CLOUD DEVELOPER

Udacity Nanodegrees

**Author:** Pham Tat Dat - DatPT45

**Update 09/23**: TrungLV10, HuyN73

Giới thiệu tổng quan và một số thông tin cần chú ý khi học và tham gia làm project.

# Overview

Khóa học bao gồm 4 modules, tương ứng với 4 project bao gồm:

* Cloud Fundamentals: Giới thiệu cơ bản về AWS
* Full Stack Apps on AWS: Giới thiệu về cách lâp trình, viết test case, stora data, build &deploy một Full stack app trên AWS
* Monolith to Microservices at Scale: Giới thiệu về kiến trúc Microservice, docker, k8s và cách deploy 1 ứng dụng Microservice với docker và K8s. Ngoài ra, bạn cũng sẽ làm quen với khái niệm CI/CD trong quá trình phát triển phần mềm (basic)
* Develop & Deploy Serverless App: Làm quen với Rest API, deploy ứng dụng sử dụng serverless framework trên AWS, các best practices khi deploy ứng dụng với serverless framework

Mỗi project sẽ là một bước nhỏ hướng tới việc giúp cho lập trình viên có thể phát triển ứng dụng trên Cloud, cụ thể là AWS, từ ứng dụng Monolith tới Microservice. Với 2 project đầu khá là basic, sẽ cần nhiều kiến thức về AWS hơn 1 chút, nhưng 2 project sau sẽ cần khá nhiều về kiến thức lập trình Typescript. Lưu ý, cần bám sát từng bài giảng và exercise, các phần này sẽ giúp làm project nhanh hơn, đặc biệt là Project 04.

Ngoài ra, cuối khóa sẽ có 1 Capstone project. Capstone project là project tổng hợp, yêu cầu bạn phải sử dụng hoặc Project 3 hoặc Project 4 để làm, chú ý follow theo Project Rubric.

# Prerequisite

Khóa học sẽ cần một số kiến thức về các mảng sau:

* AWS Skills (chủ yếu về các service phổ biến như IAM, S3, Lambda, API Gateway, CloudFormation)
* Programing: Nodejs
* IDE: Visual studio code
* AWS CLI
* Serverless
* Auth0 account
* Có kiến thức về security là một lợi thế

Tất cả các project sẽ làm lab thực tế trên AWS, môi trường do Udacity cung cấp.

**Attention**

Mỗi người sẽ được cấp 1 Account AWS với Budget $75 cho toàn khóa học, và 1 lần xin extend Budget $25. Nếu vượt quá số này bạn sẽ phải tự sử dụng môi trường riêng để làm. Nên mọi người cần chú ý một số vấn đề:

* Cố gắng làm project gói gọn trong 1-3 ngày, tránh kéo dài vì resource đã tạo để lâu sẽ càng mất nhiều chi phí, toàn bộ resource đang tính tiền theo model PAY AS YOU GO, nghĩa là cứ tạo là phải trả phí, dùng bao nhiêu trả bấy nhiêu
* Luôn luôn xóa hết resource sau khi làm xong 1 project, có thể vào từng service trên AWS Console để xóa.
* Project 03 - Monolith to Microservices at Scale là project làm mọi người tốn nhiều chi phí nhất do tạo rất nhiều resource EC2. Đặc biết lưu ý thời gian làm project này. Cụ thể cách để giảm chi phí cho project này mình sẽ nói ở phần sau.

Khi chuẩn bị submit project để review, hãy dành thời gian đọc Project Rubric và xem xem mình đã làm đủ các yêu cầu trong Rubric chưa. Trừ phần standout suggestion thì các phần còn lại trong Rubric phải đáp ứng hết thì mới có thể **PASS** được.

**Lưu ý**: khi làm nhóm, bài nộp của các thành viên không được giống nhau, bị chấm đạo văn thì thành viên đó bị block account không cho submit bài nữa.

# Project 01: Cloud Fundamentals

Trong học phần này bạn sẽ được học về các khái niệm cơ bản về Cloud Computing, một số service gắn với domain quan trong trong cloud như:

* EC2, S3, CloudFront, DynamoDB
* Security (AWS Firewall, AWS Shield và IAM)
* Networking (Route53, Autoscaling, ALB)
* Messaging & Containers: SNS, SQS, Docker, ECS
* AWS management: Cloutrail, Cloudformation, AWS CL

**Project01**: Public 1 ứng dụng cơ bản lên Cloud, sử dụng S3, CloudFront và IAM policy để secure S3 bucket:

Với project này, mình đánh giá khá là cơ bản, mọi người chỉ cần follow theo từng bước trong project là hoàn thành. *Estimated time*: 2h

# Project 02: Full Stack Apps on AWS

Trong học phần này bạn sẽ tìm hiểu các nguyên tắc cơ bản về thiết kế, triển khai và cung cấp dịch vụ cho các ứng dụng đơn giản trên Cloud bao gồm:

* Design app logic và database
* Viết các function và test case
* Implement authentication, implement JWT
* Deploy ứng dụng với Elastic Beanstalk
* Làm quen và hiểu cách hoạt động của DNS, CDN Một số kiến thức cần:
* Python
* Nodejs
* Github
* AWS CLI
* Postman

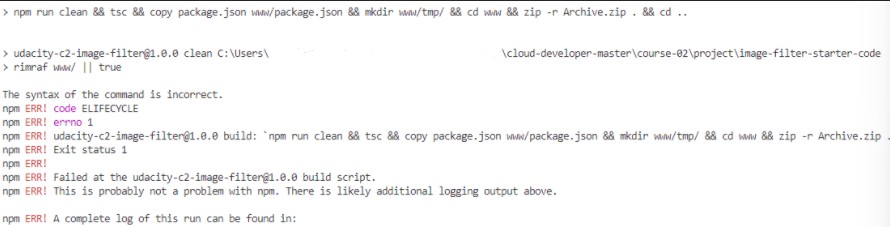
**Project02**: Chỉnh sửa một project có sẵn “Image Filter” cho phép download Image với URL. User nhập URL image, hệ thống trả về ảnh.

Một số lưu ý:

* Cần verify URL user nhập vào (http validator)
* Sử dụng trực tiếp credential của IAM user mà cấp mà không cần tạo IAM user mới, dễ phát sinh lỗi thiếu quyền
* Code cần lưu trên github để tracking và submit project Đối với phần standout suggestion:
* Add basic authen cho api
* Sử dụng Route53 để add custom domain, tuy nhiên bạn phải có sẵn domain riêng của mình và add vào Route53, nếu không có thì bỏ qua cũng được.

Nhìn chung project này không yêu cầu code nhiều, khoảng 20 dòng là có thể pass, tập trung vào phần deploy ứng dụng với Elastic Beanstalk, làm theo hướng dẫn là có thể pass.

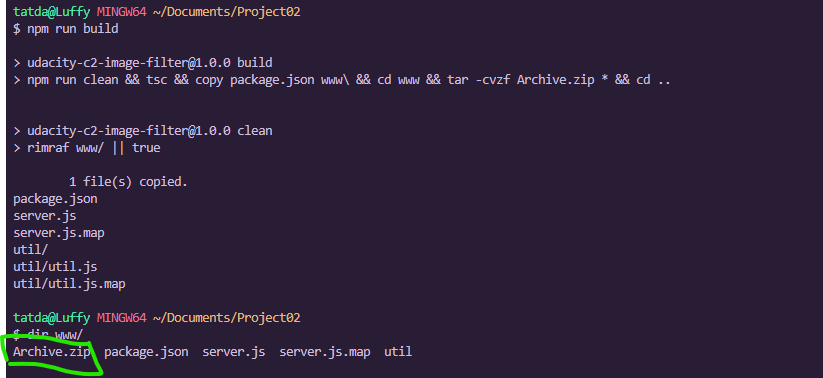
**Note**: Một số bạn sử dụng máy Windows để làm sẽ gặp lỗi khi build tương tự như sau:



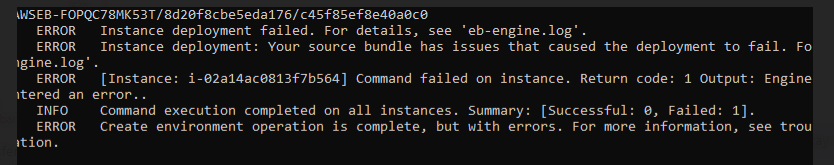
Lỗi này xuất hiện do Windows không có lệnh **ZIP.** Có thể tham khảo cách sau:

* Hướng dẫn fix từ Udacity trong course material: [Building and Deploying (udacity.com)](https://learn.udacity.com/nanodegrees/nd9990/parts/cd0353/lessons/9f582f4a-6bea-4b07-9cc0-00eb6e20b8b9/concepts/6b058fe5-6cd2-4281-9867-c569f2d99ab0) – Note for Windows Users
* Hoặc thay lệnh Build trong file package.json bằng lệnh sau. Lệnh này mình tự chỉnh sửa và thay thế dựa trên mục đích của lệnh trên linux.

npm run clean && tsc && copy package.json www\\ && cd www && tar -cvzf Archive.zip \* && cd ..



