

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 lập trình, viết test case, store 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 biệt 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 trọng 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, để 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:

```
> npm run clean && tsc && copy package.json www/package.json && mkdir www/tmp/ && cd www && zip -r Archive.zip . && cd ..

> udacity-c2-image-filter@1.0.0 clean C:\Users\... \cloud-developer-master\course-02\project\image-filter-starter-code
> rimraf www/ || true

The syntax of the command is incorrect.
npm ERR! code ELIFECYCLE
npm ERR! errno 1
npm ERR! udacity-c2-image-filter@1.0.0 build: `npm run clean && tsc && copy package.json www/package.json && mkdir www/tmp/ && cd www && zip -r Archive.zip .`
npm ERR! Exit status 1
npm ERR!
npm ERR! Failed at the udacity-c2-image-filter@1.0.0 build script.
npm ERR! This is probably not a problem with npm. There is likely additional logging output above.

npm ERR! A complete log of this run can be found in:
```

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 \(udacity.com\)](#) – 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 ..
```

```
tatda@Luffy MINGW64 ~/Documents/Project02
$ npm run build

> udacity-c2-image-filter@1.0.0 build
> npm run clean && tsc && copy package.json www\ && cd www && tar -cvzf Archive.zip * && cd ..

> udacity-c2-image-filter@1.0.0 clean
> rimraf www/ || true

    1 file(s) copied.
package.json
server.js
server.js.map
util/
util/util.js
util/util.js.map

tatda@Luffy MINGW64 ~/Documents/Project02
$ dir www/
Archive.zip package.json server.js server.js.map util
```

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

