# BÁO CÁO KHẢO SÁT HỆ THỐNG DNS CỦA MỘT SỐ NHÀ CUNG CẤP DỊCH VỤ

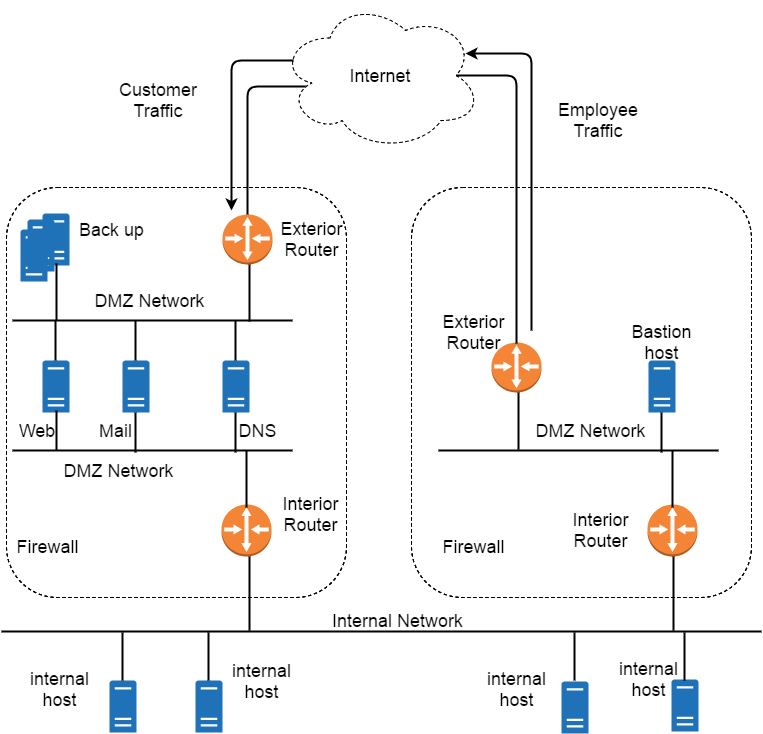
## 1. Các mô hình triển khai DNS

### *1.1. Master - Slave*

### *1.2. Multi - Master*

### *1.3. Stealth - DNS*

## 2. Mô hình cơ bản của nhà cung cấp dịch vụ



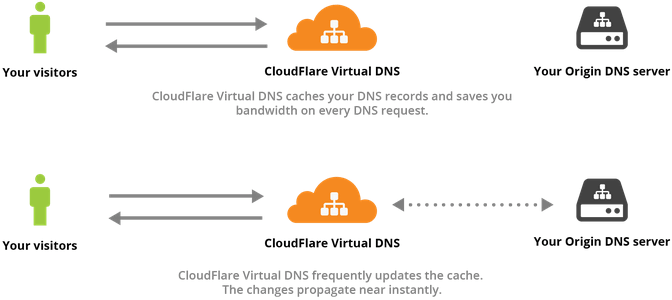
*(Tham khảo* ***Ebook Building Internet Firewalls*** *–* ***Chương 6*** *–* ***Firewall Architectures****)*

