Contents

[Volume 2](#_Toc178535313)

[Anonymous Volumes xóa libs 5](#_Toc178535314)

[Bind mount read only ro (best pratice) 6](#_Toc178535315)

[ENV 7](#_Toc178535316)

[Delete Volume 9](#_Toc178535317)

[Docker compose (tự động cmd) 10](#_Toc178535318)

[Production 12](#_Toc178535319)

[MongoDB 16](#_Toc178535320)

[Network and DNS mongoose 19](#_Toc178535321)

[Docker compose depend on property 23](#_Toc178535322)

[CRUD 25](#_Toc178535323)

[Redis for auth:1 Signin, up 32](#_Toc178535324)

[Express session 37](#_Toc178535325)

[Application Architecture Review 45](#_Toc178535326)

[NGINX 46](#_Toc178535327)

[cors 49](#_Toc178535328)

[install docker and docker compose anlone in ubuntu machine 50](#_Toc178535329)

[up code to github 51](#_Toc178535330)

[var env ubuntu 52](#_Toc178535331)

[deploy 54](#_Toc178535332)

[update deployment 55](#_Toc178535333)

[rebuild one image if config 56](#_Toc178535334)

[dev production flow 57](#_Toc178535335)

Docker part1

Volume

NNhân: thay đổi code mà ctrình ko update -> tạo container trước rồi mới sửa code, ko update dc

Volume: persistent data in the container, type volume bind: syn folder local hostmachine windowmachine ,… => not create image, up speed

Cli:

* docker run -v pathtofolderlocation:pathtocontainer -p 3000:3000 -d --name node-app node-app-image
* docker run -v D:\Uni\Mr.Q.X.Truong\DevOps\Team\Docker1\node-docker\:/app -p 3000:3000 -d --name node-app node-app-image

cmd:

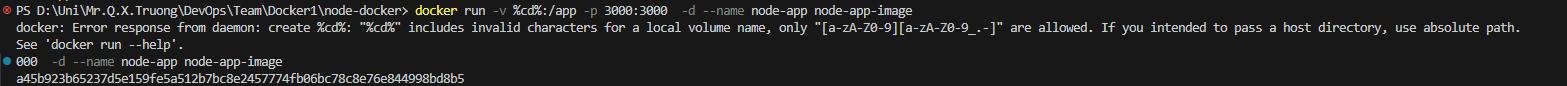
docker run -v %cd%:/app -p 3000:3000 -d --name node-app node-app-image

ps:

docker run -v ${pwd}:/app -p 3000:3000 -d --name node-app node-app-image

Mac, Linux:

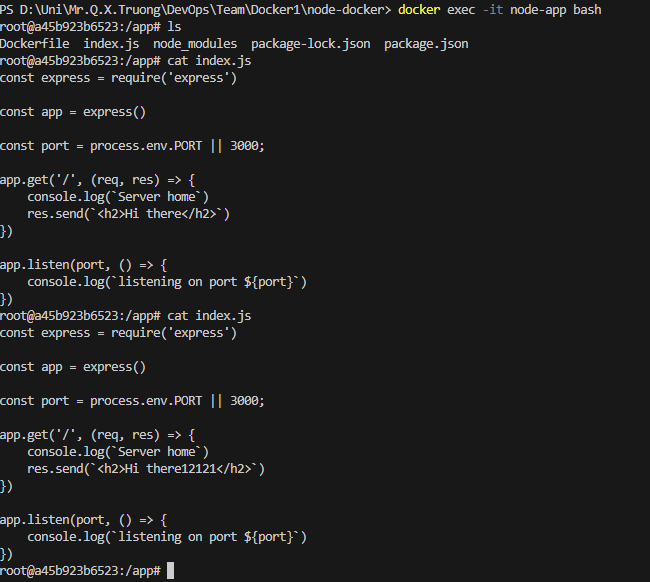
docker run -v $(pwd):/app -p 3000:3000 -d --name node-app node-app-image



-> sync toàn bộ code code index.js với code trong /app

Vấn đề: Thay đổi code trong file index.js lại ko tự build lai

+ kiem tra trong /app -> đã thấy load động





* do ko khởi động lại node js app => dufng nodemoon ở dev deêpnndence, vì cần update package.json + update script

: npm install nodemon --save-dev

  "scripts": {

    "start": "node index.js",

    "dev": "nodemon -L index.js"

  }

"devDependencies": {

    "nodemon": "^3.1.7"

  }

Xóa và build lại:

docker ps

docker rm node-app -f

docker build -t node-app-image .

* vì package.json đã đổi nên sẽ lâu

Update Dockerfile cmd

CMD ["npm", "run", "dev"]

Build again

docker build -t node-app-image .

bind mout lại

docker run -v ${pwd}:/app -p 3000:3000 -d --name node-app node-app-image

* chương trình chạy như dev thường

=>

=>

=>

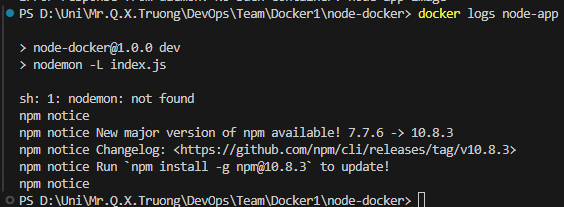
=>

# Anonymous Volumes xóa libs

Delete node modules floder vì ko dev trên localmachine => error

- : docker ps -a : dựa vào time => crash container

-: docker logs node-app



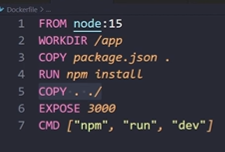
why?

bind mout sync from local folder to /app folder, but deleted libs -> sync delete -> error

=> ngăn ghi dè node\_modules, ngăn bind mout node\_m

docker run -v ${pwd}:/app -v /app/node\_modules -p 3000:3000 -d --name node-app node-app-image

? tối ưu: liệu cần copy all to /app if làm bind mout ?



