GUIDE PRATIQUE · ÉDITION GRATUITE · AOÛT 2026

Automatiser WhatsApp sans l'API Meta

D'un VPS vide à un bot WhatsApp qui répond avec une IA, sur ton propre serveur. En une soirée, pour environ 10 $ de démarrage. Sans compte Business API, sans validation Meta, sans numéro vérifié. Et sans te faire couper.

`Client WhatsApp → gowa (VPS) → webhook FastAPI → DeepSeek → réponse différée`

`12 chapitres · 11 pannes réelles résolues · chaque commande avec sa sortie attendue`

Par **Wapay** · Ce livre est gratuit et le restera. Ce qui se paie, c'est mon assistance : voir la page « Aller plus vite ». · Questions : wa.me/2250712116258

AVANT DE COMMENCER

Comment lire ce livre

La promesse, et rien de plus : à la fin, ton WhatsApp répond automatiquement avec une IA, sur ton propre serveur, et tu sais pourquoi chaque brique est là.

Ce que ce livre ne promet pas, noir sur blanc :

- Ce n'est pas l'API officielle Meta. Aucune garantie de service.
- Ce n'est pas fait pour envoyer 1 000 messages à froid. Le chapitre 11 t'explique pourquoi c'est la façon la plus sûre de perdre ton numéro.
- Ce n'est pas un produit clé en main revendable tel quel à un client sans travail de ta part.
- Ce livre promet un résultat technique, jamais un revenu.

Les conventions, à retenir une fois pour toutes

Chaque commande est donnée **avec ce qu'elle affiche quand elle réussit**. Si tu ne sais pas à quoi ressemble le succès, tu ne sais pas si tu peux continuer. Chaque chapitre se termine par un point de contrôle vérifiable : tant qu'il ne passe pas, ne commence pas le chapitre suivant.

PRÉFIXE	OÙ TAPER LA COMMANDE
`$`	Sur ta machine (Windows, Mac ou Linux)
`#`	Sur le VPS, connecté en SSH

La confusion « je tape ça où ? » est la question de support numéro 2. Cette convention tient sur tout le livre.

> L'avertissement qui compte, avant la première commande
> gowa s'appuie sur un client WhatsApp **non officiel**. Le numéro utilisé **peut être banni** par Meta, et un numéro banni ne se récupère pas. Trois règles absolues :
>
> 1. **Jamais ton numéro personnel.** Une puce dédiée à 500 F est le premier achat de ce tutoriel.
> 2. Ce qui fait bannir, ce n'est pas la vitesse d'écriture : c'est le **premier contact non sollicité** et le taux de blocage. Un bot qui répond à des gens qui lui écrivent vit dans un régime radicalement différent d'un bot qui prospecte.
> 3. Le chapitre 11 entier t'apprend à ne pas te faire couper. Lis-le avant de mettre un vrai commerce sur le numéro.

UNE PAGE, PAS PLUS

Aller plus vite : mon assistance

Tout le code de ce livre est dedans, gratuit, sans rien de caché. Tu peux tout faire seul en suivant les chapitres. Ce qui se paie, c'est mon temps : chaque erreur que tu vas rencontrer, je l'ai déjà réparée, et une séance avec moi remplace une nuit de recherche.

Ce livre **0 F**

Le chemin complet, les templates, le code du serveur, le chapitre dépannage, le chapitre anti-ban. Tu t'arrêtes ici et tu as tout pour réussir seul.

Assistance serveur IA **10 000 F CFA**

Une séance de 45 minutes en visio avec moi, à utiliser sous 30 jours : tu partages ton écran, on installe ou on répare ensemble ton gowa et ton serveur FastAPI. Plus l'accès au groupe WhatsApp privé des assistés, où je poste les correctifs quand gowa ou DeepSeek changent quelque chose.

Assistance encaissement **20 000 F CFA**

Tout ce qui précède, avec deux séances de 45 minutes au lieu d'une, et un accompagnement dédié sur le chapitre 12 : brancher MoneyFusion, tester le webhook de paiement idempotent, et facturer ton premier client. Les deux premières offres t'aident à construire ; celle-ci t'aide à te faire payer.

Pour réserver : écris-moi sur WhatsApp au **wa.me/2250712116258** avec le mot « assistance », ou rejoins directement le groupe WhatsApp par le lien en pied de chaque page de ce livre.

À LIRE AVANT DE CLIQUER

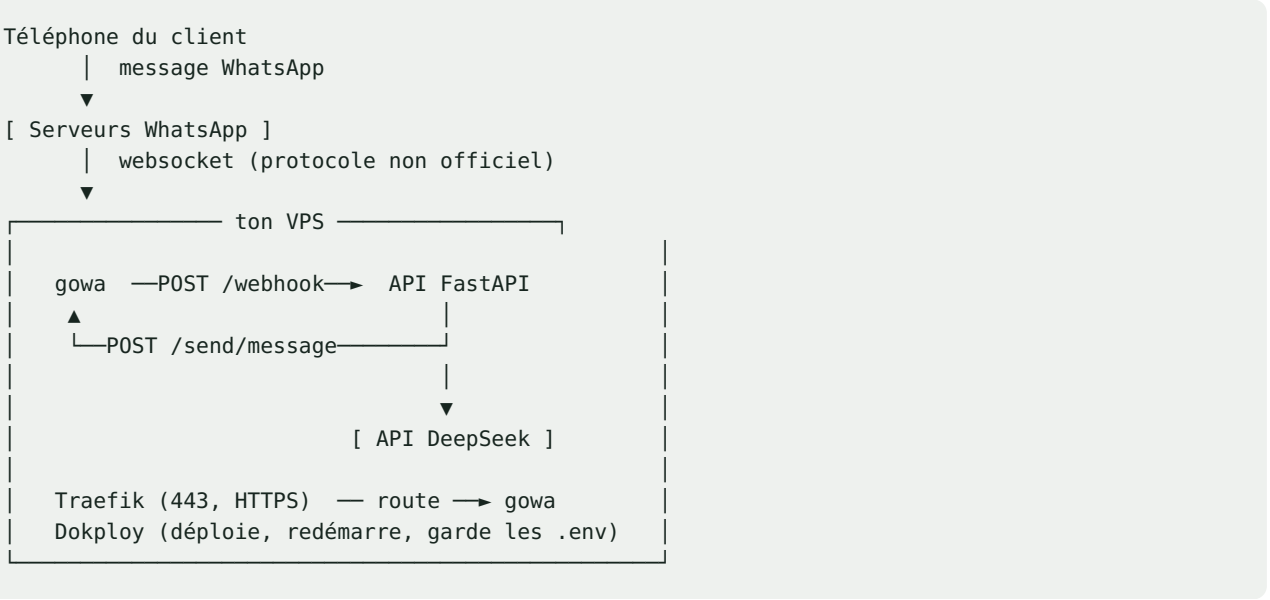
Le groupe WhatsApp est réservé à l'assistance payante. En le rejoignant, tu confirmes que tu veux être assisté et que tu es prêt à régler l'une des deux offres ci-dessus (10 000 ou 20 000 F). Si tu veux seulement suivre le livre gratuitement, c'est très bien : tout est dedans, mais ne rejoins pas le groupe.

