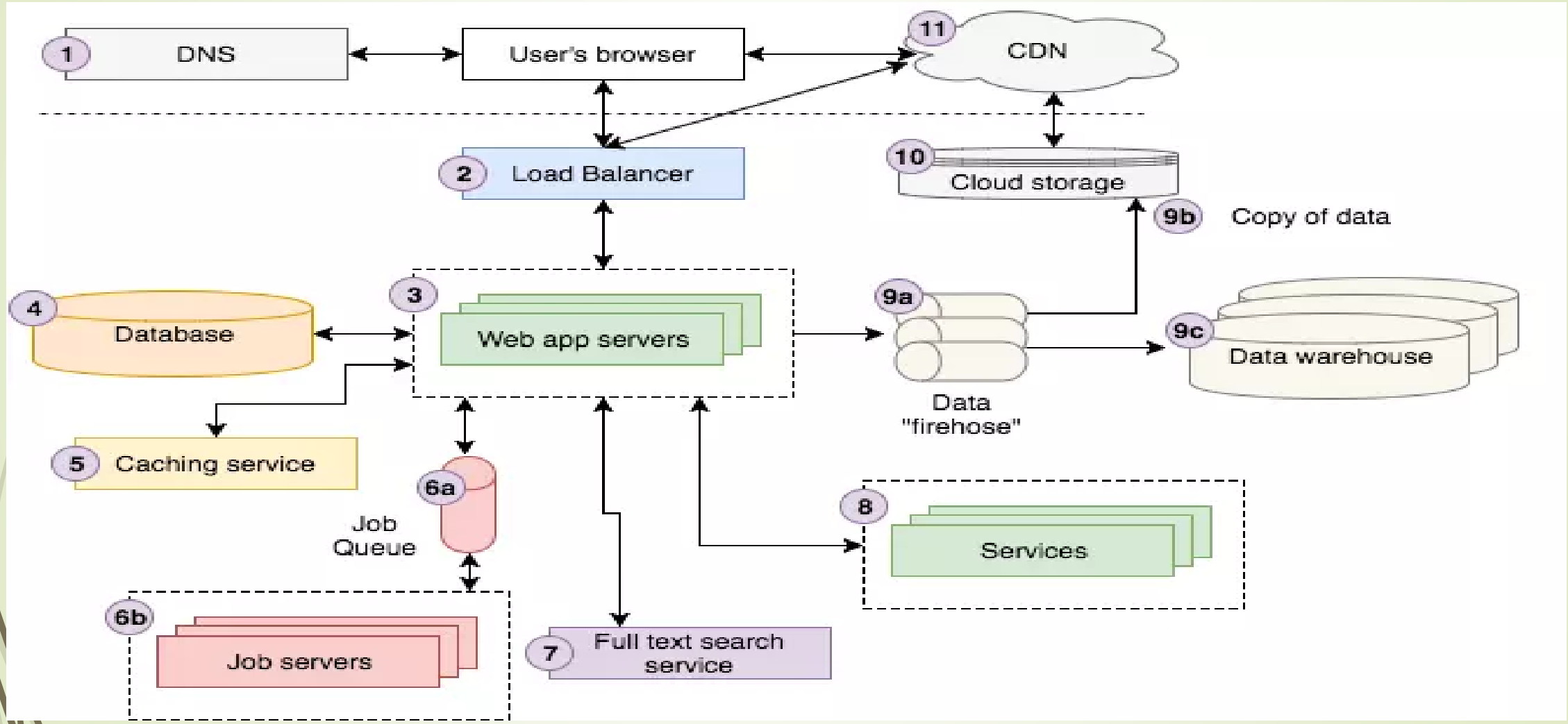
HỆ THỐNG VÀ CÔNG NGHỆ WEB

ThS. Nguyễn Thị Hồng Yến

WEB APPLICATION ARCHITECTURE

Web Application Architecture 101

Kiến trúc của web ứng dụng hiện đại



CÂU HỎI

- **Điều gì sẽ xảy ra từ khi nhập một tên miền vào trình duyệt đến khi trang được hiển thị?**
  - Trình duyệt gửi request đến DNS
  - DNS đưa ra địa chỉ IP máy chủ và gửi tiếp request đến đó
  - Request đến được với máy chủ, sẽ đi qua một con Load balancer, chọn 1 trong nhiều replicas web server để xử lý request này.
  - Web server thực hiện render giao diện HTML và gửi nó quay trở lại cho trình duyệt của bạn.
  - Trang HTML này sẽ chứa Javascript, CSS,.. và chúng được trình duyệt tải về từ các service CDN, các đường dẫn (thứ mà được cung cấp trong trang HTML trên).
  - Cuối cùng thì trình duyệt sẽ kết hợp những thứ trên và hiển thị trang web tới bạn.

1. DNS

- DNS: Domain Name System, giúp "phân giải" các domain (tên miền) thành các địa chỉ máy tính (IP) trên mạng.
- Là 1 công nghệ cốt lõi để tạo nên mạng lưới website trên thế giới (world wide web). DNS cung cấp 1 cặp key / value để tìm kiếm tên miền tới địa chỉ IP, nó giúp máy tính có thể điều hướng request đi tới chính xác máy chủ của ứng dụng web.
- Ví dụ: gọi điện thoại qua danh bạ

Scaling

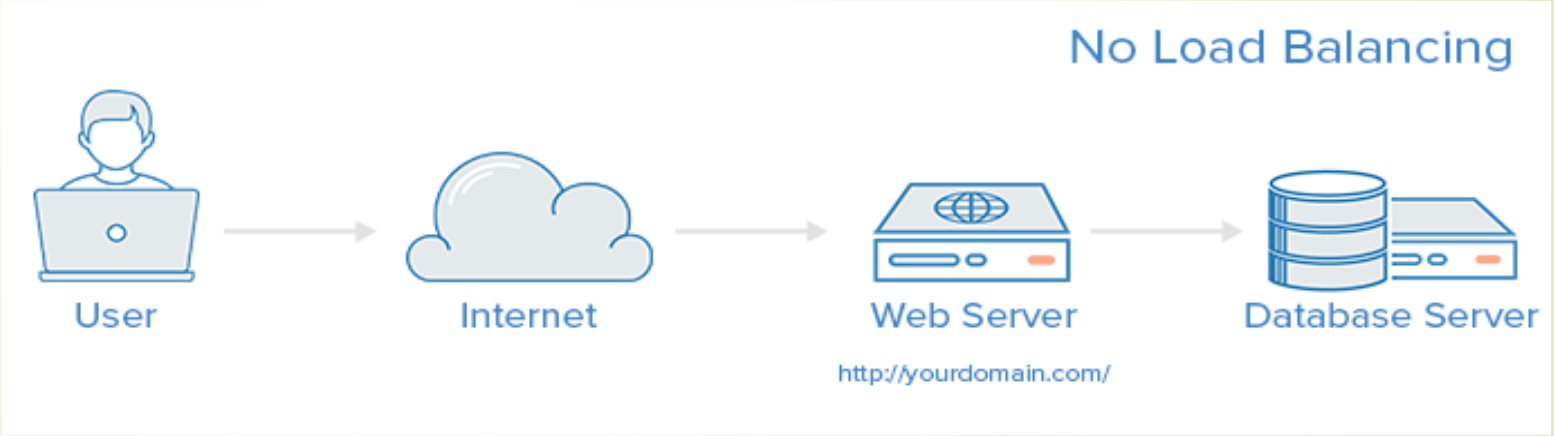
- Theo chiều dọc:
  - Tăng cường khả năng phục vụ của server bằng cách nâng cấp:
    - thêm RAM
    - thêm CPU
    - Thêm ổ cứng
  - Tiết kiệm tiết kiệm rất nhiều thời gian và hầu như không thay đổi kiến trúc code cũng như mô hình mạng hiện tại bởi số lượng server không thay đổi. Tuy nhiên, scale theo cách này có thể chi phí theo hàm mũ
  - Ví dụ:
    - nâng cấp từ 128GB lên 256GB không phải lúc nào giá cũng gấp đôi
    - Khi lên tới ngưỡng hỗ trợ của máy.

Scaling

- Theo chiều ngang
  - bổ sung thêm máy tính vào mạng để tăng khả năng phục vụ của hệ thống
  - không cần phải sở hữu những con server mạnh, quan trọng là giá cả phù hợp, ổn định và có thể mua nhiều server đồng dạng để giúp hệ thống đồng bộ phần cứng, tránh các rủi ro do trong một mạng có quá nhiều dòng thiết bị khác nhau.

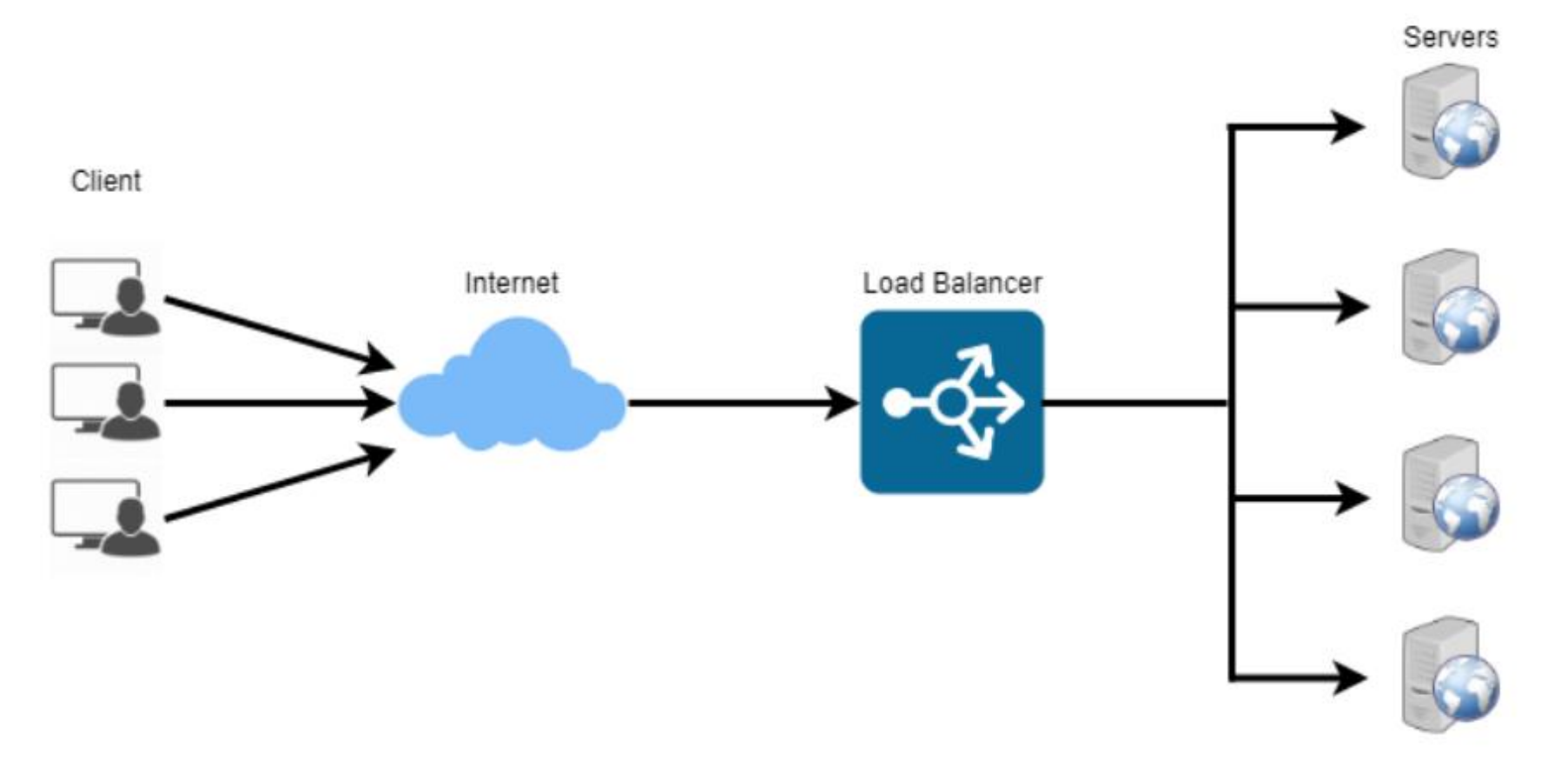
2. Load balancing

- Load balancing là một thành phần quan trọng của cơ sở hạ tầng, được sử dụng để cải thiện hiệu suất và độ tin cậy của các trang web, các ứng dụng, cơ sở dữ liệu và các dịch vụ khác bằng cách phân phối khối lượng công việc trên nhiều máy chủ.
- Đã học và làm:



Load Balancing

- Bộ cân bằng tải



Load Balancing

- Load Balancer đơn giản là một hệ thống (phần mềm, thiết bị chuyên dụng…) hỗ trợ việc chia tải trong trường hợp bạn có nhiều server có vai trò ngang nhau (giữa các web server) hoặc vai trò khác nhau (giữa các web server và database server).
- Ví dụ: có 5 server và bạn muốn khi có người truy cập vào hệ thống thì load balancer tự động điều hướng xử lý đến các server đã được chỉ định này
- Phát hiện các node (server) bị chết để cách ly không điều hướng truy vấn tới server này, và tự động điều hướng lại khi server này “sống” trở lại.

Load balancing

- Hướng tiếp cận
  - Phần mềm
    - cài đặt lên các server trong mạng và làm nhiệm vụ điều hướng tới các server được chỉ định trong mạng khi có truy vấn từ ngoài mạng. Cách này hiện được ưa chuộng hơn bởi không cần mua các thiết bị đắt đỏ và có thể tùy chỉnh hệ thống vì hầu hết là phần mềm nguồn mở.
  - Phần cứng
    - Các thiết bị chuyên dụng, các mạch điều hướng trong thiết bị được thiết kế tối ưu cho việc điều hướng
    - Chi phí mua thường không nhỏ và khả năng tùy biến không lớn

Load balancing

- Mô hình 7 lớp OSI, quy định các layer khác nhau trong quá trình truyền nhận dữ liệu (Xem thêm về OSI Model). Mỗi tầng trong mô hình OSI có các chức năng khác nhau giúp cho việc kết nối và trao đổi dữ liệu giữa các thiết bị được đảm bảo, an toàn và chính xác.
- Chọn chia tải ở tầng nào trong 7 tầng mạng của mô hình OSI:
  - Hoạt động ở 2 tầng: tầng 4 (Transport) hoặc tầng 7 (Application).

Câu hỏi

- Sự khác nhau của Load balancer giữa tầng 4 và 7
- Tìm hiểu về HAProxy
- Khái niệm về HAProxy chia tải cho web server và database server

3. Web Application Servers

- Là nơi tập trung xử lý các logic của ứng dụng, chúng xử lý các request của người dùng, gửi HTML lại cho trình duyệt.
- Giao tiếp với rất nhiều thành phần của hệ thống backend: database, caching layers, search service, data / logging queues,...
- Với một ứng dụng web có Load balancer thì thường có tối thiểu 2 web server, cắm vào load balancer để xử lý request của người dùng.

4. Database

- Là nơi lưu trữ toàn bộ dữ liệu trong hệ thống.
- Cung cấp cách để lưu trữ dữ liệu theo cấu trúc, chèn dữ liệu mới, tìm kiếm, sửa và xóa chúng

5. Caching service

- Trong quá trình hệ thống hoạt động, có những thành phần dữ liệu thường xuyên được truy xuất, tìm kiếm và tổng hợp dữ liệu phức tạp => cache
- Cung cấp kho dữ liệu **key/value** để lưu trữ và tra cứu thông tin trong thời gian gần.
- Các ứng dụng thường tận dụng caching service để lưu trữ kết quả của các xử lý phức tạp để có thể lấy lại kết quả từ bộ nhớ đệm mà không phải thực hiện xử lý lần nữa. Cache có thể dùng để lưu kết quả truy vấn database, services, HTML từ url, ...

Caching service

- Ví dụ:
  - Google hay lưu kết quả search của những từ thông thường thay vì query lại mọi lần.
  - Facebook lưu lại phần lớn những bài viết bạn thấy khi đăng nhập.
- Hiện tại có 2 hệ thống lưu trữ phổ biến là Redis và Memcache.

Bài tập

- Tìm hiểu tổng quan về Redis

6. Job Queue (6a) & Servers (6b)

- Các ứng dụng web có một số hoạt động bất đồng bộ không trực tiếp liên quan với kết quả trả về request từ user.
- Job Queues là một danh sách các job đang cần được xử lý một cách bất đồng bộ.
- Các job sẽ được hoạt động theo thời gian đã được lên kế hoạch từ trước hoặc job sẽ được chạy theo hoạt động của người dùng. Và Job Server là một server riêng nơi job sẽ được chạy.

Job Queue (6a) & Servers (6b)

- VD: Đơn hàng tạo thành công, hệ thống cần gởi email thông báo, gởi notification cho các bên có liên quan và một loạt các nghiệp vụ khác. Nếu để client phải đợi hệ thống xong hết các công việc này sẽ không hợp lý, resource tại servers tiếp nhận request cũng tiêu tốn rất lớn.

7. Full-text Search Service

- Cách tự nhiên nhất để tìm kiếm thông tin, như Google, gõ từ khóa và nhấn enter.
- Service này chuyên dùng để tìm kiếm full-text.
- Là kỹ thuật tìm kiếm trên "Full text database": là cơ sở dữ liệu chứa "toàn bộ" các kí tự (text) của một hoặc một số các tài liệu, bài báo.. (document), hoặc là của websites.

Full-text Search Service

- Toán tử LIKE trong SQL.
  - Chỉ tìm kiếm ở column đã định trước =>lượng thông tin phải tìm giới hạn lại chỉ trong các column đó.
  - matching pattern cho “từng” chuỗi của từng dòng (rows) của field tương ứng, do đó về độ phức tạp sẽ là tuyến tính với số dòng, và số kí tự của từng dòng, hay chính là “toàn bộ kí tự chứa trong field cần tìm kiếm”. Do đó sử dụng LIKE query sẽ gặp các vấn đề:

Full-text Search Service

- Không chính xác
  - Độ nhiễu cao
  - Từ đồng nghĩa (synonyms)
  - Từ cấu tạo bằng chữ đầu của cụm từ (acronym)
- Performance không tốt (Tốc độ truy vấn chậm, '%keyword%' không dùng index)
- Vấn đề với tìm kiếm tiếng Việt có dấu và không dấu

Full-text Search Service: inverted index

- Full-text search tận dụng inverted index để nhanh chóng tìm kiếm văn bảng có từ khóa cần tìm.
- Inverted index là kỹ thuật thay vì index theo từng đơn vị row (document) giống như mysql thì chúng ta sẽ biến thành index theo đơn vị term =>Inverted index là một cấu trúc dữ liệu, nhằm mục đích map giữa term và các document chứa term đó.

Ví dụ: **Full-text Search Service:** inverted index

Documents (Photo titles)	
id	**title**
1	Man running in the mountains
2	Mountains with snow
3	Man running marathon

Inverted Index	
keyword	**photo_ids**
man	1, 3
running	1, 3
mountains	1, 2
snow	2
marathon	3

Full-text Search Service: inverted index

- Để tạo ra Inverted Index thì phải tách được một string ra thành các term, sau đó sẽ index string đó theo term đã tách được
- Công nghệ Tokenize là một bài toán con quan trọng nằm trong bài toán lớn của Full Text Search: N-gram và Morphological Analysis

Full-text Search Service: Tokenize

- **N-gram** là kĩ thuật tokenize một chuỗi thành các chuỗi con, thông qua việc chia đều chuỗi đã có thành các chuỗi con đều nhau, có độ dài là N.
- **Morphological Analysis** là một kĩ thuật phổ biến trong xử lý ngôn ngữ tự nhiên (Natural Language Processing). Morphological chính là "cấu trúc" của từ, như vậy MA sẽ là "phân tích cấu trúc của từ", hay nói một cách rõ ràng hơn, MA sẽ là kĩ thuật tokenize mà để tách một chuỗi ra thành các từ có ý nghĩa.

Bài tập

- Full-text search trong mysql

8. Services

- Khi hệ thống đạt một ngưỡng tải nhất định, hệ thống thường nên được chia nhỏ ra thành các service riêng lẻ.
- Các service này đóng một vai trò nhất định: Account Service, Product Service, Order Service, Payment Online… Mục đích vẫn là chia để trị tốt hơn, tránh dồn hết vào một server, 1 service bị nghẽn sẽ ảnh hưởng tới các toàn bộ service khác.
- Web Server chỉ còn làm nhiệm vụ của nó là render HTML, các service bên dưới mới thực sự xử lý thông tin.

9. Data (9a, 9b, 9c)

- Khi các app đạt đến trạng thái hoạt động ổn định, nó sử dụng quy trình xử lý dữ liệu để chắc chắn rằng dữ liệu được thu thập, lưu trữ, phân tích. Một quy trình xử lý dữ liệu điển hình có ba giai đoạn:
  - App gửi dữ liệu, thông qua tương tác user, đến data firehose để lưu trữ và xử lý dữ liệu. Thường thì dữ liệu gốc được biến đổi hoặc tăng thêm và chuyển đến cho firehose khác. AWS Kinesis và Kafka là hai công nghệ thường sử dụng cho mục đích này.
  - Dữ liệu gốc sau khi được biến đổi sẽ được lưu trữ trong cloud storage. AWS Kinesis cung cấp thiết lập được gọi là firehose để lưu trữ dữ liệu gốc vào cloud storage (S3).
  - Dữ liệu đã biến đổi được đưa vào trong kho dữ liệu để phân tích. Sử dụng AWS Redshift sẽ giải quyết được việc này.

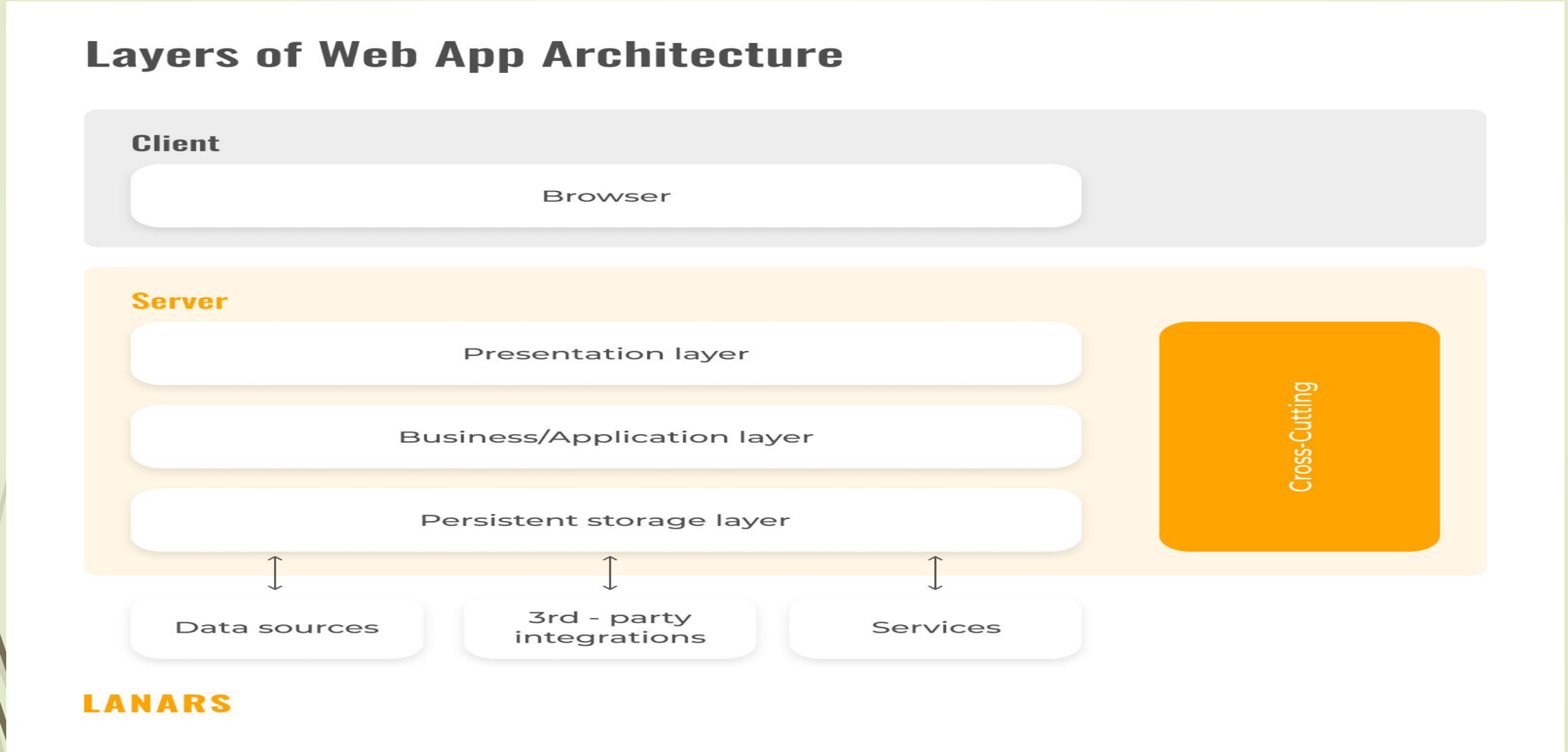
10. Cloud storage

- Là cách đơn giản để lưu trữ, truy cập và chia sẻ dữ liệu trên Internet.
- Các service này thường là bên thứ ba (third-party / 3rd cung cấp) như: AWS S3, Google Storage, Fshare
- Dùng lưu trữ: video, ảnh, audio, file css, js, ...

11. CDN

- Content Delivery Network
- Người dùng từ khắp mọi nơi trên thế giới, khi hệ thống cần phục vụ file vật lý tốt hơn: **nhanh, ổn định** => dùng CDN
- Phân phối các file vật lý lưu trữ ở rất nhiều server trên thế giới thay vì một nơi. Từ đó người dùng ở gần server nào thì sẽ được server ấy phục vụ, thay vì web server chính => không còn lo lắng việc backup, hoặc server chịu tải và băng thông lớn.

Layers of Web App Architecture



Layers of Web App Architecture

- Presentation layer (PL)
  - Người dùng giao tiếp thông qua trình duyệt web.
  - Gồm UX và UI hỗ trợ tương tác hệ thống: HTML, CSS, Javascript.
  - Giao tiếp với tầng khác qua application program interface (API)

Layers of Web App Architecture

- Business logic layer (BLL)
  - Application Layer
  - Xây dựng bằng cách dùng những ngôn ngữ lập trình: Java, .Net, NodeJS, PHP, Python, Ruby
  - Chọn monolithic or microservice

Layers of Web App Architecture

- Persistence storage layer
  - Data access layer
  - Nhận tất cả các lệnh gọi dữ liệu và cung cấp quyền truy cập vào bộ lưu trữ của một ứng dụng, kết nối chặt chẽ với tầng nghiệp vụ, biết cơ sở dữ liệu nào cần trao đổi và tối ưu hóa quá trình truy xuất dữ liệu.

Layers of Web App Architecture

- Cross-cutting code
  - xử lý các mối quan tâm khác của ứng dụng như thông tin liên lạc, quản lý hoạt động và bảo mật. Nó ảnh hưởng đến tất cả các bộ phận của hệ thống nhưng không bao giờ được trộn lẫn với chúng.

Web Application Components

- UI/UX Web Application Components: gồm activity logs, dashboards, notifications, settings, statistics ….một phần của layout web app.
- Structural Components: có 2 mảng lớn là client and server sides.

Web Application Components

- Structural Components
  - Client Component
    - Phát triển HTML, CSS, Javascript, tồn tại ở trình duyệt web phía người dùng.
  - Server Component
    - Xây dựng bằng cách dùng những ngôn ngữ lập trình: Java, .Net, NodeJS, PHP, Python, Ruby. Gồm 2 thành phần: app logic and database

Models of Web Application Components

- One Web Server, One Database
  - Mô hình kém tin cậy, không được sử dụng để xây dựng web application
  - Thích hợp dùng cho thử nghiệm hoặc học các nguyên lý cơ bản của web application

Models of Web Application Components

- Multiple Web Servers, One Database
  - Cần ít nhất 2 web servers
  - Web server không lưu trữ dữ liệu.
  - Tất cả resquest chuyển đến server mới nếu 1 cái down
  - Linh động hơn single server

Models của Web Application Components

- Multiple Web Server, Multiple Databases
  - Mô hình có hiệu quả nhất trong xây dựng web application.
  - Ví dụ:
    - 1 csdl lưu nội dung như MySQL, MariaDB, PostgreSQL
    - Redis or Memcached lưu caching
    - Chạy background job queue
    - Elastic cho full-text search
    - Data Warehouse cho tập hợp dữ liệu và chạy phân tích.

Types of Web Application Architecture

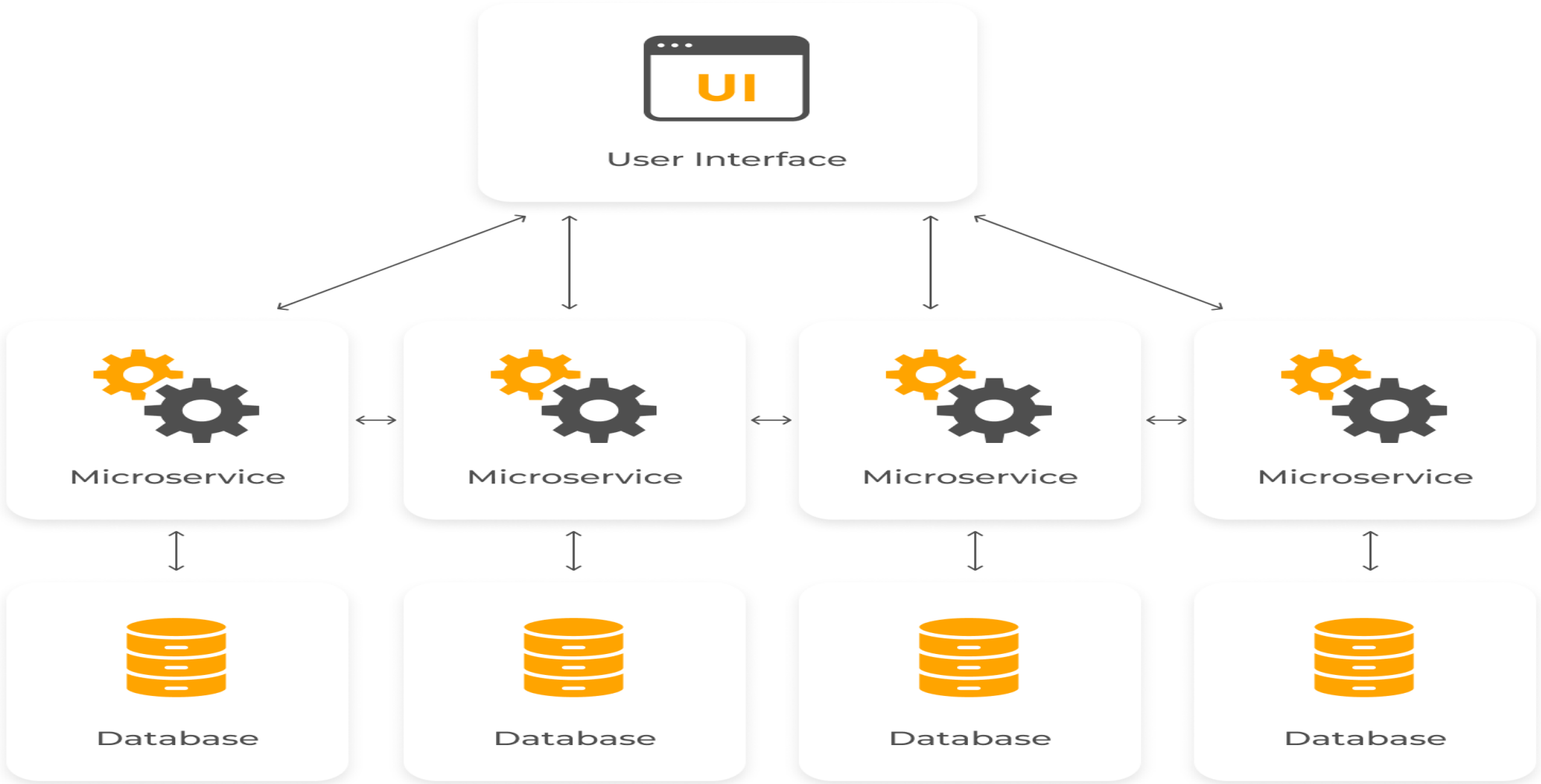
Types of Web Application Architecture

- Single-Page Application (SPA)
  - Người dùng không cần phải tải các trang hoàn toàn mới mỗi lần chuyển hướng (chỉ tải mới hoàn toàn cho lần đầu tiên), chỉ tải những dữ liệu cần thiết cho các lần sau.
- Microservice
- Serverless
- Progressive web app

Microservice

- Khác biệt với kiến trúc Monolith, hay vì gom tất cả module thành một khối (monolith), ta tách các module thành những service siêu nhỏ. Mỗi service sẽ được đặt trên một server riêng (có thể dùng server cloud như AWS hoặc Azure), giao tiếp với nhau thông qua mạng (Gửi nhận message qua giao thức HTTP hoặc sử dụng MessageQueue)...

Microservices



LANARS

Microservice

- Ưu điểm
  - Các service có thể được bảo trì độc lập: Các service là riêng biệt, vì vậy khi có một service được bảo trì nó không làm ảnh hưởng tới các service khác.
    - không gây "chết" các service khác nếu service đang bảo trì gặp sự cố, nhưng hệ thống hoạt động sẽ không đầy đủ tính năng.

Microservices

- Dễ dàng nắm bắt các business logic: Mỗi service xử lý một nghiệp vụ riêng, vì vậy mà ta có thể dễ dàng nắm bắt được các business logic.
- Áp dụng được nhiều công nghệ: mỗi service được coi như một dự án riêng.
- Dễ dàng mở rộng khi hệ thống trở nên phức tạp: Microservice có kiến trúc khá tương đồng với kiến trúc của một hệ phân tán, nên nó cũng có khả năng mở rộng theo chiều ngang.

Microservices

- Nhược điểm
  - Có độ trễ cao: Các service giao tiếp với nhau thông qua môi trường mạng, nên sẽ có độ trễ cao hơn so với kiến trúc monolithic
  - Khó phát triển: Để có thể phát triển thì các developer phải kéo toàn bộ hệ thống về máy local để phát triển, trong khi một hệ thống microservice rất phức tạp, khó mà giả lập được trên môi trường local.

Microservices

- Khó đảm bảo tính toàn vẹn dữ liệu: Vì dữ liệu có thể được lưu trên nhiều service khác nhau, nên khó có thể đảm bảo được tính toàn vẹn

monolithic

- Monolithic là kiến trúc phần mềm dạng nguyên khối.
- Toàn bộ các module (view, business, database, report) đều được gom chung vào một project lớn. Khi deploy, chúng sẽ quăng khối code này lên server và config để nó chạy.

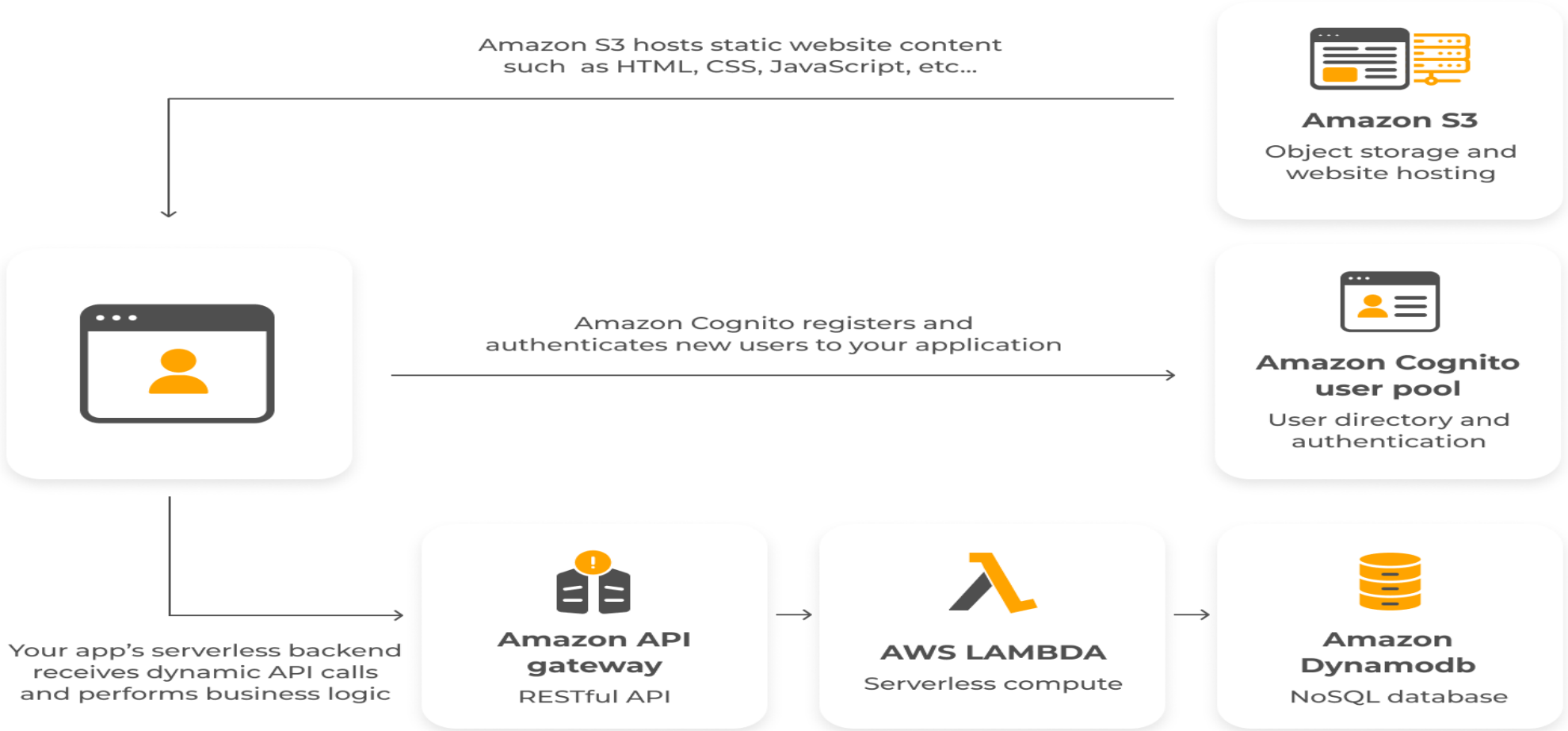
Bài tập

- Ưu – nhược điểm của monolithic

Serverless

- Serverless là mô hình thực thi điện toán đám mây trong đó nhà cung cấp đám mây tự động quản lý việc phân bổ và cung cấp máy chủ, giá cả của mô hình này dựa trên số lượng tài nguyên thực tế mà ứng dụng sử dụng thay vì phải trả trước một khoản nhất định trong một khoảng thời gian

Servless



LANARS

Serverless Architectures

- Serverless được dùng để chỉ hai khái niệm mô hình dịch vụ khác nhau đó là
  - BaaS – Backend as a Service: phần lớn code logic của chúng ta sẽ chuyển về xử lý ở phía Frontend. Còn Backend thì sử dụng các API có sẵn của bên thứ ba.
  - FaaS – Function as a Service: tự viết các API cho mục đích của mình, và triển khai chúng lên Server

Ưu điểm Serverless

- Đầu tiên là chi phí:
  - Thuê server trả tiền theo tháng hoặc theo năm
  - Serverless tính phí theo thời gian và số lần gọi Function nên chi phí sẽ rẻ hơn, giảm rất nhiều các chi phí đi kèm như bảo trì máy móc trang thiết bị…

Ưu điểm Serverless

- Dễ dàng mở rộng quy mô:
  - Khi số lượng request tới ứng dụng tăng cao:
    - Thuê hoặc tự xây dựng server thì phải nâng cấp để đảm bảo tốc độ cho ứng dụng, điều này sẽ tốn nhiều thời gian và nhân lực.
    - Serverless, các nhà cung cấp bên thứ ba sẽ tự mở rộng thêm các tiến trình và tài nguyên để cân bằng tải khi có nhiều request

Ưu điểm Serverless

- Deploy code đơn giản hơn:
  - Chỉ cần đẩy code lên, mọi việc còn lại đã có nhà cung cấp dịch vụ xử lý
  - Cho phép xây dựng các ứng dụng khác nhau tùy mục đích, như các ứng dụng chuyên xử lý request/response hoặc các ứng dụng xử lý hàng loạt (batch processing)

Nhược điểm Serverless

- Khó khăn cho việc phát triển code ứng dụng ở máy local.
- Khi cần 1 thay đổi nhỏ, cũng phải đẩy code lên máy chủ để chạy thử code, như vậy sẽ tốn thời gian phát triển hơn vì phải đợi và mất đi một khoản chi phí lúc phát triển.
- Không biết được code ở đâu, tạo cảm giác không an toàn

Nhược điểm Serverless

- Về hiệu năng có thể không nhanh được bằng server riêng bởi code được chạy mỗi khi có yêu cầu, sẽ mất khoảng 20-50 millisecond để start request.
- Về việc Debug: những tài nguyên như CPU hay RAM chỉ được quản lý ở bên phía nhà cung cấp dịch vụ cloud nên khó tái tạo lại môi trường ở máy local để debug ứng dụng

Bài tập

- Tìm hiểu về Progressive web app (PWA)