Jahong Liu

System Design Basics 11: Content Delivery Network (CDNs)

Content Delivery Network

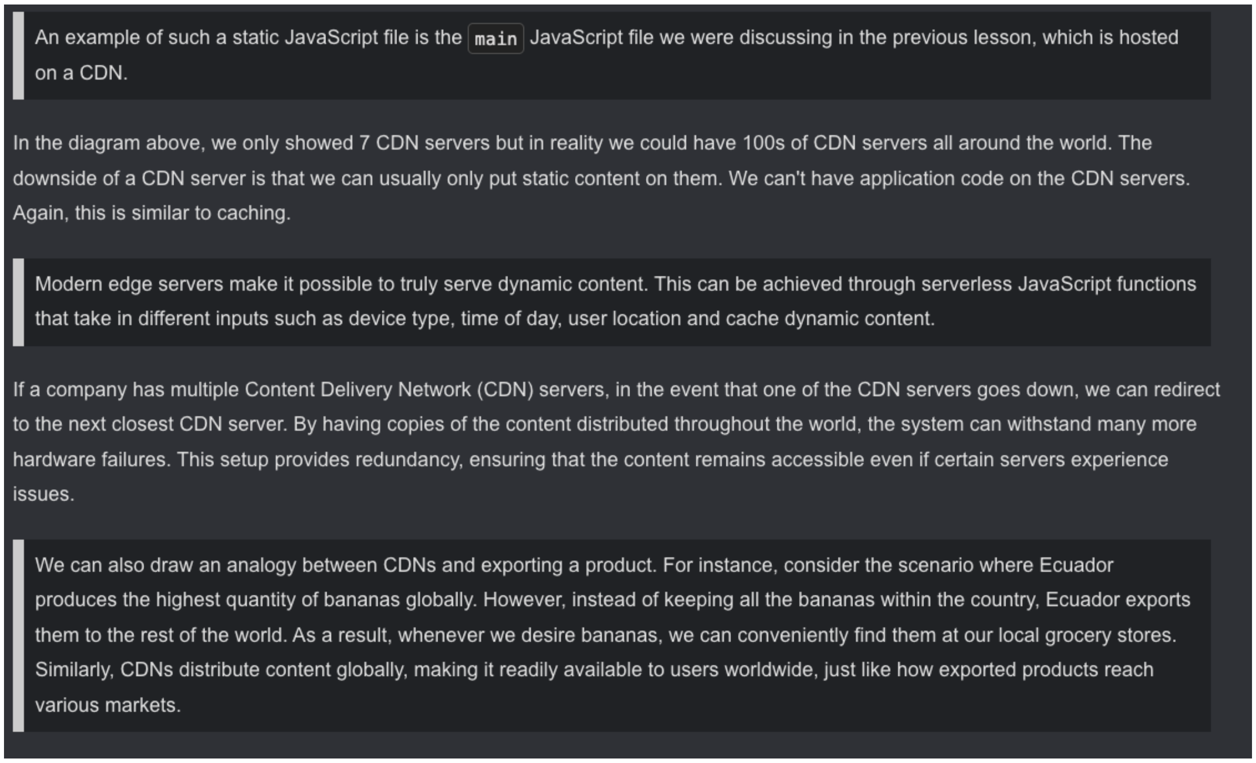
Origin Server
CDN
CDN
CDN
CDN
CDN
CDN
CDN
CDN

A **Content delivery network** (CDN) is a group of **cache servers** (or edge serveres) that are located around the world so they can cache content close to end users

Using a CDN, data can be accessed faster than fetching it from the **original server**. If a website is accessed throughout the world, it's beneficial to use a CDN, so even the clients that are far from the origin server can access the content without any noticeable delays.

It's simple from a developer's perspective, because code is deployed to the origin server. That said, managing the CDNs configurations can still be apart of the developer's job depending on how 'auto-managed' the CDN is.

Caching and CDNs are closely related because the basic concept behind a CDN is to store copies of a website's content on multiple servers in different geographical locations. This not only speeds up content delivery but also reduces the load on the original server and the amount of data that has to be transmitted over long distances, which saves bandwidth and improves performance. The kind of content that we would use a CDN for would be static. This includes all of the static content, including HTML, CSS and static JavaScript files.


> An example of such a static JavaScript file is the `main` JavaScript file we were discussing in the previous lesson, which is hosted on a CDN.

In the diagram above, we only showed 7 CDN servers but in reality we could have 100s of CDN servers all around the world. The downside of a CDN server is that we can usually only put static content on them. We can't have application code on the CDN servers. Again, this is similar to caching.

