

ĐẠI HỌC QUỐC GIA HÀ NỘI

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

KỸ THUẬT TERRAFORM REGISTRY AWS CHO BÀI TOÁN XÂY DỰNG, TRIỂN KHAI HỆ THỐNG WEB SERVER

**ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC HỆ CHÍNH QUY**

**Ngành: Công Nghệ Thông Tin**

**HÀ NỘI – 2022**

ĐẠI HỌC QUỐC GIA HÀ NỘI

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

**Nguyễn Hồng Thái**

KỸ THUẬT TERRAFORM REGISTRY AWS CHO BÀI TOÁN XÂY DỰNG, TRIỂN KHAI HỆ THỐNG WEB SERVER

**ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC HỆ CHÍNH QUY**

**Ngành: Công Nghệ Thông Tin**

**Giảng viên hướng dẫn**: TS. Phạm Mạnh Linh

**HÀ NỘI, 2022**

# ****LỜI CẢM ƠN****

Trong quá trình làm khóa luận tốt nghiệp, tôi đã nhận được rất nhiều sự giúp đỡ, đóng góp ý kiến và dạy bảo nhiệt tình từ thầy cô trong và ngoài nhà trường, bạn bè và gia đình.

Đầu tiên, tôi xin gửi lời cảm ơn chân thành nhất tới TS. Phạm Mạnh Linh - người đã tận tình chỉ bảo, hỗ trợ và động viên giúp đỡ tôi trong việc nghiên cứu và hoàn thành dự án công nghệ này.

Tiếp theo, tôi cũng xin chân thành cảm ơn các Thầy, Cô giảng viên giảng dạy tại trường Đại học Công Nghệ - Đại học Quốc gia Hà Nội đã dạy dỗ và giúp đỡ em trong suốt quá trình học tập và nghiên cứu.

Cuối cùng, tôi xin cảm ơn đến những người thân trong gia đình và bạn bè đã luôn quan tâm, động viên tôi trong suốt quá trình học tập và hoàn thành đồ án này.

Tôi xin chân thành cảm ơn!

Hà Nội, tháng 08 năm 2022

Sinh viên

Nguyễn Hồng Thái

# ****LỜI CAM ĐOAN****

Tôi xin cam đoan tất cả các kết quả được trình bày trong khóa luận này là công trình nghiên cứu của chính bản thân tôi thực hiện dưới sự hướng dẫn của TS. Phạm Mạnh Linh.

Tất cả những bài báo tham khảo, nghiên cứu liên quan đều được nêu nguồn gốc rõ ràng. Trong khóa luận này, không có sự sao chép tài liệu, công trình nghiên cứu của những người khác mà không chỉ rõ về tài liệu đã tham khảo.

Nếu có điều gì sai trái, tôi xin chịu mọi trách nhiệm trước hội đồng về kết quả thực hiện khóa luận tốt nghiệp của mình.

Hà Nội, tháng 08 năm 2022

Sinh viên

Nguyễn Hồng Thái

Mục lục

[LỜI CẢM ƠN 1](#_Toc109654749)

[LỜI CAM ĐOAN 2](#_Toc109654750)

[Bảng thuật ngữ và từ viết tắt 6](#_Toc109654751)

[Danh sách hình ảnh và bảng biểu 7](#_Toc109654752)

[Tóm tắt 9](#_Toc109654753)

[Mở đầu 10](#_Toc109654754)

[Chương 1: Giới thiệu 12](#_Toc109654755)

[1. Tổng quan về Công nghệ thông tin 12](#_Toc109654756)

[2. Ngành devops và giới thiệu dự án 13](#_Toc109654757)

[2.1. Tổng quan về ngành devops 13](#_Toc109654758)

[2.2. DevOps Concepts và những Practices 15](#_Toc109654759)

[3. Tổng quan về dự án laravel: 18](#_Toc109654760)

[3.1 Giới thiệu, mô tả dự án 18](#_Toc109654761)

[Chương 2: Tổng quan về Amazon Web Services, Infrastructure as code và Web server. 20](#_Toc109654762)

[1. Amazon web services và các dịch vụ 20](#_Toc109654763)

[1.1. Giới thiệu chung 20](#_Toc109654764)

[1.2. EC2, Autoscaling group, Load balancing: 21](#_Toc109654765)

[1.3. AWS S3 (AWS Simple Cloud Storage) 24](#_Toc109654766)

[1.4. AWS CodeDeploy 25](#_Toc109654767)

[1.5. RDS 26](#_Toc109654768)

[2. Các dịch vụ sử dụng trong webserver: 27](#_Toc109654769)

[2.1. Webserver và nginx 27](#_Toc109654770)

[2.2. PHP và laravel 28](#_Toc109654771)

[2.3. Cở sở dữ liệu và Mysql 29](#_Toc109654772)

[3. Infrastructure as code – Terraform: 30](#_Toc109654773)

[3.1. Giới thiệu chung 30](#_Toc109654774)

[3.2. Terraform 30](#_Toc109654775)

[4. CI/CD Bitbucket 38](#_Toc109654776)

[Chương 3: Xây dựng mô hình cơ sở hạ tầng 40](#_Toc109654777)

[1. Mô hình cơ sở hạ tầng 40](#_Toc109654778)

[2. Tiêu chí mô hình 40](#_Toc109654779)

[3. Thiết kế mô hình 41](#_Toc109654780)

[3.1 Bản thiết kế 41](#_Toc109654781)

[3.2. Đáp ứng so với tiêu chí 44](#_Toc109654782)

[4. Cách thức hoạt động của mô hình 45](#_Toc109654783)

[Chương 4: Triển khai mô hình 46](#_Toc109654784)

[1. Môi trường phát triển: 46](#_Toc109654785)

[2. Môi trường thực nghiệm: 46](#_Toc109654786)

[3. Đối tượng và phạm vi nghiên cứu 46](#_Toc109654787)

[4. Phương pháp nghiên cứu 47](#_Toc109654788)

[5. Nội dung nghiên cứu 47](#_Toc109654789)

[6. Dữ liệu thực nghiệm: 47](#_Toc109654790)

[7. Quy trình triển khai 47](#_Toc109654791)

[7.1. Tạo server AMI 47](#_Toc109654792)

[7.2. Tạo VPC theo mô hình đã thiết kế 52](#_Toc109654793)

[7.3. Các dịch vụ khởi chạy webserver 56](#_Toc109654794)

[7.4. Cấu hình CI/CD cho website 65](#_Toc109654795)

[7.5. Cấu hình Codedeploy, bitbucket runner và repository variables 67](#_Toc109654796)

[8. Kết quả thực nghiệm 72](#_Toc109654797)

[8.1. Triển khai trên môi trường thực tế 72](#_Toc109654798)

[8.2. Kết quả triển khai 73](#_Toc109654799)

[5.3. Kiểm thử 74](#_Toc109654800)

[Kết luận 79](#_Toc109654801)

[1. Kết quả hiện tại 79](#_Toc109654802)

[1.1. Ưu điểm 79](#_Toc109654803)

[1.2. Nhược điểm 79](#_Toc109654804)

[2. Hướng phát triển sau này 79](#_Toc109654805)

[Tài liệu tham khảo 81](#_Toc109654806)

# Bảng thuật ngữ và từ viết tắt

**AWS** (Amazon web services): Nhà cung cấp dịch vụ Amazon

**Reverse proxy:** Máy chủ trung gian quản lý các yêu cầu của máy khách

**Rollback**: Trả về phiên bản cũ

**Script**: Đoạn mã

**Folder:** Thư mục

**Model**: Mô hình chạy mô phỏng

**Tool**: Công cụ

**Background**: Giao diện

**IDE**: môi trường tích hợp dùng để viết code

**Release:** Giải phóng

**Infra** (Infrastructure): Cơ sở hạ tầng

**Artifact:** tạo tác

**Private subnet:** Mạng con riêng tư

**Public subnet:** Mạng con công cộng

**Coupon:** Phiếu mua hàng

**Cloud:** Đám mây

**File:** Tập tin, tệp

**Code:** Dòng mã

**Bucket:** Kho lưu trữ

**Source code:** Chương trình được tạo ra

# Danh sách hình ảnh

[Hình 1.1 Mô hình làm việc truyền thống-Traditional Silos 12](#_Toc109748601)

[Hình 1.2 Hình minh họa AWS EC2 18](#_Toc109748602)

[Hình 1.3 Tổng quan Autoscaling group 20](#_Toc109748603)

[Hình 1.4 Amazon S3 bucket 21](#_Toc109748604)

[Hình 1.5 Hình minh họa AWS CodeDeploy 22](#_Toc109748605)

[Hình 1.6 Amazone RDS 23](#_Toc109748606)

[Hình 1. 7 Ví dụ về các file “.tf” trong terraform 28](#_Toc109748607)

[Hình 1.8 Terraform init cập nhật các version 29](#_Toc109748608)

[Hình 1.9 Terraform plan hiển thị các thay đổi 30](#_Toc109748609)

[Hình 1.10 Terraform apply cập nhật các thay đổi môi trường 31](#_Toc109748610)

[Hình 1.11 Apply thành công, hiển thị các output 31](#_Toc109748611)

[Hình 1.12 Hoàn thành destroy, xóa toàn bộ tài nguyên 32](#_Toc109748612)

[Hình 1. 13 Cấu trúc folder khởi chạy 33](#_Toc109748613)

[Hình 1.14 Cấu trúc folder module 34](#_Toc109748614)

[Hình 1.15 Bitbucket-codedeploy flow 36](#_Toc109748615)

[Hình 2.1 Mô hình hạ tầng xây dựng 38](#_Toc109748616)

[Hình 3.1 Cấu hình của máy tính sử dụng 46](#_Toc109748621)

[Hình 3.2 Cấu trúc module VPC 52](#_Toc109748622)

[Hình 3.3 Cấu trúc module webserver-cluster 57](#_Toc109748623)

[Hình 3.4 Hệ thống tệp của dự án 65](#_Toc109748624)

[Hình 3.5 Cấu trúc folder devopsBuild-server.sh: Cài đặt các gói còn thiếu (chủ yếu đã được cài trên AMI) 67](#_Toc109748625)

[Hình 3.6 Tạo repository 70](#_Toc109748626)

[Hình 3.7 Pipelines cho repo 71](#_Toc109748627)

[Hình 3.8 Thông số cho repossitory variables 71](#_Toc109748628)

[Hình 3.9 Quá trình pipelines thành công 72](#_Toc109748629)

[Hình 3.10 Flow codedeploy gọi dữ liệu 73](#_Toc109748630)

[Hình 3.11 Quá trình triển khai 73](#_Toc109748631)

[Hình 3.12 Các sourcode lưu trên s3 74](#_Toc109748632)

[Hình 3.13 Giao diện của website 74](#_Toc109748633)

[Hình 3.14 Cập nhật lên bitbucket thành công 75](#_Toc109748634)

[Hình 3.15 Codedeploy cập nhật thành công 75](#_Toc109748635)

[Hình 3.16 Số điện thoại đã được thay đổi 76](#_Toc109748636)

[Hình 3.17 Cơ sở dữ liệu được quản lý trên máy cá nhân 76](#_Toc109748637)

[Hình 3.18 Lệnh giả lập cpu lên 100% 77](#_Toc109748638)

[Hình 3.19 Cloudwatch thông báo tạo thêm instance khi CPU trên 85% 77](#_Toc109748639)

[Hình 3.20 Tạo thêm instance 77](#_Toc109748640)

[Hình 3.21 Cloudwatch thông báo xóa bớt instance khi CPU dưới 30% 78](#_Toc109748641)

[Hình 3.22 Instance đang được xóa 78](#_Toc109748642)

# Tóm tắt

Sử dụng ngôn ngữ terraform xây dựng hệ thống webserver trên AWS, sau đó triển khai CI/CD bằng Bitbucket với các chức năng sau:

+ Tự động mở rộng.

+ Triển khai tự động website.

+ Bảo mật cho website.

Tóm tắt nội dung các chương:

Mở đầu: Giới thiệu một cách tổng quan về ngành công nghệ thông tin và devops. Đồng thời giới thiệu dự án laravel sử dụng cho việc thử nghiệm sản phẩm.

Chương 1: Tổng quan về Devops, Amazon Web Services, Infrastructure as code và Web server.

Nội dung chương này trình bày lý thuyết về các dịch vụ trong AWS và các công nghệ sử dụng trong đồ án.

Chương 2: Xây dựng mô hình cơ sở hạ tầng.

Nội dung chương này diễn giải mô hình được xây dựng, tiêu chí của mô hình và cách thức hoạt động của mô hình.

Chương 3: Triển khai mô hình.

Nội dung chương này nói về quy trình triển khai, các kết quả đạt được sau khi triển khai và các kiểm thử sau khi triển khai.

Kết luận: Nêu các ưu nhược điểm và hướng phát triển sau này.

# Mở đầu

Tính đến tháng 12/2021, dân số Việt Nam đạt mốc 98 triệu người, với tỷ lệ dân thành thị gần 38%. Trong đó, có khoảng 68.3 triệu người đang sử dụng Internet hàng ngày (chiếm 70.3% dân số) thông qua các nền tảng, ứng dụng khác nhau, với thời lượng trung bình là 6 giờ 50 phút. Đây được xem là khoảng thời gian cực kỳ lớn được sử dụng trong 1 ngày. Tính đa dạng của những website cũng ngày càng phải được phát triển để phù hợp với nhu cầu ngày càng cao của con người. Cùng với sự bùng nổ của Internet và công nghệ 4.0, con người đag tiến tới việc xử lý được nhiều công việc với tốc độ nhanh hơn và chi phí thấp hơn so với làm việc truyền thống.

Những vấn đề kể trên đã thúc đẩy và phát sinh ra nhiều công việc liên quan đến ngành công nghệ thông tin, có tốc độ phát triển nhanh và mạnh mẽ, đóng góp to lớn vào sự tăng trưởng GDP của Việt Nam trong nhiều năm qua. Ngành công nghệ thông tin ở Việt Nam đã có những chuyển biến tích cực hơn khi triển khai ứng dụng công nghệ thông tin trong đời sống, xã hội người dân và doanh nghiệp. Một trong những ngành mới thuộc công nghệ thông tin và có tương lai phát triển rất tốt đó là Devops. Ngành Devops được phát triển như một xu hướng thiết yếu, Devops là một văn hóa làm việc kết hợp giữa kỹ sư phát triển phần mềm (developer) với kỹ sư hệ thống, kỹ sư mạng, kỹ sư hạ tầng, nhân viên bảo mật,...(operator) nhằm mục đích có thể rút ngắn vòng đời trong việc phát triển sản phẩm. Các kỹ sư devops là những người phụ trách phần thiết kế và triển khai hạ tầng đưa website đến tay người dùng và tạo ra những trải nghiệm tốt nhất.

Việc hiểu biết về cloud và các dịch vụ là 1 phần bắt buộc với các kỹ sư devops. Công nghệ đang dần phát triển và cloud hướng tới việc đưa dữ liệu lên đám mây, giảm đi lượng máy chủ vật lý cho các doanh nghiệp. Trên thế giới và ngay tại Việt Nam các tổ chức, doanh nghiệp đều đang dần chuyển sang dùng các dịch vụ về cloud. Nền tảng cung cấp các dịch vụ cloud đang được phổ biến nhất là AWS (Amazon web services).

Về công việc của các kỹ sư Devops, quá trình triển khai một hệ thống từ khâu lắng nghe mong muốn của khách hàng, lập trình ra sản phẩm, và triển khai được website đó nên môi trường mạng để khách hàng có thể truy cập đòi hỏi trình độ chuyên môn và sự nghiêm túc. Đội ngũ phát triển sẽ chia ra 5 quá trình: thu thập phân tích yêu cầu, thiết kế, tiến hành xây dựng, kiểm thử, triển khai, bảo trì. Những công việc này có thể được thực hiện thủ công, tuy nhiên quá trình làm việc thủ công có thể dẫn đến những lỗi không mong muốn. Vì vậy, trong dự án lần này tôi đã tiếp nhận yêu cầu và tìm ra một cách thức triển khai hạ tầng đang dần trở nên phổ biến đó là Infrastructure as code, tự động hóa quy trình triển khai.

Trong dự án lần này, tôi nhận được yêu cầu từ khách hàng - một doanh nghiệp kinh doanh mặt hàng nhu yếu phẩm ở Hà Nội: Chúng tôi cần thiết kế 1 hệ thống có chi phí triển khai thấp đáp ứng cho 1 doanh nghiệp vừa và nhỏ, có thể dễ dàng nâng cấp hệ thống trong trường hợp công việc kinh doanh phát triển thuận lợi, có chi phí duy trì rành mạch, dễ dàng thanh toán. Về phía yêu cầu từ đội ngũ phát triển website, tôi cần thiết kế một hệ thống dễ dàng triển khai các tính năng mới lên môi trường, có các môi trường độc lập dành cho việc kiểm thử và môi trường thực tế cho khách hàng. Tốc độ triển khai nhanh, hỗ trợ giao diện quản lý cho người dùng.

Với các yêu cầu nêu trên, tôi quyết định sẽ chọn đề tài cho đồ án là "Kỹ Thuật Terraform Registry AWS cho bài toán xây dựng, triển khai hệ thống webserver" xây dựng một hệ thống sử dựng kỹ thuật Terraform để triển khai website dựa trên nền tảng cloud, cụ thể là AWS (Amazon Web Services ), kèm theo tính năng tự động tích hợp, tự động triển khai qua Bitbucket. Đề tài giải quyết vấn đề phát sinh lỗi trong quá trình triển khai ứng dụng, quản lý các dịch vụ bằng code và tình trạng phải xây dựng lại một hệ thống từ đầu khi chuyển sang dự án khác.Những kết quả dự kiến của dự án:

* Xây dựng thành công mô hình webserver với Terraform.
* Mô hình sẽ tiết kiệm thời gian cho việc triển khai webite.
* Mô hình tăng cường bảo mật hơn cho website.
* Các module đã xây dựng có thể tiếp tục sử dụng cho các dự án sau.

