APPRENDRE · THE STACK

Cloudflare Pages + Workers : la stack statique-mais-dynamique

Cloudflare Pages gère la surface statique. Cloudflare Workers gèrent la petite surface dynamique — paiements, formulaires, webhooks. Ensemble, ils coûtent presque rien, servent depuis 330+ villes en bord de réseau, et ne demandent aucune infrastructure backend à administrer.

---

11 min de lecture

Dernière mise à jour: 10 juin 2026

> *Un site « contenu d'abord » n'a pas besoin d'un serveur Node permanent ni d'une base de données : 99 % du trafic peut être servi en HTML statique depuis le edge, et le 1 % restant — paiements, soumissions de formulaires, redirections — tient dans un ou deux Workers JavaScript de quelques dizaines de lignes.*

La thèse

- Pourquoi le modèle statique-mais-dynamique est économiquement et techniquement supérieur à un monolithe SSR pour les sites à fort contenu
- Le framework du Sandwich statique-mais-dynamique : comment penser la séparation entre le pain (HTML pré-rendu) et la garniture (logique éphémère)
- Les commandes `wrangler` exactes pour passer d'un dossier `./build` à un déploiement mondial en moins de cinq minutes
- Comment Cloudflare se compare à Vercel, Netlify et AWS S3+CloudFront sur le coût, la latence et l'ergonomie
- Quand vous devez quitter cette stack et passer à quelque chose de plus lourd (et pourquoi c'est plus rare que vous ne le pensez)

01 — Le framework : le Sandwich statique-mais-

dynamique

The Static-but-Dynamic Sandwich

La plupart des décisions d'architecture web partent d'une fausse dichotomie : « statique ou dynamique ». La réalité est que presque tous les sites sont à 99 % statiques et à 1 % dynamiques. Le Sandwich statique-mais-dynamique modélise cette répartition explicitement : une couche épaisse de HTML pré-rendu (le pain), une fine couche de logique éphémère exécutée à la demande (la garniture), et zéro couche persistante en dessous (pas de base de données, pas de serveur Node permanent, pas de VM à surveiller).

1

Le pain : HTML statique pré-construit

Chaque page de votre site — article, fiche produit, landing, FAQ — est générée à l'avance pendant le build et stockée comme fichier `.html` sur Cloudflare Pages. Le contenu est servi en quelques millisecondes depuis le nœud edge le plus proche, sans qu'aucun code applicatif ne s'exécute. C'est le modèle le plus rapide, le moins cher et le plus sûr qui existe pour le contenu.

2

La garniture : Workers pour la logique éphémère

Quand un visiteur clique sur « Acheter », soumet un formulaire ou reçoit un webhook tiers, un Cloudflare Worker s'éveille en moins d'une milliseconde, exécute quelques dizaines de lignes de JavaScript, et s'éteint. Il n'y a ni serveur permanent ni cold start — les V8 isolates démarrent instantanément. Un seul Worker peut gérer le paiement, un autre la redirection, un autre encore l'A/B test.

3

Pas de couche du bas : aucune base de données persistante par défaut

Le Sandwich classique n'a pas de troisième couche. Pas de PostgreSQL à répliquer, pas de Redis à redimensionner, pas de RDS à sauvegarder. Si vous avez besoin d'état, vous le déléguez à un service tiers spécialisé (Stripe pour le paiement, Formspree pour les soumissions, un CMS headless pour le contenu). Cela élimine une catégorie entière d'opérations — et de pannes.

4

Le bord (edge) comme tissu de déploiement

Pages et Workers s'exécutent tous deux sur le même réseau : 330+ villes dans 100+ pays, ~50 ms de latence pour environ 95 % des internautes. Vous ne gérez pas de régions, vous ne choisissez pas *us-east-1*. Le même artefact est répliqué mondialement et servi depuis le nœud le plus proche de chaque visiteur.

5

Git comme source de vérité du déploiement

Vous ne déployez pas en SSH-ant sur un serveur. Vous déployez en faisant un `git push` sur la branche de production, ou en lançant `wrangler pages deploy ./build` depuis une CI. L'état de la production est exactement égal au contenu d'un commit Git. Cela rend les rollbacks triviaux et l'audit linéaire.