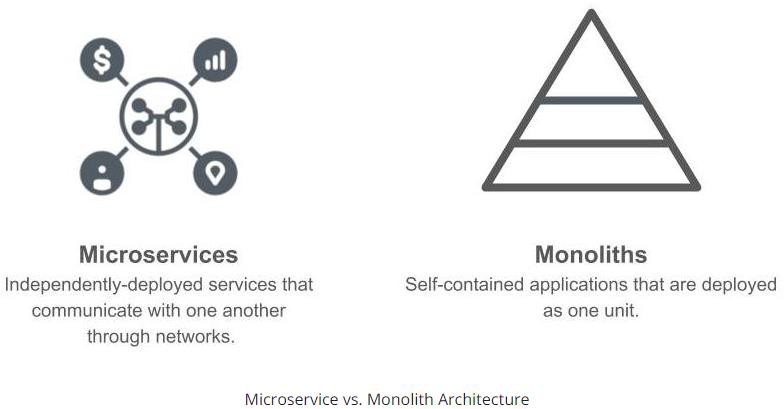
* Hoặc nếu thử các cách trên ko được, vẫn còn lỗi sau:



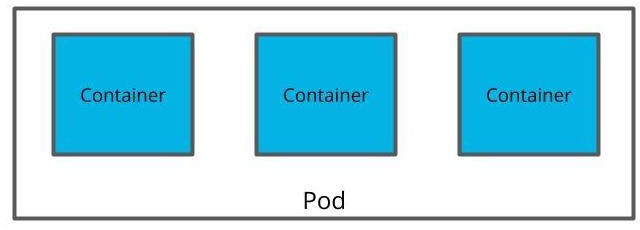
* + Sửa lại lệnh build trong file package.json thành: "npm run clean && tsc && cp package.json www/package.json"
  + Build lại, sau đó vào thư mục /www rồi thực hiện zip manual các file trong thư mục / www thành file Archive.zip

# Project 03: Monolith to Microservices at Scale

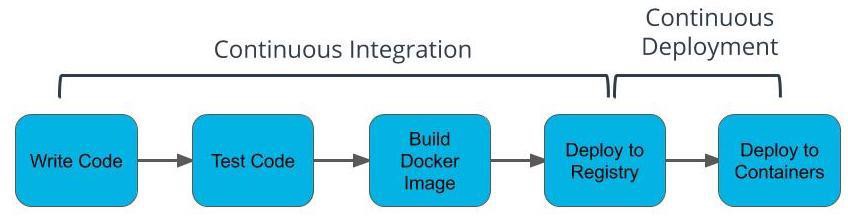
Học phần này tiếp nối Project 2 tuy nhiên sẽ có sử thay đổi lớn trong kiến trúc ứng dụng và kiến trúc triển khai. Nếu như project 2 là ứng dụng Monolith thì trong phần này sẽ hướng dẫn chuyển từ ứng dụng Monolith sang microservice. Bạn sẽ được giới thiệu và hiểu thế nào là Microservice, ưu nhược điểm, các deploy một ứng dụng Microservice sao cho hiệu quả.



Bạn sẽ được học về container, Docker và Kubenestes, những công nghệ về Container Orchestration đang phổ biến và sử dụng rộng rãi trên thế giới.



Bạn sẽ được học về Automating the Application Development Lifecycle – CI/CD

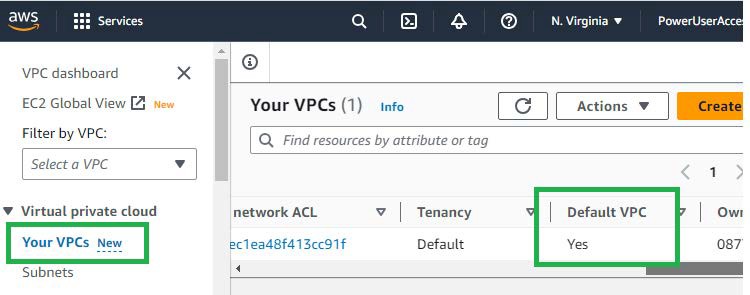


**Prerequisites:** để làm Project 3, các bạn chuẩn bị:

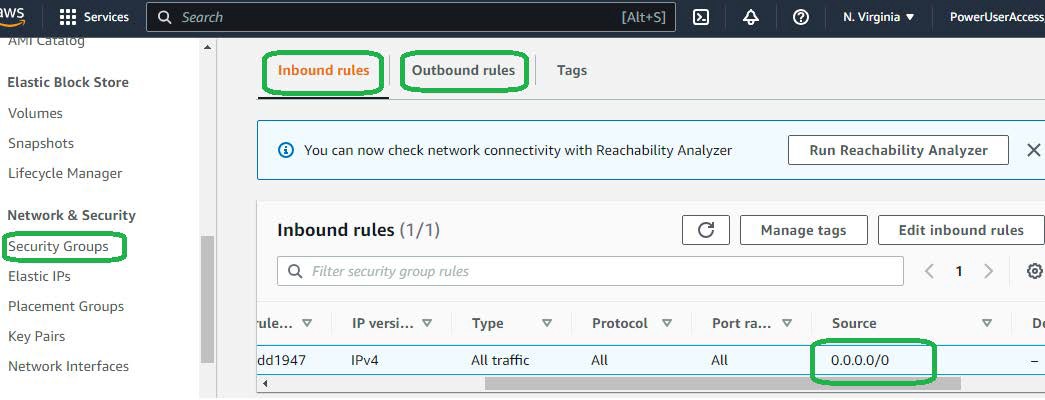
* Git
* PostgreSQL 14.8: [https://www.enterprisedb.com/downloads/postgres-postgresql-](http://www.enterprisedb.com/downloads/postgres-postgresql-) downloads (lưu ý chúng ta chỉ cần cài Comand line tools)
* Download source code
* Docker (trong khoá học hưỡng dẫn sử dụng docker để chạy ở local, tức ko nhất thiết phải cài docker, mình khuyến khích các bạn cài để test và hiểu cơ chế làm việc)
* Kubectl: Để manage k8s (https://kubernetes.io/docs/tasks/tools/#kubectl)
* AWS CLI: để giao tiếp với AWS
  + sau khi cài aws command line, cần phải cấu hình profile ở local để tương tác với AWS(các bạn tham khảo phần Project Overview trong phần "Refactor Monolith to Microservices and Deploy" của khoá học)

**Getting Started**: phần này chúng ta thao tác trên AWS console, Step nào mình đánh **(\*)** thì các bạn cứ follow trong khoá học là làm được.

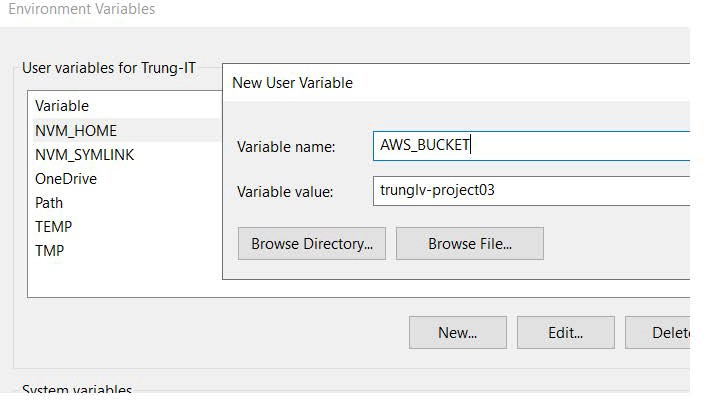
* Create an S3 bucket:
  + Set Bucket policy (\*)
  + Set CORS configuration (\*)
* Create PostgreSQL database, các bạn thực hiện theo các steps:
  + Trước tiên cần có default VPC (nếu chưa có các bạn phải tạo)



* + Create security group (trong E2C)
    - setting Inbound/Outbound to 0.0.0.0/0
    - Tham khảo:



* + Create RDS (\*)
  + Kiểm tra kết nối từ PostgreSQL command line (\*)
* Set up the Environment variables (\*)
  + Tuỳ theo hệ điều hành từng máy mà cách setup môi trường khác nhau
  + Ví dụ trên Windown:



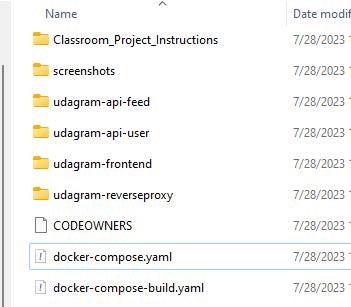
**Part 1**: Monolithic application

* Backend (\*)
* Frontend (\*)
  + Lưu ý ở step ["ionic build"](https://docs.aws.amazon.com/eks/latest/userguide/service_IAM_role.html) [có thể sẽ báo lỗi](https://docs.aws.amazon.com/eks/latest/userguide/create-node-role.html) phiên bản nodejs đã cũ, nó bắt update nodejs mới nhất thì các bạn skip qua step này, ko cần update, các step trong **part 1** để chạy ở local nên các bạn đọc hiểu.