***Từ khoá: Công nghệ thông tin, Triển khai tự động, Devops, Infrastructure as Code, Terraform cho AWS , Amazon Web Services.***

# Chương 1: TỔNG QUAN VỀ NGÀNH DEVOPS, AMAZON WEB SERVICES, INFRASTRUCTURE AS CODE VÀ WEB SERVER

## 1. Ngành devops

### 1.1. Tổng quan về ngành devops

#### 1.1.1. Giới thiệu về ngành devops

##### 1.1.1.1. Khái niệm

DevOps là viết tắt của Development (Dev) và Operations (Ops). DevOps là một văn hóa làm việc kết hợp giữa kỹ sư phát triển phần mềm (dev) với operator (kỹ sư hệ thống, nhân viên bảo mật, kỹ sư mạng, kỹ sư hạ tầng,...) nhằm mục đích có thể rút ngắn vòng đời trong việc phát triển sản phẩm (SDLC).

DevOps không phải là một công cụ hỗ trợ làm việc, mà nó là một phương pháp làm việc cần sử dụng đến sự hỗ trợ của các công cụ. Ngoài ra, không có bất cứ một tiêu chuẩn nào để những những người sử dụng DevOps phải tuân theo. Nó chỉ đơn giản là một văn hóa, một cách làm việc kết hợp giữa dev và nhân viên operations thay vì làm việc một mình hay chỉ một team độc lập nào đó.

Những điều cần biết để áp dụng trong DevOps, bao gồm:

DevOps Culture: văn hóa làm việc kết hợp giữa Dev và Ops.

DevOps Practices: cách để thực hiện DevOps.

DevOps Tools: những Tools cần để thực hiện DevOps.

DevOps và Cloud: mối quan hệ giữa DevOps và cloud.

##### 1.1.1.2. Văn hóa devops

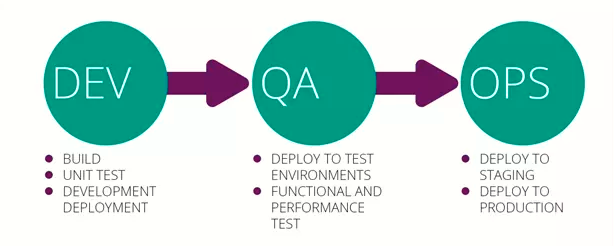
Với cách làm việc truyền thống trước đây, nhóm dev và nhóm operations làm việc riêng lẻ, mỗi nhóm sẽ có mục tiêu công việc khác nhau. Mục tiêu của nhóm dev là xây dựng thuật toán đúng với yêu cầu và cập nhật sản phẩm cho khách hàng một cách nhanh chóng nhất thì mục tiêu của operations là tạo ra, vận hành một hệ thống ổn định. Trong trường hợp xảy ra sự không đồng nhất về mục tiêu, thuật toán có thể chạy tốt nhưng khi cập nhật sản phẩm sẽ làm cho hệ thống chạy chậm hoặc có thể xuất hiện nhiều lỗi. Hoặc một sản phẩm đầu tư nhiều về quản trị hệ thống sẽ làm chậm thời gian phải phát hành phiên bản mới. Vì thế, DevOps được sinh ra với ý tưởng là kết hợp giữa lập trình viên và kỹ sư vận hành thống nhất một mục tiêu chung là vừa nhanh chóng vừa ổn định.

***Những mục tiêu đạt đến cụ thể 1 team DevOps như sau:***

* Fast time - to market (TTM) : code nhanh, triển khai nhanh.
* Few production failures : Sản phẩm được đảm bảo tính thống nhất và ổn định nên trong quá trình hoạt động không xảy ra nhiều lỗi.
* Immediate recovery from failures: Nếu xảy ra sự cố và lỗi, những thành viên trong nhóm do có cả dev và ops nên có thể nhận định nguyên nhân lỗi là do code của dev hay là do hệ thống từ đó giúp giải quyết vấn đề gặp phải một cách nhanh chóng.

Trong văn hóa DevOps, hai nhóm làm việc cùng nhau để tối ưu hóa cả năng suất của các nhà phát triển và độ tin cậy của hoạt động. Các thành viên phải cố gắng liên lạc thường xuyên, tăng hiệu quả và cải thiện chất lượng dịch vụ cung cấp cho khách hàng. Thành viên nhóm devops có quyền và vai trò lớn hơn theo chức danh truyền thống bằng cách suy nghĩ về nhu cầu của khách hàng cuối và cách họ có thể đóng góp để giải quyết những nhu cầu đó. Các nhóm đảm bảo chất lượng và bảo mật cũng có thể trở nên tích hợp chặt chẽ hơn.

So sánh mô hình DevOps với mô hình làm việc truyền thống - Traditional Silos, có thể thấy như sau:



Hình 1.1 Mô hình làm việc truyền thống-Traditional Silos

Ở mô hình Traditional Silos, dev viết code đưa sang QA test. QA phát hiện ra bug sẽ đưa cho dev fix, rồi chuyển lại cho QA để test. Quá trình này sẽ lặp đi lặp lại nhiều lần cho đến khi sản phẩm không còn xảy ra lỗi, sản phẩm được chuyển giao cho operation để triển khai code lên môi trường. Lúc này, QA sẽ kiểm thử lại lần nữa. Do dev và operations làm việc tách biệt nên việc phát sinh ra lỗi sẽ mất thời gian sửa chữa vì cần QA xác định xảy ra do code hay do hệ thống.

Tuy nhiên, với mô hình DevOps: Cả 3 đơn vị dev,ops và QA đã trở thành nhóm thống nhất cùng hoàn thiện một dự án, họ sẽ sử dụng các công cụ đóng gói giúp chạy trên nhiều môi trường như như Jenkins hoặc docker. Điều đó tạo ra một hệ thống hoàn toàn tự động từ khi build code, kiểm thử đến khi triển khai. Nếu xảy ra lỗi họ có thể nhanh chóng hoàn trả lại phiên bản cũ, cùng nhau tìm hiểu nguyên nhân, sửa lỗi rồi triển khai lại code mới.

#### 1.1.2. Thực trạng ngành devops trong những năm trở lại đây:

Giờ đây, Devops đã khẳng định được vị thế riêng của mình. Trong giới công nghệ thông tin hiện nay, các kỹ sư Devops luôn đứng top đầu về lương. Và về nhu cầu thị trường, các công ty, doanh nghiệp sử dụng hướng tiếp cận Devops ngày càng tăng cao nên công việc này đang và sẽ trở thành một xu hướng mới của các kỹ sư công nghệ thông tin.

Puppetlabs – một tổ chức nghiên cứu, khảo sát về các ngành nghề vừa xuất bản một báo cáo chỉ ra: Các công ty, tổ chức sử dụng Devops trong quá trình triển khai có số lượt triển khai nhiều hơn gấp 30 lần so với các công ty không dùng và số lượng lỗi xảy ra trong quá trình triển khai giảm đi tới hơn 50%.

Trên thế giới, chỉ trong 2 năm các công việc có hướng tiếp cận với DevOps trên trang tuyển dụng Indeed.com đã tăng đến 74%, trên LinkedIn.com, số lượng này đã tăng 50%. Cũng theo Puppetlabs, trong một cuộc khảo sát gần đây, với 4000 nhân viên tại các công ty trên hơn 90 quốc gia được khảo sát cho biết rằng công ty nơi họ làm việc quan tâm nhiều đến kỹ năng Devops với các ứng viên xin việc.

### 1.2. Ý tưởng ngành devops

#### 1.2.1. Build Automation

Build Automation - Tự động hóa xây dựng là quá trình tự động hóa việc truy xuất mã nguồn, biên dịch nó thành mã nhị phân, thực hiện các bài kiểm thử tự động và xuất bản nó vào một kho lưu trữ tập trung, được chia sẻ. Tự động hóa xây dựng là rất quan trọng để các quy trình DevOps thành công. Thông thường việc build automation cũng giống như chạy một công cụ hoặc lệnh điều khiển để chạy đoạn code đã được viết hoặc được cài đặt trong tập tin cấu hình. Các lập trình viên có thể tiến hành build code bất cứ khi nào mà không cần lo lắng đến cấu hình của máy tính cũng như không phụ thuộc vào các IDE từ đó thuận tiện cho việc làm việc từ xa với khách hàng.

Với build automation một hệ thống tự động hóa xây dựng được tổ chức hợp lý mang lại nhiều lợi ích. Kiểm thử tự động là một trong những thử nghiệm lớn nhất đối với nhiều dự án lớn. Hệ thống xây dựng nâng cao có thể chạy các bài kiểm tra đơn vị và nhiều bài kiểm tra khác như một phần của quy trình chuẩn, sớm phát hiện các lỗi phổ biến. Một số hệ thống xây dựng thậm chí có thể chạy các bài kiểm tra nâng cao mô phỏng hành vi của người dùng.

Một lợi ích chính khác của build automation là chúng đơn giản hóa việc phân phối dự án trong nhóm. Nếu không có hệ thống xây dựng, mọi thành viên trong nhóm sẽ phải tải xuống phiên bản mới nhất của cơ sở mã và tự biên dịch nó để xem bất kỳ thay đổi nào gần đây.

#### 1.2.2. Continuous Intergration

Tích hợp liên tục (Continuous Intergration – CI) là một phương pháp đòi hỏi các lập trình viên phải thường xuyên hợp nhất code thay đổi. Với cách làm việc truyền thống, các lập trình viên trong các nhóm không làm việc cùng với nhau nên họ thường đặt ra một khoảng thời gian cụ thể để đưa code lên. Tuy nhiên, với CI thì lập trình viên phải hợp nhất code của họ mỗi ngày và sẽ chạy kiểm thử tự động để phát hiện những vấn đề khi hợp nhất code. CI cũng có thể được tự động hóa, thông thường việc tự động hóa được hỗ trợ bởi một CI Server. Khi lập trình viên cập nhật những thay đổi về code lên môi trường, CI Server sẽ ngay lập tức thấy được sự thay đổi này và bắt đầu thực hiện build, kiểm thử source code thay đổi một cách tự động. Quá trình trên có thể được lặp lại nhiều lần trong một ngày, từ đó kịp thời thông báo đến lập trình viên khi xảy ra sự cố.

Trường hợp có một thành viên khác triển khai phần code của họ lên môi trường và xảy ra lỗi trong quá trình build thì hệ thống sẽ thông báo lỗi cho người đó để tiến hành sửa lỗi, đồng thời sẽ cập nhật lại phiên bản cũ, không làm ảnh hưởng đến thành viên khác.

Những lợi ích mà CI mang lại: Đầu tiên, CI sẽ giúp phát hiện ra lỗi sớm, thông báo lại cho lập trình viên. Họ có thể sửa lỗi gặp phải ngay lập tức hoặc trả về phiên bản cũ để không làm ảnh hưởng đến thành viên khác. Tiếp theo, áp dụng CI sẽ giúp tránh việc phải hợp nhất một số lượng lớn code khi phiên bản mới đến khách hàng. Thay vào đó mỗi ngày code sẽ được hợp nhất một cách tự động. Cùng với đó, nhờ việc code được hợp nhất mỗi ngày nên người sử dụng có thể nhận bản cập nhật trong cả giai đoạn làm việc chứ không cần chờ đến khi tất cả mọi thứ đã hoàn thành mới có thể nhận được bản cập nhật. Việc kiểm thử cũng được diễn ra trong suốt cả quá trình hoàn thành dự án, đảm bảo độ hoàn thiện sản phẩm là tốt nhất. CI sẽ tạo ra thói quen làm việc khoa học cho các lập trình viên, thường xuyên cập nhật code sẽ làm họ viết ra những đoạn code đúng chuẩn và đơn giản.

#### 1.2.3. Continuous Delivery

Continuous Delivery (CD) là phương thức giúp đảm bảo code luôn luôn sẵn sàng cho quá trình triển khai. Thay vì quyết định có nên triển khai code hay không thì nhóm phải build, hợp nhất, kiểm thử,… để đảm bảo code luôn ở trạng thái có thể triển khai.

Trong dự án thực tế Continuous Delivery được thực hiện theo luồng sau:

Mỗi phiên bản code được triển khai đều trải qua một quy trình giống nhau, bao gồm: build tự động, kiểm thử tự động,... Nếu không phát sinh thêm vấn đề thì có nghĩa là source code đó sẵn sàng để triển khai.

Sau khi quyết định deploy thì người dùng sẽ thực hiện triển khai tự động (deploy automation).

Nếu việc triển khai gặp lỗi, hệ thống sẽ tự động rollback về phiên bản gần nhất.

Nhà phát triển sẽ nhanh chóng sửa lỗi và triển khai lại những phiên bản đã sửa lỗi.

**Với cách làm này, Continuous Delivery sẽ đem lại những lợi ích sau:**

* Hoàn thiện, bàn giao sản phẩm cho khách hàng nhanh chóng.
* Giảm thiếu các vấn đề, lỗi xảy ra khi triển khai.
* Giảm thiếu rủi ro: lượng triển khai trong 1 lần càng nhiều, rủi ro càng cao. Việc chia nhỏ số lượng lần triển khai sẽ giảm thiểu rủi ro.
* Rollback lập tức mỗi khi xảy ra lỗi hoặc sư cố. Giúp lập trình viên không còn phải lo lắng khi triển khai vì đã có chức năng tự động rollback.

#### 1.2.4. Infrastructure as Code (IaC)

**Khái niệm**: Infrastructure as Code(IaC) là phương thức giúp xây dựng, quản lý cơ sở hạ tầng (Infra) cho ứng dụng một cách thuận tiện và khoa học. Với Infrastructure as Code, thay vì thực hiện công việc theo các thủ công. Người dùng có thể tự viết code và cho chạy tự động.

Hiện nay, việc quản lý các web applications khá là phức tạp, ví dụ việc phải quản lý cài web servers, load balancer, database servers, ... Với việc sử dụng Infrastructure as Code các kỹ sư có thể rút ngắn rất nhiều thời gian để xây dựng và quản lý những hệ thống này.

Cơ sở hạ tầng xây dựng các đối tượng: Máy chủ, Instance, môi trường, Container,...

**Những ưu điểm của IaC**

*Chuẩn mực*: IaC tạo và quản lý resource theo các thông số tiêu chuẩn. Tất cả mọi dịch vụ được thực hiện bằng code, không có những sai sót như việc cấu hình bằng tay.

*Tính linh hoạt*: Với IaC, các thay đổi giống nhau có thể thực hiện được trên nhiều máy chủ, code cũng có thể lưu lại sử dụng trong tương lai.

*Tính co dãn (Scalability)* : Khi cần tạo thêm máy chủ người dùng không cần tạo lại mà chỉ cần sử dụng cấu hình có sẵn và tạo một instance mới cực kỳ nhanh chóng.

*Self-documenting*: Để xem những cấu hình của hệ thống với IaC thì chỉ cần xem trong source code tại máy tính của lập trình viên, người dùng không cần phải lưu lại lịch sử hoặc tạo bất kỳ một tài liệu nào để lưu giữ thông tin cấu hình. Chính bản thân source code dùng để xây dựng hạ tầng chính là tài liệu lưu trữ.

## 2. Amazon web services và các dịch vụ

### 2.1. Giới thiệu chung

Amazon Web Services (AWS) là nền tảng điện toán đám mây toàn diện và được sử dụng tương đối rộng rãi hiện nay, cung cấp trên 200 dịch vụ đầy đủ mọi tính năng, mỗi loại có thể được cấu hình theo những cách khác nhau dựa trên nhu cầu của người dùng. Người dùng sẽ có thể xem các tùy chọn cấu hình và bản đồ máy chủ riêng lẻ cho một dịch vụ AWS.

AWS ra mắt vào năm 2006 từ cơ sở hạ tầng nội bộ mà Amavailability zoneon.com đã xây dựng để xử lý các hoạt động bán lẻ trực tuyến của mình. AWS là một trong những công ty đầu tiên giới thiệu mô hình điện toán đám mây trả tiền theo quy mô để cung cấp cho người dùng khả năng tính toán, lưu trữ hoặc thông lượng khi cần thiết.

AWS có 84 Vùng sẵn sàng tại 26 khu vực địa lý trên khắp thế giới và đã thông báo về kế hoạch tăng thêm 24 Vùng sẵn sàng và thêm 8 Khu vực AWS khác tại Úc, Canada, Ấn Độ, Israel, New Zealand, Tây Ban Nha, Thụy Sĩ và các Tiểu Vương quốc Ả Rập Thống nhất (UAE). Gartner đã công nhận mô hình Khu vực AWS và Vùng sẵn sàng là phương pháp được khuyến nghị để chạy các ứng dụng doanh nghiệp cần độ khả dụng cao.

AWS có cộng đồng lớn và linh hoạt nhất, hàng triệu khách hàng - hầu hết các ngành công nghiệp thuộc mọi quy mô, bao gồm công ty khởi nghiệp, doanh nghiệp và tổ chức công, đang chạy mọi trường hợp sử dụng có thể diễn ra trên AWS. Mạng lưới đối tác AWS (APN) bao gồm hàng nghìn đơn vị tích hợp hệ thống chuyên về dịch vụ AWS và hàng chục nghìn nhà cung cấp phần mềm độc lập (ISV) điều chỉnh công nghệ của mình để hoạt động trên AWS.

AWS có nhiều dịch vụ hơn và nhiều tính năng mới nổi bật trong các dịch vụ đó, hơn bất kỳ nhà cung cấp đám mây nào khác - từ các công nghệ cơ sở hạ tầng như máy tính, ổ lưu trữ và cơ sở dữ liệu đến các công nghệ mới như trí tuệ nhân tạo và machine learning, kho dữ liệu và phân tích, và Internet of Things. Điều này giúp người sử dụng chuyển các ứng dụng hiện có lên đám mây nhanh hơn, dễ dàng hơn và hiệu quả hơn về chi phí cũng như xây dựng gần như mọi thứ cho nganh công nghệ thông tin đặc biệt là devops.

