BY+ · API · PRODUCTION

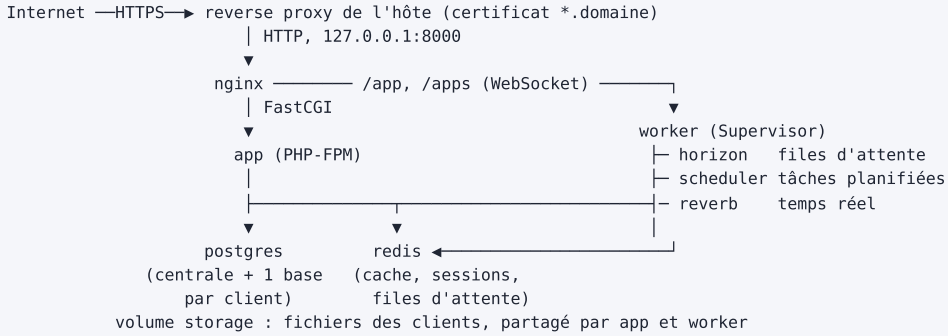
Installer l'API BY+ en production avec Docker

Le serveur, la première installation, les files d'attente (Horizon sous Supervisor), les mises à jour, la surveillance et les sauvegardes.

Pour la personne qui installe et met à jour l'API sur un serveur.
Fichiers du dépôt : `compose.prod.yaml`, `docker/prod/`, `.env.production.example`.
Octobre 2026 · le développement local reste décrit dans DOCKER.md.

1 Ce qui tourne sur le serveur

Une seule image construite depuis le dépôt (le code est dedans, rien n'est monté depuis le disque), utilisée par deux conteneurs, plus nginx, PostgreSQL et Redis. Seul nginx est joignable de l'extérieur, et seulement par le reverse proxy HTTPS de l'hôte.



Conteneur	Image	Rôle
app	`byplus/api` (cible app)	PHP-FPM : répond aux requêtes de l'API
worker	`byplus/api`, commande `supervisord`	Supervisor lance et relance Horizon, le planificateur (`schedule:work`) et Reverb
nginx	`byplus/api-web` (cible web)	Serveur web : PHP vers app, WebSocket vers worker. Domaine central et `*.domaine`
postgres	postgres:17-alpine	Base centrale et une base par client (créée automatiquement). Remplaçable par une base gérée
redis	redis:7-alpine, AOF, mot de passe	Cache, sessions, files d'attente

> **Pourquoi Supervisor dans un conteneur :** Horizon, le planificateur et Reverb doivent tourner sans fin et redémarrer s'ils tombent. Supervisor s'en charge dans le conteneur `worker` ; Docker, lui, relance le conteneur entier s'il s'arrête (`restart: unless-stopped`) et vérifie Horizon toutes les 30 s (`horizon:status`).

Les fichiers

Fichier	Contenu
`compose.prod.yaml`	Les cinq conteneurs, les volumes, les vérifications de santé
`docker/prod/Dockerfile`	Image en trois étapes : dépendances Composer sans les outils de développement, nginx (cible web), PHP (cible app)
`docker/prod/supervisord.conf`	Les trois programmes du worker : horizon, scheduler, reverb
`docker/prod/entrypoint.sh`	Au démarrage : prépare storage/, met configuration et routes en cache (`artisan optimize`)
`docker/prod/php.ini`	OPcache sans relecture des fichiers, erreurs dans les journaux, pas à l'écran
`docker/prod/nginx/default.conf.template`	Virtual host : domaine central et clients, WebSocket vers Reverb
`.env.production.example`	Modèle du `.env` du serveur