**Part 2** Microservices application

* Refactor the Backend API (\*)
* Refactor the Frontend Application (\*)
* Create Reverseproxy (\*)
* Create docker-compose-build.yaml (\*)
* Create docker-compose.yaml (\*)

**=>**follow theo hướng dẫn các bạn có cây thư mục sau:



* Sau đó Build docker-compose-build.yaml file và run docker-compose.yaml file (bước này test Docker containers ở local thôi)
* Lưu ý: **part 2** chạy để test local nên các bạn cũng có thể đọc hiểu.

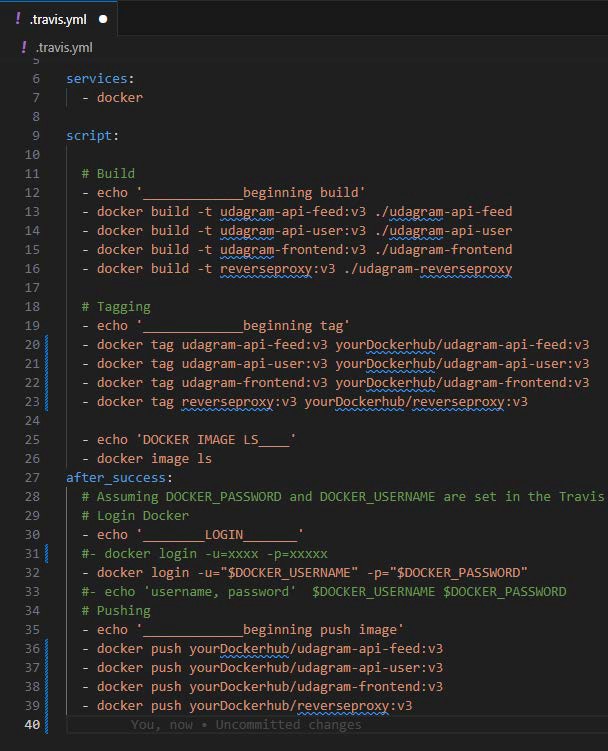
**Part 2** Troubleshoot (phần này flollow đọc hiểu các tình huống xử lý lỗi trong các **Part** trước đó)

**Part 3** Continuous Integration

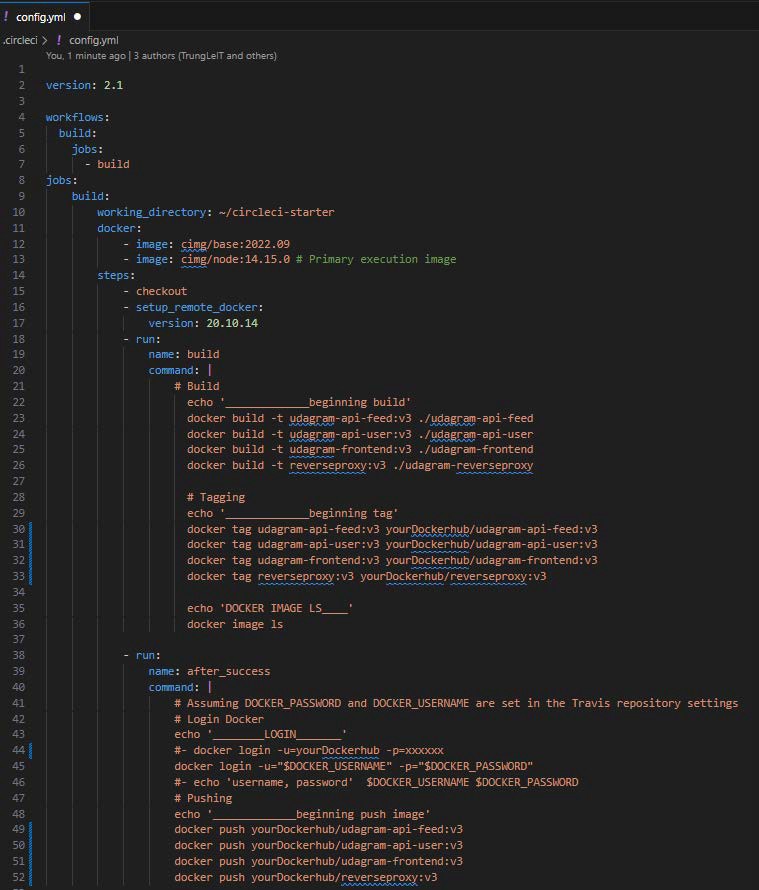
* Create Dockerhub Repositories (\*)
* Set up CI/CD (travis ci cần có thẻ credit card mới làm được, nếu ko có thẻ credit các bạn có thể dùng circle ci hoặc github action)
  + Travis CI: Sau khi tạo account liên kết với github, các bạn chọn repo github của mình để CI/CD

P Tham khảo file cấu hình bên dưới (lưu ý trong Travis CI các bạn nhớ set biến môi trường DOCKER\_USERNAME và DOCKER\_PASSWORD trỏ tới dockerhub của các bạn, Circle CI cũng tương tự)

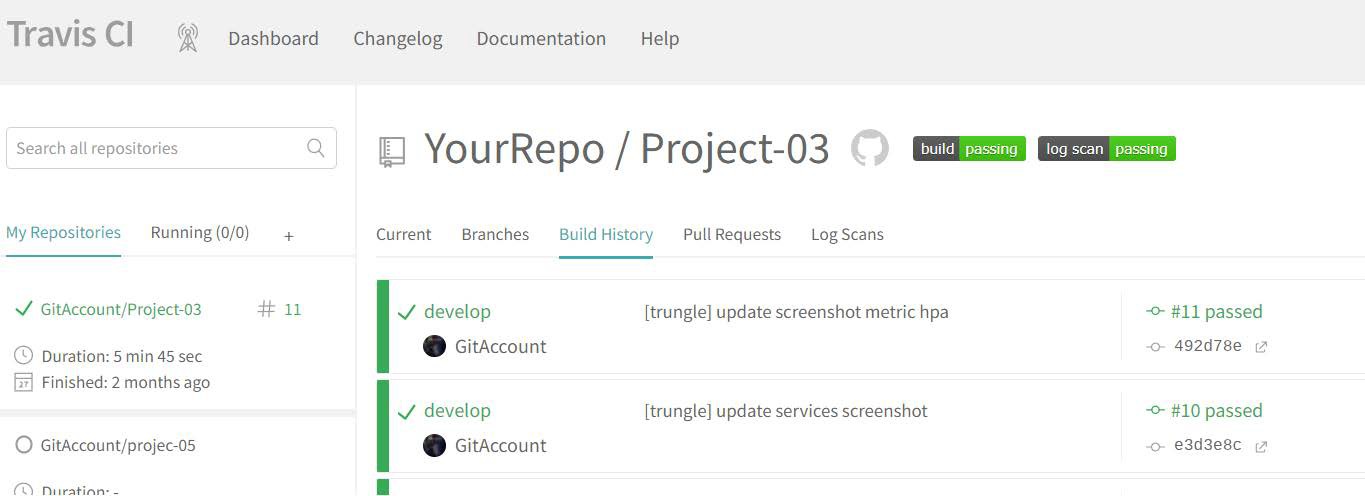
* Đây là 2 config travis ci và circle ci các bạn tham khảo để test:



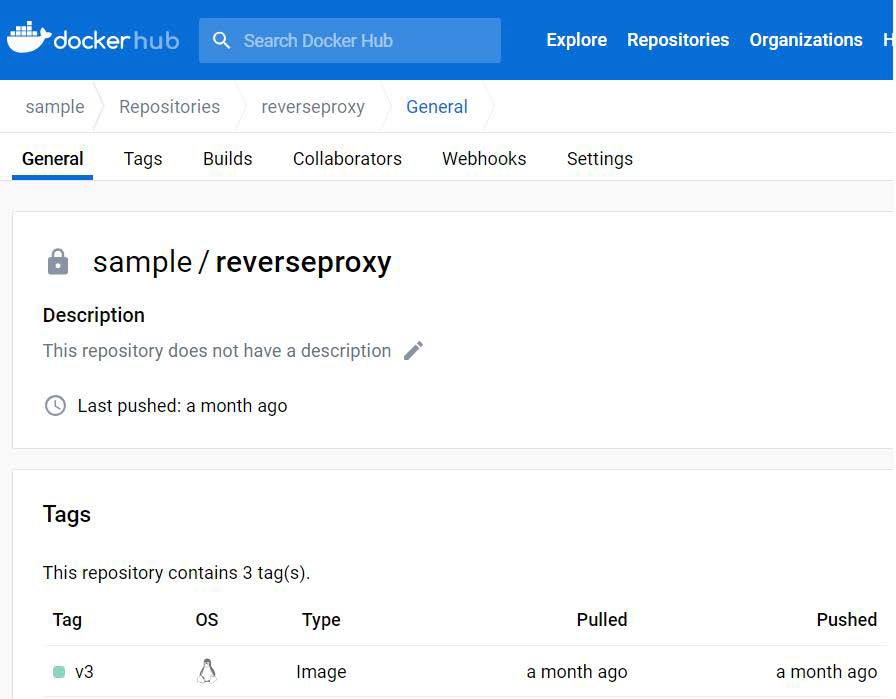
* + Circle CI:



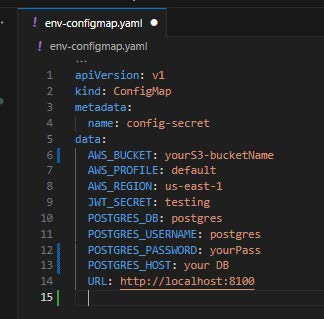
* Và kết quả CI/CD thành công

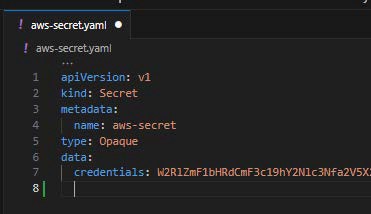


* Docker hub lúc này đã có image tương ứng:

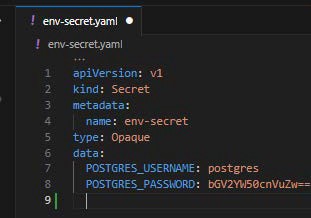


**Part 4** Container Orchestration with Kubernetes

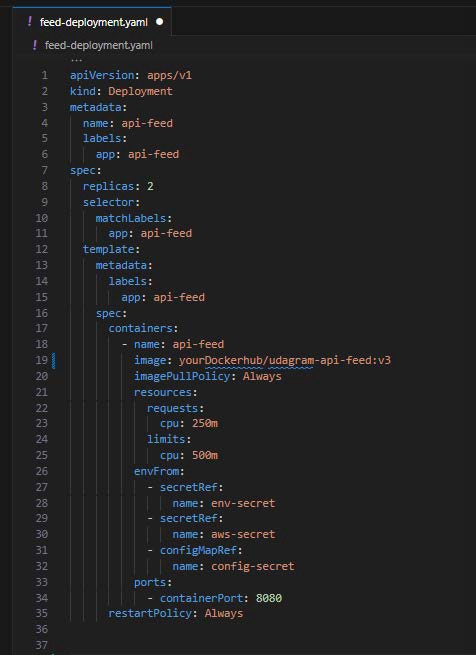
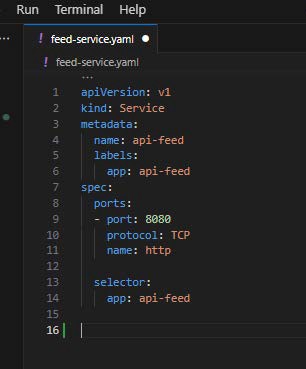
* Các bạn cứ làm theo trình tự như sau (\*):
  + Tạo EKS cluster IAM role: https://docs.aws.amazon.com/eks/latest/userguide/ service\_IAM\_role.html#create-service-role
  + Tạo cluster https://docs.aws.amazon.com/eks/latest/userguide/create-cluster.html
  + Tạo roles node group: https://docs.aws.amazon.com/eks/latest/userguide/create- node-role.html
  + Tạo node group https://docs.aws.amazon.com/eks/latest/userguide/create- managed-node-group.html
  + Tạo cube config: https://docs.aws.amazon.com/eks/latest/userguide/create- kubeconfig.html
    - run command: aws eks update-kubeconfig --name yourCluster --region us- east-1 --profile yourProfile
  + Deployment:
    - create aws-secret.yaml
    - create env-configmap.yaml



* + - create env-secret.yaml
    - create nginx.conf (tham khảo udagram-reverseproxy)



* + - feed-deployment.yaml
    - feed-service.yaml



* + - Các file deployment và service còn lại tìm hiểu và

làm tương tự:

* + - * frontend-deployment.yaml
      * frontend-service.yaml
      * reverseproxy-deployment.yaml
      * reverseproxy-service.yaml
* Sau khi tạo xong thì deploy lên K8s bằng lệnh kubectl apply -f <yaml\_file>

Enable Horizontal Pod Autoscaling (HPA)

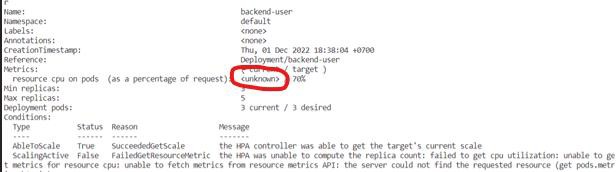
* Chạy lệnh sau để enable HPA:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| kubectl autoscale deployment <deployment\_name> --cpu-percent=70 --min= | 2 | --max= | 3 |  |

Trong đó:

* cpu-percent : Ngưỡng để scale
* min: Num of minimum pod
* max: Num of maximum pod

Nếu status CPU là Unknow như này, bạn install metric server vào k8s để có thể show CPU metrics



Để cài đặt chạy lệnh

kubectl

apply

-f

https://github.com/kubernetes-sigs/metrics-

server/releases/latest/download/components.yaml

Nếu gặp lỗi trong quá trình deploy, check mục Troubleshoot trong bài, có một số lỗi phổ biến và câu lệnh hay dùng để tìm lỗi.

**Một số lệnh phổ biến:**

|  |  |  |
| --- | --- | --- |
| **STT** | **Command** | **Description** |
| 1 | kubectl get deployment | Get all deployment |
| 2 | kubectl get pods | Get all pods |
| 3 | kubectl get svc | Get all service |

|  |  |  |
| --- | --- | --- |
| 4 | kubectl get hpa | Get all HPA |
| 5 | kubectl describe deployment  <deployment\_name> | Xem chi tiết về 1 deployment |
| 6 | kubectl describe pod <pod\_name> | Xem chi tiết về 1 pod |
| 7 | kubectl exec --stdin --tty <pod\_name> --  /bin/bash | SSH vào 1 pod |

# Project 04: Develop & Deploy Serverless App

Project 04 là học phần cuối trong khóa học, tập trung dạy về việc phát triển và triển khai ứng dụng thông qua serverless.

Nội dung khóa học bao gồm:

* Giới thiệu về serverless
* Event processing
* Xác thực phân quyền trên serverless
* Best practice
* Deploy ứng dụng với serverless trên cloud sử dụng API Gateway, Lambda

Phần này theo mình thì khá nặng và nhiều kiến thức, từ việc sử dụng serverless framework, cách sử dụng với DynamoDB, indexing; phân quyền, validator …. Với các phần trước có thể học lướt bỏ một số nội dung vẫn có thể làm được project nhưng phần này bắt buộc phải học kỹ.