### 2.2. EC2, Autoscaling group, Load balancing:



Hình 1.2 Hình minh họa AWS EC2

#### 2.2.1. Giới thiệu EC2

Amazon Elastic Compute Cloud (Amazon EC2) là một cơ sở hạ tầng điện toán đám mây được cung cấp bởi Amazon Web Services (AWS) giúp cung cấp tài nguyên máy tính ảo hoá theo yêu cầu.

Amazon EC2 cung cấp rất nhiều các phiên bản hệ điều hành có thể tự động mở rộng và quản lý các thành phần phần cứng ảo như bộ nhớ máy tính (ram), vi xử lý, phân vùng một cách linh hoạt trên môi trường ảo hóa. Nhờ kiến trúc ảo hoá đám mây của Aamzon EC2 luôn đảm bảo sự an toàn trong việc việc lưu trữ dữ liệu và quản lý các dịch vụ.

Amazon EC2 có thể cung cấp một hoặc nhiều máy chủ ảo kết hợp với nhau trở thành một cụm máy chủ để đảm bảo tính sẵn sàng cao nhất và dễ dàng triển khai ứng dụng một cách nhanh nhất. Các mức chi phí cần thanh toán được liệt kê rõ ràng và có thông báo cho phí dựa trên thông tin tài nguyên đã được sử dụng.

**Đặc điểm:**

***- Khả năng mở rộng (scaling):***

+ Scaling In/Out: Tăng hoặc giảm số lượng EC2 Instance.

+ Scaling Up/Down: Tăng/Giảm cấu hình (RAM, CPU,...) của Instance.

***- Độ bảo mật (security):***

+ Có thể cấu hình dải mạng dành riêng cho EC2.

+ Sử dụng Security Group và Network ACLS để thiết lập các quyền truy cập nội bộ và công khai (inbound/outbound) .

+ Có thể thiết lập IPsec VPN giữa Data Center và AWS Cloud.

+ Thậm chí có thể tạo EC2 trên một máy chủ vật lý sử dụng cho một khách hàng duy nhất.

***- Chi phí (cost):***

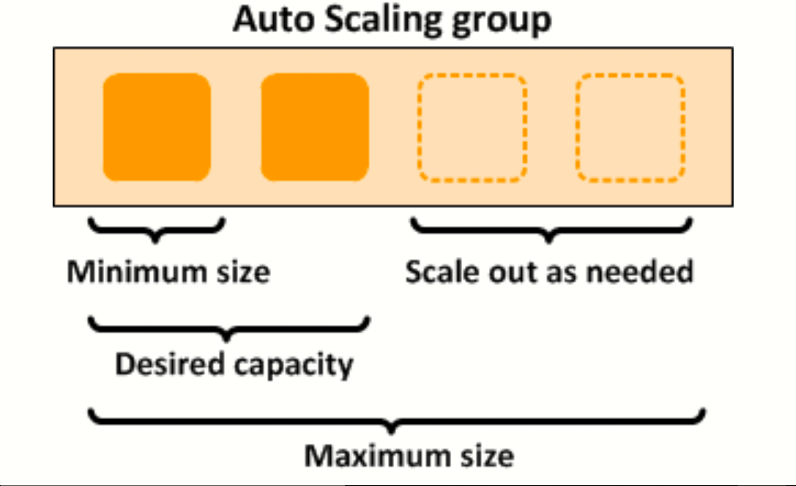
+ Tuỳ thuộc vào nhu cầu: Được khuyến mãi 750 giờ sử dụng với các máy có cấu hình thực hành thí nghiệm, đáp ứng nhu cầu dùng trong thời gian ngắn, trả tiền theo thời lượng sử dụng. Khá linh hoạt và có thể dễ dàng thanh toán qua tài khoản visa.

+ Khuyến mại trả trước: Có thể trả phí trước cho các máy chủ.

+ Nếu không cần sử dụng máy chủ trong một thời gian dài, người dùng có thể bán lại, thường là trên Reserved Instance Marketplace.

#### 2.2.2. Autoscaling group

AWS Auto Scaling là tính năng tự động mở rộng để đảm bảo rằng các phiên bản EC2 đủ cấu hình để chạy các ứng dụng. Người dùng có thể tạo nhóm AWS Auto Scaling trong các EC2 instance. Người dùng có thể chỉ định số lượng phiên bản EC2 tối thiểu trong nhóm đó và tự động mở rộng sẽ duy trì và đảm bảo số lượng phiên bản EC2 tối thiểu. Người dùng cũng có thể chỉ định số lượng phiên bản EC2 tối đa trong AWS Auto Scaling đảm bảo các instance không bao giờ được phép vượt quá giới hạn tối đa đó. Cũng có thể chỉ định các chính sách dung lượng và tự động mở rộng mong muốn cho phần AWS Auto Scailing trong Amazon EC2. Với việc cấu hình chính sách tự động mở rộng, AWS Auto Scaling có thể khởi tạo hoặc chấm dứt các EC2 instance tùy theo nhu cầu.



Hình 1.3 Tổng quan Autoscaling group

Trong quá trình sử dụng các dịch vụ đám mây, việc các công ty, tổ chức đang sử dụng hệ thống điện toán đám mây và bỗng dưng gặp vấn đề như không đồng bộ được dữ liệu hay web bị quá tải, hệ thống ngừng hoạt động là điều không thể tránh khỏi. Chính vì vậy, tính năng Auto Scaling của Amazon Web Services đã được sinh ra để giải quyết vấn đề này.

#### 2.2.3. Auto loadbalancing

Load Balancing là tính năng cân bằng tải, một trong những tính năng rất quan trọng với những nhà phát triển, lập trình mạng. Khi máy chủ ngừng hoạt động hoặc không thể xử lý, một Load Balancer sẽ được bổ sung. Người dùng truy cập vào load balancer, tiếp tục được chuyển đến một máy chủ khác để thực hiện tác vụ. Dù máy chủ chính bị ngừng hoạt động hoặc xảy ra tắc nghẽn thì tất cả các yêu cầu của người dùng đều có thể được giải quyết.

### 2.3. AWS S3 (AWS Simple Cloud Storage)

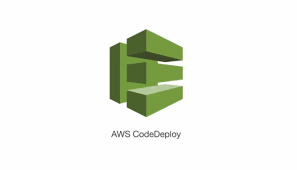


Hình 1.4 Amazon S3 bucket

Là bộ lưu trữ dữ liệu và các đối tượng có thể mở rộng an toàn, bảo mật và khả năng có thể mở rộng cao trong môi trường đám mây. Amazon S3 được sử dụng để lưu trữ và truy xuất bất kỳ số lượng dữ liệu bất kỳ lúc nào và ở đâu, giúp người dùng dễ dàng lưu trữ dữ liệu càng nhiều càng tốt và truy cập bằng bất cứ khi nào họ cần. Điều này cho phép các công ty, doanh nghiệp tránh việc phải mua phần cứng và phải chi trả chi phí cho phần bộ nhớ không được sử dụng. Theo AWS, dữ liệu cần được trên website bởi vì người dùng chỉ cần trả chi phí cho bộ nhớ và băng thông mà họ đã thực sự sử dụng. Amazon S3 thường được các công ty, doanh nghiệp sử dụng để sao lưu trữ dữ liệu cho ứng dụng, chạy các ứng dụng, lưu trữ các trang web có lưu lượng truy cập lớn, sao lưu dữ liệu trang web đều đặn theo thời gian. Nó cũng có thể cho phép người dùng ghi lại những thay đổi và rollback lại phiên bản cũ hơn trước đó.

Amazon S3 dễ dàng sử dụng và có nhiều cách truy cập, quản lý khác nhau. S3 cung cấp các API và dịch vụ web như REST và SOAP để lưu trữ, quản lý và truy xuất dữ liệu.Người sử dụng có thể điều khiển thông qua truy cập vào giao diện điều khiển trên Amazon. Dữ liệu trong Amazon S3 được lưu trữ dưới dạng những đối tượng, các đối tượng được lưu trữ trong thư mục, thư mục lúc này được gọi là bucket. Bucket là nơi chứa dữ liệu, mỗi dự án có thể sử dụng một hoặc nhiều bucket. Đối với mỗi bucket người dùng có thể điều khiển truy cập như tạo, hiển thị và xóa các đối tượng của bucket. Để lưu trữ Object trong Amazon S3, người dùng trước tiên cần đăng nhập vào trang aws.amazon.com với tài khoản đã đăng kí và tải lên dữ liệu đến bucket được chỉ định.

### 2.4. AWS CodeDeploy

****

Hình 1.5 Hình minh họa AWS CodeDeploy

AWS CodeDeploy là dịch vụ triển khai được quản lý toàn phần được tích hợp trên AWS có khả năng tự động hóa việc triển khai phần mềm trên các dịch vụ điện toán đám mây như Amazon EC2, AWS Fargate, máy chủ Lambda và các máy chủ chạy tại chỗ. AWS CodeDeploy giúp người dùng dễ dàng triển khai các tính năng mới một cách nhanh chóng hơn, tránh thời gian ngừng hoạt động của ứng dụng trong quá trình triển khai ứng và giúp cập nhật ứng dụng một cách đơn giản hơn.

Người dùng có thể sử dụng AWS CodeDeploy để tự động hóa việc triển khai, cập nhật phần mềm, không cần phải thực hiện các tác vụ thủ công dễ bị xảy ra lỗi. Dịch vụ này có thể thay đổi quy mô để phù hợp với yêu cầu triển khai.

***Lợi ích của Code Deploy***

- AWS CodeDeploy tự động hóa hoàn toàn việc triển khai ứng dụng của khách hàng, cho phép người sử dụng triển khai ứng dụng một cách tin cậy và nhanh chóng.

- Giúp tối đa hóa khả năng sử dụng ứng dụng trong quá trình triển khai.

- AWS CodeDeploy cho phép người dùng dễ dàng khởi chạy và theo dõi trạng thái của quá trình triển khai ứng dụng qua bảng điều khiển quản lý AWS hay AWS CLI.

- Có khả năng tương thích chung về mặt ngôn ngữ và nền tảng, hoạt động tốt với bất kỳ nền tảng, ứng dụng nào và mang lại trải nghiệm giống nhau.

### 2.5. RDS



Hình 1.6 Amazone RDS

Dịch vụ cơ sở dữ liệu quan hệ của Amazon (Amazon RDS) là một tập hợp các dịch vụ được quản lý giúp người sử dụng dễ dàng thiết lập, vận hành và thay đổi quy mô cho cơ sở dữ liệu trên đám mây. Chọn từ các công cụ phổ biến MySQL, Amazon Aurora, PostgreSQL, MariaDB, PostgreSQL, Oracle cùng khả năng tương thích với và SQL Server cũng như triển khai tại chỗ với Amazon RDS on AWS Outposts.

**Lợi ích của RDS**

**Giảm nhẹ gánh nặng quản trị**

* Đơn giản, dễ sử dụng.
* Tự động vá lỗi cho ứng dụng.

**Khả năng thay đổi quy mô**

* Thay đổi quy mô điện toán chỉ bằng một vài cấu hình đơn giản.
* Dễ dàng kiểm soát quy mô lưu trữ.

**Độ bền và độ khả dụng**

* Kết xuất cơ sở dữ liệu nhanh.
* Triển khai linh hoạt trên nhiều vùng sẵn sàng.
* Máy chủ lưu dữ liệu có thể được thay thế.

**Bảo mật**

* Mã hóa khi đang lưu trữ và khi đang truyền.
* Cách ly mạng.
* Quyền ở cấp độ tài nguyên.

**Dễ quản lý**

* Giám sát và thông số.
* Thông báo sự kiện.
* Quản lý cấu hình.

**Hiệu quả chi phí**

* Chỉ cần trả chi phí cho những gì sử dụng.
* Phiên bản dự trữ.
* Dừng và khởi động lại dễ dàng.

## 3. Các dịch vụ sử dụng trong webserver:

### 3.1. Webserver và nginx

#### 3.1.1. Giới thiệu webserver

Máy chủ web (Webserver) là máy chủ cài đặt các dịch vụ, chương trình phục vụ việc khởi chạy ứng dụng web. Webserver có khả năng nhận các yêu cầu từ trình duyệt và gửi phản hồi đến máy khách thông qua các giao thức HTTP, HTTPS… Có nhiều webserver khác nhau như: Apache, Nginx, IIS... Nginx và Apache là 2 webserver phổ biến nhất với các nhà phát triển phần mềm.

Quy trình hoạt động của webserver:

* Trình duyệt phân giải tên miền thành địa chỉ IP.
* Webserver gửi lại client Trang được yêu cầu.
* Trang web hiển thị trên trình duyệt.

#### 3.1.2. Nginx

Nginx là phần mềm mã nguồn mở để phục vụ web, reverse proxying, lưu vào bộ nhớ đệm, cân bằng tải và nhiều chức năng khác nữa. Ban đầu, nó là một máy chủ web được thiết kế để có hiệu suất và độ ổn định tối đa. Nginx khá là ổn định, cấu hình đơn giản và hiệu suất cao. Nginx phục vụ web, reverse proxying, caching, load balancing, media streaming... Nó bắt đầu như một máy chủ web được thiết kế để có hiệu suất và sự ổn định tối đa. Nginx có các tính năng như chứng thực người dùng, máy chủ ảo, hỗ trợ CGI, HTTPS, SCGI, FCGI, WCGI, SSI, ISAPI, Ipv6, …

Ưu điểm:

* Có khả năng xử lý rất nhiều kết nối cùng một lúc mà không cần bộ nhớ lớn.
* Phục vụ cho tập tin tĩnh và lập chỉ mục tập tin.
* Tăng tốc reverse proxy bằng bộ nhớ đệm, cân bằng tải đơn giản và khả năng chịu lỗi.
* Hỗ trợ tăng tốc với bộ nhớ đệm và các máy chủ lưu trữ.
* Tự động nén để tăng tốc độ trang.
* Hỗ trợ mã hoá TLS và SSL.
* Cấu hình hoạt động linh hoạt, hỗ trợ lưu lại nhật ký truy vấn.
* Chuyển hướng lỗi 3XX-5XX.
* Rewrite URL (URL rewriting) dùng regular expressions.
* Số lượng kết nối hoặc truy vấn từ một địa chỉ bị giới hạn.
* Khả năng nhúng mã PERL.
* Hỗ trợ với IPv6, WebSockets, hỗ trợ cả việc truyền tải file MP4 và FLV.

### 3.2. PHP và laravel

PHP (Hypertext Preprocessor) là một ngôn ngữ lập trình với nhiều mục đích khác nhau. Cụ thể hơn, PHP là ngôn ngữ kịch bản mã nguồn mở, chạy ở phía máy chủ và được dùng để tạo ra các ứng dụng web.

PHP tương đối phổ biến và được rất nhiều lập trình viên học tập, sử dụng. Cơ hội việc làm dành cho những người thành thạo ngôn ngữ PHP vô cùng rộng mở. Các ứng viên làm việc với ngôn ngữ này có thể trở thành một lập trình viên Web hoặc lập trình viên Full-Stack.

PHP phù hợp với những dự án có yêu cầu:

* Không yêu cầu những xử lí, tính toán phức tạp.
* Số lượng người dùng truy cập nhỏ hoặc trung bình, hoặc số lượng truy cập lớn, nhưng logic không quá phức tạp.
* Rất phù hợp cho các vấn đề liên quan đến giao diện Web.

Ngoài ra, vì một vài yếu điểm mà PHP không phù hợp với những dự án:

* Có yêu cầu thời gian thực với cực kỳ nhiều kết nối và yêu cầu thời gian xử lý cực nhanh.
* Yêu cầu logic phức tạp.
* Dữ liệu lớn cần xử lí.

#### 3.2.1. Framework Laravel

Laravel là một bộ khung (framework) của ngôn ngữ PHP, là mã nguồn mở và sử dụng miễn phí, được xây dựng nhằm hỗ trợ phát triển các phần mềm, ứng dụng. Hiện nay, Laravel đang là PHP framework tốt nhất và phổ biến nhất.

#### 3.2.2. Composer

Composer là một quản lý phụ thuộc trong PHP, quản lý các thư viện mà dự án Php của người sử dụng. Một cách chính xác hơn Composer quản lý sự phụ thuộc các tài nguyên trong dự án. Composer cho phép khai báo các thư viện mà dự án sử dụng, nó sẽ tự động tải code của những thư viện được liệt kê.

### 3.3. Cở sở dữ liệu và Mysql

Cơ sở dữ liệu là hệ thống bao gồm rất nhiều thông tin, dữ liệu được xây dựng theo một cấu trúc nhất định nhằm đáp ứng nhu cầu khai thác, sử dụng của nhiều người hay chạy nhiều chương trình ứng dụng cùng một lúc. Khi áp dụng hình thức lưu trữ này sẽ giúp khắc phục được những điểm yếu của việc lưu file thông thường trên máy tính. Các thông tin lưu trữ sẽ đảm bảo được nhất quán, hạn chế tình trạng trùng lặp thông tin.

MySQL là một hệ thống quản trị cơ sở dữ liệu mã nguồn mở (Relational Database Management System, viết tắt là RDBMS) hoạt động theo mô hình máy khách - máy chủ.

Ưu điểm:

* Dễ sử dụng: MySQL là cơ sở dữ liệu tốc độ cao, ổn định, dễ sử dụng và có thể hoạt động được trên nhiều hệ điều hành. Mysql cung cấp một thư viện lớn các hàm tiện ích rất mạnh.
* Độ bảo mật cao: MySQL rất phù hợp cho các hệ thống và ứng dụng có truy cập cơ sở dữ liệu trên Internet khi sở hữu nhiều nhiều tính năng bảo mật thậm chí là ở cấp cao.
* Đa tính năng: MySQL hỗ trợ rất nhiều chức năng SQL từ một hệ quản trị cơ sở dữ liệu quan hệ theo cách trực tiếp và gián tiếp.
* Khả năng mở rộng và mạnh mẽ: MySQL có thể xử lý được rất nhiều dữ liệu và có thể được mở rộng nếu cần thiết.
* Nhanh chóng: Việc đưa ra một số tiêu chuẩn cho phép MySQL làm việc rất hiệu quả, tăng tốc độ thực thi và giảm chi phí đi rất nhiều.