## 3. Dịch vụ DNS của Cloudflare

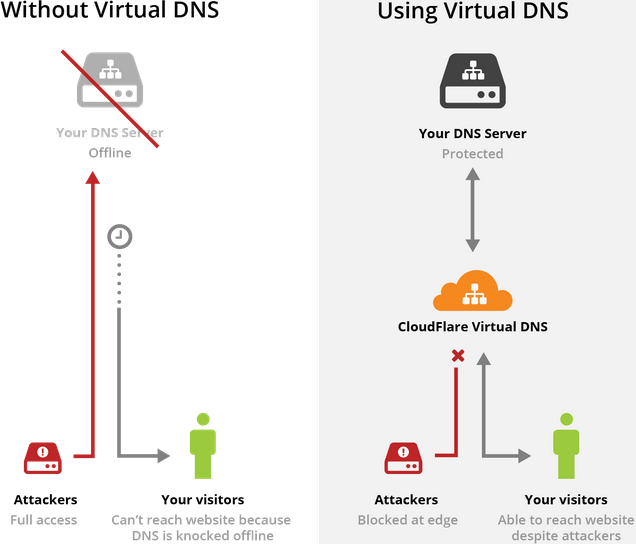
### *3.1. Authoritative Name Server*

* Giả sử với một website đặt tại server với địa chỉ IP 1.1.1.1. Thông thường khi mọi người truy cập website này, điều đầu tiên mà máy khách làm là thực hiện truy vấn tới hệ thống DNS để lấy được IP tương ứng với tên miền của website đó rồi bắt đầu duyệt.
* Khi sử dụng Cloudflare, chủ website chỉ cần vào hệ thống của nơi đăng ký tên miền và thêm name server của Cloudflare cung cấp khi đăng ký. Chủ website không cần thay đổi code của web, mọi thông tin tại nơi cung cấp host và tên miền vẫn giữ nguyên. Tuy nhiên lúc này khi đứng đằng sau hệ thống DNS của Cloudflare thì website sẽ có lớp bảo vệ và tăng tốc độ truy cập cấp độ toàn cầu.
* Để làm được điều này, CloudFlare sử dụng công nghệ định tuyến là Anycast và một số trick để định tuyến bản tin DNS lookup đối với một domain nào đó tới data center gần với khách truy cập nhất. Cloudflare có nhiều data center phân bố ở nhiều vị trí trên toàn cầu (PoPs). Tại mỗi nơi đó Cloudflare vận hành các cluster của các server. Và giả sử rằng datacenter gần nhất kia nhận được DNS lookup, nó sẽ trả lời với địa chỉ tương ứng Content Server tốt nhất so với vị trí địa lý của khách và điều hướng các yêu cầu tiếp theo của khách truy cập tới Server đó.
* Sau khi trình duyệt thực hiện xong DNS lookup thì nó bắt đầu gửi yêu cầu lấy dữ liệu thực của trang web. Lúc này trình duyệt sẽ truy cập tới địa chỉ của Content Server được cung cấp thay vì tới server đặt website ban đầu. Lúc này Content Server sẽ thực hiện phân tích yêu cầu và nguy cơ về bảo mật trong bản yêu cầu gửi tới.
* Nếu như yêu cầu sau phân tích hợp lệ thì Content Server bắt đầu kiểm tra xem tài nguyên được yêu cầu có nằm trong cache cục bộ tại đó không, nếu có thì nó trả về ngay cho người dùng. Thông thường những tài nguyên được cache là tài nguyên tĩnh như ảnh, các tệp CSS, JS,… Còn với những tài nguyên như HTML hay các video, Cloudflare không cache nên sẽ thực hiện yêu cầu tới server thực sự đặt trang web sử dụng công nghệ Smart Routing Argo để định tuyến thông minh lấy các nội dung được yêu cầu trong thời gian ngắn nhất.

### *3.2. Virtual DNS*

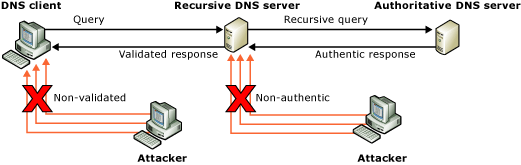


Virtual DNS sử dụng để bảo vệ hạ tầng DNS của khách hàng, trong trường hợp khách hàng vẫn muốn sử dụng hệ thống DNS của riêng họ. Cloudflare không thay đổi hoạt động của các DNS server này mà đặt chúng sau lớp bảo vệ là Virtual DNS. Virtual DNS sẽ là lá chắn trước các cuộc tấn công nhằm tới hệ thống DNS. Đồng thời khi DNS resolvers (ví dụ như ISP) thực hiện DNS lookup với authoritative DNS của khách hàng, truy vấn này đầu tiên sẽ tới CloudFlare Virtual DNS. Hoặc là Virtual DNS sử dụng cache để trả lời truy vấn, hoặc sẽ đi tới name servers để lấy phản hồi gửi về DNS Resolver. Trong trường hợp DNS Server của khách hàng không sẵn sàng thì Virtual DNS sẽ phản hồi lại với câu trả lời lấy từ cache. Các record đó sẽ được dự phòng cho tới khi hệ thống DNS của khách hàng trở về trạng thái online.

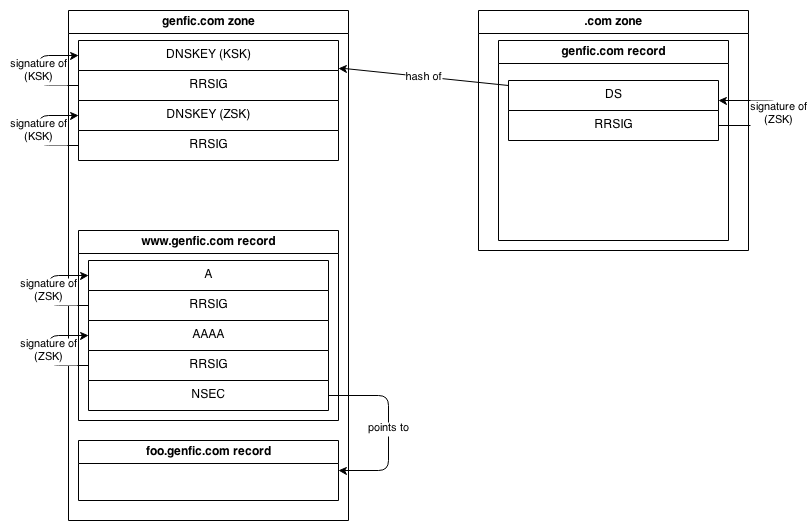


### *3.3. Bảo mật DNS – DNSSEC và ECDSA*

DNS protocol dễ bị tấn công do sự thiếu vắng về cơ chế xác thực và kiểm tra toàn vẹn dữ liệu khi trao đổi giữa các DNS server hoặc giữa DNS server với DNS client. Cụ thể hơn, tấn công giả mạo DNS hoàn toàn có thể xảy ra khi attacker đoán ra rằng có một DNS client hoặc server nào đó gửi DNS query và chờ đợi bản tin DNS response. Nếu như vụ tấn công này thành công, điều đó nghĩa là attacker sẽ chèn bản tin DNS response giả mạo vào cache của DNS server hay gây nhiễm độc cache. DNS server bị giảo mạo không có cách nào xác thực được DNS data là hợp lệ và sẽ gửi phản hồi từ cache của nó sử dụng thông tin giả mạo.



