ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG

**BÁO CÁO ĐỒ ÁN CHUYÊN NGÀNH**

**Bộ khung tạo mã Terraform tự động cho việc triển khai tài nguyên trên nhiều cơ sở hạ tầng đám mây**

***An Automated Terraform Code Generation Framework for Resource Deployment on Multi-Cloud Infrastructure***

**Sinh viên thực hiện:**

Nguyễn Đức Toàn - 22521490

GIẢNG VIÊN HƯỚNG DẪN

Ths. Đỗ Hoàng Hiển

**TP. HỒ CHÍ MINH, 2025**

LỜI CẢM ƠN

Em xin gửi lời cảm ơn chân thành đến **thầy ThS. Đỗ Hoàng Hiển** – giáo viên hướng dẫn em trong quá trình thực hiện đồ án chuyên ngành.

Thông qua các buổi báo cáo, thầy đã truyền đạt cho em những kiến thức quan trọng về Infrastructure as Code (IaC), Terraform, cũng như các nền tảng hạ tầng đám mây như AWS và OpenStack. Thầy không chỉ giúp em hiểu rõ khái niệm kỹ thuật mà còn định hướng cách tiếp cận bài toán thực tế, xử lý logic và tổ chức mã nguồn hiệu quả.

Những góp ý, phản hồi kịp thời và sự hỗ trợ chuyên môn từ thầy là yếu tố quan trọng giúp em từng bước hoàn thiện công cụ. Thông qua quá trình thực hiện đồ án, em cũng đã tích lũy thêm nhiều kinh nghiệm thực tiễn, rèn luyện kỹ năng lập trình, DevOps và tư duy triển khai hệ thống.

Mặc dù đã cố gắng hoàn thành tốt nhất trong khả năng, nhưng do hạn chế về thời gian và kinh nghiệm thực tế, báo cáo chắc chắn vẫn còn những thiếu sót. Em rất mong nhận được những ý kiến đóng góp từ thầy để có thể hoàn thiện hơn trong các nghiên cứu và dự án sau này.

Một lần nữa, em xin cảm ơn thầy vì sự hỗ trợ và đồng hành trong suốt quá trình thực hiện đồ án.

MỤC LỤC

[Chương 1: Giới thiệu 2](#_Toc203085175)

[1.1. Bối cảnh 2](#_Toc203085176)

[1.2. Vấn đề đặt ra 2](#_Toc203085177)

[1.3. Mục tiêu đồ án 3](#_Toc203085178)

[Chương 2: Cơ sở lý thuyết 4](#_Toc203085179)

[2.1. Infrastructure as Code (IaC) 4](#_Toc203085180)

[2.1.1. Khái niệm 4](#_Toc203085181)

[2.1.2. Lợi ích 5](#_Toc203085182)

[2.2. Terraform 6](#_Toc203085183)

[2.2.1. Giới thiệu tông quan 6](#_Toc203085184)

[2.2.2. Kiến trúc 6](#_Toc203085185)

[2.2.3. Quy trình sử dụng 8](#_Toc203085186)

[2.2.4. Các đặc điểm nổi bật 9](#_Toc203085187)

[2.3. Amazon Web Services (AWS) 10](#_Toc203085188)

[2.3.1. Giới thiệu tổng quan 10](#_Toc203085189)

[2.3.2. Ứng dụng trong đồ án 11](#_Toc203085190)

[2.4. OpenStack 13](#_Toc203085191)

[2.4.1. Giới thiệu tổng quan 13](#_Toc203085192)

[2.4.2. Kiến trúc 13](#_Toc203085193)

[2.4.3. Ứng dụng trong đồ án 14](#_Toc203085194)

[2.5. Cloud Init 15](#_Toc203085195)

[2.5.1. Giới thiệu tổng quan 15](#_Toc203085196)

[2.5.2. Chức năng chính 16](#_Toc203085197)

[2.5.3. Các định dạng hỗ trợ 17](#_Toc203085198)

[2.5.4. Dữ liệu đầu vào 19](#_Toc203085199)

[2.5.5. Các giai đoạn thực thi 20](#_Toc203085200)

[2.5.6. Các pha thực thi 21](#_Toc203085201)

[2.5.7. Ứng dụng trong đồ án 22](#_Toc203085202)

[Chương 3: Phân tích thiết kế hệ thống 23](#_Toc203085203)

[3.1. Kiến trúc 23](#_Toc203085204)

[3.2. File json mô tả hệ thống 24](#_Toc203085205)

[3.3. Thư mục mẫu cấu hình hạ tầng 26](#_Toc203085206)

[3.4. Cấu trúc mã nguồn hệ thống Terraform Generator 30](#_Toc203085207)

[3.5. Luồng hoạt động 32](#_Toc203085208)

[Chương 4: Hiện thực hệ thống 34](#_Toc203085209)

[4.1. Tải mã nguồn từ github 34](#_Toc203085210)

[4.2. Cài đặt công cụ và thư viện cần thiết 35](#_Toc203085211)

[4.3. Chuẩn bị dữ liệu xác thực và cấu hình: 35](#_Toc203085212)

[4.4. Triển khai và thực thi script 36](#_Toc203085213)

[Chương 5: Thực nghiệm và đánh giá 37](#_Toc203085214)

[5.1. Kịch bản 1: Tạo topology openstack 2 bản sao 37](#_Toc203085215)

[5.2. Kịch bản 2: Tạo topology aws 2 bản sao 43](#_Toc203085216)

[5.3. Kịch bản 3: Tạo cụm kubernetest 1 master 1 worker trên AWS 46](#_Toc203085217)

[5.4. Kịch bản 4: Demo quản lý hạ tầng bằng giao diện 49](#_Toc203085218)

[5.5. Kết quả triển khai 50](#_Toc203085219)

[Chương 6: Hạn chế và hướng phát triển. 51](#_Toc203085220)

[6.1. Một số hạn chế của hệ thống 51](#_Toc203085221)

[6.2. Hướng phát triển trong tương lai 52](#_Toc203085222)

DANH MỤC HÌNH

[Hình 1 Logo Terraform 6](#_Toc203087525)

[Hình 2 Kiến trúc terraform 6](#_Toc203087526)

[Hình 3 Quy trình hoạt động của terraform 8](#_Toc203087527)

[Hình 4 Logo AWS 10](#_Toc203087528)

[Hình 5 Các dịch vụ nổi tiếng của AWS 11](#_Toc203087529)

[Hình 6 Logo Openstack 13](#_Toc203087530)

[Hình 7 Các thành phần của Openstack 14](#_Toc203087531)

[Hình 8 Định dạng được cloud init hỗ trợ 17](#_Toc203087532)

[Hình 9 quy trình khởi tạo và cấu hình một cloud instance 19](#_Toc203087533)