## 4. Infrastructure as code – Terraform:

### 4.1. Giới thiệu chung

Infrastructure as code (IaC) là cách thức xây dựng hạ tầng công nghệ thông tin ( IT ) bằng cách chạy các dòng mã (code) thay vì thực hiện một cách thủ công như truyền thống.

IaC được xem như là “immutable infrastructure”. Ưu điểm của việc này là việc kết hợp nay phân chia hạ tầng một cách linh hoạt chỉ cần thực hiện trên một máy local. Điều này mang đến sự chuẩn hóa và thống nhất cho việc xây dựng hạ tầng, tất các các thông số cấu hình, các bước thực hiện được thể hiện trên “source code” thay vì qua “trí nhớ” hay hay hệ thống tài liệu phức tạp. Việc triển khai trở nên nhanh chóng, linh hoạt và tiện lợi hơn.

Có rất nhiều công cụ và frameworks cho việc xây dựng hạ tầng trên Cloud. Tuy nhiên, Terraform là một công cụ khá phổ biến và mạnh mẽ được các công ty và tập đoàn lớn sử dụng tương đối nhiều. Vì vậy tôi sẽ sử dụng Terraform cho dự án này.

### 4.2. Terraform

#### 4.2.1. Giới thiệu

Terraform là công cụ có khả năng quản lý cơ sở hạ tầng dưới dạng mã Code của HashiCorp. Nó là một công cụ để xây dựng, thay đổi và quản lý cơ sở hạ tầng một cách an toàn. Các nhóm vận hành hệ thống và cơ sở hạ tầng có thể sử dụng Terraform để quản lý môi trường bằng ngôn ngữ được gọi là Ngôn ngữ cấu hình HashiCorp (HCL) để con người có thể đọc được, và tự động hóa triển khai.

#### 4.2.2. Lợi ích

**Hỗ trợ đa nền tảng**

Trong một trung tâm dữ liệu hiện đại, người dùng có thể sẽ sử dụng nhiều nền tảng Cloud để hỗ trợ cho các ứng dụng khác nhau của mình. Với Terraform, người dùng có thể dễ dàng quản lý các môi trường không đồng nhất, với cùng một quy trình làm việc, bằng cách tạo tệp (file) cấu hình để phù hợp với nhu cầu của dự án hoặc tổ chức.

**Quản lý trạng thái**

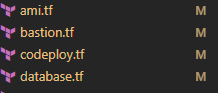
Terraform tạo ra một tệp trạng thái khi một dự án được khởi tạo lần đầu. Tệp trạng thái này được sử dụng để lập kế hoạch và thực hiện các thay đổi đối với cơ sở hạ tầng. Trước bất kỳ thay đổi nào trong tệp cấu hình, Terraform sẽ đối chiếu, so sánh lại và cập nhật các trạng thái tương ứng trên cơ sở hạ tầng thực.

**Vận hành tin cậy**

Quy trình làm việc được tích hợp trong Terraform nhằm mục đích tạo tin cậy cho người dùng, bằng cách làm cho các hoạt động có thể lặp lại dễ dàng và hỗ trợ phần lập kế hoạch giúp người dùng xác minh lại các tiến trình nào sẽ được áp dụng vào hệ thống của họ, đảm bảo các hành động được hoàn toàn thực hiện bởi Terraform.

#### 4.2.3. Cấu trúc

Cấu trúc hạ tầng với Terraform bao gồm các file có định dạng “.tf’’. Trong file là các resource để cấu hình các dịch vụ và khai báo biến. Lưu ý cú pháp, biến phải bắt đầu với ‘var’ và có giá trị mặc định.



Hình 1.7 Ví dụ về các file “.tf” trong terraform

variable “name” {

  type = “string”,

  default =“(Giá trị cho variable)”

}

output “developer”{

  value=“${var.name}”

}

Output trích xuất giá trị của service. Nếu muốn xuất ra thông số nào thì chỉ cần gọi đến resource theo cấu trúc:

output "(tên out put)" {

value = (tên resource).(tên khai báo).(giá trị cần trích xuất)

}

Giá trị cần trích xuất thường là name hoặc id của service.

Ví dụ: Trích xuất id của security\_group có tên sgroup-rds với tên sg\_rds\_id:

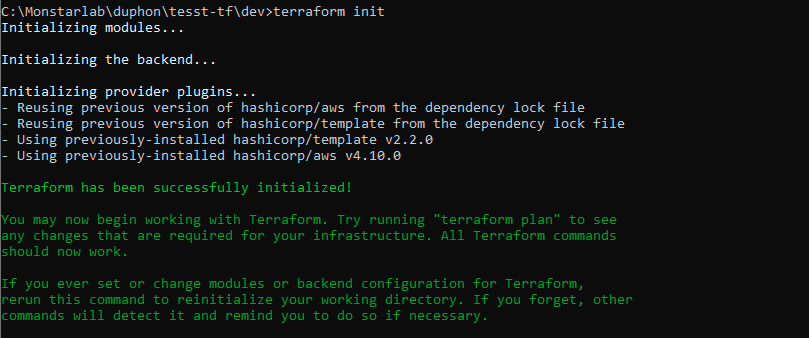
output "sg\_rds\_id" {

    value = aws\_security\_group.sgroup-rds.id

}

**Terraform init**

Trước khi chạy các file mã nguồn, cần phải thiết lập ban đầu cho Terraform. Terraform init được sử dụng để thiết lập cấu hình cho Terraform. Cấu trúc câu lệnh là ***terraform init***.



Hình 1.8 Terraform init cập nhật các version

**Terraform Plan**

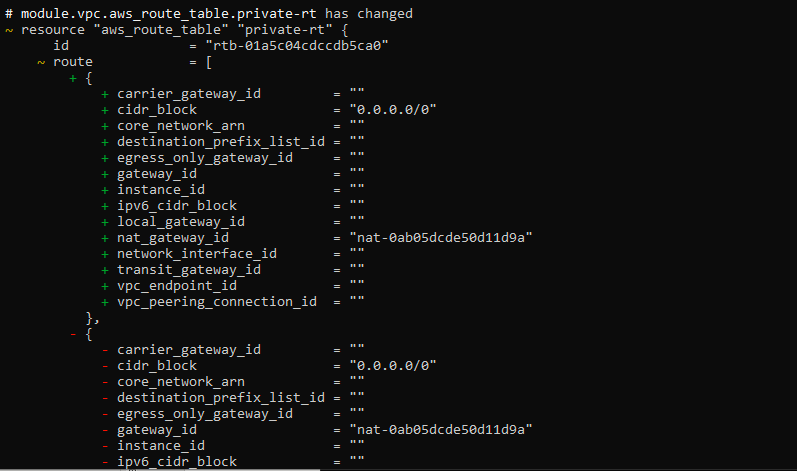
Để khởi chạy được các dịch vụ, trước tiên cần phải tạo một “plan”. Plan này sẽ chỉ ra các tài nguyên sẽ được tạo ra giúp rà soát kiểm tra toàn bộ các bước và tài nguyên sẽ sử dụng trước khi thực thi. Người dùng sẽ lưu ra một file ‘plan’ và sau đó apply file vào hạ tầng cần xây dựng hoặc có thể apply trực tiếp.

Ví dụ việc lưu file plan là first.terraform, sau đó để thực thi, người dùng chạy lệnh apply file first.terraform này:

terraform plan -out first.terraform

terraform apply first.terraform

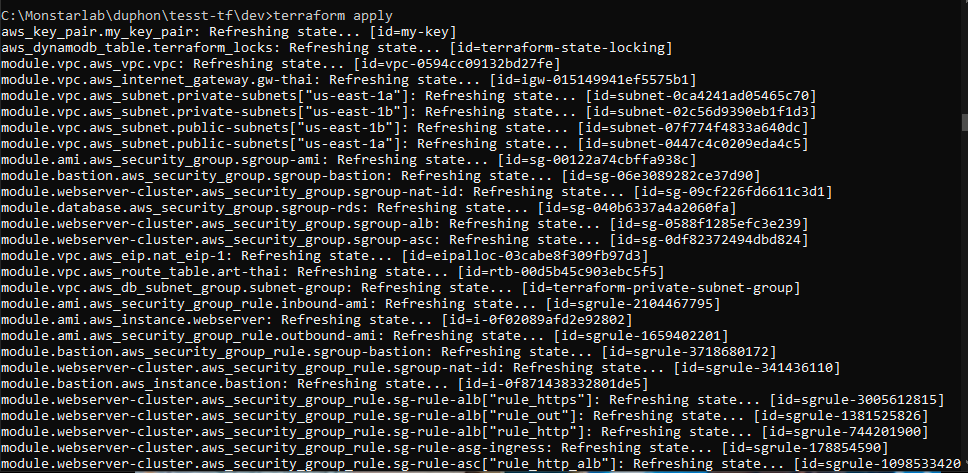
Các thay đổi được so sánh và hiển thị sau khi chạy chạy **terraform plan**



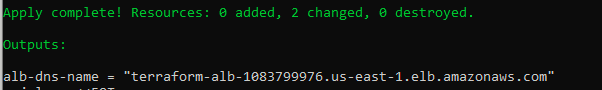
Hình 1.9 Terraform plan hiển thị các thay đổi

**Terraform apply**

Với lệnh “apply”, Terraform sẽ cập nhật các thay đổi trên môi trường đám mây cần xây dựng, và “output” trả về các thông tin về các tài nguyên đã được tạo.



Hình 1.10 Terraform apply cập nhật các thay đổi môi trường



Hình 1.11 Apply thành công, hiển thị các output

**Terraform Providers**

Providers có nhiệm vụ quản lý vòng đời của tài nguyên như tạo ra một tài nguyên mới, hoặc cập nhật nó. Có rất nhiêu nhà cung cấp dịch vụ cloud hỗ trợ Terraform, ví dụ: AWS, Google Cloud....

Ví dụ dưới đây là một mẫu cấu hình provider trên AWS. Người dùng có thể xuất ra AWS secret key id và access key id bằng biến môi trường.

provider “aws” {

   region = “us-east-1”

access\_key = "$(var.acc\_key)"

   secret\_key = "$(var.secret\_key)"

}

resource “aws\_instance” “my-vm-automated” {

   ami = “ami-a4c7edb2”

   instance\_type = “t3.micro”

   tags {

     Name = “my-vm-automated”

}

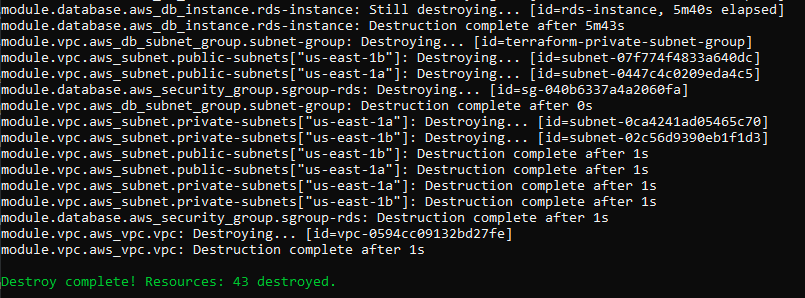
Ở đây người dùng sẽ tạo 1 EC2 instance loại t3.micro, và được gắn thẻ với tên ‘my-vm-automated’.

**Terraform State**

Khi người dùng cập nhật thay đổi, file ‘terraform.tfstate’ sẽ dược tạo ra. File này được sử dụng để lưu các cấu hình hiện tại trên hạ tầng và được sử dụng khi người dùng chạy các files tuần tự. Terraform sẽ biết thay đổi nào cần cập nhật. Người dùng nên commit file này lên kho lưu mã nguồn (repository), sẽ cần trạng thái của hạ tầng hiện tại khi thực thi files.

**Destroying Resources**

Người dùng có thể hủy toàn bộ tài nguyên đã được tạo ra với lệnh ***terraform destroy***.



Hình 1.12 Hoàn thành destroy, xóa toàn bộ tài nguyên

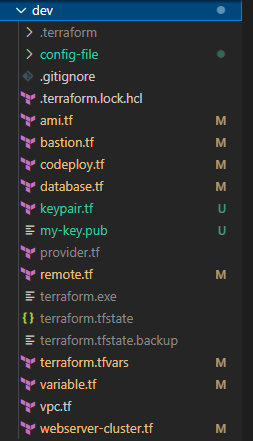
#### 4.2.4. Module trong Terraform

Module như đúng tên của nó, là việc người sử dụng gom các cấu hình lại và tái sử dụng ở nhiều nơi, nhiều lần.

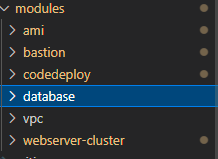
Với cách viết thông thường, các resource của terraform sẽ khai báo trực tiếp các biến ngay tại file cấu hình hoặc file variable.tf, điều này làm cho việc tái sử dụng code sẽ bị hạn chế bởi vì người dùng phải tìm đến từng resource và thay đổi cấu hình. Việc làm này gây mất không ít thời gian tìm kiếm và sửa chữa.

Với việc sử dụng module Terraform, người sử dụng chỉ cần tạo những file cấu hình tại modules và khai báo biến sử dụng mặc định. Tất cả các biến được thay đổi trên phần file triển khai gọi đến module, việc tìm kiếm sửa chữa thuận lợi hơn rất nhiều.

Module Terraform thường được chia làm 2 thư mục cho phần module và phần khai báo, khởi chạy:



Hình 1. 13 Cấu trúc folder khởi chạy



Hình 1.14 Cấu trúc folder module

Các biến được khai báo giá trị trong file ***terraform.tfvars*** (Phải đặt đúng tên file)

db-engine = "mysql"

db-engine-version = "8.0"

db-instance-type = "db.t2.micro"

db-name = "\*\*\*\*\*\*"

username = "\*\*\*\*\*"

password = "\*\*\*\*"

port = 3306

#VPC

cidr = "10.0.0.0/16"

public-all = {

    "us-east-1a" = 1

    "us-east-1b" = 2

}

private-all = {

    "us-east-1a" = 3

    "us-east-1b" = 4

}

#  Bastion

bastion-ami = "ami-0c4f7023847b90238"

type\_bastion = "t2.micro"

ingress-bastion = 0

egress-bastion = 0

key\_name = "beanstalk"

#  Webserver-cluster

ami-webserver = "ami-0356b1cd4aa0ee970"

type-webserver = "t2.micro"

min =2

max = 4

#  Webserver-cluster-alb

port-alb-access = 80

protocol-alb-access = "HTTP"

path-alb-healthcheck = "/menu.php"

port-alb-health = 80

protocol-alb-health = "HTTP"

action-alb-request = "forward"

#ami

ingress-ami = 0

egress-ami = 0

# Code deploy

# deployment-config-name = "CodeDeployDefault.OneAtATime"

#### 4.2.5. Quy trình với Terraform

Bước 1: Phạm vi – Xác nhận những tài nguyên nào cần được tạo cho một dự án nhất định.

Bước 2:Tạo file cấu hình – Tạo tệp cấu hình với HCL dựa trên các tham số trong phạm vi đã xác định trước đó.

Bước 3:Khởi tạo – Chạy terraform init trong thư mục dự án với các tệp cấu hình.

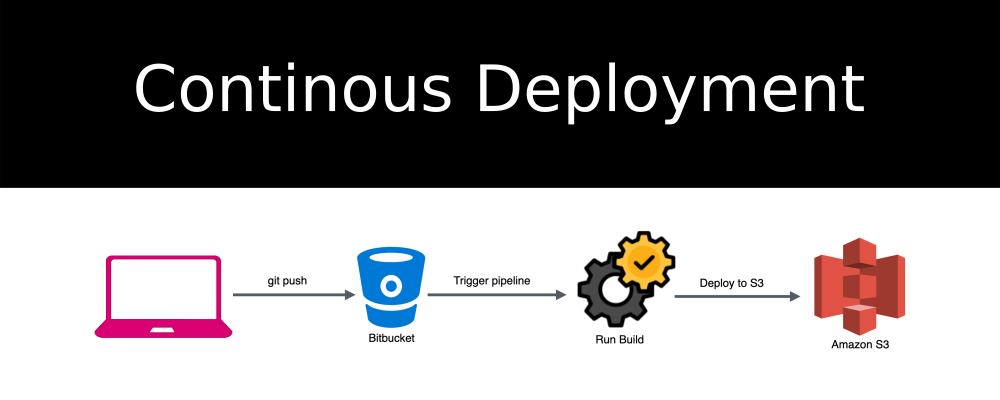
Bước 4: Lập kế hoạch & Áp dụng – Chạy terraform plan để kiểm tra, xác minh lại các quá trình sẽ được thực thi và sau đó sử dụng terraform apply để áp dụng các tài nguyên được cấu hình sẵn vào hệ thống thực.

## 5. CI/CD Bitbucket

CI/CD là một bộ đôi công việc, bao gồm Continuous Integration (CI) và Continuous Delivery (CD), là quá trình tích hợp thường xuyên, nhanh chóng hơn khi code cũng như thường xuyên cập nhật phiên bản mới.

Ở đây tôi sử dụng Bitbucket-pipelines phục vụ cho mục đích triển khai code từ môi trường máy tính cá nhân lên trên máy chủ. Bitbucket-pipelines có vai trò đóng gói project thành một artifact, di chuyển và lưu artifact đến aws S3 dưới dạng file nén. Và sau đó sẽ là nhiệm vụ của AWS codepipeline và AWS codedeploy.

**Flow Bitbucket pipeline**



Hình 1.15 Bitbucket-codedeploy flow

Khi lập trình viên hoàn thành một tính năng nào đó và cập nhật code lên hệ thống Bitbucket để mọi người kiểm tra và nhận xét. Khi đó Bitbucket CI cũng bắt đầu thực hiện công việc đã được cấu hình sẵn. Hệ thống Bitbucket pipelines sử dụng file .bitbucket-pipeline.yml nằm trong thư mục gốc của dự án để cấu hình project sử dụng các Runner. Một pipelines CI sinh ra và các thông báo và tiến độ sẽ được hiển thị trên giao diện. Pipeline thông thường sẽ bao gồm thực hiện kiểm thử từng phần, build lại ứng dụng, và triển khai lên máy chủ.