```
WSEB-FOPQC78MK53T/8d20f8cbeSeda176/c45f85ef8e40a0c0
ERROR Instance deployment failed. For details, see 'eb-engine.log'.
ERROR Instance deployment: Your source bundle has issues that caused the deployment to fail. For details, see 'eb-engine.log'.
ERROR [Instance: i-02a14ac0813f7b564] Command failed on instance. Return code: 1 Output: Engine entered an error..
INFO Command execution completed on all instances. Summary: [Successful: 0, Failed: 1].
ERROR Create environment operation is complete, but with errors. For more information, see troubleshooting.
```

- 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

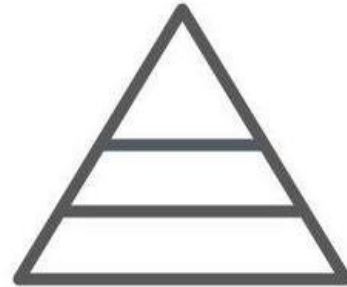
Project 03: Monolith to Microservices at Scale

Học phần này tiếp nối Project 2 tuy nhiên sẽ có sự 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.



Microservices

Independently-deployed services that communicate with one another through networks.

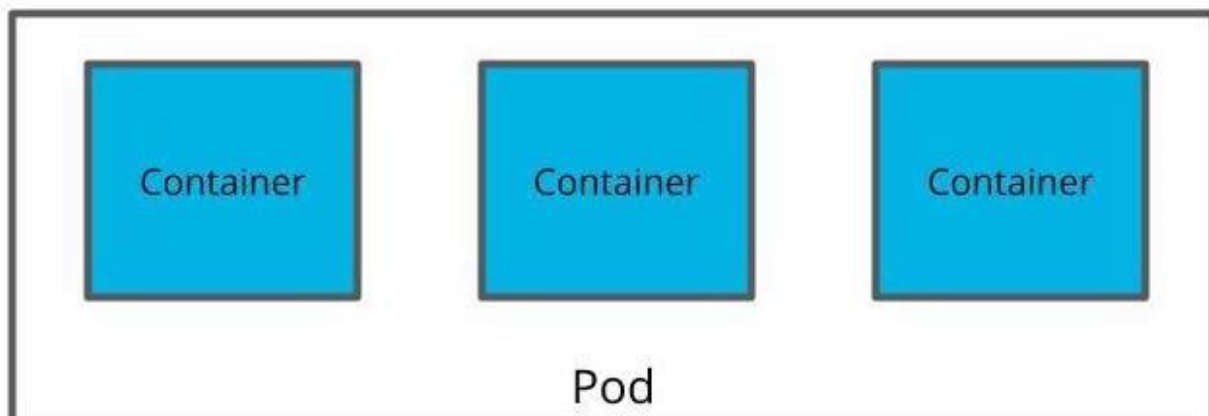


Monoliths

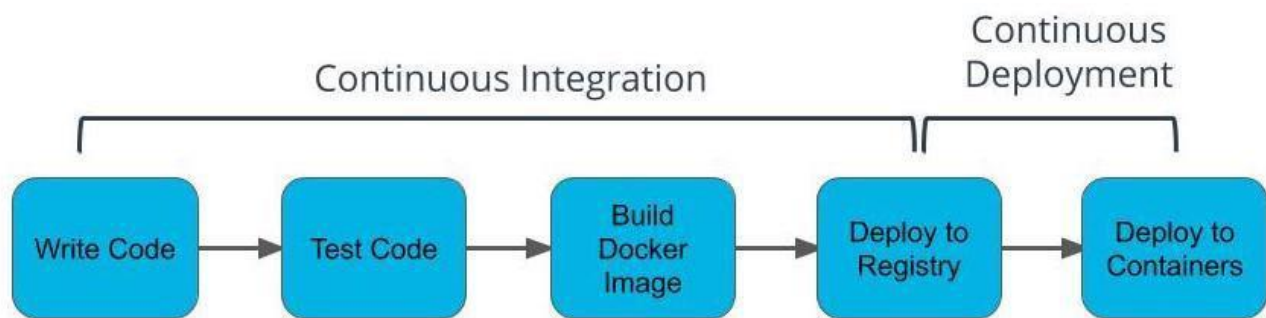
Self-contained applications that are deployed as one unit.

Microservice vs. Monolith Architecture

Bạn sẽ được học về container, Docker và Kubernetes, 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

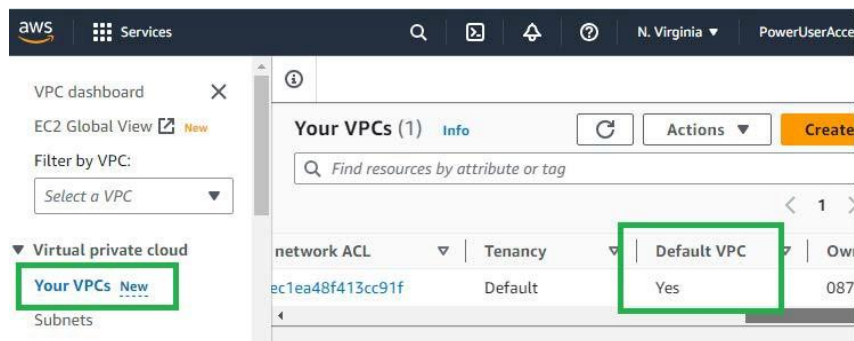


Prerequisites: để làm Project 3, các bạn chuẩn bị:

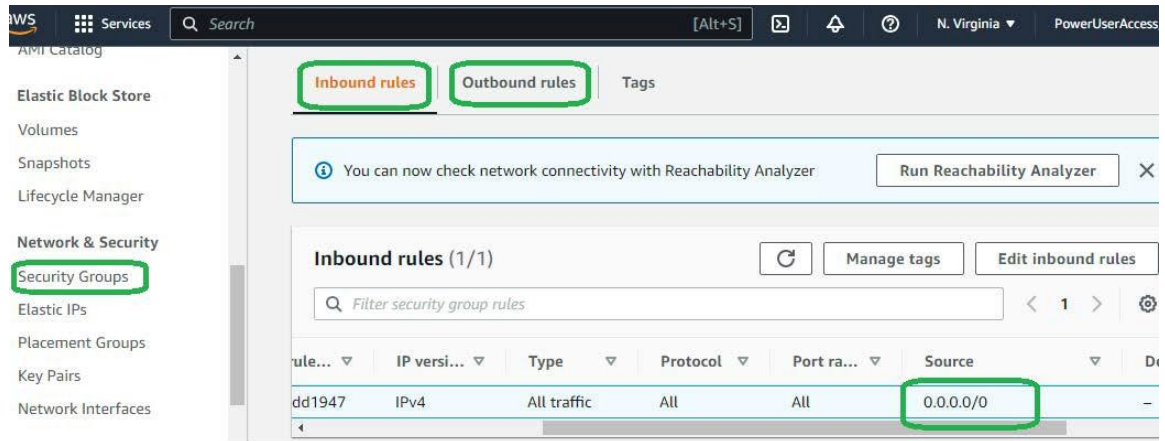
- Git
- PostgreSQL 14.8: <https://www.enterprisedb.com/downloads/postgres-postgresql-downloads> (lưu ý chúng ta chỉ cần cài Command line tools)
- Download source code
- Docker (trong khoá học hướng dẫn sử dụng docker để chạy ở local, tức ko nhất thiết phải cài docker, mình khuyến khích các bạn cài để 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.

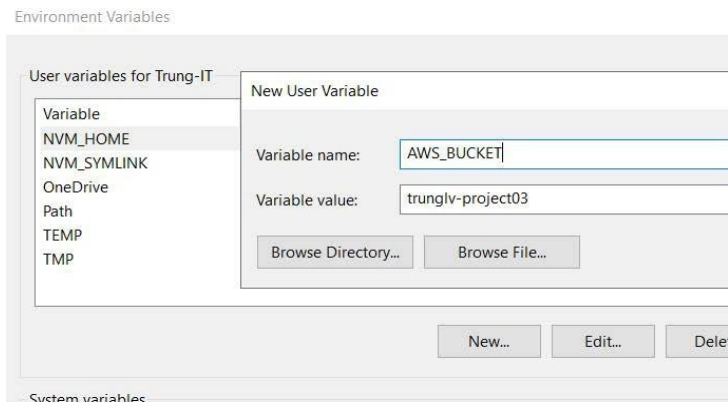
- Create an S3 bucket:
 - Set Bucket policy (*)
 - Set CORS configuration (*)
- Create PostgreSQL database, các bạn thực hiện theo các steps:
 - Trước tiên cần có default VPC (nếu chưa có các bạn phải tạo)



- 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 (*)
- Tùy theo hệ điều hành từng máy mà cách setup môi trường khác nhau
 - Ví dụ trên Window:

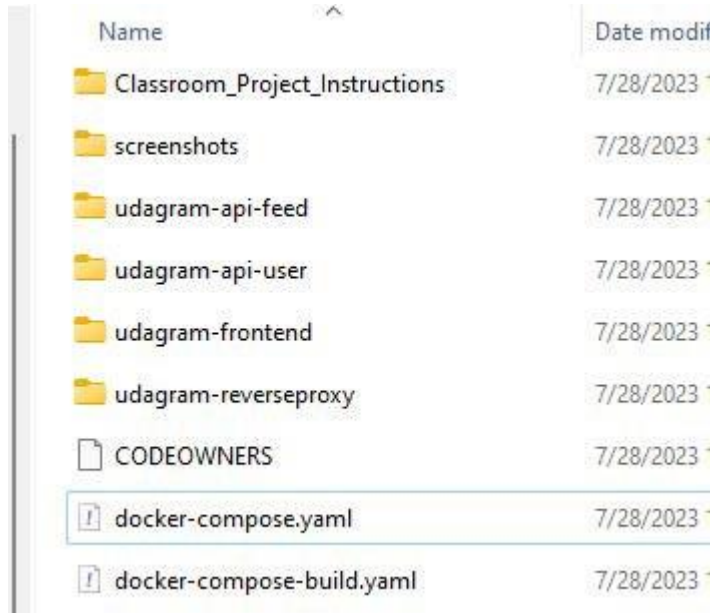


Part 1: Monolithic application

- Backend (*)
- Frontend (*)
 - Lưu ý ở step "ionic build" có thể sẽ báo lỗi 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:



| Name | Date modified |
|--------------------------------|---------------|
| Classroom_Project_Instructions | 7/28/2023 |
| screenshots | 7/28/2023 |
| udagram-api-feed | 7/28/2023 |
| udagram-api-user | 7/28/2023 |
| udagram-frontend | 7/28/2023 |
| udagram-reverseproxy | 7/28/2023 |
| CODEOWNERS | 7/28/2023 |
| docker-compose.yaml | 7/28/2023 |
| docker-compose-build.yaml | 7/28/2023 |

- 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 follow 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:

