UPRISING NODE

STACK DECISIONS + PROD SETUP + REALTIME SPEC + OBSERVABILITY

Version: v1.0 • Date: 2026-01-26

But: verrouiller les choix techniques les plus simples a gérer et les moins couteux, puis définir un setup production-ready, une spec realtime stable, et une base d'observabilite qui évite les semaines de debug.

1. Stack Decisions (facile a creer, facile a gerer, low-cost)

Principe: moins de services = moins d'opérations. On privilégie un backend unique qui couvre DB + auth + storage + realtime.

Bloc	Decision	Pourquoi (ops)	Cout de depart
DB	Supabase Postgres	Postgres managé + tooling + migrations standard	$0 (free plan)
Auth (OTP)	Supabase Auth	OTP/magic link integrés, pas de provider externe a maintenir	$0 (free plan)
Realtime (presence, chat)	Supabase Realtime	Broadcast + Presence + DB changes, un seul système	$0 (free plan)
Storage (PDF/assets)	Supabase Storage	Buckets + URLs signées, simple pour PDFs	$0 (free plan)
Queue	Upstash Redis + BullMQ	Redis managé (low ops) + orchestration jobs BullMQ	$0 (free tier)
Web	Next.js (App Router)	DX rapide, perf, deployment simple, desktop-first	$0 (hosting selon)
API	NestJS (REST)	Structure modulaire + guards RBAC	$0 (self code)
AI	FastAPI	Service isolé, scalable et simple	$0 (self code)
Observabilite	Sentry	Erreurs + tracing + budget PAYG (drop au dela)	$0 (free)
Hosting	Render (MVP)	Services gratuits pour MVP (limites), upgrade ensuite	$0 (free instances)

Decisions critiques (a ne pas changer ensuite)

- **Source de verite**: la DB (Supabase Postgres).
- **Realtime**: events ephemeres via Broadcast + presence; events persistants via DB (messages, pipeline events).
- **Jobs**: BullMQ orchestre; Redis = transport et etat job; AI = exécution.
- **Mobile**: desktop-first maintenant, mais layouts Tailwind compatibles mobile (base + md/lg).

2. Prod Setup (Next.js + services) - production-ready des maintenant

Objectif: appliquer les bases de securite, perf et configuration avant d'empiler des features.

2.1 Environnements et secrets

- 3 environnements: local, staging, production.
- Aucun secret dans le code. Tout passe par variables d'environnement.
- Cote web: seules les variables prefixees NEXT_PUBLIC_* sont exposees au navigateur.

```
# Files
.env.local
.env.staging
.env.production

# Minimal env set
NEXT_PUBLIC_APP_URL=
NEXT_PUBLIC_API_URL=
DATABASE_URL= # server-only
SUPABASE_URL= # server-only
SUPABASE_ANON_KEY= # client use ok (public)
SUPABASE_SERVICE_ROLE_KEY= # server-only
UPSTASH_REDIS_REST_URL= # server-only
UPSTASH_REDIS_REST_TOKEN= # server-only
SENTRY_DSN=
```

2.2 Build, headers, et hygiene

- Activer ESLint + TypeScript strict.
- Configurer des headers securite (CSP, X-Frame-Options, etc.) selon les besoins.
- Ne jamais faire de scraping depuis le navigateur (toujours via worker/AI).
- Limiter la surface publique: seules les routes web; API derriere auth.

2.3 Deployment minimal (MVP)

- Web: Render Web Service (ou Vercel si tu veux).
- API + Worker + AI: Render Web Services separes (un par service) ou un seul multi-process si tu veux simplifier.
- Supabase: projet (staging) + projet (prod).
- Upstash: une DB Redis par environnement.

3. Realtime Spec (rooms, events, idempotence)

Regle: un event doit etre **idempotent**. Le client ne doit jamais 'douter' si un event est plus recent.

3.1 Rooms (naming fixe)

- room:workspace:{workspaceId}
- room:project:{projectId}
- room:lead:{leadId}
- room:channel:{channelId}

3.2 Event envelope (format unique)

```
EventEnvelope = {
type: string,
traceId: string,
actorId?: string,
workspaceId: string,
entityId: string,
entityVersion: number, # or updatedAt ISO
payload: object,
emittedAt: ISODate
}
```

3.3 Events minimum (MVP)

- **lead.updated** {leadId, status, updatedAt, version}
- **lead.job.status** {leadId, job, state, progress, updatedAt}
- **pipeline.moved** {leadId, from, to, version, updatedAt}
- **chat.message.new** {channelId, messageId, createdAt}
- **presence.update** {scope, users[]} (ephemere)
- **typing.start/stop** {channelId, userId} (ephemere)

3.4 Persistance vs ephemere

- Persistant (DB): messages, pipeline events, lead status, payouts, ledger.
- Ephemere (realtime): typing, presence, cursors, notifications transient.

3.5 Anti-chaos (regles)

- Le client ignore un event si entityVersion <= currentVersion.
- Chaque write API doit etre transaction DB puis emission event (apres commit).
- Tous les messages chat sont stockes en DB; le realtime ne sert qu'a notifier.

4. Observability (minimum vital)

Sans observabilite: tu vas debug a l'aveugle. On instrumente des le debut.

4.1 Logs structures (JSON)

- Un seul format: JSON logs partout (web, api, worker, ai).
- Champs obligatoires: traceId, leadId, workspaceId, userId, jobName, durationMs, status, errorCode.

```
log.info({
traceId, workspaceId, leadId, userId,
jobName: "ai_analysis",
durationMs: 5234,
status: "ok"
})
```

4.2 Sentry (erreurs + tracing + budget)

- Activer Sentry sur web + api + ai.
- Tagger chaque event avec traceId/leadId/workspaceId.
- Configurer un budget PAYG pour eviter facture surprise; au-dela, les events sont dropped.

4.3 5 metriques a suivre (dashboard Health)

- **P95 LeadDrop -> READY** (min).
- **Taux d'echec jobs** = failed / total.
- **P95 latence API** (create lead, get war room, post message).
- **Queue depth** (jobs en attente par queue).
- **P95 temps generation PDF**.

4.4 Alerting (simple)

- Alerte si P95 LeadDrop->READY > 10 min.
- Alerte si job failure rate > 5% sur 1h.
- Alerte si API P95 > 1.2s sur endpoints critiques.

5. Runbook - demarrer et verifier (checklist)

But: 15 minutes pour boot un environnement local complet et verifier que tout marche.

5.1 Boot local

```
# Infra
docker compose up -d

# Node deps
pnpm install

# DB
pnpm --filter api prisma:generate
pnpm --filter api prisma:migrate

# AI deps
cd apps/ai
python -m venv .venv
source .venv/bin/activate # Windows: .venv\Scripts\activate
pip install -r requirements.txt

# Run
pnpm dev
```

5.2 Smoke tests

- API: GET /health -> 200 (db+redis ok).
- Web: page dashboard charge en < 2s.
- Lead Drop: POST /leads cree un lead + job en queue.
- Realtime: rejoindre room:lead:{id} et recevoir lead.job.status.
- War Room: status READY + PDF disponible (signed URL).
- Chat: POST message -> event chat.message.new recu sur autre client.