HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA ĐÀO TẠO SAU ĐẠI HỌC

BÁO CÁO THỰC TẬP TỐT NGHIỆP CAO HỌC

Đề tài:
Phát triển một hạ tầng cloud-native có thể mở rộng và chịu lỗi trên AWS

Giảng viên hướng dẫn : **TS. NGUYỄN TẤT THẮNG**

Học viên thực hiện : **Nguyễn Huy Tâm**

Lớp : **M23CQIS02-B**

Mã học viên : **B23CHIS059**

Thời gian thực tập : **26/10 - 23/11**

Hệ : **Chính quy**

Hà Nội, Năm 2024

LỜI CẢM ƠN

Lời đầu tiên, em xin gửi lời cảm ơn chân thành và sâu sắc tới thầy TS. Nguyễn Tất Thắng, người đã trực tiếp hướng dẫn tận tình, chu đáo, chia sẻ những ý kiến và kinh nghiệm quý báu trong suốt quá trình em thực hiện đề tài thực tập.

Sau đó, em xin gửi lời cảm ơn các thầy, cô trong Học Viện nói chung và Khoa Đào tạo Sau đại học nói riêng đã luôn nhiệt huyết, tận tình trong từng bài giảng và tạo điều kiện thuận lợi nhất cho em trong thời gian học tập và nghiên cứu tại trường Học Viện Công Nghệ Bưu Chính Viễn Thông.

Bên cạnh đó, em xin gửi lời cảm ơn tới các bạn đồng nghiệp và bạn bè đã luôn bên cạnh, hỗ trợ và động viên em trong suốt quá trình thực hiện đề tài. Những ý kiến đóng góp, chia sẻ và sự giúp đỡ của các bạn đã giúp em hoàn thiện hơn những công việc của mình.

Em cũng xin cảm ơn các nhân viên trong Học viện, những người đã làm việc âm thầm và cống hiến để tạo ra một môi trường học tập và nghiên cứu thuận lợi. Sự nhiệt tình và chu đáo của các bạn đã góp phần không nhỏ vào thành công của em.

Em hy vọng rằng những kiến thức và kinh nghiệm thu được trong thời gian thực tập này sẽ là nền tảng vững chắc cho em trong những bước đi tiếp theo của sự nghiệp.

Em xin chân thành cảm ơn!

Hà Nội, tháng 11 năm 2024

Học viên thực hiện

Nguyễn Huy Tâm

NHẬN XÉT, ĐÁNH GIÁ, CHO ĐIỂM

(Của người hướng dẫn)

……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……

Điểm:…… **(bằng chữ:** ……**)**

……**, ngày**…..... **tháng**…..... **năm 202…**

CÁN BỘ GIẢNG VIÊN HƯỚNG DẪN

TS. Nguyễn Tất Thắng

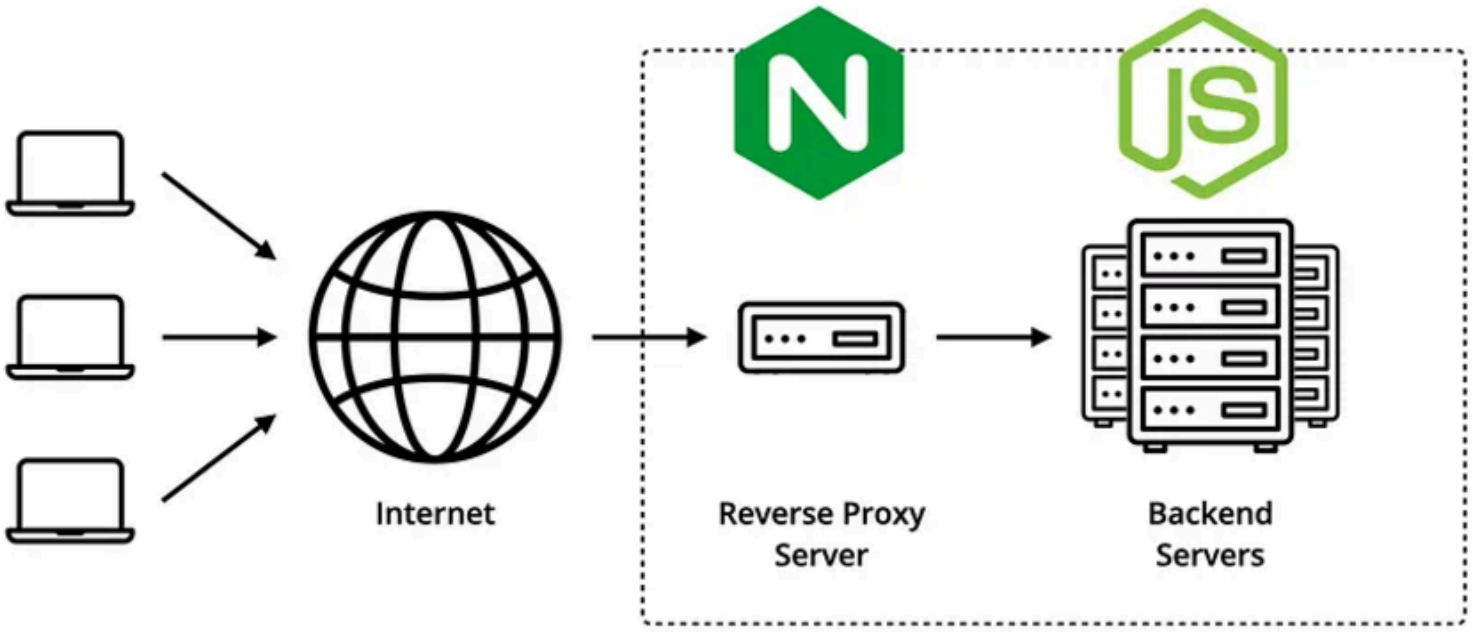
MỤC LỤC

DANH MỤC TÀI LIỆU THAM KHẢO 25

(Phần này chỉ cần chọn toàn bộ các dòng và click chuột trái, chọn Update Field, text tự động cập nhật, nếu sai format font thì format lại font: Times New Roman size 13. Để làm tự động như thế thì ở tiêu đề các đề mục, phải chọn đúng Heading. Tên chương là Heading 1, đề mục 2 số là Heading 2, 3 số là Heading 3 v.v.)

