APRENDE · LA PILA

Cloudflare Pages + Workers: La pila estática-pero-dinámica

La mayoría de los sitios orientados a contenido son 95% estáticos y 5% dinámicos. Cloudflare Pages cubre la superficie estática con HTML cacheado en el borde; Cloudflare Workers cubre la pequeña superficie dinámica (pagos, formularios, webhooks) con isolates de V8. El resultado: servicio global a ~50ms para ~95% de los usuarios de internet, sin servidor que mantener.

---

10 min de lectura

Última actualización: 10 de junio de 2026

> *No necesitas un backend tradicional para ejecutar un sitio de contenido moderno a escala global. Necesitas HTML pre-renderizado en CDN y una capa de cómputo de borde minúscula para lo poco que es realmente dinámico.*

La tesis

- Por qué la arquitectura de "dos capas" (estática + dinámica) supera al renderizado del lado del servidor para sitios orientados a contenido
- Cómo Cloudflare Pages sirve HTML desde 330+ ciudades sin configuración de CDN
- Cuándo escribir un Worker en lugar de añadir un backend completo (y cuándo no)
- Cómo Pillar despliega 14,000+ archivos HTML con un solo Worker para la lógica de pagos
- Cómo se compara el costo y la operación contra Vercel, Netlify y AWS S3+CloudFront

01 — El marco: el sándwich estático-pero-

dinámico

The Static-but-Dynamic Sandwich

Piensa en tu sitio como un sándwich. El pan de arriba y el de abajo son estáticos: HTML pre-construido servido desde el borde. El relleno en el medio es dinámico: un pequeño núcleo de lógica que necesita ejecutarse por solicitud (un pago, un envío de formulario, una redirección A/B). El truco arquitectónico es mantener el relleno tan delgado como sea posible. Cuanto más grande sea el relleno, más te acercas a un monolito de servidor tradicional y pierdes las ventajas del borde.

1

1. La capa estática (pan)

HTML, CSS, JS, imágenes, fuentes — todo precompilado en tiempo de build y servido como archivos planos. Cloudflare Pages cachea esto en 330+ ciudades automáticamente. No hay base de datos que consultar, no hay servidor que despertar, no hay arranque en frío. La latencia se mide en decenas de milisegundos para casi cualquier usuario del planeta.

2

2. La capa dinámica (relleno)

Funciones pequeñas y enfocadas escritas como Cloudflare Workers. Cada Worker es un solo archivo JavaScript con un controlador `fetch`. Se ejecuta en un isolate de V8 (no en un contenedor), arranca en menos de 1ms y escala globalmente sin configuración. Cobra solo por solicitud, no por tiempo de inactividad.

3

3. El borde como sustrato

Pages y Workers viven en la misma red de Cloudflare. Esto significa que tu HTML estático y tu código dinámico están co-localizados en cada uno de los 330+ PoPs. Un usuario en Bogotá, Lagos o Yakarta toca el mismo núcleo: el más cercano. No hay región primaria. No hay replicación manual.

4

4. Servicios externos como dependencias delgadas

Cualquier cosa que no quepa en un Worker (pasarela de pagos, captura de formularios, envío de email) se delega a un servicio especializado: Stripe, Formspree, un proveedor SMTP. El Worker actúa como pegamento de un solo propósito, no como aplicación. Esto es lo que permite que la arquitectura permanezca sin secretos y sin estado en el lado de Cloudflare.

5

5. La regla del relleno delgado

Si un Worker comienza a crecer hacia 500+ líneas de lógica de negocio, es una señal de que necesitas separarlo en varios Workers, no convertirlo en un monolito. La frontera del sándwich es deliberada: HTML estático arriba, lógica delgada en medio, servicios externos abajo. Cada capa es reemplazable independientemente.

02 — Los datos.

330+

Ciudades en la red de borde de Cloudflare a las que Pages y Workers se despliegan automáticamente

CLOUDFLARE 2024 QUARTERLY

~50ms

Latencia típica a ~95% de los usuarios de internet desde el PoP de Cloudflare más cercano

CLOUDFLARE 2024 QUARTERLY

14,000+

Archivos HTML que Pillar despliega a Cloudflare Pages como capa estática

PILLAR INTERNAL, 2026

1

Worker que Pillar opera para toda su superficie dinámica (redirección a Stripe Payment Link)

PILLAR INTERNAL, 2026

10ms	$0
Límite de CPU por solicitud en el plan gratuito de Workers, suficiente para la mayoría de la lógica de pegamento	Costo de inicio para Pages + Workers con dominio personalizado, ancho de banda ilimitado y certificado TLS automático
CLOUDFLARE 2024 QUARTERLY	CLOUDFLARE 2024 QUARTERLY

Por qué el HTML estático gana para sitios de contenido

Existe un mito persistente de que el renderizado del lado del servidor (SSR) es necesario para un buen SEO. No lo es. Googlebot rastrea HTML pre-renderizado más rápido y de manera más confiable que cualquier respuesta SSR, porque no hay esperas de TTFB, no hay riesgo de timeouts, no hay vértigo de hidratación. Un archivo HTML servido desde un PoP de borde a 30ms es la señal de rendimiento más fuerte que puedes enviar a un buscador.