[Hình 10 Mối quan hệ giữa các giai đoạn khởi động (Stages) và các pha xử lý module (Phases) 21](#_Toc203087534)

[Hình 11 Kiến trúc Terraform Generator 23](#_Toc203087535)

[Hình 12 Cấu trúc module xử lý của AWS và Openstack 29](#_Toc203087536)

[Hình 13 Vị trí thư mục mẫu trong terraform generatỏ 30](#_Toc203087537)

[Hình 14 Cấu trúc terraform generator 33](#_Toc203087538)

[Hình 15 Luồng hoạt động của terraform generator 35](#_Toc203087539)

[Hình 16 Cấu hình tfvar cho openstack 39](#_Toc203087540)

[Hình 17 Kết quả generate 41](#_Toc203087541)

[Hình 18 Thư mục được tạo trong terraform project 42](#_Toc203087542)

[Hình 19 Apply tự động các cấu hình 42](#_Toc203087543)

[Hình 20 Các máy ảo được tạo ra 42](#_Toc203087544)

[Hình 21 Thuộc tính của vm1 43](#_Toc203087545)

[Hình 22 Cloud init thành công 43](#_Toc203087546)

[Hình 23 Network được tạo 44](#_Toc203087547)

[Hình 24 Subnet tương ứng với network 44](#_Toc203087548)

[Hình 25 Router được tạo 44](#_Toc203087549)

[Hình 26 Router có interface tương ứng 45](#_Toc203087550)

[Hình 27 Topology tạo ra đúng với định nghĩa 45](#_Toc203087551)

[Hình 28 Máy ảo được tạo 47](#_Toc203087552)

[Hình 29 Cloud init trên vm1 được thực hiện thành công 48](#_Toc203087553)

[Hình 30 Các subnet được tạo 48](#_Toc203087554)

[Hình 31 NAT Gateway được tạo 49](#_Toc203087555)

[Hình 32 Elastic IP được tạo 49](#_Toc203087556)

[Hình 33 Cloud init output trên master 51](#_Toc203087557)

[Hình 34 Node worker 52](#_Toc203087558)

[Hình 35 Worker join cluster thành công 52](#_Toc203087559)

[Hình 36 Thực hiện Destroy trên 1 thư mục cấu hình Open stack 53](#_Toc203087560)

DANH MỤC BẢNG

[Bảng 1 Các thành phần của openstack 14](#_Toc203085263)

[Bảng 2 Thuộc tính của instance 27](#_Toc203085264)

[Bảng 3 Thuộc tính của network 28](#_Toc203085265)

[Bảng 4 Thuộc tính của router 28](#_Toc203085266)

**DANH MỤC TỪ VIẾT TẮT**

API: Application Programming Interface

AWS: Amazon Web Services

CIDR: Classless Inter-Domain Routing

CI/CD: Continuous Integration / Continuous Deployment

CLI: Command-Line Interface

CPU: Central Processing Unit

EIP: Elastic IP

HCL: HashiCorp Configuration Language

IaC: Infrastructure as Code

JSON: JavaScript Object Notation

K8s: Kubernetes

NAT: Network Address Translation

RAM: Random Access Memory

SG: Security Group

SSH: Secure Shell

VM: Virtual Machine

VPC: Virtual Private Cloud

YAML: Yet Another Markup Language

TÓM TẮT

Trong bối cảnh phát triển phần mềm hiện đại, đặc biệt là khi các tổ chức đang chuyển dịch mạnh mẽ sang mô hình DevOps và hạ tầng đám mây, nhu cầu triển khai và quản lý hạ tầng một cách linh hoạt, nhanh chóng và có thể kiểm soát được phiên bản là yếu tố then chốt để duy trì hiệu quả vận hành. Việc triển khai hạ tầng theo cách thủ công truyền thống không còn phù hợp khi các hệ thống ngày càng phức tạp, đòi hỏi sự tự động hóa để đảm bảo tính nhất quán giữa các môi trường, giảm thiểu lỗi con người và tăng khả năng mở rộng.

**Infrastructure as Code (IaC)** được phát triển như một cách tiếp cận hiện đại nhằm mô tả toàn bộ hạ tầng dưới dạng mã nguồn, từ đó có thể áp dụng các nguyên tắc phát triển phần mềm như kiểm soát phiên bản, kiểm thử, CI/CD cho quá trình triển khai hệ thống. Trong số các công cụ hỗ trợ IaC, **Terraform** nổi bật bởi khả năng khai báo tài nguyên hạ tầng trên nhiều nhà cung cấp khác nhau như AWS, OpenStack, GCP hay Azure, đồng thời cung cấp khả năng mô-đun hóa, giúp người dùng tái sử dụng và chia sẻ cấu hình dễ dàng.

Tuy nhiên, việc viết mã Terraform cho các hệ thống phức tạp hoặc đa nền tảng vẫn là một thách thức lớn, đòi hỏi thời gian và kiến thức chuyên sâu về từng nhà cung cấp dịch vụ. Đặc biệt, khi triển khai nhiều kiến trúc tương tự nhau hoặc cần tùy biến nhanh theo nhu cầu, việc thao tác thủ công với hàng loạt file cấu hình trở nên kém hiệu quả và dễ sinh lỗi.

Đồ án này được thực hiện với mục tiêu xây dựng một **công cụ hỗ trợ sinh mã Terraform tự động từ file JSON mô tả hạ tầng**. File JSON này sẽ đóng vai trò như một bản thiết kế tổng thể, có thể mô tả các thành phần chính của hệ thống như mạng, máy ảo,... Từ đó, công cụ có thể tạo ra các tập tin Terraform sẵn sàng triển khai trên nhiều nền tảng cloud khác nhau, đặc biệt là AWS và OpenStack.

# Giới thiệu

## Bối cảnh

Sự phát triển nhanh chóng của công nghệ điện toán đám mây và mô hình phát triển phần mềm theo hướng DevOps đã làm thay đổi cách các tổ chức thiết kế và vận hành hạ tầng CNTT. Việc triển khai hệ thống theo phương pháp truyền thống – tức là cấu hình hạ tầng thủ công từng bước – không còn đáp ứng được nhu cầu về tính linh hoạt, hiệu quả và độ tin cậy trong môi trường hiện đại. Khi số lượng môi trường (dev, staging, production) tăng lên, nhu cầu đảm bảo sự đồng nhất giữa các cấu hình hạ tầng càng trở nên quan trọng.