02 — Les données.

330+

villes Cloudflare dans 100+ pays servant Pages et Workers depuis le même réseau edge

CLOUDFLARE 2024

~50 ms

latence médiane vers environ 95 % des internautes depuis n'importe quel nœud Cloudflare

CLOUDFLARE 2024

0 $

coût mensuel d'entrée : bande passante illimitée, 500 builds/mois et domaines personnalisés gratuits sur le plan free de Pages

CLOUDFLARE PAGES PRICING 2024

10 ms

CPU par requête inclus dans le plan free Workers — largement suffisant pour une redirection Stripe ou un proxy de formulaire

CLOUDFLARE WORKERS PRICING 2024

14 000+

fichiers HTML déployés par Pillar sur Cloudflare Pages pour servir son portefeuille de plus de 100 000 domaines

PILLAR 2024

1

Worker en production chez Pillar — un helper de redirection vers les Stripe Payment Links

PILLAR 2024

Pourquoi cette architecture gagne sur le SSR pour les sites de contenu

Le rendu côté serveur (SSR) a été vendu pendant des années comme la solution moderne au SEO et à la performance. En pratique, pour un site à fort contenu — landing pages, articles, fiches produits — un HTML statique pré-rendu bat un SSR sur presque toutes les dimensions mesurées. Le HTML statique est lui-même déjà un rendu côté serveur : simplement, ce rendu s'est produit au moment du build plutôt qu'au moment de la requête. Google indexe le HTML statique aussi bien que le HTML SSR — mieux, même, parce qu'il est plus rapide à servir et donc mieux noté sur les Core Web Vitals.

Le coût opérationnel est l'autre différence décisive. Un site SSR sur Vercel ou Heroku doit maintenir un processus Node tourner en permanence, scaler horizontalement aux pics de trafic, payer la bande passante à mesure que l'audience croît. Un site statique sur Cloudflare Pages n'a aucun de ces coûts : le HTML est cache-able indéfiniment au edge, la bande passante est illimitée sur le plan gratuit, et il n'y a aucun processus à surveiller. Le coût marginal d'une page supplémentaire est nul.

La contrepartie est que toute véritable logique dynamique — afficher le nom de l'utilisateur connecté, lister son panier, calculer un prix personnalisé — doit être déléguée : soit à du JavaScript côté client qui appelle une API, soit à un Worker invoqué sur un endpoint spécifique. Pour la plupart des sites à vocation de contenu, ces moments dynamiques sont rares et isolés. Le Sandwich statique-mais-dynamique modélise exactement cette réalité.

Comparaison concrète : Cloudflare vs Vercel vs Netlify vs AWS

S3+CloudFront

Vercel et Netlify offrent une expérience de développement excellente, mais leur modèle tarifaire devient agressif à mesure que vous gagnez en audience. Vercel facture à partir de 20 $/mois le plan Pro et ajoute des surcoûts de bande passante au-delà des limites incluses. Netlify facture les minutes de build, ce qui devient sensible dès qu'un site a quelques centaines de pages générées dynamiquement. Pour un projet personnel ou un MVP, ces plans sont raisonnables ; pour un site qui déploie plus de 10 000 pages comme celui de Pillar, ils deviennent prohibitifs.

AWS S3 + CloudFront reste l'option historique « cloud lourd ». C'est techniquement solide, mais l'ergonomie est franchement hostile : vous configurez manuellement le bucket, vous attachez une distribution CloudFront, vous gérez les invalidations de cache, vous ajoutez un certificat ACM, vous écrivez une fonction Lambda@Edge si vous avez besoin de logique. Vous êtes responsable de chaque petit cran — ce qui est exactement le coût caché qu'un développeur ne veut pas porter sur dix années.

Cloudflare Pages occupe le milieu : le sucré ergonomique de Vercel sans l'extension tarifaire, la robustesse mondiale d'AWS sans la configuration. Le plan gratuit est genuinement utilisable en production. Les Workers s'intègrent en quelques lignes : un fichier JavaScript, une route, un déploiement. Le même CLI `wrangler` gère Pages, Workers, KV, R2, D1 — toute la pile reste cohérente.