```

! .travis.yml •
! .travis.yml
>
6  services:
7    - docker
8
9  script:
10
11    # Build
12    - echo '_____beginning build'
13    - docker build -t udagram-api-feed:v3 ./udagram-api-feed
14    - docker build -t udagram-api-user:v3 ./udagram-api-user
15    - docker build -t udagram-frontend:v3 ./udagram-frontend
16    - docker build -t reverseproxy:v3 ./udagram-reverseproxy
17
18    # Tagging
19    - echo '_____beginning tag'
20    - docker tag udagram-api-feed:v3 yourDockerhub/udagram-api-feed:v3
21    - docker tag udagram-api-user:v3 yourDockerhub/udagram-api-user:v3
22    - docker tag udagram-frontend:v3 yourDockerhub/udagram-frontend:v3
23    - docker tag reverseproxy:v3 yourDockerhub/reverseproxy:v3
24
25    - echo 'DOCKER IMAGE LS ____'
26    - docker image ls
27  after_success:
28    # Assuming DOCKER_PASSWORD and DOCKER_USERNAME are set in the Travis
29    # Login Docker
30    - echo '_____LOGIN_____'
31    #- docker login -u=xxxx -p=xxxxx
32    - docker login -u="$DOCKER_USERNAME" -p="$DOCKER_PASSWORD"
33    #- echo 'username, password' $DOCKER_USERNAME $DOCKER_PASSWORD
34    # Pushing
35    - echo '_____beginning push image'
36    - docker push yourDockerhub/udagram-api-feed:v3
37    - docker push yourDockerhub/udagram-api-user:v3
38    - docker push yourDockerhub/udagram-frontend:v3
39    - docker push yourDockerhub/reverseproxy:v3
40    You, now • Uncommitted changes

```

- Circle CI:

```

! config.yml •
! config.yml
You, 1 minute ago [3 authors (TrungLeIT and others)]
1
2  version: 2.1
3
4  workflows:
5    build:
6      jobs:
7        - build
8  jobs:
9    build:
10     working_directory: ~/circleci-starter
11     docker:
12       - image: cimg/base:2022.09
13       - image: cimg/node:14.15.0 # Primary execution image
14     steps:
15       - checkout
16       - setup_remote_docker:
17         version: 20.10.14
18       - run:
19         name: build
20         command: |
21           # Build
22           echo '_____beginning build'
23           docker build -t udagram-api-feed:v3 ./udagram-api-feed
24           docker build -t udagram-api-user:v3 ./udagram-api-user
25           docker build -t udagram-frontend:v3 ./udagram-frontend
26           docker build -t reverseproxy:v3 ./udagram-reverseproxy
27
28           # Tagging
29           echo '_____beginning tag'
30           docker tag udagram-api-feed:v3 yourDockerhub/udagram-api-feed:v3
31           docker tag udagram-api-user:v3 yourDockerhub/udagram-api-user:v3
32           docker tag udagram-frontend:v3 yourDockerhub/udagram-frontend:v3
33           docker tag reverseproxy:v3 yourDockerhub/reverseproxy:v3
34
35           echo 'DOCKER IMAGE LS ____'
36           docker image ls
37
38       - run:
39         name: after_success
40         command: |
41           # Assuming DOCKER_PASSWORD and DOCKER_USERNAME are set in the Travis repository settings
42           # Login Docker
43           echo '_____LOGIN_____'
44           #- docker login -u=yourDockerhub -p=xxxxxx
45           docker login -u="$DOCKER_USERNAME" -p="$DOCKER_PASSWORD"
46           #- echo 'username, password' $DOCKER_USERNAME $DOCKER_PASSWORD
47           # Pushing
48           echo '_____beginning push image'
49           docker push yourDockerhub/udagram-api-feed:v3
50           docker push yourDockerhub/udagram-api-user:v3
51           docker push yourDockerhub/udagram-frontend:v3
52           docker push yourDockerhub/reverseproxy:v3

```

- Và kết quả CI/CD thành công

The screenshot shows the Travis CI web interface. At the top, there's a navigation bar with 'Travis CI' logo and links to Dashboard, Changelog, Documentation, and Help. Below this is a search bar and a section for 'My Repositories' showing 'GitAccount/Project-03' with a duration of 5 min 45 sec and finished 2 months ago. The main section is titled 'YourRepo / Project-03' and shows the 'Build History' tab. It lists two successful builds on the 'develop' branch, both triggered by '[trungle] update screenshot metric hpa' and '[trungle] update services screenshot'. Both builds passed, with commit hashes 492d78e and e3d3e8c respectively.

- Docker hub lúc này đã có image tương ứng:

The screenshot shows the Docker Hub repository page for 'sample / reverseproxy'. The page has a blue header with the Docker Hub logo and a search bar. Below the header, there are tabs for 'sample', 'Repositories', 'reverseproxy', and 'General'. The 'General' tab is selected, showing the repository name 'sample / reverseproxy' and a description 'This repository does not have a description'. It also shows 'Last pushed: a month ago'. Below this, there's a 'Tags' section stating 'This repository contains 3 tag(s)'. A table lists the tags:

| Tag | OS | Type | Pulled | Pushed |
|-----|-------|-------|-------------|-------------|
| v3 | linux | Image | a month ago | a month ago |

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 kube 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

```
! aws-secret.yaml
! aws-secret.yaml
...
1  apiVersion: v1
2  kind: Secret
3  metadata:
4    name: aws-secret
5  type: Opaque
6  data:
7    credentials: W2RlZmF1bHRdCmF3c19hY2N1c3Nfa2V5X
8
```