-> y: bind mout for dev process, bind mout for change code, but production, not. But need copy all code for after version

# Bind mount read only ro (best pratice)

=> volume đang là con đường 2 chiều, tạo file trên local + add file trển /app cũng thay đổi lẫn nhau

=> read only ko cho sửa code: ko tạo được file trong /app

docker run -v ${pwd}:/app:ro -v /app/node\_modules -p 3000:3000 -d --name node-app node-app-image

Xóa node\_modules trong folder -> lỗi với read only

# ENV

Thay đổi trong Docker file

them: chỉ định port + edit:

ENV  PORT 3000

EXPOSE ${PORT}

remove container:

docker ps

docker rm -f node-app

- build again

docker build -t node-app-image .

docker run -v ${pwd}:/app -v /app/node\_modules –-env PORT=4000 -p 3000:4000 -d --name node-app node-app-image

or:

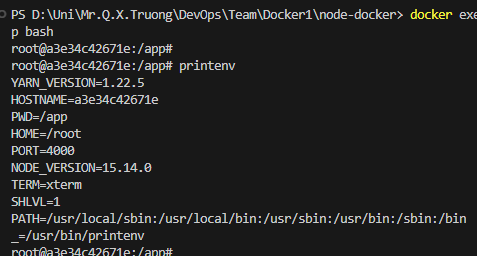
docker run -v ${pwd}:/app -v /app/node\_modules –e PORT=4000 -p 3000:4000 -d --name node-app node-app-image

ghi đè port trong Dockerfile,…

- kiem tra env trong app:

docker exec -it node-app bash

printenv



vấn đ?: quá nhiều var env => long command

create .env:

PORT=4000

- rm container

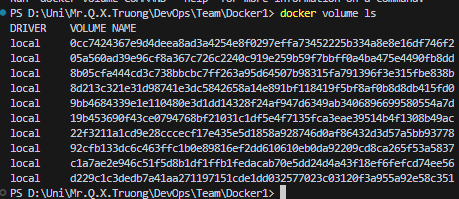
- set var in .env:

docker run -v ${pwd}:/app -v /app/node\_modules --env-file ./.env -p 3000:4000 -d --name node-app node-app-image

# Delete Volume

docker volume ls

->



Why?

docker volume rm 0cc74…

or del with rm:

docker rm node-app -fv

or del many:

docker volume prune

use: docker volume prune

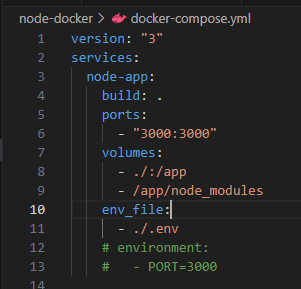
use:docker rm node-app -fv

# Docker compose (tự động cmd)

vì có thể có nhiều container trong dự án -> nhieu cmd

- create file: docker-compose.yml

- content:



tab same py

version -> docs

services = containers

node-app:

+build , path dockerfile

+ ports: mở port s

+ volumes: . current path = ${pwd} or ./:app:ro

persistent folder

+ env\_file | environment:

cmd build and run container: docker-compose up -d

\* name auto: node-docker-node-app

\* port in env must match ports in docker-compose

cmd stop and rm container: docker-compose down -v

-v: del volume

=> del, docker ps

\* compose up again -> faster: cach, tìm image name auto: node-docker….

! thay đổi nhưng ko build new image -> not update container

=> docker-compose up -d --build



? Làm thế nào cho production ? vì production ko bind mount, mà dockercompose có => setup seprate command for dev/production

# Production

solution1: create 2 dockerfile for dev/production

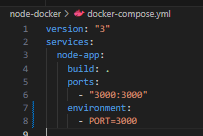
sol2: 1 file

- rename docker-compose.yml ->docker-compose.backup.yml => purpose: create new docker-compose.yml

- create: docker-compose.yml ; docker-compose.dev.yml ; docker-compose.prod.yml

config d-compose : production = dev port

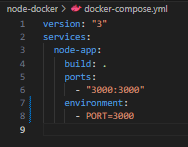
docker-compose.yml



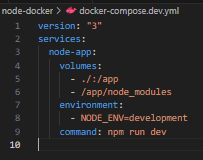
Dockerfile

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

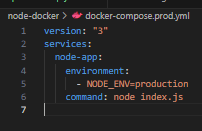
docker-compose.yml



docker-compose.dev.yml



docker-compose.prod.yml



run dev env: có volumes và nodemon

+ cmd:

docker-compose -f docker-compose.yml -f docker-compose.dev.yml up -d –build

+ cmd down

docker-compose -f docker-compose.yml -f docker-compose.dev.yml down -v

run prod env: ko volumes và not auto restart

+ cmd:

docker-compose -f docker-compose.yml -f docker-compose.prod.yml up -d –build

+ cmd down

docker-compose -f docker-compose.yml -f docker-compose.prod.yml down -v

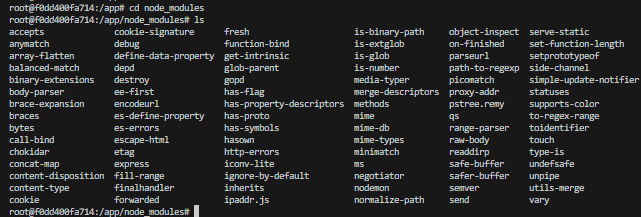
! issue:





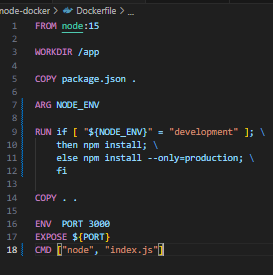
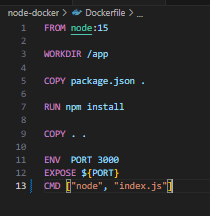
=> add into .dockerignore: docker-compose\*

!issue: nodemon install trong prod envnment, nodemon is dev dependence, waste memory but not work



