# SYSTEM

### ▼ HTTP ≠ HTTPS, SSL

- **HTTP**: giao thức không bảo mật, mọi dữ liệu truyền trên internet sẽ không được mã hoá, nghĩa là khi ta submit một form trên trang đó (VD: đăng nhập bằng password, gửi thông tin tài khoản ngân hàng, hay thậm chí các tin nhắn thông thường) thì mọi thông tin bạn nhập và gửi nếu để ai đó trên internet bắt được gói tin, bạn sẽ bị lộ những thông tin đó.
- **HTTPS**: giao thức bảo mật, giúp cho gói tin gửi tới server được mã hoá khi đi trên internet, ai đó có bắt được gói tin họ cũng không thể đọc được thông tin nằm trong gói tin đó.
  - **HTTPS** sử dụng giao thức **SSL (còn gọi là TLS)** để mã hoá dữ liệu của **HTTP**
  - Cách giao thức **SSL** hoạt động: https://blog.daovanhung.com/post/cach-hoat-dong-cua-ssl-va-https

### ▼ stateless ≠ stateful

- **stateless**: là design không lưu dữ liệu của client trên server. Có nghĩa là sau khi client gửi dữ liệu lên server, server thực thi xong, trả kết quả thì "quan hệ" giữa client và server bị "cắt đứt" – server không lưu bất cứ dữ liệu gì của client.
- **stateful**: là một design ngược với stateless, server cần lưu dữ liệu của client, điều đó đồng nghĩa với việc ràng buộc giữa client và server vẫn được giữ sau mỗi request (yêu cầu) của client. Data được lưu lại phía server có thể làm đầu vào (input parameters) cho lần kế tiếp, hoặc là dữ kiện dùng trong quá trình xử lý hay phục vụ cho bất cứ nhu cầu nào phụ thuộc vào bussiness (nghiệp vụ) cài đặt.

### ▼ Vertical scale ≠ Horizontal scale

- **scaling vertically**: mở rộng theo chiều dọc, là cách mở rộng server hiện tại bằng cách nâng cấp độ mạnh (power) bằng cách nâng cấp CPU, Ram, Storage, v.V… Vertical-scaling thường bị giới hạn bởi vượt quá khả năng về cấu hình vật lý hiện đại hay độ trễ khi "chẳng may" Server bị downtime để nâng cấp hay deploy hệ thống.
- **scaling horizontally**: mở rộng theo chiều ngang, là cách mở rộng bằng cách thêm nhiều Node/Server vào một mạng lưới đang có, làm tăng khả năng chịu tải có hệ thống. Cách làm này rẻ và dễ làm hơn so với Vertical-scaling, đặc biệt là rất dễ dàng downsize cũng như upsize hệ thống.

### ▼ CDN

- CDN là một nhóm server đặt tại nhiều vị trái khác nhau để hỗ trợ nội dung được trải dài ở nhiều khu vực vị trí địa lý khác nhau.
- CDN cũng được gọi là "distribution networks". Ý tưởng là tạo được nhiều điểm truy cập (Point of Presence – PoPs) ngoài server gốc. Việc này giúp website quản lý tốt traffic hơn bằng cách xử lý

nhanh hơn yêu cầu của khách, tăng trải nghiệm người dùng.

- Cách CDN hoạt động:
  - Trong mạng lưới Content Delivery. Mỗi điểm hiện diện (location) được gọi là một **PoPs**.
  - Để tăng thời gian phản hồi giữa client và server (người dùng và trang web), các PoPs (node trong mạng lưới) sẽ **lưu nội dung trang web vào bộ nhớ (cached)** của mình và làm mới nó thường xuyên.
  - Khi người dùng yêu cầu nội dung trang web, người dùng sẽ không trực tiếp truy cập tới trang web (ở bờ Tây nước Mỹ chẳng hạn) mà chỉ truy cập với một điểm CDN **gần mình nhất**.
- Khi nào nên sử dụng?
  - Nếu content của bạn chỉ có **một lượng nhỏ truy cập** ở vị trí địa lí **gần nơi đặt máy chủ**, không cần thiết phải dùng CDN.
  - Ngược lại, nếu nội dung của bạn được truy cập và sử dụng ở khắp nơi trên thế giới. Đăng kí tham gia mạng lưới CDN là cần thiết giúp **tăng trải nghiệm người dùng**.