```
! env-configmap.yaml
! env-configmap.yaml
...
1  apiVersion: v1
2  kind: ConfigMap
3  metadata:
4    name: config-secret
5  data:
6    AWS_BUCKET: yourS3-bucketName
7    AWS_PROFILE: default
8    AWS_REGION: us-east-1
9    JWT_SECRET: testing
10   POSTGRES_DB: postgres
11   POSTGRES_USERNAME: postgres
12   POSTGRES_PASSWORD: yourPass
13   POSTGRES_HOST: your DB
14   URL: http://localhost:8100
15
```

- create env-secret.yaml
- create nginx.conf (tham khảo udiagram-reverseproxy)

```
! env-secret.yaml
! env-secret.yaml
...
1  apiVersion: v1
2  kind: Secret
3  metadata:
4    name: env-secret
5  type: Opaque
6  data:
7    POSTGRES_USERNAME: postgres
8    POSTGRES_PASSWORD: bGV2YW50cnVuZW==
9
```

- feed-deployment.yaml
- feed-service.yaml

```
! feed-service.yaml
! feed-service.yaml
...
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: api-feed
5  labels:
6    app: api-feed
7  spec:
8    ports:
9      - port: 8080
10     protocol: TCP
11     name: http
12  selector:
13    app: api-feed
14
15
16
```

```
! feed-deployment.yaml
! feed-deployment.yaml
...
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: api-feed
5  labels:
6    app: api-feed
7  spec:
8    replicas: 2
9    selector:
10     matchLabels:
11       app: api-feed
12  template:
13    metadata:
14     labels:
15       app: api-feed
16    spec:
17     containers:
18       - name: api-feed
19         image: yourDockerhub/udagram-api-feed:v3
20         imagePullPolicy: Always
21         resources:
22           requests:
23             cpu: 250m
24           limits:
25             cpu: 500m
26         envFrom:
27           - secretRef:
28               name: env-secret
29           - secretRef:
30               name: aws-secret
31           - configMapRef:
32               name: config-secret
33         ports:
34           - containerPort: 8080
35         restartPolicy: Always
36
37
```

- 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 Unknown như này, bạn install metric server vào k8s để có thể show CPU metrics

```

Name: backend-user
Namespace: default
Labels: <none>
Annotations: <none>
CreationTimestamp: Thu, 01 Dec 2022 18:38:04 +0700
Reference: Deployment/backend-user
Metrics:
  resource cpu on pods (as a percentage of request): <unknown> 70%
Min replicas: 2
Max replicas: 3
Deployment pods: 3 current / 3 desired
Conditions:
  Type           Status  Reason
  ----           -
  AbleToScale    True    SucceededGetScale
  ScalingActive  False   FailedGetResourceMetric
t metrics for resource cpu: unable to fetch metrics from resource metrics API: the server could not find the requested resource (get pods.metr

```

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 | <code>kubectl get deployment</code> | Get all deployment |
| 2 | <code>kubectl get pods</code> | Get all pods |
| 3 | <code>kubectl get svc</code> | Get all service |
| 4 | <code>kubectl get hpa</code> | Get all HPA |
| 5 | <code>kubectl describe deployment <deployment_name></code> | Xem chi tiết về 1 deployment |
| 6 | <code>kubectl describe pod <pod_name></code> | Xem chi tiết về 1 pod |
| 7 | <code>kubectl exec --stdin --tty <pod_name> -- /bin/bash</code> | 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.

1. Chuẩn bị account, env

1.1. Auth0 account: <https://manage.auth0.com/>

1.2. Github account

1.3. NodeJS version 14.15.0 (quá trình mình làm thì version này là ổn, các version khác có thể dẫn đến lỗi)

1.4. Serverless

1.4.1. Create account serverless: <https://dashboard.serverless.com/>

1.4.2. Install Serverless Framework's CLI (suggest version 3)

- `npm install -g serverless@3`
- `serverless --version` (check serverless version)

2. Backend

Link tham khảo Serverless Application/Backend: [guide](#)

2.1. File serverless.yml

- Các bạn điền các thông tin như framework version, profile, runtime (same nodejs version)

```
1  # TODO: Change the name of the org
2  org: sample
3  app: serverless-demo
4  service: serverless-demo
5  frameworkVersion: '3'
```

- Cập nhật environment

```
environment:
  TODOS_TABLE: Todos-${self:provider.stage}
  TODOS_CREATED_AT_INDEX: CreatedAtIndex
  ATTACHMENT_S3_BUCKET: serverless-c4-todo-images-${self:provider.stage}
  SIGNED_URL_EXPIRATION: 300
```

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:

```
<FUNCTION-NAME>:
  handler: <YOUR-HANDLER>
  events:
    - http:
        method: get
        path: todos
        cors: true
        authorizer: Auth
  iamRoleStatementsName: <yourNameRole>
  iamRoleStatementsInherit: true
  iamRoleStatements:
    - Effect: Allow
      Action:
        - dynamodb:GetItem
        - dynamodb:UpdateItem
        - dynamodb:PutItem
      Resource: arn:aws:dynamodb:<TABLE-NAME>
```

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