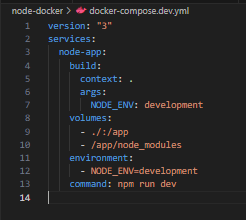


=>+ change Dockerfile run command





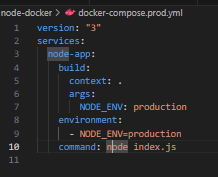
+ change docker-compose.dev.yml





context: path

+ change docker-compose.prod.yml

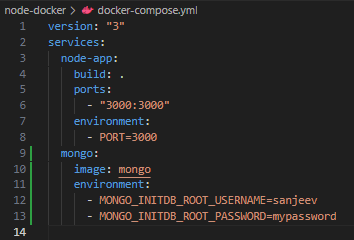




# MongoDB

docs: <https://hub.docker.com/_/mongo>

- change docker-compose.yml



uname, password: freedom

cmd build:

docker-compose -f docker-compose.yml -f docker-compose.dev.yml up -d –build

-> pull image

then run 2 container: node-app + mongo

? connect to mongo:

- go in container:

cmd: docker ps ; + getname mongo

cmd: docker exec -it node-docker-mongo-1 bash

- login: (mongo/mongosh)

cmd: mongosh -u “sanjeev” -p “mypassword”

- check current use db:

db

- create and use db:

use mydb

- show all db:

show dbs

-> mydb not show cause it’s empty

insert to db:

db.books.insertOne({"name": "harry porter"})

show record in db:

db.books.find()

logout:

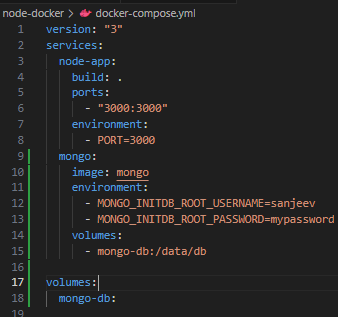
exit

fusion longin to container + login db

docker exec -it node-docker-mongo-1 mongosh -u "sanjeev" -p "mypassword"

!issue: Not save db when down -v

- add volumes in d-compose.yml





out volume ?

re login and create db, insert record

\* down: xóa -v vì -v sẽ xóa volume lưu db=> lost data

=> thỉnh thoảng cmd: docker volume prune

, when container running

# Network and DNS mongoose

mongoose to connect mongodb

- install mongoose: npm install mongoose

\* error change version older

- rebuild:

down: not -v

docker-compose -f docker-compose.yml -f docker-compose.dev.yml down

up:

docker-compose -f docker-compose.yml -f docker-compose.dev.yml up –build

- import mongoose

const mongoose = require('mongoose');

- connect mongoose:

+ lấy Ip conatainer mongo: docker inspect node-docker-mongo…

NetworkSettings.IpAdress: ….

+

27017: port mongodb

const ConnectionString = "mongodb://sanjeev:mypassword@172.18.0.3:27017/?authSource=admin"

mongoose.connect(ConnectionString)

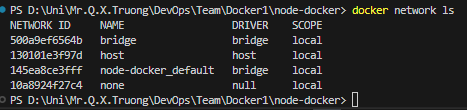
?? them catch

!issue: ko biết cách lấy IpAdress ? dung DNS

Ip: …. => mongo (name image)

docker network ls

docker network inspect name

=> 

name bridge + host : 2 mạng default của docker

- node- app bash

ping mongo

var ENV: IP, …

\* có thể sau này ko làm việc với dns container: chuyển docker container db ra clound

- tạo folder config: + config.js

module.exports = {

    MONGO\_IP: process.env.MONGO\_IP || "mongo",

    MONGO\_PORT: process.env.MONGO\_PORT || 27017,

    MONGO\_USER: process.env.MONGO\_USER,

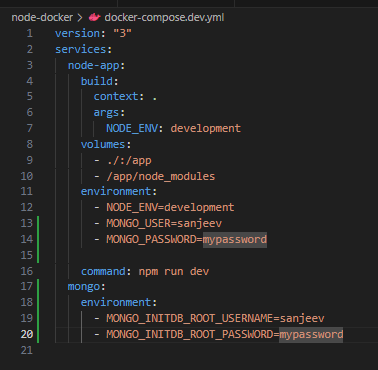
    MONGO\_PASSWORD: process.env.MONGO\_PASSWORD

}

- change code in index.js:

const ConnectionString = `mongodb://${process.env.MONGO\_USER}:${MONGO\_PASSWORD}@${MONGO\_IP}:${MONGO\_PORT}/?authSource=admin`

add env var in .dev.yml



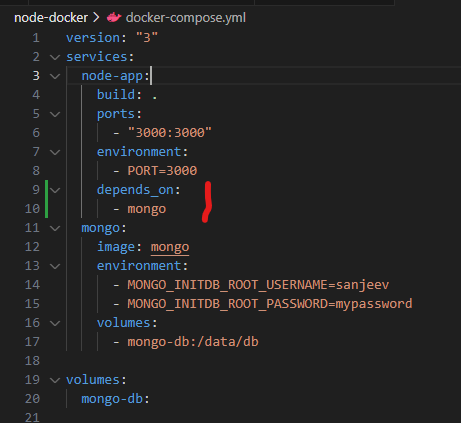


- down and rebuild

# Docker compose depend on property

!issue: chưa có express, run db connect first -> lost

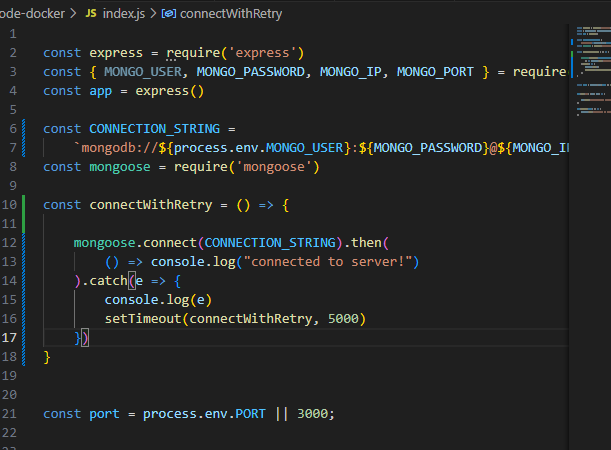
- change docker-compose.yml



-> always mongo run first

issue: ko chac đã init mongo co hay ko, ko có verify -> mongo error -> crash

=> try wait trong 30.secs: ko phải tốt nhất





connectWithRetry()

chạy lại

\* check mongo fist start: cách chứng minh: https://shorturl.at/s3HCg

--help for help

docker-compose -f docker-compose.yml -f docker-compose.dev.yml up -d --no-deps

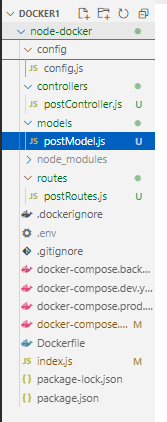
# CRUD

- create models folder

- create controllers folder

- create routes foler

…



postController.js

const Post = require("../models/postModel")

exports.getAllPosts = async (req, res, next) => {

    try {

        const posts = await Post.find()

        res.status(200).json({

            status: 'success',

            results: posts.length,

            data: {

                posts

            }

        })