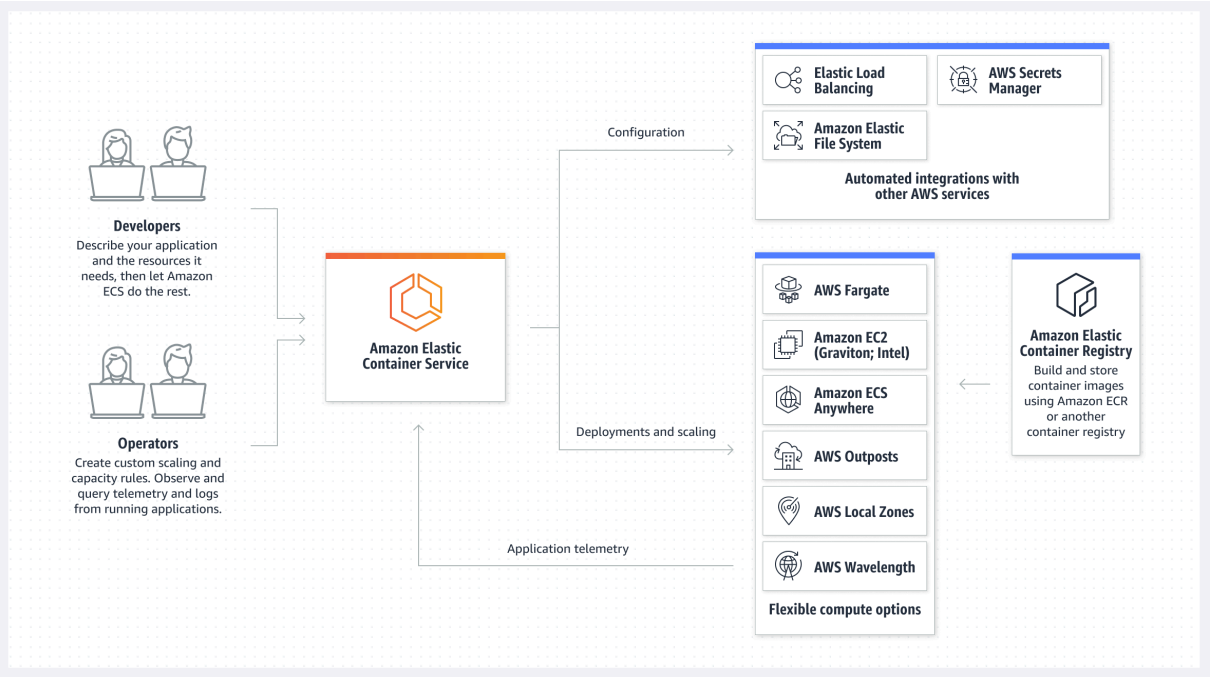
DANH SÁCH HÌNH VẼ

Hình 1.1 Sơ đồ tổng quát

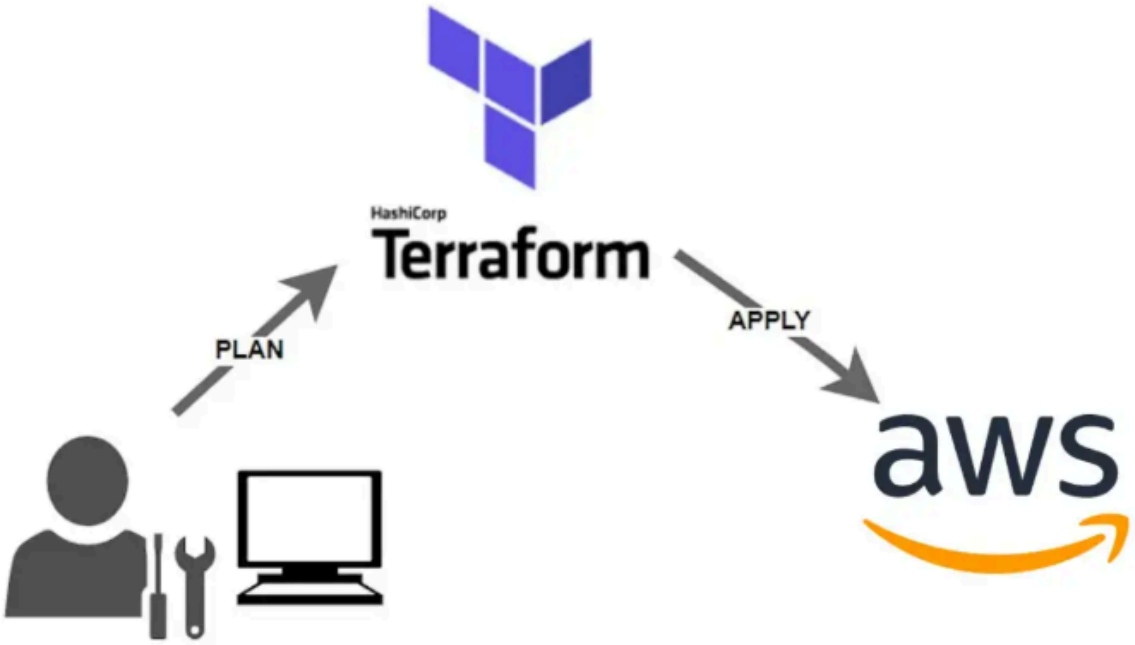


Hình 1.2.Cách thức hoạt động của AWS ECR

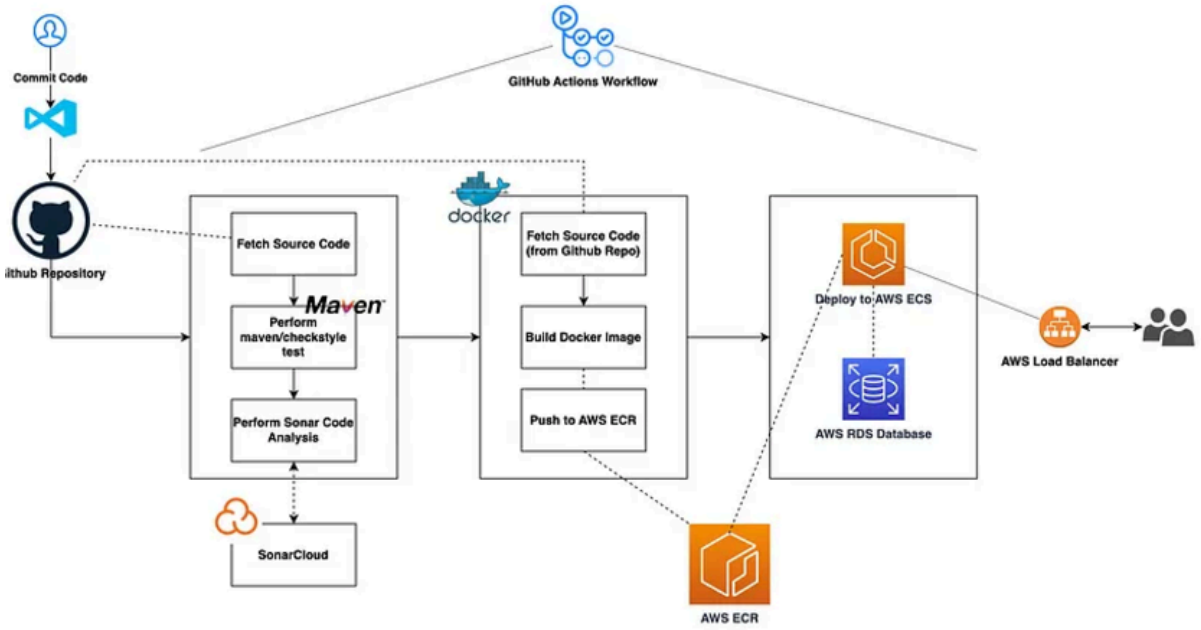
Hình 1.3.Cách thức hoạt động của AWS ECS



