

FIELD NOTES FROM A PRESALES SOLUTIONS ARCHITECT

Your Cloudflare Edge Is in the Wrong Country — colo= vs loc= and a 6-Second Page That Became 1 Second

A presales war story about a slow site behind Cloudflare — the edge location almost nobody checks — and the two-line test I now run before anyone touches the server.

Daanial Farrukh · 20 October 2026 · <https://quabin.com/cloudflare-colo-wrong-country-slow-site/>

The complaint came in the usual way. "The website is slow. Can you look at the server?"

It was a public-sector portal. The server was in Pakistan. Almost every visitor was in Pakistan. And the site sat behind Cloudflare, because "Cloudflare makes it faster."

The team had already done the normal things. They checked CPU. They checked memory. They restarted Apache once, just in case.

"The server is fine," they said. "But the page still takes six seconds."

I asked one question. "How long does it take if you skip Cloudflare and hit the server directly?"

Silence. Nobody had tried.

So we tried. Same laptop, same browser, same page — pointed straight at the origin IP.

About one second.

Six seconds through Cloudflare. One second without it. Same server, same content, same moment.

"So Cloudflare is slowing us down?"

Not Cloudflare itself. Where Cloudflare was serving them from.

The Trap — The Edge You Never Look At

That's the trap. A slow site behind a CDN does not show up as a slow server. Every server metric is green, so everyone keeps looking at the server.

The problem was not on the server. It was on the map.

What a Cloudflare "colo" Actually Is

Cloudflare runs data centres all over the world. Each one is called a **colo** (short for colocation), and each has a three-letter airport code: LHE for Lahore, DXB for Dubai, MCT for Muscat, SIN for Singapore.

The colo is the Cloudflare data centre that your visitor actually talks to. It is not a setting on your site. Cloudflare uses Anycast: the same IP address is announced from every data centre, and the visitor's internet provider decides — through its own routing — which one the traffic lands on.

When the colo is close to both the visitor and the origin, the path is short:

Visitor (Pakistan) → Cloudflare colo (Pakistan) → Origin (Pakistan)

When the visitor's provider sends traffic abroad, the same request looks like this:

Visitor (Pakistan) → Cloudflare colo (Oman) → Origin (Pakistan) → Cloudflare colo (Oman) → Visitor (Pakistan)

That is an international round trip for something that should never have left the country. Once is fine. But a modern page is not one request.

This portal's homepage made about **140 requests** — 18 MB, mostly unoptimised images and about 15 separate CSS files. Every small CSS or JS file took **300–600 ms through Cloudflare** and about **30 ms direct**. Multiply by 140 and the six seconds explain themselves.

This pattern is common where Cloudflare has weak local peering. In Pakistan, visitors on non-Enterprise plans are often served from Muscat, Dubai or Singapore. Cloudflare's own documentation describes Pakistani traffic being routed to SIN, MCT and DXB, driven by local provider peering and submarine-cable damage.

Where you see it most:

- Countries where the main providers do not peer well with Cloudflare locally.
- Sites whose origin and audience are in the same country.
- Heavy pages with many small, uncached files.

Why the Usual Check Misses It

There are three reasons nobody found it.

First, **server monitoring cannot see it**. The origin answered every request quickly. CPU, memory and Apache were all healthy, because they were.

Second, **the colo is per visitor, not per site**. There is no "site colo" in the dashboard. A visitor on one provider may land in Lahore, while a visitor on another lands in Muscat. The person testing from the office may get a different path from the people complaining.

Third, **it changes over time**. Cloudflare steers traffic for congestion, maintenance and capacity, and its documentation says it may route self-service plans to other data centres when peering is missing or load is high. A test from last month proves nothing about today.

Here's the part that trips people up: the first instinct is "the cache is not working". But direct-to-origin has no cache at all — and it was still six times faster. **The cost was the route, not the cache.**

How to Actually Fix This

Step 1: Read colo= and loc= — the two-line test

Every Cloudflare-proxied site answers on a special path. Open it in the browser of a real visitor, or run:

```
curl https://example.com/cdn-cgi/trace
```

Look at two lines:

```
colo=MCT  
loc=PK
```

loc= is the visitor's country. colo= is the Cloudflare data centre serving them. In this case: visitor in Pakistan, edge in Muscat, Oman.