Trong bối cảnh đó, mô hình **Infrastructure as Code (IaC)** đã nổi lên như một phương pháp tiếp cận tiên tiến, cho phép mô tả toàn bộ hạ tầng bằng mã nguồn có thể quản lý, lưu trữ trong hệ thống kiểm soát phiên bản và tích hợp vào quy trình CI/CD. Thay vì thực hiện từng thao tác cấu hình, các kỹ sư có thể triển khai hệ thống chỉ bằng một câu lệnh. Điều này mang lại rất nhiều lợi ích như giảm thiểu lỗi thao tác, tăng khả năng tái sử dụng, dễ dàng triển khai hàng loạt môi trường giống nhau và đảm bảo việc ghi nhận lịch sử thay đổi.

**Terraform** là một trong những công cụ đi đầu trong lĩnh vực IaC, với khả năng hỗ trợ nhiều nhà cung cấp cloud, đồng thời cho phép người dùng định nghĩa hạ tầng bằng cách khai báo trạng thái mong muốn. Với cú pháp thân thiện, khả năng mô-đun hóa và tích hợp tốt với các quy trình DevOps, Terraform nhanh chóng trở thành lựa chọn phổ biến trong cộng đồng kỹ sư hạ tầng. Tuy nhiên, dù mạnh mẽ, Terraform vẫn yêu cầu người dùng phải hiểu rõ cấu trúc tài nguyên của từng nền tảng, dẫn đến khó khăn khi làm việc với môi trường đa cloud hoặc khi có nhu cầu mở rộng hệ thống sang nhà cung cấp khác.

## Vấn đề đặt ra

Mặc dù Infrastructure as Code (IaC) và công cụ Terraform đã mang lại nhiều lợi ích trong việc tự động hóa và kiểm soát phiên bản hạ tầng, nhưng quá trình xây dựng và duy trì các tập tin cấu hình Terraform cho từng nền tảng đám mây vẫn còn gặp nhiều thách thức. Nguyên nhân chính là do sự khác biệt trong cách định nghĩa tài nguyên, cấu trúc dữ liệu và nguyên tắc thiết kế hạ tầng của mỗi nhà cung cấp.

Ví dụ điển hình là khi tạo mạng ảo và mạng con trên hai nền tảng phổ biến: **AWS** và **OpenStack**. Trên AWS, khái niệm **VPC (Virtual Private Cloud)** được sử dụng để đại diện cho một mạng ảo tách biệt, trong đó các **Subnet** được định nghĩa để phân chia vùng địa chỉ IP. Trong khi đó, **OpenStack** sử dụng tài nguyên **network** và **subnet** tương ứng, nhưng cách tổ chức và liên kết tài nguyên lại khác biệt. Cụ thể, AWS yêu cầu cấu hình thêm các tài nguyên như Internet Gateway, Route Table để kết nối mạng ra ngoài, trong khi OpenStack sử dụng Router để liên kết giữa subnet nội bộ và external network. Những khác biệt này khiến việc chuyển đổi hoặc dùng chung mô hình hạ tầng giữa các nền tảng trở nên phức tạp nếu không có một lớp trừu tượng hóa phù hợp.

Không chỉ vậy, trong các tổ chức cần triển khai nhiều môi trường có kiến trúc tương tự — chẳng hạn như triển khai hạ tầng cho các nhóm phát triển riêng biệt, môi trường staging và production độc lập, hoặc các vùng địa lý khác nhau — thì việc lặp đi lặp lại thao tác viết mã Terraform thủ công sẽ dẫn đến dư thừa, khó kiểm soát, dễ sai sót và mất nhiều thời gian bảo trì.

## Mục tiêu đồ án

Đồ án hướng đến việc xây dựng một **công cụ tự động sinh mã Terraform** từ file JSON mô tả topology hệ thống, nhằm hỗ trợ triển khai hạ tầng nhanh chóng, chính xác và linh hoạt trên nhiều nền tảng đám mây. Các mục tiêu cụ thể gồm:

* **Phát triển framework sinh mã Terraform đa nền tảng**:   
  Cho phép chuyển đổi dữ liệu từ file JSON sang các resource Terraform phù hợp với từng nền tảng cloud như **AWS** và **OpenStack**, hỗ trợ định nghĩa đầy đủ các thành phần như máy ảo, mạng, subnet, router, gateway và bảo mật.
* **Hỗ trợ nhân bản topology với định danh riêng**:  
  Cho phép tạo nhiều bản sao của cùng một kiến trúc hạ tầng với tên định danh khác nhau, giúp triển khai nhanh các môi trường độc lập như dev, test hoặc multi-region.
* **Xác thực dữ liệu đầu vào và ánh xạ chính xác**:  
  Tích hợp kiểm tra cấu trúc file JSON (schema validation), đảm bảo dữ liệu hợp lệ và ánh xạ đúng với các resource Terraform tương ứng của từng nhà cung cấp.
* **Tổ chức mã sinh ra theo chuẩn và dễ tích hợp DevOps**:  
  Mã Terraform được tạo theo cấu trúc thư mục rõ ràng, phân tách module hợp lý, dễ quản lý, dễ mở rộng và thuận tiện đưa vào pipeline CI/CD.
* **Hỗ trợ các tính năng nâng cao phục vụ triển khai thực tế**:  
  Bao gồm cloud init để cấu hình VM ban đầu, floating IP, security group và keypair – giúp mô tả các hệ thống mạng phức tạp và đáp ứng nhu cầu triển khai đa dạng.

# Cơ sở lý thuyết

## Infrastructure as Code (IaC)

### Khái niệm

Infrastructure as Code (IaC) là một phương pháp tiếp cận trong quản trị hạ tầng hiện đại, cho phép người quản trị mô tả và triển khai hạ tầng thông qua mã nguồn, thay vì phải cấu hình thủ công trên giao diện người dùng hoặc can thiệp trực tiếp vào phần cứng. Thông qua IaC, toàn bộ tài nguyên như máy ảo, mạng, hệ thống lưu trữ … được định nghĩa rõ ràng dưới dạng cấu hình có thể đọc được bởi con người và xử lý được bởi máy tính. [1]

Các mô tả này thường được viết bằng các ngôn ngữ khai báo như **HCL**, **YAML** hoặc **JSON**, tùy thuộc vào công cụ IaC được sử dụng (ví dụ: Terraform, AWS CloudFormation, Ansible,...). Khi thay đổi cấu hình, người dùng chỉ cần cập nhật mã nguồn và chạy lại quá trình triển khai để đồng bộ trạng thái thực tế với trạng thái mong muốn. Điều này giúp đảm bảo tính nhất quán giữa các môi trường (dev, test, prod), tránh sai lệch do thao tác thủ công.

### Lợi ích

Việc áp dụng Infrastructure as Code (IaC) mang lại nhiều lợi thế rõ rệt trong việc quản lý hạ tầng công nghệ thông tin hiện đại. Một số lợi ích nổi bật bao gồm:

* **Tự động hóa toàn diện quá trình triển khai hạ tầng:**

IaC giúp loại bỏ phần lớn thao tác thủ công trong việc khởi tạo và cấu hình tài nguyên, từ đó tăng tốc độ triển khai, đảm bảo sự chính xác và giảm thiểu lỗi do con người gây ra.