DNSSEC tạo ra một lớp bảo mật cho các bản tin DNS responses bằng cách cung cấp cho DNS server chứng thực được bản tin DNS responses. Với DNSSEC, các resource record (A, AAAA, NS, PTR, etc.) đi kèm với các chữ ký số. Các chữ ký số này được tạo ra khi DNSSEC được áp dụng cho DNS zone sử dụng thao tác zone signing. Khi một resolver (ví dụ ISP) gửi DNS querey hỏi lấy resource record trong một ***signed zone*** thì chữ ký số sẽ được trả về cùng với bản tin DNS response để thực hiện chứng thực. Nếu chứng thực thành công thì điều đó chứng tỏ rằng dữ liệu không bị thay đổi.



Ý tưởng ở đây là có một record (trong ví dụ ở đây là record cho <www.genfic.com>) có nhiều loại được ký, ví dụ loại A (cho ipv4) và AAAA (cho ipv6) và mỗi loại có một chữ ký riêng nó định nghĩa trong trường RRSIG. Trường RRSIG là các chữ ký số tạo ra bởi private key gọi là Zone Signing Key (ZSK) mà phần public của nó cũng nằm trong zone file định nghĩa với trường DNSKEY. Khi resolver gửi truy vấn phân giải tên thì RRSIG record sẽ được trả về trong bản tin phản hồi. Khi đó public key được cung cấp bởi trường DNSKEY sẽ được sử dụng để xác nhận chữ ký số đó. Và vì chỉ có DNS server giữ private key mới có thể thay đổi lại các record và ký lại nên attacker không thể giả mạo được (trừ việc có được private key hoặc phá giải thuật toán tạo chữ ký số).

Trường DNSKEY cũng được ký sử dụng private key là Key Signing Key (KSK). Phần KSK public key cũng được định nghĩa trong một trường DNSKEY. Vấn đề đặt ra là để chứng thực KSK public là key hợp lệ, nó sẽ được ký sử dụng KSK private key của zone ***genfic.com,*** đồng thời cũng được ký bởi ZSK của domain cha (trong ví dụ này là zone ***.com***). Domain cha có trường là DS (Delegation Signer) trong record phân giải cho domain ***genfic.com.*** Trường này chứa KSK public key đã được băm, và bản băm này được ký sử dụng ZSK key tạo ra chữ ký trong trường RRSIG trên domain cha ***.com.***

Thuật toán tạo chữ ký số: có hai thuật toán tạo key được Cloudflare thay đổi theo thời gian. Độ bảo mật của key phụ thuộc vào kích thước và thuật toán sử dụng. Một số thuật toán dễ phá vỡ hơn và yêu cầu key lớn hơn với cùng mức độ bảo mật với thuật toán khác. Điều này đúng với thuật toán ECDSA khi để phá vỡ ECDSA key yêu cầu phải giả đường cong elliptic ECDLP (Elliptic Curve Discrete Logarithm Problem). Nghĩa là để cung cấp cùng một mức độ bảo mật thì thuật toán ECDSA sử dụng key kích thước nhỏ hơn. Việc này giúp giảm gánh nặng cho CPU khi thao tác chứng thực diễn ra với mức độ dày đặc hàng ngày tại Cloudflare.

Tham khảo thêm tại:

<https://blog.cloudflare.com/ecdsa-the-digital-signature-algorithm-of-a-better-internet/>

### *3.4. Bảo mật DNS – Bot-filtering với Berkeley Packet Filter*

Berkeley Packet Filter cung cấp tap interface kết nối với network interface cho phép đọc gói tin trên interface đó ở mức độ cực kì chi tiết. Nếu như với những công cụ packet capture như tcpdump, việc lọc gói dựa trên những từ khóa như “net”, “port”, “addr”, “src”, “dst” thì có thể coi những từ khóa đó là short cuts đối với BPF. Ví dụ, byte thứ 9 trong IP header là trường embedded protocol, và nếu với giao thức tcp, giá trị này bằng 6. Với tcpdump ta sử dụng từ khóa “tcp” để lọc gói TCP. Còn với BPF, hành động đó tương đương với cú pháp: “ip[9] = 0x06”.

BPF được sử dụng để phân tích lưu lượng, chủ yếu là DNS và sử dụng trên các Edge Server của Cloudflare như một tiện ích nhỏ hỗ trợ cho việc cản lọc lưu lượng bất hợp lệ.

Một số link tham khảo tới BPF:

* <http://www.tcpdump.org/papers/bpf-usenix93.pdf>
* <https://blog.cloudflare.com/bpf-the-forgotten-bytecode/>
* <https://blog.cloudflare.com/introducing-the-bpf-tools/>
* <http://www.infosecwriters.com/text_resources/pdf/JStebelton_BPF.pdf>
* <https://github.com/cloudflare/bpftools>

### *3.5. Global Loab Balancing (GBLB) - Anycast DNS*

### *3.6. Web Content Optimization*

Một trong những tính năng nổi trội của Cloudflare là khả năng cache nội dung trang web (hình ảnh, JS, CSS, HTTML). Do các data center của Cloudflare phân tán trên toàn cầu nên các nội dung đã cache sẽ được điều chuyển nhanh chóng với latency thấp. Nhưng chỉ có 66% nội dung có thể cache là các nội dung tĩnh. Còn lại 34% là các nội dung động thì buộc phải cập nhật từ web server chính thức. **Railgun** được Cloudflare đưa ra để giải quyết vấn đề này để cache các webpage với nội dung động nhằm giảm đi bandwidth sử dụng và thời gian download từ web server chính thức.