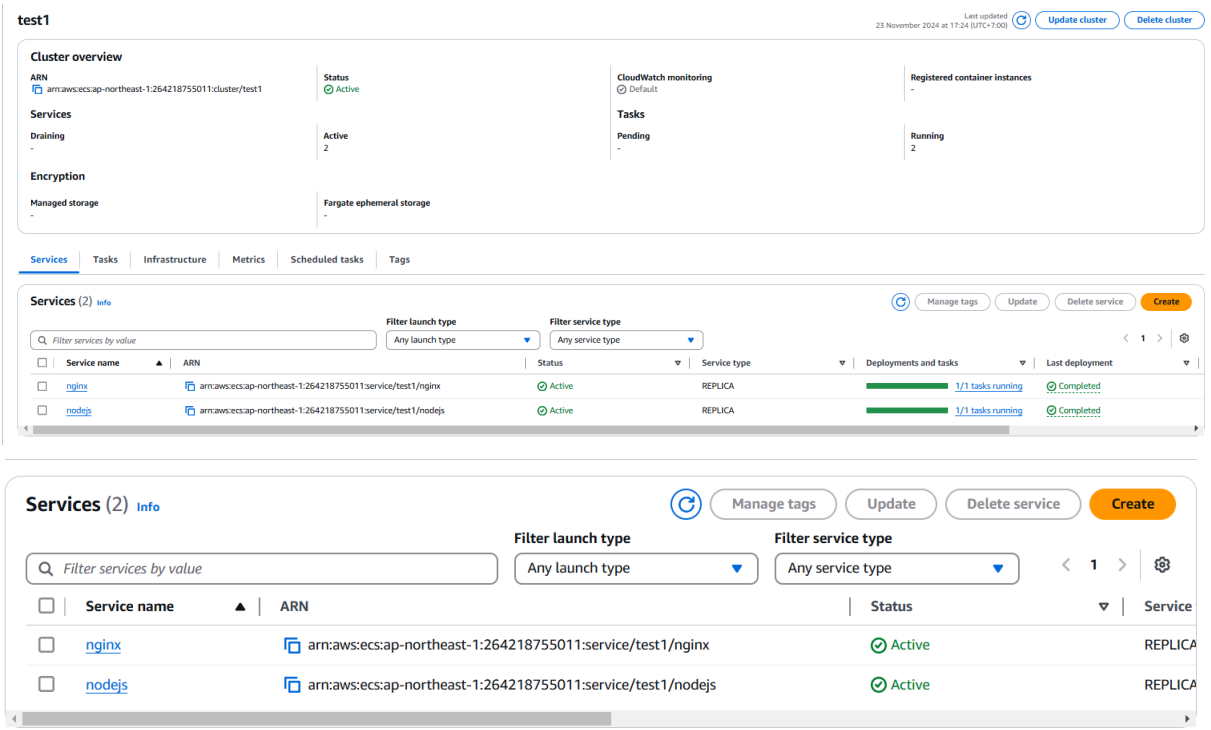
Hình 1.4.Terraform và AWS



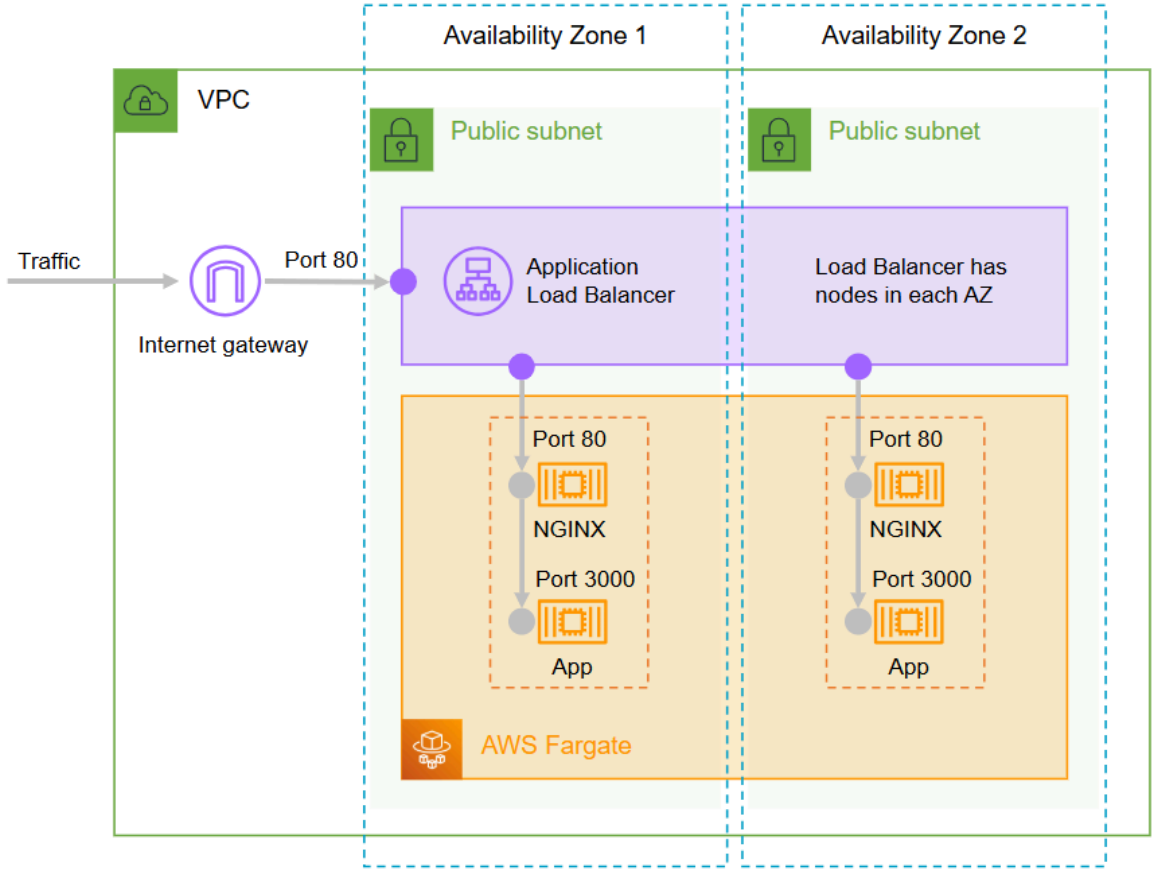
Hình 1.5. Github đến AWS pipeline



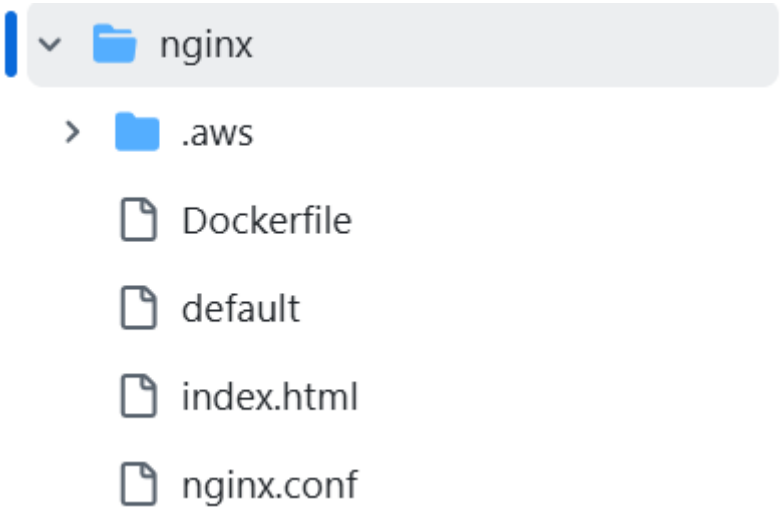
Hình 1.6. Hình vẽ minh hoạ cluster và 2 service đang chạy ECS



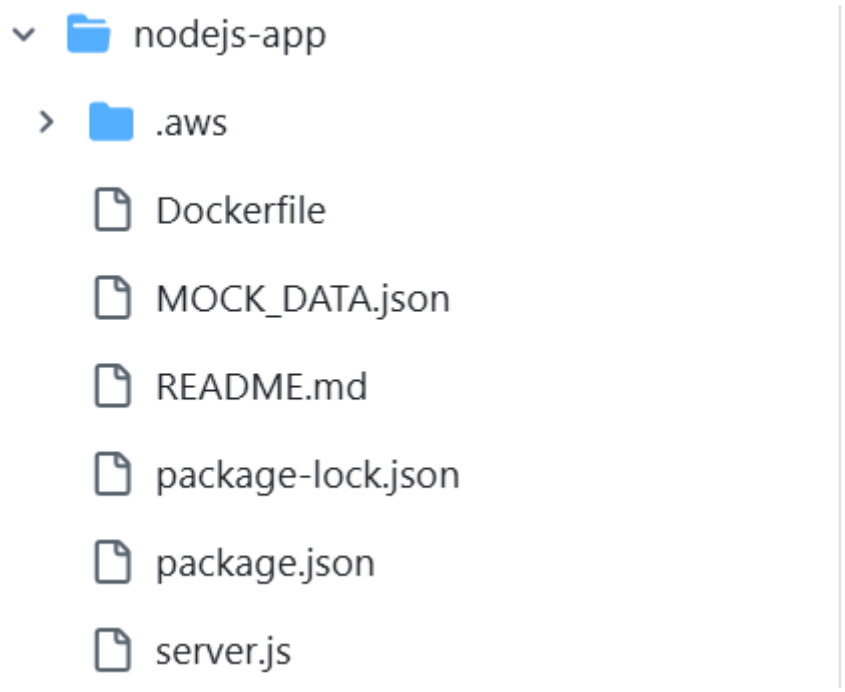
Hình 2.1 Kiến của NGINX và NODEJS trên AWS



Hình 2.2 Kiến trúc Nginx trên github…



Hình 2.4 Kiến trúc nodejs trên github



(Phần này chỉ cần chọn toàn bộ các dòng và click chuột trái, chọn Update Field, text tự động cập nhật, nếu sai format font thì format lại font: Times New Roman size 13. Để làm tự động như thé thì ở các hình, tên hình phài chọn là Heading 7)

DANH SÁCH BẢNG

(Phần này chỉ cần chọn toàn bộ các dòng và click chuột trái, chọn Update Field, text tự động cập nhật, nếu sai format font thì format lại font: Times New Roman size 13. Để làm tự động như thé thì ở các bảng, tên bảng phải chọn là Heading 8)

MỞ ĐẦU

GitHub Actions là một công cụ mạnh mẽ cho phép bạn tự động hóa quy trình phát triển phần mềm thông qua các hành động (actions) dựa trên sự kiện trong kho mã nguồn của bạn. Với GitHub Actions, bạn có thể thiết lập quy trình CI/CD (Continuous Integration/Continuous Deployment) dễ dàng để triển khai ứng dụng lên các dịch vụ của Amazon Web Services (AWS).

Bằng cách sử dụng các workflow được định nghĩa trong file YAML, bạn có thể tự động chạy các bài kiểm tra, biên dịch mã nguồn và triển khai ứng dụng của mình lên AWS mỗi khi có thay đổi trong kho mã. Điều này không chỉ giúp tiết kiệm thời gian mà còn giảm thiểu lỗi do quy trình thủ công.

Ngoài ra, GitHub Actions cung cấp khả năng tích hợp với nhiều dịch vụ khác nhau của AWS như Elastic Beanstalk, Lambda, hoặc ECS, giúp bạn xây dựng các pipeline linh hoạt và mạnh mẽ. Việc kết hợp GitHub Actions với AWS không chỉ nâng cao hiệu suất phát triển mà còn đảm bảo rằng mã của bạn luôn được kiểm tra và triển khai một cách an toàn và hiệu quả.

Nội dung của báo cáo … bao gồm các phần sau:

Chương I: TÌM HIỂU VỀ CÁC DỊCH VỤ AWS, GITHUB, TERRAFORM

Amazon ECR (Elastic Container Registry) là dịch vụ quản lý kho chứa hình ảnh container, cho phép lưu trữ, quản lý và triển khai hình ảnh Docker một cách dễ dàng. Kết hợp với Amazon ECS (Elastic Container Service), dịch vụ này giúp bạn chạy và quản lý các ứng dụng container trên AWS, hỗ trợ cả Docker và các định dạng container khác. Để tự động hóa quy trình phát triển phần mềm, GitHub Actions cung cấp khả năng tích hợp liên tục (CI) và triển khai liên tục (CD), cho phép bạn định nghĩa các workflow tự động hóa kiểm tra, xây dựng và triển khai ứng dụng. Cuối cùng, Terraform là công cụ mã hóa hạ tầng (Infrastructure as Code - IaC) giúp bạn định nghĩa và quản lý hạ tầng AWS thông qua mã nguồn, đảm bảo tính nhất quán và dễ dàng tái sử dụng. Khi kết hợp ECR, ECS, GitHub Actions và Terraform, bạn có thể xây dựng một pipeline CI/CD hiệu quả cho ứng dụng container, từ mã nguồn đến môi trường sản xuất.

Chương II: TRIỂN KHAI NGINX VÀ NODE.JS

riển khai Nginx và Node.js trên AWS bằng GitHub Actions là một quy trình tự động hóa mạnh mẽ giúp bạn dễ dàng quản lý và triển khai ứng dụng web của mình. Đầu tiên, bạn có thể lưu trữ mã nguồn của ứng dụng Node.js trong kho GitHub. Khi có thay đổi trong mã nguồn, GitHub Actions sẽ tự động kích hoạt các workflow để chạy các bài kiểm tra, xây dựng ứng dụng và tạo hình ảnh Docker. Hình ảnh này sau đó được đẩy lên Amazon ECR (Elastic Container Registry).

Tiếp theo, bạn có thể sử dụng Amazon ECS (Elastic Container Service) để triển khai ứng dụng Node.js và Nginx, mà Nginx sẽ đóng vai trò là máy chủ proxy hoặc cân bằng tải. GitHub Actions sẽ tự động cập nhật dịch vụ ECS mỗi khi có phiên bản mới của hình ảnh được đẩy lên. Quy trình này không chỉ giúp bạn giảm thiểu lỗi do các tác vụ thủ công mà còn tối ưu hóa thời gian triển khai, đảm bảo rằng ứng dụng của bạn luôn sẵn sàng và hoạt động hiệu quả trên AWS.