* **Dễ dàng nhân bản và mở rộng môi trường:**

Một cấu hình hạ tầng khi đã được định nghĩa bằng mã có thể được áp dụng lại ở nhiều môi trường khác nhau (dev, test, staging, production) chỉ với vài thay đổi nhỏ.

* **Quản lý cấu hình có kiểm soát và có thể truy vết:**

Việc lưu trữ mã hạ tầng trong hệ thống quản lý mã nguồn như Git cho phép theo dõi lịch sử thay đổi, xác định ai thay đổi cái gì và khi nào, từ đó dễ dàng quay lại phiên bản trước nếu có sự cố.

* **Hỗ trợ tích hợp liên tục và triển khai liên tục (CI/CD):**

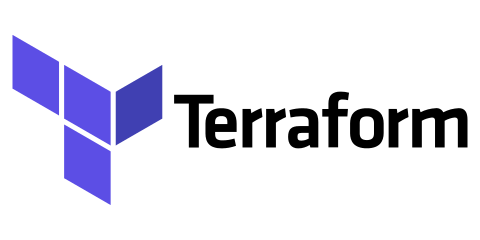
IaC cho phép đưa hạ tầng vào cùng quy trình phát triển phần mềm, giúp tự động kiểm thử, triển khai và cập nhật hạ tầng khi có thay đổi trong mã nguồn, góp phần đẩy nhanh vòng đời phát hành sản phẩm.

* **Thuận tiện trong việc kiểm thử hạ tầng:**

Nhờ cấu trúc định nghĩa hạ tầng như mã, người dùng có thể áp dụng kỹ thuật kiểm thử tự động để kiểm tra cấu hình, đánh giá sự tương thích và đảm bảo chất lượng trước khi triển khai thực tế.

## Terraform

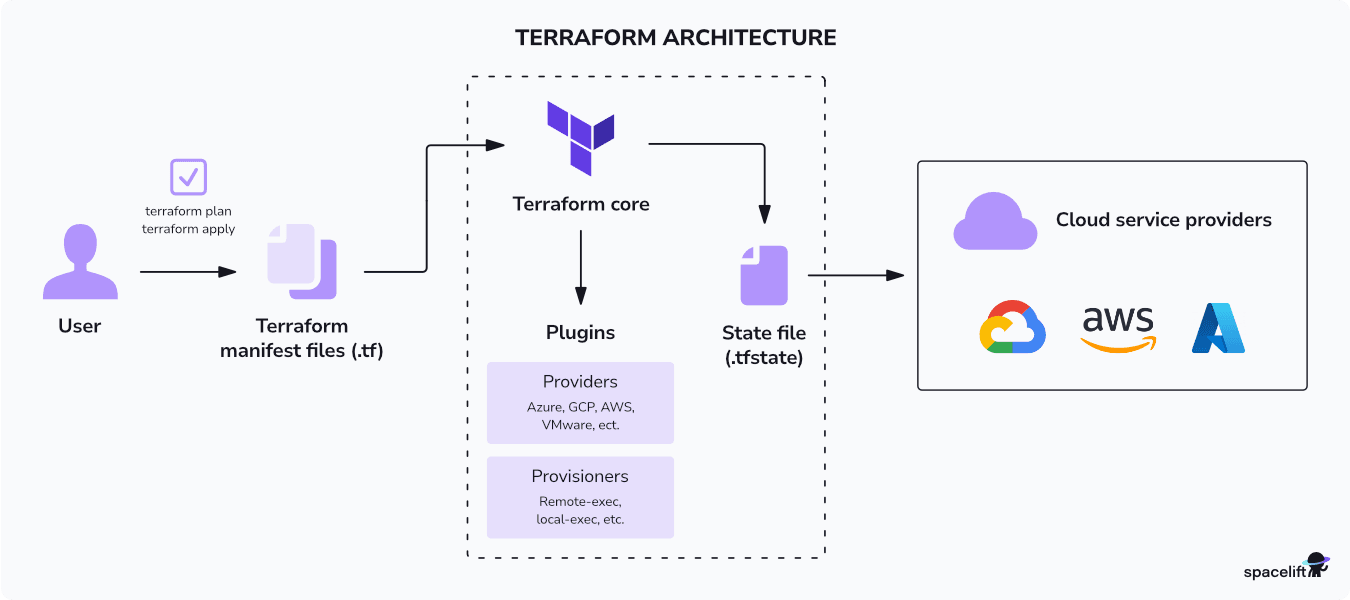
### Giới thiệu tổng quan



Hình Logo Terraform

Terraform là một công cụ mã nguồn mở thuộc mô hình Infrastructure as Code (IaC), được phát triển bởi HashiCorp, nhằm hỗ trợ việc xây dựng và quản lý hạ tầng một cách có hệ thống, tự động và nhất quán. Công cụ này cho phép người dùng mô tả các thành phần hạ tầng dưới dạng mã cấu hình, từ đó thực hiện triển khai và điều chỉnh tài nguyên theo cách có thể lặp lại và dễ kiểm soát [2].