# Chương 2: XÂY DỰNG MÔ HÌNH CƠ SỞ HẠ TẦNG

## 1. Khái niệm

Mô hình cơ sở hạ tầng là việc nghiên cứu và kết hợp các dịch vụ trên môi trường điện toán đám mây thành một hệ thống hoàn chỉnh do các kỹ sư devops thiết kế. Mô hình này sẽ là bản mẫu giúp những thành viên trong nhóm có thể cùng đánh giá và việc xây dựng có quy chuẩn. Mỗi mô hình sẽ nhằm phục vụ cho mục đích khác nhau tùy thuộc vào yêu cầu của dự án hoặc của khách hàng.

Dựa trên mô hình đã thiết kế, các kỹ sư sẽ triển khai mô hình tạo ra một môi trường mạng riêng ảo nơi mà người quản trị có thể sử dụng để kết nối nội bộ giữa các service Cloud theo cách người quản trị phục vụ cho các mục tiêu bảo mật hệ thống. Người quản trị sẽ có toàn quyền quản lý môi trường hệ thống mạng riêng ảo bao gồm : tạo subnet, cấu hình bảng routing, gateway mạng, lựa chọn sử dụng IPv4 hay IPv6 trong VPC.

## 2. Tiêu chí mô hình

* Về bảo mật:

+ Quá trình xử lý dữ liệu cá nhân cần phải được bảo mật tốt.

+ Tính bảo mật cần phải được đảm bảo và cần có một thỏa thuận thích hợp cho việc đó để dữ liệu phải được duy trì an toàn.

+ Chống lại được các cuộc tấn công vừa và nhỏ.

+ Dữ liệu phải được cung cấp và phải thực hiện giám sát liên tục.

* Về tính liên tục:

+ Website phải đảm bảo thời gian máy chủ ngưng hoạt động mỗi lần cập nhật phiên bản được tối ưu.

+ Website đảm bảo tính duy trì và khả năng tự động mở rộng.

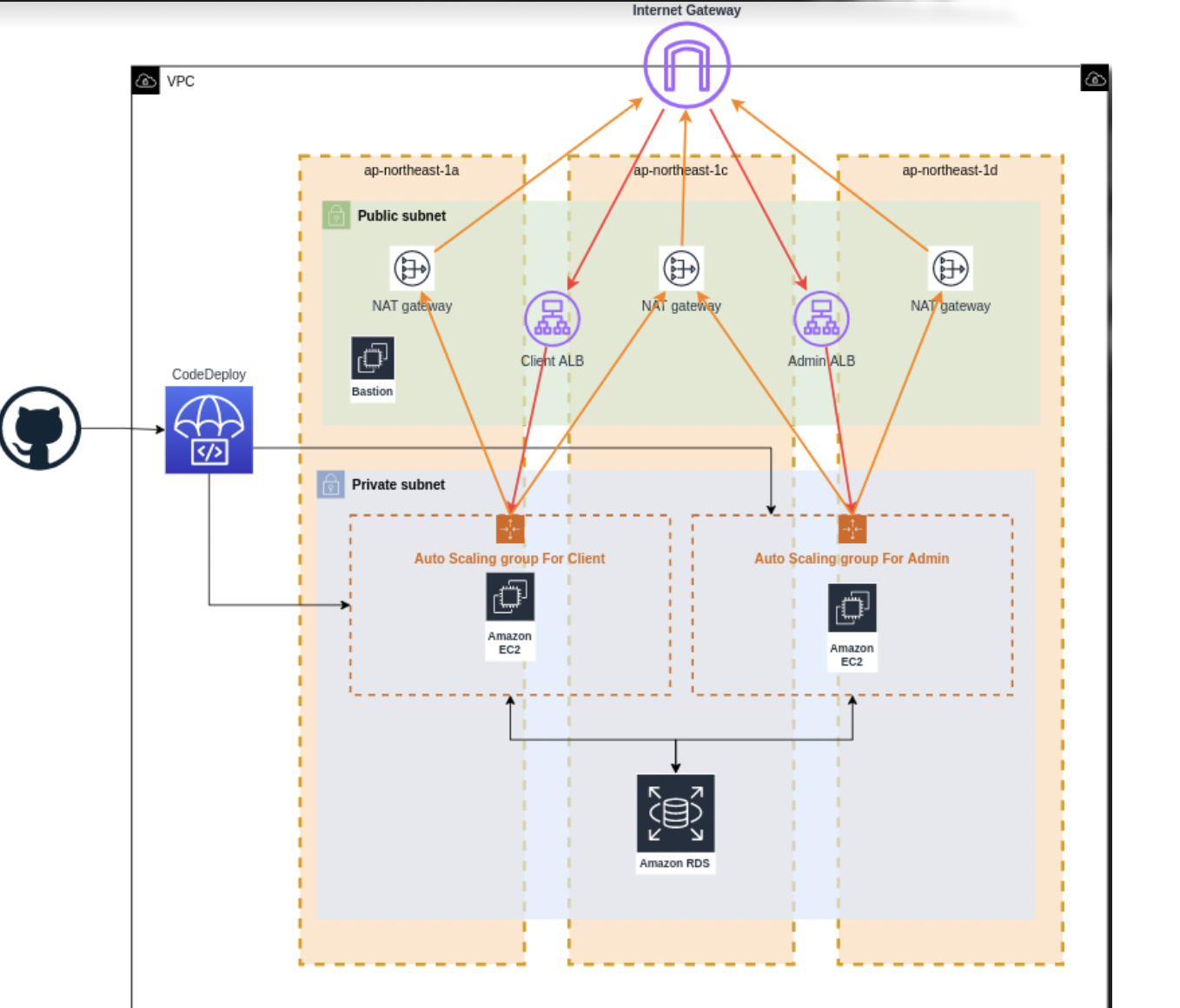
* Chi phí

+ Hệ thống đảm bảo chi phí bỏ ra đáp ứng được yêu cầu.

+ Các dịch vụ được quản lý thông tin chi phí.

## 3. Thiết kế mô hình

### 3.1 Bản thiết kế



Hình 2.1 Mô hình hạ tầng xây dựng

Mô hình được xây dựng nhằm đảm bảo tính bảo mật và tính sẵn sàng cao cho website với các máy chủ và các dịch vụ được đặt ở nhiều availability zone khác nhau trên cùng 1 khu vực. Khi máy chủ tại 1 vùng không hoạt động do sự cố hoặc khi update phiên bản thì lập truy cập đến máy chủ ở vùng khác giúp đảm bảo thời gian là máy chủ ngưng hoạt động là ngắn nhất.

Các dịch vụ cần truy cập ra ngoài internet và không cần bảo mật sẽ được đặt ở public subnet. Máy chủ website và cơ sở dữ liệu sẽ được đặt ở private subnet bởi vì không cần truy cập ra ngoài internet, đảm bảo tính bảo mật là tốt nhất.

Mô hình sử dụng source code của website laravel để demo và trình bày kết quả.

Mô hình và các dịch vụ sẽ được diễn giải như sau:

**+ Subnet:**

Các subnet được mô tả bằng các khung hình chữ nhật với 3 màu như trên hình:

. Public subnet được biểu thị bằng màu xanh lá nằm trên 3 khu vực.

. Màu tím biểu thị các private subnet cũng được nằm trên 3 khu vực.

. Mỗi khung chữ nhật màu cam biểu thị cho một vùng sẵn sàng (availability zone).

**+ Máy chủ EC2:**

. Số lượng : Linh động (từ 2 đến 4 phụ thuộc tài nguyên sử dụng)

. Availability zone: 3 availability zone, nằm trong autoscaling group

. Subnet: private

. Tác dụng, ý nghĩa: Máy chủ EC2 là nơi lưu trữ và cấu hình các dịch vụ để hiển thị được website. Không cần truy cập ra ngoài internet, cần được đảm bảo tính bảo mật là tốt nhất.

**+ Autoscaling group:**

. Số lượng: 2

. Availability zone: Mỗi autoscaling group đặt trên 2 availability zone

. Subnet: private

. Tác dụng, ý nghĩa: Mở rộng khi instance quá tải và xóa bớt instance khi không cần sử dụng giúp tối ưu được chi phí.

**+ ALB:**

. Số lượng: 2, mỗi ALB truy cập đến 1 autoscaling group

. Availability zone: Mỗi ALB được đặt trên 2 availability zone

. Subnet: public

. Tác dụng, ý nghĩa: Truy cập được vào máy chủ thông qua autoscaling group, việc cài đặt các thông số quy định quá trình scaling, url của website nếu không mua domain sẽ là url của ALB.

**+ Bastion:**

. Số lượng: 1

. Availability zone: 1

. Subnet: public

. Tác dụng, ý nghĩa: Bastion cũng là 1 máy EC2 nhưng nằm ở public subnet, giúp truy cập đến autoscaling group thông qua SSH. Điều khiển và quản lý EC2.

**+ Nat gatewway:**

. Số lượng: 3

. Availability zone: Mỗi Nat gateway nằm trên 1 Availability zone

. Subnet: public

. Tác dụng, ý nghĩa: Nat gateway giúp các máy chủ EC2 đi ra được ngoài internet để có thể cài được các dịch vụ mà không cần các EC2 phải cấu hình public IP. Giúp máy chủ trở nên bảo mật.

**+ RDS:**

. Số lượng: 1, cơ sở dữ liệu được dùng chung cho tất cả máy chủ.

. Availability zone: 3

. Subnet: private

. Tác dụng, ý nghĩa: Tất cả cơ sở dữ liệu được lưu trữ bảo mật, chỉ có EC2 truy cập đến.

**+ S3:**

. Là dịch vụ global, có thể gọi đến từ tất cả các khu vực.

. Số lượng: 1

. Availability zone: Global

. Subnet: Global

. Tác dụng, ý nghĩa: lưu trữ data của website dưới dạng nén.

**+ Bitbucket:**

Được biểu thị hình con mèo đen trong vòng tròn trỏ đến codedeploy.

### 3.2. Đáp ứng so với tiêu chí

#### 3.2.1. Tính bảo mật

* Tối ưu code và truy vấn nhằm hạn chế thời gian máy chủ ngưng hoạt động.
* Dữ liệu và cơ sở dữ liệu được đặt ở private subnet, không thể truy cập. Chỉ mở những cổng cần thiết cho website (HTTP: 80, HTTPS: 443, MySql :3306).
* Các cuộc tấn công khó xảy ra do không có địa chỉ IP public của webserver. Giới hạn các cổng được mở nhằm hạn chế các công cụ tấn công.
* Dữ liệu vẫn có thể đảm bảo được quản lý và giám sát thông qua việc truy cập vào máy chủ cho những quản trị viên nắm quyền kiểm soát (Thông qua 2 lớp key).

#### 3.2.2. Tính liên tục

* Tối ưu code và truy vấn nhằm hạn chế thời gian máy chủ ngưng hoạt động.
* Sử dụng AMI giúp không cần cài đặt lại dịch vụ mỗi lần cập nhật.
* Sử dụng autoscaling group và load balancer đảm bảo quá trình mở rộng, khi máy chủ quá tải ngay lập tức tăng thêm server khác giúp chia sẻ tài nguyên. Tránh việc treo máy.
* Có nhiều hơn 1 vùng cho máy chủ, mỗi khi máy chủ ở vùng này gặp sự cố, ngay lập tức có máy chủ khác hoạt động thay thế.

#### 3.2.3. Tối ưu chi phí

* Việc tối ưu chi phí dựa trên thời gian sử dụng dịch vụ và theo yêu cầu của khách hàng.
* Sử dụng auto scaling group giúp tối ưu thời gian sử dụng máy chủ, khi không có nhiều request đến thì tự động giảm số lượng máy chủ giúp tiết kiệm được chi phí.

## 4. Hoạt động của mô hình

Mô hình bao gồm 1 VPC và 6 subnet, bao gồm 3 public và 3 private subnet, mỗi subnet đặt trên 1 vùng khác nhau. Các subnet kết nối với nhau qua route table. VPC tạo ra 1 mạng riêng ảo, các máy chủ và dịch vụ khác kết nối thông qua route table, private IP.

Khi dữ liệu được triển khai lên bitbucket, thông qua codedeploy nén lại và lưu trữ trên S3 thành 1 bản ghi mới. Code deploy tiếp tục kết nối dữ liệu từ S3 đến máy chủ EC2 rồi giải nén và lưu vào địa chỉ tập tin được cấu hình sẵn. Quá trình CI/CD đã tự động hóa toàn bộ cấu hình để website được đưa lên web server. Người dùng truy cập website thông qua tên miền (domain nameserver) đã được cấu hình (Mua qua Freenom).

Người quản lý có thể truy cập máy chủ trong private subnet bằng cách SSH thông qua Bastion rồi từ Bastion SSH đến máy chủ để điều khiển và thay đổi cấu hình. Máy chủ không thể tự truy cập ra ngoài internet thông qua IP mà phải thông qua Nat gateway.

## 5. Tổng quan về dự án laravel

### 5.1 Giới thiệu, mô tả dự án

Tên dự án : **“Phát triển hệ thống bán hàng điện tử Vegefoods”**.

#### 5.1.1. Yêu cầu chức năng

“Hệ thống thương mại điện tử Vegefoods” là một hệ thống nhằm giúp người tiêu dùng có thể dễ dàng tiếp cận những nguồn rau củ sạch đảm bảo chất lượng với những đánh giá tin cậy từ những người tiêu dùng khác. Một số chức năng cơ bản:

- Về phía người dùng

o Đăng ký

o Đăng nhập (Tài khoản đăng ký, Google)

o Quên mật khẩu

o Tìm kiếm sản phẩm , lọc sản phẩm

o Mua sản phẩm

o Đánh giá, bình luận về sản phẩm

o Đăng bài viết chia sẻ những điều thú vị

o Chỉnh sửa thông tin cá nhân

o Nhận thông báo về tình trạng đơn hàng

o Theo dõi các đơn hàng

- Về phía người quản trị

o Phân quyền người dùng admin theo các chức năng.

o Quản lý danh mục sản phẩm

o Quản lý tag sản phẩm

o Quản lý sản phẩm

o Quản lý coupon

o Nhận thông báo realtime khi người dùng đặt đơn mới

o Quản lý đặt hàng

o Thống kê sản phẩm, doanh thu theo tháng, quý, năm

o Xuất file excel

- Yêu cầu về trang quản trị

Ứng dụng quản trị này do quản trị viên sử dụng. Quản trị viên là người cao nhất (super admin) nên hoàn toàn làm chủ mọi hoạt động trong hệ thống. Người này có thể phân quyền cho các quản trị viên cấp thấp hơn thực hiện quản lý các chức năng của hệ thống.

#### 5.1.2 Yêu cầu phi chức năng

Các yêu cầu phi chức năng:

*Hiệu năng hoạt động*: Yêu cầu sử dụng tài nguyên tối ưu, thời gian phản hồi nhanh.

*Tương thích*: Ứng dụng có thể chạy tốt trên các trình duyệt khác nhau.

*Tính khả dụng*: Giao diện dễ sử dụng, các thao tác đơn giản.

*Tính tin cậy*: Website luôn đảm bảo sẵn sàng hoạt động, thời gian giữa các lần gián đoạn hoạt động của ứng dụng thấp.

*An toàn thông tin*: Ứng dụng cần được đảm bảo tính bảo mật, toàn vẹn và xác thực.

# Chương 3: TRIỂN KHAI MÔ HÌNH

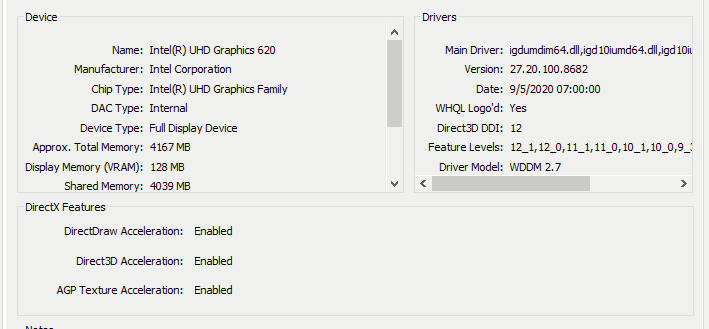
## 1. Môi trường phát triển

* Môi trường cloud : AWS
* Hệ điều hành: Ubuntu Linux 22.04
* Ngôn ngữ lập trình: Terraform, PHP
* Link github: <https://github.com/thai2305dh/thai-tf/tree/develop>
* Link bitbucket: <https://bitbucket.org/thai2305/laravel-devops/src/master/>

## 2. Môi trường thực nghiệm

Máy tính của sinh viên là nơi lưu trữ source code , ở đây sinh viên cấu hình và quản lý các tài nguyên đồng thời thực thi các lệnh để triển khai ứng dụng.

Hệ thống được thực nghiệm ở máy tính của sinh viên có cấu hình như sau:



Hình 3.1 Cấu hình của máy tính sử dụng

## 3. Đối tượng nghiên cứu, phạm vi nghiên cứu

**Đối tượng nghiên cứu**:

+ Các dịch vụ trên AWS sử dụng terraform trong 1 dự án laravel.

+ Quá trình triển khai tự động (CI/CD) trên Bitbucket và Codedeploy.

**Phạm vi nghiên cứu**: Với các dự án laravel thực tế.

## 4. Phương pháp nghiên cứu

**Phương pháp nghiên cứu lý thuyết**: Nghiên cứu, hiểu rõ các service của AWS và cách thức hoạt động của chúng.

**Nghiên cứu trong thực tế**:

+ Đọc tài liệu về các service

+ Nghiên cứu bằng cách thực hành xây dựng các service bằng cách thủ công nhằm nắm rõ các cấu hình cần thiết.

+Tiến hành đi xây dựng hạ tầng với các service được cấu hình bằng terraform.