El segundo argumento contra lo estático es la "personalización". En la práctica, los sitios orientados a contenido rara vez personalizan el *contenido del documento*; personalizan widgets accesorios (un saludo, un carrito, una bandera regional). Esos widgets pueden hidratarse en el lado del cliente después de la pintura inicial, o renderizarse en un Worker, sin tocar el HTML cacheado. El documento canon permanece cacheable.

El tercer argumento es la "experiencia de desarrollador". Aquí el panorama ha cambiado: generadores de sitios estáticos modernos (Astro, Eleventy, Hugo, Next.js en modo de exportación estática) compilan miles de páginas en segundos. La construcción de Pillar de 14,000+ archivos HTML se ejecuta en menos de dos minutos. No hay penalización de DX por elegir estático cuando los datos viven en archivos locales o JSON.

El Worker como pegamento, no como aplicación

La tentación con cualquier herramienta de cómputo serverless es construir gradualmente una aplicación dentro de ella. Empiezas con una función de redirección, añades validación, añades autenticación, añades una capa de base de datos, y de repente has reinventado Express.js dentro de un isolate de V8. Resiste esto. Un Worker debe ser pegamento: traduce de la solicitud entrante a una llamada de servicio externo, o aplica una transformación trivial (firma una URL, establece una cookie, dirige a Stripe).

Pillar tiene un solo Worker. Su trabajo: recibir una solicitud al endpoint de checkout, mirar qué dominio está comprando el usuario y emitir una redirección HTTP 302 al Stripe Payment Link correcto. Eso es. Sin sesión, sin base de datos, sin secretos en el código del Worker (los IDs de Stripe Payment Link son referencias públicas, no credenciales). El Worker es esencialmente una tabla de búsqueda con conciencia HTTP.

Esta restricción —mantén el Worker como pegamento— es lo que mantiene la arquitectura comprensible. Cuando un nuevo desarrollador se incorpora, no hay un grafo de servicios que aprender. Hay HTML estático, y hay un Worker pequeño. Cualquier lógica de pago real vive en Stripe, donde pertenece. Cualquier captura de formulario vive en Formspree. Cualquier envío de email vive en un proveedor SMTP. El sustrato de Cloudflare no es donde reside el estado; es donde se enrutan las solicitudes.

Cómo se compara contra Vercel, Netlify y AWS

Vercel se construyó alrededor de Next.js y su modelo de precios refleja eso: vas a pagar por funciones serverless, por ancho de banda y por minutos de build una vez que el tráfico crece. Para sitios mayormente estáticos, estás pagando por capacidades de SSR que no usas. El plan Pro comienza alrededor de $20/mes por miembro del equipo y los excedentes de ancho de banda pueden volverse significativos con el aumento del tráfico orgánico.

Netlify ofrece una historia estática más limpia pero cobra agresivamente por minutos de build, lo cual duele cuando tienes una capa estática grande que se reconstruye a menudo. La integración de funciones es funcional pero menos elegante que el modelo de isolates de Workers, y su huella de borde es más pequeña que la de Cloudflare.

AWS S3 + CloudFront es el clásico "ensambla tú mismo". Es poderoso, configurable y barato a escala, pero requiere que conozcas IAM, políticas de bucket, comportamientos de caché, certificados ACM, configuración de Route 53 y Lambda@Edge si necesitas lógica dinámica. El costo total de propiedad —contando el tiempo del operador— rara vez supera a Cloudflare Pages para un equipo pequeño. Cloudflare ganó en la categoría de "no tienes que pensar en ello".

El modelo operacional: builds, despliegues, rollbacks

El flujo en Cloudflare Pages es deliberadamente aburrido. Conectas un repositorio de Git, configuras un comando de build (`npm run build` o equivalente) y un directorio de salida (`./dist`, `./build`, `./public`). En cada push, Pages ejecuta el build, captura el output y lo despliega a la red. Cada build obtiene una URL única de vista previa, lo que significa que los rollbacks son una operación de un clic: apunta el alias de producción a un despliegue anterior y estás atrás en segundos.

Para los Workers, el flujo es similar vía la CLI de Wrangler. `wrangler deploy` empuja una nueva versión; `wrangler rollback` revierte. Las versiones se mantienen, por lo que puedes hacer despliegues progresivos (10% de tráfico al nuevo Worker, 90% al antiguo) sin infraestructura adicional. Esto es la liberación canary integrada.

El observability está incluido pero es ligero: registros en tiempo real vía `wrangler tail`, métricas en el dashboard, y la capacidad de exportar logs a un sink externo si necesitas análisis profundo. Para un sitio cuya superficie dinámica son unos pocos Workers, esto es suficiente. Si necesitas APM completo, estás probablemente operando algo más grande que un sándwich estático-pero-dinámico, y deberías reconsiderar la arquitectura.