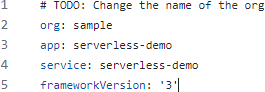
## Chuẩn bị account, env

* 1. Auth0 account: https://manage.auth0.com/
  2. Github account
  3. NodeJS version 14.15.0 (quá trình mình làm thì version này là ổn, các version khác có thể dẫn đến lỗi)
  4. Serverless
     1. Create account serverless: https://dashboard.serverless.com/
     2. Install Serverless Framework’s CLI (suggest version 3)
        + npm install -g serverless@3
        + serverless –version (check serverless version)

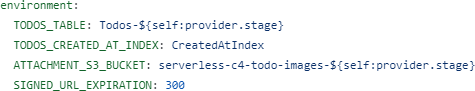
## Backend

Link tham khảo Serverless Application/Backend: [guide](https://learn.udacity.com/nanodegrees/nd9990/parts/cd0352/lessons/ea62b697-059a-40bc-a3a0-e8c2cf584d4f/concepts/9383e2cc-c549-45cc-bc39-7af2bcac89a7)

* 1. File serverless.yml
* Các bạn điền các thông tin như framework version, profile, runtime (same nodejs version)

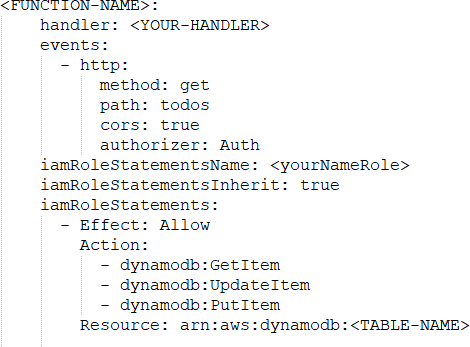


* Cập nhật environment

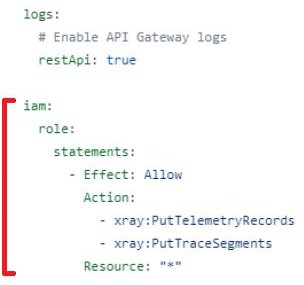


**ATTACHMENT\_S3\_BUCKET** phải là duy nhất. Nếu bucket name đã tồn tại, thì quá trình deploy app sẽ lỗi

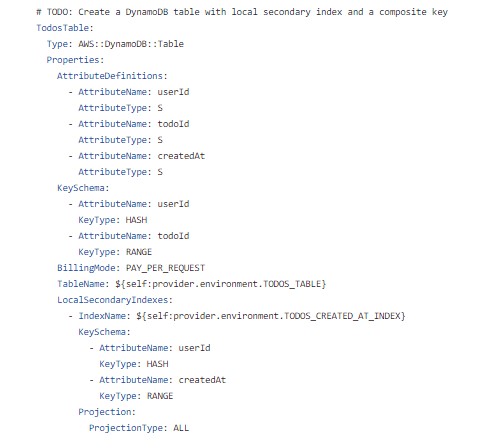
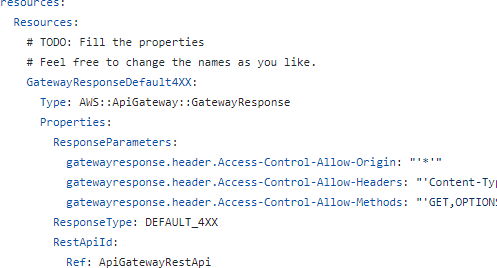
* + Thêm role iamRoleStatements cho functions:



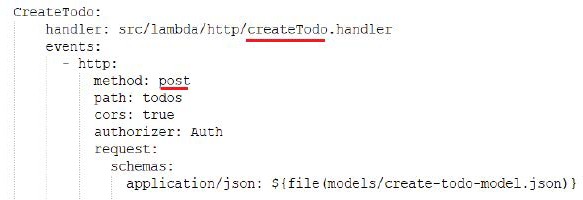
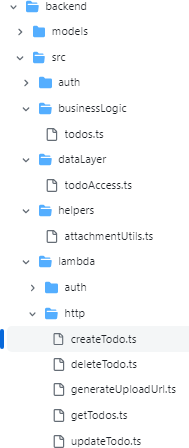
Lưu ý: iamRoleStatementsInherit: true, nghĩa là bạn cũng cần khai báo iam role, cú pháp như sau:



* + - AWS Resources (tham khảo Serverless Application/Backend) mẫu tham khảo:

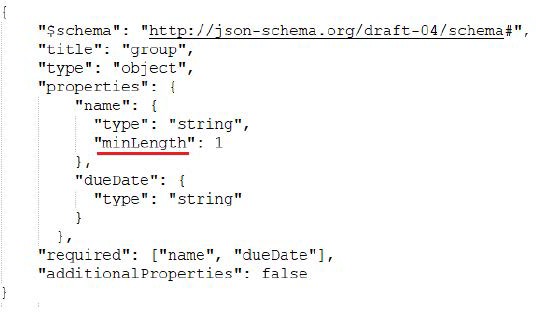


2.5. Thêm Validate min length



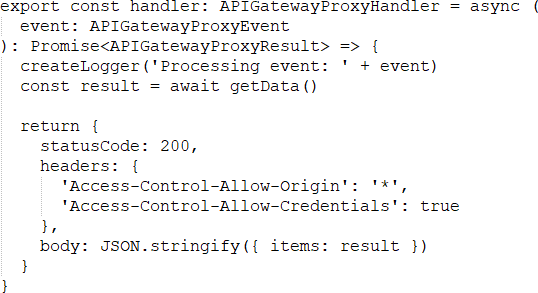
* 1. File attachmentUtils (cd src/helpers)
  2. Implement Todos, TodoAccess
  3. cd Lambda/http
     + CRUD Todo
     + GenerateUploadURL
     + The file name and method in lambda/http must be same in serverless.yml:

Mình gợi ý các bạn validate by schema request validator:

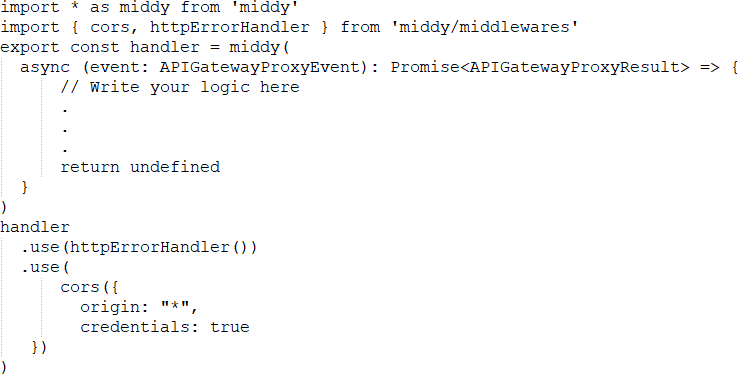


2.6. Cấu hình CORS Configuration trong handler functions, có 2 cách:

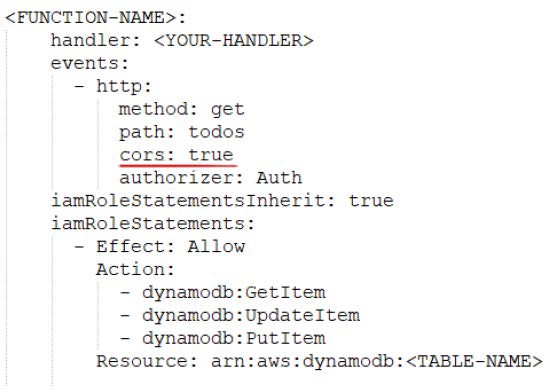
* Dùng **CORS** header



* Dùng **middy** middleware



Và thêm CORS trong file serverless.yml

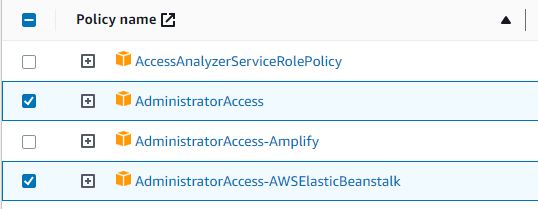


## Client

* 1. Login vào Auth0
  2. Tạo một "Single Page Web Applications" type Auth0 application
  3. Đi đến App settings thiết lập Allowed CallBack URLs, Allowed Web Origins for CORS option

## Run and Deploy

* Tham khảo Serverless Application/Deploy: [guide](https://learn.udacity.com/nanodegrees/nd9990/parts/cd0352/lessons/ea62b697-059a-40bc-a3a0-e8c2cf584d4f/concepts/e313de78-fcb3-4332-a8c4-b8bc34c4688c)
* Trên trang AWS, tạo IAM User với 2 Role như dưới



* Sau khi tạo user thành công, bạn click on user => chọn tab Security credentials

=> Create Access keys => Choose use case Command Line Interface (CLI)

* Mở cmd cd đến backend của project:
  + Login serverless:



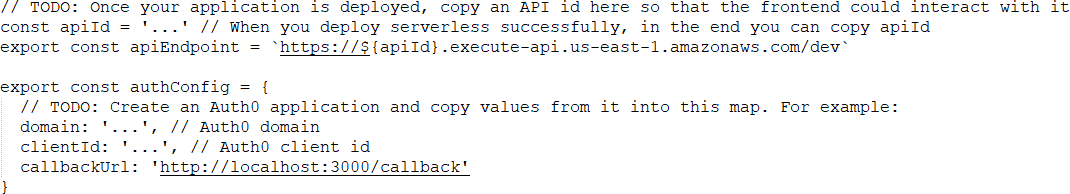
* + Config serverless credentials:



* Ở lần đầu tiên, create an application in serverless (chọn option create)
* Tiếp đến, deploy app và ghi nhớ endpoint url khi chạy lệnh:
  + serverless deploy –versbose

Configure Frontend

* + Update file config.ts



## Một số lỗi mình gặp trong quá trình làm project:

## Serverless account và Serverless.yaml

Khi bạn tạo serverless account, có 2 option là personal hoặc org, nếu bạn chọn personal thì phải xóa dòng “Org” (line 01) trong file serverless.yaml. Còn nếu tạo account type là org thì điền org name của bạn vào.

