Engineer

CDN

servers near the reader that hand out copies

What it is

A CDN, a content delivery network, is a fleet of servers spread across many cities that stands between readers and the server a site really runs on, called the origin, and hands out copies of what the origin publishes. A reader in Sydney asking for a page whose origin sits in Toronto gets it from a machine in or near Sydney, and the request may never cross the Pacific at all.

Two mechanisms steer each reader to a nearby server. Some networks give a different DNS answer depending on where the question comes from. Others announce the same IP address from every location and let internet routing carry each packet to the closest one. Either way, the server that answers usually terminates TLS on the site's behalf, which means it holds a certificate for the domain and reads each request in the clear before deciding what to do with it.

What it does next is the job of a shared cache. If it already holds a fresh copy, it answers at once, which is a hit. If not, it fetches the response from the origin, keeps a copy and answers, which is a miss. How long a copy counts as fresh is the origin's decision, written in response headers that the HTTP caching standard defines:

```
# A fingerprinted stylesheet: never changes under this name.
Cache-Control: public, max-age=31536000, immutable

# A page: browsers check back each time, the CDN keeps it ten minutes.
Cache-Control: no-cache
CDN-Cache-Control: max-age=600

# An account page: no cache may keep a copy.
Cache-Control: private, no-store
```

`max-age` gives a lifetime in seconds. `immutable` tells the browser not to revalidate during that time. `no-cache` lets a cache store a response but makes it check with the origin before reusing it, while `private` keeps it out of shared caches altogether. `CDN-Cache-Control` is read only by CDNs, so a site can let the edge hold a page longer than browsers do.

The same machinery now carries much more than static files. Most CDNs also proxy uncacheable requests to the origin, absorb floods of hostile traffic and run small programs at the edge, which is why the term covers a broad range of services that all start from the same idea of serving copies close to the reader.

Use the concept in a decision

Put “What is a CDN?” to work in a real decision. We can help relate the concept to the system you are building or buying. Servers sized for what you actually use, moved into the cloud or back out of it, with the monthly bill worked out first.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Ran 20 Docker apps on Google Cloud and AWS — 20 containerised applications deployed and watched

Ran 20 Docker apps on Google Cloud and AWS Get in touch →

Why it matters

Distance costs time that no amount of server tuning can win back. Light in fibre covers roughly two hundred kilometres per millisecond, so a round trip between Europe and Australia takes well over a hundred milliseconds before any server has done any work, and a new secure connection spends one or two of those round trips before the first request is sent. Serving from a nearby copy removes most of that wait for every reader far from the origin.

Load is the second reason. When most requests are hits, the origin sees only a small share of the traffic, and a sudden spike lands on a network built for spikes rather than on one machine. With `stale-if-error` the edge can keep serving the last good copy while the origin is down, so a short outage stays invisible to many readers.

The cost is less obvious than the invoice. A CDN in front of a site is another company in the middle of every visit. It sees each reader’s IP address, every URL they request and, because it terminates TLS, the content of the exchange, and it handles all of it under its own policy. It is also a shared dependency: when a large provider has a bad hour, thousands of unrelated sites have it together.

The old argument for loading libraries and fonts from a popular public CDN has weakened too. Browsers now keep a separate cache for each site, so a file fetched for one site is fetched again for the next, and the shared copy brings a third-party request with no cache benefit to show for it.

A site whose readers sit mostly in one region, served from a fast machine in that region, may simply not need a CDN. A site with a worldwide audience, heavy media or traffic that arrives in bursts usually does.

What it looks like in practice

On this site the CDN question was settled by trying one and switching it off. The company's DNS work records that proxying the two web faces through the provider's content network was built, worked, and was then turned back off. With the proxy in place, another company handled every visitor's address and every URL they asked for, before the site did and under that company's policy rather than this one's. For a business whose distinguishing claim is that nobody is watching, the page judges that a worse trade than the attacks the proxy defends against, and the reasoning is written down so that nobody reopens the question by accident.

The same position shows in what the pages load. The credits page names the five kinds of file that reach a reader's browser and explains why the typefaces come from the site's own machine: a webfont from someone else's CDN is a third party watching you read. Cloudflare answers the site's DNS and does nothing more, the site is deliberately not proxied, and a reader who opens the browser's network panel counts one host.

Where a CDN earns its place, the notes say so. The note on flash sales argues that turning a buyer away has to be the cheapest thing the system does. With a million buyers and ten thousand units, 990,000 requests have to fail. Among the ways it lists to shed them early are a waiting room, a gate sized to the stock, a per-account rate limit and a `sold_out` flag that the CDN hands out without consulting anything behind it. Its advice on load testing makes the same point from the other side: when the sold-out path is 99% of the traffic, that traffic belongs at the edge.

Where it goes wrong

The edge does exactly what the headers and its configuration say, which is not always what was meant.

- Someone else's page is served to the next visitor. A response that shows an account, a basket or a greeting is cached because nothing marked it private. Setting a cookie does not stop caching, as the HTTP caching standard points out, so the next reader gets the first reader's page. Mark every per-user response `private` or `no-store`, and tell the CDN to bypass its cache when a session cookie is present.
- A deploy depends on a purge. New HTML goes live while some edges still hold the old stylesheet, or the reverse, and pages render broken until every copy expires. Purges take time to reach every location and vary between providers. Put a hash of the content in each asset's file name, cache those files for a year as `immutable`, and keep only the HTML short-lived.
- The cache key is wrong in one direction or the other. Tracking parameters such as `utm_source` turn every shared link into a separate cache entry, and the hit ratio collapses. Leaving out a header that changes the response, such as `Accept-Language`, serves one reader's language to everyone. Strip parameters the origin ignores, and name in `Vary` the headers it reads.
- A popular object expires and the origin is flooded. Thousands of readers miss at the same instant and every one of those requests goes through. Request collapsing at the CDN and `stale-while-revalidate`, which serves the old copy while a single request fetches the new one, keep the origin's load flat.
- The origin answers anyone who finds it. Protection at the edge means little if attackers can reach the origin's address directly, and old DNS records, mail headers or a forgotten subdomain often give it away. Accept traffic at the origin only from the CDN, by network range or by an authenticated connection.

Further reading

- RFC 6707, section 1.1: CDN terminology — the IETF's definitions of a CDN, its surrogates and request routing.
- RFC 9111: HTTP Caching — what shared caches may store, for how long, and what `private` means.
- RFC 9213: Targeted HTTP Cache Control — the `CDN-Cache-Control` field that speaks only to CDNs.
- RFC 5861: HTTP Cache-Control Extensions for Stale Content — `stale-while-revalidate` and `stale-if-error`.

- RFC 8246: HTTP Immutable Responses — how a fingerprinted file avoids being checked again.

Glossary

Choose for your actual constraints

Bring the deployment path, the operational risks and the people on call. We can scope a change with checks, a recovery route and clear ownership.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is a CDN?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/cdn/