2 Préparer le serveur

- **Système :** Linux 64 bits, 4 Go de mémoire au moins (8 Go conseillés), 40 Go de disque. Docker Engine 24 ou plus, avec le plugin Compose.
- **DNS :** deux enregistrements vers le serveur : le domaine central (`byplus.example.com`) et le joker de ses clients (`*.byplus.example.com`). Un client `acme` est servi sur `acme.byplus.example.com`.
- **Certificat :** un certificat joker `*.byplus.example.com` (Let's Encrypt par défi DNS, ou certificat de l'entreprise), plus le domaine central.
- **Pare-feu :** seuls 80 et 443 (et SSH) sont ouverts. PostgreSQL, Redis et nginx ne sont pas exposés : nginx écoute sur `127.0.0.1:8000`.
- **Base gérée (facultatif) :** l'utilisateur doit avoir le droit `CREATEDB` : la création d'un client crée sa base.

Le reverse proxy HTTPS de l'hôte

Exemple avec nginx sur l'hôte (Caddy ou Traefik conviennent aussi). Il termine le HTTPS et renvoie tout vers le conteneur nginx, WebSocket compris :

```
server {
    listen 443 ssl http2;
    server_name byplus.example.com *.byplus.example.com;
    ssl_certificate     /etc/letsencrypt/live/byplus.example.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/byplus.example.com/privkey.pem;
    client_max_body_size 64M;

    location / {
        proxy_pass http://127.0.0.1:8000;
        proxy_http_version 1.1;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto https;
        proxy_set_header Upgrade $http_upgrade;       # temps réel (Reverb)
        proxy_set_header Connection "upgrade";
        proxy_read_timeout 120s;
    }
}
server { listen 80; server_name byplus.example.com *.byplus.example.com; return 301 https://$host$request_uri; }
```

L'API ne croit `X-Forwarded-For` que des proxys de `TRUSTED_PROXIES` (par défaut le réseau privé, où se trouvent le nginx du conteneur et le passage par l'hôte) : c'est ce qui donne aux limites de connexion et au journal d'audit l'adresse réelle de la personne. Le reverse proxy de l'hôte doit donc toujours ajouter `X-Forwarded-For`, comme ci-dessus.

> **Le nom d'hôte doit arriver intact** (`proxy_set_header Host $host`) : la tenancy reconnaît le client à son sous-domaine.

3 Première installation

1. **Récupérer le code** (branche de production) :

```
git clone git@gitlab…/core-api.git /opt/byplus/api && cd /opt/byplus/api
git checkout main
```

2. **Écrire le** `.env` à partir du modèle, et le protéger :

```
cp .env.production.example .env && chmod 600 .env
# compléter : APP_DOMAIN, DB_PASSWORD, REDIS_PASSWORD, REVERB_APP_KEY/SECRET, MAIL_*, WHATSAPP_*…
openssl rand -base64 32        # pour chaque mot de passe ou secret
```

3. **Construire les images** :

```
docker compose -f compose.prod.yaml build
```

4. **Générer la clé de l'application** une seule fois, la coller dans `APP_KEY` du `.env` :

```
docker compose -f compose.prod.yaml run --rm --no-deps app php artisan key:generate --show
```

> Cette clé chiffre les jobs en file, les sessions et certains secrets (mot de passe de l'annuaire des clients). **Ne jamais la changer** ensuite, et la garder dans le coffre de mots de passe.

5. **Démarrer la base et Redis, puis créer les tables** (base centrale, puis bases des clients) :

```
docker compose -f compose.prod.yaml up -d postgres redis
docker compose -f compose.prod.yaml run --rm app php artisan migrate --force
docker compose -f compose.prod.yaml run --rm app php artisan tenants:migrate --force
```

6. **Tout démarrer** et vérifier :

```
docker compose -f compose.prod.yaml up -d
docker compose -f compose.prod.yaml ps                       # tous « healthy »
docker compose -f compose.prod.yaml exec worker supervisorctl status
docker compose -f compose.prod.yaml exec app php artisan horizon:status
```