03 — Mira: un recorrido real

04 — Lista de configuración: poniendo en marcha la pila

Esto te lleva de cero a un sitio en producción en Cloudflare Pages con un Worker para lógica dinámica. Asume que ya tienes un sitio estático que se construye localmente.

1. Instala la CLI de Wrangler globalmente: `npm install -g wrangler`. Esta es la única herramienta que necesitas para interactuar con Pages y Workers.
2. Autentícate con tu cuenta de Cloudflare: `wrangler login`. Esto abre el navegador para OAuth y guarda un token local. No comprometas el token en Git.
3. Crea un proyecto de Pages: `wrangler pages project create <YOUR_PROJECT>`. Esto reserva el subdominio `<YOUR_PROJECT>.pages.dev` y registra el proyecto bajo tu cuenta.
4. Despliega tu directorio de build: `wrangler pages deploy ./build --project-name=<YOUR_PROJECT>`. La primera vez sube todos los archivos; los despliegues subsecuentes solo suben los cambios.
5. Conecta un dominio personalizado vía el dashboard de Cloudflare. Si tu dominio ya está en Cloudflare DNS, la verificación y el certificado TLS son automáticos. Si no, mueve tus nameservers o configura un registro CNAME.
6. Para lógica dinámica, escribe un Worker como un solo archivo `worker.js` con un controlador `fetch` exportado. Despliégalo con `wrangler deploy`. Víncula una ruta personalizada (por ejemplo `example.com/api/*`) vía el archivo `wrangler.toml`.
7. Configura un dominio en tu registrador (/learn/es/the-stack/cloudflare-pages-workers-architecture/) apuntado a los nameservers de Cloudflare. Esto desbloquea DNS, certificados TLS y reglas de páginas automáticas en una sola plataforma.

Preguntas frecuentes.

¿Realmente puedo correr un sitio de producción en el plan gratuito de Cloudflare?

Sí, para sitios mayormente estáticos. Pages ofrece ancho de banda ilimitado, builds ilimitados (500/mes en el plan gratuito), dominios personalizados gratuitos y TLS automático. Workers ofrece 100,000 solicitudes por día gratis con 10ms de CPU por solicitud. Para Pillar —14,000+ archivos HTML y un Worker de pegamento— estamos cómodamente dentro del plan gratuito. Cuando excedes el plan gratuito, los planes pagados son significativamente más baratos que los equivalentes de Vercel o Netlify a la misma escala.

¿Y si necesito una base de datos o sesiones de usuario?

Tienes varias opciones, en orden de complejidad creciente. Para datos pequeños, usa Workers KV (almacén clave-valor con consistencia eventual a escala global). Para datos relacionales, Cloudflare D1 (SQLite en el borde, todavía en evolución). Para sesiones, considera firmar un JWT en un Worker y almacenarlo en una cookie —sin estado en el servidor—. Si tu necesidad supera estos primitivos, la arquitectura estática-pero-dinámica probablemente no sea la opción correcta; estás construyendo una aplicación, no un sitio.

¿Cómo manejo redirecciones, cabeceras y reglas de página?

Pages soporta dos archivos especiales en la raíz de tu directorio de build: `_redirects` (una redirección por línea, sintaxis estilo Netlify) y `_headers` (cabeceras HTTP personalizadas por ruta). Estos se aplican en el borde antes de servir el archivo. Para lógica más compleja (redirecciones condicionales por geo, A/B testing), eleva esa lógica a un Worker. La regla práctica: empieza con `_redirects`, graduándose a un Worker solo cuando la lógica supere lo que una tabla declarativa puede expresar.

¿Qué pasa si Cloudflare tiene una interrupción?

Cloudflare opera múltiples redes anycast redundantes, pero las interrupciones suceden. La mitigación es arquitectónica: porque tu sitio es HTML estático, puedes desplegar el mismo build a un proveedor secundario (Netlify, AWS S3) y cambiar el DNS para failover. Pillar mantiene un build de respaldo precisamente por esta razón. La Worker capa es más difícil de reflejar, pero porque es *pegamento* y no estado, es reemplazable: podrías reescribirlo como una Lambda de AWS en menos de una hora si fuera necesario.

¿Vale la pena migrar un sitio existente desde Vercel o Netlify?

Depende de tu mezcla de estático vs dinámico. Si tu sitio es 90%+ estático (un blog, sitio de documentación, sitio de marketing, páginas de aterrizaje) y tu factura mensual está subiendo, la migración usualmente toma un día y reduce el costo a casi cero. Si tu sitio depende fuertemente del SSR de Next.js, ISR, o las API Routes de Vercel, la migración es más involucrada —tendrías que refactorizar la lógica del servidor en Workers o cambiar a la exportación estática de Next.js. El criterio de decisión es: ¿qué tan grande es tu relleno?
