APRENDA · THE STACK

Cloudflare Pages + Workers: O Stack Estático-mas-Dinâmico

Pages cuida da superfície estática. Workers cuidam da pequena superfície dinâmica. Juntos, eles custam quase nada, servem de 330+ cidades de edge e não exigem infraestrutura de backend. Esta é a arquitetura por trás de sites de conteúdo que escalam sem dor.

---

9 min de leitura

Última atualização: 10 de junho de 2026

> *A maioria dos sites de conteúdo não precisa de um servidor. Precisa de HTML estático cacheado no edge para 99% do tráfego, e de uma fina camada de lógica dinâmica (pagamentos, formulários, webhooks) para o 1% restante. Cloudflare Pages + Workers entrega exatamente isso, sem reinventar AWS.*

A tese

- Por que separar a superfície estática da superfície dinâmica reduz custo e complexidade em uma ordem de magnitude
- Como o modelo de V8 isolates do Workers elimina cold starts que afligem Lambda e Vercel Functions
- O fluxo concreto de deploy: `wrangler login`, `wrangler pages project create`, `wrangler pages deploy`
- Quando escolher Cloudflare em vez de Vercel, Netlify ou AWS S3+CloudFront — e quando não escolher
- Como Pillar serve 14.000+ páginas HTML com um único Worker para redirecionamento de pagamento

01 — O framework: O Sanduíche Estático-mas-

Dinâmico

The Static-but-Dynamic Sandwich

A maioria dos times de engenharia comete o mesmo erro ao construir sites de conteúdo: eles assumem que precisam de um servidor porque *algumas* coisas são dinâmicas. Então instalam Next.js em modo SSR, alugam um servidor Node, configuram um banco de dados e pagam por capacidade ociosa 24/7. O Sanduíche Estático-mas-Dinâmico inverte essa lógica: a base e o topo são HTML estático servidos do edge, e o recheio — fino, específico, sob demanda — são Workers. Você paga apenas pelo recheio, e o recheio é mensurado em milissegundos.

1

Pão de baixo: HTML pré-gerado

Todo o seu conteúdo (artigos, páginas de produto, landing pages) é gerado em build time como arquivos `.html` estáticos. Sem renderização em runtime, sem chamadas a banco de dados por requisição. O Cloudflare Pages distribui esses arquivos para 330+ cidades de edge, e o navegador do usuário recebe HTML pronto em ~50ms.

2

Recheio: Workers para o dinâmico

Para o que precisa ser dinâmico — processar um formulário, redirecionar para um Stripe Payment Link, validar um webhook — você escreve um arquivo `.js` com um `fetch` handler. O Worker roda em um V8 isolate, sem cold start, em qualquer um dos 330+ POPs. Você não gerencia servidor; você gerencia uma função.

3

Pão de cima: CDN nativa

Como Pages e Workers vivem dentro da mesma rede da Cloudflare, você não paga por CDN separada, não configura origem, não lida com SSL manualmente. O certificado TLS é emitido automaticamente. A cache do edge é padrão. O DNS aponta direto para a infraestrutura da Cloudflare.

4

Roteamento: domínio + paths

Você configura uma rota como `/api/checkout/*` apontando para o Worker, e tudo mais cai no Pages. O roteamento acontece no edge, sem proxy reverso, sem balanceador de carga. É uma linha em um arquivo `wrangler.toml`.

5

Preço: zero ou quase zero

O tier gratuito do Pages oferece bandwidth ilimitado, 500 builds por mês, domínios customizados gratuitos com SSL. O tier gratuito do Workers permite 100.000 requisições por dia. Para um site de conteúdo com tráfego significativo, o custo mensal geralmente permanece em USD 0 ou abaixo de USD 5, comparado a USD 20+ no Vercel Pro.

02 — Os dados.

330+

cidades de edge na rede Cloudflare

CLOUDFLARE 2024

~50ms

latência para ~95% dos usuários da internet

CLOUDFLARE NETWORK STATISTICS 2024

14.000+

arquivos HTML que Pillar serve via Cloudflare Pages

PILLAR DEPLOYMENT METRICS

10ms

CPU por requisição no tier gratuito do Workers

CLOUDFLARE WORKERS PRICING 2024

500

builds por mês gratuitos no Cloudflare Pages

CLOUDFLARE PAGES LIMITS 2024

USD 0

custo inicial para começar a hospedar um site

CLOUDFLARE FREE TIER 2024

Por que estático vence em SEO (e por que SSR é teatro)

Existe um mito persistente de que sites modernos precisam de Server-Side Rendering (SSR) para SEO. Esse mito vendeu muitas licenças Vercel. A realidade técnica: o Googlebot prefere HTML pronto, completo, estável. HTML pré-gerado — o tipo que sai de um build estático — é *indistinguível* de HTML renderizado por servidor do ponto de vista do crawler, exceto em uma dimensão: é mais rápido, mais consistente, e nunca falha por timeout de banco de dados.

