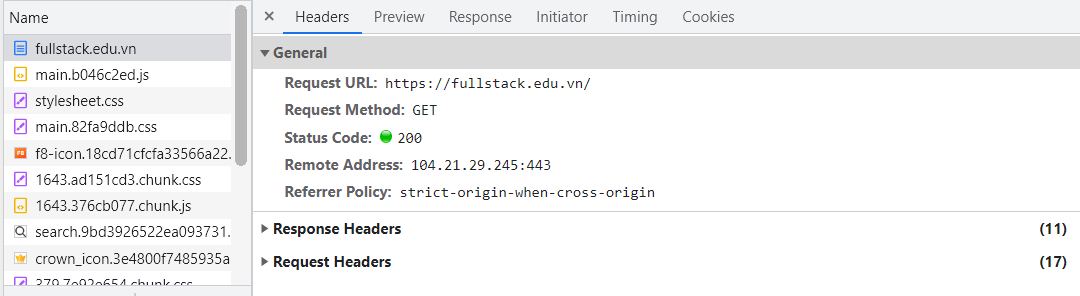
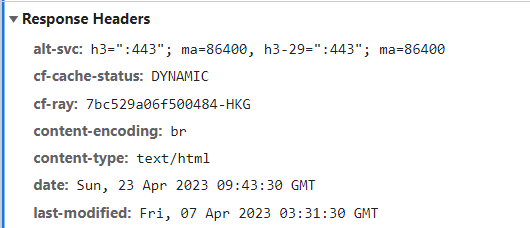
# Mở đầu

Nodejs





Web HTML nhưng dạng động

Đầu 2xx là thành công

304 là ko có sự thay đổi, lấy cache

4xx lỗi client

5xx lỗi server

II. khái niệm sale

Server side rendering

Client side rendering

Server side rendering: sử dụng cho web site dùng trong yếu tố sale, sale là làm tối ưu hóa công cụ tìm kiếm, làm tăng thứ hạng tìm kiếm trên google

Ưu điểm: context của nội dung chứa mã => google bot sẽ tìm kiếm được nội dung web site (tìm từ khóa và tăng thứ hạng)

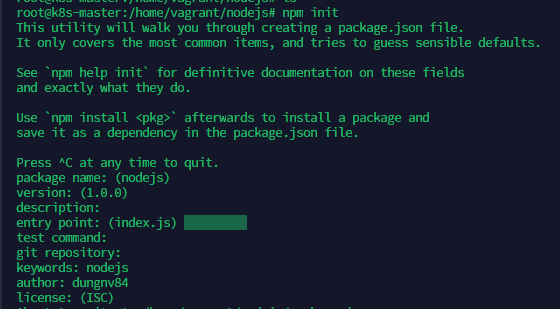
Trả đủ dữ liệu cho client, dữ liệu client trọc vào trang đó là như nhau,

Còn client side rendering thì dữ liệu tùy biến

Client side rendering: chuyển trang mượt mà, ko cần tải lại, ko cần reload lại trang, nhưng ko tốt cho sale