Chương III: TRIỂN KHAI MOODLE VÀ CI/CD

(Sơ lược Chương 3)

…

Kết luận

(Sơ lược Kết luận)

…

CHƯƠNG I. TÌM HIỂU VỀ CÁC DỊCH VỤ AWS, GITHUB, TERRAFORM

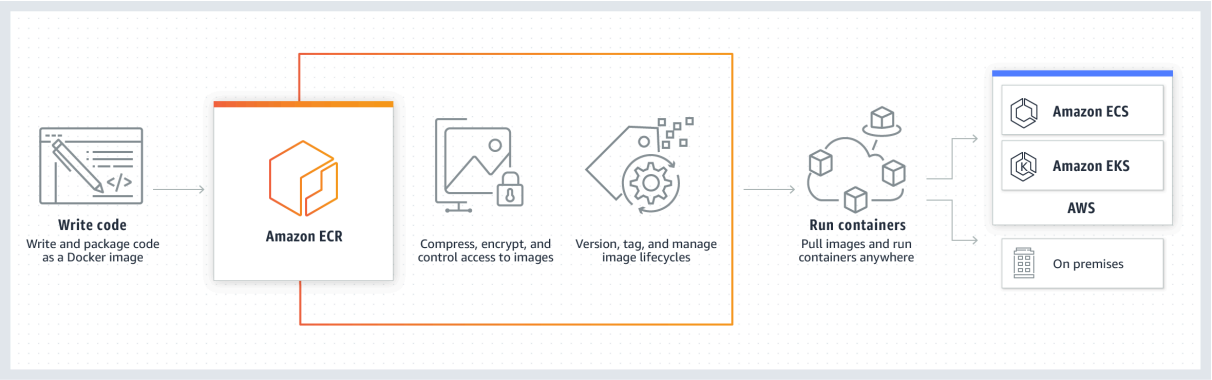
Giới thiệu sơ lược về các dịch vụ của AWS, Github, Terraform

1.1 Các dịch vụ của AWS

1.1.1 Amazon Elastic Container Registry(ECR)

Amazon Elastic Container Registry (ECR) là một dịch vụ quản lý kho chứa hình ảnh container được cung cấp bởi Amazon Web Services (AWS). ECR cho phép bạn lưu trữ, quản lý và triển khai các hình ảnh Docker một cách an toàn và dễ dàng. Dưới đây là một số điểm nổi bật về ECR:

i. **Quản lý hình ảnh dễ dàng**: ECR giúp bạn tổ chức và lưu trữ các hình ảnh Docker, cho phép bạn dễ dàng tìm kiếm và quản lý phiên bản hình ảnh của ứng dụng.
ii. **Tích hợp với AWS**: ECR tích hợp chặt chẽ với các dịch vụ khác của AWS như Amazon ECS (Elastic Container Service) và Amazon EKS (Elastic Kubernetes Service), giúp bạn triển khai ứng dụng container một cách nhanh chóng và hiệu quả.
iii. **Bảo mật**: ECR cung cấp các tính năng bảo mật mạnh mẽ, bao gồm việc quản lý quyền truy cập thông qua AWS Identity and Access Management (IAM), mã hóa hình ảnh khi lưu trữ và khả năng quét hình ảnh để phát hiện lỗ hổng bảo mật.
iv. **Tự động hóa**: ECR hỗ trợ tự động hóa quy trình xây dựng và triển khai hình ảnh, giúp giảm thiểu thời gian và công sức cần thiết cho việc quản lý ứng dụng container.
v. **Hiệu suất cao**: Với khả năng mở rộng linh hoạt, ECR có thể đáp ứng nhu cầu của các ứng dụng lớn và phức tạp, đảm bảo hiệu suất và độ tin cậy cao.

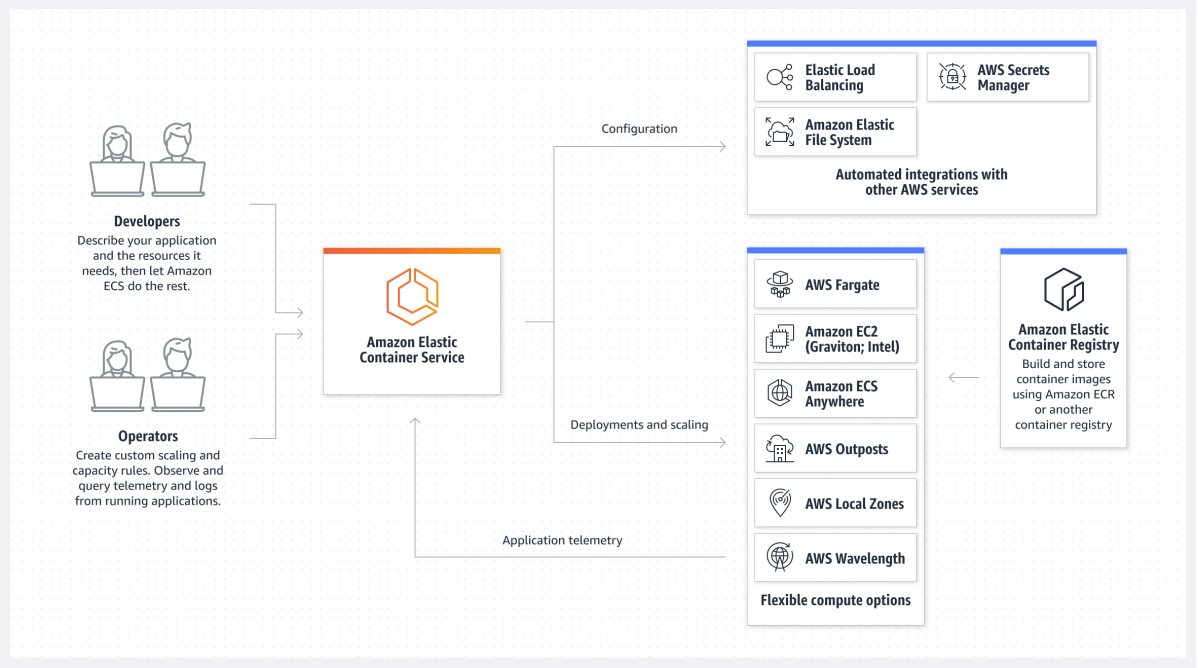


(Cách thức hoạt động của AWS ECR)

1.1.2 Amazon Elastic Container Service (ECS)

Amazon Elastic Container Service (ECS) là một dịch vụ quản lý container của AWS giúp bạn dễ dàng triển khai, quản lý và điều chỉnh quy mô các ứng dụng được đóng gói trong container. Với ECS, bạn không cần phải lo lắng về việc quản lý các máy chủ vật lý, thay vào đó, bạn chỉ cần tập trung vào việc phát triển ứng dụng của mình. ECS cung cấp các tính năng mạnh mẽ như tự động điều chỉnh quy mô, tích hợp với các dịch vụ khác của AWS và dễ sử dụng, giúp bạn tiết kiệm thời gian và chi phí.

=> Nói cách khác, ECS là một công cụ hữu ích để bạn chạy các ứng dụng container trên đám mây AWS một cách hiệu quả.



(Cách thức hoạt động của AWS ECS)

1.2 Sơ lược về Github

1.2.1 Các lệnh Github thường gặp

Các lệnh Git cơ bản (sử dụng trên terminal)

- **git init:** Khởi tạo một kho lưu trữ Git mới trong thư mục hiện tại.
- **git clone:** Sao chép một kho lưu trữ từ xa (trên GitHub) về máy cục bộ.
- **git add:** Thêm các thay đổi vào vùng chờ (staging area) để chuẩn bị commit.
- **git commit:** Lưu các thay đổi đã được thêm vào vùng chờ vào lịch sử của kho lưu trữ.

- **git push:** Đẩy các commit từ kho lưu trữ cục bộ lên kho lưu trữ từ xa (trên GitHub).
- **git pull:** Lấy các commit mới từ kho lưu trữ từ xa và hợp nhất chúng vào kho lưu trữ cục bộ.
- **git status:** Hiển thị trạng thái hiện tại của kho lưu trữ, bao gồm các tệp đã thay đổi, các tệp chưa được theo dõi.
- **git branch:** Tạo, liệt kê, xóa các nhánh.
- **git checkout:** Chuyển đổi giữa các nhánh.
- **git merge:** Hợp nhất các nhánh lại với nhau.

Các lệnh Git nâng cao và liên quan đến GitHub

- **git remote:** Quản lý các kho lưu trữ từ xa (ví dụ: thêm, xóa, xem danh sách).
- **git fetch:** Lấy thông tin về các nhánh và commit mới từ kho lưu trữ từ xa mà không tự động hợp nhất.
- **git rebase:** Thay đổi cơ sở của một nhánh, thường được sử dụng để sắp xếp lại lịch sử commit.
- **git revert:** Hoàn tác một commit bằng cách tạo ra một commit mới để đảo ngược các thay đổi.
- **git tag:** Tạo các nhãn (tag) để đánh dấu các điểm quan trọng trong lịch sử commit.

Các lệnh tương tác với GitHub (thường dùng trên giao diện web)

- **Tạo một repository mới:** Tạo một kho lưu trữ mới trên GitHub.
- **Fork một repository:** Tạo một bản sao của một repository hiện có để thực hiện các thay đổi riêng.
- **Pull request:** Gửi yêu cầu hợp nhất các thay đổi từ một nhánh vào nhánh chính (thường là main hoặc master).
- **Issue:** Tạo các vấn đề (issue) để theo dõi lỗi, yêu cầu tính năng hoặc thảo luận.
- **Review code:** Đánh giá các thay đổi được đề xuất trong pull request.
- **Merge pull request:** Hợp nhất các thay đổi từ pull request vào nhánh chính.

Giải thích thêm về các lệnh liên quan đến GitHub

- **Fork:** Khi bạn fork một repository, bạn tạo một bản sao độc lập của repository đó vào tài khoản của mình. Bạn có thể thực hiện các thay đổi trên bản sao này mà không ảnh hưởng đến repository gốc.
- **Pull request:** Khi bạn muốn đóng góp vào một project trên GitHub, bạn sẽ tạo một pull request. Điều này sẽ thông báo cho chủ sở hữu project về các thay đổi bạn đã thực hiện và họ có thể xem xét và hợp nhất các thay đổi của bạn.
- **Issue:** Issues được sử dụng để theo dõi các công việc cần làm, các lỗi phát sinh, các ý tưởng mới hoặc các yêu cầu tính năng. Mỗi issue có thể được gắn nhãn, gán cho người dùng và có các bình luận.

1.2.2 Các bước để tạo một github action cho repo của mình

Bước 1: Tạo kho lưu trữ mới hoặc sử dụng kho hiện có

- Nếu bạn chưa có kho lưu trữ, hãy tạo một kho mới trên GitHub.
- Nếu bạn đã có kho lưu trữ, hãy truy cập vào kho đó.