Quando você serve 14.000 páginas HTML estáticas do edge da Cloudflare, cada uma é cacheada em 330+ POPs ao redor do mundo. O Googlebot, que ele mesmo rastreia distribuído globalmente, recebe a página do POP mais próximo. O Time to First Byte (TTFB) é consistentemente abaixo de 100ms. Isso não é um detalhe estético — o Core Web Vitals do Google penaliza TTFB alto, e SSR introduz latência que static não introduz.

A pergunta correta não é *'devo usar SSR para SEO?'*, é *'meu conteúdo muda a cada requisição?'*. Se a resposta é não (e para sites de conteúdo, quase sempre é não), SSR é complexidade injetada por hábito, não por necessidade. Pillar publica artigos do Learn Library (/learn/en/the-stack/) como HTML estático, e os mesmos artigos aparecem no índice do Google dentro de horas.

A anatomia de um Worker: um arquivo, um fetch handler

Um Worker não é um servidor. Não é um container. É uma função JavaScript que recebe um objeto `Request` e retorna um objeto `Response`. O runtime é V8 isolate — o mesmo motor do Chrome — e isolates iniciam em microssegundos, não em segundos. Por isso não existe cold start. Compare com AWS Lambda, onde um cold start em Node pode levar 200-800ms; um Worker simplesmente não tem esse problema arquitetônico.

O exemplo mínimo: você cria um arquivo `worker.js` com `export default { async fetch(request) { return new Response('hello') } }`. Você roda `wrangler deploy`. Em 30 segundos, esse Worker está rodando em 330+ cidades, com SSL, DDoS protection e uma URL pública. Não existe etapa intermediária de provisionamento, não existe arquivo de configuração YAML de 200 linhas.

Pillar usa exatamente um Worker em produção: um helper de redirecionamento para Stripe Payment Links. Quando alguém clica em 'Comprar' em uma página estática de produto, a requisição vai para o Worker, que adiciona contexto (UTM tags, ID do produto), e responde com um `Response.redirect()` para o Payment Link correto do Stripe. Toda a lógica dinâmica de e-commerce, em ~30 linhas de JavaScript, custando essencialmente USD 0 por mês.

O fluxo de deploy: do `git push` ao edge global

Existem duas formas de fazer deploy no Cloudflare Pages, e ambas são produtivas. A primeira é via integração Git: você conecta o repositório GitHub no dashboard da Cloudflare, define o comando de build (`npm run build` ou similar), e cada `git push` dispara um build e um deploy automático. Branches viram preview URLs; `main` vira produção.

A segunda forma é via Wrangler CLI, que é o caminho mais rápido para times que querem controle explícito. Você instala com `npm install -g wrangler`, autentica com `wrangler login` (abre o browser para OAuth), cria o projeto com `wrangler pages project create <YOUR_PROJECT>`, e faz deploy com `wrangler pages deploy ./build`. Em segundos, o conteúdo está ao vivo em uma URL como `<YOUR_PROJECT>.pages.dev`.

Adicionar um domínio customizado é igualmente direto: você vai no dashboard, adiciona o domínio, e a Cloudflare configura o SSL automaticamente via Let's Encrypt. Se o DNS já estiver na Cloudflare, não há configuração adicional. Se estiver em outro registrador (Dynadot, Namecheap, GoDaddy), você aponta um CNAME e a Cloudflare cuida do resto. Veja The Stack (/learn/en/the-stack/) para a visão completa da arquitetura.

Cloudflare versus Vercel, Netlify e AWS: a economia honesta

Vercel é um produto excelente, mas tem um modelo de preço que pune o sucesso. O tier gratuito tem limites estreitos de bandwidth (~100GB/mês); ao passar disso, você cai no Pro a USD 20/mês, e bandwidth adicional é cobrada por GB. Para um site de conteúdo que cresce, isso vira uma conta que sobe com o tráfego — exatamente o oposto do que você quer enquanto escala orgânicamente.

Netlify segue padrão similar: tier gratuito generoso para hobbies, mas o tier comercial cobra por build minutes (problema chato quando você tem 14.000 páginas para gerar) e por bandwidth. AWS S3+CloudFront é tecnicamente competitivo em preço, mas a configuração exige conhecimento de bucket policies, CloudFront distributions, Route 53, ACM certificates — horas de trabalho para algo que Cloudflare faz em cliques.

Cloudflare Pages oferece bandwidth ilimitado no tier gratuito. Não existe overage. Workers cobram USD 5/mês após ultrapassar 100.000 requisições/dia, com custo marginal previsível. Para o caso de uso de Pillar — 14.000+ arquivos HTML mais um Worker de pagamento — o custo total é literalmente USD 0/mês. A mesma infraestrutura em Vercel custaria USD 20-60/mês. A diferença não é trivial quando você gerencia vários sites no portfólio Pillar.