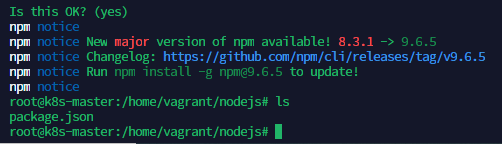
# Khởi tạo package nodejs

npm init



Entry point là đọc file index.js, có thể để thành app.js, …

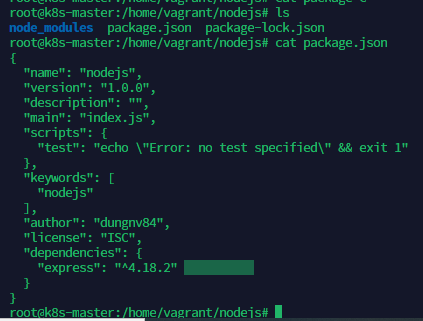
OKE



## Install 1 package

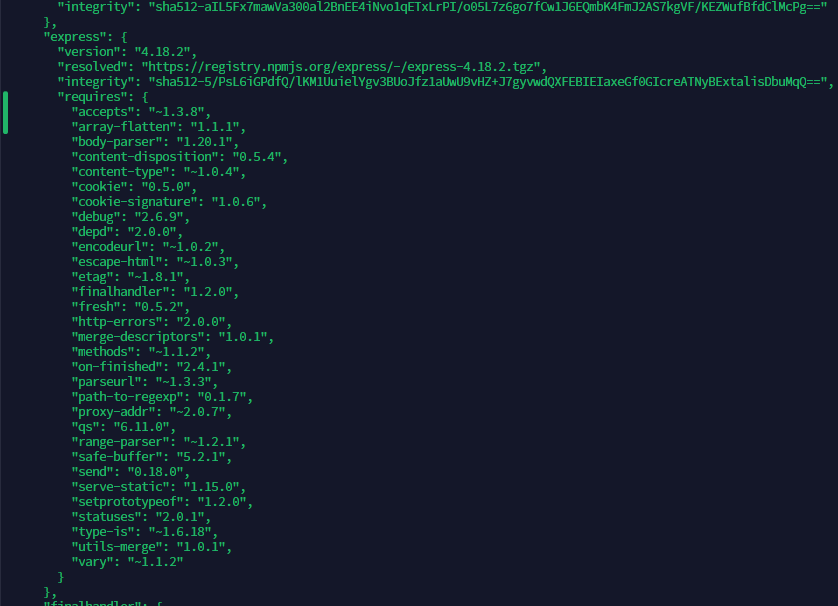
Install express

npm install express



Sau khi install xog thì n sẽ sinh ra folder **node\_modules** và **file package-lock.json** và trong file **package.json** đã được add thêm dependencies là express

Package-lock.json là quản lý sự phụ thuộc của sự phụ thuộc

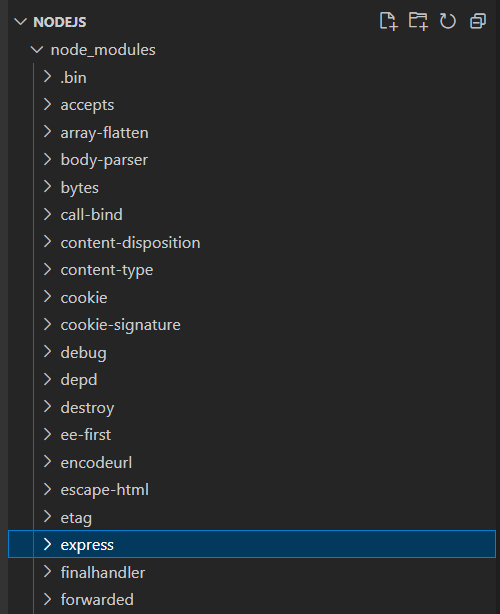


Các package khác sẽ được cài để tương thích với express version kia

* Chạy HELLO WORLD



N tải thư viện trong node\_modules để chạy

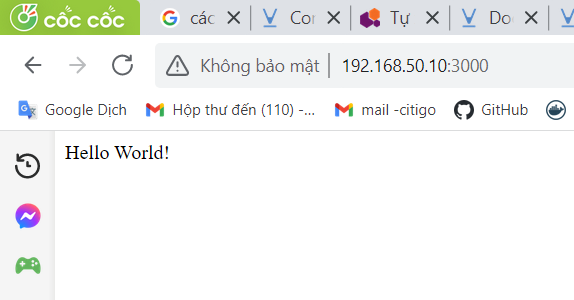
 express sau khi cài sẽ ở trong node module