Sommaire

CHAPITRE 0

Ce que tu vas construire

Un seul schéma, qu'on retrouvera à chaque chapitre. Si tu comprends cette page, tu comprends tout le livre.



Pourquoi chaque brique existe

C'est la moitié de la valeur de ce chapitre : savoir ce qui casse quand une brique manque.

BRIQUE	CE QU'ELLE FAIT	SANS ELLE
gowa	parle le protocole WhatsApp à ta place, expose une API HTTP simple	tu devrais implémenter le protocole Signal et le multi-device toi-même
FastAPI	ton cerveau : reçoit le webhook, décide, répond	gowa sait envoyer, il ne sait pas quoi dire
DeepSeek	génère la réponse	tu retombes sur un menu à touches « tapez 1 »
Traefik	reçoit le 443, termine le TLS, aiguille vers le bon conteneur	tu exposes des ports en clair et tu bricoles les certificats à la main
Dokploy	interface au-dessus de Docker : déploie, garde les variables, redémarre	tu fais tout en SSH, et tu perds ta config au premier `docker rm`
Let's Encrypt	certificat HTTPS gratuit, renouvelé tout seul	le webhook en HTTP simple, et un navigateur qui hurle

Le sens du trafic, que tout le monde confond

Traefik ne sert **que** pour ce qui vient d'Internet vers toi : l'interface web de gowa, ton dashboard Dokploy. La conversation entre gowa et ton API FastAPI, elle, ne sort **jamais** du VPS : elle passe par le réseau Docker interne, en HTTP simple, par nom de conteneur. C'est pour ça qu'on n'a pas besoin de

certificat entre les deux. Retiens-le dès maintenant : ça t'évitera le grand classique « pourquoi mon webhook en `https://` fait un timeout ».

> Contrôle
> Tu sais répondre à la question : à quoi sert Traefik, et pourquoi l'API n'en a pas besoin pour parler à gowa ?

CHAPITRE 1

Le VPS : commander, sécuriser, installer Docker

C'est le chapitre le plus long du livre, et c'est normal : c'est le seul où une erreur se paie en serveur miné ou en base de données volée. Tout le reste est rattrapable.

1.1 Commander

Un VPS avec **4 Go de RAM** suffit largement pour gowa, Dokploy, Traefik et ton API. Prends **Ubuntu 24.04 LTS**, pas une version intermédiaire : Docker et Dokploy sont testés sur LTS. Localisation : l'Europe est le meilleur compromis latence-prix depuis l'Afrique de l'Ouest. Tu reçois par mail une IP, un utilisateur `root` et un mot de passe.

> Attention
> Ce mot de passe est arrivé en clair dans une boîte mail. Il ne survivra pas à ce chapitre.

> Note tarifs
> Compte autour de 4 000 F par mois pour un VPS 4 Go, tarifs constatés en août 2026, avec parfois des frais de mise en service au premier paiement. Vérifie les prix du jour avant de commander.

1.2 Première connexion

Partout dans ce livre, `203.0.113.10` est une IP d'exemple : remplace-la par la tienne.

```
$ ssh root@203.0.113.10
root@vps:~#
```

Sur Windows, ouvre PowerShell : `ssh` y est disponible d'office depuis Windows 10. À la première connexion, la question sur le « fingerprint » est normale : réponds `yes`.

1.3 Mettre à jour, créer ton utilisateur

```
# apt update && apt upgrade -y
# adduser deploy
# usermod -aG sudo deploy
```

Tout le reste du livre se fait avec cet utilisateur `deploy`. Travailler en root en permanence, c'est conduire sans ceinture : ça marche jusqu'au jour où ça ne pardonne pas.

1.4 Les clés SSH, à la place du mot de passe

Sur **ta machine**, génère une paire de clés si tu n'en as pas déjà une :

```
$ ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
```

Accepte l'emplacement proposé (Entrée), mets une phrase de passe si tu veux. Puis envoie la clé publique sur le VPS :

```
$ ssh-copy-id deploy@203.0.113.10
```

Sur Windows sans `ssh-copy-id` :

```
$ type $env:USERPROFILE\.ssh\id_ed25519.pub | ssh deploy@203.0.113.10 "mkdir -p ~/.ssh && cat
>> ~/.ssh/authorized_keys"
```

Vérifie que tu entres **sans mot de passe** avant de continuer :

```
$ ssh deploy@203.0.113.10
deploy@vps:~$
```

1.5 Fermer la porte : plus de root, plus de mot de passe

> Piège
> Ne fais cette étape **qu'après** avoir vérifié que la connexion par clé fonctionne, dans un second terminal, en gardant le premier ouvert. Si tu te coupes l'accès, il te reste la console web de ton hébergeur, mais autant ne pas y aller.

```
# nano /etc/ssh/sshd_config
```

Trouve et mets ces deux lignes (enlève le `#` devant si besoin) :

```
PermitRootLogin no
PasswordAuthentication no
```

```
# systemctl restart ssh
```

Test : `ssh root@203.0.113.10` doit maintenant être **refusé**. C'est le but.

1.6 Le pare-feu, et la commande qui dit la vérité

```
# ufw allow 22/tcp
# ufw allow 80/tcp
# ufw allow 443/tcp
# ufw enable
# ufw status
Status: active
22/tcp    ALLOW    Anywhere
80/tcp    ALLOW    Anywhere
443/tcp   ALLOW    Anywhere
```

Et la commande à connaître par cœur, celle qui montre ce qui est **réellement** exposé sur Internet, quoi qu'en dise le pare-feu :

```
# ss -tulnp | grep -v '127.0.0.1\|::1'
```

> À savoir
> Docker publie ses ports **en contournant UFW**. Un `ports: ["3080"]` dans un docker-compose expose le port au monde entier même si UFW dit le contraire. C'est pour ça que ce livre écrit toujours `127.0.0.1:3080:3080`, et que `ss -tulnp` est ton juge de paix, pas `ufw status`.

1.7 Installer Docker

```
# curl -fsSL https://get.docker.com | sh
```

Tu pipes un script d'Internet dans un shell root : tu as le droit de le lire d'abord avec `curl -fsSL https://get.docker.com | less`. C'est même une bonne habitude.

1.8 Vérifier

```
# docker --version
Docker version 27.x.x, build ...
# docker compose version
Docker Compose version v2.x.x
# systemctl is-active docker
active
```

`systemctl status docker` ouvre un pager : on en sort avec `q`. Ce détail évite sincèrement quelques messages de panique.

Le test qui prouve toute la chaîne (téléchargement d'image + exécution) :

```
# docker run --rm hello-world
Hello from Docker!
This message shows that your installation appears to be working correctly.
```

Et pour ne plus taper `sudo` devant chaque `docker` :

```
# usermod -aG docker deploy
```

> À dire explicitement
> Appartenir au groupe `docker` équivaut à être root. N'y mets que toi. Déconnecte-toi puis reconnecte-toi pour que ça prenne effet (`newgrp docker` pour la session en cours).