## 5. Nội dung nghiên cứu

* Thu thập kiến thức về AWS, tìm hiểu về ứng dụng của AWS trong các dự án thực tế, tìm hiểu từng loại service và cách thức sử dụng.
* Thiết kế mô hình thực tế cho project.
* Xây dựng mô hình thực tế thông qua terraform và triển khai tự động thông qua Bitbucket, codedeploy.

## 6. Dữ liệu thực nghiệm:

Thực nghiệm được tiến hành trên website laravel và có thể sử dụng 1 số website khác như Nodejs hoặc python tùy vào nhu cầu thực tế.

Các file cấu hình là source code của trang web được lưu trên github và bitbucket.

## 7. Quy trình triển khai

### 7.1. Tạo máy chủ AMI

Các ứng dụng được sử dụng, cài đặt trong AMI:

* Môi trường cloud : AWS
* Nginx: Yêu cầu cấu hình webserver, ssl.
* Mysql: Yêu cầu chạy và lưu trữ được database của dự án.
* PHP (cho laravel): Yêu cầu cài các dịch vụ liên quan.
* Codedeploy agent : Yêu cầu có thể giúp Codedeploy thực thi quá trình gọi file. Cài đặt codedeploy agent (dịch vụ này sẽ giúp code deploy thao tác với EC2):
* Composer: Cài đặt thư viện cho laravel.
* File .env: Yêu cầu cấu hình cho môi trường hoạt động.

Các dịch vụ được cài đặt thông qua việc gọi 1 file cấu hình test.sh. Khi khởi chạy, các dịch vụ tự động được cài đặt trên máy ảo EC2.

data "template\_file" "webserver" {

    template = file("./config-file/test.sh")

}

Nội dung file cấu hình test.sh

#!/bin/bash -ex

sudo rm -rf /var/lib/apt/lists/lock

sudo apt update

sudo apt install nginx -y

sudo apt dist-upgrade -y

sudo apt install software-properties-common -y

sudo add-apt-repository ppa:ondrej/php -y

sudo apt install php8.0 -y

sudo apt install php-curl php-cli php-mbstring wget git unzip php8.0-mysql php8.0-dom php8.0-xml php8.0-fpm php8.0-xmlwriter phpunit php-mbstring php-xml -y

sudo systemctl restart nginx

#install codedeploy agent

sudo apt install ruby-full -y

wget https://aws-codedeploy-us-east-2.s3.us-east-2.amavailability zoneonaws.com/latest/install

chmod +x ./install

sudo ./install auto > /tmp/logfile

#cài đặt composer

php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"

php -r "if (hash\_file('sha113', 'composer-setup.php') === '7rea8ds87cs92h3br') { echo 'Installer verified'; } else { echo 'Installer corrupt'; unlink('composer-setup.php');};"

php composer-setup.php

php -r "unlink('composer-setup.php');"

sudo mv composer.phar /usr/local/bin/composer

File khởi chạy cấu hình của AMI.

resource "aws\_instance" "webserver" {

    ami = "ami-08a5a3fc362c1e4f7" //Phiên bản HĐH

    instance\_type = "t2.micro"

    subnet\_id = var.ami-subnet

    associate\_public\_ip\_address = true

    vpc\_security\_group\_ids = [aws\_security\_group.sgroup-ami.id]

    key\_name = var.key\_name

    user\_data = var.user-data

    tags = {

    Name = "AMI-server"

  }

}

resource "aws\_ami\_from\_instance" "ami-server" {

    name               = "ami-server"

    source\_instance\_id = "${aws\_instance.webserver.id}"

}

Resource ***aws\_instance***: Tạo instance và khởi chạy instance. Các thông số quan trọng:

* ami: Lựa chọn phiên bản hệ điều hành.
* Instance type: quy định các kích thước từng lọa tương ứng cho các Instance và loại ổ cứng (HDD/SSD).
* Key-name: Thiết lập key để kết nối SSH đến instance.
* Vpc-security-group-ids: Mở các port cần sử dụng.
* User-data: Tên user-data trong instance.

Resource ***aws\_ami\_from\_instance***: Khởi chạy server AMI từ instance vừa tạo.

Cấu hình các tệp hệ thống (.env)

APP\_NAME=Laravel

APP\_ENV=local

APP\_KEY=base64:1CS6Fs85WqdxiuBvE28Yq5QQPCfZWN2HeHl85N7NWZM=

APP\_DEBUG=true

APP\_URL=http://localhost

LOG\_CHANNEL=stack

LOG\_DEPRECATIONS\_CHANNEL=null

LOG\_LEVEL=debug

DB\_CONNECTION=mysql

DB\_HOST=localhost

DB\_PORT=3306

DB\_DATABASE=vegefood

DB\_USERNAME=vegefood

DB\_PASSWORD=Thai2305@

CACHE\_DRIVER=file

FILESYSTEM\_DRIVER=local

QUEUE\_CONNECTION=sync

SESSION\_DRIVER=file

SESSION\_LIFETIME=120

MEMCACHED\_HOST=127.0.0.1

REDIS\_HOST=127.0.0.1

REDIS\_PASSWORD=null

REDIS\_PORT=6379

MAIL\_MAILER=smtp

MAIL\_HOST=smtp.gmail.com

MAIL\_PORT=465

MAIL\_USERNAME=vegefoodmanager@gmail.com

MAIL\_PASSWORD=waukjkilphmfzxtg

MAIL\_ENCRYPTION=ssl

MAIL\_FROM\_ADDRESS=vegefoodmanager@gmail.com

MAIL\_FROM\_NAME="${APP\_NAME}"

AWS\_ACCESS\_KEY\_ID=

AWS\_SECRET\_ACCESS\_KEY=

AWS\_DEFAULT\_REGION=us-east-1

AWS\_BUCKET=

AWS\_USE\_PATH\_STYLE\_ENDPOINT=false

MIX\_PUSHER\_APP\_KEY="${PUSHER\_APP\_KEY}"

MIX\_PUSHER\_APP\_CLUSTER="${PUSHER\_APP\_CLUSTER}"

CLOUDINARY\_API\_KEY=675556124738988

CLOUDINARY\_API\_SECRET=QMugOpDg1ra7OoRlNICR4YASpR0

CLOUDINARY\_URL=cloudinary://675556124738988:QMugOpDg1ra7OoRlNICR4YASpR0@suppermarket

CLOUDINARY\_UPLOAD\_PRESET=https://api.cloudinary.com/v1\_1/suppermarket/image/upload

CLOUDINARY\_NOTIFICATION\_URL=

BROADCAST\_DRIVER=pusher

PUSHER\_APP\_ID=1405232

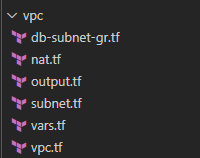
PUSHER\_APP\_KEY=5b2b9c3cd9245e856ddb

PUSHER\_APP\_SECRET=142b647ebad8f5278080

PUSHER\_APP\_CLUSTER=ap1

### 7.2. Tạo VPC theo mô hình đã thiết kế

Cấu trúc module VPC bao gồm:



Hình 3.2 Cấu trúc module VPC

File db-subnet-gr.tf tạo subnet group, cho việc đặt rds trên nhiều subnet dễ dàng hơn:

# subnet-group cho database

resource "aws\_db\_subnet\_group" "subnet-group" {

    subnet\_ids = values(aws\_subnet.private-subnets)[\*].id

    name = "terraform-private-subnet-group"

}

Resource ***aws\_db\_instance\_group***: tạo ra subnet-group cho database, database sẽ được đặt trên nhiều availability zone khác nhau.

* Subnet-ids: gọi đến subnet đã tạo.

File subnet.tf tạo ra các subnet, routable. Các public-subnet và private được tạo ra và đặt tên 1 cách tự động với **for\_each**, người dùng chỉ cần nhập số lượng subnet muốn tạo.

resource "aws\_subnet" "public-subnets" {

    for\_each = var.public-subnet-stt

    vpc\_id = aws\_vpc.vpc.id

    availability\_zone = each.key

    cidr\_block = cidrsubnet(var.cidr-vpc, 4, each.value)

    map\_public\_ip\_on\_launch = true

}

resource "aws\_subnet" "private-subnets" {

    for\_each = var.private-subnet-stt

    vpc\_id = aws\_vpc.vpc.id

    availability\_zone = each.key

    cidr\_block = cidrsubnet(var.cidr-vpc, 4, each.value)

    map\_public\_ip\_on\_launch = false

}

Resource ***aws\_subnet*** tạo các subnet (public hoặc private). Các thông số quan trọng:

* Vpc-id: Vpc mà subnet được tạo.
* Availability-zone: vùng sẵn sàng.
* Cidr-block: Dải mạng theo dải mạng của vpc.
* Map-public-ip-on-launch: yes/no tương ứng với chấp thuận hoặc không chấp thuận kết nối đến internet-gateway.

Tạo route table định hướng đường đi trong VPC

resource "aws\_route\_table" "art-thai" {

    vpc\_id = aws\_vpc.vpc.id

    route  {

        cidr\_block = "0.0.0.0/0"

        gateway\_id = aws\_internet\_gateway.gw-thai.id

    }

    tags = {

        "Name" = "art-thai"

    }

}

resource "aws\_route\_table\_association" "public-subnets" {

    depends\_on = [aws\_subnet.public-subnets]

    for\_each = aws\_subnet.public-subnets

    subnet\_id = each.value.id

    route\_table\_id = aws\_route\_table.art-thai.id

}

Resource ***aws-route-table*** tạo ra các route table.

Resource ***aws-route-table-association*** thiết lập đường đi của các subnet. Các public-subnet được association với Internet-gateway. Có thể truy cập internet trực tiếp mà không cần thông qua trung gian.

File nat.tf tạo nat gateway, liên kết đến private subnet:

resource "aws\_eip" "nat\_eip-1" {

    vpc = true

    depends\_on = [aws\_internet\_gateway.gw-thai]

    tags = {

        Name = "NAT Gateway EIP 1"

    }

}

resource "aws\_nat\_gateway" "nat-1a" {

    allocation\_id = aws\_eip.nat\_eip-1.id

    subnet\_id = values(aws\_subnet.public-subnets)[1].id

    tags = {

        Name = "Main NAT Gateway"

    }

}

resource "aws\_route\_table" "private-rt" {

    vpc\_id = aws\_vpc.vpc.id

    route  {

    cidr\_block = "0.0.0.0/0"

    gateway\_id =  aws\_nat\_gateway.nat-1a.id

    }

    tags = {

        Name = "terraform-route-table-private-subnet"

    }

}

resource "aws\_route\_table\_association" "private-subnets" {

    depends\_on = [aws\_subnet.private-subnets]

    for\_each = aws\_subnet.private-subnets

    subnet\_id = each.value.id

    route\_table\_id = aws\_route\_table.private-rt.id

}

Resource ***aws\_eip***: Tạo ra một IP tĩnh cho nat-gateway có thể kết nối đến internet-gateway.

Resource ***aws\_nat\_gateway***: Tạo ra một nat-gateway.

Các private-subnet được associate với Nat-gateway, Nat-gateway mới thông ra ngoài qua internet-gateway.

### 7.3. Các dịch vụ khởi chạy webserver

- AWS S3 bucket: Tạo ra một bucket tương đối đơn giản:

resource "aws\_s3\_bucket" "data" {

  bucket = var.data\_bucket\_name

  acl    = "private"

  versioning {

    enabled = "true"

  }

  tags = var.tags

  lifecycle\_rule {

    enabled = var.lifecycle\_rule\_enabled

    expiration {

      days = var.expiration

    }

  }

}

Resource ***aws\_s3\_bucket***: Tạo ra một bucket lưu trữ dữ liệu. Các thông số cần thiết:

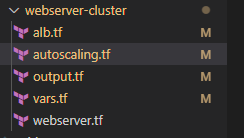
o Bucket: Tên bucket.

o versioning: Cung cấp tài nguyên để kiểm soát việc lập phiên bản trên nhóm S3.

o lifecycle: Cấu hình vòng đời cho S3.

- Webserver-cluster (EC2, Autoscaling group, Load balancing)

Bao gồm các service kể trên. Máy EC2 lúc này được gọi là *launch\_configuration.* Nằm cùng file cấu hình với autoscaling group.



Hình 3.3 Cấu trúc module webserver-cluster

File cấu hình autoscaling group

resource "aws\_autoscaling\_group" "webserver-cluster" {

    name = "terraform-webserver-cluster"

    vpc\_zone\_identifier = [ "${var.asg-subnets}" ]

    min\_size = var.min

    max\_size = var.max

    launch\_configuration = aws\_launch\_configuration.webserver-cluster.id

    force\_delete = true

    lifecycle {

      create\_before\_destroy = true

    }

    tag {

      key = "Name"

      value = "terraform-webserver-cluster"

      propagate\_at\_launch = true

    }

Resource ***aws\_autoscaling\_group*** tạo ra một autoscaling group có thể mở rộng hoặc xóa bỏ linh động các launch\_configuration. Các thông số cần thiết:

o vpc\_zone\_identifier: những availability zone đặt autoscaling group.

o min\_size: Số lượng instance tối thiểu.

o max\_size: Số lượng instance tối đa.

o launch\_configuration: cấu hình của máy chủ EC2 .

o force\_delete: Cho phép xóa autoscaling group mà không cần đợi tất cả các phiên bản trong nhóm kết thúc.

File module cấu hình Load balancing

# ALB -----------------------------------------------------------------

resource "aws\_alb" "alb" {

    name = "terraform-alb"

    internal = var.alb-internal //Check xem ALB có phải nội bộ không

    security\_groups = [ "${aws\_security\_group.sgroup-alb.id}" ]

    subnets = var.public-subnets

    ip\_address\_type = "ipv4"

    load\_balancer\_type = "application"

    idle\_timeout = 60

    enable\_deletion\_protection = false

    enable\_cross\_zone\_load\_balancing = true

    enable\_http2 = false

    depends\_on = [ aws\_autoscaling\_group.webserver-cluster ]

    tags = {

        Name = "terraform-alb"

    }

}

resource "aws\_alb\_listener" "alb-listener-http" {

    count = var.alb-listener-https ? 0 : 1

    load\_balancer\_arn = aws\_alb.alb.arn

    port = var.alb-request-access-port

    protocol = var.alb-request-access-protocol

    default\_action {

        type = "forward"

        target\_group\_arn = aws\_alb\_target\_group.alb.arn

    }

}

resource "aws\_alb\_listener" "alb-listener-https" {

    count = var.alb-listener-https ? 1 : 0

    load\_balancer\_arn = aws\_alb.alb.arn

    port = var.alb-listener-port

    protocol = var.alb-listener-protocol

    ssl\_policy = ""

    certificate\_arn = ""

    default\_action {

        type = var.alb-action

        target\_group\_arn = aws\_alb\_target\_group.alb.arn

    }

}

resource "aws\_alb\_target\_group" "alb" {

    name = "terraform-target"

    vpc\_id = var.vpc-id

    target\_type = "instance"

    port = var.alb-request-access-port

    protocol = var.alb-request-access-protocol

    health\_check {

        path = var.alb-request-health-path

        protocol = var.alb-request-health-protocol

        port = var.alb-request-health-port

    }

    depends\_on = [ aws\_autoscaling\_group.webserver-cluster ]

}

resource "aws\_autoscaling\_attachment" "attach" {

    lb\_target\_group\_arn = aws\_alb\_target\_group.alb.arn

    autoscaling\_group\_name = aws\_autoscaling\_group.webserver-cluster.id

}

Resource ***aws\_alb***: Tạo ra 1 auto loadbalancing liên kết đến autoscaling group. Các thông số cần thiết:

o internal: Kiểm tra xem loadbalancing là nội bộ hay không.

o security\_groups: Cấu hình các port cho loadbalancing.

o ip\_address\_type: Loại IP (IPv4 hoặc IPv6).

o load\_balancer\_type: Loại cân bằng tải.

o idle\_timeout: Thời gian tính bằng giây mà kết nối được phép ở chế độ chờ.

Resource ***aws\_alb\_listener***: Cung cấp tài nguyên Trình xử lý cân bằng tải.

Resource ***aws\_alb\_target\_group***: Cung cấp tài nguyên Nhóm mục tiêu để sử dụng với tài nguyên cân bằng tải.

File webserver-cluster.tf tôi dùng để cấu hình security group cho các dịch vụ kể trên

locals {

  sg\_alb\_rules = {

    "rule\_http" = ["ingress", 80, 80, "tcp", ["0.0.0.0/0"]]

    "rule\_https" = ["ingress", 443, 443, "tcp", ["0.0.0.0/0

    "rule\_out" = ["egress", 0, 0, "-1", ["0.0.0.0/0"]]

  }

  sg\_asc\_rules = {

    "rule\_http\_alb" = ["ingress", 80, 80, "tcp", "${aws\_security\_group.sgroup-alb.id}"]

    "rule\_https\_alb" = ["ingress", 443, 443, "tcp", "${aws\_security\_group.sgroup-alb.id}"]

  }

}

resource "aws\_security\_group" "sgroup-alb" {

    name = "sgroup-alb"

    vpc\_id = var.vpc-id

}

resource "aws\_security\_group\_rule" "sg-rule-alb" {

    security\_group\_id = aws\_security\_group.sgroup-alb.id

    for\_each = local.sg\_alb\_rules

    type = each.value[0]

    to\_port = each.value[1]

    from\_port = each.value[2]

    protocol = each.value[3]

    cidr\_blocks = each.value[4]

}

resource "aws\_security\_group" "sgroup-asc" {

    name = "sgroup-asc"

    vpc\_id = var.vpc-id

}

resource "aws\_security\_group\_rule" "sg-rule-asc" {

    security\_group\_id = aws\_security\_group.sgroup-asc.id

    for\_each = local.sg\_asc\_rules

    type = each.value[0]

    to\_port = each.value[1]

    from\_port = each.value[2]

    protocol = each.value[3]

    source\_security\_group\_id = each.value[4]

}

resource "aws\_security\_group" "sgroup-nat-id" {

    name = "sgroup-nat-id"

    vpc\_id = var.vpc-id

}

resource "aws\_security\_group\_rule" "sgroup-nat-id" {

    security\_group\_id = aws\_security\_group.sgroup-nat-id.id

    type = "ingress"

    to\_port = 0

    from\_port = 0

    protocol = "-1"

    cidr\_blocks = ["0.0.0.0/0"]

}

# Cho phép giao tiếp giữa các hosts trong vpc

resource "aws\_security\_group\_rule" "sg-rule-asg-ingress" {

    security\_group\_id = aws\_security\_group.sgroup-asc.id

    type = "ingress"

    from\_port = 0

    to\_port = 65535

    protocol = "tcp"

    source\_security\_group\_id = aws\_security\_group.sgroup-asc.id

}

* Cấu hình RDS lưu trữ database:

resource "aws\_db\_instance" "rds-instance" {

    identifier = "rds-instance"

    allocated\_storage = 5

    engine = var.engine

    engine\_version = var.engine-version

    instance\_class = var.type

    db\_name = var.db-name

    username = var.username

    password = var.password

    port = var.port

    availability\_zone =  var.availability zone

    db\_subnet\_group\_name = var.subnet-group

    vpc\_security\_group\_ids = [aws\_security\_group.sgroup-rds.id]

    publicly\_accessible = var.db-public-access

    skip\_final\_snapshot = true

    final\_snapshot\_identifier = "rds-final"

}

resource "aws\_security\_group" "sgroup-rds" {

    name = "sgroup-rds"

    vpc\_id = var.vpc-id

}

Resource ***aws\_db\_instance***: Tạo ra một máy chủ RDS với các thông số cần thiết:

o allocated\_storage: Dung lượng lưu trữ.

o engine: Loại database.

o engine\_version: phiên bản sử dụng cho database.

o instance\_class: Loại cấu hình của máy chủ RDS.