03 — Assista: um percurso real

04 — Checklist tático: setup completo em 30 minutos

Se você tem um site estático (gerado por qualquer SSG — Astro, Hugo, Eleventy, scripts customizados) e quer movê-lo para Cloudflare Pages + Workers, este é o caminho mínimo. Substitua `<YOUR_PROJECT>` pelo nome real do seu projeto e `<YOUR_DOMAIN>` pelo seu domínio.

1. Instale o Wrangler CLI globalmente: `npm install -g wrangler`. Verifique a versão com `wrangler --version` para confirmar que está em uma versão recente (3.x ou superior).
2. Autentique com sua conta Cloudflare: `wrangler login`. Isso abre o browser para OAuth e armazena credenciais localmente. Você não precisa criar nem gerenciar tokens manualmente para deploys locais.
3. Crie o projeto Pages: `wrangler pages project create <YOUR_PROJECT>`. Isso registra o projeto no dashboard da Cloudflare e gera o subdomínio `<YOUR_PROJECT>.pages.dev` automaticamente.
4. Faça o primeiro deploy do diretório de build: `wrangler pages deploy ./build --project-name=<YOUR_PROJECT>`. O Wrangler faz upload incremental — arquivos não alterados não são re-enviados, o que torna deploys subsequentes muito rápidos mesmo com milhares de páginas.
5. Conecte um domínio customizado pelo dashboard: Pages → seu projeto → Custom domains → Set up a domain. Se o DNS já está na Cloudflare, é um clique. Se não, você recebe o CNAME para apontar no seu registrador.
6. Para a camada dinâmica, crie um arquivo `worker.js` com a estrutura `export default { async fetch(request, env, ctx) { /* sua lógica */ return new Response(...) } }` e faça deploy com `wrangler deploy worker.js`. Configure a rota no `wrangler.toml` apontando, por exemplo, `<YOUR_DOMAIN>/api/*` para o Worker.
7. Adicione um arquivo `_headers` na raiz do build para controlar cache: defina `Cache-Control: public, max-age=3600` para HTML e `max-age=31536000, immutable` para assets com hash. Isso aumenta a hit rate da cache do edge e reduz custos de origin.

Perguntas frequentes.

E se eu precisar de um banco de dados? Cloudflare Pages funciona com Postgres?

Sim, mas a pergunta certa vem antes: você *realmente* precisa de banco de dados? Para sites de conteúdo, quase nunca — o conteúdo vive em arquivos Markdown no Git, não em linhas de banco. Quando você precisa de persistência (formulários, contagem de leitura), as opções nativas da Cloudflare são D1 (SQLite serverless), KV (key-value de baixa latência) ou Durable Objects (estado consistente). Para casos onde você quer Postgres gerenciado externo, Workers conectam via Hyperdrive ou diretamente a Neon, Supabase ou qualquer Postgres acessível via HTTP.

Posso usar Next.js com Cloudflare Pages?

Sim, mas com cuidado. Next.js em modo estático (`output: 'export'`) funciona perfeitamente — você gera HTML estático e faz deploy. Next.js em modo SSR/ISR funciona via adapter (Cloudflare Workers runtime), mas você perde algumas features e adiciona complexidade. A recomendação pragmática: se você está começando do zero e tem flexibilidade, considere Astro, Hugo ou Eleventy, que são nativamente estáticos e não exigem adapter.

Como faço processamento de formulários sem servidor?

Duas opções boas. Opção 1: use Formspree, Basin ou similar — o `action` do `<form>` aponta para um endpoint externo que recebe a submissão e te envia por email. Zero código. Opção 2: escreva um Worker que recebe `POST`, valida campos, e envia via email (Mailchannels é gratuito para Workers) ou armazena em KV. Opção 1 é mais rápida para MVP; opção 2 dá mais controle quando você tem escala.

Os 10ms de CPU por requisição do tier gratuito são suficientes?

Para a grande maioria dos casos de uso, sim — e é importante entender que 10ms de *CPU* não é 10ms de wall clock. Espera por I/O (chamadas HTTP, banco) não conta. Um Worker que recebe um `POST`, faz um `fetch` a uma API Stripe e retorna uma resposta facilmente fica abaixo de 1ms de CPU, porque o tempo dele é majoritariamente esperando o Stripe responder. Se você precisar de mais, o tier Paid (USD 5/mês) levanta o limite para 50ms de CPU e adiciona 10 milhões de requisições.

Como debugo um Worker em produção?

Wrangler oferece `wrangler tail`, que faz streaming dos logs em tempo real do Worker em produção para seu terminal. Você vê cada `console.log`, cada exception, cada requisição. Para debugging mais profundo, o dashboard tem uma seção de Logs e você pode habilitar Logpush para enviar logs estruturados para R2, S3 ou um SIEM. Para desenvolvimento local, `wrangler dev` roda o Worker no V8 isolate local com bindings reais aos serviços Cloudflare, então paridade dev/prod é alta.