## ▼ DNS

- DNS là viết tắt của Domain Name System tạm dịch là "Hệ thống phân giải tên miền". Về bản chất cách để máy tính truy cập được một trang web là nhờ địa chỉ IP.
- Ví dụ bạn muốn truy cập vào **google.com** thì tức là browser đang request tới IP máy chủ của google.
- Tuy nhiên có cả triệu website và bạn phải nhớ địa chỉ IP của từng trang web, điều đó là bất khả thi và chưa kể trường hợp địa chỉ IP của trang web đó có thể thay đổi liên tục. Đó cũng chính là nguyên nhân mà DNS được sinh ra.
- DNS sẽ đóng vai trò như một cuốn danh bạ, thay vì phải nhớ 1 dãy IP loằng ngoằng thì bạn sẽ nhớ đến tên miền của trang web đó
- Ví dụ như **google.com** và tất nhiên như vậy sẽ thân thiện với người sử dụng hơn, và DNS sẽ có vai trò là phân giải tên miền thành địa chỉ IP tương ứng nhờ đó browser có thể gửi request tới server.

## ▼ Load balancer

- Load Balancing là quá trình của việc phân phối lưu lượng truy cập một cách hiệu quả thông qua nhiều server hay còn được gọi là `server farm` hay `server pool`
- Việc phân phối đồng một cách đồng đều sẽ cải thiện khả năng đáp ứng và tăng tính khả dụng của các ứng dụng.
- Phương pháp này ngày càng cần thiết vì các ứng dụng ngày nay đã phức tạp hơn, cùng với nhu cầu của người sử dụng tăng và lưu lượng truy cập tăng lên.
- Load balancer đã giải quyết được các vấn đề như:
  - Performance: dễ dàng scale up (vertical scaling) và scale out (horizontal scaling)

    - Availability: có server dự phòng và cơ chế tự động khôi phục. Nếu 1 server bị lỗi sẽ không ảnh hưởng đến toàn bộ hệ thống
    - Economy: triển khai một server có hiệu năng lớn thì đắt hơn so với một cụm server có hiệu năng nhỏ. Chi phí để duy trì một cụm server nhỏ thì rẻ hơn và dễ dàng thêm hoặc nâng cấp server trong cụm này so với việc nâng cấp và thay thế một server lớn
- Load balancer có 3 kiến trúc cơ bản:
    - Dựa trên DNS
    - Dựa trên phần cứng
    - Dựa trên phần mềm
- Các thuật toán cơ bản:
    - Round Robin
    - Weighted Round Robin
    - Dynamic Round Robin
    - Fastest
    - Least Connections

**Link**: https://anonystick.com/blog-developer/load-balancer-neu-ban-khong-hieu-khong-sao-nhung-neu-ban-la-mot-ky-su-thi-khong-the-khong-hieu-202006243445464

▼ **Nginx**

- là một máy chủ web server, open source, được thiết kế hướng đến mục đích cải thiện tối đa hiệu năng và sự ổn định.
- Ngoài ra nhờ vào khả năng của máy chủ http, nginx còn có thể hoạt động như proxy server cho email, reverse proxy, http caching hay load balancer cho các máy chủ http, tcp, udp
- Nginx được sử dụng kiến trúc đơn luồng và event driven (hướng sự kiện) vì thế nó hiệu quả hơn apache server nếu cấu hình chính xác.
- NGINX được cấu hình theo kiểu bất đồng bộ (asynchronous): nghĩa là 1 NGINX process có thể xử lý nhiều request liên tục, dựa vào số lượng tài nguyên còn lại của hệ thống.
- Nhờ kiểu cấu hình như vậy, NGINX có thể “nhúng” các file lập trình (như .php) vào process riêng của nó. Nghĩa là mọi request yêu cầu data được 1 process riêng của NGINX thực hiện, và trả data lại cho client bằng reverse proxy.
- Bên cạnh đó, đối với những file tĩnh (file .txt, file .css hay các file hình ảnh), NGINX sẽ trả dữ liệu mà không cần sự can thiệp của các module server side.

▼ **TCP ≠ UDP**

- **TCP** là giao thức truyền tải hướng kết nối (connection-oriented), nghĩa là phải thực hiện thiết lập kết nối với đầu xa trước khi thực hiện truyền dữ liệu. Tiến trình thiết lập kết nối ở TCP được gọi

là tiến trình ***bắt tay 3 bước*** (*threeway handshake*).

- Cung cấp cơ chế báo nhận (Acknowledgement) :Khi A gửi dữ liệu cho B, B nhận được thì gửi gói tin cho A xác nhận là đã nhận. Nếu không nhận được tin xác nhận thì A sẽ gửi cho đến khi B báo nhận thì thôi.