Bước 2: Tạo thư mục cho GitHub Actions

- Trong kho lưu trữ của bạn, hãy tạo một thư mục có tên `.github/workflows`. Thư mục này sẽ chứa các file cấu hình cho workflow.

Bước 3: Tạo file workflow

- Trong thư mục `.github/workflows`, tạo một file YAML cho workflow của bạn. Ví dụ, bạn có thể đặt tên file là `ci.yml`.

Bước 4: Định nghĩa workflow trong file YAML

- Mở file `ci.yml` và thêm cấu hình cho workflow của bạn. Dưới đây là một ví dụ đơn giản:

```yaml
name: CI

on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
```

```
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Set up Node.js
        uses: actions/setup-node@v2
        with:
          node-version: '14'

      - name: Install dependencies
        run: npm install

      - name: Run tests
        run: npm test
```

Bước 5: Lưu và commit file

- Lưu các thay đổi trong file `ci.yml` và commit vào kho lưu trữ của bạn:

bash
Copy

```
git add .github/workflows/ci.yml
git commit -m "Add CI workflow"
git push origin main
```

Ví dụ về file .yml push code lên AWS ECS

Dockerfile | nginx.yml | default | server.js

.github > workflows > nginx.yml > {} jobs > {} deploy > [ ] steps > {} 3 > run

GitHub Workflow - YAML GitHub Workflow (github-workflow.json)

```yaml
name: Deploy to Amazon ECS

on:
  push:
    paths:
      - "nginx/**"
    branches: [ "main" ]

env:
  AWS_REGION: ap-northeast-1                  # set this to your preferred AWS region, e.g. us-west-1
  ECR_REPOSITORY: nginx           # set this to your Amazon ECR repository name
  ECS_SERVICE: nginx                 # set this to your Amazon ECS service name
  ECS_CLUSTER: test1               # set this to your Amazon ECS cluster name
  ECS_TASK_DEFINITION: nginx/.aws/nginx-revision3.json # set this to the path to your Amazon ECS task definition
                                                       # file, e.g. .aws/task-definition.json
  CONTAINER_NAME: nginx-container           # set this to the name of the container in the
                                                       # containerDefinitions section of your task definition

permissions:
  contents: read

jobs:
  deploy:
    name: Deploy
    runs-on: ubuntu-22.04
    steps:
    - name: Checkout
      uses: actions/checkout@v4

    - name: Configure AWS credentials
      uses: aws-actions/configure-aws-credentials@v1
      with:
        aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
        aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
        aws-region: ${{ env.AWS_REGION }}

    - name: Login to Amazon ECR
      id: login-ecr
      uses: aws-actions/amazon-ecr-login@v1

    - name: Build, tag, and push image to Amazon ECR
      id: build-image
      env:
        ECR_REGISTRY: ${{ steps.login-ecr.outputs.registry }}
        IMAGE_TAG: ${{ github.sha }}
      run: |
        # Build a docker container and
        # push it to ECR so that it can
        # be deployed to ECS.
        cd nginx
        docker build -t $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG .
        docker push $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG
        echo "image=$ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG" >> $GITHUB_OUTPUT

    - name: Fill in the new image ID in the Amazon ECS task definition
      id: task-def
      uses: aws-actions/amazon-ecs-render-task-definition@v1
      with:
        task-definition: ${{ env.ECS_TASK_DEFINITION }}
        container-name: ${{ env.CONTAINER_NAME }}
        image: ${{ steps.build-image.outputs.image }}

    - name: Deploy Amazon ECS task definition
      uses: aws-actions/amazon-ecs-deploy-task-definition@v1
      with:
        task-definition: ${{ steps.task-def.outputs.task-definition }}
        service: ${{ env.ECS SERVICE }}
```

1.3 Tìm hiểu về Terraform

1.3.1 Các khái niệm cơ bản về Terraform

1. Infrastructure as Code (IaC)

Infrastructure as Code (IaC) là một phương pháp quản lý hạ tầng thông qua mã nguồn. Thay vì cấu hình hạ tầng bằng tay, bạn viết mã để mô tả các thành phần hạ tầng, như máy chủ, mạng, và dịch vụ. IaC giúp tự động hóa việc triển khai, cải thiện tính nhất quán và dễ dàng theo dõi các thay đổi trong hạ tầng. Terraform cho phép bạn viết mã cấu hình cho hạ tầng, giúp dễ dàng quản lý và tái sử dụng.

2. Provider

Providers là các plugin trong Terraform cho phép bạn tương tác với các dịch vụ và API của nhà cung cấp hạ tầng như AWS, Azure, Google Cloud, và nhiều dịch vụ khác. Mỗi provider có các tài nguyên riêng mà bạn có thể sử dụng trong file cấu hình. Ví dụ, provider cho AWS sẽ bao gồm các tài nguyên như EC2 instances, S3 buckets, và RDS databases. Bạn cần cấu hình provider trong file Terraform để thiết lập kết nối với dịch vụ mà bạn muốn quản lý.

3. Resource

Resource là các thành phần cụ thể trong hạ tầng mà bạn muốn quản lý, như máy chủ ảo, cơ sở dữ liệu, hoặc dịch vụ lưu trữ. Mỗi resource được định nghĩa trong file cấu hình Terraform bằng cách sử dụng cú pháp resource "type" "name". Ví dụ:
Copy

```
resource "aws_instance" "example" {
  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = "t2.micro"
}
```

Trong ví dụ này, một EC2 instance mới sẽ được tạo ra với AMI và loại máy chủ được chỉ định.

4. Module

Modules là các nhóm resource có thể tái sử dụng, giúp bạn tổ chức và quản lý hạ tầng một cách hiệu quả. Bạn có thể tạo một module cho một phần hạ tầng cụ thể (chẳng hạn như một ứng dụng web) và sử dụng lại module đó trong nhiều dự án khác nhau. Một module có thể chứa nhiều resource và có thể được định nghĩa trong thư mục riêng. Bạn sử dụng từ khóa module để gọi module trong file cấu hình.

5. State File

File trạng thái (state file) là nơi Terraform lưu trữ thông tin về trạng thái hiện tại của hạ tầng. File này giúp Terraform theo dõi các resource đã được tạo và trạng thái của chúng. Khi bạn thực hiện thay đổi, Terraform so sánh file trạng thái với cấu hình hiện tại để xác định những gì cần thay đổi. File trạng thái là rất quan trọng và cần được bảo vệ, vì nó chứa thông tin nhạy cảm.

6. Configuration File

Các file cấu hình được viết bằng ngôn ngữ HCL (HashiCorp Configuration Language). HCL rất dễ đọc và giống với JSON nhưng cung cấp cú pháp mạnh mẽ hơn cho các biểu thức. File cấu hình định nghĩa các resource, provider, và các thuộc tính liên quan đến hạ tầng. File cấu hình thường có đuôi .tf.

7. Execution Plan

Khi bạn chạy lệnh terraform plan, Terraform tạo ra một kế hoạch thực thi (execution plan) mô tả những thay đổi mà nó sẽ thực hiện dựa trên cấu hình và trạng thái hiện tại. Kế hoạch này cho phép bạn xem trước các thay đổi mà Terraform sẽ thực hiện trước khi chúng được áp dụng, giúp tránh những thay đổi không mong muốn.

8. Apply

Lệnh terraform apply được sử dụng để thực hiện các thay đổi được định nghĩa trong file cấu hình. Terraform sẽ tạo, sửa đổi, hoặc xóa resource dựa trên kế hoạch đã được xác định. Trước khi thực hiện, Terraform sẽ hỏi bạn xác nhận, trừ khi bạn sử dụng tham số -auto-approve.

9. Destroy

Lệnh terraform destroy được sử dụng để xóa tất cả các resource được quản lý bởi Terraform trong một cấu hình cụ thể. Đây là cách nhanh chóng để làm sạch hạ tầng mà không cần phải xóa từng resource một.

10. Variable

Các biến (variables) cho phép bạn cấu hình các giá trị mà không cần thay đổi trực tiếp trong file cấu hình. Bạn có thể định nghĩa biến bằng cách sử dụng từ khóa variable và có thể cung cấp giá trị cho chúng qua dòng lệnh, file cấu hình, hoặc biến môi trường. Điều này giúp tăng tính linh hoạt và tái sử dụng cho file cấu hình.

11. Output

Outputs là các giá trị mà bạn có thể xuất ra sau khi thực thi Terraform. Chúng cho phép bạn hiển thị thông tin quan trọng về các resource, như địa chỉ IP hoặc URL của ứng dụng. Outputs có thể được sử dụng trong các module khác hoặc để cung cấp thông tin cho người dùng.

12. Data Source

Data sources cho phép bạn truy xuất thông tin từ các resource hiện có mà không cần quản lý chúng bằng Terraform. Bạn có thể sử dụng data sources để lấy thông tin như danh sách subnet, AMI mới nhất, hoặc thông tin về người dùng trong IAM. Điều này rất hữu ích khi bạn muốn cấu hình resource dựa vào thông tin từ hạ tầng hiện có.

13. Provisioner

Provisioners cho phép bạn thực hiện các tác vụ cấu hình trên resource sau khi chúng được tạo ra. Bạn có thể sử dụng provisioners để cài đặt phần mềm, chạy script, hoặc cấu hình resource

theo cách mà bạn muốn. Tuy nhiên, việc sử dụng provisioners nên được hạn chế, vì chúng có thể làm phức tạp hóa quá trình quản lý hạ tầng.