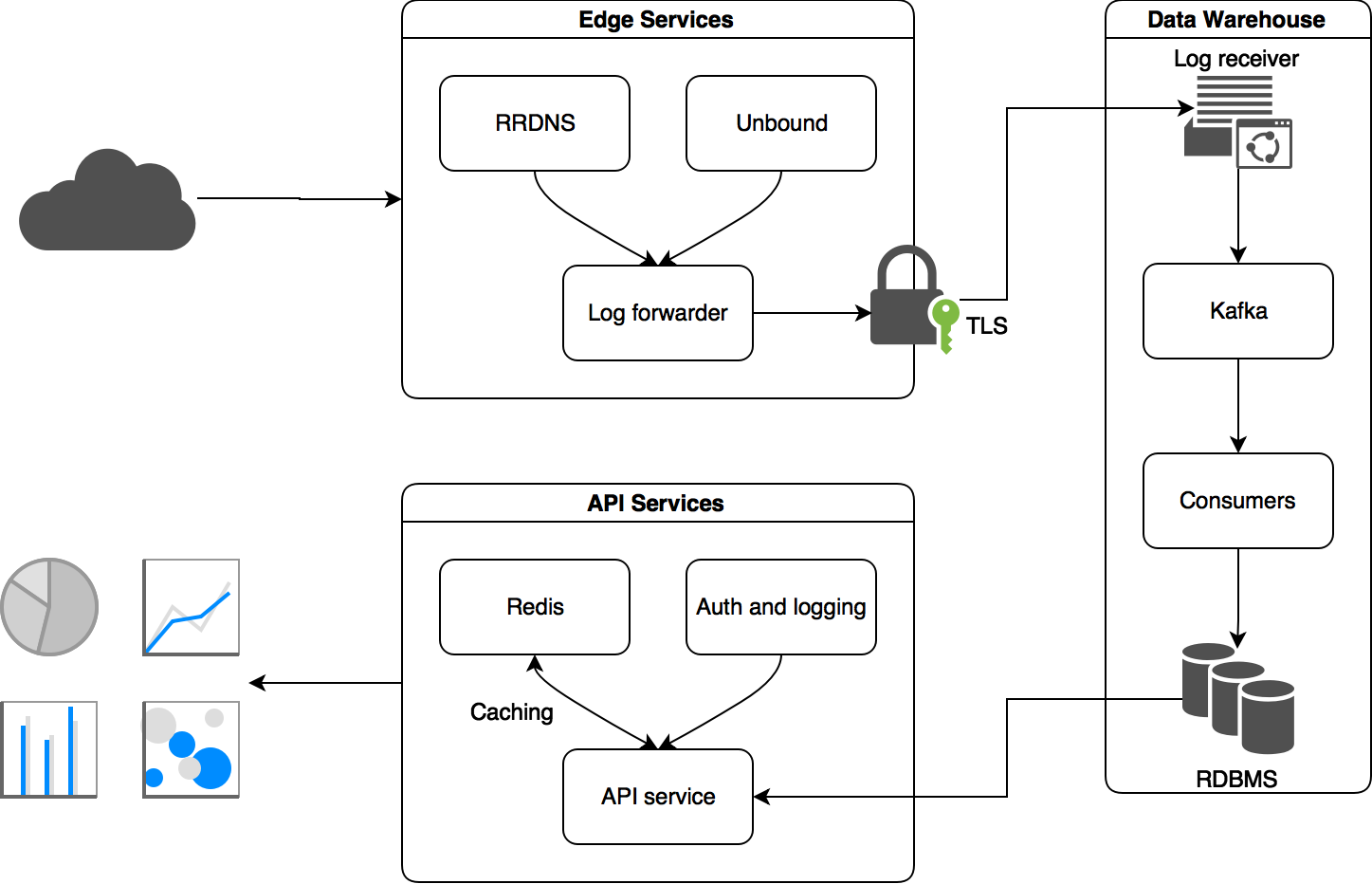
**Railgun** là single daemon chạy trên hệ thống 64 bit sử dụng công nghệ nén để tăng tốc đáng kể hiệu suất qua đường truyền WAN. Railgun dựa vào sự thực rằng markup của websites hoặc body của bản tin JSON API response không thay đổi thường xuyên với mỗi request. Do đó thay vì phải truyền toàn bộ nội dung động giữa web server chính tới các CloudFlare Edge server, Railgun chỉ truyền duy nhất những thay đổi trên markup của trang web giữa các yêu cầu kế tiếp nhau. Điều này giảm đáng kể băng thông sử dụng, thời gian truyền cũng như thời gian load trang.

