une instance par application vendue ? La proposition ci-dessus suppose la première, plus simple pour influencer. Il faut alors préciser dans la console la liste des applications qu'une instance distributrice peut vendre.
- **Prix** : la console calcule-t-elle un prix pour une licence émise par un distributeur (facturation NetByUs ↔ distributeur), ou le prix reste-t-il dans influencer (prix de cession, portefeuille) ? Proposition : la console enregistre l'offre sans facturer, influencer reste maître du prix.
- **Identité unique** : à terme, un compte NetByUs unique (SSO) pour les distributeurs et la console ? Pas nécessaire pour cette connexion.
- **Clients des partenaires** : un client de partenaire qui devient client BY+ (un espace) doit-il être rattaché au partenaire dans la console (apport d'affaires, commissions) ? À décider avec le commercial.

Sources : `influencer-api` (`app/Services/LicenceApiService.php`, `app/v1/Helpdesk`, `app/v1/Admin/routes/service.php`, `app/Services/WebhookService.php`, `docs/TECHNICAL_DOCUMENTATION.md`) ; `NewByPlus/core-api` (`packages/byplus-console/src/Applications`, `Licensing`, `Support`, `core/Integrations`, `core/Link`) ; documents « Console, mise en place » (décision D7), « Le support et ses widgets ».