If colo= is in the same country as loc= and your origin, the route is fine and the slowness is somewhere else. If it is in another country, keep going.

Plain takeaway: **two lines tell you whether your traffic is leaving the country.**

Step 2: Time one small file both ways

Pick a small static file, like a CSS file. Time it through Cloudflare:

```
curl -s -o /dev/null -w "connect=%{time_connect} ttfb=%{time_starttransfer} total=%{time_total}\n" https://example.com/wp-includes/css/dist/block-library/style.min.css
```

Then time the same file directly from the origin. --resolve sends the request to the origin IP while keeping the hostname, so TLS and virtual hosts still work:

```
curl -s -o /dev/null -w "connect=%{time_connect} ttfb=%{time_starttransfer} total=%{time_total}\n" --resolve example.com:443:<ORIGIN_IP> https://example.com/wp-includes/css/dist/block-library/style.min.css
```

Add -k if the origin uses a Cloudflare Origin Certificate, because that certificate is only trusted by Cloudflare.

On this portal the numbers were roughly 500 ms per file through the edge against 30 ms direct. That difference, times 140 files, is the whole problem.

Plain takeaway: **if one static file is ten times slower through the edge, the page will be too.**

Step 3: Confirm with the full page

One file can lie. A full page cannot. Point your own machine at the origin with a hosts-file entry:

```
<ORIGIN_IP> example.com
```

On Windows it lives in C:\Windows\System32\drivers\etc\hosts, on Linux and macOS in /etc/hosts. Load the page with DevTools open (Network tab, cache disabled) and note the total load time. Remove the line and load it again through Cloudflare.

This step is where most tutorials stop and where mine starts. Remember to delete the hosts entry afterwards — otherwise you will keep testing the origin and wonder why the "fix" worked only for you.

Our result: about 1 second direct, about 6 seconds through the domain.

Step 4: Log CF-Ray to see what real visitors get

The trace shows only your own colo. To see what your visitors get, log the CF-Ray header at the origin. Its value ends with the colo code, for example 8a1b2c3d4e5f6789-MCT.

Check it on one response:

```
curl -sI https://example.com | grep -i cf-ray
```

To log it for every visitor, add it to the access log format. In nginx the variable is `$http_cf_ray`; in Apache it is `%(CF-Ray)i`. After a day, count the codes at the end of each value. If most of your traffic ends in a foreign code, you have your answer from real users, not from one laptop.

Plain takeaway: **there is no single site colo — measure it across real visitors.**

Step 5: Choose the fix — and be honest about the trade-off

You cannot force the colo on a normal plan. Only the Enterprise plan can pin traffic to the nearest data centre. So the choices are these:

Option A — cache static files at the edge. This removes the colo → origin leg for cached files. It does not remove the visitor → colo leg, so a foreign colo is still slower than local direct traffic. Worth doing anyway.

Option B — make the page lighter. Combine the 15 CSS files, compress the 18 MB of images, cut the request count. Fewer round trips means less damage from each slow one. Worth doing anyway, too.

Option C — DNS-only (grey cloud). In the Cloudflare dashboard, go to DNS → Records, edit the record and switch Proxy status from Proxied to DNS only. Visitors then go straight to the origin over the local network.

This customer chose Option C for a domestic audience. It is the fastest, but it is a real trade:

- Cloudflare's DDoS protection and WAF no longer sit in front of the site.
- The origin IP is now public in DNS.
- The origin must already have a publicly trusted certificate. A Cloudflare Origin Certificate will give every visitor a browser error.

Check the certificate **before** you flip the switch:

```
openssl s_client -connect <ORIGIN_IP>:443 -servername example.com </dev/null 2>/dev/null |  
openssl x509 -noout -issuer -dates
```

If the issuer is a public authority like Let's Encrypt or Sectigo and the dates are valid, you are safe. If it says Cloudflare Origin, fix the certificate first.