Vì file main là index.js

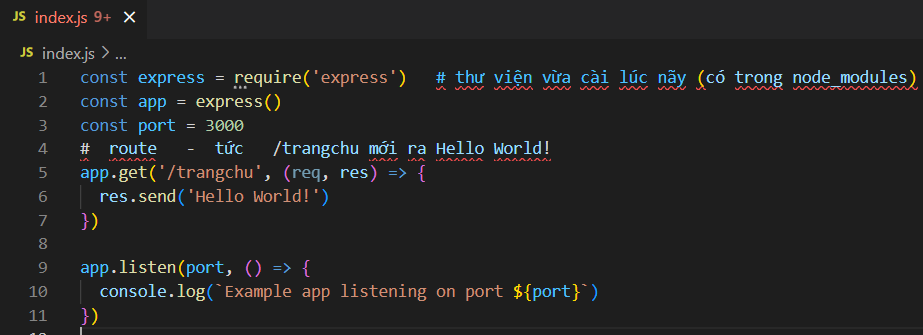
* Để chạy

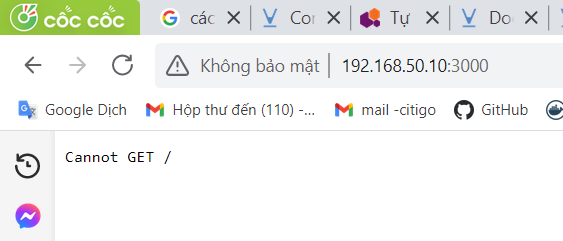
node index.js



 VÀ OKE

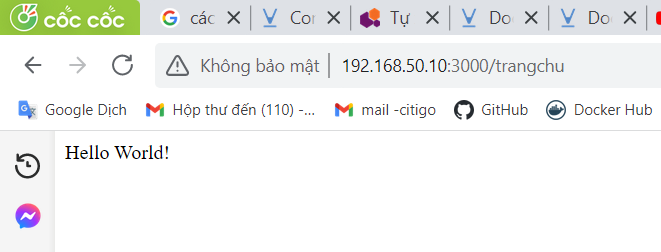
## Route path trong index.js





Cannot GET / là ko đúng path

Phải là **192.168.50.10:3000/trangchu**



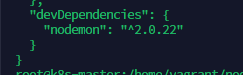
## Nodemon

Nodemon là công cụ phát triển ứng dụng nodejs, n giúp các bạn tự động restart ứng dụng khi mà có sự thay đổi về file trong source code

Ví dụ khi thay đổi path /trangchu -> /home mà nodejs đang chạy thì n sẽ ko được áp dụng ngay mà phải chạy lại node index.js

npm install nodemon --save-dev

có thêm devDependencies



Để ko ảnh hưởng đến server khi chạy mà ko dùng modules của dev

## Chạy nodejs vs docker

Index.js

const express = require('express')   # thư viện vừa cài lúc nãy (có trong node\_modules)

const app = express()

const port = process.env.PORT || 3000;

#  route   -  tức   / mới ra Hello World!

app.get('/', (req, res) => {

  res.send('Hello World!')

})

app.listen(port, () => {

  console.log(`Example app listening on port ${port}`)

})

Dockerfile

FROM node:16.14.0

WORKDIR /app

COPY package.json  .

RUN npm install

COPY  .  .

EXPOSE 4000

CMD ["node", "index.js"]

Chạy vs docker-compose và file .env

docker-compose.yml

version: "3.7"

services:

  app:

    image: vietdung:v1

    build:

      context: .

      dockerfile: dockerfile

    env\_file:

      - .env

    ports:

      - 4000:${PORT}

File .env là

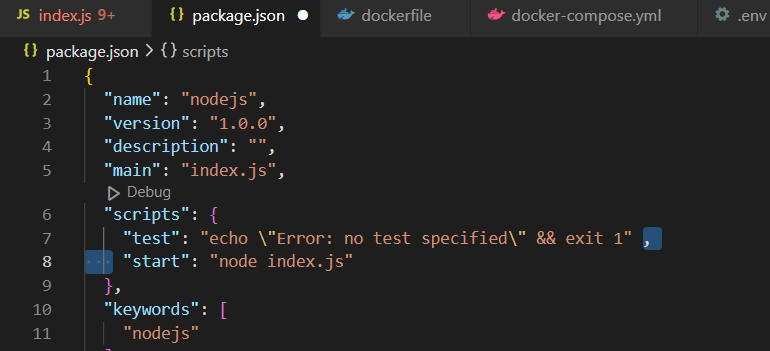


docker-compose up –d --build

## Npm start

npm start thực ra là script của node index.js

đầu tiên update script trong package.json



* Node index.js = npm start

## Docker-compose volume

version: "3.7"

services:

  app:

    image: vietdung:v4

    build:

      context: .

      dockerfile: dockerfile

    env\_file:

      - .env

    ports:

      - 4000:${PORT}

    volumes:

      - ./:/app            # mount dữ liệu ngoài vào bên trong

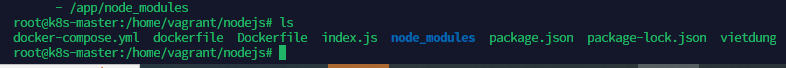
      - /app/node\_modules  # vì ... giữ lại /app/node\_modules trong container để ko bị ghi đè