1.4 Kết luận Chương I

Trong chương này, chúng ta đã đi sâu vào các dịch vụ của Amazon Web Services (AWS), cụ thể là Amazon ECS (Elastic Container Service) và Amazon ECR (Elastic Container Registry), cùng với GitHub Actions, các lệnh GitHub và Terraform. AWS ECS cung cấp khả năng quản lý và triển khai ứng dụng container trên quy mô lớn, cho phép các nhà phát triển dễ dàng chạy và mở rộng ứng dụng mà không cần lo lắng về cơ sở hạ tầng. Kết hợp với Amazon ECR, dịch vụ lưu trữ hình ảnh container, quy trình triển khai ứng dụng trở nên mượt mà và hiệu quả hơn, giúp các tổ chức tối ưu hóa việc sử dụng tài nguyên và giảm thiểu rủi ro trong khi triển khai.

GitHub Actions, với khả năng tự động hóa quy trình CI/CD, đã tạo ra một bước tiến lớn trong việc quản lý mã nguồn và tự động hóa quy trình phát triển phần mềm. Việc tích hợp GitHub Actions với các dịch vụ AWS giúp tự động hóa không chỉ việc kiểm tra và xây dựng mã, mà còn cả việc triển khai ứng dụng lên môi trường sản xuất. Bên cạnh đó, các lệnh GitHub giúp người dùng dễ dàng quản lý và theo dõi thay đổi trong kho mã, đảm bảo tính nhất quán và an toàn trong quá trình phát triển.

Terraform, với khả năng quản lý hạ tầng dưới dạng mã (Infrastructure as Code - IaC), đóng vai trò quan trọng trong việc định nghĩa và kiểm soát toàn bộ hạ tầng một cách tự động và nhất quán. Việc sử dụng Terraform cho phép các đội phát triển triển khai và quản lý tài nguyên AWS một cách hiệu quả, từ đó giảm thiểu lỗi và tăng cường khả năng tái sử dụng cấu hình hạ tầng.

Sự kết hợp giữa AWS ECS, ECR, GitHub Actions và Terraform không chỉ tối ưu hóa quy trình phát triển và triển khai ứng dụng mà còn nâng cao tính linh hoạt và khả năng mở rộng của hạ tầng. Điều này giúp các tổ chức đáp ứng nhanh chóng với thay đổi và nhu cầu thị trường, đồng thời thúc đẩy sự đổi mới và phát triển bền vững. Khi công nghệ tiếp tục phát triển, việc áp dụng các công cụ và dịch vụ này sẽ trở thành yếu tố then chốt cho sự thành công trong môi trường phát triển phần mềm hiện đại.

CHƯƠNG II. Triển Khai Nodejs và Nginx

Triển khai Nginx làm reverse proxy cho ứng dụng Node.js là một giải pháp hiệu quả giúp tối ưu hóa hiệu suất và bảo mật cho các ứng dụng web. Nginx, với khả năng xử lý hàng triệu kết nối đồng thời, sẽ tiếp nhận tất cả các yêu cầu từ người dùng và chuyển tiếp chúng đến ứng dụng Node.js đang chạy ở phía sau. Việc này không chỉ giúp ẩn thông tin chi tiết về ứng dụng mà còn bảo vệ nó khỏi các tấn công trực tiếp. Hơn nữa, Nginx có thể phục vụ các tệp tĩnh nhanh chóng và thực hiện cân bằng tải giữa nhiều instance của ứng dụng Node.js, từ đó cải thiện khả năng mở rộng và độ tin cậy. Với những lợi ích này, việc sử dụng Nginx như một reverse proxy trở thành một phần quan trọng trong kiến trúc của nhiều ứng dụng hiện đại.

2.1 Cài đặt Nginx

2.1.1 Tạo môi trường cho Nginx

Tạo Dockerfile với các môi trường cần thiết, mở các port cần thiết

```
# Sử dụng image Ubuntu 22.04 làm base
FROM ubuntu:22.04

# Cài đặt Nginx và các gói cần thiết
RUN apt-get update && \
    apt-get install -y nginx && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

RUN ls -la /etc/nginx/sites-available
RUN ls -la /etc/nginx/sites-available/
RUN ls -la /etc/nginx/sites-enabled
RUN ls -la /etc/nginx/sites-enabled
RUN cat /etc/nginx/sites-available/default


COPY index.html /var/www/html
COPY default /etc/nginx/sites-available

RUN cat /etc/nginx/sites-available/default

# Mở cổng 80
EXPOSE 80
EXPOSE 85
EXPOSE 5000

# Khởi động Nginx
CMD ["nginx", "-g", "daemon off;"]
```

2.1.2 Thay đổi các file config sao cho Nginx như là một reverse proxy

Tạo các file config phù hợp để khiến Nginx như là một reverse proxy

1, File default sẽ được thay thế trong đường dẫn /etc/nginx/sites-availble

```
nginx > default
# /etc/nginx/sites-available/default

server {
    listen 80;
    location / {
        # Cấu hình reverse proxy cho đường dẫn /node
        proxy_pass http://18.183.251.98:5000;  # Thay backend_server bằng địa chỉ server backend của bạn
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

    }

}

```

2, File index.html để config trang chủ của nginx

```
nginx > index.html > ...
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title>Hello, Nginx!</title>
</head>
<body>
    <h1>Hello, Nginx!</h1>
    <p>We have just configured ouraaaaaaaabbbbaavvvvaaaccccaaaddaaaaaefffeeaaa Nginx web server on Ubuntu Server!</p>
</body>
</html>

```

2.2 Cài đặt Nodejs

2.2.1 Tạo môi trường cho nodejs app

Tạo Dockerfile với các môi trường cần thiết, mở các port cần thiết

```
FROM node:16.20.1
WORKDIR /app
COPY package.json ./
RUN npm install
COPY . .
EXPOSE 5000
CMD ["npm","run","start"]
```

Hình 2.1 (dockerfile nodejs)

(Chèn Hình vào đây nếu có)

Hình 2.2 ...

...

...

2.2.2 Cài đặt code và chỉnh sửa config

Thay đổi port và code đơn giản về nodejs

Trước khi thực hiện cần chạy câu lệnh "**npm install**" trong folder mà đang chưa file code để nó install các file cần thiết như là **package.json**

```
const express = require("express");
const app = express();
const PORT = 5000;
const userData = require("./MOCK_DATA.json");
const graphql = require("graphql")
const { GraphQLObjectType, GraphQLSchema, GraphQLList, GraphQLID, GraphQLInt,
GraphQLString } = graphql
const { graphqlHTTP } = require("express-graphql")

const UserType = new GraphQLObjectType({
    name: "User",
    fields: () => ({
        id: { type: GraphQLInt },
        firstName: { type: GraphQLString },
        lastName: { type: GraphQLString },
        email: { type: GraphQLString },
        password: { type: GraphQLString },
    })
})

const RootQuery = new GraphQLObjectType({
    name: "RootQueryType",
    fields: {
        getAllUsers: {
            type: new GraphQLList(UserType),
            args: { id: {type: GraphQLInt}},
            resolve(parent, args) {
                return userData;
            }
        },
        findUserById: {
            type: UserType,
            description: "fetch single user",
            args: { id: {type: GraphQLInt}},
            resolve(parent, args) {
                return userData.find((a) => a.id == args.id);
            }
```

```
        }
    }
})
const Mutation = new GraphQLObjectType({
    name: "Mutation",
    fields: {
        createUser: {
            type: UserType,
            args: {
                firstName: {type: GraphQLString},
                lastName: { type: GraphQLString },
                email: { type: GraphQLString },
                password: { type: GraphQLString },
            },
            resolve(parent, args) {
                userData.push({
                    id: userData.length + 1,
                    firstName: args.firstName,
                    lastName: args.lastName,
                    email: args.email,
                    password: args.password
                })
                return args
            }
        }
    }
})

const schema = new GraphQLSchema({query: RootQuery, mutation: Mutation})
app.use("/graphql", graphqlHTTP({
    schema,
    graphiql: true,
  })
);
// Health check endpoint
app.get('/', (req, res) => {
    res.status(200).send('Healthy');
});

app.get("/rest/getAllUsers", (req, res) => {
    res.send(userData)
   });

app.listen(PORT, () => {
  console.log("Server runninaaaaagggg");
});
```

Hình 2.3 (Code đơn giản về app nodejs)

...

2.3 Kết luận Chương II

Trong kết luận này, chúng ta đã xem xét một quy trình chi tiết từ cài đặt Node.js, cấu hình Nginx làm reverse proxy, cho đến việc triển khai ứng dụng này trên AWS ECS (Elastic Container Service). Bắt đầu từ việc cài đặt Node.js, chúng ta đã thiết lập một nền tảng mạnh mẽ cho các ứng dụng backend, cho phép xử lý các yêu cầu từ người dùng một cách hiệu quả. Node.js không chỉ nổi bật với hiệu suất cao mà còn cho phép phát triển ứng dụng theo kiểu bất đồng bộ, rất phù hợp cho các ứng dụng web hiện đại.

Tiếp theo, việc cấu hình Nginx làm reverse proxy là một bước quan trọng trong việc tối ưu hóa hiệu suất và bảo mật cho ứng dụng. Nginx tiếp nhận tất cả các yêu cầu từ người dùng và chuyển tiếp chúng đến Node.js, giúp ẩn đi thông tin chi tiết về ứng dụng và bảo vệ nó khỏi các cuộc tấn công trực tiếp. Ngoài ra, Nginx có khả năng phục vụ các tệp tĩnh nhanh chóng, xử lý hàng triệu kết nối đồng thời và thực hiện cân bằng tải giữa nhiều instance của ứng dụng Node.js. Điều này không chỉ cải thiện hiệu suất mà còn giúp tăng cường khả năng mở rộng và độ tin cậy của ứng dụng.