Step 6: Verify the switch

After the change, the domain must resolve to the origin, not to Cloudflare:

```
nslookup example.com
```

You should see your origin IP. If you still see Cloudflare addresses, wait for the old DNS record to expire and test again.

Then the trace must disappear:

```
curl -I https://example.com/cdn-cgi/trace
```

A 404 here is good news. **This error is not a bug. It is the verification.** The path only exists on Cloudflare, so a 404 proves traffic no longer passes through it.

Finally, load the full page again from a normal visitor connection. Our portal went from about 6 seconds to about 1 second, with no browser certificate warnings.

Step 7: Re-protect the origin — the step people skip

Here is the reversal nobody warns you about. Many hardening guides tell you to lock port 443 so it only accepts Cloudflare's IP ranges. That is correct when the site is proxied. **After going DNS-only, that same rule blocks every real visitor**, because they now arrive from their own IPs.

So the protection has to move to the origin: the server firewall, connection limits, rate limiting, and fail2ban or CSF for abuse. Plan that the same day you grey-cloud the record, not after the first attack.

The Bigger Lesson — A Short Checklist

- 1 **Edge location** — `curl https://example.com/cdn-cgi/trace`, compare `colo=` with `loc=`.
- 2 **Edge vs direct, one file** — the two `curl -w` timings above.
- 3 **Edge vs direct, full page** — hosts-file test with DevTools.
- 4 **Real visitor colos** — log CF-Ray and count the `colo` codes.
- 5 **Page weight** — DevTools Network tab: request count and total MB.
- 6 **Origin certificate** — the `openssl s_client` check before any grey-cloud.
- 7 **Origin firewall after DNS-only** — no Cloudflare-only rule left on 443.

What I Tell Every Customer Now

A CDN is a promise about distance. It only keeps that promise if the edge is close to the people using it.

So before anyone opens top on a "slow server" behind Cloudflare, I ask one question:

"Which colo are your visitors actually landing on?"

If the answer is a city in their own country, they're in good shape, and we go back to the server. If the answer is "What's a colo?", we're not done yet.

I write about the questions I ask in real customer rooms — the ones that change architectures and surface the problems no one else noticed. If you found this useful, drop your email below and I'll send the next one when it's ready.

FAQs

Why is my website slower with Cloudflare than without it?

The most common reason is the route. If your visitors are served by a Cloudflare data centre in another country while your server is in their own country, every uncached request makes an international round trip. Compare `colo=` and `loc=` at `/cdn-cgi/trace` to check.

What do colo and loc mean in Cloudflare's cdn-cgi/trace?

colo is the three-letter airport code of the Cloudflare data centre that served your request, such as MCT for Muscat. loc is the country Cloudflare thinks you are in. If they are in different countries, your traffic is leaving the country before it reaches your site.

Can I choose which Cloudflare data centre serves my visitors?

Not on the Free, Pro or Business plans. Cloudflare uses Anycast, so the visitor's internet provider decides which data centre their traffic reaches. Only the Enterprise plan can pin traffic to the closest data centre.

Will caching fix a slow Cloudflare colo?

Partly. Caching static files removes the trip from the colo to your origin, but the visitor still has to reach the distant colo. Caching and a lighter page reduce the damage; they do not make a foreign edge as fast as a local direct path.

Is it safe to switch a Cloudflare record to DNS only?

It is safe if you accept the trade-offs and prepare for them. You lose Cloudflare's DDoS protection and WAF, your origin IP becomes public, and the origin needs a publicly trusted certificate. Also remove any firewall rule that only allows Cloudflare IPs, or real visitors will be blocked.

How do I see which Cloudflare colo my real visitors use?

Log the CF-Ray request header at your origin. Its value ends with the colo code. In nginx use `$http_cf_ray`, in Apache use `%{CF-Ray}i`, then count the codes across a day of traffic.

Read online: <https://quabin.com/cloudflare-colo-wrong-country-slow-site/>