7. **Créer un client et son administrateur** :

```
docker compose -f compose.prod.yaml exec app php artisan tinker --execute \
 '$t = Core\Tenancy\Models\Tenant::create(["id" => "acme"]); $t->domains()->create(["domain" => "acme.byplus.example.com"]);'
docker compose -f compose.prod.yaml exec app php artisan byplus:user:create admin@acme.com --tenant=acme --admin
```

Le serveur NetByUs : la console

La console (package `netbyus/byplus-console`) n'entre que dans l'image du serveur NetByUs. Dans son `.env` : `BYPLUS_WITH_CONSOLE=1`, puis :

```
docker compose -f compose.prod.yaml build
docker compose -f compose.prod.yaml up -d
docker compose -f compose.prod.yaml exec app php artisan byplus:console:install \
  --email=awa@netbyus.com --name="Awa Houngbo" --provider=netbyus --login=awa@netbyus.com
```

La commande refuse un serveur enrôlé comme instance ; elle joue les migrations, crée la clé de signature des licences, aligne le catalogue inscrit le premier super-administrateur et crée le client `netbyus`, dont il est l'administrateur. Il se connecte à BY+ netbyus, ouvre « Console NetByUs » dans le menu et saisit le code reçu à son adresse e-mail (double authentification de la console). Le front du serveur NetByUs est construit avec `VITE_WITH_CONSOLE=1` ; les autres fronts gardent `0` : la console n'est pas dans leur build. Elle se relance sans risque après chaque mise à jour. Les autres serveurs gardent `BYPLUS_WITH_CONSOLE=0` : leur image ne contient pas le code de la console.

Un serveur SaaS ou dédié : l'enrôler

```
docker compose -f compose.prod.yaml exec app php artisan byplus:link:enroll https://byplus.io <jeton> --url=https://serveur-du-client
```

Le jeton est donné par un agent dans la console (Serveurs → Nouveau serveur), valable 24 h, une fois. Ensuite le serveur se synchronise seul toutes les 5 minutes.

Astuce : un alias évite de répéter le fichier : `alias dcp='docker compose -f compose.prod.yaml'`.

4 Les files d'attente

Tout ce qui peut attendre quelques secondes part en file : la requête répond tout de suite, un ouvrier fait le travail. Les travaux sont rangés par nature (`Core\Kernel\Queue\Queue`), et Horizon donne à chaque file ses ouvriers. **Un traitement long ne retarde jamais un code de connexion.**

File	Ce qu'on y met	Superviseur Horizon	Réglages (production)
urgent	Une personne attend devant l'écran : code de connexion (e-mail, WhatsApp), mot de passe oublié, invitation	`urgent`	3 ouvriers toujours prêts, 30 s par job, 3 essais à 5 s d'écart, alerte au-delà de 15 s d'attente
notifications	Notifications courantes : dans l'application, mail, push, WhatsApp	`courant`	De 2 à 10 ouvriers selon la charge (balance auto), 90 s par job, 3 essais à 30 s, alerte au-delà de 2 min
default	Le reste, court (moins d'une minute)		
heavy	Long ou gourmand : vérification de nuit de l'annuaire (un job par client), imports, exports, rapports, PDF en masse	`lourd`	2 ouvriers, 30 min par job, 512 Mo, 1 essai, priorité basse (nice 10). Connexion Redis à part : `redis-heavy`

> **Pourquoi une connexion à part pour heavy :** Redis rend un job à la file s'il n'est pas terminé après `retry_after` secondes. Pour les files courtes, 90 s ; pour heavy, 1 900 s, plus que les 30 min d'un job lourd. Avec une seule valeur, soit un job lourd serait relancé en double, soit un job court bloqué attendrait 30 min.