Hiện tại image này có folder **node\_modules**,

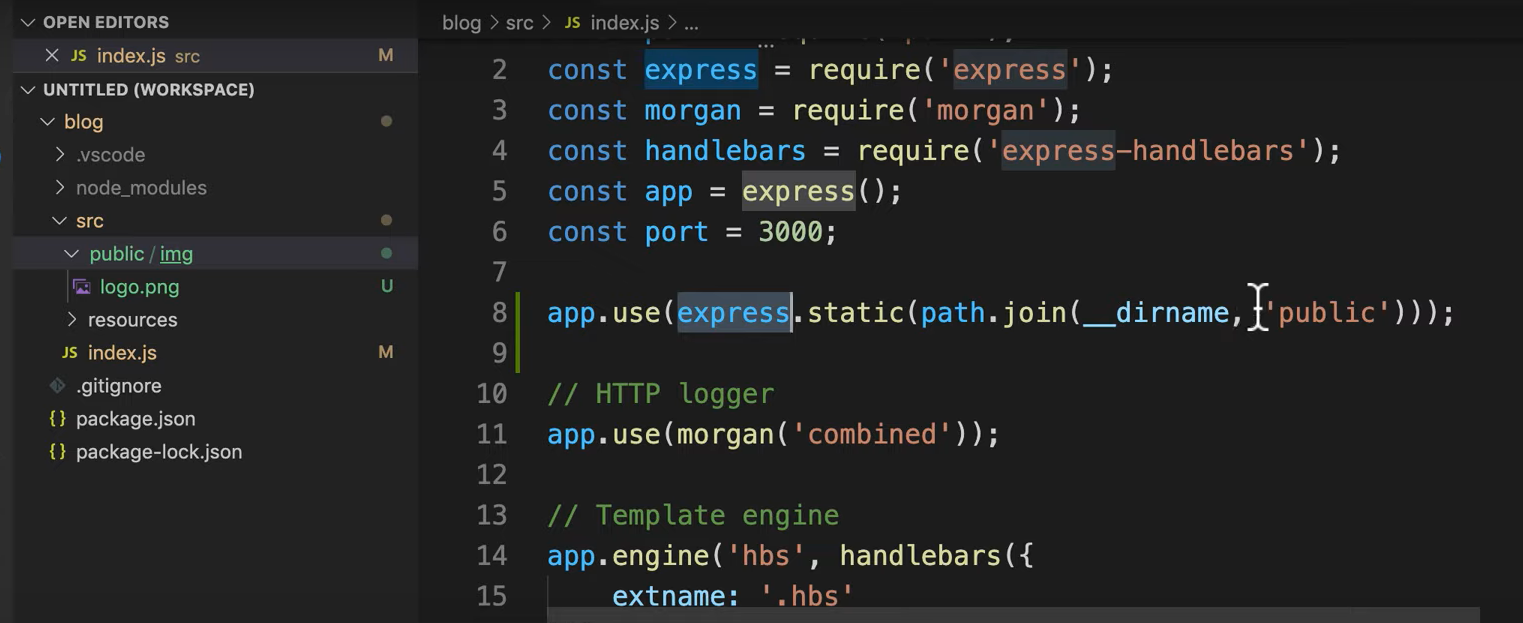
Nhưng với lệnh **./:/app ,** nó sẽ override dữ liệu ở bên ngoài, môi trường gốc vào container,

* Trong container sẽ mất folder **node\_modules**

Vì vậy tôi thêm **/app/node\_modules** để nói rằng khi mount thì folder node\_modules sẽ ko bị overide, tức là ko bị mất khi môi trường gốc ko có folder này, và ngược lại n lại mount node\_modules từ container ra môi trường gốc



# Node js



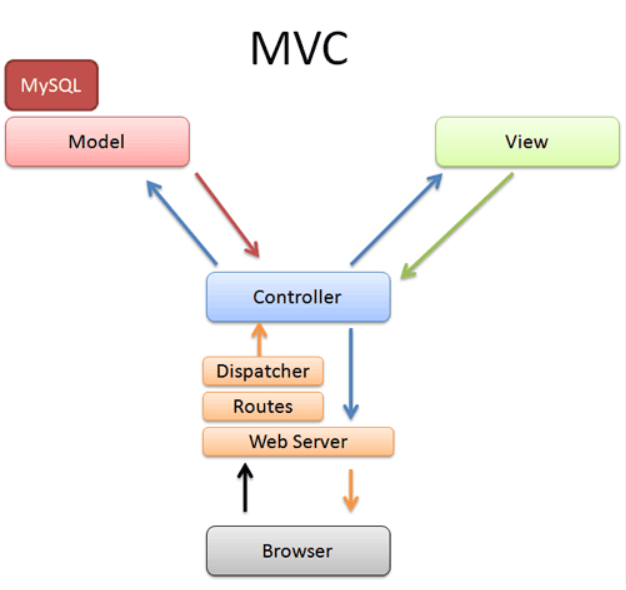
Bthuong file index.js để ở ngoài để chạy , vứt trong folder nào cũng được nhưng phải khai báo path