## Change package.json trong backend

Khi mình build backend thì phát hiện file package.json ban đầu họ đưa bị lỗi, sẽ không cài được 1 số package đã không tồn tại để mà cài nữa, việc cần làm là bạn load file package json mới ở đây về : https://knowledge.udacity.com/questions/912446 sau đó thì “npm install” như bình thường.

## Error when import aws-xray-sdk

Trong code starter có import sẵn XRAY module (file helper/attachmentUtils.ts và helper/todosAccess.ts) nhưng báo lỗi import. Nếu bạn gặp lỗi này thì có thể đổi thành như sau:

*import* \* *as* AWS *from* 'aws-sdk'

const AWSXRay = require('aws-xray-sdk')

*import* { createLogger } *from* '../utils/logger' const XAWS = AWSXRay.captureAWS(AWS)

Link tham khảo <https://knowledge.udacity.com/questions/70893>

## Change S3 bucket name trong serverless.yml

Trên AWS, S3 bucket name là global name nghĩa là tên không thể trùng nhau được. Default trong serverless.yml file, S3 bucket đã được setname.

environment:

ATTACHMENT\_S3\_BUCKET: serverless-c4-todo-images-${self:provider.stage}

Điều này dẫn tới việc nếu mọi người để nguyên giá trị của “ATTACHMENT\_S3\_BUCKET” thì lúc deploy sẽ báo lỗi ở S3 bucketname, có thể đã tồn tại do cũng có người dùng S3 bucketname này rồi.

Để fix lỗi này thì bạn chỉ cần thêm 1 random string trước S3 bucket name là xong, đảm bảo không bị trùng với ai khác nữa:

environment:

ATTACHMENT\_S3\_BUCKET:

${self:provider.stage}

asweq1d13aa-datpt45-serverless-c4-todo-images-

## Validator request

Có 2 cách để validate request tới API:

* + Dùng validator plugin trong serverless famework
  + Dùng schema validate

Mình khuyến khích dùng schema validate vì nó đơn giản và config nhanh. Schema sẽ dự vào json model để validate request.

Ví dụ về validate request trong function create todo:

CreateTodo:

handler: src/lambda/http/createTodo.handler events:

- http:

method: post path: todos cors: true authorizer: Auth request:

schemas:

application/json: ${file(models/create-todo-model.json)}

## Check empty todo name

Trong Project rubric không có nói tới việc phải check TodoName empty nhưng khi review thì reviewer luôn nhắc tới phần này và yêu cầu mình làm. Việc này khá đơn giản, dùng schema request validator để xử lý. Mở file models/create-todo-model.json và sửa như sau:

{

"$schema": "[http://json-schema.org/draft-04/schema#](http://json-schema.org/draft-04/schema)", "title": "group",

"type": "object", "properties": {

"name": {

"type": "string", "minLength": 1

},

"dueDate": {

"type": "string"

}

},

"required": [

"name", "dueDate"

],

"additionalProperties": false

}

## Enable CORS

Để enable CORS sẽ cần thực hiện 2 chỗ là trong response trả về của Lambda func và trên API Gateway thông qua serverless.yml config:

Với serverless.yml thì bạn thêm **cosr: true** như dưới:

CreateTodo:

handler: src/lambda/http/createTodo.handler events:

- http:

method: post

path: todos cors: true

Với response của lambda function thì thêm **Handler** như dưới

*export* const handler = middy(

async (*event*: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> => {

const todoId = *event*.pathParameters.todoId

*// TODO: Remove a TODO item by id*

}

)

handler

.use(httpErrorHandler())

.use(

cors({

credentials: true

})

)

## Thêm iamRoleStatements trong từng lambda function thông qua serverless.yml

Lưu ý: Việc thêm IAM role này phải đảm bảo permission tối thiểu cần để func có thể chạy không được thêm thừa, vì reviewer cũng sẽ yêu cầu bạn sửa lại nếu thừa:

Ví dụ về iamRoleStatement cho một lambda function

*# Provide property for setting up CORS, Authorizer, iamRoleStatements*

DeleteTodo:

handler: src/lambda/http/deleteTodo.handler events:

* http:

method: delete path: todos/{todoId} cors: true authorizer: Auth

iamRoleStatementsName: delete-todo-role-function iamRoleStatementsInherit: true iamRoleStatements:

* Effect: Allow Action:
  + dynamodb:Query
  + dynamodb:DeleteItem

Resource: arn:aws:dynamodb:${self:provider.region}:\*:table/${self:provider.environment.TODOS\_TABLE

}

* Effect: Allow Action:
  + s3:DeleteObject

Resource: 'arn:aws:s3:::${self:provider.environment.ATTACHMENT\_S3\_BUCKET}/\*'

* Effect: Allow Action:
  + xray:PutTraceSegments
  + xray:PutTelemetryRecords Resource:

- "\*"

Trong config cho Delete Todo function trên minh đang define permission cho 3 resource. Do function là function delete nên không cần tới quyền update hay create gì ở đây cả:

* + S3: Delete object
  + Dynamodb: Có quyền query và delete item
  + Xray: Put segment và record

Cuối cùng, bạn có thể tham khảo code của các bạn đã làm trước đó và check lại 1 lượt cái yêu cầu trong Project rubric trước khi submit.

# Project 05: CAPSTONE

Project 05 là là project cuối khóa, trong project này cho phép bạn lựa chọn 1 trong 2 project đã làm để phát triển thêm tính năng dựa trên các kiến thức bạn đã học từ đầu tới giờ.

Tham khảo project rubric: [Udacity Reviews](https://review.udacity.com/%23!/rubrics/2578/view)

* + Project 03: Bạn chọn project này nếu kiến thức về K8s của bạn tương đối và có kiến thức về mảng Operation vì với project 03 này, yêu cầu sẽ thiên về việc cấu hình, deploy ứng dụng trên K8S
  + Project 04: Chọn project này nếu bạn mạnh về coding Nodejs hơn.

Mình thấy tới 98% anh em đã đang học khóa này chọn Project 04 để làm Project 05 vì có thể là hầu hết là Developer và Project 04 vừa làm nóng hổi xong vẫn còn nắm trong lòng bàn tay =)).

Với mình thì mình chọn Project 03 để làm, 1 phần là thích thử thách chút, phần vì mình có kinh nghiệm với k8s hơn là Nodejs, nói thật là mới học code NodeJS khi học khóa này, từ Project tới Projecr 03 mình làm mất 2 ngày là xong vì nó cũng không nhiều code tới P4 thì tắc tịt vì Code hơi nhiều =)) và Job Title của mình cũng không hẳn là Developer :D.

Nếu các bạn chọn project 4, mình gợi ý các bạn có thể thay đổi project TodoList thành BookList,

PostList, CourseList, … và bạn chỉ cần chọn 1 function phía dưới để implement cho project:

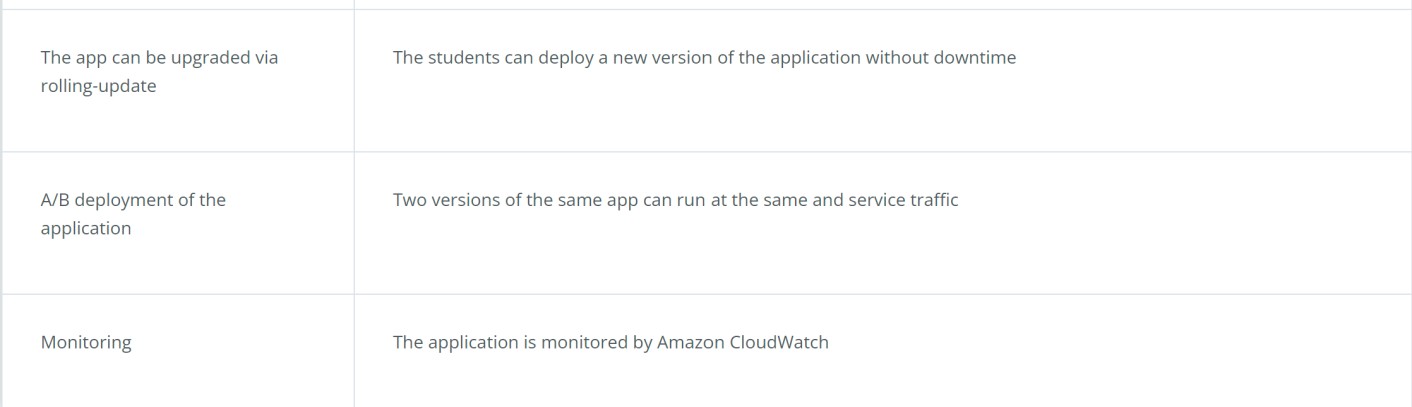
+ search function

+ Instead add new item from the homepage, you add new create page with all fields of your model (name, date, …), and edit all fields of your model, v.v... tuỳ các bạn xào nấu :D