```

logs:
  # Enable API Gateway logs
  restApi: true

iam:
  role:
    statements:
      - Effect: Allow
        Action:
          - xray:PutTelemetryRecords
          - xray:PutTraceSegments
        Resource: "*"

```

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

```

resources:
  Resources:
    # TODO: Fill the properties
    # Feel free to change the names as you like.
    GatewayResponseDefault4XX:
      Type: AWS::ApiGateway::GatewayResponse
      Properties:
        ResponseParameters:
          gatewayresponse.header.Access-Control-Allow-Origin: "*"
          gatewayresponse.header.Access-Control-Allow-Headers: "'Content-Type'"
          gatewayresponse.header.Access-Control-Allow-Methods: "'GET,OPTIONS'"
        ResponseType: DEFAULT_4XX
        RestApiId:
          Ref: ApiGatewayRestApi
    # TODO: Create a DynamoDB table with local secondary index and a composite key
    TodosTable:
      Type: AWS::DynamoDB::Table
      Properties:
        AttributeDefinitions:
          - AttributeName: userId
            AttributeType: S
          - AttributeName: todoId
            AttributeType: S
          - AttributeName: createdAt
            AttributeType: S
        KeySchema:
          - AttributeName: userId
            KeyType: HASH
          - AttributeName: todoId
            KeyType: RANGE
        BillingMode: PAY_PER_REQUEST
        TableName: ${self:provider.environment.TODOS_TABLE}
        LocalSecondaryIndexes:
          - IndexName: ${self:provider.environment.TODOS_CREATED_AT_INDEX}
            KeySchema:
              - AttributeName: userId
                KeyType: HASH
              - AttributeName: createdAt
                KeyType: RANGE
            Projection:
              ProjectionType: ALL

```

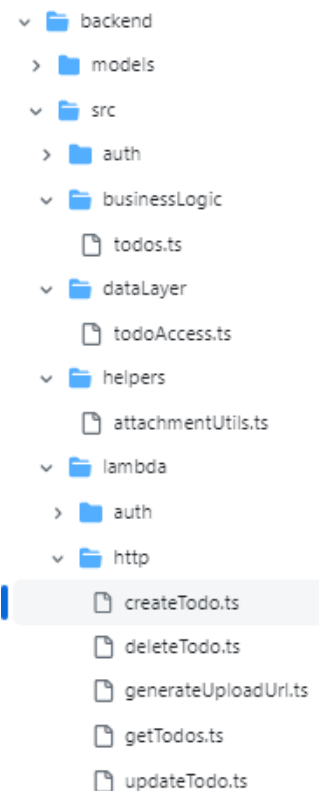

2.2. File attachmentUtils (cd src/helpers)

2.3. Implement Todos, TodoAccess

2.4. cd Lambda/http

- CRUD Todo
- GenerateUploadURL
- The file name and method in lambda/http must be same in serverless.yml:

```
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)}
```



2.5. Thêm Validate min length

Mình gợi ý các bạn validate by schema request validator:

```
{
  "$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
}
```

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

- Dùng **CORS** header

```
export const handler: APIGatewayProxyHandler = async (
  event: APIGatewayProxyEvent
): Promise<APIGatewayProxyResult> => {
  createLogger('Processing event: ' + event)
  const result = await getData()

  return {
    statusCode: 200,
    headers: {
      'Access-Control-Allow-Origin': '*',
      'Access-Control-Allow-Credentials': true
    },
    body: JSON.stringify({ items: result })
  }
}
```

- Dùng **middy** middleware

```
import * as middy from 'middy'
import { cors, httpErrorHandler } from 'middy/middlewares'
export const handler = middy(
  async (event: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> => {
    // Write your logic here
    .
    .
    .
    return undefined
  }
)
handler
  .use(httpErrorHandler())
  .use(
    cors({
      origin: "*",
      credentials: true
    })
  )
)
```

Và thêm CORS trong file serverless.yml