### Kiến trúc

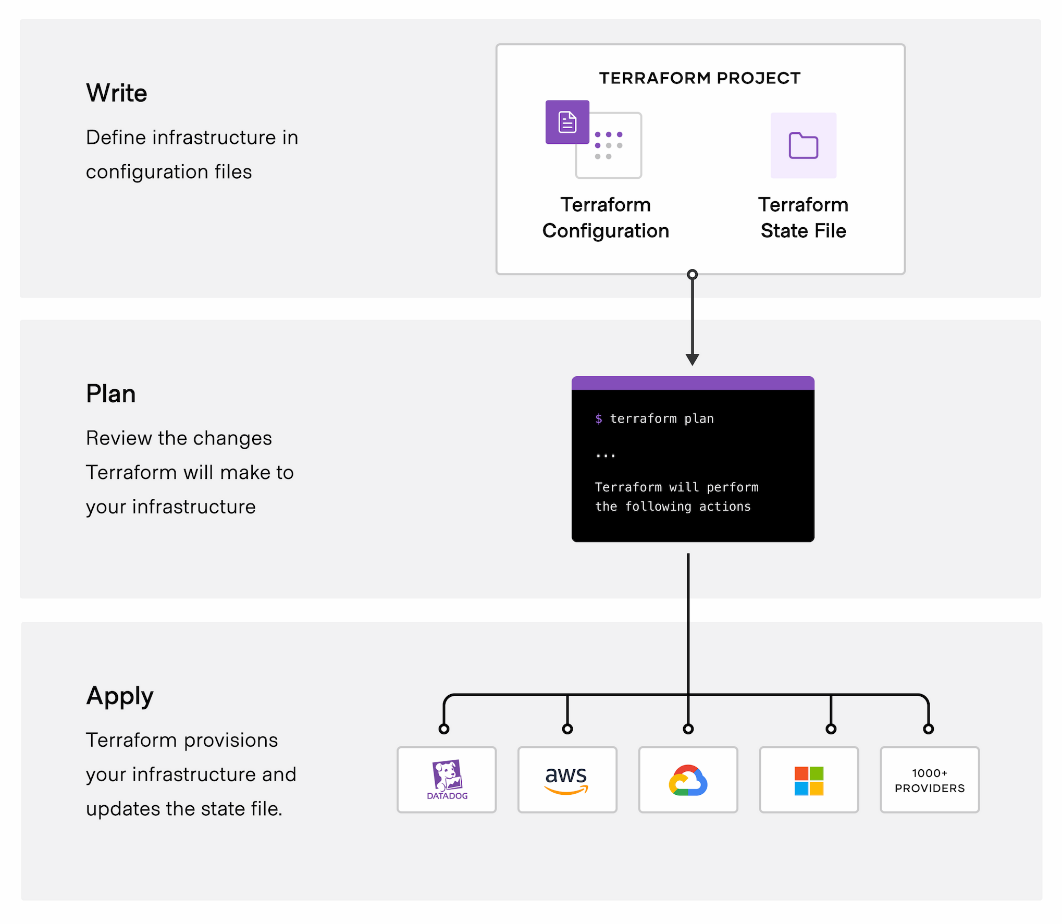


Hình Kiến trúc terraform

Terraform có kiến trúc modular và client-server, bao gồm các thành phần chính sau:

* **Terraform Core:** Là engine xử lý chính, chịu trách nhiệm đọc các file cấu hình, quản lý trạng thái (state), lập kế hoạch (plan) và thực thi (apply) các thay đổi hạ tầng. Thành phần này được biên dịch thành một file nhị phân và cung cấp giao diện dòng lệnh (CLI) để người dùng tương tác và điều khiển quá trình triển khai hạ tầng một cách hiệu quả.
* **Providers:** Là các module kết nối Terraform với API của các dịch vụ như AWS, Azure, Kubernetes để quản lý tài nguyên tương ứng. Mỗi provider định nghĩa các loại tài nguyên mà Terraform có thể thao tác và chuyển đổi các cấu hình thành các lệnh gọi API đặc thù của dịch vụ đó. Có rất nhiều provider do HashiCorp và cộng đồng phát triển, giúp Terraform có thể quản lý đa dạng tài nguyên.
* **State File:** Đây là tệp JSON lưu trữ trạng thái hiện tại của hạ tầng do Terraform quản lý. State file cho phép Terraform theo dõi các tài nguyên đã tạo, so sánh trạng thái hiện tại với cấu hình mong muốn và xác định các thay đổi cần thiết. State file có thể lưu trữ cục bộ hoặc trên các backend từ xa như Amazon S3, Azure Storage, Consul để hỗ trợ làm việc nhóm và bảo mật.
* **Configuration Files:** Các file cấu hình Terraform được viết bằng ngôn ngữ HCL, có cấu trúc rõ ràng, dễ đọc và dễ bảo trì. File cấu hình bao gồm các block như:
  + **Terraform block:** Khai báo provider cần thiết và phiên bản.
  + **Provider block:** Cấu hình kết nối tới nhà cung cấp dịch vụ.
  + **Resource block:** Định nghĩa tài nguyên cần tạo hoặc quản lý.
  + **Data block:** Lấy dữ liệu từ tài nguyên đã tồn tại.
  + **Variable block:** Khai báo biến đầu vào cho cấu hình.
  + **Output block:** Xuất thông tin sau khi triển khai.
* **Provisioners:** Là các đoạn script hoặc lệnh được thực thi trên tài nguyên sau khi được tạo, dùng để cấu hình thêm hoặc chạy các tác vụ bổ sung. Provisioners thường được dùng khi không thể đạt được mục tiêu chỉ bằng các resource block.

### Quy trình sử dụng



Hình Quy trình hoạt động của terraform

Để triển khai và quản lý hạ tầng một cách tự động và chính xác, Terraform tuân theo một quy trình làm việc tuần tự gồm ba bước chính:

* **Write (Viết cấu hình):** Người dùng mô tả hạ tầng dưới dạng các tệp .tf, xác định loại tài nguyên, tham số cấu hình và mối quan hệ giữa chúng. Ví dụ: tạo một máy ảo trong mạng ảo được bảo vệ bởi security group và gắn IP công khai.
* **Plan (Lập kế hoạch):** Terraform đọc các tệp cấu hình và so sánh với trạng thái hiện tại của hạ tầng để đưa ra kế hoạch thay đổi (thêm, xóa, cập nhật). Tính năng này giúp người dùng dự đoán được kết quả trước khi thực hiện, hạn chế sai sót.
* **Apply (Thực thi):** Sau khi xác nhận, Terraform sẽ thực hiện các thay đổi theo đúng kế hoạch, đảm bảo xử lý theo thứ tự phụ thuộc tài nguyên.

### Các đặc điểm nổi bật

Những đặc điểm sau đây giúp Terraform trở thành một trong những giải pháp IaC được ưa chuộng và ứng dụng rộng rãi trong thực tiễn triển khai hạ tầng:

* **Hỗ trợ đa nền tảng:**

Terraform sử dụng các provider để giao tiếp với các API của nền tảng cloud hoặc dịch vụ bên ngoài. Mỗi provider định nghĩa các loại tài nguyên mà nó hỗ trợ, cho phép người dùng mở rộng Terraform để quản lý nhiều công nghệ khác nhau trong cùng một cấu hình.

* **Quản lý trạng thái:**

Terraform lưu trữ thông tin về hạ tầng thực tế trong file terraform.tfstate. Tệp này giúp Terraform theo dõi tài nguyên nào đã được triển khai, định danh cụ thể của chúng, từ đó dễ dàng xác định thay đổi khi cấu hình được cập nhật. Trạng thái có thể lưu cục bộ hoặc lưu trữ từ xa (backend như S3, GCS, Terraform Cloud) để phục vụ làm việc nhóm.

* **Khả năng tái sử dụng thông qua module:**

Terraform hỗ trợ **module hóa**, cho phép đóng gói các nhóm tài nguyên thành đơn vị tái sử dụng. Điều này giúp chuẩn hóa cấu hình, giảm trùng lặp mã, đồng thời nâng cao khả năng mở rộng và bảo trì hệ thống.

* **Hỗ trợ biến và đầu ra:**

Terraform cho phép sử dụng biến đầu vào (input variables) để tạo cấu hình linh hoạt và biến đầu ra (output values) để truyền thông tin giữa các module hoặc công cụ ngoài (như Ansible, Jenkins,...).