1.9 Le swap : 5 minutes qui évitent une soirée perdue

Sur un VPS 4 Go, un build d'image ou une installation de dépendances Python peut se faire tuer par le noyau. Le message est `Killed`, sans explication, et tu croiras que ton Dockerfile est cassé.

```
# fallocate -l 2G /swapfile
# chmod 600 /swapfile
# mkswap /swapfile
# swapon /swapfile
# echo '/swapfile none swap sw 0 0' >> /etc/fstab
# free -h
               total   used   free   ...
Mem:           3.8Gi   ...
Swap:          2.0Gi     0B   2.0Gi
```

La ligne `Swap:` doit montrer `2.0Gi` au lieu de `0B`.

1.10 Les mises à jour de sécurité automatiques

```
# apt install -y unattended-upgrades
# dpkg-reconfigure --priority=low unattended-upgrades
```

Contrôle de fin de chapitre

Les cinq commandes, et ce qu'elles doivent montrer :

```
$ ssh deploy@203.0.113.10            # entre sans mot de passe
# ufw status                          # active : 22, 80, 443
# systemctl is-active docker          # active
# docker run --rm hello-world         # "Hello from Docker!"
# ss -tulnp | grep -v '127.0.0.1\|::1'  # seulement 22, bientôt 80/443
```

CHAPITRE 2

Le domaine et le DNS

Quinze minutes de configuration, et le piège du wildcard que tout le monde rate.

2.1 Acheter

Namecheap, Porkbun, ou un `.ci` local. Vérifie deux choses avant de payer : le **prix de renouvellement** (les promos à 1 $ se renouvellent souvent 5 à 10 fois plus cher, voir l'annexe), et que le registrar donne accès à la zone DNS, ce que tous ne font pas sur les promos.

2.2 Les deux enregistrements

TYPE	NOM	VALEUR	TTL
A	`@`	`203.0.113.10`	300
A	`*`	`203.0.113.10`	300

Le premier fait pointer `mondomaine.com`. **Le second, le wildcard, est celui que tout le monde rate** : il fait pointer d'un coup `gowa.mondomaine.com`, `api.mondomaine.com`, `dokploy.mondomaine.com` et tous ceux que tu inventeras plus tard. Sans lui, chaque nouveau sous-domaine demande un retour chez le registrar et 15 minutes d'attente, et comme tu auras oublié pourquoi, tu concluras que Traefik est cassé.

TTL à 300 (5 minutes) pendant l'installation : si tu te trompes d'IP, tu corriges en 5 minutes au lieu de 4 heures. Tu le remonteras à 3600 quand tout marchera.

2.3 Vérifier la propagation

```
$ dig +short gowa.mondomaine.com
203.0.113.10
```

Sur Windows, `dig` n'existe pas par défaut :

```
$ nslookup gowa.mondomaine.com 1.1.1.1
```

On interroge explicitement `1.1.1.1` : le résolveur de ton fournisseur d'accès garde souvent en cache la réponse « ce domaine n'existe pas » pendant des heures. C'est la cause numéro 1 des « chez moi ça ne marche pas alors que chez toi si ».

> Le message à retenir
> Si `dig` ne renvoie rien, attends 15 minutes et recommence. **Ne touche à rien d'autre.** Modifier Traefik pendant que le DNS se propage, c'est se retrouver avec deux problèmes au lieu d'un.

2.4 Le piège Cloudflare

Si tu mets ton domaine chez Cloudflare, l'icône **orange** (proxy activé) casse la délivrance du certificat Let's Encrypt du chapitre 4 et fait passer tout ton trafic par Cloudflare, webhook compris.

Consigne du livre : **nuage gris, « DNS only », pour tout ce tutoriel**. Tu pourras activer le proxy plus tard, en connaissance de cause.

> Contrôle
> `dig +short test.mondomaine.com` renvoie l'IP du VPS, avec un sous-domaine inventé, pour prouver que le wildcard marche.

CHAPITRE 3

Installer Dokploy

Ton interface de déploiement : elle garde tes variables, redémarre tes conteneurs, et te donne des logs cliquables.

3.1 L'installation

```
# curl -sSL https://dokploy.com/install.sh | sh
```

Ce que le script fait vraiment, puisqu'on te demande de piper un script inconnu dans un shell root :

1. installe Docker s'il manque (chez nous il est déjà là, chapitre 1) ;
2. initialise **Docker Swarm** (`docker swarm init`). Dokploy déploie en mode Swarm : c'est pour ça que certains conteneurs apparaissent dans `docker service ls` et pas seulement dans `docker ps` ;
3. crée le réseau `dokploy-network`. **Retiens ce nom : c'est le cœur de la panne 504 du chapitre 10** ;
4. lance quatre conteneurs : `dokploy`, `dokploy-traefik`, `dokploy-postgres`, `dokploy-redis` ;
5. sert l'interface sur le port **3000**.

Tu veux lire le script avant de l'exécuter ? Tu as raison : `curl -sSL https://dokploy.com/install.sh | less`.

3.2 Ouvrir le 3000, temporairement

```
# ufw allow 3000/tcp
# ufw status
```

3.3 Créer le compte admin, tout de suite

Ouvre `http://203.0.113.10:3000` (en HTTP, sans certificat, c'est normal à ce stade).

> Piège critique
> **L'inscription est ouverte tant que personne ne s'est inscrit.** Entre l'installation et la création de ton compte, n'importe qui scannant le port 3000 peut prendre le contrôle de ton serveur. Ce n'est pas une hypothèse : les scanners trouvent une IP fraîche en quelques minutes. **Crée le compte dans la minute qui suit l'installation**, avant même de faire le tour de l'interface.

3.4 Le vocabulaire Dokploy, en quatre mots

TERME	CE QUE C'EST VRAIMENT
Project	un dossier de rangement. Aucun effet technique.
Application	une app construite par Dokploy depuis un dépôt Git
Compose	un `docker-compose.yml` que Dokploy exécute. **C'est ce qu'on utilise dans tout le livre.**
Domain	un sous-domaine + un port de conteneur ; Dokploy en déduit les labels Traefik

3.5 Donner un domaine à Dokploy, puis refermer le 3000

Dans *Settings → Server → Domain* : `dokploy.mondomaine.com`, port 3000, HTTPS et Let's Encrypt activés. Une fois que `https://dokploy.mondomaine.com` répond :

```
# ufw delete allow 3000/tcp
# ufw status
```

L'interface reste accessible par le domaine, en HTTPS, via Traefik. Le port brut n'est plus ouvert au monde. C'est exactement le raisonnement du paragraphe 1.6, appliqué à ta première vraie application.

```
# docker ps --format 'table {{.Names}}\t{{.Status}}\t{{.Ports}}'
```

> Contrôle
> `https://dokploy.mondomaine.com` répond avec un cadenas valide, et `ufw status` ne contient plus 3000.

CHAPITRE 4

Traefik et le HTTPS

Le seul processus qui écoute sur 80 et 443, et les six labels qu'il faut avoir décortiqués une fois dans sa vie.