> Modern edge servers make it possible to truly serve dynamic content. This can be achieved through serverless JavaScript functions that take in different inputs such as device type, time of day, user location and cache dynamic content.

If a company has multiple Content Delivery Network (CDN) servers, in the event that one of the CDN servers goes down, we can redirect to the next closest CDN server. By having copies of the content distributed throughout the world, the system can withstand many more hardware failures. This setup provides redundancy, ensuring that the content remains accessible even if certain servers experience issues.

> We can also draw an analogy between CDNs and exporting a product. For instance, consider the scenario where Ecuador produces the highest quantity of bananas globally. However, instead of keeping all the bananas within the country, Ecuador exports them to the rest of the world. As a result, whenever we desire bananas, we can conveniently find them at our local grocery stores. Similarly, CDNs distribute content globally, making it readily available to users worldwide, just like how exported products reach various markets.

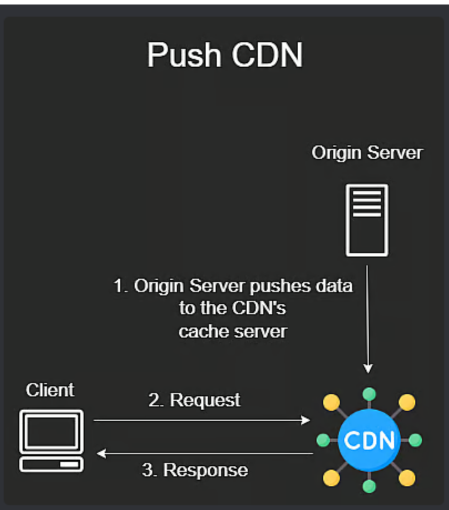
Push CDN

A push CDN is particularly well-suited for websites where the content is static and doesn't change frequently. In a push CDN, once new data is added to the origin server, it is immediately pushed to all of the cache servers.

It's also important to note that the content need to be widely requested for a push CDN to be efficient. Since push CDNs distribute content to all cache servers, they can be inefficient if the content isn't requested by users near those servers.

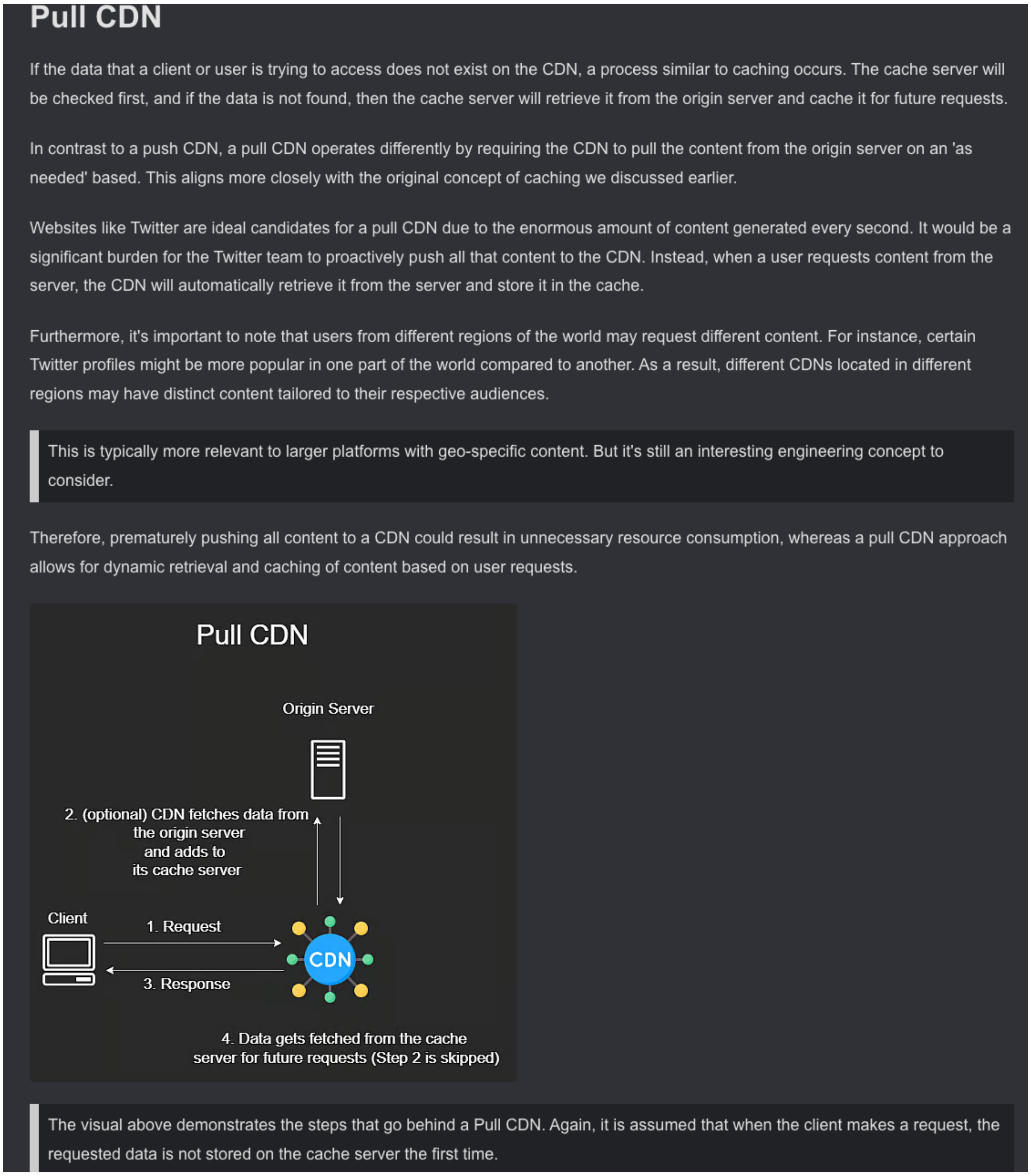
A good example of a suitable use case for push CDNs is websites that host video conent consumed around the globe. Push CDNs offer an advantage in that users don't have to wait for the content to be transferred from the original server to the CDN.