* **Cộng đồng lớn, tài liệu phong phú:**

Nhờ có cộng đồng phát triển rộng rãi và được sử dụng phổ biến trong ngành, Terraform có tài nguyên học tập, tài liệu hướng dẫn và module sẵn có vô cùng phong phú trên Terraform Registry.

## Amazon Web Services (AWS)

### Giới thiệu tổng quan

A logo of a company

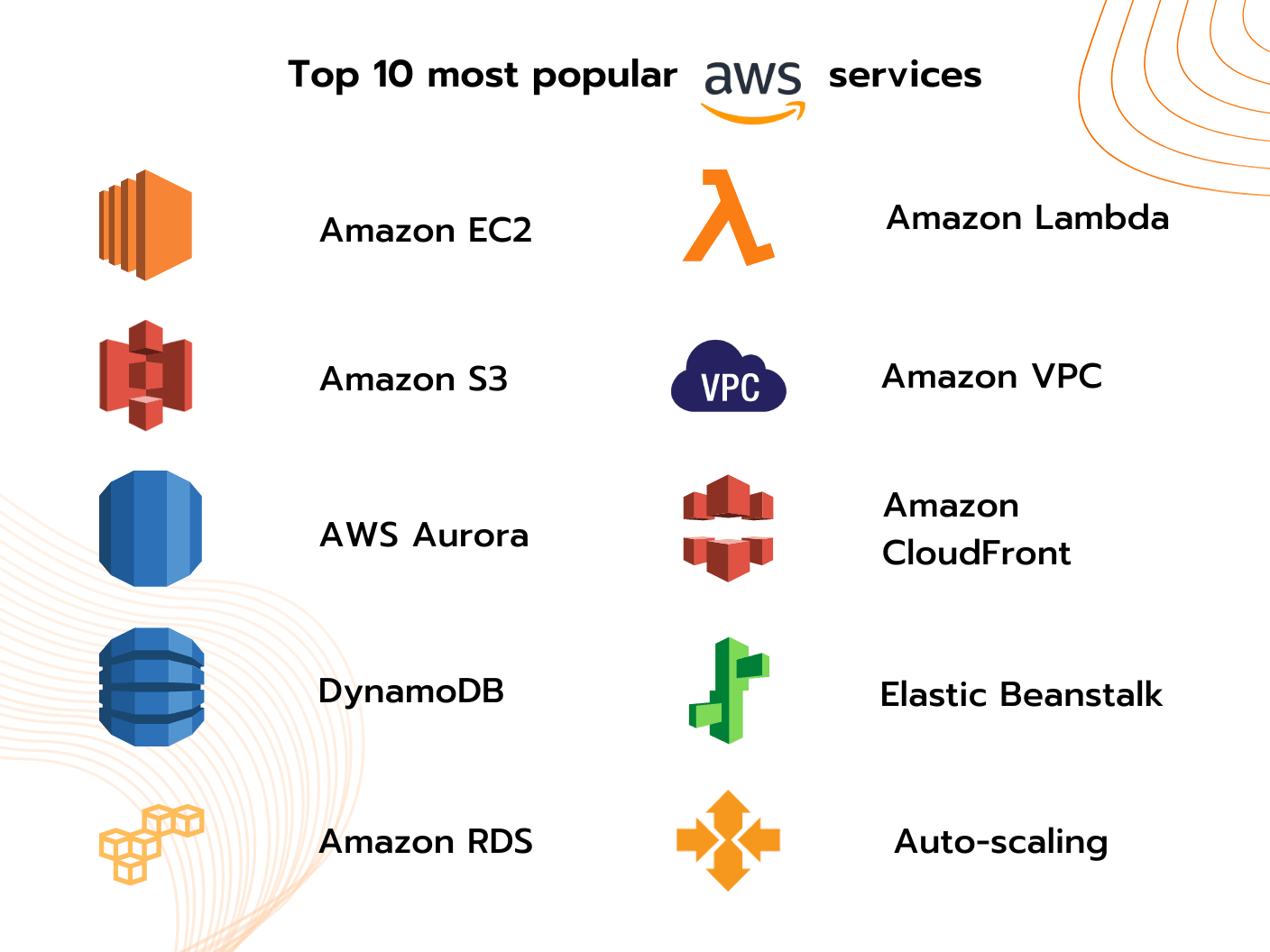
AI-generated content may be incorrect.

Hình Logo AWS

Amazon Web Services (AWS) là nền tảng điện toán đám mây hàng đầu do Amazon phát triển, cung cấp một hệ sinh thái dịch vụ phong phú trải dài trên nhiều lĩnh vực như tính toán, lưu trữ, mạng, cơ sở dữ liệu, bảo mật, trí tuệ nhân tạo và DevOps. AWS vận hành theo mô hình "trả phí theo mức sử dụng" (pay-as-you-go) và sở hữu cơ sở hạ tầng toàn cầu thông qua mạng lưới các region và availability zone, mang lại độ tin cậy và khả năng phục hồi cao cho hệ thống [3].

Đặc điểm nổi bật của AWS:

* **Độ tin cậy và bảo mật:** AWS cung cấp các lớp bảo mật vật lý và phần mềm, tuân thủ nhiều tiêu chuẩn quốc tế, tích hợp các công cụ kiểm soát truy cập, mã hóa dữ liệu, giám sát và cảnh báo bảo mật.
* **Khả năng mở rộng linh hoạt:** Người dùng có thể dễ dàng tăng giảm tài nguyên theo nhu cầu, phù hợp với cả doanh nghiệp nhỏ lẫn các tổ chức quy mô lớn.
* **Hệ sinh thái dịch vụ phong phú:** AWS hỗ trợ đầy đủ các dịch vụ từ hạ tầng cơ bản đến dịch vụ chuyên sâu, giúp doanh nghiệp xây dựng mọi loại ứng dụng và giải pháp công nghệ.
* **Tích hợp mạnh mẽ với IaC:** AWS hỗ trợ các công cụ quản lý hạ tầng bằng mã như Terraform, CloudFormation, giúp tự động hóa triển khai, quản lý và kiểm soát phiên bản tài nguyên.



Hình Các dịch vụ nổi tiếng của AWS

### Ứng dụng trong đồ án

Trong khuôn khổ đồ án, Amazon Web Services (AWS) được lựa chọn làm một trong hai nền tảng thử nghiệm để triển khai hạ tầng đám mây nhờ vào sự ổn định, khả năng mở rộng cao và mức độ tích hợp tốt với các công cụ IaC như Terraform. Mục tiêu chính là xây dựng hạ tầng mạng hoàn chỉnh gồm máy ảo, phân vùng mạng, các thành phần bảo mật và hệ thống địa chỉ IP công khai để kiểm tra tính đúng đắn và khả năng sinh mã tự động từ công cụ đề xuất.