Bạn cần tạo thêm function trong businessLogic, dataLayer, serverless.yml, lambda/http

Với Project 03, bạn cần Implement Blue/Green deployment và enable Cloudwatch metric, nó sẽ fit với

3. requirement này:



Mình sẽ guide từng phần:

## Enable Cloudwatch metrics

Tạo service account trên cluster

kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon- cloudwatch-container-insights/latest/k8s-deployment-manifest- templates/deployment-mode/daemonset/container-insights- monitoring/cwagent/cwagent-serviceaccount.yaml

Tải configmap sample

curl -O https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch- container-insights/latest/k8s-deployment-manifest-templates/deployment- mode/daemonset/container-insights-monitoring/cwagent/cwagent- configmap.yaml

Mở file và thay thế giá trị **{{cluster\_name}}** bằng tên EKS cluster (ekycbpo-prod) và sau đó áp dụng config

kubectl apply -f cwagent-configmap.yaml

Deploy Cloudwatch agent với DaemonSet

kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon- cloudwatch-container-insights/latest/k8s-deployment-manifest- templates/deployment-mode/daemonset/container-insights- monitoring/cwagent/cwagent-daemonset.yaml

Kiểm tra việc deploy

kubectl get pods -n amazon-cloudwatch

Kết quả như dưới là thành công (**running**)

NAME

READY STATUS RESTARTS AGE

cloudwatch-agent-dhhvc 1/1 Running 0

4m37s

cloudwatch-agent-dtrr5 1/1 Running 0 4m37s

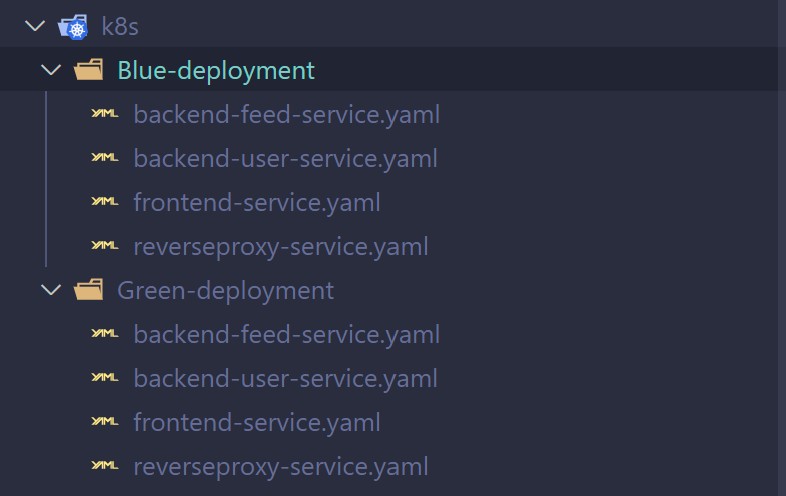
## Enable Blue/Green deployment

Blue/Green deployment là một thuật ngữ chỉ kỹ thuật duy trình cùng lúc 2 version của app và cho phép route request từ version cũ sang version mới mà không bị downtime đồng thời cũng có thể test app trên version mới với traffic thật.

Tham khảo: [Using AWS Load Balancer Controller for blue/green deployment, canary deployment](https://aws.amazon.com/blogs/containers/using-aws-load-balancer-controller-for-blue-green-deployment-canary-deployment-and-a-b-testing/) [and A/B testing | Containers (amazon.com)](https://aws.amazon.com/blogs/containers/using-aws-load-balancer-controller-for-blue-green-deployment-canary-deployment-and-a-b-testing/)

Bạn sẽ cần tạo 2 bộ config cho deployment và service cho các app và deploy lên K8s:

* Backend-feed
* Backend-user
* Frontend
* Reverseproxy



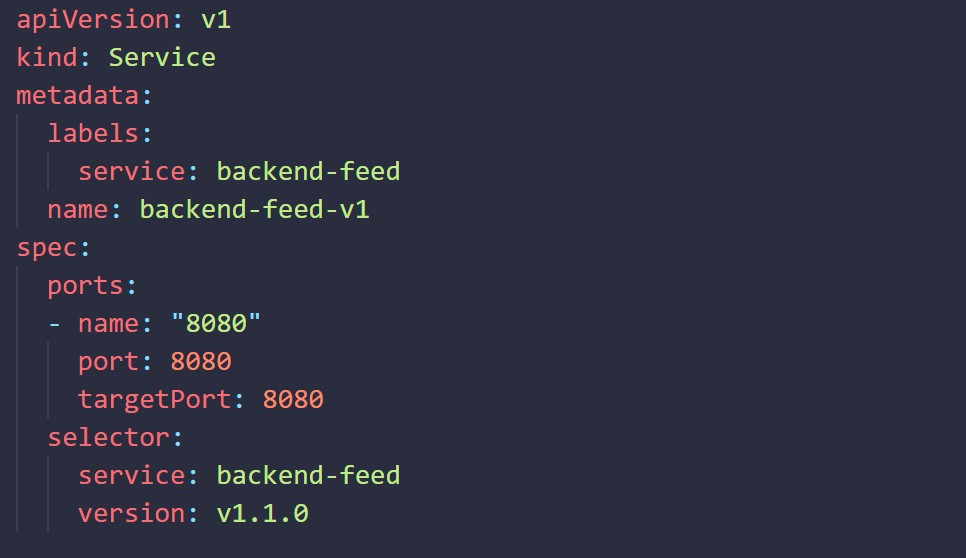
Về cơ bản thì configuration sẽ giữ nguyên như project 03, chỉ khác là bạn sẽ cần thêm selector trong metada và đổi tên service.

## Ví dụ cho blue deployment:

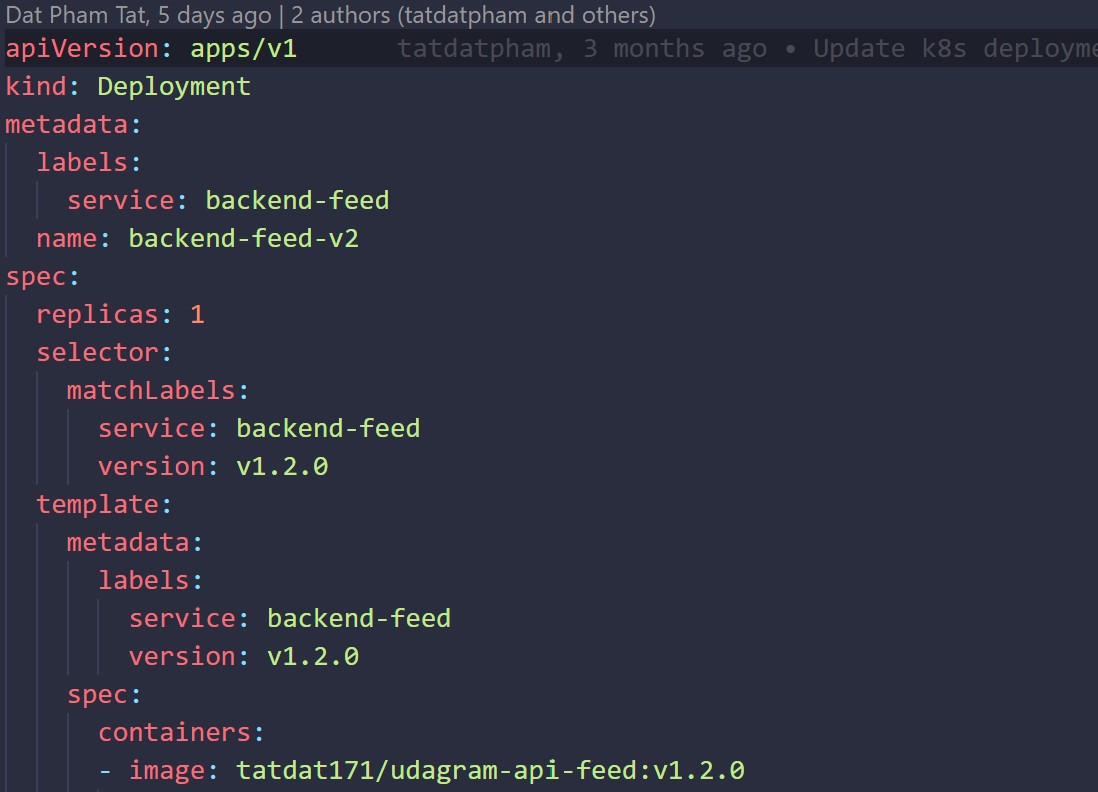


Trong config mình có thêm version vào lable để phân biệt giữa 2 phiên bản ứng dụng. Blue deployment mình đang để version v1.1.0