Dans le code

```
use Core\Kernel\Queue\Queue;

class ExporterLesFactures implements ShouldQueue
{
    use Queueable;

    public int $timeout = 1800;

    public function __construct(public readonly string $periode)
    {
        Queue::Heavy->assign($this);          // file heavy, connexion redis-heavy
    }
}

// Un message que quelqu'un attend : file urgent
$notifier->send($destinataire, Message::fromTemplate('auth.one_time_code', [...])->urgent(), [Channel::Mail]);
```

- Un job sans file choisie va dans **default** : il doit être court.
- Les envois de notifications sont **chiffrés** dans Redis et dans Horizon : un code ou un lien de mot de passe n'y est pas lisible.
- Chaque job porte des étiquettes (`client:acme` , `channel:mail` …) : dans Horizon, on retrouve les travaux d'un client.

Régler le nombre d'ouvriers

Dans le `.env` , sans reconstruire : `HORIZON_URGENT_PROCESSES` , `HORIZON_DEFAULT_MAX_PROCESSES` , `HORIZON_HEAVY_PROCESSES` , puis `docker compose -f compose.prod.yaml up -d worker` . Repère : chaque ouvrier occupe 60 à 150 Mo.

5 Mettre à jour

1. **Récupérer le code et construire** les nouvelles images (le service continue pendant ce temps) :

```
cd /opt/byplus/api && git pull
docker compose -f compose.prod.yaml build
```

2. **Arrêter proprement les ouvriers** : Horizon finit les jobs en cours et n'en prend plus.

```
docker compose -f compose.prod.yaml exec worker php artisan horizon:terminate
```

3. **Migrer** la base centrale et les bases des clients :

```
docker compose -f compose.prod.yaml run --rm app php artisan migrate --force
docker compose -f compose.prod.yaml run --rm app php artisan tenants:migrate --force
```

4. **Remplacer les conteneurs** (configuration et routes sont remises en cache au démarrage) :

```
docker compose -f compose.prod.yaml up -d
docker compose -f compose.prod.yaml ps
```

> **Une modification du `.env`** n'est prise en compte qu'au redémarrage des conteneurs : `docker compose -f compose.prod.yaml up -d --force-recreate app worker`. **Une mise à jour longue** (migration lourde) : `php artisan down` avant, `php artisan up` après.

Revenir en arrière : garder l'étiquette de l'image précédente (`IMAGE_TAG`), la remettre dans le `.env` et `up -d`. Une migration déjà passée ne se défait pas toute seule : la tester avant sur une copie de la base.

6 Surveiller

Horizon et Telescope

Les deux tableaux de bord ne sont servis que sur le **domaine central** (`/horizon` , `/telescope`), jamais sur le sous-domaine d'un client. Leur affichage se commande dans le `.env` , comme dans core-api :

Variable	Production	Effet
`HORIZON_VIEW`	`false`	Affiche le tableau de bord d'Horizon. Horizon tourne dans tous les cas
`HORIZON_NOTIFY_EMAIL`	adresse d'astreinte	Reçoit une alerte quand une file attend trop (15 s pour urgent, 2 min pour les autres)
`TELESCOPE_ENABLED`	`false` ou `true`	Enregistre. Hors local, seulement les erreurs, requêtes et jobs échoués, tâches planifiées ; gardé 48 h
`TELESCOPE_VIEW`	`false`	Affiche le tableau de bord de Telescope

> **Pas encore de comptes sur le domaine central :** un tableau de bord ouvert l'est à tout Internet. En production, pour consulter, ouvrir un tunnel SSH vers le serveur et passer `*_VIEW=true` seulement le temps de la consultation, ou n'autoriser que l'adresse IP de l'équipe dans le reverse proxy (`location /horizon { allow 1.2.3.4; deny all; … }`). Quand la console NetByUs aura ses comptes, l'accès lui sera rattaché.

Journaux et santé

```
docker compose -f compose.prod.yaml logs -f --tail=100 app worker nginx
docker compose -f compose.prod.yaml exec worker supervisorctl status       # horizon, scheduler, reverb : RUNNING
docker compose -f compose.prod.yaml exec app php artisan horizon:status
docker compose -f compose.prod.yaml exec app php artisan queue:failed     # jobs échoués
docker compose -f compose.prod.yaml exec app php artisan queue:retry all  # les relancer
```

Les jobs échoués restent 7 jours dans Horizon ; les terminés, 1 heure.