```
<FUNCTION-NAME>:
  handler: <YOUR-HANDLER>
  events:
    - http:
        method: get
        path: todos
        cors: true
        authorizer: Auth
  iamRoleStatementsInherit: true
  iamRoleStatements:
    - Effect: Allow
      Action:
        - dynamodb:GetItem
        - dynamodb:UpdateItem
        - dynamodb:PutItem
      Resource: arn:aws:dynamodb:<TABLE-NAME>
```

3. Client







3.1. Login vào Auth0

3.2. Tạo một "Single Page Web Applications" type Auth0 application

3.3. Đi đến App settings thiết lập Allowed CallBack URLs, Allowed Web Origins for CORS option

4. Run and Deploy

- Tham khảo Serverless Application/Deploy: [guide](#)
- Trên trang AWS, tạo IAM User với 2 Role như dưới

| <input type="checkbox"/> | <input type="checkbox"/> | Policy name  |  |
|-------------------------------------|--------------------------|---|---|
| <input type="checkbox"/> | <input type="checkbox"/> |  AccessAnalyzerServiceRolePolicy | |
| <input checked="" type="checkbox"/> | <input type="checkbox"/> |  AdministratorAccess | |
| <input type="checkbox"/> | <input type="checkbox"/> |  AdministratorAccess-Amplify | |
| <input checked="" type="checkbox"/> | <input type="checkbox"/> |  AdministratorAccess-AWSElasticBeanstalk | |

- 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:
 - o Login serverless:

```
serverless login
```

- o Config serverless credentials:

```
sls config credentials -o --provider aws --key yourKey -secret yourSecretKey --profile serverless (IAM user)
```

- Ở lần đầu tiên, create an application in serverless (chọn option create)
- Tiếp đến, deploy app và ghi nhớ endpoint url khi chạy lệnh:
 - o serverless deploy --verbose

Configure Frontend

- o Update file config.ts

```
// TODO: Once your application is deployed, copy an API id here so that the frontend could interact with it
const apiId = '...' // When you deploy serverless successfully, in the end you can copy apiId
export const apiEndpoint = `https://${apiId}.execute-api.us-east-1.amazonaws.com/dev`

export const authConfig = {
  // TODO: Create an Auth0 application and copy values from it into this map. For example:
  domain: '...', // Auth0 domain
  clientId: '...', // Auth0 client id
  callbackUrl: 'http://localhost:3000/callback'
}
```

Một số lỗi mình gặp trong quá trình làm project:

1. 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.

2. 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.

3. 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>

4. 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: asweq1d13aa-datpt45-serverless-c4-todo-images-
${self:provider.stage}
```

5. Validator request

Có 2 cách để validate request tới API:

- Dùng validator plugin trong serverless framework
- Dùng schema validate

Mình khuyến khích dùng schema validate vì nó đơn giản và config nhanh. Schema sẽ dựa 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)}
```

6. 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#",
  "title": "group",
  "type": "object",
  "properties": {
    "name": {
      "type": "string",
      "minLength": 1
    },
    "dueDate": {
      "type": "string"
    }
  },
  "required": [
    "name",
    "dueDate"
  ],
  "additionalProperties": false
}
```

7. 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 **cors: 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
    })
  )
)

```

8. 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 mình 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](#)

- Project 03: Bạn chọn project này nếu kiến thức về K8s của bạn tương đối và có kiến thức về mạng 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 Project 03 mình làm mất 2 ngày là xong vì nó cũng không nhiều code tới P4 thì tắc tị 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... tùy các bạn xào nẫu :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:

| | |
|--|---|
| The app can be upgraded via rolling-update | The students can deploy a new version of the application without downtime |
| A/B deployment of the application | Two versions of the same app can run at the same and service traffic |
| Monitoring | The application is monitored by Amazon CloudWatch |

Mình sẽ guide từng phần:

1. 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 |

2. 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 and A/B testing | Containers \(amazon.com\)](#)

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 metadata và đổi tên service.

Ví dụ cho blue deployment:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    service: backend-feed
  name: backend-feed-v1
spec:
  replicas: 1
  selector:
    matchLabels:
      service: backend-feed
      version: v1.1.0
  template:
    metadata:
      labels:
        service: backend-feed
        version: v1.1.0
    spec:
      containers:
        - image: tatdat171/udagram-api-feed:v1.1.0

```

Trong config mình có thêm version vào label 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:

```

apiVersion: v1
kind: Service
metadata:
  labels:
    service: backend-feed
  name: backend-feed-v1
spec:
  ports:
    - name: "8080"
      port: 8080
      targetPort: 8080
  selector:
    service: backend-feed
    version: v1.1.0

```

Ví dụ cho Green deployment

Dat Pham Tat, 5 days ago | 2 authors (tatdatpham and others)
apiVersion: apps/v1 tatdatpham, 3 months ago • Update k8s deployment
kind: Deployment
metadata:
 labels:
 service: backend-feed
 name: backend-feed-v2
spec:
 replicas: 1
 selector:
 matchLabels:
 service: backend-feed
 version: v1.2.0
 template:
 metadata:
 labels:
 service: backend-feed
 version: v1.2.0
 spec:
 containers:
 - image: tatdat171/udagram-api-feed:v1.2.0