Anatomie d'un Worker minimal pour la logique dynamique

Un Worker Cloudflare est, dans sa forme la plus simple, un fichier JavaScript qui exporte un objet avec une méthode `fetch`. Cette méthode reçoit une `Request` et renvoie une `Response`. C'est tout : pas de framework requis, pas de routeur obligatoire, pas de dossier `node_modules` à bundler. Pour un cas d'usage typique — rediriger l'utilisateur vers un Stripe Payment Link en fonction d'un slug de produit — le Worker complet tient en une vingtaine de lignes : lire l'URL entrante, extraire le slug, mapper le slug vers un `https://buy.stripe.com/<ID>` via un objet, renvoyer une `Response.redirect`.

Le modèle d'exécution diffère fondamentalement d'AWS Lambda. Lambda enveloppe chaque fonction dans un conteneur, ce qui entraîne des cold starts de 200 ms à 2 s sur Node. Les Workers utilisent les V8 isolates — le même mécanisme que les onglets de Chrome — et démarrent en moins d'une milliseconde. Cela élimine toute une catégorie de problèmes : vous n'avez pas besoin de garder le Worker « chaud », pas besoin d'optimiser le startup, pas besoin de provisionner des concurrences. Le Worker s'exécute, répond, disparaît.

Le plan gratuit Workers inclut 100 000 requêtes/jour et 10 ms de CPU par requête. Pour une redirection Stripe ou un proxy de formulaire vers Formspree, 10 ms sont largement suffisants : la logique réelle consomme typiquement 1 à 2 ms. Une fois sur le plan payé (5 $/mois minimum), vous obtenez 10 millions de requêtes incluses et 50 ms par requête. Pour la plupart des sites à vocation de contenu, vous resterez sur le free tier indéfiniment.

Le cas Pillar : 14 000+ pages, 1 Worker, ~0 $/mois

Pillar exploite un portefeuille de plus de 100 000 domaines premium. Cela se traduit, côté site public, par plus de 14 000 pages HTML statiques déployées sur Cloudflare Pages : fiches de domaine, pages catégorie, articles de la Learn Library, landings produit. Toutes ces pages sont générées au moment du build et servies depuis le edge sans qu'aucun code applicatif ne s'exécute à la requête.

La couche dynamique tient dans un seul Worker : un helper de redirection qui, étant donné un slug de domaine, renvoie vers le Stripe Payment Link correspondant. Quand un visiteur clique sur « Acheter ce domaine », le Worker s'éveille en moins d'une milliseconde, fait un lookup en mémoire, et renvoie une réponse 302 vers `https://buy.stripe.com/<ID>`. Stripe gère alors le paiement, le reçu, la sécurité. Aucune donnée de carte ne transite par notre infrastructure. C'est l'application directe de notre Zero-Secret Architecture (/learn/en/the-stack/zero-secret-architecture/).

L'empreinte opérationnelle est minimale : pas de serveur à patcher, pas de base de données à sauvegarder, pas d'astreinte à tenir. Le coût Cloudflare mensuel pour servir cette infrastructure mondialement est essentiellement nul. Le même argent qui aurait payé un cluster Kubernetes finance directement l'acquisition de domaines — ce qui est la vraie source de valeur de l'entreprise.

03 — Regardez : une démonstration réelle

04 — Checklist tactique : déployer votre premier

Sandwich

Suivez cette séquence pour passer d'un dossier `./build` à un site mondial avec une logique dynamique en moins d'une heure. Aucune carte de crédit requise pour le plan gratuit.