**Railgun** gồm hai phần. Một phần là **sender** cài đặt tại các Content Server của Cloudflare, phần còn lại là **listener** cài đặt tại máy chủ web của khách hàng. Sender sẽ gửi yêu cầu những nội dung thay đổi từ phía web server. Web server sử dụng listener để lắng nghe các yêu cầu này. Kết nối giữa hai phần này là TCP, bảo mật bởi Transport Layer Security (TLS) .

***Railgun link:*** [***https://www.cloudflare.com/docs/railgun/intro.html***](https://www.cloudflare.com/docs/railgun/intro.html)

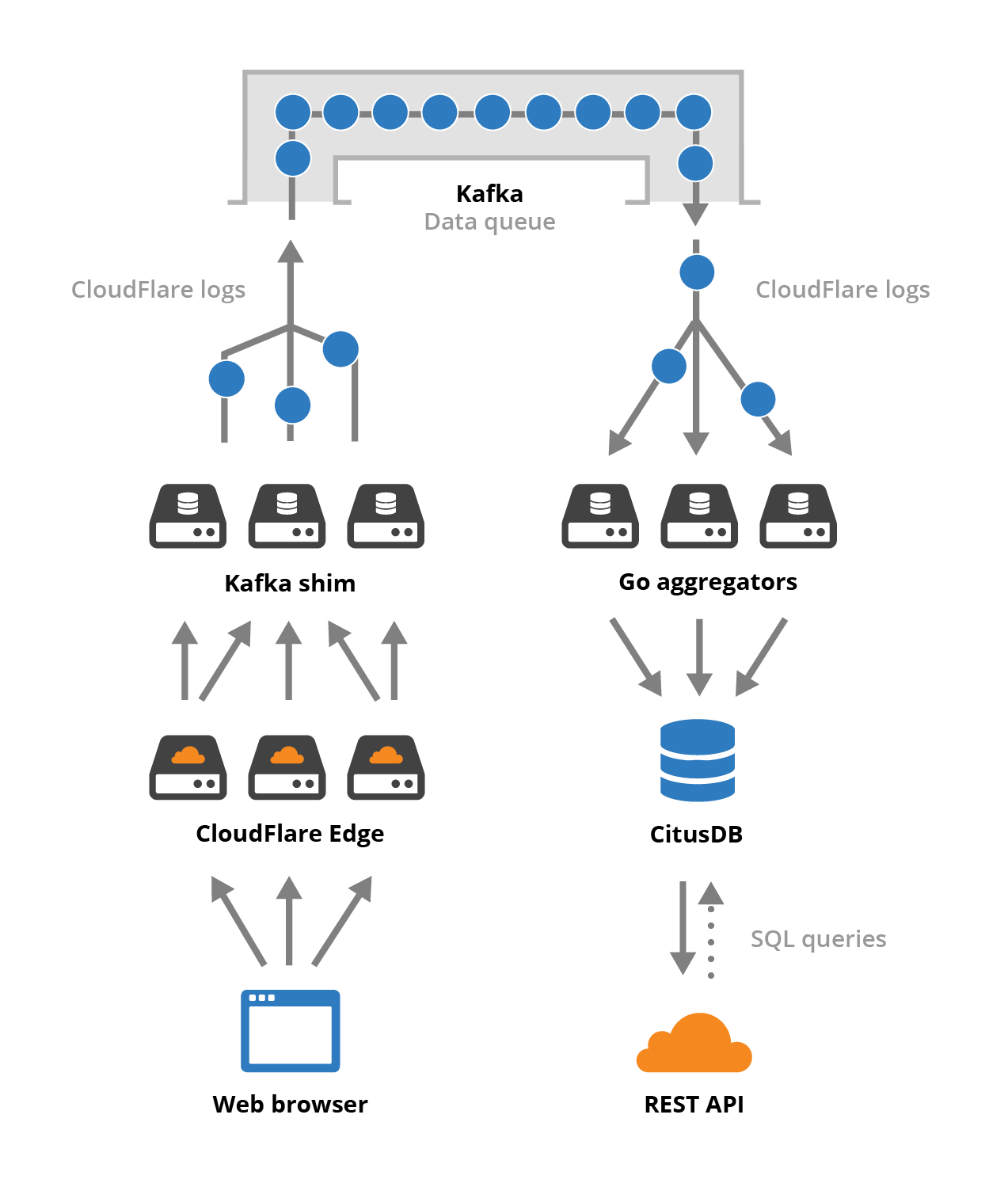
### *3.7. Xử lý dữ liệu và log*

Các công nghệ xử lý dữ liệu và log cho phép CloudFlare lưu trữ và phân tích hàng triệu DNS request mỗi giây. Sau đây là data pipeline của Cloudflare sử dụng để thu thập dữ liệu từ phía các Edge Server và xử lý.