Các thành phần chính trong hạ tầng AWS được sử dụng như sau:

* **aws\_instance**: Đại diện cho các máy chủ ảo EC2, đây là thành phần trung tâm thực hiện tác vụ tính toán. Một trong các instance có thể được cấu hình làm điểm trung gian truy cập (bastion host), đóng vai trò kết nối từ internet đến các tài nguyên trong mạng nội bộ.
* **aws\_vpc**: Tạo vùng mạng ảo riêng (VPC), nơi định nghĩa không gian mạng bao gồm các subnet, bảng định tuyến và các thành phần liên quan.
* **aws\_subnet**: Thiết lập các vùng mạng con trong VPC. Các subnet này được phân chia dựa trên vai trò truy cập – một số cho phép truy cập từ internet (public subnet), trong khi số khác phục vụ giao tiếp nội bộ (private subnet).
* **aws\_security\_group**: Cấu hình nhóm bảo mật đóng vai trò lọc lưu lượng mạng đến và đi từ các instance. Người dùng có thể chỉ định các quy tắc mở cổng tùy theo dịch vụ chạy trên máy ảo (SSH, HTTP, v.v.).
* **aws\_key\_pair**: Tạo và quản lý cặp khóa dùng cho truy cập SSH bảo mật đến các máy EC2, đảm bảo mỗi người dùng hoặc hệ thống có thể kiểm soát truy cập riêng biệt.
* **aws\_route\_table** & **aws\_route\_table\_association**: Thiết lập bảng định tuyến nhằm điều hướng lưu lượng giữa các subnet hoặc ra ngoài thông qua gateway.
* **aws\_internet\_gateway**: Kết nối VPC với internet, đóng vai trò cung cấp khả năng truy cập từ bên ngoài đến các subnet công khai.
* **aws\_nat\_gateway**: Được sử dụng để cho phép các máy ảo trong private subnet truy cập internet mà không cần gán địa chỉ IP công cộng. Điều này rất hữu ích cho các dịch vụ cần cập nhật phần mềm mà vẫn đảm bảo bảo mật.
* **aws\_eip**: Gán địa chỉ IP tĩnh công khai cho các instance cần khả năng truy cập ổn định từ bên ngoài, điển hình như các máy chủ đầu mối hoặc cổng giao tiếp của hệ thống.

## OpenStack

### Giới thiệu tổng quan

A logo with a black background

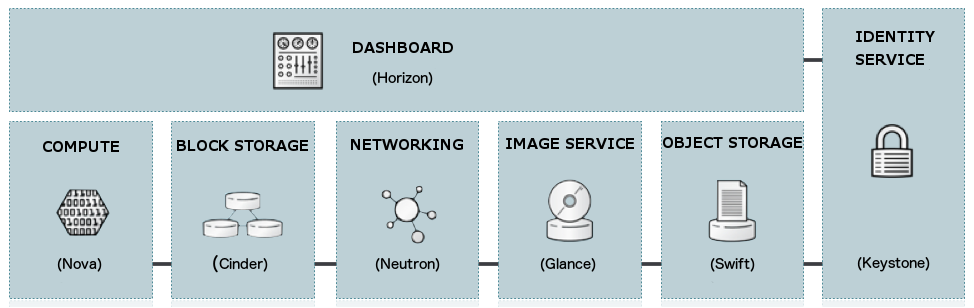
AI-generated content may be incorrect.

Hình Logo OpenStack

OpenStack là một nền tảng hạ tầng đám mây mã nguồn mở được thiết kế để triển khai và quản lý các môi trường điện toán đám mây theo mô hình IaaS (Infrastructure as a Service). Khác với các nền tảng thương mại như AWS hay Azure, OpenStack cho phép các tổ chức xây dựng hệ thống cloud riêng (private cloud) với mức độ kiểm soát, tùy biến và linh hoạt cao. Nền tảng này được duy trì và phát triển bởi cộng đồng quốc tế với sự tham gia của nhiều công ty lớn trong lĩnh vực công nghệ, như Red Hat, Canonical, IBM, và SUSE [4].

### Kiến trúc

Một trong những điểm mạnh của OpenStack là khả năng mở rộng theo chiều ngang và hỗ trợ đa dạng phần cứng. Do có kiến trúc dạng mô-đun, người dùng có thể triển khai linh hoạt các thành phần cần thiết, đồng thời dễ dàng tích hợp với các hệ thống hiện có hoặc mở rộng theo nhu cầu phát triển. Hơn nữa, việc không bị ràng buộc bởi nhà cung cấp cụ thể (vendor lock-in) giúp các tổ chức chủ động kiểm soát chi phí, bảo mật và chiến lược triển khai hạ tầng dài hạn.



Hình Các thành phần của OpenStack

**Các thành phần chính của OpenStack**

|  |  |
| --- | --- |
| Thành phần | Vai trò chính |
| Nova | Quản lý tài nguyên tính toán, tạo và vận hành máy chủ ảo (VM) |
| Neutron | Quản lý mạng ảo, subnet, router, firewall, cân bằng tải |
| Cinder | Cung cấp lưu trữ khối (block storage) cho máy ảo |
| Swift | Lưu trữ đối tượng (object storage) |
| Keystone | Quản lý xác thực, người dùng, nhóm, vai trò và quyền truy cập |
| Glance | Quản lý image máy ảo, hỗ trợ nhiều định dạng |
| Horizon | Giao diện web quản trị toàn bộ dịch vụ OpenStack |

Bảng Các thành phần của OpenStack

### Ứng dụng trong đồ án

Trong đồ án này, OpenStack được sử dụng để kiểm thử khả năng sinh mã Terraform cho môi trường đám mây riêng. Các tài nguyên được mô tả tương ứng với hạ tầng thực tế, bao gồm máy ảo, mạng ảo, bảo mật và định tuyến IP. Cụ thể, các thành phần chính được sử dụng bao gồm:

* **Openstack compute instance *(Nova)*:** Cung cấp và quản lý các máy chủ ảo, là thành phần thực hiện tác vụ tính toán trong hệ thống. Mỗi instance có thể được cấu hình theo nhu cầu như gán floating IP hoặc liên kết với keypair.
* **Openstack networking network và openstack networking subnet *(Neutron)*:** Xây dựng các mạng ảo và subnet tương ứng, cho phép phân chia hệ thống thành các phân vùng mạng độc lập.
* **Openstack networking router và openstack networking router interface**: Thiết lập các bộ định tuyến ảo giúp liên kết các subnet với nhau và định tuyến lưu lượng ra/vào hệ thống, đặc biệt khi cần kết nối đến mạng ngoài hoặc Internet thông qua floating IP.
* **Openstack networking floatingip:** Cấp phát và ánh xạ địa chỉ IP công khai đến các máy ảo hoặc router, đảm bảo khả năng truy cập từ bên ngoài đến các dịch vụ nội bộ.
* **Openstack compute keypair:** Quản lý cặp khóa SSH dùng để xác thực truy cập từ xa, đảm bảo an toàn khi vận hành các dịch vụ từ bên ngoài.
* **Openstack networking secgroup và openstack networking secgroup rule:** Định nghĩa và cấu hình các nhóm bảo mật, thiết lập chính sách kiểm soát truy cập dựa trên giao thức, cổng và IP. Đây là công cụ chính để bảo vệ các tài nguyên tính toán khỏi truy cập trái phép.