Each user request will result in a cache hit since the data has already been copied over. While the content owner has complete flexibility in determining what content makes it to the CDN, they also bear the responsibility of constantly updating and maintaining the files on the server.



> The visual above serves as a demonstration of the process behind a push CDN. The diagram above assumes that the CDN does not have the cached content when the client makes a request.

Pull CDN



Pull CDN

If the data that a client or user is trying to access does not exist on the CDN, a process similar to caching occurs. The cache server will be checked first, and if the data is not found, then the cache server will retrieve it from the origin server and cache it for future requests.

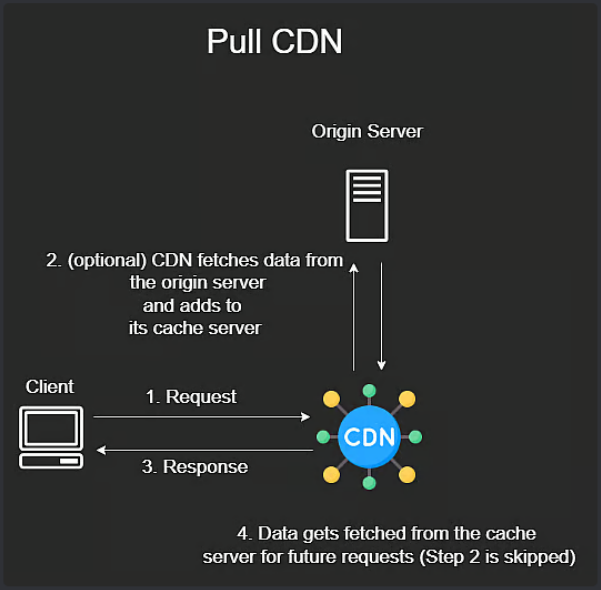
In contrast to a push CDN, a pull CDN operates differently by requiring the CDN to pull the content from the origin server on an 'as needed' based. This aligns more closely with the original concept of caching we discussed earlier.

Websites like Twitter are ideal candidates for a pull CDN due to the enormous amount of content generated every second. It would be a significant burden for the Twitter team to proactively push all that content to the CDN. Instead, when a user requests content from the server, the CDN will automatically retrieve it from the server and store it in the cache.

Furthermore, it's important to note that users from different regions of the world may request different content. For instance, certain Twitter profiles might be more popular in one part of the world compared to another. As a result, different CDNs located in different regions may have distinct content tailored to their respective audiences.

> This is typically more relevant to larger platforms with geo-specific content. But it's still an interesting engineering concept to consider.

Therefore, prematurely pushing all content to a CDN could result in unnecessary resource consumption, whereas a pull CDN approach allows for dynamic retrieval and caching of content based on user requests.