### 7.4. Cấu hình CI/CD cho website

Tôi đã thiết kế dự án có hệ thống tệp và thư mục như sau:



Hình 3.4 Hệ thống tệp của dự án

File bitbucket-pipelines.yml và appspec.yml thực thi các script khởi chạy các file trong thư mục devops:

File bitbucket-pipeline.yml

  image: php:8.0-fpm

  pipelines:

    default:

      - step:

          services:

            - mysql

          name: 'Build and Test'

          script:

            - bash ./devops/build-server.sh

            - bash ./devops/build-project.sh

            - bash ./devops/run-tests.sh

      - step:

          name: 'Deployment to Production'

          deployment: production

          script:

            - bash ./devops/build-server.sh

            - bash ./devops/scripts/production-server.sh

            - bash ./devops/deploy-production.sh

* Build-server.sh: Khởi chạy file ~/devops/buildserver.sh.
* Deploy-production.sh: Khởi chạy file ~/devops/deploy-production.sh.

File appspec.yml thực thi các lệnh sau khi triển khai.

version: 0.0

os: linux

files:

  - source: /

    destination: /var/www/vegefood

hooks:

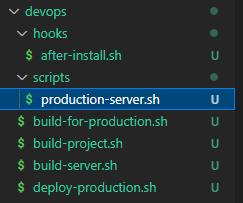
  AfterInstall:

    - location: ./devops/hooks/after-install.sh

      runas: ubuntu

* destination : /var/www/(tên thư mục): Gọi đến thư mục
* Thực thi lệnh của file after-install.sh
* Runas: Thực thi với quyền của user ubuntu

Cấu trúc phần devops



Hình 3.5 Cấu trúc folder devopsBuild-server.sh: Cài đặt các gói còn thiếu (chủ yếu đã được cài trên AMI)

* After-install.sh: Phân quyền, cài thư viện và chạy các lệnh sau cùng.
* Deploy-production.sh: Nén tệp đẩy lên S3 và create deployment.

### 7.5. Cấu hình Codedeploy, bitbucket runner và repository variables

#### 7.5.1. Code deploy

resource "aws\_codedeploy\_app" "laravel" {

    compute\_platform = "Server" // Chọn loại deploy là server

    name             = "codedeploy-laravel"

    tags             = var.tags

}

resource "aws\_codedeploy\_deployment\_group" "laravel" {

  app\_name               = aws\_codedeploy\_app.laravel.name

  deployment\_group\_name  = "${aws\_codedeploy\_app.laravel.name}-deployment-group"

  deployment\_config\_name = var.deployment\_config\_name

  service\_role\_arn       = aws\_iam\_role.roles-scaling.arn

  autoscaling\_groups     =  [var.codedeploy-asc-gr]

}

Resource ***aws\_codedeploy\_app***: tạo ra một application lưu trữ thông tin các deployment. Ở đây chọn loại triển khai là “Server” vì yêu cầu triển khai lên webserver.

Resource ***aws\_codedeploy\_deployment\_group***: tạo ra một deployment group, deployment group này liên kết đến EC2 hoặc Autoscaling group.

resource "aws\_iam\_role" "roles-scaling" {

  name = "codeploy-role"

  assume\_role\_policy = <<EOF

{

  "Version": "2012-10-17",

  "Statement": [

    {

      "Sid": "",

      "Effect": "Allow",

      "Principal": {

        "Service": [

          "codedeploy.amazonaws.com"

        ]

      },

      "Action": "sts:AssumeRole"

    }

  ]

}

EOF

}

resource "aws\_iam\_role\_policy\_attachment" "codedeploy\_attach" {

  role       = aws\_iam\_role.roles-scaling.name

  policy\_arn = "arn:aws:iam::aws:policy/service-role/AWSCodeDeployRole"

}

Resource ***aws\_iam\_role***:tạo ra một phân quyền cho Codedeploy. Codedeploy cần sự phân quyền này mới có thể hoạt động.

#### 7.5.2. Bitbucket và repository variables

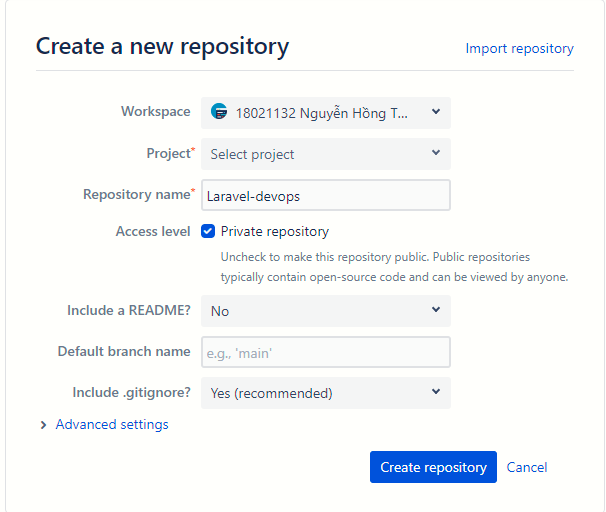
* Tạo tài khoản Bitbucket và tạo repository cho dự án

+ Repository name : Tên dự án.

+ Accsess level: Khi chọn repository là private, người truy cập cần được cho phép thông qua các user và mật khẩu được cấu hình trong bitbucket.

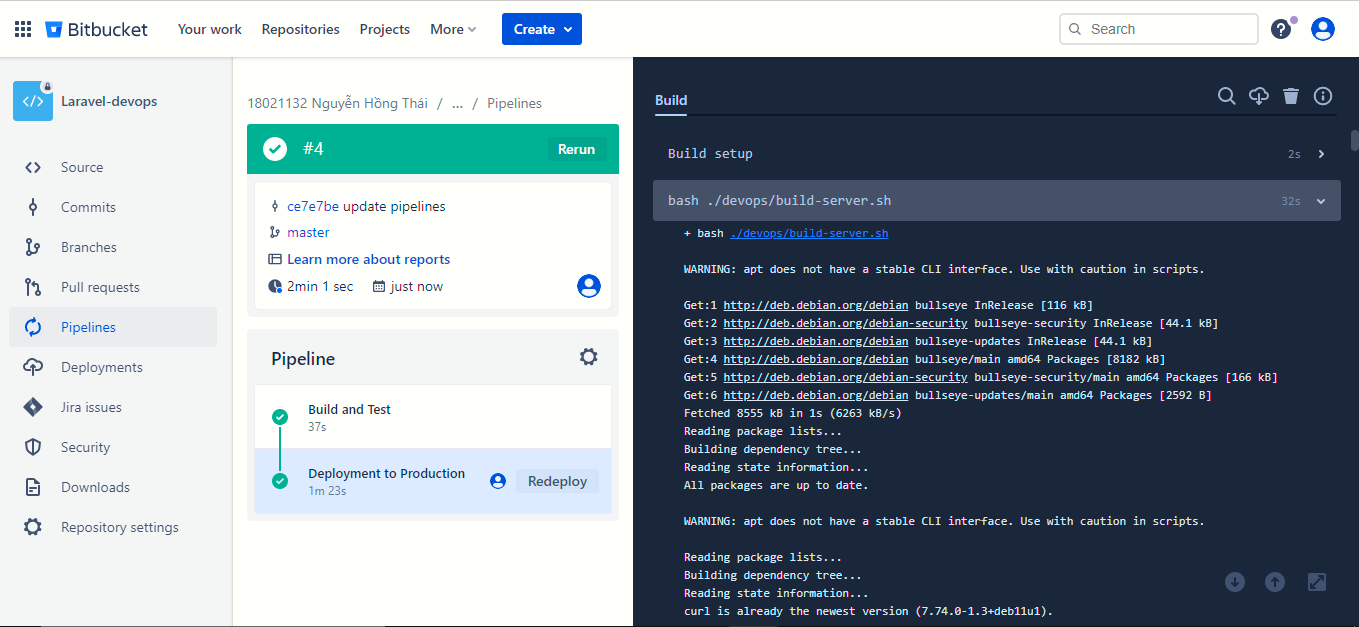
+ Default branch name: Nhánh lưu trữ code. Nếu không chọn thì mặc định data được lưu trữ trên nhánh master.

+ Include .gitignore: Tạo thư viện lưu trữ những file cần bảo mật.



Hình 3.6 Tạo repository

* Pipelines cho repository chạy tự động thông qua file các file bitbucket-pipelines.yml, appspec.yml và các file chứa script được đặt chung với source code trong quá trình triển khai.



Hình 3.7 Pipelines cho repo

* Để repository có thế liên kết đúng đến AWS, ta cần cấu hình biến môi trường. Các biến môi trường được thêm vào ở cấp độ kho lưu trữ có thể được truy cập bởi bất kỳ người dùng nào có quyền đẩy trong kho lưu trữ. Để truy cập một biến, đặt biểu tượng **$** trước tên của nó. Ví dụ: truy cập AWS\_SECRET bằng cách sử dụng $ AWS\_SECRET.

Trong dự án này Repository variables được liên kết đến codedeploy của AWS với các biến:



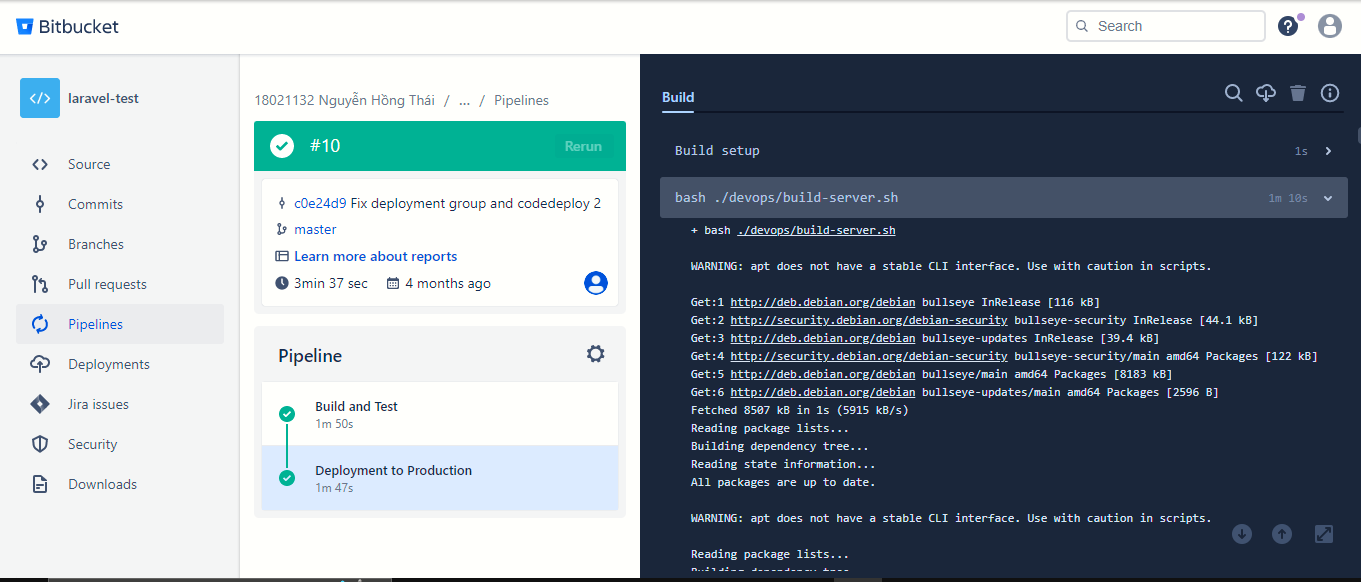
Hình 3.8 Thông số cho repossitory variables

* S3\_bucket: Gọi đến bucket trên AWS
* Access\_key: Key để liên kết triển khai mã nguồn dự án lên AWS
* AWS\_default\_region: Region mà website hoạt động
* Application\_name: Tên dự án
* Deployment\_group\_name: Tên deployment group
* Deployment\_config\_name: Lựa chọn loại deployment

## 8. Kết quả thực nghiệm

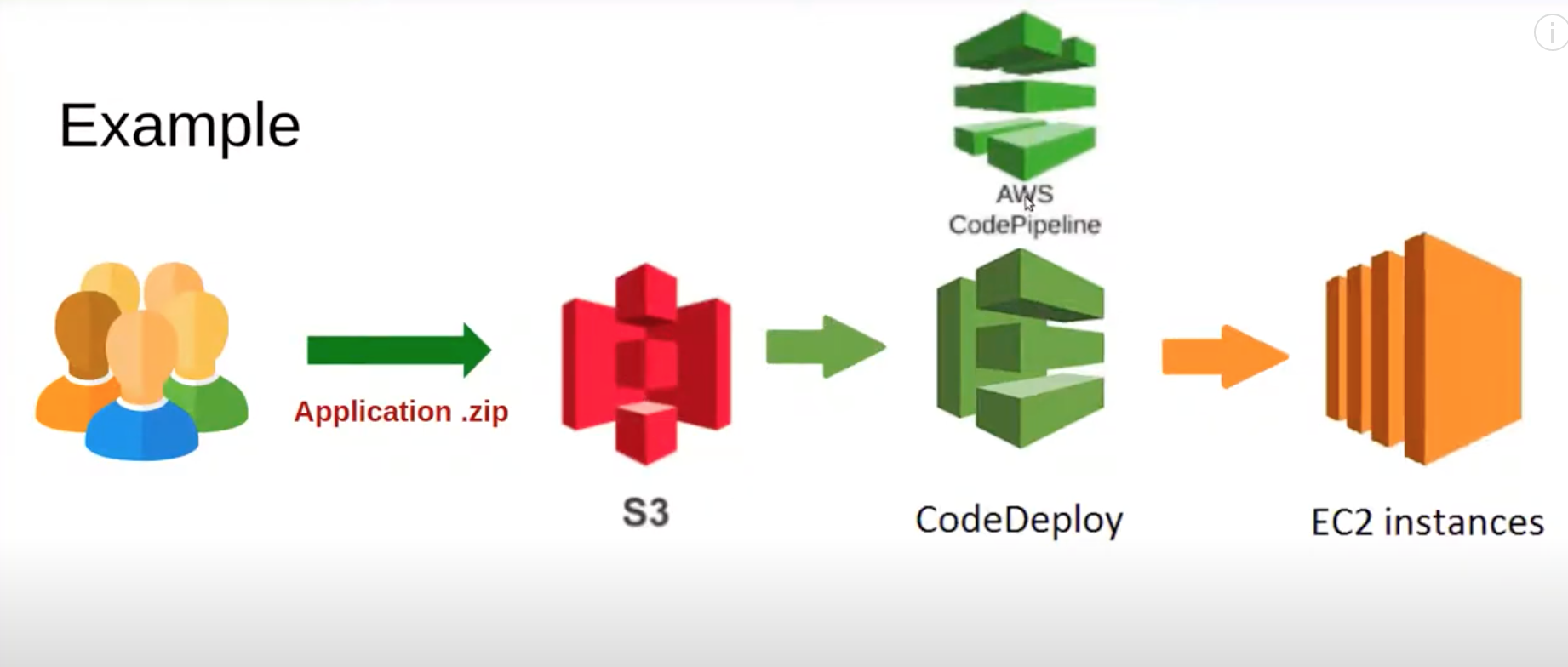
### 8.1. Triển khai trên môi trường thực tế

* Đầu tiên ta thực hiện tạo repository và triển khai website laravel lên trên bitbucket. Code mới sẽ được lập trình viên đẩy lên repository. Bitbucket tự đọc file bitbucket-pipelines.yml và làm việc kiểm tra sai sót trong phần triển khai.

