**Content Delivery Network Architecture**

Purpose: To deliver the static web content to the user as soon as possible (minimizing latency). Static web content consist of images, CSS, JS, multimedia etc. are the major contributor in today’s internet traffic. Again most of such content is repeatable and usually do not change frequently in short span of time which means it is cacheable by nature.

Here the concept of CDNs comes. These are kind of locally available datacenters primarily used to cache the static resources which can be used to satisfy the frequent web requests.

Low latency is achievable if the content can be delivered by the nearest server or PoP(point of presence).

In a typical CDN setup, a request will first ask the CDN node for a piece of static media, the CDN will serve that content if it has it locally available. If it isn’t available, the CDN will query the back-end application servers for the content and then cache it locally and serve it to the requesting user.

Some of the benefits of using CDN are-

* Latency
* Security
* Scalability
* Availability

Various CDN products/services are available in market to fulfill the above use-case.

* Cloudflare
* Alfresco
* Akamai
* Incapsula

Let’s break the requirement into high level FR & NFR so that the ambiguities in requirement can be identified and cleared out.

**Functional requirement**

There could be many functional requirements depending on the nature of business and users. However, for the sake of simplicity assuming the following FRs only.

1. **System should be capable of delivering the resource on demand to the client.**
2. There should be an interface through which user can post the request (REST/UI).
3. System should have a database to backup/store all of the content and information.
4. Multiple users should be allowed to access the system concurrently for the same resource.
5. User registration and authentication module.
6. System security against malicious attacks.
7. etc.

**Non-Functional requirement**

Again, out of many following are some of important NFRs.

1. **System should be capable to handle 1 billion request a day.**
2. **Avg. latency to grab a resource should not be more than 100 ms.**
3. System should be highly available.
4. System should be able to handle the peak load time.
5. System should be resilience in failover scenarios.
6. etc.

Following is a high level diagram which states how various components in a typical CDN interacts -

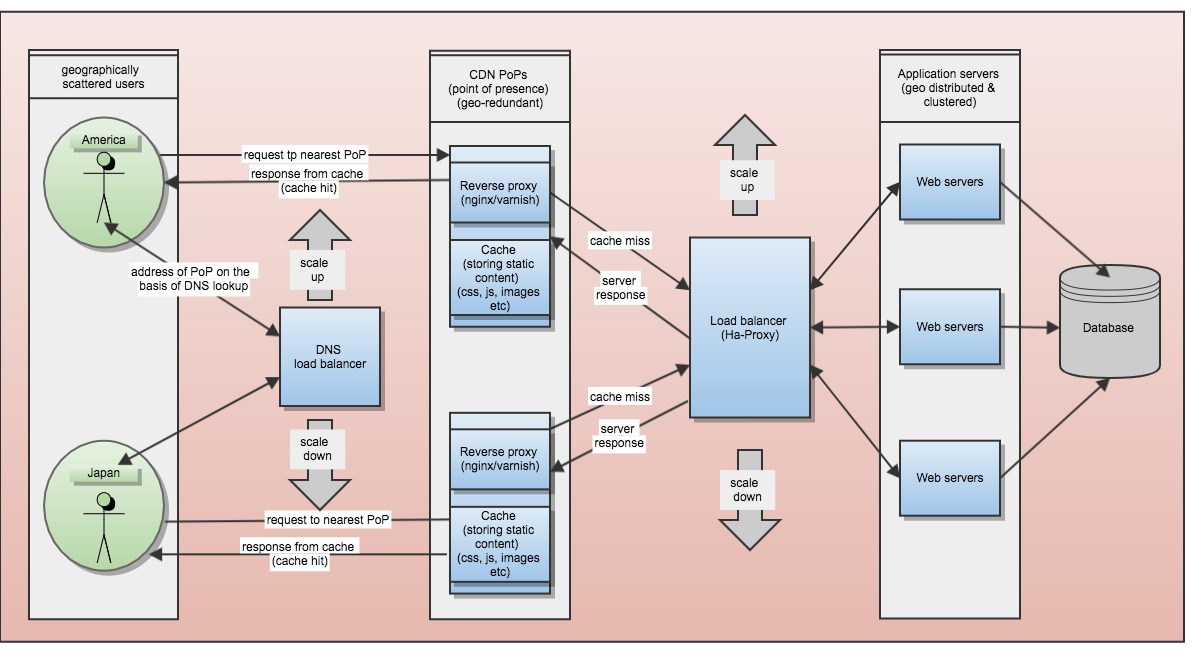


Figure 1: CDN architecture

Working: CDN infra introduces another layer of servers between application and the client which keeps static data (mostly) in their cache. Also, serves as a reverse proxy.