    } catch (e) {

        res.status(400).json({

            status: "fail"

        })

    }

}

//localhost:3000/posts/:id

exports.getOnePost = async (req, res, next) => {

    try {

        const post = await Post.findById(req.params.id)

        res.status(200).json({

            status: 'success',

            data: {

                post

            }

        })

    } catch (e) {

        res.status(400).json({

            status: "fail"

        })

    }

}

//localhost:3000/posts/

// body->raw->json

exports.createPost = async (req, res, next) => {

    try {

        const post = await Post.create(req.body)

        res.status(200).json({

            status: 'success',

            data: {

                post

            }

        })

    } catch (e) {

        console.log(e)

        res.status(400).json({

            status: "fail"

        })

    }

}

exports.updatePost = async (req, res, next) => {

    try {

        const post = await Post.findByIdAndUpdate(req.params.id, req.body, {

            new: true,

            runValidators: true,

        })

        res.status(200).json({

            status: 'success',

            data: {

                post

            }

        })

    } catch (e) {

        res.status(400).json({

            status: "fail"

        })

    }

}

exports.deletePost = async (req, res, next) => {

    try {

        const post = await Post.findByIdAndDelete(req.params.id)

        res.status(200).json({

            status: 'success'

        })

    } catch (e) {

        res.status(400).json({

            status: "fail"

        })

    }

}

model

const mongoose = require("mongoose")

const postSchema = new mongoose.Schema({

    title: {

        type: String,

        required: [true, "Post must have title"]

    },

    body: {

        type: String,

        required: [true, "Post must have body"]

    }

})

const Post = mongoose.model("Post", postSchema)

module.exports = Post;

router

const express = require("express")

const postController = require("../controllers/postController")

const router = express.Router()

//localhost:3000/

router.route("/").get(postController.getAllPosts)

    .post(postController.createPost)

router.route("/:id").get(postController.getOnePost)

    .patch(postController.updatePost)

    .delete(postController.deletePost)

module.exports = router;

index

const express = require('express')

const {

    MONGO\_USER,

    MONGO\_PASSWORD,

    MONGO\_IP,

    MONGO\_PORT

} = require("./config/config")

const postRouter = require("./routes/postRoutes")

const app = express()

const CONNECTION\_STRING =

    `mongodb://${process.env.MONGO\_USER}:${MONGO\_PASSWORD}@${MONGO\_IP}:${MONGO\_PORT}/?authSource=admin`

const mongoose = require('mongoose')

const connectWithRetry = () => {

    mongoose.connect(CONNECTION\_STRING, {

        useNewUrlParser: true,

        useUnifiedTopology: true

    }).then(

        () => console.log("connected to server!")

    ).catch(e => {

        console.log(e)

        setTimeout(connectWithRetry, 5000)

    })

}

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

connectWithRetry()

app.use(express.json())

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

    res.send(`<h2 style="color: red;">fhdhfdh </h2>`)

})

//localhost:3000/api/v1/post/

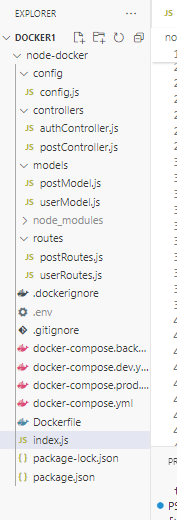
app.use("/api/v1/posts", postRouter)

app.listen(port, () => {

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

})

# Redis for auth:1 Signin, up



install: npm I bcryptjs

rebuild

userModel.js

const mongoose = require('mongoose')

const userSchema = new mongoose.Schema({

    username: {

        type: String,

        require: [true, "User must have a username"],

        unique: true

    },

    password: {

        type: String,

        require: [true, "User must have a password"],

    }

})

const User = mongoose.model("User", userSchema)

module.exports = User

authController.js

const User = require("../models/userModel")

const bcrypt = require("bcryptjs")

exports.signUp = async (req, res) => {

    try {

        const { username, password } = req.body

        const hashPassword = await bcrypt.hash(password, 12)

        const newUser = await User.create({

            username: username,

            password: hashPassword

        })

        res.status(200).json({

            status: "success",

            user: newUser

        })

    } catch (e) {

        res.status(400).json({

            status: "fail"

        })

    }

}

exports.signIn = async (req, res) => {

    try {

        const { username, password } = req.body

        const user = await User.findOne({

            username

        })

        if (!user) {

            return res.status(404).json({

                status: "fail",

                message: "User not found"

            })

        }

        const isCorrect = await bcrypt.compare(password, user.password)

        if (!isCorrect) {

            res.status(200).json({

                status: "fail",

                message: "username or Password not correct"

            })

        } else {

            res.status(200).json({

                status: "success",

            })

        }

    } catch (e) {

        console.log(e)

        res.status(400).json({

            status: "fail"

        })

    }

}

user route

const express = require('express')

const authController = require("../controllers/authController")

const router = express.Router()

router.post("/signup", authController.signUp)

router.post("/signin", authController.signIn)

module.exports = router

index.js

const express = require('express')

const {

    MONGO\_USER,

    MONGO\_PASSWORD,

    MONGO\_IP,

    MONGO\_PORT

} = require("./config/config")

const postRouter = require("./routes/postRoutes")

const userRouter = require("./routes/userRoutes")

const app = express()

const CONNECTION\_STRING =

    `mongodb://${process.env.MONGO\_USER}:${MONGO\_PASSWORD}@${MONGO\_IP}:${MONGO\_PORT}/?authSource=admin`

const mongoose = require('mongoose')

const connectWithRetry = () => {

    mongoose.connect(CONNECTION\_STRING, {

        useNewUrlParser: true,

        useUnifiedTopology: true

    }).then(

        () => console.log("connected to server!")