## Cloud Init

### Giới thiệu tổng quan

Cloud-Init là công cụ tiêu chuẩn trong môi trường điện toán đám mây để tự động hóa việc khởi tạo và cấu hình máy ảo ngay khi khởi động lần đầu. Công cụ này được tất cả các nhà cung cấp đám mây hàng đầu hỗ trợ và có mặt trong hầu hết các ảnh hệ điều hành Linux trên các nền tảng như AWS, Azure, OpenStack, KVM…. Khi máy ảo khởi động, Cloud-Init tự động phát hiện môi trường đám mây, truy xuất thông tin metadata và thực hiện các tác vụ khởi tạo hệ thống (cấu hình mạng, lưu trữ, khóa SSH, v.v.) theo dữ liệu đầu vào do người dùng hoặc nhà cung cấp đám mây cung cấp. Nhờ vậy, việc chuẩn bị máy chủ chuyển từ thao tác thủ công sang hoàn toàn tự động hóa, đảm bảo tính nhất quán và có thể tái sử dụng giữa các phiên bản máy ảo [5].

### Chức năng chính

Mục tiêu chính của công cụ này là tự động cấu hình hệ thống trong lần khởi động đầu tiên, giúp VM sẵn sàng sử dụng mà không cần thao tác thủ công. Cloud-init sử dụng các dữ liệu cấu hình như user-data, vendor-data và metadata để thực thi hàng loạt tác vụ thiết lập như sau:

* Cấu hình hệ thống mạng và hostname: Tự động đặt hostname, cấu hình địa chỉ IP (DHCP hoặc tĩnh) và các thiết lập mạng khác dựa trên thông tin metadata từ nhà cung cấp đám mây.
* Tạo người dùng và quản lý SSH: Tạo account người dùng mới, gán quyền sudo, và thêm khóa SSH công khai để cho phép truy cập từ xa an toàn không cần mật khẩu.
* Cài đặt phần mềm và khởi động dịch vụ: Tự động cài đặt các gói phần mềm cần thiết và kích hoạt các dịch vụ hệ thống theo chỉ định. Cloud-Init hỗ trợ liệt kê gói cài đặt hoặc chạy các script cài đặt do người dùng định nghĩa.
* Chạy script tùy biến: Thực thi các script do người dùng cung cấp (bằng shell script hoặc qua cloud-config) trong giai đoạn cuối của quá trình khởi động, giúp thực hiện các công việc tùy chỉnh ngoài khả năng cấu hình sẵn.
* Cấu hình lưu trữ: Cloud-Init có thể thiết lập phân vùng đĩa, định dạng và mount thêm ổ đĩa dữ liệu theo cấu hình. Ví dụ, các module disk\_setup và mounts có thể chia vùng, format đĩa và cập nhật /etc/fstab tự động.

### Các định dạng hỗ trợ

A screenshot of a computer program

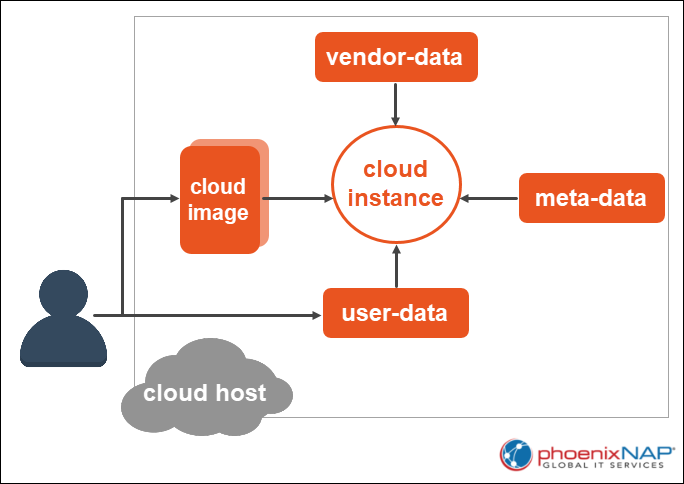
AI-generated content may be incorrect.

Hình Định dạng được cloud init hỗ trợ

Cloud-init hỗ trợ nhiều định dạng đầu vào (user-data) để tự động cấu hình hệ thống ngay từ lần khởi động đầu tiên của máy ảo. Các định dạng này được thiết kế nhằm đáp ứng các nhu cầu cấu hình từ đơn giản (chạy lệnh shell) đến phức tạp (ghép nhiều thành phần cấu hình), giúp người dùng có thể kiểm soát linh hoạt quá trình khởi tạo hệ thống. Các định dạng phổ biến bao gồm [6]:

* Shell script thuần túy: Tập tin kịch bản bắt đầu bằng #!/bin/sh hoặc #!/bin/bash. Được Cloud-Init chạy trong giai đoạn cuối (final stage) của khởi động, thường dùng để cài đặt phần mềm, ghi file cấu hình hoặc khởi động dịch vụ tùy ý.
* Cloud-config (YAML): Tập tin bắt đầu bằng dòng #cloud-config định dạng YAML. Đây là định dạng chính được hỗ trợ đầy đủ nhất. Cloud-config cho phép mô tả cấu hình mong muốn (hostname, người dùng, gói cần cài, khóa SSH, lệnh runcmd, v.v.) theo kiểu rõ ràng, dễ duy trì. Đây cũng là định dạng được khuyến nghị nhờ khả năng mở rộng và tái sử dụng cao.
* Cloud-boothook: Tập tin bắt đầu bằng dòng #cloud-boothook và thẻ shebang. Được chạy rất sớm trong quá trình khởi động (Network stage), trước cả khi thiết lập mạng hoàn tất. Thường dùng để cấu hình mạng, DNS, hoặc chỉnh sửa /etc/hosts trước khi các module khác của cloud-init chạy. Boothook chạy trên mọi lần khởi động của máy.
* MIME multi-part archive: Định dạng kết hợp nhiều phần (multipart/mixed). Có thể chứa nhiều phần khác nhau như shell script, cloud-config, boothook,… trong một tệp tin duy nhất. Mