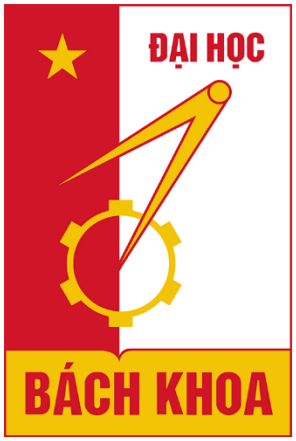
ĐẠI HỌC BÁCH KHOA HÀ NỘI

TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

——————— * ———————



Báo cáo bài tập lớn

HP: Phát triển phần mềm phân tán

Đề tài: NodeJS trong phát triển ứng dụng phân tán

Giảng viên hướng dẫn: **Ts. Nguyễn Nhất Hải**

Nhóm sinh viên thực hiện: Nhóm 7

Họ và tên	MSSV	Lớp
Nguyễn Tất Thành	20175661	G-INP 16
Vũ Hoàng Tuấn	20175995	G-INP 16
Nguyễn Đình Tiến	20175992	G-INP 16

Hà Nội – 2021

1. Mở đầu – Tìm hiểu lý thuyết

Hệ phân tán là một tập hợp các máy tính độc lập kết nối với nhau cung cấp dịch vụ tới người dùng như là một hệ thống duy nhất. – Tanenbaum 2006.

Ngày nay hệ phân tán đang phát triển dựa trên nhu cầu chia sẻ tài nguyên và thông tin lớn mà các hệ điều hành đã có từ trước không đáp ứng được, đối với một hệ thống thống nhất – hệ điều hành tập trung và hệ điều hành mạng thuần túy không đáp ứng được nhu cầu đối với sự tăng trưởng đó. Ngoài ra, trên cơ sở việc kết nối mạng để triển khai hệ điều hành mạng tạo nên một cơ sở kỹ thuật hạ tầng (phần cứng, kết nối mạng, phần mềm) làm nền tảng phát triển hệ phân tán.

Hệ phân tán có 4 đặc trưng bởi các tính chất: chia sẻ tài nguyên, tính trong suốt, tính mở và tính co dãn

Chia sẻ tài nguyên chỉ khả năng chia sẻ thông tin của các máy tính / thiết bị trong hệ phân tán. Chương trình quản lý tài nguyên có khả năng nhận các yêu cầu do các chương trình khác gửi đến, chuyển các yêu cầu này thành các yêu cầu truy cập tài nguyên vật lý rồi nhận trả lời từ tài nguyên vật lý và cung cấp ngược lại cho các chương trình yêu cầu tài nguyên. Việc chia sẻ tài nguyên giúp:

- Kết nối thông tin / tài nguyên
- Giảm chi phí (chi phí đầu tư các thiết bị,…)
- Tăng tính sẵn sàng (làm việc từ xa, remote, …)
- Cho khả năng làm việc nhóm

Tuy nhiên cần phải chú ý đến các vấn đề về bảo mật, rủi ro về an toàn thông tin khi chia sẻ đặc biệt là qua môi trưởng mạng.

Tính trong suốt là khả năng cung cấp một khung cảnh lôgic của hệ thống cho người dùng, độc lập với hạ tầng vật lý. Hệ thống luôn là duy nhất đối với người dùng song nó sẽ che giấu được tính phân tán của hệ phân tán phía dưới. Có nhiều góc độ trong suốt khác nhau. Việc đảm bảo tính trong suốt là 1 trong những y/c chắc chắn phải thực hiên để đảm bảo định nghĩa của hệ thống phân tán. Tuy nhiên để có được tính trong suốt ở mức độ tuyệt đối sẽ kéo theo chi phí về TN rất cao. Do đó không phải lúc nào cũng hướng tới trong suốt tuyệt đối và phải xem xét cân bằng trong suốt

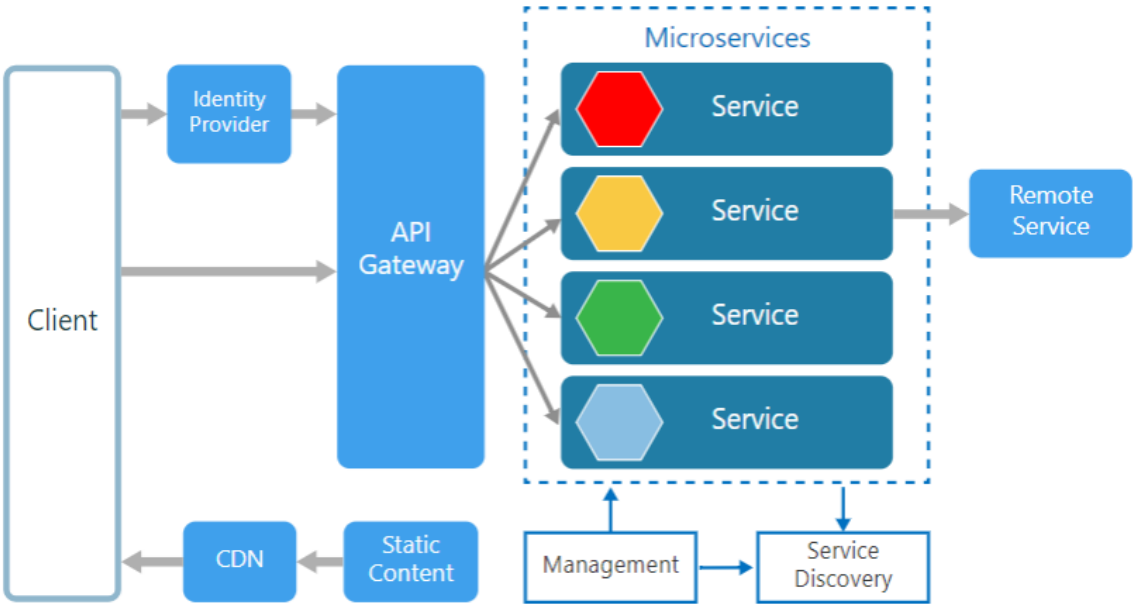
Một hệ thống phân tán tương tự một hệ thống nhiều thành phần, các thành phần có thể đến từ cấc nguồn gốc và nhà sản xuất khác nhau. Để thống nhất được cần cung cấp các *interface* để cho các nhà sản xuất, nhà phát triển cài đặt hoặc sử dụng và cũng phải đảm bảo tính trung lập giữa các bên. Đó là yêu cầu về tính mở của một hệ phân tán

Tính co giãn là khả năng của hệ thống có thể đáp ứng được các thay đổi của hạ tầng của môi trường xung quanh, xem xét về mặt qui mô (đáp ứng nhu cầu hệ thống khi số lượng người sử dụng tăng), mặt địa lý (đảm bảo trao đổi thông tin), và về mặt tổ chức (phân bố, tổ chức hệ thống thành các hệ thống con, sẵn sàng thay thế khi cần thiết)

2. Kiến trúc trong hệ phân tán

Hệ phân tán cần có kiến trúc – cách sắp xếp và tổ chức hệ thống hợp lý. Kiểu kiến trúc thường thấy có thể kể đến kiến trúc phân tầng, hướng đối tượng, hướng sự kiện, hướng dữ liệu. Kiến trúc thường thấy thì có client – server, phân tầng, hỗn hợp,máy chủ biên, …. Trong thời điểm sử dụng web đang phát triển mạnh hiện nay thì chúng ta có thể thấy sự nổi bật của kiểu kiến trúc microservices và mô hình kiến trúc client server đang được ứng dụng nhiều.

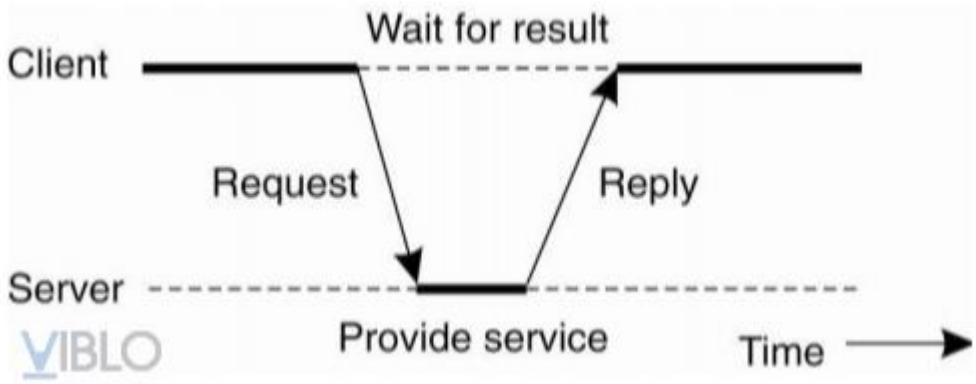
Khi xây dựng phần mềm theo kiến trúc monolith (một khối). Toàn bộ các module (view, business, database, report) đều được gom chung vào một project lớn. Khi deploy, chúng ta sẽ để khối này lên server và config để server chạy. Tuy nhiên, khi phần mềm trở nên lớn và phức tạp thì nó lại dần bộc lộ nhược điểm. Do các module đóng với nhau thành một bánh, khi muốn nâng cấp một module, ta phải deploy lại toàn bộ code (người dùng cuối sẽ không sử dụng được toàn bộ các chức năng của hệ thống khi deploy); khi muốn phục vụ nhiều người dùng, ta phải nâng cấp server..., vì vậy kiểu kiến trúc microservices ra đời, chia nhỏ các module thành các services nhằm:



- Đơn giản triển khai

- Khối trở nên đơn giản để hiểu
- Tăng tính tái sử dụng
- Nhanh chóng cách ly thành phần hỏng nếu có
- Giảm thiểu nguy cơ khi thực hiện thay đổi

Mô hình client server (mô hình mạng máy khách – máy chủ) là một cấu trúc ứng dụng phân tán. Nó phân vùng các nhiệm vụ hay workload giữa các nhà cung cấp tài nguyên hoặc dịch vụ, gọi là server, và người yêu cầu dịch vụ (Client). Trong kiến trúc Client Server, khi máy Client gửi yêu cầu dữ liệu đến Server thông qua Internet, server sẽ chấp nhận quy trình được yêu cầu. Sau đó gửi các gói dữ liệu được yêu cầu trở lại client. Client không chia sẻ bất kỳ tài nguyên nào của họ.



Nguyên tắc hoạt động của mô hình có thể tóm gọn như sau:

- Client: là bên yêu cầu, sử dụng dịch vụ. Client có khả năng nhận thông tin hoặc sử dụng một dịch vụ cụ thể từ các nhà cung cấp dịch vụ (Server). Là tiến trình yêu cầu dịch vụ từ phía server bằng cách gửi yêu cầu, chờ server trả lời, nhận kết quả và hiển thị cho người sử dụng
- Server: bên cung cấp dịch vụ, nó cung cấp các thông tin (dữ liệu) hoặc quyền truy cập vào các dịch vụ cụ thể. Là tiến trình triển khai 1 dịch vụ cụ thể. Công việc của nó là lắng nghe, nhận yêu cầu, xử lý, trả lời

Đặc điểm của mô hình:

- Tập trung (quyền hạn)
- Bảo mật do quyền hạn tập trung cho một nhóm quản trị nhất định
- Khả năng mở rộng tốt
- Khả năng truy cập nhanh do tập trung vào một server xử lý
  - Dễ bị tắc nghẽn do tập trung quá vào một server
  - Kéo theo server giảm độ bền hoặc ổn định
  - Chi phí thường cao
  - Việc bảo trì ảnh hưởng tới client

- Chia sẻ tài nguyên còn hạn chế, phải thông qua server

3. Quản lý tiến trình và nguồn trong hệ phân tán

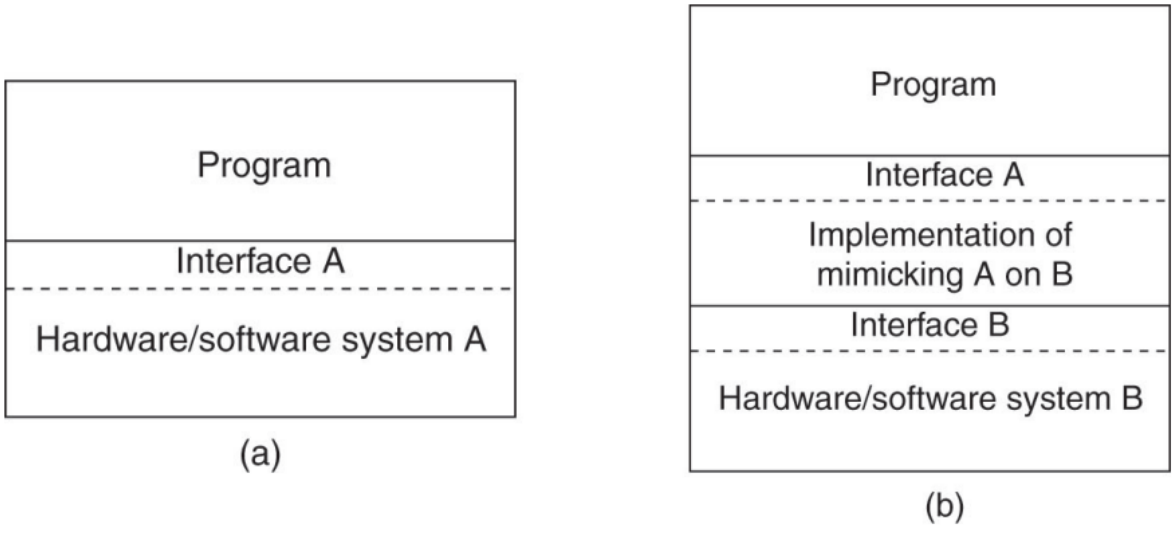
Tiến trình là 1 chương trình đang được chạy trên 1 trong các bộ xử lý ảo của hệ điều hành. Một tiến trình được định nghĩa như một chương trình đang chạy trên một bộ xử lý ảo. Vấn đề quan trọng là HĐH làm cho hoạt động của các tiến trình đó là trong suốt với nhau. Chúng hoàn toàn độc lập và không gây ảnh hưởng được lẫn nhau.

Khi 1 tiến trình chạy 1 chương trình nó sẽ sinh ra 1 hay nhiều luồng. Các luồng không cần phải hoạt động hoàn toàn độc lập và trong suốt với nhau. Vì vậy mỗi luồng chỉ cần lưu trữ thông tin ít nhất để có thể cho phép các luồng chia sẻ CPU. Các ứng dụng đa luồng thường giúp chương trình có hiệu năng cao hơn, tuy nhiên việc lập trình để cho các luồng không ảnh hưởng lẫn nhau là khá khó khăn và tốn chi phí. Cần có cách tổ chức cũng nhử quản lý các luồng.

Trong hệ phân tán có nhiều mô hình quản lý luồng: server đơn luồng, client và server đa luồng, server có điều phối, server đa luồng, máy trạng thái hữu hạn hay client đa luồng. Việc lựa chọn mô hình phụ thuộc vào mục đích của nhà phát triển.

4. Ảo hóa

Một cách dễ hiểu, ảo hóa là một thuật ngữ dùng để chỉ một phương pháp cô lập tài nguyên để tạo ra một phiên bản ảo của một cái gì đó như là máy ảo(VM), ổ đĩa ảo(virtual disk), mạng ảo(virtual network).



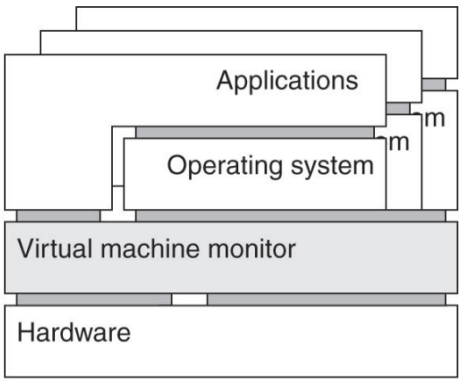
a): Mô hình hoạt động của 1 hệ thống thông thường với việc sử dụng giao diện như 1 tầng trung gian giữa các chương trình ứng dụng và tầng dưới là hệ thống phần cứng A và phần mềm A. Tàng giao diện này cung cấp các phương thức được

triển khai ở tầng hệ thống A bên dưới, tầng chương trình ở trên chỉ việc sử dụng giao diện này để gọi các phương thức đó

b): Khi đem cùng những chương trình kèm theo giao diện A sang 1 hệ thống khác (hệ thống B), lúc này cần 1 tầng trung gian để mô phỏng sự triển khai của hệ thống A trên hệ thống B. Cụ thể là triển khai các phương thức hệ thống A bằng các gọi phương thức tương ứng của hệ thống B mà giao diện B cung cấp. Tầng trung gian này mô phỏng ý tưởng của thực hiện ảo hóa (giả lập môi trường làm việc của hệ thống này trên môi trường làm việc của hệ thống khác)

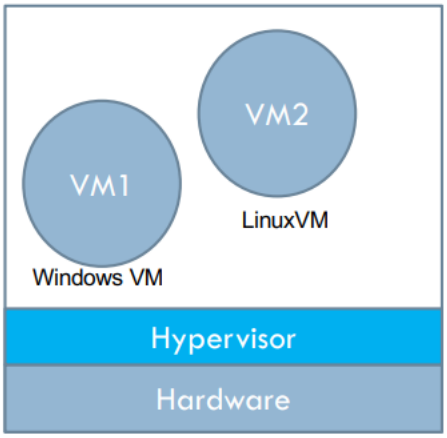
Một số kiến trúc máy ảo:

- Kiến trúc mô phỏng hoàn toàn Host – based (vd: Java Virtual Machine). Kiến trúc này sử dụng một lớp hypervisor chạy trên nền tảng hệ điều hành, sử dụng các dịch vụ được hệ điều hành cung cấp để phân chia tài nguyên tới các máy ảo. Nếu ta xem hypervisor này là một lớp phần mềm riêng biệt, thì các hệ điều hành khách của máy ảo sẽ nằm trên lớp thứ 3 so với phần cứng máy chủ.
- Bare-metal hypervisor. Trong mô hình này, lớp phần mềm hypervisor chạy trực tiếp trên nền tảng phần cứng của máy chủ, không thông qua bất kì một hệ điều hành hay một nền tảng nào khác. Qua đó, các hypervisor này có khả năng điều khiển, kiểm soát phần cứng của máy chủ. Đồng thời, nó cũng có khả năng quản lý các hệ điều hành chạy trên nó. Nói cách khác, các hệ điều hành sẽ chạy trên một lớp nằm phía trên các hypervisor dạng bare-metal (vd: Hyper-V)
- Hybrid là một kiểu ảo hóa mới hơn và có nhiều ưu điểm. Trong đó lớp ảo hóa hypervisor chạy song song với hệ điều hành máy chủ. Tuy nhiên trong cấu trúc ảo hóa này, các máy chủ ảo vẫn phải đi qua hệ điều hành máy chủ để truy cập phần cứng nhưng khác biệt ở chỗ cả hệ điều hành máy chủ và các máy chủ ảo đều chạy trong chế độ hạt nhân. Khi một trong hệ điều hành máy chủ hoặc một máy chủ ảo cần xử lý tác vụ thì CPU sẽ phục vụ nhu cầu cho hệ điều hành máy chủ hoặc máy chủ ảo tương ứng. Lý do khiến Hyrbird nhanh hơn là lớp ảo hóa chạy trong chế độ hạt nhân (chạy song song với hệ điều hành) (vd: Docker, kurbernete)

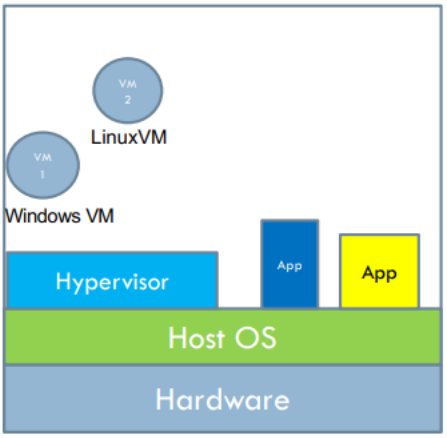


Host-Based Model

Type 1: Bare-metal supervisor



Type 2 **Hybrid**



5. Áp dụng công nghệ

5.1 Node JS

Là một công nghệ back-end được sử dụng bởi server viết bằng ngôn ngữ javascript. Bản thân ngôn ngữ viết nên công nghệ là một ngôn ngữ có tính chất hướng sự kiện, phù hợp để xây dựng các kiểu kiến trúc hướng sự kiện. Node Js khi chạy mở ra một luồng đón các sự kiện bằng cách "lắng nghe" chúng và trả về các kết quả đã được lập trình từ trước.

Ngoài ra NodeJS là một công nghệ không quá khó để xây dựng. Chúng ta có thể xây dựng một kiến trúc Client – Server áp dụng mô hình máy trạng thái hữu hạn chỉ với số lượng mã nguồn nhỏ.

Ví dụ: chỉ với 13 dòng mã nguồn ta đã mở được một luồng lắng nghe yêu cầu như là một server

```
...
const express = require("express");

// default route
app.get("/", (req, res) => {
  res.json({ message: "Welcome to application." });
});

// open port to listen for request
const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}.`);
});

```

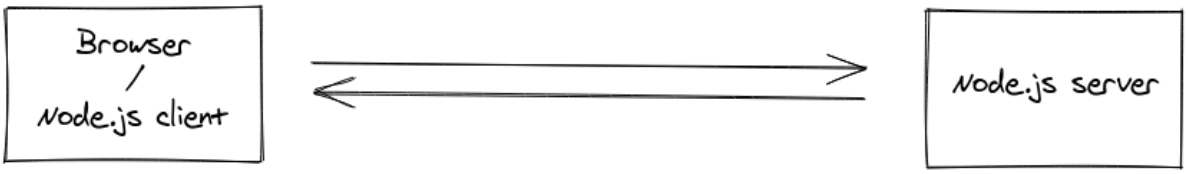
5.2. WebSocket và SocketIO

Thông thường khi xây dựng hệ thống theo kiến trúc Client – Server, chỉ khi nào client gọi yêu cầu đến server thì server mới có phản hồi lại cho client đó. cách thức trên thì kết quả về chậm và tốn rất nhiều tài nguyên và không khả thi cho các ứng dụng lớn. Bởi vậy, **Socket.io** ra đời cho phép bạn xây dựng để xử lý việc giao tiếp giữa server và client ngay lập tức và chiếm ít tài nguyên nhất.

Socket.IO là một thư viện JavaScript cho các ứng dụng web thời gian thực. Nó cho phép trao đổi thông tin 2 chiều và thời gian thực giữa các web clients và servers. Nó có 2 phần: thư viện cho phía client chạy trên trình duyệt, và thư viện cho server cho Node.js. Cả 2 thành phần đó đều có chung bộ API. Như Node.js, nó có tính hướng sự kiện. Socket.IO sử dụng giao thức WebSocket, tuy nhiên nó có nhiều chức năng hơn như có khả năng quảng bá tới nhiều sockets, lưu trữ dữ

liệu liên quan với mỗi client, và thực hiện trao đổi vào ra bất đồng bộ. Với WebSocket, cả client và server có thể gửi dữ liệu cho nhau cùng lúc.

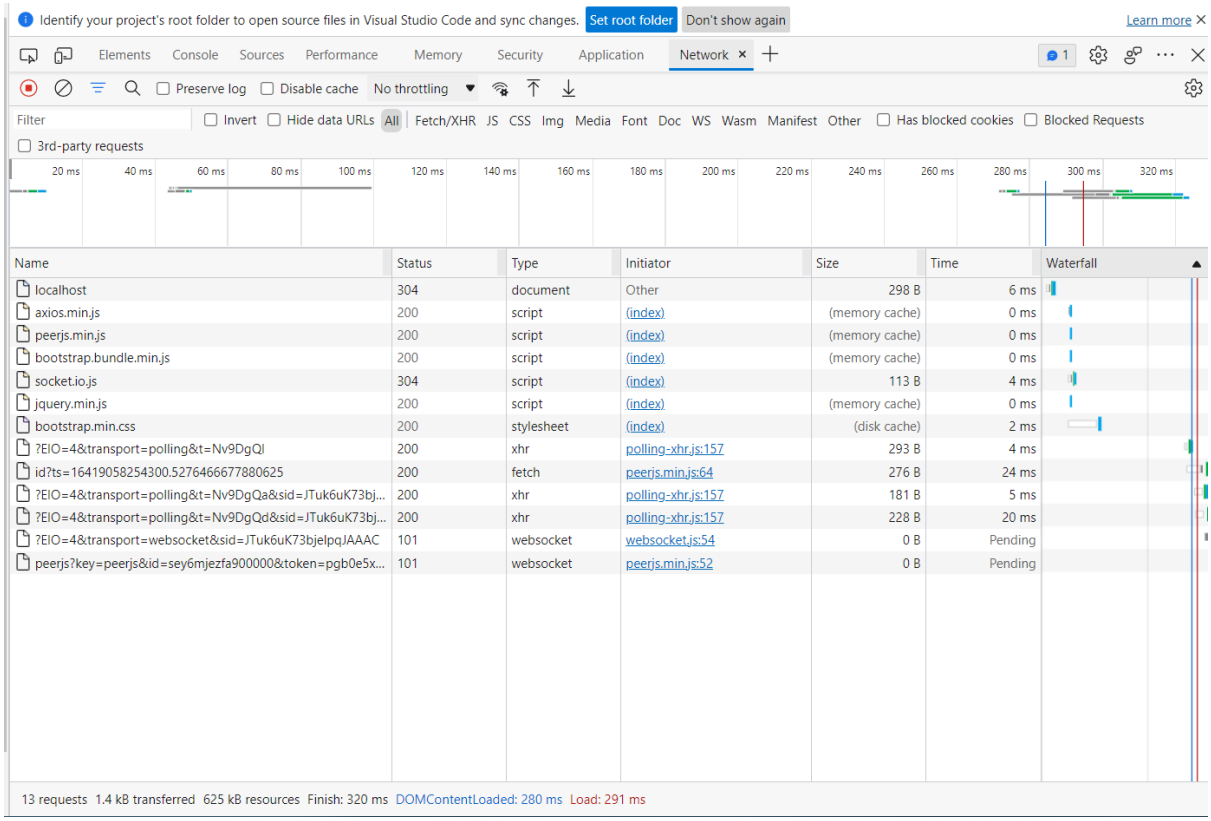
Websocket là một giao thức trao đổi thông tin cung ứng cơ chế truyền tin 2 chiều song song trên cùng một kết nối TCP. Giao thức Websocket đã được chuẩn hóa bởi IETF như RFC 6455 năm 2011, và bộ WebSocket API trong Web IDL đã được chuẩn hóa bởi W3C.



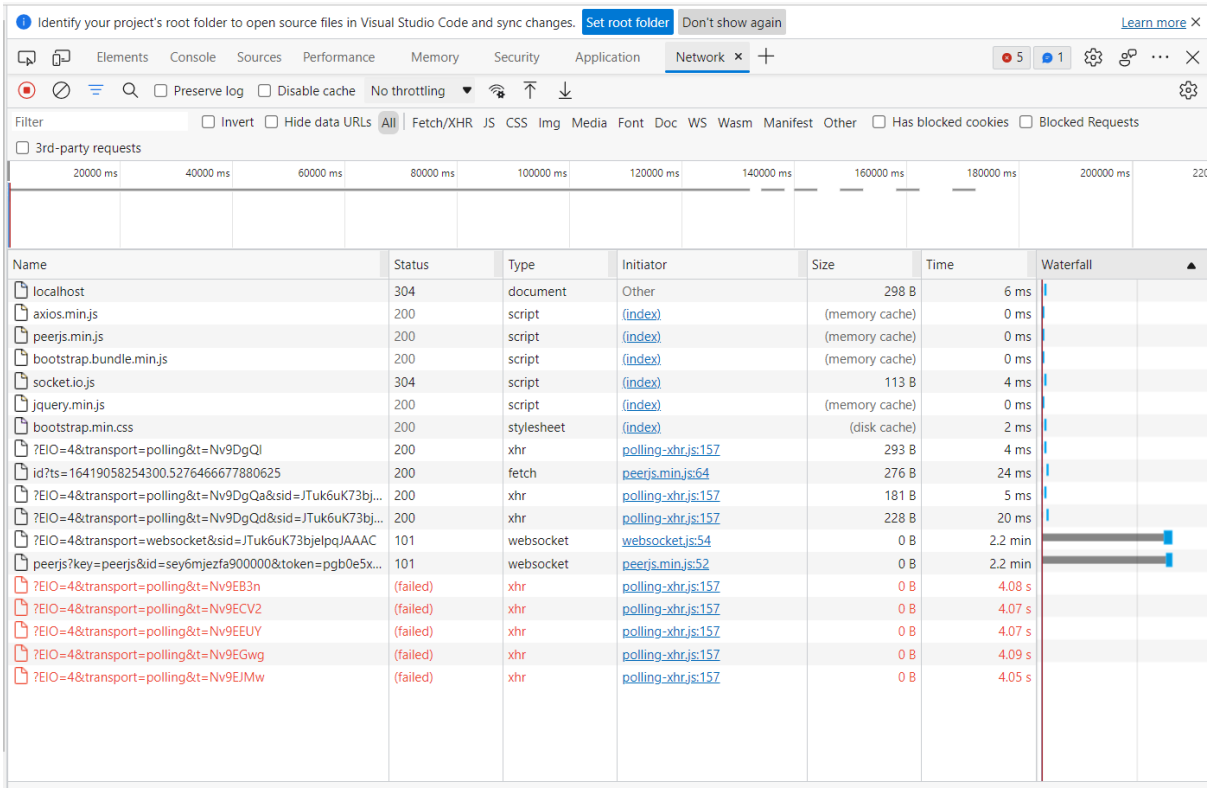
Một cách đơn giản ta có cơ chế hoạt động của SocketIO:

- Client và cà Server đều đã cài đặt SocketIO
- Client gửi request tới Server.
- EngineIO của thư viện chạy trên Server nhận thấy trong request có chứa giao thức HTTP long-polling yêu cầu kết nối. Server chấp nhận kết nối với client đó, thiết lập một kết nối bằng giao thức websocket
- Hai bên bắt đầu lắng nghe sự kiện của nhau.

Minh họa thiết lập kết nối của một client tới một server



Trước khi thiết lập 1 kết nốt type = websocket ta thấy 2 kết nối xhr chính là các request sử dụng giao thức http long-polling để kết nối

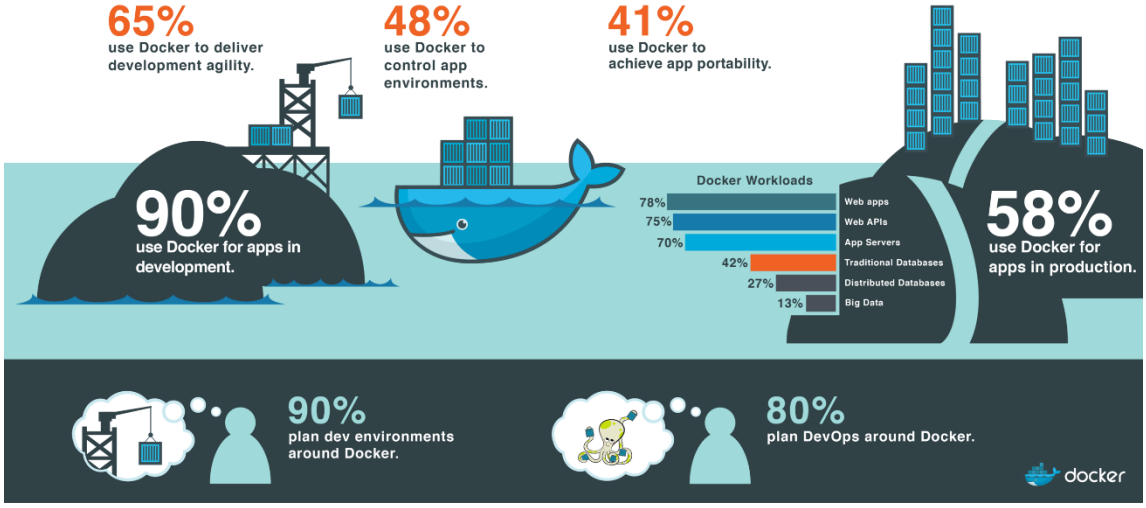


Trường hợp server ngắt kết nối, client vẫn liên tục gọi đến server nhưng không thành công, không thể thiết lập kết nối websocket.

Sử dụng NodeJs cùng với SocketIO là một ví dụ điển hình cho mô hình máy trạng thái hữu hạn, mô hình Client – Server, kiểu kiến trúc hướng sự kiện trong hệ thống phân tán . Server liên tục lắng nghe request và gửi về response bất đồng bộ, tuy là đơn luồng nhưng lời gọi được thực hiện liên tục, không bị block,hiệu suất không thua kém một server đa luồng.

5.3. Docker.

Chúng ta quay lại với kiểu kiến trúc Microservices. Kết hợp với các nhu cầu xây dựng ứng dụng, hệ thống "xuyên" hệ điều hành, phần cứng ta có Docker. 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), chủ yếu là cho các ứng dụng web.



Docker ra đời với sự kết hợp của 2 yếu tố:

- Xây dựng ứng dụng, hệ thống theo hướng microservices
- Áp dụng công nghệ ảo hóa

Hiện tại có rất nhiều công nghệ để xây dựng một ứng dụng web, mỗi công nghệ lại có thể đóng một vai trò khác nhau chạy riêng thành một khối giống ý tưởng kiến trúc Microservices. Vấn đề gặp phải là mỗi một công nghệ tuy có thể chạy tách rời nhưng lại cần có những yêu cầu môi trường khác nhau. Cài đặt quá nhiều môi trường trên một máy chủ có thể gây nên sự thiếu hụt tài nguyên. Để giải quyết vấn đề này, công nghệ container ra đời. Docker áp dụng công nghệ này bằng cách xây dựng một máy ảo, đưa khối services vào trong môi trường đó – gọi là một container. Container có thể hoạt động riêng lẻ, chỉ cài đặt các phụ thuộc cần thiết để chạy công nghệ. Có thể có các container nhỏ trong một container lớn, chúng giao tiếp được với nhau qua các lời gọi API. Các máy áo này chạy theo kiểu ảo hóa hybrid, tăng hiệu suất làm việc.

5.4. Ứng dụng chat:

Kết hợp các công nghệ trên, nhóm tiến hành xây dựng một ứng dụng chat nhỏ để demo các công nghệ và cách chúng hoạt động với nhau.

Phân tích cho thấy các chức năng cơ bản có Đăng nhập, Đăng ký, Chat và Gọi video.

Khi chat, tin nhắn của user này gửi lên phải hiển thị ở bên cửa sổ chat của user bên kia.

Tiến hành thiết kế và xây dựng ứng dụng với các công nghệ đã nêu, chúng ta xác định ứng dụng sẽ chia làm 3 phần:

- *Frontend*: Client, cũng có thể là một User interface. Nhóm đã thiết kế xây dựng một container chạy sử dụng thư viện ReactJS:

  Trong các phụ thuộc cần chú ý Client đã chứa thư viện socket.io-client


```
# Stage 1
FROM node:14 as build-stage

WORKDIR /ui
COPY package.json .
RUN npm install
COPY . .

ARG REACT_APP_API_BASE_URL
ENV REACT_APP_API_BASE_URL=$REACT_APP_API_BASE_URL

RUN npm run build

# Stage 2
FROM nginx:1.17.0-alpine

COPY --from=build-stage /ui/build /usr/share/nginx/html
EXPOSE $REACT_DOCKER_PORT

CMD nginx -g 'daemon off;'
```

Thông tin để tạo container gồm có phiên bản nodejs sẽ dùng, các câu lệnh để thư viện build thành các file HTML thông thường và cuối cùng là thông tin máy chủ nginx để khi gọi đến sẽ render ra các file HTML đó

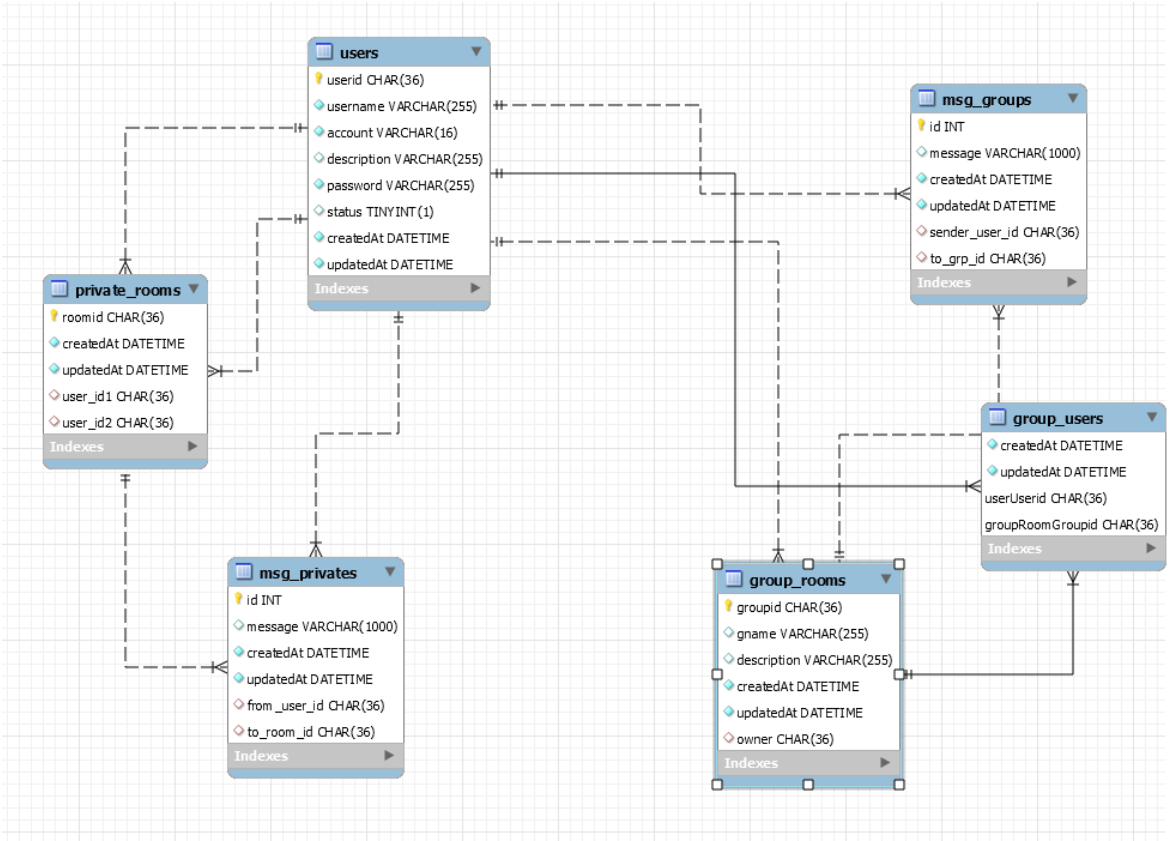
- *Backend*: Server. Thiết kế thông tin server:

Chú ý Server có thư viện SocketIO để có thể chat như đã phân tích. Dùng thư viện Peerjs để tạo kết nối peer-to-peer với giao thức websocket khi cần cuộc gọi video

```
FROM node
WORKDIR /be1
COPY package.json .
RUN npm install
COPY . .
CMD [ "node", "index.js" ]
```

Thông tin môi trường container của service chỉ đơn giản là có nodejs

- *Database*: Thông thường database sẽ đi liền với Backend, tuy nhiên nhóm sẽ để database thành một service riêng. Thiết kế dữ liệu của Database:



Nhóm có thiết kế dữ liệu cho chat nhóm nhưng do thiếu hụt về mặt kỹ năng nên vẫn chưa hoàn thành.

Thông tin Container chứa các service:

```
You, seconds ago | 1 author (You)
version: "3.8"
services:
  mysqldb:
    image: mysql
    restart: unless-stopped
    env_file: ./.env
    environment:
      - MYSQL_ROOT_PASSWORD=$MYSQLDB_ROOT_PASSWORD
      - MYSQL_DATABASE=$MYSQLDB_DATABASE
    ports:
      - $MYSQLDB_LOCAL_PORT:$MYSQLDB_DOCKER_PORT
    command: --init-file /data/application/init.sql
    volumes:
      - db:/var/lib/mysql
      - ./init.sql:/data/application/init.sql
    networks:
      - backend

  backend:
    depends_on:
      - mysqldb
    build: ./be1
    restart: unless-stopped
    env_file: ./.env
    ports:
      - $BACKEND_LOCAL_PORT:$BACKEND_DOCKER_PORT
```

```
    ports:
      - $BACKEND_LOCAL_PORT:$BACKEND_DOCKER_PORT
    environment:
      - DB_HOST=$MYSQLDB_HOST
      - DB_USER=$MYSQLDB_USER
      - DB_PASSWORD=$MYSQLDB_ROOT_PASSWORD
      - DB_NAME=$MYSQLDB_DATABASE
      - DB_PORT=$MYSQLDB_DOCKER_PORT
      - SECRET_KEY=$BACKEND_SECRET_KEY
    stdin_open: true
    tty: true
    networks:
      - backend
      - frontend
  frontend:
    depends_on:
      - backend
    build:
      context: ./ui
      args:
        - REACT_APP_API_BASE_URL=$REACT_APP_API_BASE_URL
    ports:
      - $REACT_LOCAL_PORT:$REACT_DOCKER_PORT
    networks:       You, 13 hours ago • complete
      - frontend
volumes:
  db:
networks:
  frontend:
  backend:
```

Chú ý cần khai báo các biến môi trường của DB và Backend service: thông tin về database (host, user, password, port, database name)