Khi triển khai trên AWS ECS, chúng ta tận dụng sức mạnh của hạ tầng đám mây để quản lý và mở rộng ứng dụng một cách linh hoạt. AWS ECS cho phép tự động mở rộng các container dựa trên lưu lượng và nhu cầu, giúp tiết kiệm chi phí và tối ưu hóa tài nguyên. Việc tích hợp với các dịch vụ khác của AWS như RDS cho cơ sở dữ liệu hoặc S3 cho lưu trữ tệp cũng tạo ra một hệ sinh thái toàn diện, hỗ trợ cho việc phát triển và triển khai ứng dụng.

Đặc biệt, việc lưu trữ mã nguồn và cấu hình trong kho GitHub không chỉ giúp dễ dàng theo dõi và quản lý các thay đổi mà còn mở ra cơ hội cho quy trình CI/CD thông qua GitHub Actions. Điều này cho phép tự động hóa các bước kiểm tra, xây dựng và triển khai, giúp tăng tốc độ phát triển và giảm thiểu lỗi do thao tác thủ công.

Tổng kết lại, việc cài đặt Node.js và Nginx như một reverse proxy trên AWS ECS không chỉ tạo ra một kiến trúc ứng dụng hiện đại và hiệu quả mà còn giúp các tổ chức nhanh chóng đáp ứng với các thay đổi và nhu cầu của thị trường. Sự tích hợp chặt chẽ giữa các công nghệ này không chỉ nâng cao tính bảo mật và khả năng phục hồi của ứng dụng mà còn thúc đẩy sự phát triển bền vững trong bối cảnh công nghệ ngày càng phát triển và cạnh tranh. Với những lợi ích này, tổ chức của bạn có thể tự tin tiến bước trong việc phát triển và triển khai các ứng dụng web mạnh mẽ, đáp ứng nhu cầu ngày càng cao của người dùng.

CHƯƠNG III. Giám Sát, Tối Ưu Chi Phí

AWS CloudWatch là một dịch vụ giám sát và theo dõi hiệu năng của các ứng dụng và tài nguyên trên AWS. Nó cung cấp một cái nhìn tổng quan về hoạt động của hệ thống của bạn, giúp bạn phát hiện và giải quyết các vấn đề một cách nhanh chóng.

3.1 Giám sát bằng cloudwatch

AWS CloudWatch là một công cụ không thể thiếu đối với các tổ chức đang sử dụng AWS. Nó cung cấp một giải pháp toàn diện để giám sát và quản lý hiệu năng của các ứng dụng và tài nguyên trên đám mây. Bằng cách tận dụng các tính năng mạnh mẽ của CloudWatch, bạn có thể đảm bảo rằng hệ thống của mình luôn hoạt động ổn định và hiệu quả.

1. Giám sát toàn diện:

- **Theo dõi đa dạng:** CloudWatch cho phép bạn theo dõi các chỉ số từ hàng chục dịch vụ AWS khác nhau, bao gồm EC2, S3, RDS, Lambda, và nhiều hơn nữa.
- **Tùy chỉnh metric:** Bạn có thể tạo các metric tùy chỉnh để theo dõi các chỉ số đặc biệt quan trọng đối với ứng dụng của mình.
- **Logs:** Ngoài metric, CloudWatch Logs còn giúp bạn thu thập và phân tích các log của ứng dụng để tìm ra nguyên nhân gốc rễ của vấn đề.

2. Phân tích dữ liệu sâu sắc:

- **Metric Math:** Thực hiện các phép tính toán học trên các metric để tạo ra các chỉ số mới, giúp bạn hiểu rõ hơn về hệ thống của mình.
- **Alarms:** Tạo các cảnh báo tự động khi các metric vượt quá ngưỡng cho phép, giúp bạn nhanh chóng nhận biết các vấn đề.
- **Dashboard:** Tạo các dashboard tùy chỉnh để trực quan hóa dữ liệu và theo dõi hiệu suất của hệ thống.

3. Tích hợp với các dịch vụ khác:

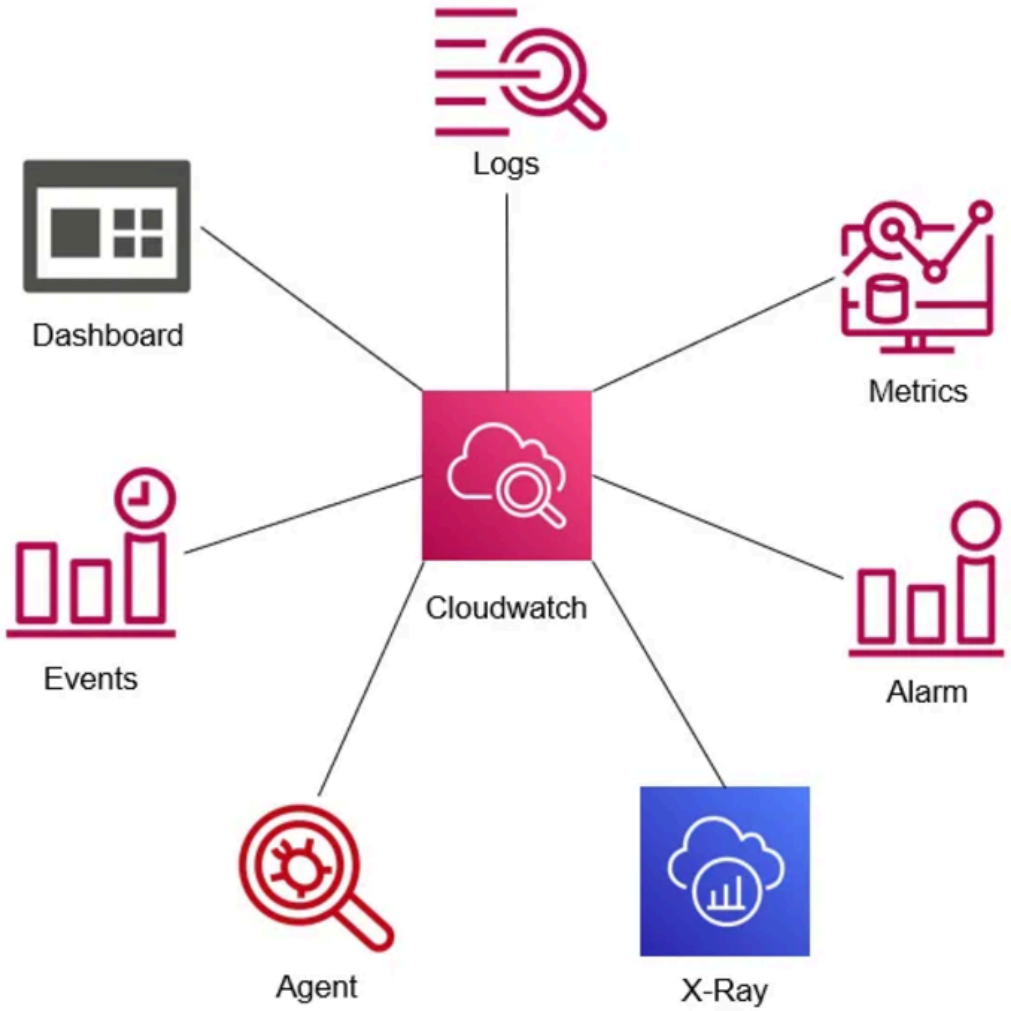
- **AWS Lambda:** Tạo các hàm Lambda để tự động hóa các tác vụ dựa trên các cảnh báo của CloudWatch.
- **Amazon SNS:** Gửi thông báo đến các kênh khác nhau như email, SMS khi có sự cố xảy ra.
- **Amazon SQS:** Đặt các thông báo vào hàng đợi để xử lý sau.

4. Tối ưu hóa chi phí:

- **Phân tích chi phí:** Theo dõi chi phí sử dụng các dịch vụ AWS để xác định các khu vực cần tối ưu hóa.
- **Tạo báo cáo chi phí:** Tạo các báo cáo chi tiết để theo dõi xu hướng chi tiêu.

5. Bảo mật và tuân thủ:

- **Bảo mật:** CloudWatch được tích hợp với các dịch vụ bảo mật của AWS để bảo vệ dữ liệu của bạn.
- **Tuân thủ:** CloudWatch tuân thủ các tiêu chuẩn bảo mật và tuân thủ như PCI DSS và HIPAA.



Hình 3.1 (Các chức năng của cloudwatch)

....

....

3.2 So sánh chi phí giữa AWS EC2 và AWS ECS Fargate

Dưới đây là bảng so sánh chi phí giữa việc sử dụng AWS EC2 và AWS ECS Fargate, bao gồm các lợi ích và hạn chế của từng hệ thống.

Tiêu chí	AWS EC2	AWS ECS Fargate
Chi phí tính theo	Chi phí theo giờ hoặc theo giây cho phiên bản EC2, tính phí cho EBS và băng thông.	Chi phí dựa trên số lượng CPU và RAM sử dụng, tính phí theo giây.
Quản lý tài nguyên	Người dùng phải tự quản lý, triển khai và cấu hình các instance, bao gồm cả cập nhật và bảo trì.	Tự động quản lý tài nguyên, không cần lo lắng về việc cấu hình server.
Khả năng mở rộng	Cần phải cấu hình Auto Scaling Groups để tự động mở rộng, có thể phức tạp hơn.	Tự động mở rộng dựa trên nhu cầu, dễ dàng hơn trong việc điều chỉnh quy mô.
Bảo trì	Người dùng phải thực hiện bảo trì, cập nhật và quản lý các instance.	AWS tự động xử lý việc bảo trì và cập nhật cho dịch vụ.
Thời gian khởi động	Thời gian khởi động có thể lâu hơn, tùy thuộc vào loại instance.	Thời gian khởi động nhanh chóng cho các container.
Chi phí cố định	Có thể có chi phí cố định cao nếu sử dụng phiên bản Reserved Instances.	Không có chi phí cố định, người dùng chỉ trả cho tài nguyên thực tế sử dụng.
Tính linh hoạt	Linh hoạt trong việc chọn loại instance, hệ điều hành, và cấu hình.	Tính linh hoạt trong việc chạy nhiều container mà không cần quản lý server.
Quản lý ứng dụng	Cần quản lý các ứng dụng và môi trường trên instance riêng lẻ.	Quản lý ứng dụng dễ dàng hơn thông qua các service và task.
Chạy ứng dụng không liên tục	Có thể tốn chi phí khi chạy các instance không sử dụng.	Chỉ tính phí cho thời gian chạy, giúp tiết kiệm chi phí cho các ứng dụng không liên tục.