    ).catch(e => {

        console.log(e)

        setTimeout(connectWithRetry, 5000)

    })

}

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

connectWithRetry()

app.use(express.json())

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

    res.send(`<h2 style="color: red;">fhdhfdh </h2>`)

})

//localhost:3000/api/v1/post/

app.use("/api/v1/posts", postRouter)

app.use("/api/v1/users", userRouter)

app.listen(port, () => {

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

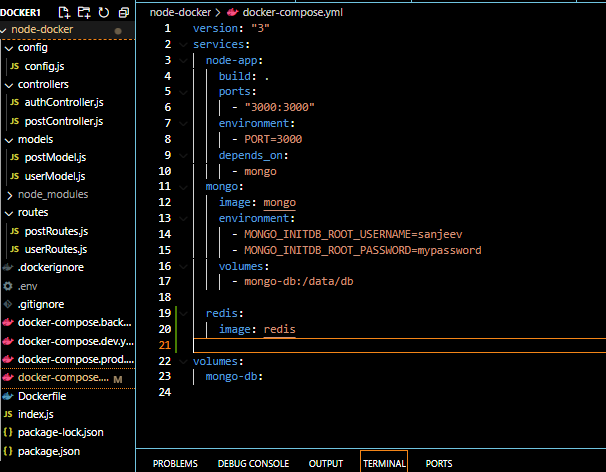
})

# Express session

session/jWT

- search redis in dockerhub

- add redis in docker-compose



rebuild again:

install redis connect

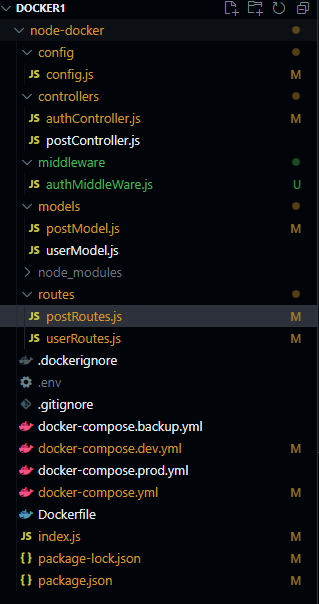
npm install redis connect-redis express-session

Why 3?