(Tham khảo: [https://blog.cloudflare.com/how-cloudflare-analyzes-1m-dns-queries-per-second](https://blog.cloudflare.com/how-cloudflare-analyzes-1m-dns-queries-per-second/)/)

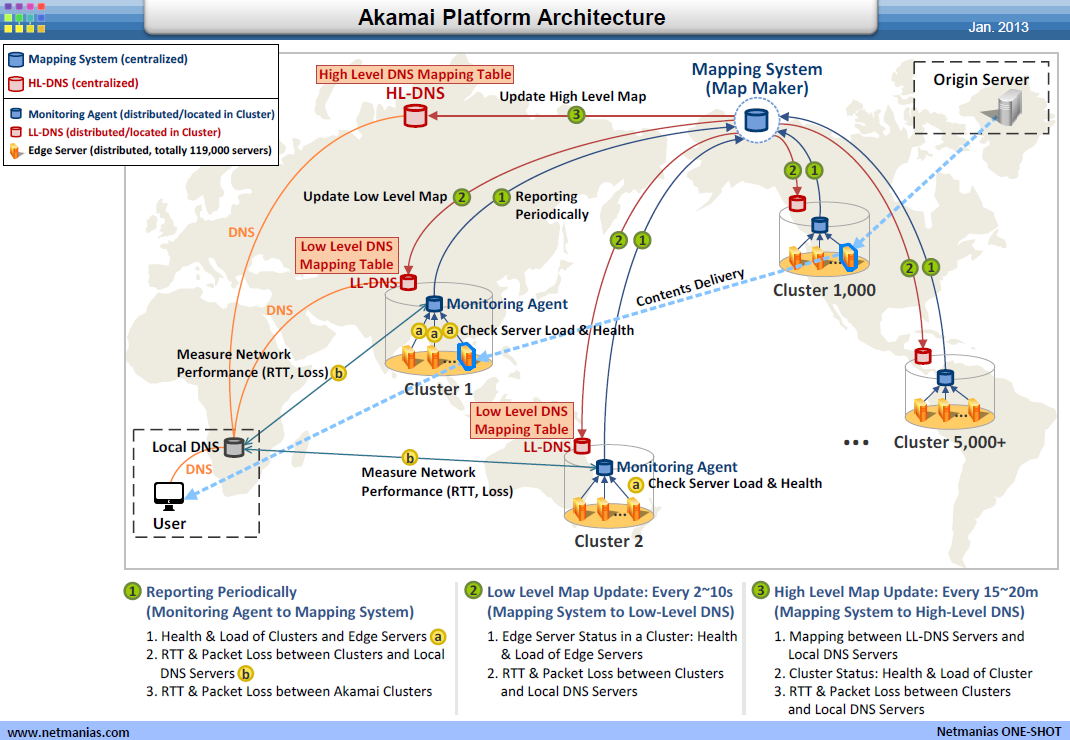
Dưới đây là chi tiết các công nghệ sử dụng để phân tích và xử lý log, bao gồm CitusDB tương ứng với RDBMS trong vùng DataWarehouse ở hình trên và kafka làm hàng đợi xử lý dữ liệu.



(Tham khảo: <https://blog.cloudflare.com/scaling-out-postgresql-for-cloudflare-analytics-using-citusdb/>)

## 4. Dịch vụ DNS của Akamai

### *4.1. Kiến trúc nền tảng của Akamai*

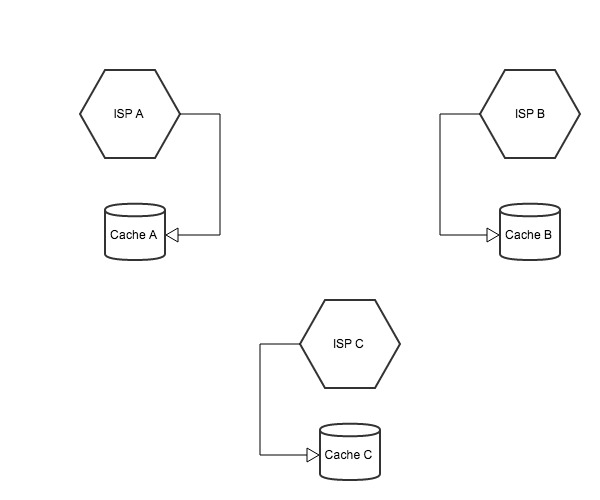


### *4.2. Fast DNS*

### *4.3. Global Load Balancing – DNS-based*

Mạng CDN được thiết kế để cung cấp nội dung (ví dụ nội dung tĩnh của Google) tới các Edge Server nằm trên Edge network gần với các điểm truy cập trên toàn cầu. Để có được khả năng cung cấp nội dung tới các edge server như vậy cần phải thiết kế Global Load Balancing cho các Edge Server, để đảm bảo mọi người dùng có thể truy cập tới CDN node gần họ nhất.

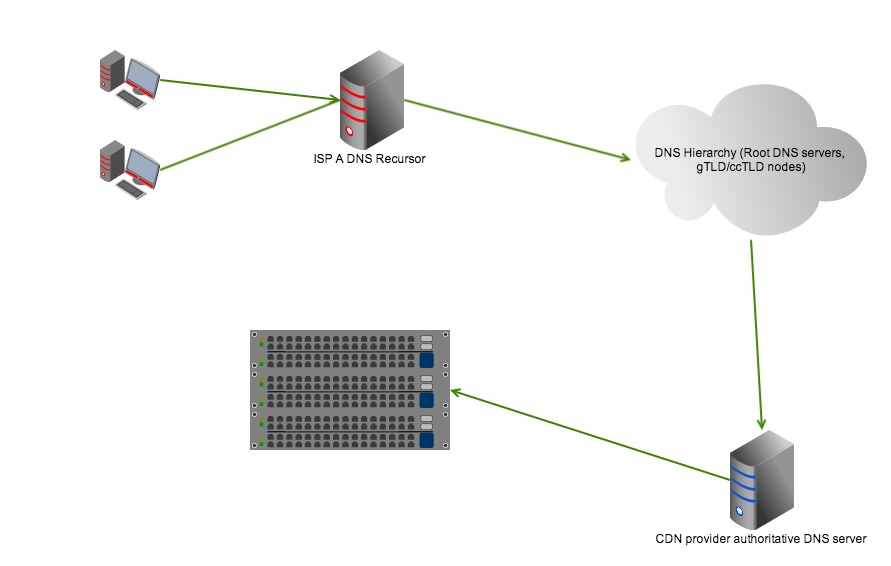
Giả sử dưới đây có 3 Edger Server của một nhà cung cấp dịch vụ CDN cho một web site là ***cdn.website.com***. Server Cache A nằm trong mạng của ISP A, Server Cache B nằm trong mạng của ISP B, Server Cache C nằm trong mạng của ISP C.