1. Installez le CLI `wrangler` via `npm install -g wrangler`, puis authentifiez-vous avec `wrangler login` (ouvre un onglet navigateur vers votre compte Cloudflare).
2. Créez le projet Pages avec `wrangler pages project create <YOUR_PROJECT>`. Cela réserve un sous-domaine `<YOUR_PROJECT>.pages.dev` immédiatement utilisable.
3. Déployez votre dossier de build avec `wrangler pages deploy ./build --project-name=<YOUR_PROJECT>`. Le déploiement prend typiquement 10 à 30 secondes et propage mondialement.
4. Ajoutez votre domaine personnalisé via le dashboard Cloudflare Pages : section « Custom domains », entrez `www.exemple.com`. Si votre DNS est déjà chez Cloudflare, le certificat TLS est provisionné automatiquement et gratuitement.
5. Écrivez votre premier Worker : créez `worker.js` avec un `export default { async fetch(request) { ... } }`, puis déployez avec `wrangler deploy`. Bindez-le à une route comme `api.exemple.com/*` via le dashboard ou via `wrangler.toml`.
6. Pour les formulaires, pointez le `action` vers un endpoint Formspree (ou un Worker proxy qui appelle Formspree). Pour les paiements, redirigez vers un Stripe Payment Link — jamais d'API key côté client.
7. Vérifiez la couverture edge avec `curl -I https://www.exemple.com` depuis plusieurs régions (ou utilisez un service comme `cf-trace`) pour confirmer que le contenu est servi depuis le POP le plus proche.

Questions fréquentes.

Et si j'ai vraiment besoin de SSR pour la personnalisation par utilisateur ?

La plupart des cas qu'on croit nécessiter du SSR sont en réalité résolvables par du HTML statique plus du JavaScript côté client qui appelle un Worker. Pour afficher « Bonjour [prénom] » en haut de page, vous chargez la page statique, puis un `fetch` client vers un Worker injecte le prénom. Si vous avez une véritable contrainte SSR — SEO sur du contenu strictement personnalisé et indexable — Cloudflare Pages supporte aussi les frameworks SSR (Next.js, SvelteKit) via les Pages Functions, qui sont des Workers déguisés. Le modèle mérite quand même d'être questionné en premier.

Quelles sont les limites réelles du plan gratuit ?

Pages : bande passante illimitée, 500 builds/mois, 100 déploiements custom/jour, 20 000 fichiers maximum par déploiement, taille de fichier max 25 MiB. Workers : 100 000 requêtes/jour, 10 ms CPU par requête, 1 MiB de script. Pour la grande majorité des sites de contenu, vous resterez largement sous ces seuils. Le passage au plan payé Workers (5 $/mois) ouvre 10 millions de requêtes et 50 ms CPU par requête — c'est encore très peu cher comparé à une instance EC2 ou un dyno Heroku.

Comment je gère les variables d'environnement et les secrets dans les Workers ?

Les variables non sensibles vont dans `wrangler.toml` sous `[vars]`. Les secrets utilisent `wrangler secret put <NAME>`, qui les stocke chiffrés côté Cloudflare et les expose au Worker via `env.<NAME>`. Ne jamais commiter de secrets dans le repo. Cela dit, l'esprit du Sandwich statique-mais-dynamique est de minimiser les secrets au maximum : si votre Worker se contente de rediriger vers Stripe ou Formspree, il n'a même pas besoin de clé API — les URL de redirection sont publiques par construction.

Et la base de données ? Vraiment aucune ?

Pour un site à vocation de contenu, oui, vraiment aucune en propre. Le contenu vit en Markdown ou JSON dans le repo Git et est compilé en HTML au build. L'état utilisateur (paiement, soumission) est délégué à un service spécialisé qui gère sa propre persistance (Stripe, Formspree, Auth0). Si vous avez vraiment besoin d'une base — par exemple pour stocker des compteurs ou des sessions — Cloudflare propose KV (key-value), D1 (SQLite distribué) et R2 (stockage objet) accessibles depuis les Workers. Mais commencez sans, et n'ajoutez une couche persistante que quand un besoin réel l'exige.

Quand devrais-je quitter cette stack pour autre chose ?

Quand votre logique applicative devient réellement complexe et état-ful : traitement par batch de plusieurs minutes, jobs en file d'attente persistante, calculs lourds en ML, jointures SQL multi-tables sur des millions de lignes à chaque requête. À ce stade, vous voulez probablement un backend dédié (Fly.io, Render, Railway, ou un cluster Kubernetes si l'équipe le justifie). Mais même alors, gardez Cloudflare Pages pour le frontend statique : le Sandwich devient simplement un Sandwich avec une couche backend, et vous gagnez encore la rapidité edge sur le contenu public.