docker compose -f docker-compose.yml -f docker-compose.dev.yml up -d --build -V

= down + up –build

change code



index

**const express = require('express')**

**const {**

**MONGO\_USER,**

**MONGO\_PASSWORD,**

**MONGO\_IP,**

**MONGO\_PORT,**

**REDIS\_URL,**

**REDIS\_PORT,**

**SESSION\_SECRET**

**} = require("./config/config")**

**const session = require('express-session')**

**const redis = require("redis")**

**let RedisStore = require("connect-redis")(session)**

**let redisClient = redis.createClient({**

**legacyMode: true,**

**url: `redis://${REDIS\_URL}:${REDIS\_PORT}`**

**})**

**const postRouter = require("./routes/postRoutes")**

**const userRouter = require("./routes/userRoutes")**

**const app = express()**

**const CONNECTION\_STRING =**

**`mongodb://${MONGO\_USER}:${MONGO\_PASSWORD}@${MONGO\_IP}:${MONGO\_PORT}/?authSource=admin`**

**const mongoose = require('mongoose')**

**const connectWithRetry = () => {**

**mongoose.connect(CONNECTION\_STRING, {**

**useNewUrlParser: true,**

**useUnifiedTopology: true**

**}).then(**

**() => console.log("connected to server!")**

**).catch(e => {**

**console.log(e)**

**setTimeout(connectWithRetry, 5000)**

**})**

**}**

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

**connectWithRetry()**

**redisClient.connect()**

**app.use(**

**session({**

**store: new RedisStore({ client: redisClient }),**

**secret: SESSION\_SECRET,**

**cookie: {**

**secure: false,**

**resave: false,**

**saveUninitialized: false,**

**httpOnly: true,**

**maxAge: 30000**

**}**

**})**

**)**

**app.use(express.json())**

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

**res.send(`<h2 style="color: red;">hasdyq </h2>`)**

**})**

**//localhost:3000/api/v1/post/**

**app.use("/api/v1/posts", postRouter)**

**app.use("/api/v1/users", userRouter)**

**app.listen(port, () => {**

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

**})**

docker-comopose.yml

**version: "3"**

**services:**

**node-app:**

**build: .**

**ports:**

**- "3000:3000"**

**environment:**

**- PORT=3000**

**depends\_on:**

**- mongo**

**mongo:**

**image: mongo**

**environment:**

**- MONGO\_INITDB\_ROOT\_USERNAME=sanjeev**

**- MONGO\_INITDB\_ROOT\_PASSWORD=mypassword**

**volumes:**

**- mongo-db:/data/db**

**redis:**

**image: redis**

**volumes:**

**mongo-db:**

d compose.dev

**version: "3"**

**services:**

**node-app:**

**build:**

**context: .**

**args:**

**NODE\_ENV: development**

**volumes:**

**- ./:/app**

**- /app/node\_modules**

**environment:**

**- NODE\_ENV=development**

**- MONGO\_USER=sanjeev**

**- MONGO\_PASSWORD=mypassword**

**- SESSION\_SECRET=secret**

**command: npm run dev**

**mongo:**

**environment:**

**- MONGO\_INITDB\_ROOT\_USERNAME=sanjeev**

**- MONGO\_INITDB\_ROOT\_PASSWORD=mypassword**

postRoutes

**const express = require("express")**

**const postController = require("../controllers/postController")**

**const protect = require("../middleware/authMiddleWare")**

**const router = express.Router()**

**//localhost:3000/**

**router.route("/").get(protect, postController.getAllPosts)**

**.post(protect, postController.createPost)**

**router.route("/:id").get(protect, postController.getOnePost)**

**.patch(protect, postController.updatePost)**

**.delete(protect, postController.deletePost)**

**module.exports = router;**

authMiddleware