As in the above HLD 3rd layer of CDN PoPs is the layer which is responsible for delivering the static content to the user.

For example user has requested for some document. A typical GET request <http://server.com/documents/id>. Now, if there is no CDN installed every request would be served by the application sever only which could be situated very far geographically (probably in different continent).

In such scenarios overall latency from domain lookup to response resolved could add up to become significant. Even worst, if systems aren’t efficiently scaled at peak time and if there is any DDOS attack, sudden outage and so on.. any ways high latency is simply not acceptable in any scenario.

These cache datacenters keeps static content with them and can provide as per the request.

CDNs moves data closer to the user. In above example if we have setup a CDN node then the request can be served by local PoP quickly from its cache. Even in case of outage of main application server, requests can be served for some time.

There are various policies to make cache efficient and fresh like

LRU (least recently used) , MRU (most recently used) etc. which can be used as per the nature of business/requests.

Cache miss vs Cache hit concept.

Whenever any request is fulfilled by the CDN PoP servers it is assumed to be a hit, if the resource isn’t available that request gets transferred to the main application server to fulfil, this scenario is called cache-miss.

Any successful CDN service would have high number of cache-hits. There are various policies to keep the CDN PoP’s cache fresh and efficient like push (server pushes delta) or pull (cache-miss).

**Capacity planning –**

***Note – Below values are on pure assumptions basis. There are many parameters that needs to be consider doing real approximations like bandwidth available, network load, computation power, memory footprint, physical distance etc.***

Let’s assume the CDN infrastructure and network infrastructure is up and running and typical request cycle goes like –

**Worst case scenario**

The requested content is not available with the CDN PoP which means cache-miss scenario. Here whole round trip to the database is needed to deliver the resource and update the PoP.

Following is a matrix to show some latency approximations with the assumption that servers are responding almost immediately. (no processing cycles)

|  |
| --- |
| **Various latencies in cache-miss case** |
| DNS query 🡪 10 ms  DNS response 🡨 10 ms  Reverse Proxy request 🡪 10 ms  Reverse Proxy response 🡨 10ms  Load-balancer request 🡪 20 ms  webserver request 🡪 20 ms  Database query 🡪 50 ms  Webserver response 🡨 20 ms  Load-balancer response 🡨 20 ms  **Total -------------- 170 ms** |

**Best case scenario**

The requested content is available with the CDN PoP which means cache-hit scenario. Here request can be satisfied by the PoP almost instantly through its cache.

Following is a matrix to show some latency approximations with the assumption that PoP is responding almost immediately. (no processing cycles)

|  |
| --- |
| **Various latencies in cache-miss case** |
| DNS query 🡪 10 ms  DNS response 🡨 10 ms  Reverse Proxy request 🡪 10 ms  Reverse Proxy response 🡨 10ms  **Total -------------- 40 ms** |

Above matrices shows that use of CDN can cut the latency to 1/4th of the total latency.

Which means resources can be served as fast as almost 75%.

System scaling is the foremost thing to consider when requirement is to serve 1 billion requests. So, couple of scaling measures to consider are -

* Horizontal scaling (adding more machines in clusters/pool)
* Vertical scaling (increasing the capacity of existing machines)
* Data sharding
* Data replication

Question of billion request per day

* 1 billion request / 24 hrs = ~41 million per hour
* 41 million / 60 mins = ~6,94,000 per minute
* 6,94,000 / 60 sec = 11,574 requests per second
* If we consider the avg cache-hit ratio (90%) ~10,400 requests can be served from the cache locally with approx. ~40 ms for each request.

Further, to achieve throughput of ~11,000 request each second constantly. We would need to scale up the system. Various measures can be taken to achieve this goal –

* Identification of major traffic generating locations. We could install multiple PoPs around those locations so that customers requests can be fulfilled locally and efficiently.
* Decreasing the DNS lookups latency, may be caching the dns data with TTL parameter or installing the premium ones which provides lookups in nanoseconds.
* Extending the caching size in PoPs, this can significantly increase the hit ratio and the throughput at last.
* Increasing the data processing on the edge.

-Vinit Saini.