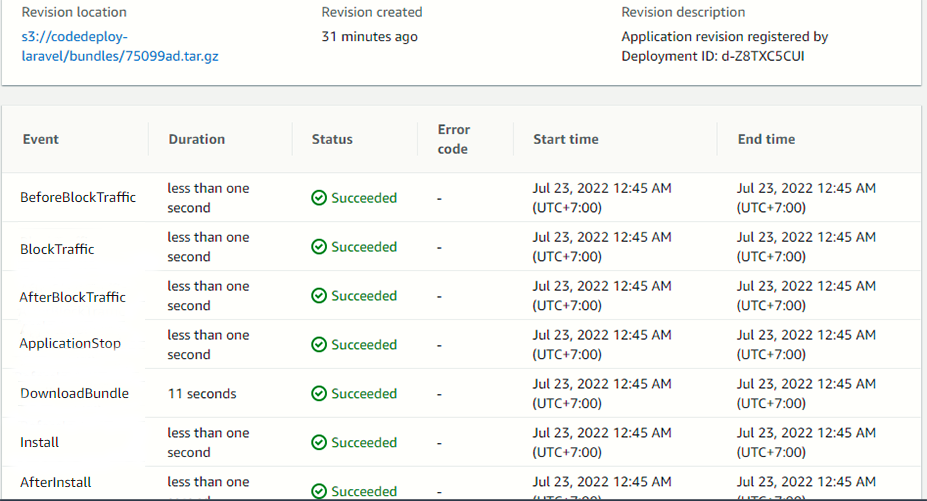


Hình 3.9 Quá trình pipelines thành công

* Ngay khi quá trình kiểm tra được thông qua, Codedeploy sẽ làm việc và tiến hành cập nhật dữ liệu S3 thành một phiên mới, từ EC2 gọi đến dữ liệu trong S3, hiển thị quá trình cập nhật, hệ thống tìm ra lỗi trong code. Nếu không có gì sai sót, website đã được triển khai thành công sau bước này.



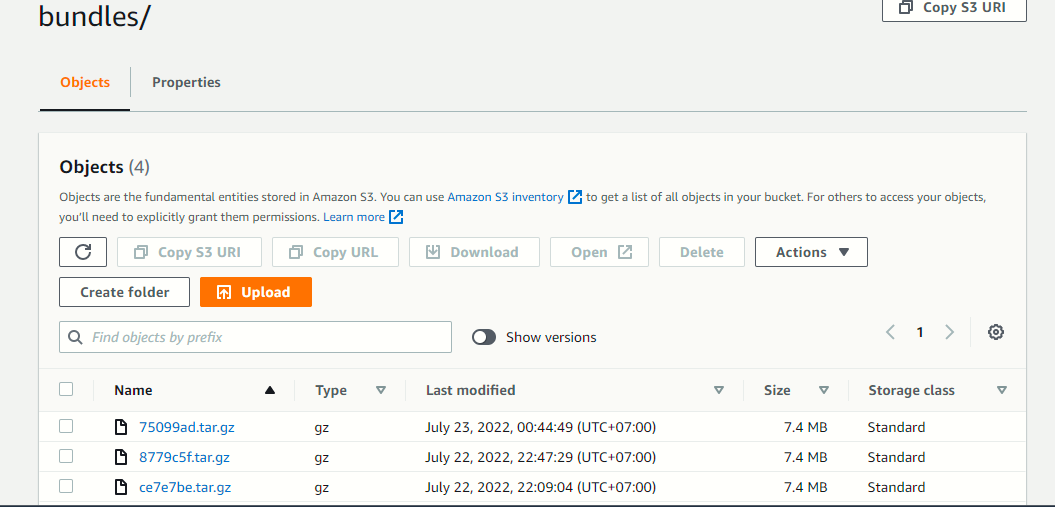
Hình 3.10 Flow codedeploy gọi dữ liệu



Hình 3.11 Quá trình triển khai

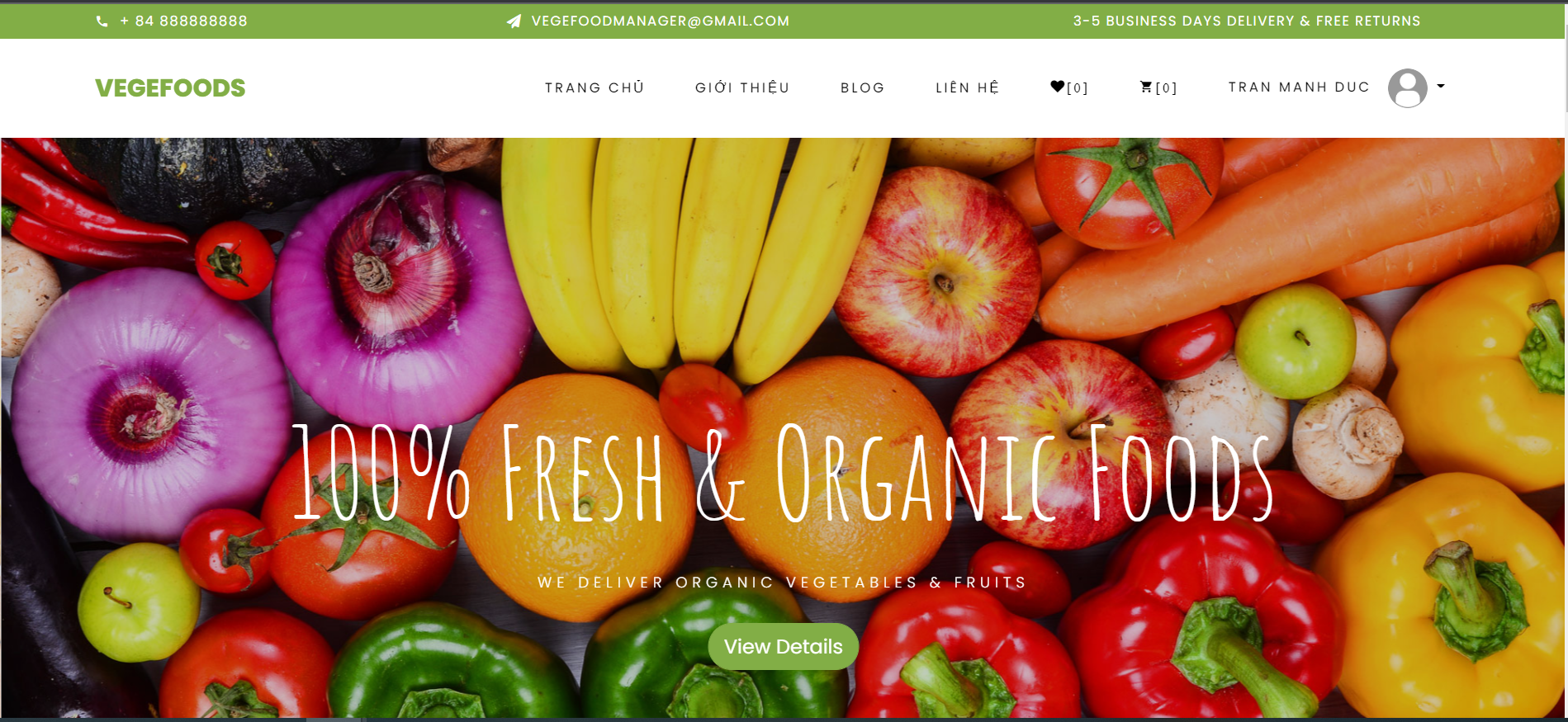
### 8.2. Kết quả triển khai

Sau mỗi lần triển khai, source code được nén lại lưu lại trên S3 bucket. Điều này đảm bảo cho việc rollback lại các phiên bản trước có thể diễn ra ngay lập tức.



Hình 3.12 Các sourcode lưu trên s3

Giao diện của trang web sau khi triển khai thành công



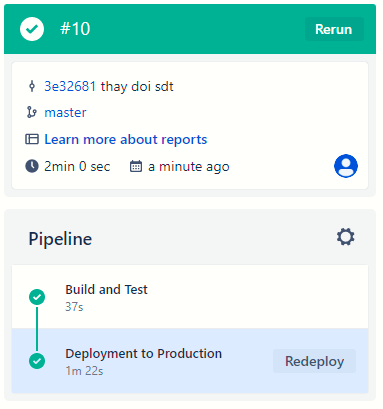
Hình 3.13 Giao diện của website

### 5.3. Kiểm thử

#### 8.3.1. Kiểm thử cập nhật

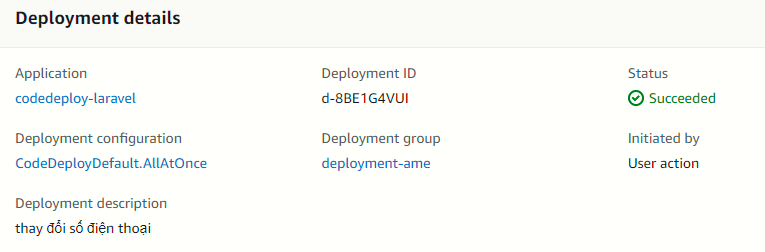
Thay đổi thử số điện thoại hiển thị trên web site, triển khai lại và kiểm tra số điện thoại được cập nhật không.

Kết quả cho thấy sau khi triển khai lại, quá trình cập nhật lên bitbucket thành công



Hình 3.14 Cập nhật lên bitbucket thành công

Codedeploy làm việc thành công:



Hình 3.15 Codedeploy cập nhật thành công

Số điện thoại đã được cập nhật từ 8888888 sang 365341179:

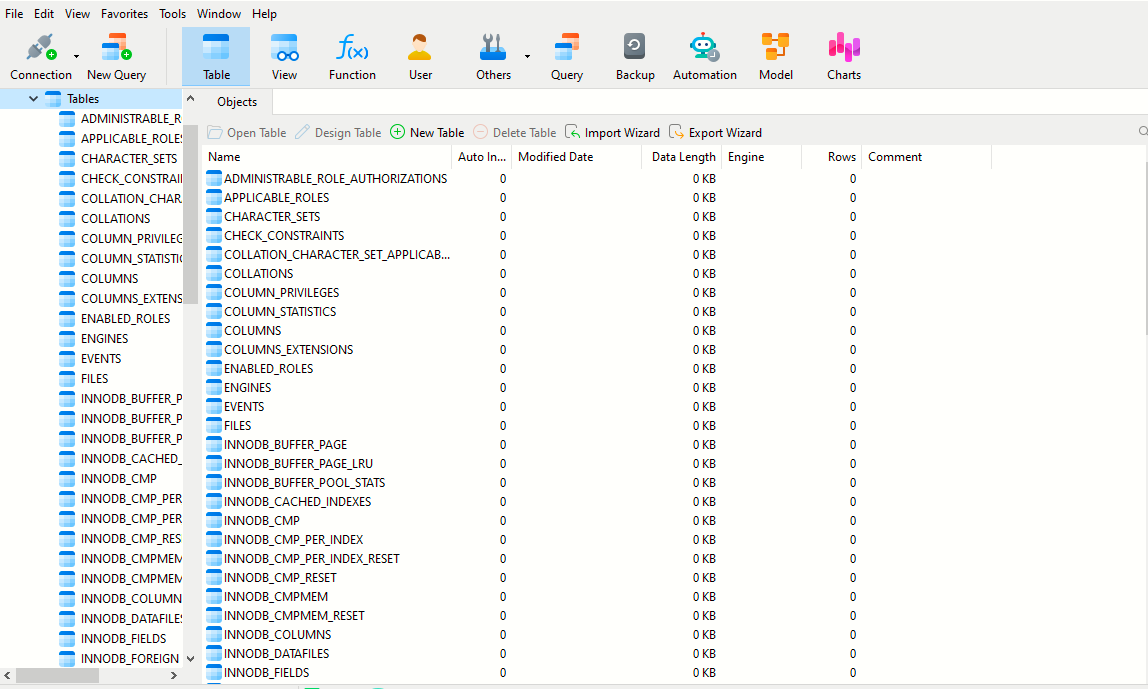


Hình 3.16 Số điện thoại đã được thay đổi

#### 8.3.2. Kiểm thử quản lý

Kiểm tra bằng cách kết nối đến cơ sở dữ liệu bằng máy cá nhân.

Kết quả: Cơ sở dữ liệu vẫn có thể được quản lý bởi thành viên có quyền truy cập



Hình 3.17 Cơ sở dữ liệu được quản lý trên máy cá nhân

#### 8.3.3. Kiểm thử tính sẵn sàng, khả năng mở rộng

Sử dụng lệnh giả lập CPU của instance lên 100%.



Hình 3.18 Lệnh giả lập cpu lên 100%

Kết quả: Cloudwatch thông báo CPu đạt trên 85% quá 1 phút nên tạo thêm instance.



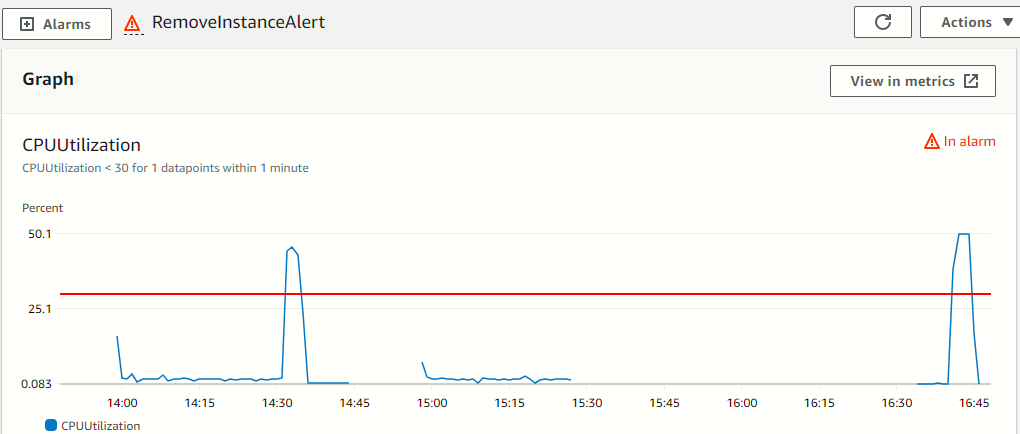
Hình 3.19 Cloudwatch thông báo tạo thêm instance khi CPU trên 85%

Instance đang được tạo thêm



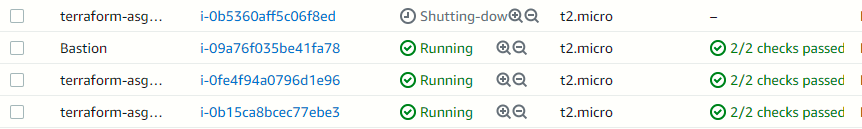
Hình 3.20 Tạo thêm instance

Khi hủy bỏ lệnh trên bằng Ctr + C, CPU trở lại mức bình thường, cloudwatch thông báo CPU dưới 30% trong 1 phút, xóa bớt instance.



Hình 3.21 Cloudwatch thông báo xóa bớt instance khi CPU dưới 30%

Instance đang được xóa bỏ:



Hình 3.22 Instance đang được xóa

Kết luận: Kiểm thử thành công trong các trường hợp được đặt ra

# Kết luận

## 1. Kết quả hiện tại

### Ưu điểm

Mô hình đã đáp ứng được yêu cầu của bài toán về ứng dụng kỹ thuật terraform aws cho bài toán xây dựng và triển khai webserver. Hiển thị trực quan các chức năng, mục tiêu yêu cầu đã đặt ra khi bắt đầu. Xây dựng được mô hình có tính sẵn sàng tốt, có thể đáp ứng tốt cho dự án thức tế . Mô phỏng được quá trình triển khai, cập nhật lại ứng dụng trên thực nghiệm.

Ngôn ngữ terraform đơn giản, dễ hiểu, có thể cài đặt môi trường dễ dàng, thuận tiện cho người dùng. Các bước thực hiện dễ dàng thao tác, phù hợp với mọi đối tượng sử dụng. Mỗi dịch vụ được chia làm một module nhỏ khác nhau, thuận tiện cho việc tìm kiếm và quản lý các dịch vụ. Các file cấu hình có thể sử dụng lại được trong những dự án khác sau này.

Mô hình chạy được trên mọi cấu hình máy tính. Các dịch vụ được tự động hóa rút ngắn thời gian cấu hình và tránh được phát sinh lỗi không mong muốn. Việc tự động hóa các module giúp tối ưu thời gian lao động, giúp hiệu quả trong công việc tăng lên, khả năng phù hợp ứng dụng trong thực tế.

### 1.2. Nhược điểm

Do thời gian và kinh nghiệm còn hạn chế nên kết quả còn chưa nghiên cứu được thành thạo và đầy đủ các dịch vụ. Source code cần được viết dễ hiểu, các dịch vụ cần được cấu hình phù hợp với mục đích của dự án.

Chưa tối ưu triệt để quá trình CI/CD, cần rút ngắn thêm thời gian máy chủ ngưng hoạt động. Tại bước cấu hình cloudwatch chỉ giới hạn thông số cho CPU, chưa thực nghiệm cho các loại tài nguyên khác. Chi phí khi sử dụng dịch vụ Amazon web service làm ảnh hưởng đến việc tự tìm hiểu, nghiên cứu.

Mô hình dừng lại ở việc nghiên cứu và tự thiết kế trên nền tảng desktop, cloud nên khả năng chia sẻ, sử dụng vào thực tế chưa được chứng minh hiệu quả.

## 2. Hướng phát triển sau này

Mở rộng các dịch vụ , tối ưu chi phí cho các dịch vụ. Phát triển khả năng kết nối với các quy trình triển khai khác.

Nghiên cứu bổ sung thêm các dịch vụ có thể sử dụng nhiều trong thực tế như máy chủ ảo (serverless) hoặc docker. Nghiên cứu chuyên sâu hơn về ngôn ngữ terraform, tìm ra các thuật toán hiệu quả, rút ngắn được việc code.

Mở rộng kỹ thuật terraform có thể liên kết với một số dịch vụ khác ngoài AWS, tự động hóa được mọi dịch vụ. Qua đó giúp cho chúng ta có đánh giá tổng quan và chi tiết hơn về độ thuận tiện của kỹ thuật này.

# Tài liệu tham khảo

[1] Amazon web service: <https://aws.amazon.com/>

[2] Learn Terraform: [https://registry.terraform.io/providers/hashicorp/aws/](https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/db_instance)

[3] Bitbucket: [https://bitbucket.org/](https://bitbucket.org/thai2305/laravel-test/pipelines/results/page/1)

[4] Quy trình triển khai: <https://www.ahtcloud.com/aws-codedeploy-bitbucket-laravel>

[5] Các dịch vụ cần cho website laravel: <https://galaxyz.net/cach-cai-dat-laravel-voi-web-server-nginx-tren-ubuntu-1404.418.anews>

[6] Tài liệu về laravel: <https://ngocthanh06.wordpress.com/2018/12/31/tai-lieu-laravel-can-ban-php/>