**const protect = (req, res, next) => {**

**const { user } = req.session**

**if (!user) {**

**return res.status(400).json({**

**status: "fail", message: "unauthorized"**

**})**

**}**

**req.user = user**

**next()**

**}**

**module.exports = protect**

authController

**const User = require("../models/userModel")**

**const bcrypt = require("bcryptjs")**

**exports.signUp = async (req, res) => {**

**try {**

**const { username, password } = req.body**

**const hashPassword = await bcrypt.hash(password, 12)**

**const newUser = await User.create({**

**username: username,**

**password: hashPassword**

**})**

**req.session.user = newUser**

**res.status(200).json({**

**status: "success",**

**user: newUser**

**})**

**} catch (e) {**

**res.status(400).json({**

**status: "fail"**

**})**

**}**

**}**

**exports.signIn = async (req, res) => {**

**try {**

**const { username, password } = req.body**

**const user = await User.findOne({**

**username**

**})**

**if (!user) {**

**return res.status(404).json({**

**status: "fail",**

**message: "User not found"**

**})**

**}**

**const isCorrect = await bcrypt.compare(password, user.password)**

**if (!isCorrect) {**

**res.status(200).json({**

**status: "fail",**

**message: "username or Password not correct"**

**})**

**} else {**

**res.status(200).json({**

**status: "success",**

**})**

**}**

**} catch (e) {**

**console.log(e)**

**res.status(400).json({**

**status: "fail"**

**})**

**}**

**}**

config

**module.exports = {**

**MONGO\_IP: process.env.MONGO\_IP || "mongo",**

**MONGO\_PORT: process.env.MONGO\_PORT || 27017,**

**MONGO\_USER: process.env.MONGO\_USER,**

**MONGO\_PASSWORD: process.env.MONGO\_PASSWORD,**

**REDIS\_URL: process.env.REDIS\_URL || "redis",**

**REDIS\_PORT: process.env.REDIS\_PORT || 6379,**

**SESSION\_SECRET: process.env.SESSION\_SECRET**

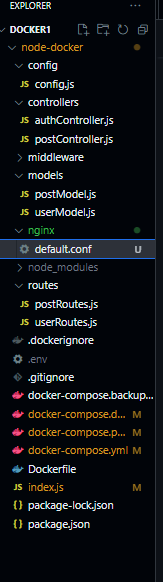
**}**

session nodejs

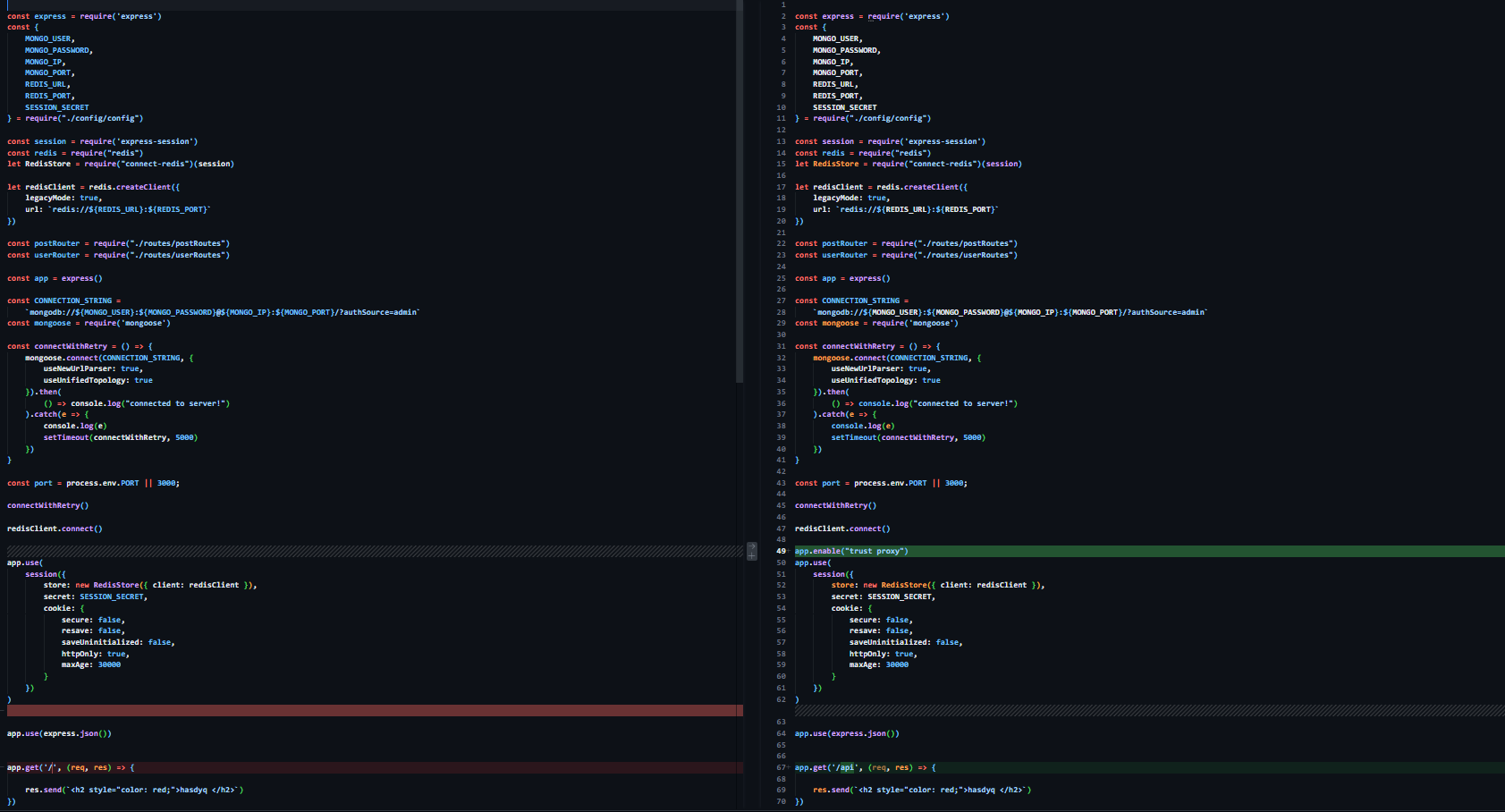
# Application Architecture Review

# NGINX

create folder nginx/default.conf to config this container



change index.js



default.conf

**server {**

**listen 80;**

**server\_name localhost;**

**location /api {**

**proxy\_pass http://node-app:3000;**

**proxy\_set\_header X-Real-IP $remote\_addr;**

**proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;**

**proxy\_set\_header Host $http\_host;**

**proxy\_set\_header X-Forwarded-Proto $scheme;**

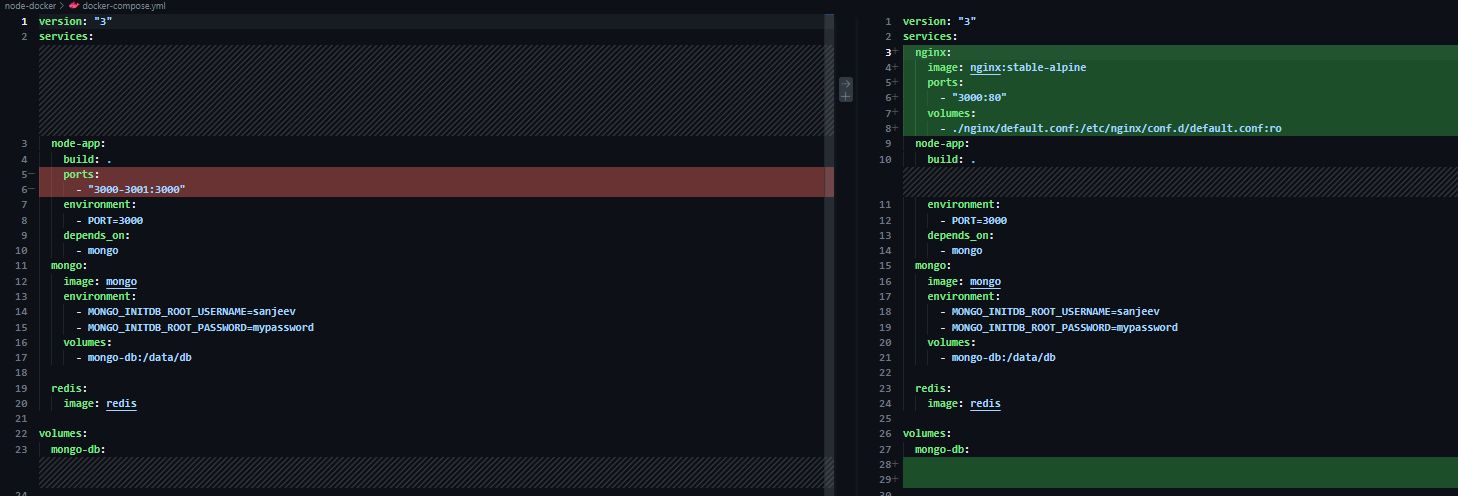
**proxy\_set\_header X-NginX-Proxy true;**

**proxy\_redirect off;**

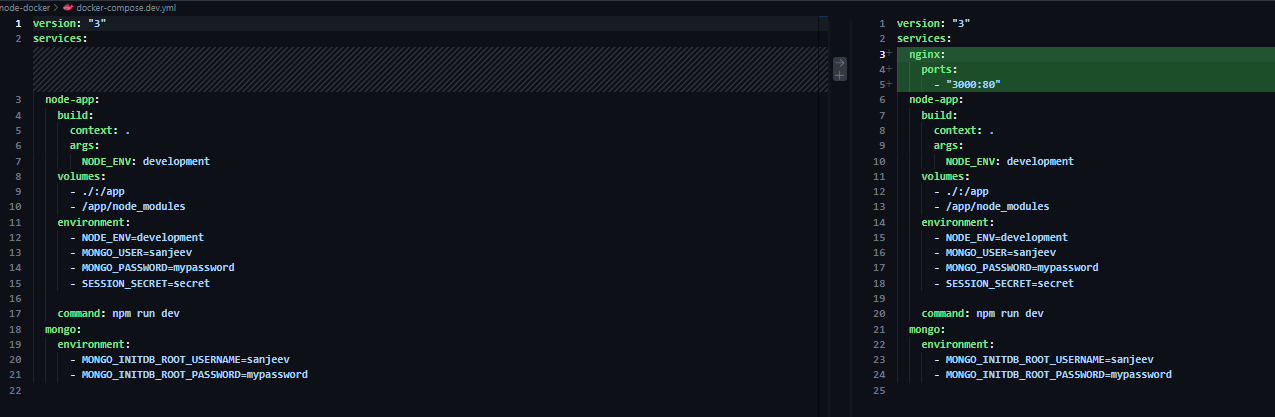
**}**

**}**

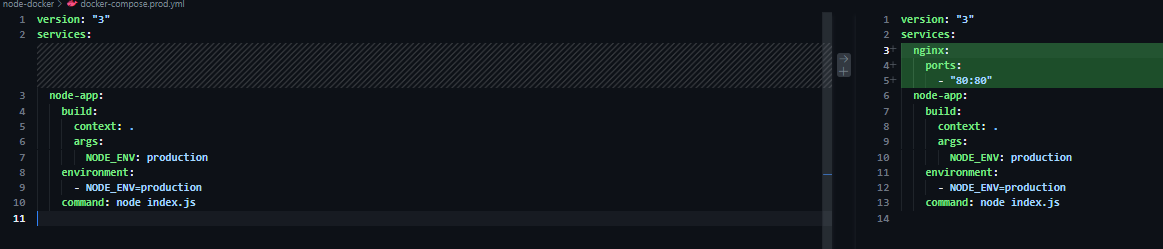
docker-compose.yml



docker-compose.dev.yml



docekr-compose.prod.yml



# cors

install cors:

npm I cors

require cors



# install docker and docker compose anlone in ubuntu machine

access vps by ip

change root user/ sudo

indtruction install docker:

get.docker.com

instruction install docker-compose :