Le bilan de santé (`/health`)

Chaque API publie son bilan sur son domaine central (`spatie/laravel-health`), relancé chaque minute par le planificateur : failles des dépendances, disque, base et nombre de connexions, Redis, cache, Horizon, files d'attente (un travail témoin par file), planificateur, mode debug, environnement, cache de configuration et de routes (production), Reverb quand l'API diffuse, LiveKit sur le serveur qui le configure. L'historique est gardé 5 jours (purge chaque nuit).

Adresse	Pour qui	Accès
`https://{domaine central}/health`	Une personne : la page des résultats	HTTP Basic : `HEALTH_USERNAME` , `HEALTH_PASSWORD`
`https://{domaine central}/health.json`	Un outil de surveillance (Uptime Kuma, Oh Dear…) ; `?fresh` relance les contrôles	Jeton : en-tête `X-Health-Token` ou `?token=` (`HEALTH_SECRET_TOKEN`)

Sans identifiants ou sans jeton dans le `.env` , l'adresse est refusée (403). Dix essais par minute et par adresse IP. Les alertes partent par mail à `HEALTH_TO_ADDRESS` quand `HEALTH_NOTIFICATIONS_ENABLED=true` (une par heure au plus).

7 Sauvegarder

Quoi	Où	Comment
Toutes les bases (centrale + clients)	volume `postgres-data`	Chaque nuit, hors du serveur : `docker compose -f compose.prod.yaml exec -T postgres pg_dumpall -U byplus | gzip > byplus-$(date +%F).sql.gz`
Fichiers des clients	volume `storage`	`docker run --rm -v byplus-api_storage:/s -v $PWD:/b alpine tar czf /b/storage-$(date +%F).tgz -C /s .` (ou disque S3/MinIO, sauvegardé à part)
Le `.env`	serveur	Dans le coffre de mots de passe, surtout `APP_KEY`
Redis	volume `redis-data`	Pas indispensable (cache, files en cours). AOF activé : les jobs en attente survivent à un redémarrage

Tester la restauration une fois par trimestre sur un serveur de test : une sauvegarde jamais restaurée n'est pas une sauvegarde.

8 En cas de problème

Symptôme	Cause probable	Que faire
Les codes de connexion n'arrivent pas	Worker arrêté, ou fournisseur mail/WhatsApp	`supervisorctl status`, Horizon (file urgent), `logs worker`, réglages `MAIL_*`, `WHATSAPP_*`
404 sur le sous-domaine d'un client	Domaine du client absent, ou Host modifié par le proxy	Vérifier son domaine dans la table `domains` et `proxy_set_header Host $host`
502 Bad Gateway	Conteneur app arrêté ou en démarrage	`ps`, `logs app`
Le temps réel ne se connecte pas	WebSocket non transmis	En-têtes `Upgrade` / `Connection` dans le reverse proxy ; `supervisorctl status reverb`
Un changement du `.env` est ignoré	Configuration en cache	`up -d --force-recreate app worker`
Un job lourd est fait deux fois	Job plus long que 30 min	Le découper (un job par client, par lot), ou relever ensemble le timeout du superviseur lourd, `REDIS_HEAVY_QUEUE_RETRY_AFTER` et `stopwaitsecs`

Le front des clients

Le front (core-app) est une application statique : `npm ci && npm run build` avec `VITE_APP_DOMAIN=app.byplus.example.com` et `VITE_API_URL=https://{tenant}.byplus.example.com/api/v1`, puis le dossier `dist/` servi par un nginx sur `*.app.byplus.example.com`, toute adresse inconnue renvoyant vers `index.html`. Son installation détaillée fera l'objet d'un document à part.