Xét cụ thể tình huống này với hai phương pháp load Global Load Balancing phổ biến:

* **Unicast DNS-based:** khi người dùng nằm trong mạng ISP A tìm kiếm trang web ***cdn.website.com***, họ sẽ được trả về địa chỉ unicast của Server Cache A, còn nếu họ nằm trong mạng ISP B thì kết quả trả về là Server Cache B. Để làm được điều này thì người dùng thuộc ISP nào sẽ phải sử dụng DNS Resolver cung cấp bởi ISP đó.Lý do là vì khi yêu cầu tới DNS Resolver của ISP thì nó sẽ tiếp tục hỏi tới các authoritative DNS servers của nhà cung cấp dịch vụ CDN thông qua cây phả hệ DNS. Dựa vào địa chỉ IP của DNS Resolver mà Authoritative server sẽ phản hồi lại địa chỉ của Cache Server nằm trong mạng của ISP vận hành resolver đó.

Phương pháp này có nhược điểm là nếu sử dụng DNS Resolver không phải của ISP như GoogleDNS hay OpenDNS thì kết quả trả về sẽ là địa chỉ của Content Server (cache server) gần nhất với GoogleDNS Resolver mà người dùng truy vấn tới chứ không phải gần nhất với địa chỉ người dùng, vì Authoritative Server không biết được IP của người dùng.

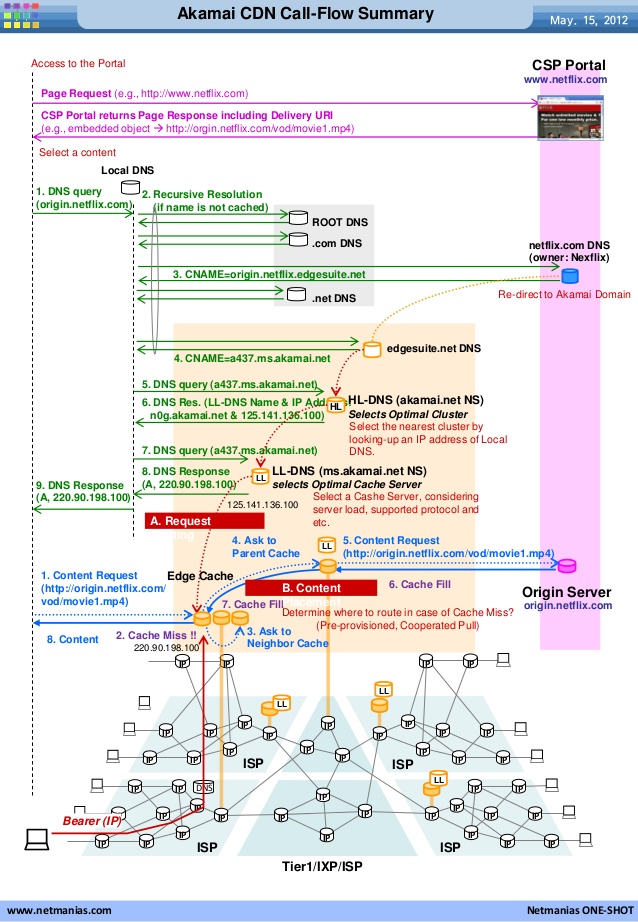
****

* **Anycast routing:** trong trường hợp này thì ba server Cache A, Cache B, Cache C sẽ sử dụng chung một địa chỉ anycast. Khi người dùng truy cập trang ***cdn.website.com***, lưu lượng sẽ được định tuyến tới server “gần nhất”. Giao thức định tuyến sử dụng cho việc tìm ra server “gần nhất” là BGP. Ví dụ ở hình dưới đây, User1 và User2 thuộc ISP A, User3 và User4 thuộc ISP B, User5 và User6 thuộc ISP C. Như vậy, ví dụ với user thuộc ISP A:
* Họ chỉ mất một chặng để tới Cache Server A
* Họ mất hai chặng để tới Cache Server B
* Họ mất ba chặng để tới Cache Server C

Do đó cache server tối ưu nhất được chọn sẽ là Cache Server A.

Phương pháp này cũng có nhược điểm là khái niệm “gần nhất” ở đây không phải là về mặt vị trí địa lý mà gần nhất trên topo mà giao thức BGP chạy, hay hiểu một cách đơn giản là số chặng nhỏ nhất tới mỗi Cache Server. Tuy nhiên số chặng nhỏ nhất không dễ là trường hợp tồi nhất.