<https://docs.docker.com/compose/install/standalone/>

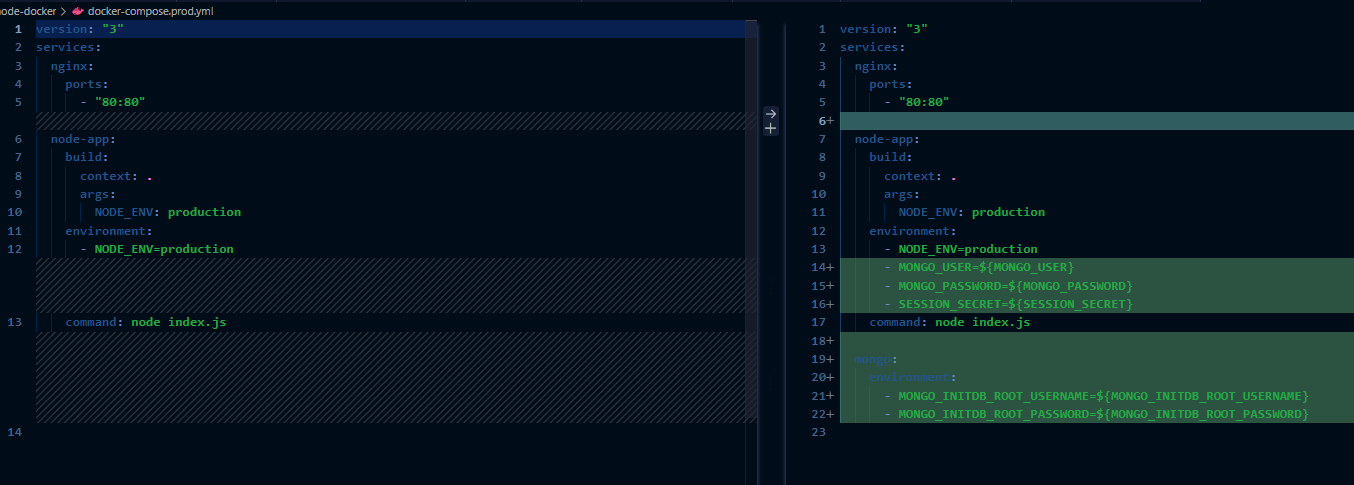
bug : not found curl

bug; cant resolve host: github

# up code to github

# var env ubuntu

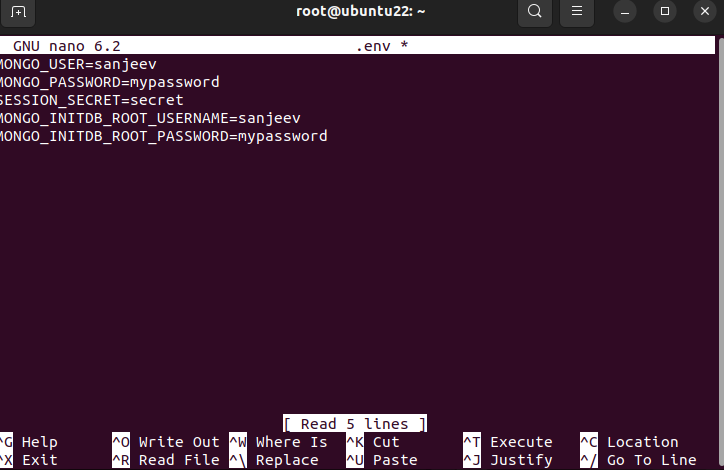
change docker-..prod.yml



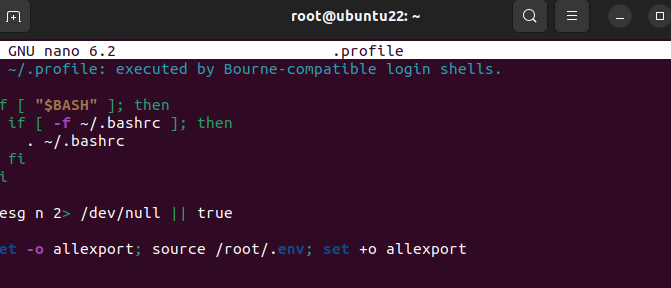
ubuntu

use root user, sudo pwd-> root

- create .env to hold variables, nano .env



- ls -la, export in .profile



restart ubuntu

printenv

# deploy

create app-> git clone …git .

run docker-compose.yml , docker-compose.prod.yml up

error: mongodb cannot cerate->down grade image

error: cannot find vm ip -> changetype Network, get ip: route -n

<https://www.youtube.com/watch?v=L8NbXczVJwo>

replace localhost to that ip, test postman

# update deployment

use git

docker-compose ….. up -d --build --no-deps node-app

run again app no dependency

# rebuild one image if config

docker-compose … up -d --no-deps --force-recreate node-app

# dev production flow

issue: build image in vm -> waste resource and cpu

=> build image from dev to dHUb and let server pull and rebuild new image

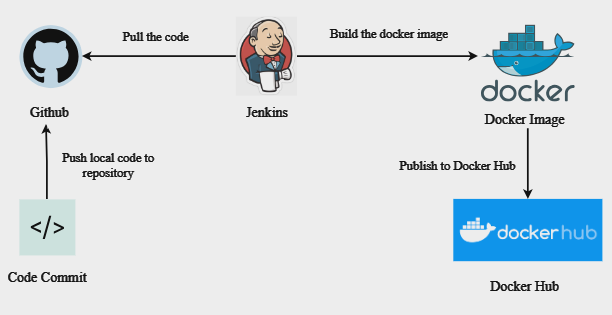
<https://www.youtube.com/watch?v=5NgoqaMwt2E&list=PL8VzFQ8k4U1JEu7BLraz8MdKJILJir7oY&index=28>

kịch bản:

Đội phát triển cần môi trường

Khách hang

Mô hình:



Nhiều lệnh Xây dựng, cấu hình khó khan

Cần nhiều hiểu biết về môi trường hoạt động cách thức hoạt động của mạng   
hiểu biết về dự án

công việc tương lai

các khái niệm cơ bản quy trình hoạt động của docker học them kiến thức liên quan mạng máy tính

thực hành nhiều dự án vừa và nhỏ sau đó nâng cao

triển khai môi trường chung

triển khai môi trường devol

triển khai dự án trên máy host