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

```
apiVersion: v1
kind: Service
metadata:
  labels:
    service: backend-feed
  name: backend-feed-v2
spec:
  ports:
    - name: "8080"
      port: 8080
      targetPort: 8080
  selector:
    service: backend-feed
    version: v1.2.0
```

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

```
tatda@Luffy MINGW64 /d/Tatdat171/Work/github/cd0354-monolith-to-microservices-project (main)
NAME                READY    UP-TO-DATE    AVAILABLE    AGE
backend-feed-v1     1/1      1              1             86m
backend-feed-v2     1/1      1              1            4m10s
backend-user-v1      1/1      1              1             86m
backend-user-v2      1/1      1              1            4m3s
frontend-v1          2/2      2              2             3m2s
frontend-v2          2/2      2              2             18s
reverseproxy-v1      2/2      2              2             85m
reverseproxy-v2      2/2      2              2             7s

tatda@Luffy MINGW64 /d/Tatdat171/Work/github/cd0354-monolith-to-microservices-project (main)
$ kubectl.exe get pods
NAME                READY    STATUS    RESTARTS    AGE
backend-feed-v1-69687c8f58-9rcjk 1/1      Running   0            64m
backend-feed-v2-69bdfb87d-5r4zb 1/1      Running   0            4m20s
backend-user-v1-7b8985dcc6-hv5tf 1/1      Running   0            64m
backend-user-v2-68cc4d456-jggcd 1/1      Running   0            4m13s
frontend-v1-7976b8d74d-2dh6n 1/1      Running   0            3m12s
frontend-v1-7976b8d74d-xxq9r 1/1      Running   0            3m12s
frontend-v2-79f85d5db4-5hdrp 1/1      Running   0            28s
frontend-v2-79f85d5db4-tcf44 1/1      Running   0            28s
reverseproxy-v1-846dc4cd88-6rv7z 1/1      Running   0            64m
reverseproxy-v1-846dc4cd88-gj5mw 1/1      Running   1 (64m ago) 64m
reverseproxy-v2-7c85c776d-5kdvt 1/1      Running   0            17s
reverseproxy-v2-7c85c776d-ppv1m 1/1      Running   0            17s
```

Tiếp theo, tạo ingress để apply Blue/Green deployment

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 add-on - Amazon EKS](#)

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 ingress 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 lệnh `kubectl apply -f <ingresss_config_file>`

Kết quả:

```

tatda@Luffy MINGW64 /d/Tatdat171/Work/github/cd0354-monolith-to-microservices-project (main)
$ kubectl.exe get ingress
NAME                CLASS    HOSTS                ADDRESS                                                                 PORTS    AGE
frontend-ingress    <none>   *                   k8s-default-frontend-4af3bd04c1-1578541941.us-east-1.elb.amazonaws.com  80      46s

tatda@Luffy MINGW64 /d/Tatdat171/Work/github/cd0354-monolith-to-microservices-project (main)
$ kubectl.exe get ingress
NAME                CLASS    HOSTS                ADDRESS                                                                 PORTS    AGE
frontend-ingress    <none>   *                   k8s-default-frontend-4af3bd04c1-1578541941.us-east-1.elb.amazonaws.com  80      53s

tatda@Luffy MINGW64 /d/Tatdat171/Work/github/cd0354-monolith-to-microservices-project (main)
$ kubectl.exe apply -f k8s/reverseproxy-ingress.yml
ingress.networking.k8s.io/reverseproxy-ingress created

tatda@Luffy MINGW64 /d/Tatdat171/Work/github/cd0354-monolith-to-microservices-project (main)
$ kubectl.exe apply -f k8s/frontend-ingress.yml
ingress.networking.k8s.io/frontend-ingress configured

tatda@Luffy MINGW64 /d/Tatdat171/Work/github/cd0354-monolith-to-microservices-project (main)
$ kubectl.exe get ingress
NAME                CLASS    HOSTS                ADDRESS                                                                 PORTS    AGE
frontend-ingress    <none>   *                   k8s-default-frontend-4af3bd04c1-1578541941.us-east-1.elb.amazonaws.com  80      93s
reverseproxy-ingress <none>   *                   k8s-default-reversep-279d502cbb-1082054293.us-east-1.elb.amazonaws.com  80      16s

```

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 FSOF **Cloud Developer AWS** trên MS Team của FSO
- **Contact point:** DanPNL (HN+OB)/ ThaoNT39 (ĐN+QN+HUE)/ VyTT4 (HCM+CT)