3.3 Kết luận Chương III

Vì sao nên chọn AWS cloudwatch

Tóm lại, AWS CloudWatch là một công cụ thiết yếu cho việc giám sát và quản lý các ứng dụng và tài nguyên trên AWS. Với khả năng theo dõi hiệu suất, thu thập log, thiết lập cảnh báo và tự động hóa phản hồi, CloudWatch giúp các tổ chức duy trì tính ổn định và hiệu suất của hệ thống, đồng thời cung cấp cái nhìn sâu sắc về hoạt động của ứng dụng. Việc tích hợp CloudWatch với các dịch vụ AWS khác cũng tạo ra một hệ sinh thái mạnh mẽ, hỗ trợ cho việc phát triển và triển khai ứng dụng một cách hiệu quả.

Lợi ích của AWS EC2

- **Kiểm soát cao**: Người dùng có toàn quyền kiểm soát hệ thống, bao gồm việc cài đặt phần mềm và cấu hình môi trường.
- **Tùy chọn đa dạng**: Cung cấp nhiều loại instance và tùy chọn cấu hình để phù hợp với nhu cầu cụ thể.
- **Tích hợp với nhiều dịch vụ khác**: Dễ dàng tích hợp với các dịch vụ khác của AWS.

Hạn chế của AWS EC2

- **Chi phí cao hơn cho tài nguyên không sử dụng**: Nếu không quản lý tốt, có thể dẫn đến chi phí cao do các instance không cần thiết vẫn đang chạy.
- **Cần quản lý phức tạp hơn**: Đòi hỏi kỹ năng và kiến thức để quản lý các instance, bảo trì và cập nhật.

Lợi ích của AWS ECS Fargate

- **Tự động hóa quản lý**: Không cần phải quản lý server, giúp giảm bớt khối lượng công việc cho nhóm phát triển.
- **Chi phí hiệu quả**: Chỉ trả tiền cho tài nguyên thực tế sử dụng, giúp tiết kiệm chi phí cho các ứng dụng không chạy liên tục.
- **Khả năng mở rộng dễ dàng**: Tự động mở rộng và thu nhỏ theo nhu cầu, giúp ứng dụng luôn đáp ứng đủ lưu lượng truy cập.

Hạn chế của AWS ECS Fargate

- **Chi phí cao cho khối lượng lớn**: Nếu ứng dụng cần nhiều tài nguyên, chi phí có thể tăng nhanh chóng, nhất là trong các tình huống tải cao.
- **Ít kiểm soát hơn**: Người dùng có ít quyền kiểm soát hơn đối với cấu hình server và môi trường so với EC2.

CHƯƠNG KẾT LUẬN

1 Kết quả đạt được

Sau khi triển khai quy trình CI/CD trên GitHub Actions cho hệ thống sử dụng Nginx làm reverse proxy và Node.js làm backend, với việc triển khai trên AWS sử dụng ECR (Elastic Container Registry) và ECS Fargate, chúng tôi đã đạt được một số kết quả đáng khích lệ. Đầu tiên, quy trình triển khai trở nên tự động hóa hoàn toàn, giúp giảm thiểu thời gian và công sức cần thiết cho việc đưa mã nguồn từ kho GitHub lên môi trường sản xuất. Mỗi khi có thay đổi trong mã, GitHub Actions tự động chạy các bài kiểm tra, xây dựng hình ảnh Docker và đẩy lên ECR, từ đó tự động cập nhật dịch vụ trên ECS Fargate.

Điều này không chỉ giúp tăng tốc độ phát triển mà còn cải thiện đáng kể tính nhất quán và độ tin cậy trong quy trình triển khai. Các lỗi do thao tác thủ công được giảm thiểu, và các phiên bản ứng dụng mới có thể được phát hành nhanh chóng hơn, đáp ứng kịp thời các yêu cầu từ phía người dùng. Ngoài ra, việc sử dụng Nginx làm reverse proxy đã cải thiện hiệu suất của ứng dụng, với khả năng xử lý hàng triệu kết nối đồng thời và cung cấp tính năng cân bằng tải, giúp tối ưu hóa tài nguyên và nâng cao khả năng phục vụ.

Hệ thống cũng đã chứng minh khả năng mở rộng tốt khi sử dụng ECS Fargate. Việc tự động mở rộng các container dựa trên lưu lượng giúp đảm bảo rằng ứng dụng duy trì hiệu suất ổn định ngay cả trong những thời điểm cao điểm. Tích hợp với các dịch vụ AWS khác như RDS cho cơ sở dữ liệu và S3 cho lưu trữ tệp đã tạo ra một kiến trúc mạnh mẽ và linh hoạt, cho phép chúng tôi dễ dàng mở rộng và nâng cấp hệ thống trong tương lai.

2 Hạn chế của hệ thống

Mặc dù đã đạt được nhiều kết quả tích cực, hệ thống vẫn gặp một số hạn chế cần được xem xét. Một trong những vấn đề chính là chi phí. Việc sử dụng AWS ECS Fargate và ECR có thể dẫn đến chi phí cao nếu không được quản lý chặt chẽ, đặc biệt là trong các môi trường có lưu lượng truy cập lớn. Người dùng cần phải theo dõi và điều chỉnh cấu hình tài nguyên để tối ưu hóa chi phí, điều này có thể đòi hỏi thêm công sức và thời gian.

Hạn chế thứ hai là độ phức tạp trong việc quản lý và giám sát. Mặc dù GitHub Actions mang lại tính tự động hóa, nhưng khi hệ thống ngày càng phát triển với nhiều dịch vụ liên quan, việc theo dõi và quản lý các pipeline CI/CD có thể trở nên phức tạp. Các lỗi trong quy trình tự động hóa hoặc các vấn đề về tích hợp giữa các dịch vụ có thể mất nhiều thời gian để khắc phục, gây ảnh hưởng đến hiệu suất tổng thể của hệ thống.

Cuối cùng, việc triển khai Nginx làm reverse proxy cũng có thể tạo ra một điểm yếu tiềm ẩn trong bảo mật. Nếu không được cấu hình chính xác, Nginx có thể trở thành mục tiêu cho các cuộc tấn công từ bên ngoài, làm lộ thông tin về ứng dụng backend. Do đó, việc đảm bảo an ninh cho hạ tầng, từ cấu hình Nginx cho đến các chính sách bảo mật trên AWS, là điều kiện tiên quyết để duy trì tính an toàn cho hệ thống.

3 Định hướng phát triển hệ thống

Để nâng cao hiệu suất và khả năng mở rộng của hệ thống trong tương lai, chúng tôi có một số định hướng phát triển rõ ràng. Trước tiên, việc tối ưu hóa chi phí là một ưu tiên hàng đầu. Chúng tôi sẽ tiếp tục theo dõi mức sử dụng tài nguyên và điều chỉnh cấu hình trên AWS để đảm bảo rằng chúng tôi chỉ sử dụng những gì cần thiết, từ đó giảm thiểu chi phí vận hành. Việc áp dụng các công cụ như AWS Cost Explorer có thể giúp chúng tôi phân tích và dự đoán chi phí một cách hiệu quả hơn.

Thứ hai, chúng tôi sẽ tập trung vào việc cải thiện khả năng giám sát và quản lý hệ thống. Việc triển khai các công cụ giám sát như AWS CloudWatch và Prometheus sẽ giúp theo dõi hiệu suất của ứng dụng và phát hiện sớm các vấn đề. Điều này không chỉ giúp cải thiện độ tin cậy của hệ thống mà còn tạo ra các báo cáo phân tích sâu về hiệu suất ứng dụng, từ đó đưa ra các quyết định phát triển hợp lý hơn.

Cuối cùng, hướng tới việc cải thiện bảo mật là một phần không thể thiếu trong định hướng phát triển. Chúng tôi sẽ xem xét các giải pháp bảo mật bổ sung như sử dụng AWS WAF (Web Application Firewall) để bảo vệ Nginx khỏi các cuộc tấn công mạng. Đồng thời, việc thường xuyên cập nhật và bảo trì các thành phần của hệ thống sẽ giúp giảm thiểu các rủi ro bảo mật.

Tóm lại, với những kết quả đã đạt được, các hạn chế cần khắc phục và định hướng phát triển rõ ràng, chúng tôi tin tưởng rằng hệ thống sẽ tiếp tục phát triển bền vững, đáp ứng tốt hơn các nhu cầu của người dùng và mở rộng quy mô một cách linh hoạt trong tương lai.

DANH MỤC TÀI LIỆU THAM KHẢO

Tài liệu, giáo trình:

[1] Priscila Heller - Automating Workflows with GitHub Actions_ Automate software development workflows and seamlessly deploy your applications using GitHub Actions-Packt Publishing (2021)

[2]Chaminda-Chandrasekara_-Pushpa-Herath-Hands-on-GitHub-Actions_-Implement-CI_CD-with-GitHub-Action-Wo

Trang web:

[1] https://github.com/

[2] https://registry.terraform.io/

[3] https://medium.com/@mudasirhaji/how-to-configure-nginx-as-a-reverse-proxy-on-aws-ec2-instance-270736ca2a50

[4] https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/How-to-setup-an-Nginx-load-balancer-example

[5] https://www.youtube.com/watch?v=nQdyiK7-VlQ