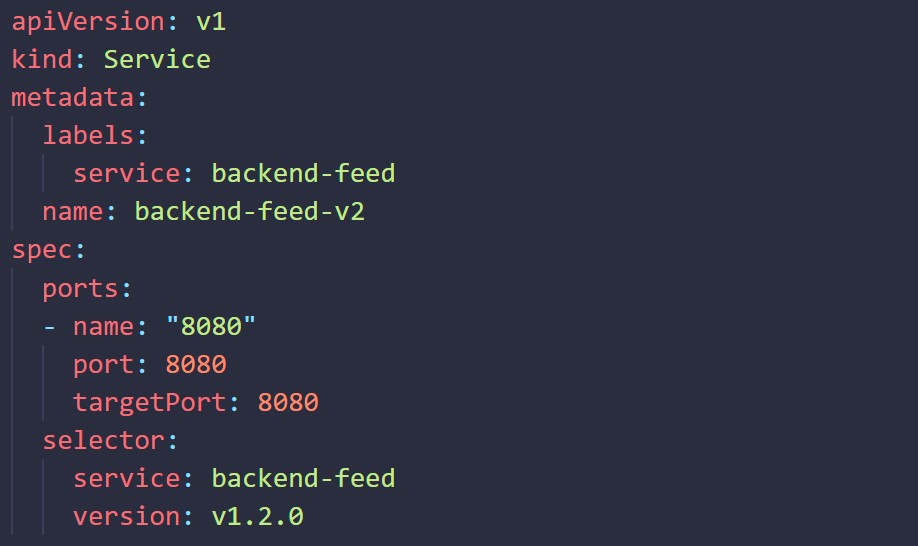
Service config cũng thêm version selector và tên service thêm “**-v1**” để phân biệt phiên bản:



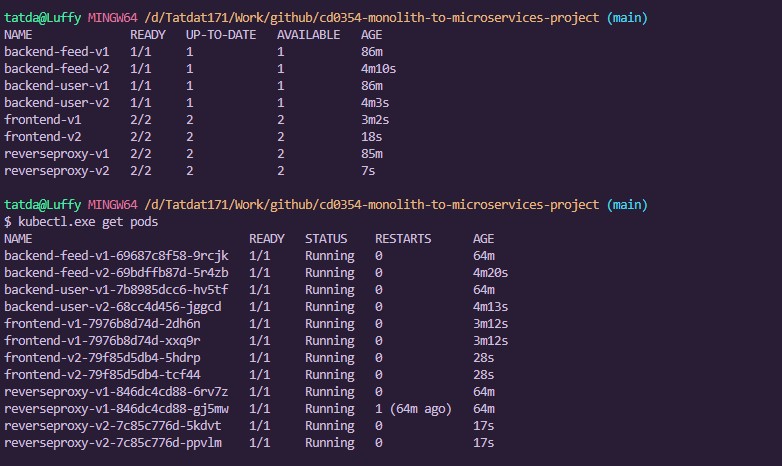
## Ví dụ cho Green deployment



Vesion đổi sang v1.2.0. Tương tự với service configuration



Sau khi chỉnh sửa xong 2 bộ config bạn deploy tất cả lên k8s sẽ được kết quả như dưới



## Tiếp theo, tạo ingress để apply Blue/Green deployent

Bạn cũng sẽ cần cài đặt AWS Loadbalancer controller trên K8s để khi bạn tạo Ingress trên K8s, ingress sẽ tự động tạo ALB trên AWS. Tham khảo: [Installing the AWS Load Balancer Controller](https://docs.aws.amazon.com/eks/latest/userguide/aws-load-balancer-controller.html) [add-on - Amazon EKS](https://docs.aws.amazon.com/eks/latest/userguide/aws-load-balancer-controller.html)

Ứng dụng có 2 public url gồm Frontend service và Reverse service (backend). Tạo ingress config như sau:

Trong đó, “alb.ingress.kubernetes.io/subnets” là public subnets ID, bạn có thể lấy ID trên AWS console, VPC service

apiVersion: networking.k8s.io/v1

kind: Ingress metadata:

name: frontend-ingress annotations:

kubernetes.io/ingress.class: alb alb.ingress.kubernetes.io/scheme: internet-facing

alb.ingress.kubernetes.io/subnets: subnet-07698675feef5f3f5,

subnet-

0de8d55ec877c7dca, subnet-0a37626814a280a75, subnet-00ce6c73d69b5b8f9

alb.ingress.kubernetes.io/target-type: 'ip' alb.ingress.kubernetes.io/actions.blue-green-frontend: *|*

{

"type":"forward", "forwardConfig":{

"targetGroups":[

{

"serviceName":"frontend-v1", "servicePort":"8100", "weight":100

},

{

"serviceName":"frontend-v2", "servicePort":"8100", "weight":0

}

]

}

}

labels:

service: frontend-ingress spec:

rules:

- http:

paths:

- path: / pathType: Prefix backend:

service:

name: blue-green-frontend port:

name: use-annotation

Và ingess service cho Reverseproxy service:

apiVersion: networking.k8s.io/v1

kind: Ingress metadata:

name: reverseproxy-ingress annotations:

kubernetes.io/ingress.class: alb alb.ingress.kubernetes.io/scheme: internet-facing

alb.ingress.kubernetes.io/subnets: subnet-07698675feef5f3f5,

subnet-

0de8d55ec877c7dca, subnet-0a37626814a280a75, subnet-00ce6c73d69b5b8f9

alb.ingress.kubernetes.io/target-type: 'ip' alb.ingress.kubernetes.io/actions.blue-green-reverseproxy: *|*

{

"type":"forward", "forwardConfig":{

"targetGroups":[

{

"serviceName":"reverseproxy-v1", "servicePort":"8080",

"weight":100

},

{

"serviceName":"reverseproxy-v2", "servicePort":"8080",

"weight":0

}

]

}

}

labels:

service: reverseproxy-ingress spec:

rules:

- http:

paths:

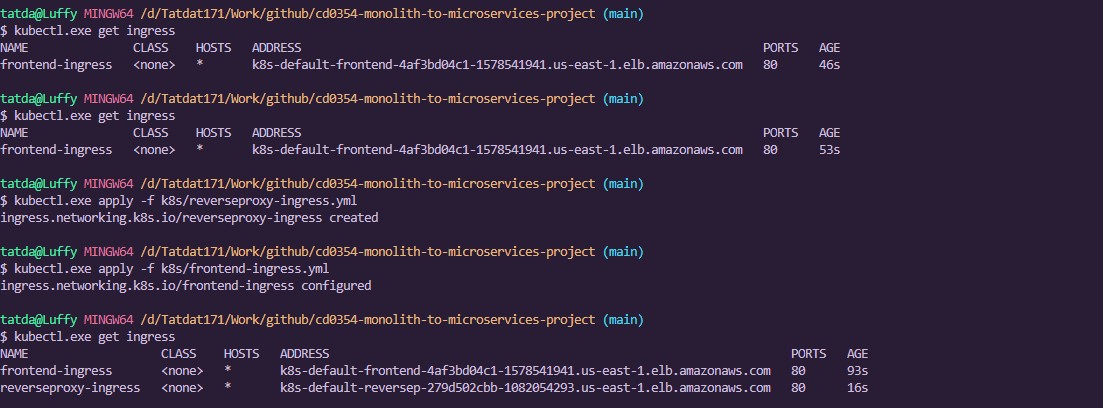
- path: / pathType: Prefix backend:

service:

name: blue-green-reverseproxy port:

name: use-annotation

Sau đó deploy configuration với lênh kubectl apply -f <ingresss\_config\_file> Kết quả:



Bạn test truy cập vào 2 endpoint kia, truy cập được là đã xong, chụp ảnh và submit thôi :D

# Support Channel

* **Sử dụng kênh support của Udacity**: Udacity GPT/ Knowledge Portal Udacity/ Mentor Help cho các vấn đề về nội dung

Account Help cho các vấn đề kĩ thuật: https://udacityenterprise.zendesk.com/hc/en- us/requests/new

* Tham gia group nội bộ FSOFT **Cloud Developer AWS** trên MS Team của FSO
* **Contact point**: DanPNL (HN+OB)/ ThaoNT39 (ĐN+QN+HUE)/ VyTT4 (HCM+CT)