Tạo 1 folder public/image để chứa ảnh và vứt ảnh vào trong đấy



**Phương thức Get**: nhận dữ liệu từ phía server về phía client

**Phương thức POST**: Ghi dữ liệu, gửi dữ liệu từ client lên server



Browser là client, gửi request đến web server

Webserver vào route -> controller -> DB để get dữ liệu

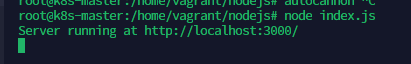
DB respone -> controller -> View -> controller -> web server -> client

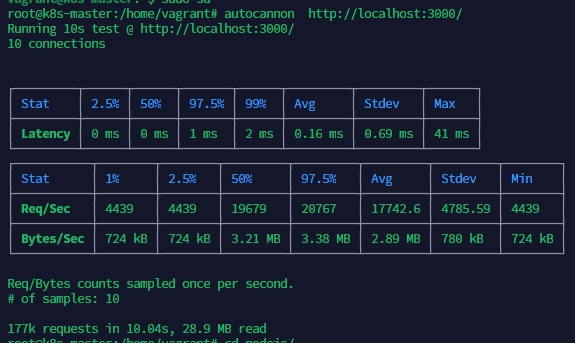
## Giả lập request

Giả lập với **autocannon**

Install autocannon

npm install autocannon –g





117k request trong vòng 10s

Avg: trung bình, trung bình 1s là 17742 request

# Check redis Mysql (sysbench)

# install sysbench

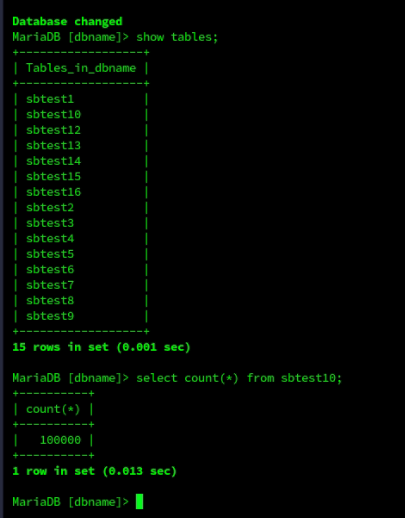
wget -qO - https://packagecloud.io/install/repositories/akopytov/sysbench/script.deb.sh | sudo bash

sudo apt install -y sysbench

# tạo dữ liệu trong mariadb

* Tạo dbname trong mariadb trước

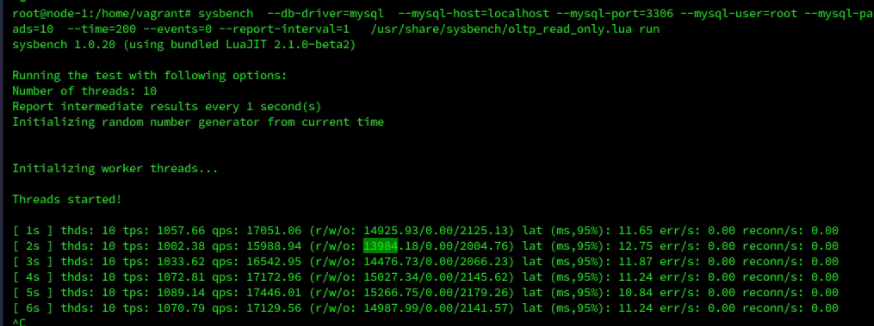
sysbench --db-driver=mysql --mysql-host=localhost --mysql-port=3306 --mysql-user=root --mysql-password=root --mysql-db=dbname --table-size=100000 --tables=16 --threads=10 --time=60 /usr/share/sysbench/oltp\_insert.lua prepare



1 tables có 10000 record

Giờ test xem mỗi giây

sysbench --db-driver=mysql --mysql-host=localhost --mysql-port=3306 --mysql-user=root --mysql-password=root --mysql-db=dbname --table-size=100000 --tables=5 --threads=10 --time=200 --events=0 --report-interval=1 /usr/share/sysbench/oltp\_read\_only.lua run



14925 request mỗi giây

Với redis, mặc định tốc độ đọc mỗi giây là 100k request

keydb-benchmark -t set,get –q