4.1 Ce que fait Traefik, en une phrase

Traefik lit en permanence les **labels** des conteneurs Docker et se reconfigure tout seul : pas de fichier de conf à recharger, pas de `nginx -s reload`. Quand tu ajoutes un domaine dans Dokploy, Dokploy ajoute des labels au conteneur, et Traefik les voit dans la seconde.

```
labels:
  - traefik.enable=true
  - traefik.http.routers.gowa.rule=Host(`gowa.mondomaine.com`)
  - traefik.http.routers.gowa.entrypoints=websecure
  - traefik.http.routers.gowa.tls.certresolver=letsencrypt
  - traefik.http.services.gowa.loadbalancer.server.port=3080
  - traefik.docker.network=dokploy-network
```

LABEL	TRADUCTION EN FRANÇAIS
`routers.gowa.rule`	« quand une requête arrive avec ce nom d'hôte… »
`entrypoints=websecure`	« …sur le port 443… »
`tls.certresolver`	« …avec un certificat que tu vas chercher chez Let's Encrypt… »
`loadbalancer.server.port`	« …envoie-la sur le port **interne** du conteneur »
`traefik.docker.network`	« …et pour l'atteindre, utilise **ce** réseau » (voir 5.4)

La confusion numéro 1 sur Traefik

Le port du label `loadbalancer.server.port` est le port **dans** le conteneur, pas un port publié. Mettre 443, ou le port publié sur l'hôte, donne un **502**. Le conteneur n'a besoin d'aucun `ports:` pour que Traefik l'atteigne : ils sont sur le même réseau Docker.

4.2 Let's Encrypt

Dans Dokploy, une adresse mail suffit (*Settings → Certificates*). Ce qui se passe derrière : à la première requête sur le domaine, Traefik demande un certificat, Let's Encrypt appelle `http://gowa.mondomaine.com/.well-known/...` sur le port **80**, Traefik répond, le certificat est délivré et stocké dans `acme.json`. Deux conséquences pratiques :

- **le port 80 doit rester ouvert** même si tout ton site est en HTTPS. Le fermer « parce qu'on est en HTTPS » casse le renouvellement 60 jours plus tard : une panne à retardement, très dure à diagnostiquer ;
- **le DNS doit être propagé avant** (chapitre 2), sinon la validation échoue.

Suivre la délivrance en direct :

```
# docker logs -f dokploy-traefik 2>&1 | grep -i -E 'acme|certificate|error'
```

> Le quota qui piège
> Let's Encrypt limite les échecs : 5 par heure et par domaine, 50 certificats par semaine. Recharger sa page 20 fois pendant que le DNS n'est pas propagé, c'est se faire bloquer une heure, et croire que tout est cassé. Tu es prévenu **avant** de le vivre.

4.3 Vérifier le certificat

```
$ curl -I https://gowa.mondomaine.com
HTTP/2 401   # 401 = Traefik a routé, gowa demande le mot de passe. C'est bon.
```

```
$ echo | openssl s_client -connect gowa.mondomaine.com:443 -servername gowa.mondomaine.com 2>/
dev/null | openssl x509 -noout -issuer -dates
issuer=C = US, O = Let's Encrypt, CN = R11
notBefore=...
notAfter=...
```

Si `issuer` contient `TRAEFIK DEFAULT CERT`, le certificat n'a **pas** été délivré : c'est le certificat bouche-trou de Traefik. Retour au paragraphe 4.2.

> Contrôle
> `https://<un-sous-domaine>.mondomaine.com` affiche un cadenas valide, et `openssl` montre Let's Encrypt comme émetteur.

CHAPITRE 5

Déployer gowa

gowa (go-whatsapp-web-multidevice) transforme un numéro WhatsApp en API REST. Le compose ci-dessous est corrigé des quatre défauts qui coûtent des soirées : lis les commentaires.

5.1 Générer ses identifiants, avant de coller quoi que ce soit

```
# openssl rand -base64 24
kQ7mZp2xN9vR4tLwEa6YcHs1BdJfXu0T
```

Ce livre ne donne **aucun** identifiant par défaut, et c'est volontaire : un même mot de passe recopié par des centaines de lecteurs, c'est des centaines d'instances exposées avec un mot de passe connu. Cette commande est une étape numérotée du chapitre, pas une note de bas de page. Lance-la deux fois : une pour le mot de passe, une pour le secret du webhook.

5.2 Le docker-compose.yml

Dans Dokploy : *Create* → *Compose* → *Raw*, puis :

```
services:
  gowa:
    image: aldinokemal2104/go-whatsapp-web-multidevice:v8.10.0   # tag épinglé, voir 5.3
    container_name: gowa
    restart: unless-stopped
    command:
      - rest
      - --port=3080
      - --debug=false                    # voir 5.3
      - --os=Chrome
      - --account-validation=false
      - --basic-auth=${WA_USER}:${WA_PASSWORD}
      - --webhook=${WEBHOOK_URL}
      - --webhook-secret=${WEBHOOK_SECRET}
    ports:
      - "127.0.0.1:3080:3080"            # débogage local, zéro exposition
    volumes:
      - gowa_storage:/app/storages       # la session WhatsApp vit ici
    networks:
      - dokploy-network
    labels:
      - traefik.enable=true
      - traefik.http.routers.gowa.rule=Host(`gowa.mondomaine.com`)
      - traefik.http.routers.gowa.entrypoints=websecure
      - traefik.http.routers.gowa.tls.certresolver=letsencrypt
      - traefik.http.services.gowa.loadbalancer.server.port=3080
      - traefik.docker.network=dokploy-network   # la ligne qui évite le 504

volumes:
  gowa_storage:

networks:
  dokploy-network:
    external: true
```

Et les variables, dans l'onglet *Environment* de Dokploy, **pas** dans le compose, qui finira sur GitHub un jour :

```
WA_USER=admin
WA_PASSWORD=kQ7mZp2xN9vR4tLwEa6YcHs1BdJfXu0T
WEBHOOK_URL=http://api:8000/webhook
WEBHOOK_SECRET=<autre openssl rand -base64 24>
```

5.3 Les quatre lignes à comprendre, pas juste coller

Le volume `gowa_storage`. La session WhatsApp, c'est-à-dire les clés de chiffrement de l'appareil apparié, est un fichier dans `/app/storages`. Sans volume nommé, elle disparaît au moindre redéploiement, et il faut rescanner le QR code. C'est la panne « le numéro est déconnecté au matin » du chapitre 10, et elle est purement due à cette ligne.

Le tag `:v8.10.0`. Sans tag, Docker prend `:latest`. Installer dans six mois, c'est alors récupérer une version dont les options ont changé, et ce livre ne correspondrait plus. Si tu veux une version plus récente : change le tag ici, et relis le changelog du dépôt avant.

`--debug=false`. En `true`, `docker logs` déverse des milliers de lignes de trafic WebSocket brut. Or `docker logs` est l'outil numéro 1 du chapitre dépannage : le debug rend inutilisable ce dont tu as le plus besoin.

`ports: 127.0.0.1:3080:3080`. Débogage possible depuis le VPS (`curl localhost:3080`), aucune exposition publique. Vérification immédiate :

```
# ss -tulnp | grep 3080
LISTEN 0 4096 127.0.0.1:3080 0.0.0.0:*
```

Si tu vois `0.0.0.0:3080`, la ligne `ports:` est mal écrite, et ton interface gowa est accessible sans HTTPS et sans passer par Traefik.

5.4 Trouver le vrai nom du réseau

Le nom du réseau que Dokploy injecte peut différer selon les versions. **Ne devine pas, lis-le** :

```
# docker inspect gowa --format '{{range $k,$v := .NetworkSettings.Networks}}{{$k}} {{end}}'
dokploy-network
```

Si cette commande renvoie **deux** noms, le conteneur est sur deux réseaux, et Traefik tire au sort l'IP du backend : une fois sur deux il tombe sur celle qu'il ne route pas, et tu obtiens un **504 intermittent**, la pire panne à diagnostiquer. Le label `traefik.docker.network` doit alors contenir le réseau que Traefik utilise aussi :

```
# docker inspect dokploy-traefik --format '{{range $k,$v := .NetworkSettings.Networks}}{{$k}}
{{end}}'
```

L'intersection des deux listes, c'est la valeur à mettre. Cette paire de commandes vaut à elle seule une soirée pour qui a déjà vécu un 504 intermittent.

> Contrôle
> `https://gowa.mondomaine.com` affiche la fenêtre de login basic auth, tes identifiants passent, et tu recharges cinq fois de suite : **aucun 504**.

CHAPITRE 6

Connecter WhatsApp

La seule étape irréversible du livre. On la fait avec une puce de test, jamais avec ton numéro.

6.1 Avant de scanner

> Rappel, au moment exact où il compte
> Un numéro banni ne se récupère pas. La puce de test à 500 F, c'est maintenant qu'elle sert.

Le téléphone qui scanne doit rester **allumé et connecté** au début : WhatsApp multi-device permet à l'appareil lié de fonctionner téléphone éteint, mais l'appairage initial, lui, exige le téléphone en ligne.

6.2 Scanner

1. `https://gowa.mondomaine.com` → basic auth ;
2. bouton *Login* (route `/app/login`) → un QR code s'affiche ;
3. sur le téléphone : *WhatsApp → Paramètres → Appareils connectés → Connecter un appareil* ;
4. scanner. Le QR expire en 20 secondes environ : c'est normal qu'il se régénère.

> Attention
> Ne filme jamais ce QR code, ne le partage jamais, ne le mets pas dans une capture d'écran. Quiconque le scanne prend la session.

6.3 Récupérer l'identifiant de l'appareil

```
$ curl -s -u admin:MOT_DE_PASSE https://gowa.mondomaine.com/app/devices
{
  "code": "SUCCESS",
  "results": [
    { "name": "Mon Commerce", "device": "22507000000:12@s.whatsapp.net" }
  ]
}
```

Si `results` est vide, l'appairage n'a pas abouti : recommence le scan. Cette valeur `device` ira dans les variables de l'API au chapitre 7 : c'est elle qui identifie depuis quel compte on envoie, le jour où tu gères plusieurs numéros.

6.4 Prouver que la session survit

Le test qui vaut mieux qu'une explication : redéploie depuis Dokploy, puis rappelle `/app/devices`. Si l'appareil est toujours là, le volume du 5.3 fonctionne. S'il faut rescanner, le volume est mal monté, et il vaut mille fois mieux le découvrir maintenant que dans trois semaines, en production, chez un client.

```
# docker volume ls | grep gowa
# docker volume inspect <nom-du-volume> --format '{{.Mountpoint}}'
```

> Contrôle

> `/app/devices` renvoie l'appareil connecté, **et le renvoie encore après un redéploiement**.

CHAPITRE 7

L'API FastAPI : le cerveau du bot

Deux conteneurs, un réseau, aucun port public. Et le code complet du serveur, commenté.

7.1 Le principe

```
gowa ──http://api:8000/webhook──► api          (réseau Docker interne)
api  ──http://gowa:3080/send/message──► gowa  (réseau Docker interne)
```

Aucun de ces deux appels ne sort du VPS. Pas de HTTPS, pas de nom de domaine, pas de port publié : **le nom du service Docker suffit**, Docker fait le DNS interne. C'est ici que tu comprends enfin pourquoi on a mis les deux services sur le même réseau.

> Le piège qui rend le bot muet
> Mettre `WEBHOOK_URL=https://gowa.mondomaine.com` ou l'IP publique du VPS. Ça part sur Internet, revient sur Traefik, et selon la configuration réseau, se perd. Le webhook doit être `http://api:8000/webhook`, en clair, par nom de conteneur.

7.2 Les fichiers du projet

Quatre fichiers dans un dépôt Git (ou un dossier collé dans Dokploy) :

`requirements.txt`

```
fastapi
uvicorn[standard]
httpx
langchain-openai
langchain-core
```

`Dockerfile`

```
FROM python:3.12-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

`docker-compose.yml (deuxième Compose dans Dokploy)`

```
services:
  api:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: api
    restart: unless-stopped
    environment:
      - GOWA_INTERNAL_URL=http://gowa:3080
      - WA_USER=${WA_USER}
      - WA_PASSWORD=${WA_PASSWORD}
      - WEBHOOK_SECRET=${WEBHOOK_SECRET}
      - DEVICE_ID=${DEVICE_ID}
      - AI_API_KEY=${AI_API_KEY}
    networks:
      - dokploy-network
    # pas de ports: — l'API n'a rien à exposer sur Internet

networks:
  dokploy-network:
    external: true
```

`GOWA_INTERNAL_URL=http://gowa:3080` : `gowa` est le `container_name` du chapitre 5, `3080` le port interne. C'est tout.

7.3 Vérifier que les deux se voient, avant d'écrire du code

```
# docker exec api curl -s -o /dev/null -w '%{http_code}\n' http://gowa:3080/
401
```

`401` est un **succès** ici : le conteneur a été trouvé, le DNS interne a résolu, la connexion TCP a abouti, et gowa a répondu « donne-moi le mot de passe ». Ce qu'on cherche, c'est un code HTTP, n'importe lequel.

SORTIE	CAUSE
`Could not resolve host: gowa`	les deux conteneurs ne sont **pas** sur le même réseau
`Connection refused`	même réseau, mais gowa est mort ou n'écoute pas sur 3080
(le curl reste bloqué)	mauvais port, ou un pare-feu entre les deux

Et le test dans l'autre sens :

```
# docker exec gowa wget -qO- http://api:8000/health
```

Si les deux passent, la plomberie est bonne. **Tout ce qui ne marchera pas ensuite sera du code, pas du réseau.** C'est un immense gain de temps de diagnostic.

7.4 Le code complet du serveur

`main.py`

```
import asyncio, hashlib, hmac, os, random, time

import httpx
from fastapi import FastAPI, HTTPException, Request

from ia import generer_reponse          # chapitre 8

GOWA = os.environ["GOWA_INTERNAL_URL"]
AUTH = (os.environ["WA_USER"], os.environ["WA_PASSWORD"])
SECRET = os.environ["WEBHOOK_SECRET"]

app = FastAPI()
file_attente: asyncio.Queue = asyncio.Queue()

# ---------- sécurité : vérifier la signature HMAC ----------
async def verifier_signature(request: Request) -> bytes:
    corps = await request.body()
    recue = request.headers.get("X-Hub-Signature-256", "").removeprefix("sha256=")
    attendue = hmac.new(SECRET.encode(), corps, hashlib.sha256).hexdigest()
    if not hmac.compare_digest(recue, attendue):
        raise HTTPException(status_code=401, detail="signature invalide")
    return corps

# ---------- réception : on répond 200 tout de suite ----------
@app.post("/webhook")
async def webhook(request: Request):
    await verifier_signature(request)
    data = await request.json()
    if data.get("event") == "message":
        await file_attente.put((time.monotonic(), data))
    return {"ok": True}

@app.get("/health")
async def health():
    return {"ok": True}

# ---------- envoi vers gowa ----------
async def gowa_post(chemin: str, corps: dict):
    async with httpx.AsyncClient(timeout=30) as c:
        r = await c.post(f"{GOWA}{chemin}", json=corps, auth=AUTH)
        r.raise_for_status()

# ---------- le worker : un seul, qui sérialise les envois ----------
async def repondre():
    while True:
        recu_a, data = await file_attente.get()
        try:
            p = data["payload"]
            expediteur = p["from"]
            texte = p.get("message", {}).get("text", "")
            if not texte:
                continue

            # 1. accusé de lecture immédiat : deux coches bleues
            await gowa_post(f"/message/{p['message']['id']}/read",
                            {"phone": expediteur})

            # 2. "en train d'écrire..."
```

```
            await gowa_post("/send/chat-presence",
                            {"phone": expediteur, "action": "start"})

            # 3. générer la réponse (DeepSeek, chapitre 8)
            reponse = await generer_reponse(texte)

            # 4. le délai humain — voir chapitre 11 :
            #    on ne rajoute JAMAIS de latence par-dessus la latence réelle
            lecture = random.uniform(0.6, 1.8)
            frappe = len(reponse) / random.uniform(8, 14)
            cible = min(max(lecture + frappe, 1.2), 8.0)
            ecoule = time.monotonic() - recu_a
            await asyncio.sleep(max(0, cible - ecoule))

            # 5. envoyer, puis arrêter l'indicateur
            await gowa_post("/send/message",
                            {"phone": expediteur, "message": reponse})
            await gowa_post("/send/chat-presence",
                            {"phone": expediteur, "action": "stop"})
        except Exception as e:
            print(f"erreur worker: {e}", flush=True)

@app.on_event("startup")
async def demarrage():
    asyncio.create_task(repondre())
```

7.5 Trois choix de ce code, expliqués

`hmac.compare_digest` et pas `==`. La comparaison naïve s'arrête au premier caractère différent, ce qui laisse fuir de l'information par le temps d'exécution.

On signe le corps brut, avant tout parsing JSON. Un `json.dumps` de l'objet reparsé ne redonne pas les mêmes octets (espaces, ordre des clés), et la signature ne correspondrait jamais.

Un seul worker. La file sérialise les envois : l'ordre des messages est garanti (une conversation dans le désordre est un bug visible) et le compteur de débit du chapitre 11 devient trivialement correct.

7.6 Le premier signe de vie

```
# docker logs -f --tail=50 api
```

Envoie un message WhatsApp au numéro de test depuis un autre téléphone. Une ligne doit apparaître dans les logs **dans la seconde**. Si rien n'apparaît, l'ordre de diagnostic compte :

```
# docker logs --tail=30 gowa | grep -i webhook    # gowa a-t-il seulement essayé ?
# docker exec gowa wget -qO- http://api:8000/health  # peut-il joindre l'API ?
# docker ps --filter name=api                     # l'API tourne-t-elle vraiment ?
```

> Contrôle
> Un message envoyé au numéro apparaît dans `docker logs api` en moins d'une seconde.

CHAPITRE 8

Brancher l'IA (DeepSeek)

La brique la plus simple du livre, à condition de tester la clé isolément avant de l'assembler au reste.

8.1 La clé

`platform.deepseek.com` → *API Keys* → créer. Recharge 5 $. La clé n'est affichée **qu'une fois** : colle-la tout de suite dans les variables Dokploy (`AI_API_KEY`).

L'API DeepSeek est compatible OpenAI : même bibliothèque, on change juste `base_url`. Inutile de chercher un « SDK DeepSeek ».

8.2 Tester la clé avant d'écrire une ligne d'application

```
$ curl https://api.deepseek.com/chat/completions \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer sk-..." \
  -d '{"model":"deepseek-chat","messages":[{"role":"user","content":"dis bonjour"}]}'
```

C'est **la** bonne habitude à prendre : on valide chaque brique isolément avant de les assembler. Trois briques testées séparément se débuggent en trois minutes ; trois briques assemblées d'un coup se débuggent en trois heures.

8.3 La chaîne minimale

`ia.py`

```
import os
from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import StrOutputParser

llm = ChatOpenAI(
    model="deepseek-chat",
    base_url="https://api.deepseek.com",
    api_key=os.environ["AI_API_KEY"],
    temperature=0.3,   # bas : on veut des infos justes, pas de la créativité
    max_tokens=400,    # une réponse WhatsApp, pas une dissertation
    timeout=30,        # sans ça, une requête bloquée fige le worker
    max_retries=2,
)

SYSTEME = """Tu es l'assistant WhatsApp de {commerce}.
Tu réponds en français simple, en 3 phrases maximum.
Horaires : {horaires}. Adresse : {adresse}.
Si tu ne connais pas la réponse, dis-le et propose d'appeler le {telephone}.
N'invente jamais un prix."""

chaine = (
    ChatPromptTemplate.from_messages([("system", SYSTEME), ("human", "{question}")])
    | llm
    | StrOutputParser()
)

async def generer_reponse(question: str) -> str:
    return await chaine.ainvoke({
        "commerce": "Pharmacie du Plateau",
        "horaires": "8h à 23h, tous les jours",
        "adresse": "Boulevard de la République, Abidjan",
        "telephone": "+225 07 00 00 00 00",
        "question": question,
    })
```

Les quatre paramètres méritent chacun une phrase : ce sont ceux qu'un débutant laisse par défaut, et qui font la différence entre un bot correct et un bot pénible.

- `temperature=0.3` : un commerce veut la même réponse à la même question ;
- `max_tokens=400` : au-delà, WhatsApp devient illisible, et le chapitre 11 impose de couper en 3 bulles maximum ;
- `timeout=30` : sans timeout, une requête suspendue immobilise le worker, et le bot cesse de répondre à tout le monde ;
- `N'invente jamais un prix` : la seule ligne du prompt qui te protège d'un vrai litige commercial.

8.4 Ce que ce chapitre ne fait volontairement pas

Pas de RAG, pas de base vectorielle, pas de mémoire de conversation. L'objectif est « ça marche », pas « c'est sophistiqué ». Quand ton bot tournera chez un vrai commerce et que tu voudras qu'il connaisse un catalogue de 300 produits, ce sera le moment d'en parler : c'est exactement le genre de chantier où mon assistance te fait gagner le plus de temps.

8.5 Le coût réel, en clair

Une conversation typique tient dans environ 800 tokens d'entrée et 200 de sortie. Aux tarifs DeepSeek constatés en août 2026, 5 $ de crédit couvrent plusieurs milliers de conversations de ce format : vérifie les tarifs du jour sur la plateforme et fais le calcul avant de vendre. Un lecteur qui sait combien coûte une conversation peut construire une offre : c'est la donnée qui transforme ce tutoriel en business.

> Contrôle
> Le numéro de test répond une phrase cohérente, générée par l'IA, qui reprend le nom du commerce mis dans le prompt système.

CHAPITRE 9

Le test de bout en bout

On mesure, on ne ressent pas.

9.1 Le parcours, vu du client

Depuis un téléphone tiers, pas celui de la puce de test, écris au numéro :

1. « bonjour » → réponse d'accueil ;
2. « vous ouvrez à quelle heure ? » → les horaires du prompt système ;
3. « c'est combien la livraison à Cocody ? » → **le bot doit dire qu'il ne sait pas** et renvoyer vers le téléphone. C'est le test le plus important des trois : il vérifie le garde-fou anti-invention.

9.2 Mesurer

```
# docker logs --tail=200 api | grep -E 'webhook|reponse'
```

Trace deux horodatages : réception du webhook, envoi de la réponse. **Objectif : moins de 10 secondes.** Au-delà, l'utilisateur croit que personne ne répond et écrit une deuxième fois. Pendant le test, dans un second terminal :

```
# docker stats --no-stream
```

Si la RAM du conteneur API grimpe à chaque message sans jamais redescendre, il y a une fuite. Mieux vaut la voir sur 10 messages que sur 10 000.

9.3 Se déconnecter proprement

```
$ curl -s -u admin:MOT_DE_PASSE -X GET https://gowa.mondomaine.com/app/logout
```

Et côté téléphone : *Paramètres → Appareils connectés → Se déconnecter*. Fais les deux. Une session fermée d'un seul côté laisse gowa croire qu'il est connecté : il tentera de se reconnecter en boucle, et ce comportement, répété, n'aide pas la réputation du numéro.

9.4 Passer sur le numéro définitif

La procédure est la même : logout, on change de puce, on rescanne. Ce qui compte, c'est le **warm-up** du chapitre 11 : un numéro tout neuf qui se met à répondre à 200 personnes le premier jour est un candidat au ban. Montée progressive sur deux semaines environ.

> Contrôle
> Aller-retour complet en moins de 10 secondes, et le bot admet son ignorance sur la question piège.

CHAPITRE 10

Dépannage : 11 pannes réelles

La section la plus rentable du livre : elle transforme 3 heures de galère en 3 minutes. Chaque panne a été vécue, pas imaginée.

10.1 Les cinq commandes à connaître par cœur

```
# docker ps -a                                    # qui tourne, qui est mort, depuis quand
# docker logs --tail=100 -f <conteneur>           # ce qu'il dit
# docker inspect <ctr> --format '{{json .Config.Labels}}'   # ce que Traefik voit
# docker exec <ctr-a> curl -s -o /dev/null -w '%{http_code}\n' http://<ctr-b>:<port>/   # se voient-ils ?
# ss -tulnp | grep -v '127.0.0.1\|::1'            # qu'est-ce qui est exposé ?
```

10.2 La table symptôme → cause → la commande qui tranche

SYMPTÔME	CAUSE LA PLUS PROBABLE	CE QUI TRANCHE
504 Gateway Timeout	Traefik atteint le conteneur mais l'IP n'est pas routable (deux réseaux, voir 5.4)	`docker exec dokploy-traefik nc -z <ip-du-conteneur> 3080` sur **chaque** IP
502 Bad Gateway	mauvais port dans les labels, ou conteneur mort	`docker ps -a`, puis comparer le port du label au port réel
404 sur le sous-domaine	pas de routeur Traefik : labels absents ou app pas démarrée	`docker inspect <ctr> --format '{{json .Config.Labels}}'`
Certificat invalide	DNS pas propagé, ou wildcard manquant (ch. 2)	`dig +short sousdomaine.mondomaine.com`
TRAEFIK DEFAULT CERT	Let's Encrypt a échoué, ou quota d'échecs atteint	`docker logs dokploy-traefik 2>&1 | grep -i acme`
Le QR ne s'affiche pas	session déjà active, ou stockage non persistant	vérifier le volume `/app/storages`
Le bot est muet	l'URL du webhook pointe dans le vide, ou les deux conteneurs ne partagent pas de réseau	`docker exec api curl -s http://gowa:3080/`
Le numéro est déconnecté au matin	volume non persistant : session perdue au redéploiement	`docker volume ls`, vérifier que le volume est nommé
Réponses très lentes	crédit DeepSeek épuisé, ou VPS qui swappe	`free -h`, puis le tableau de bord DeepSeek
`Killed` pendant un build	plus de RAM, pas de swap	`free -h`, créer le swap (ch. 1, §1.9)
Un port ouvert malgré `ufw`	Docker publie en contournant UFW	`ss -tulnp | grep -v '127.0.0.1'`, puis corriger `ports:`

10.3 La distinction 502 / 504 : le meilleur enseignement du chapitre

Un **timeout (504)** signifie que la connexion TCP a abouti mais que rien n'est revenu. Un **502** signifie qu'elle a été refusée tout de suite. Cette distinction oriente le diagnostic en dix secondes : 504, cherche un problème de routage ou de réseau (5.4) ; 502, cherche un mauvais port ou un conteneur mort. Presque personne ne l'explique, et elle vaut de l'or.

CHAPITRE 11

Ne pas se faire bannir

Ce chapitre n'a pas d'équivalent ailleurs, parce qu'il faut s'être fait couper un numéro pour l'écrire.

11.1 Deux problèmes qu'on confond toujours

	MENACE	OUTIL
Entrant	un utilisateur qui spamme ton bot	limiteur HTTP côté API
Sortant	ton compte WhatsApp qui envoie trop et se fait bannir	plafonds de débit, par appareil

Un limiteur HTTP ne protège **pas** ton numéro d'un ban : il protège ton serveur d'une surcharge. Ce sont deux mécanismes distincts, et les confondre est l'erreur classique.

11.2 La séquence d'envoi humaine

Tout est déjà exposé par gowa, il n'y a rien à inventer ; c'est exactement la séquence codée au chapitre 7 :

1. `POST /message/{id}/read` : accusé de lecture **immédiat**. Le geste le plus rentable : l'utilisateur voit les deux coches bleues et arrête d'attendre ;
2. `POST /send/presence` (`available`) : une fois par session, pas par message ;
3. `POST /send/chat-presence` (`start`) : « en train d'écrire… » ;
4. attendre le délai calculé (11.3) ;
5. `POST /send/message` ;
6. `POST /send/chat-presence` (`stop`).

> Le piège de l'indicateur de frappe
> L'indicateur expire côté serveurs WhatsApp au bout de 25 secondes environ. Pour une génération IA de 40 secondes, il faut renvoyer `start` toutes les 10 secondes, sinon l'utilisateur voit le bot « arrêter d'écrire », puis un message surgir d'un coup : **pire que pas d'indicateur du tout**. Une petite tâche `asyncio` de rafraîchissement, annulée juste avant l'envoi, règle le cas.

11.3 Le délai est presque toujours gratuit

```
lecture = uniform(0.6, 1.8)                # temps de « lire » le message reçu
frappe  = len(reponse) / uniform(8, 14)    # 8 à 14 caractères/s, dactylo mobile rapide
cible   = clamp(lecture + frappe, 1.2, 8.0)
attente = max(0, cible - temps_deja_ecoule)  # LA règle
```

On ne rajoute jamais de latence par-dessus la latence réelle. Le LLM prend déjà 3 à 15 secondes : le délai « humain » est donc gratuit dans la quasi-totalité des cas. Il ne s'applique vraiment qu'aux réponses instantanées (menus, erreurs, confirmations), et ce sont précisément celles qui, envoyées en 50 millisecondes, trahissent un robot.

Découpage : au-delà de 700 caractères environ, coupe sur les paragraphes, **3 bulles maximum**, avec indicateur de frappe entre chaque. Au-delà de 3, on ne fait plus humain, on fait spam.

Variation : 3 à 5 formulations par message système, tirées au sort. Un texte strictement identique envoyé 500 fois est un marqueur en soi.

11.4 Les plafonds sortants

Points de départ prudents, à ajuster avec l'usage réel :

- 20 messages par minute maximum ;
- 600 messages par heure maximum ;
- 20 messages par heure vers **un même** destinataire : le garde-fou anti-boucle. Un bug de récursion sur le webhook, où le bot se répond à lui-même, est la façon numéro 1 de se faire bannir en une nuit ;
- **heures calmes** : aucun envoi entre 23 h et 06 h (heure d'Abidjan), *sauf* en réponse directe à un message entrant. Là, c'est l'utilisateur qui a écrit : c'est normal ;
- **warm-up** d'un nouveau numéro : montée progressive sur deux semaines environ.

Sérialisation : une file dédiée aux envois, un seul worker par appareil (c'est le code du chapitre 7). L'ordre des messages est garanti, et le compteur de débit devient trivialement correct.

11.5 La règle qui compte plus que toutes les autres

À encadrer

Ne jamais initier de conversation en masse. Les bans viennent presque exclusivement du premier contact non sollicité et du taux de blocage ou de signalement, pas de la vitesse de frappe. Ton produit est déclenché par l'entrant : garde-le comme ça. Toute l'humanisation ci-dessus ne sert à rien si un jour tu envoies 300 messages à froid.

CHAPITRE 12

Encaisser : paiements mobile money

Le bot répond, maintenant il encaisse. MoneyFusion pour le mobile money, et le webhook idempotent qui t'évite de créditer deux fois.

12.1 Le circuit

MoneyFusion : créer le compte, obtenir les clés, générer un lien de paiement, recevoir le webhook de confirmation. Le client paie par Wave, Orange Money ou MTN MoMo, l'agrégateur notifie ton serveur, et ton bot confirme dans la conversation. Vendre ce livre avec ce même circuit, c'est d'ailleurs ce chapitre en démonstration.

12.2 Le webhook doit être idempotent

> Le bug qui coûte de l'argent réel
> Les agrégateurs **rejouent** leurs notifications : timeout de ton côté, retry du leur, et le même paiement arrive deux, trois, cinq fois. Sans protection, le client est crédité autant de fois.

La protection ne se met pas dans le code applicatif, mais dans la **base de données** :

```
CREATE TABLE paiements (
  id            SERIAL PRIMARY KEY,
  reference     TEXT NOT NULL UNIQUE,   -- la référence de l'agrégateur
  montant       INTEGER NOT NULL,
  statut        TEXT NOT NULL,
  cree_le       TIMESTAMPTZ NOT NULL DEFAULT now()
);
```

```
INSERT INTO paiements (reference, montant, statut)
VALUES ($1, $2, $3)
ON CONFLICT (reference) DO NOTHING
RETURNING id;
```

Si `RETURNING` ne renvoie rien, c'est un rejeu : on répond `200 OK` et **on ne crédite pas**. Le point à marteler : un `SELECT` suivi d'un `INSERT` ne suffit pas. Deux webhooks arrivés en même temps passent tous les deux le `SELECT`. Seule la contrainte `UNIQUE` en base est atomique.

12.3 Ne jamais faire confiance au montant reçu

Le corps du webhook est une donnée qui vient d'Internet. Deux règles :

1. **vérifier la signature** du webhook, même logique HMAC qu'au chapitre 7 ;
2. **rappeler l'API de l'agrégateur** avec la référence, pour lire le statut et le montant réels, plutôt que de croire le payload.

```
@app.post("/paiement/webhook")
async def paiement(req: Request):
    notif = await req.json()
    ref = notif["reference"]

    if not inserer_si_nouveau(ref):      # l'INSERT ... ON CONFLICT ci-dessus
        return {"ok": True}              # rejeu : on répond 200, on ne crédite pas

    statut, montant = await verifier_aupres_api(ref)   # on ne croit pas le payload
    if statut == "paid":
        crediter(ref, montant)
    return {"ok": True}
```

12.4 Répondre vite

Un webhook qui met 30 secondes à répondre sera considéré en échec et rejoué. La bonne forme : valider la signature, écrire en base, répondre `200`, et faire le travail lourd (envoyer le message WhatsApp de confirmation, générer la facture) **après**, dans une tâche de fond. C'est le même schéma file-plus-worker que le chapitre 7 : tu l'as déjà.

> Contrôle
> Renvoie deux fois la même notification de test : le compte n'est crédité qu'une fois, et les deux appels reçoivent un 200.

ANNEXE

Les vrais coûts, les licences, et après

A.1 La vérité sur le « 10 $ »

On dit « environ 10 $ **pour démarrer** », jamais « 10 $ par mois ». Voilà le détail, tarifs constatés en août 2026 : vérifie les prix du jour, ils bougent.

POSTE	ORDRE DE GRANDEUR	À SAVOIR
VPS 4 Go	~4 000 F / mois	**récurrent**, avec parfois des frais de mise en service au premier paiement
Domaine	< 1 $ la première année	prix promo : le **renouvellement** est souvent 5 à 10 fois plus cher
Crédit DeepSeek	5 $	consommé à l'usage, pas mensuel ; plusieurs milliers de conversations
Puce de test	~500 F	le premier achat du tutoriel, non négociable

A.2 Licences et droit, en deux minutes

Utiliser et documenter des logiciels open source est parfaitement légitime. gowa est sous licence MIT : son code reste sur le dépôt officiel (`github.com/aldinokemal/go-whatsapp-web-multidevice`), ce livre ne redistribue rien, il pointe vers les sources et ne livre que ses propres fichiers. Ce qui est en jeu avec la méthode elle-même, c'est autre chose : automatiser WhatsApp hors API officielle va contre les conditions d'utilisation de Meta. La sanction possible est le bannissement du numéro, d'où la puce dédiée et le chapitre 11. Si l'auteur de gowa te fait gagner de l'argent, son dépôt indique comment soutenir son travail : c'est mérité.

A.3 Et après ?

Ce que tu as construit répond à une question à la fois, sans mémoire, sans catalogue. Les chantiers suivants, quand un vrai commerce tourne dessus : la mémoire de conversation, un catalogue de produits interrogeable, plusieurs numéros sur la même instance (gowa v8 gère le multi-appareil nativement), et le tableau de bord pour ton client. Chacun de ces chantiers se vend : c'est là que ton tutoriel devient un métier.

> **UN DERNIER MOT**
>
> Si ce livre t'a économisé une soirée, il a fait son travail. Si tu veux m'économiser les tiennes : l'assistance est en page 3, et le lien du groupe est en pied de chaque page. Rappel : le groupe est réservé à ceux qui prennent l'assistance payante ; rejoindre, c'est s'engager. Bonne construction.
>
> **Wapay** · wa.me/2250712116258