- **UDP** là giao thức truyền tải hướng không kết nối (connectionless). Nó sẽ không thực hiện thao tác xây dựng kết nối trước khi truyền dữ liệu mà thực hiện truyền ngay lập tức khi có dữ liệu cần truyền (kiểu truyền best effort) => truyền tải rất nhanh cho dữ liệu của lớp ứng dụng.
    - Không đảm bảo tính tin cậy khi truyền dữ liệu và không có cơ chế phục hồi dữ liệu ( nó không quan tâm gói tin có đến đích hay không, không biết gói tin có bị mất mát trên đường đi hay không) => dễ bị lỗi.

https://viblo.asia/p/tim-hieu-giao-thuc-tcp-va-udp-jvEla11xlkw

▼ **Quy tác bắt tay 3 bước**

```
* Gói dữ liệu với cờ SYN (synchronization <=> Sự đồng bộ) dùng để bắt đầu một connection.
* ACK (acknowledgement <=> Xác nhận).
* FIN (finish <=> hoàn thành) dùng để ngắt một connection.
* ...
```

- **B1**:
    - SYN: các chương trình máy con (ví dụ yêu cầu từ browser, ftp client) bắt đầu connection với máy chủ bằng cách gửi một packet với cờ "SYN" đến máy chủ.
    - SYN packet này thường được gửi từ các cổng cao (1024 - 65535) của máy con đến những cổng trong vùng thấp (1 - 1023) của máy chủ.
    - Chương trình trên máy con sẽ hỏi hệ điều hành cung cấp cho một cổng để mở connection với máy chủ.
    - Những cổng trong vùng này được gọi là "cổng máy con" (client port range).
    - Tương tự như vậy, máy chủ sẽ hỏi HĐH để nhận được quyền chờ tín hiệu trong máy chủ, vùng cổng 1 - 1023.
    - Vùng cổng này được gọi là "vùng cổng dịch vụ" (service port).

```
- Ví dụ (mặc định):
    - Web Server sẽ luôn chờ tín hiệu ở cổng 80 và Web browser sẽ connect vào cổng 80 của máy chủ.
    - FTP Server sẽ lắng ở port 21.

 Ngoài ra trong gói dữ liệu còn có thêm địa chỉ IP của cả máy con và máy chủ.
```

- **B2**:
    - SYN/ACK: khi yêu cầu mở connection được máy chủ nhận được tại cổng đang mở, server sẽ gửi lại packet chấp nhận với 2 bit cờ là SYN và ACK.

- SYN/ACK packet được gửi ngược lại bằng cách đổi hai IP của server và client, client IP sẽ thành IP đích và server IP sẽ thành IP bắt đầu. Tương tự như vậy, cổng cũng sẽ thay đổi, server nhận được packet ở cổng nào thì cũng sẽ dùng cổng đó để gửi lại packet vào cổng mà client đã gửi.
- Server gửi lại packet này để thông báo là server đã nhận được tín hiệu và chấp nhận connection, trong trường hợp server không chấp nhận connection, thay vì SYN/ACK bits được bật, server sẽ bật bit RST/ACK (Reset Acknowledgement) và gởi ngược lại RST/ACK packet.
- Server bắt buộc phải gửi thông báo lại bởi vì TCP là chuẩn tin cậy nên nếu client không nhận được thông báo thì sẽ nghĩ rằng packet đã bị lạc và gửi lại thông báo mới.

- **B3**:
  - ACK: khi client nhận được SYN/ACK packet thì sẽ trả lời bằng ACK packet.
  - Packet này được gởi với mục đích duy báo cho máy chủ biết rằng client đã nhận được SYN/ACK packet và lúc này connection đã được thiết lập và dữ liệu sẽ bắt đầu lưu thông tự do.
  - Đây là tiến trình bắt buộc phải thực hiện khi client muốn trao đổi dữ liệu với server thông qua giao thức TCP.
  - Một số thủ thuật dựa vào đặc điểm này của TCP để tấn công máy chủ (ví dụ DOS).

## ▼ Reverse Proxy

- Reverse proxy là một loại proxy server trung gian giữa một máy chủ và các client gửi tới các yêu cầu. Nó kiểm soát yêu cầu của các client, nếu hợp lệ, sẽ luân chuyển đến các server thích ứng.
- Trái ngược với một **forward proxy**, là một trung gian cho phép các client liên hệ với nó liên lạc với bất kỳ máy chủ ảo nào, reverse proxy là một trung gian cho các máy chủ liên hệ với nó được liên lạc bởi bất kỳ client nào.
- Ưu điểm lớn nhất của việc sử dụng reverse proxy là khả năng quản lý tập trung. Nó giúp kiểm soát mọi request do client gửi lên các server được bảo vệ.
- Reverse proxy server được dùng để làm gì?
  - Reverse proxy ở giữa client và network service, như là website. Một số tính năng mà nó mang lại như:
    - Bảo mật
    - Load balancing
    - Tăng tốc độ trang web

https://viblo.asia/p/reverse-proxy-server-la-gi-eW65GW4P5DO

## ▼ Cluster, Node, Container

- **Clustering** chính là 1 kiến trúc được tạo ra với mục đích đảm bảo nâng cao khả năng sẵn sàng cho những hệ thống mạng. Clustering bao gồm những server riêng lẻ được kết nối với nhau đồng thời hoạt động lại cùng nhau trong 1 hệ thống. Những server này giao tiếp với nhau với mục đích trao đổi thông tin và giao tiếp với cả những mạng bên ngoài để thực hiện những yêu cầu. Trong trường hợp có lỗi xảy ra những dịch vụ trong cluster hoạt động tương tác với nhau để duy trì tính ổn định và độ sẵn sàng cao cho hệ thống.
- **Node** là những server con trong cụm cluster
- **Container** là nơi chứa ứng dụng hoặc service của chúng ta

▼ **Concurrent, latency, consistency**

- **Latency** được biết tới như là khoảng thời gian từ lúc chúng ta yêu cầu tải trang web cho tới khi thật sự nhìn thấy nội dung trên trang web đó.

▼ **10 mẹo cải thiện hiệu suất máy chủ**

- Sử dụng máy chủ proxy reverse để tăng tốc và bảo vệ các ứng dụng
- Load balancing
- Cache và content
- Nén dữ liệu
- Tối ưu hóa SSL / TLS
- Triển khai HTTP / 2 hoặc SPDY
- Cập nhật phiên bản phần mềm liên tục
- Tối ưu hóa hiệu suất Linux
- Tối ưu hóa hiệu suất của máy chủ Web
- Giám sát các hoạt động thời gian thực để giải quyết các vấn đề và tắc nghẽn
- https://anonystick.com/blog-developer/10-meo-de-cai-thien-hieu-suat-cua-cac-ung-dung-web-len-10-lan-2020051556310698#t-2

▼ **Notification system**

- **External software** sẽ send một JSON message qua https với message data giống như address, type, message, ...
- **Rate limiter** validate những rules để bảo vệ system khỏi bị overload và những vấn đề về bảo mật
- Notification service sẽ nhận message và chuyển message đó đến với những message queue tương ứng và sau đó sẽ writes một số logs về notification xuống **Data cache cluster**
- Các **workers** sẽ lấy message trong queue và connect đến với third part software để send message đến các thiết bị tương ứng
- **Third part software** sẽ call back bằng cách sử dụng web hook để nhận status và một số thông tin của message

- Sau đó có những **workers** sẽ chạy và lấy những thông tin response và lưu trữ vào **data store**
- Thông tin về status và analytics data lúc này sẽ có sẵn trong **Notification status & analytics service** và ta có thể truy cập service này từ bên ngoài với **external service**

⇒ **Lợi ích:**

- **Reliability**: giảm thiểu lỗi
- **Security**: chạy trên https với appKey và appSecret để đảm bảo user đã authorized mới có thể send message
- **Tracking and monitoring**: logs, status và analytics data sẽ được lưu trữ trên db có thể truy vấn cách dễ dàng
- **Rate limiting**: bảo vệ hệ thống khỏi overload, và những vấn đề bảo mật không mong muốn

## ▼ HTTP status codes

- 1xx Informational
- 2xx Succesful
- 3xx Redirection
- 4xx Client Error
- 5xx Server Error

## ▼ Xử lý đồng thời và xử lý song song

https://zalopay-oss.github.io/go-advanced/ch1-basic/ch1-05-concurrency-parallelism.html

- **Xử lý đồng thời** là khả năng phân chia và điều phối nhiều tác vụ khác nhau trong cùng một khoảng thời gian và tại một thời điểm chỉ có thể xử lý một tác vụ. Khái niệm này trái ngược với **xử lý tuần tự** (sequential processing).
  - **Xử lý tuần tự** là khả năng xử lý chỉ một tác vụ trong một khoảng thời gian, các tác vụ sẽ được thực thi theo thứ tự hết tác vụ này sẽ thực thi tiếp tác vụ khác.
- **Xử lý song song** là khả năng xử lý nhiều tác vụ khác nhau trong cùng một thời điểm, các tác vụ này hoàn toàn độc lập với nhau.
  - Xử lý song song chỉ có thể thực hiện trên máy tính có số nhân lớn hơn 1.
  - Thay vì một nhân CPU chúng ta chỉ có thể xử lý một tác vụ nhỏ tại một thời điểm thì khi số nhân CPU có nhiều hơn chúng ta có thể xử lý các tác vụ song song với nhau cùng lúc trên các nhân CPU.

## ▼ Processes & Threads

- **Process**
  - Tiến trình có thể hiểu đơn giản là một chương trình đang chạy trong máy tính.

- Khi chúng ta mở trình duyệt và truy cập một trang web thì đây được xem là một tiến trình.
- Khi chúng ta viết 1 chương trình máy tính bằng ngôn ngữ lập trình như C, Java, hay Go, sau khi tiến hành biên dịch và chạy chương trình thì hệ điều hành sẽ cấp cho chương trình một không gian bộ nhớ nhất định, PID (process ID),...
- Mỗi tiến trình có ít nhất một luồng chính (main thread) để chạy chương trình, nó như là xương sống của chương trình vậy. Khi luồng chính này ngừng hoạt động tương ứng với việc chương trình bị tắt.

- **Thread**
  - Thread hay được gọi là tiểu trình nó là một luồng trong tiến trình đang chạy.
  - Các luồng được chạy song song trong mỗi tiến trình và có thể truy cập đến vùng nhớ được cung cấp bởi tiến trình, các tài nguyên của hệ điều hành,...
  - Các thread trong process sẽ được cấp phát riêng một vùng nhớ `stack` để lưu các biến riêng của thread đó.
  - Stack được cấp phát cố định khoảng `1MB-2MB`. Ngoài ra các thread chia sẻ chung vùng nhớ `heap` của process.
  - Khi process tạo quá nhiều thread sẽ dẫn đến tình trạng stack overflow. Khi các thread sử dụng chung vùng nhớ sẽ dễ gây ra hiện tượng race condition.

https://zalopay-oss.github.io/go-advanced/ch1-basic/ch1-05-concurrency-parallelism.html

https://stackoverflow.com/questions/200469/what-is-the-difference-between-a-process-and-a-thread/200543#200543

## ▼ CI/CD

- **CI** là Continuous Integration. Nó là phương pháp phát triển phần mềm yêu cầu các thành viên của team tích hợp công việc của họ thường xuyên, mỗi ngày ít nhất một lần.
  - Mỗi tích hợp được "build" tự động (bao gồm cả test) nhằm phát hiện lỗi nhanh nhất có thể.
  - Cả team nhận thấy rằng cách tiếp cận này giảm thiểu vấn đề tích hợp và cho phép phát triển phần mềm nhanh hơn.
- Nếu CI đảm nhận nhiệm vụ xây dựng và kiểm tra một cách tự động thì CD lại có nhiệm vụ cao hơn một chút. CD được viết tắt bởi Continuous Delivery - chuyển giao liên tục.
  - Đây là quá trình nâng cao hơn chút đó là kiểm tra tất cả những thay đổi về code đã được build và code trong môi trường kiểm thử.
  - CD cho phép các lập trình viên tự động hóa phần mềm testing, kiểm tra phần mềm qua nhiều thước đo trước khi triển khai.
  - Những bài test này có thể bao gồm UI testing, integration testing, API testing,... CD sử dụng Deployment Pipeline giúp chia quy trình chuyển giao thành các giai đoạn.

    - Mỗi giai đoạn có những mục tiêu riêng để xác minh chất lượng của các tính năng từ một góc độ vô cùng khác để có thể kiểm định được chức năng và tránh những lỗi phát sinh ảnh hưởng đến người dùng.
- Trên thực tế thì **CI/CD** là một quy trình làm việc, code của bạn sẽ được build test và sau đó deploy trên server hoặc cloud một cách tự động luôn.

▼ **Docker**

- **Khái niệm**:
    - Docker là một nền tảng để cung cấp cách để building, deploying và running ứng dụng dễ dàng hơn bằng cách sử dụng các containers (trên nền tảng ảo hóa) để đóng gói ứng dụng.
    - Docker sử dụng công nghệ ảo hóa containerization để triển khai các ứng dụng vào trong container ảo hóa.
    - Docker sử dụng nhân kernel linux để chạy các container, trên hệ điều hành Linux, Docker có thể sử dụng trực tiếp nhân của máy host; còn với các hệ điều hành Windows, MacOS – có thể vì lý do bảo mật nên docker không thể trực tiếp xài chung kernel với các hệ điều hành này nên trên các hệ điều hành này docker sẽ tạo ra một máy ảo virtual guest với nhân linux để chạy các container.
    - Container là đơn vị phần mềm cung cấp cơ chế đóng gói ứng dụng, mã nguồn, thiết lập, thư viện... vào một đối tượng duy nhất. Ứng dụng sau khi được đóng gói có thể hoạt động một cách nhanh chóng và hiệu quả trên các môi trường điện toán khác nhau. Từ đó nó có thể tạo ra một môi trường hoàn hảo nơi mà có mọi thứ để chương trình có thể hoạt động được, không chịu sự tác động từ môi trường của hệ thống cũng như không làm ảnh hưởng ngược lại về phía hệ thống chứa nó.
- **Docker gồm có 3 thành phần chính:**
    - **Docker file: l**à một file dạng text không có phần đuôi mở rộng, chứa các đặc tả về một trường thực thi phần mềm, cấu trúc cho Docker image. Docker image có thể được tạo ra tự động bằng cách đọc các chỉ dẫn trong Dockerfile. Từ những câu lệnh đó, Docker sẽ build ra Docker image
    - **Image:** là 1 đơn vị đóng gói chứa mọi thứ cần thiết để 1 ứng dụng chạy. Image được tạo thành từ nhiều layer xếp chồng lên nhau, bên trong image là 1 hệ điều hành bị cắt giảm và tất cả các phụ thuộc (dependencies) cần thiết để chạy 1 ứng dụng.
    - **Container:** là đơn vị phần mềm cung cấp cơ chế đóng gói ứng dụng, mã nguồn, thiết lập, thư viện... vào một đối tượng duy nhất. Ứng dụng sau khi được đóng gói có thể hoạt động một cách nhanh chóng và hiệu quả trên các môi trường điện toán khác nhau. Từ đó nó có thể tạo ra một môi trường hoàn hảo nơi mà có mọi thứ để chương trình có thể hoạt động được, không chịu sự tác động từ môi trường của hệ thống cũng như không làm ảnh hưởng ngược lại về phía hệ thống chứa nó.

Mỗi container bao gồm mọi thứ cần thiết để chạy được nó: code, runtime, system tools, system libraries, setting. Mỗi container như 1 hệ điều hành thực sự, bên trong mỗi container sẽ chạy 1 ứng

dụng

▼ **How work when clicking on a URL on the browser?**

Làm thế nào một trang web khi truy cập bằng địa chỉ url (Ví dụ: https://google.com/) lại có thể hiển thị được. Khi anh em bật trình duyệt lên, nhập địa chỉ một trang web (Ví dụ: https://google.com/) và bấm Enter, vài giây sau nội dung của trang web sẽ được hiện ra trên trình duyệt. Quá trình đó được tóm tắt như sau:

1. Từ tên miền (Ví dụ: https://google.com/) máy của anh em sẽ sử dụng DNS để tìm ra địa chỉ IP thực sự của web server tương ứng chứa website có tên miền đó (Ví dụ: https://google.com/ tương ứng với IP: 142.251.10.99).
2. Sau khi đã tìm được địa chỉ IP, trình duyệt sẽ gửi gói tin yêu cầu – HTTP request đến địa chỉ của web server, yêu cầu trả về nội dung trang web. Gói tin yêu cầu đó cũng như tất cả các gói tin, dữ liệu khác trao đổi giữa máy chủ với máy chúng ta (gọi là máy khách) được thực hiện qua một bộ giao thức TCP/IP.
3. Khi nhận được yêu cầu từ máy khách, máy chủ sẽ tiến hành trả về các tập tin HTML, CSS, JS,... để hiển thị trên trình duyệt. Các tập tin này có thể được chia thành nhiều gói tin (packets) nhỏ và gửi về cho trình duyệt của người dùng đang ở máy khách.
4. Khi nhận được, trình duyệt sẽ ghép những gói tin nhỏ nhận được thành những tập tin hoàn chỉnh và hiển thị lên màn hình. Như thế là chúng ta có một trang web hoàn chỉnh.