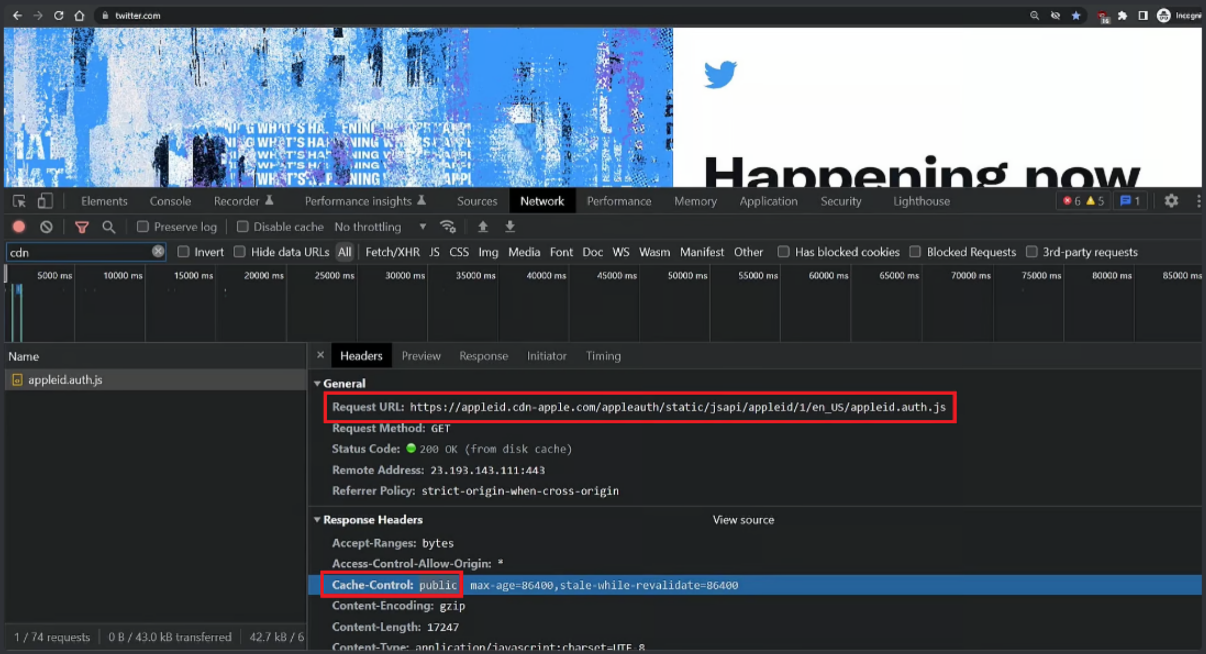

Pull CDN
Origin Server
2. (optional) CDN fetches data from the origin server and adds to its cache server
Client
1. Request
CDN
3. Response
4. Data gets fetched from the cache server for future requests (Step 2 is skipped)

> The visual above demonstrates the steps that go behind a Pull CDN. Again, it is assumed that when the client makes a request, the requested data is not stored on the cache server the first time.

Twitter Example

We can confirm that Twitter utilizes a CDN by visiting the Twitter website and inspecting the page. In the elements tab, we can observe that Twitter employs a CDN for Apple Authentication. By opening the corresponding URL in a new tab, we can view the JavaScript code associated with it. Regardless of how many times the file is refreshed, the code remains static. Twitter could have opted to host the authentication on their own server, requiring them to maintain a separate copy of the code. However, they have leveraged the CDN provided by Apple to handle this task.

It is worth noting the presence of the "cache-control" attribute in the visual representation below. When the "cache-control" value is set to "public," it signifies that the code should be cached by the CDN. On the other hand, if "cache-control" is set to "private," it instructs the CDN not to cache the content..



```
General
  Request URL: https://appleid.cdn-apple.com/appleauth/static/jsapi/appleid/1/en_US/appleid.auth.js
  Request Method: GET
  Status Code: 200 OK (from disk cache)
  Remote Address: 23.193.143.111:443
  Referrer Policy: strict-origin-when-cross-origin
Response Headers
  Accept-Ranges: bytes
  Access-Control-Allow-Origin: *
  Cache-Control: public max-age=86400,stale-while-revalidate=86400
  Content-Encoding: gzip
  Content-Length: 17247
```

> The visual substantiates the `Cache-Control` and the `Request URL` header mentioned earlier.

Closing Notes

The most crucial concept behind CDNs is their ability to bring content closer to the user. CDNs exemplify scaling, although it differs from horizontal scaling. While these two concepts may be interconnected, they are not identical.

Horizontal scaling aims to increase throughput, enabling the system to handle a larger number of requests. It involves adding more servers or resources to handle the increased load. On the other hand, CDNs focus on minimizing latency and improving content delivery by strategically placing multiple servers across the globe. By doing so, they ensure that content is geographically closer to the users, resulting in faster access and reduced network congestion.

> There is the indirect benefit of horizontal scaling, since CDNs can increase throughput, but that is not necessarily the purpose.

In summary, horizontal scaling aims to handle more requests by adding resources, while CDNs optimize content delivery by bringing it physically closer to the users through a distributed network of servers.