Mặc dù phương pháp DNS-based không tối ưu bằng hướng tiếp cận sử dụng AnyCast nhưng Akamai đã áp dụng và có một số cải thiện để khắc phục hạn chế người dùng sử dụng External DNS Resolver như GoogleDNS hay OpenDNS. Dưới đây là CDN Call Flow của Akamai khi sử dụng phương pháp DNS-based.



Đây là ví dụ cụ thể hơn với Các bước xử lý của Akamai CDN như sau:

* Client gửi truy vấn DNS cho tên miền origin.netflix.com lên DNS Resolver (bất kì Resolver nào)
* DNS resolver giả sử không có cache cho origin.netflix.com cũng như netflix.com, nó sẽ truy vấn tới root dns server để hỏi địa chỉ server .com. Root server sẽ trả về cho resolver bản tin referral chứa địa chỉ các dns server .com
* Tiếp đó resolver lại hỏi tiếp tới server .com được lựa chọn để hỏi về server netfix.com và server .com sẽ trả về địa chỉ tương ứng.
* Resolver lúc này truy vấn tới dns server netflix.com được sở hữu bởi Netflix và cho rằng Netflix sử dụng dịch vụ CDN của Akamai. Trên dns server netflix.com sẽ cấu hình một record CNAME để chuyển hướng tới Akamai domain. Giả sử CNAME này như sau: CNAME=origin.netflix.edgesuite.net. CNAME này sẽ được gửi về cho Resolver.
* Lúc này, resolver sẽ truy vấn tới dns server .net và server này sẽ trả về địa chỉ của nameserver edgesuite.net là một nameserver thuộc quản lý của Akamai.
* Resolver truy vấn tới edgesuite.net và server này sẽ trả lời lại với CNAME=a437.ms.akamai.net và địa chỉ của nameserver a437.ms.akamai.net. Đây là high level Akamai DNS server (HL-DNS nameserver) và là một trong những DNS server trên toàn cầu chịu trách nhiệm ủy quyền DNS request sang các low level DNS (LL-DNS nameserver) server phù hợp. Nhìn chung các low level DNS server sẽ gần resolver hơn so với các high level nameserver.
* Resolver truy vấn tới HL-DNS a437.ms.akamai.net. Nameserver này sẽ dựa vào IP của resolver để tìm kiếm LL-DNS nameserver gần nhất và trả về tên và địa chỉ IP tương ứng của nameserver đó. Trong trường hợp này là n0g.akamai.net & 125.141.136.100.
* Resolver gửi dns request tới LL-DNS n0.akamai.net để lựa chọn Cache Server tối ưu, dựa trên việc cân nhắc tải của server, giao thức hỗ trợ, latency,…
* LL-DNS sau khi lựa chọn Cache Server tối ưu sẽ phản hồi lại địa chỉ IP của server đó cho resolver, resolver chuyển tiếp kết quả này về phía trình duyệt người dùng. Sau đó người dùng bắt đầu gửi request tới server này để tìm kiếm tài nguyên.

Sau quá trình DNS look up thành công, khi có yêu cầu tài nguyên từ phía trình duyệt, cache server trước tiên sẽ kiểm tra cache nội bộ xem có tài nguyên được yêu cầu không. Nếu không có thì tìm kiếm sang các server lân cận thuộc cùng cluster. Nếu việc tìm kiếm trong cache không thành công thì cache server sẽ gửi yêu cầu lấy tài nguyên về máy chủ thực sự rồi trả về cho trình duyệt người dùng.

Như đã đề cập, vấn đề của hướng tiếp cận DNS-based là việc máy khách sử dụng external DNS resolver như Google DNS hoặc OpenDNS thay vì resolver của ISP thì việc DNS look up có thể sẽ trả về kết quả không mong đợi.

A more detailed example illustrated below: a user is located in Taiwan and uses a OpenDNS name server in the United States. The CDN would think the user is in the US, and would serve content from the US servers.  This would severely degrade throughput performance, and defeats the benefits of geo load balancing and CDN.



One solution to address the above issue is for the Edge servers to perform request re-mapping, based on the client IP, when the initial server connection is determined to be non-optimal.   
  
When the Edge server receives a request for an object, it first evaluates the round-trip delay time(http://en.wikipedia.org/wiki/Round-trip\_delay\_time )(RTT) of the [TCP SYN-ACK](http://en.wikipedia.org/wiki/Transmission_Control_Protocol)). If the RTT value is low (good), the edge server starts serving the object immediately. If the RTT is above a pre-defined threshold (bad), the Edge server makes a special request to the DNS server, which looks up the client IP and returns a CNAME to the Edge servers that are close to the client IP. The Edge server constructs a fully qualified redirect URL to send to the client. The client will then chase the redirect by resending the request to the newly assigned server.   
  
We conducted a production test to assess the effectiveness of this solution. The feature was enabled randomly on 50% of requests for a high volume, high bit-rate video streaming service.  The test ran for 8 days and performance data was aggregated daily to calculate the average mapping distance (between the client and Edge servers) and the average RTT for requests that triggered remapping.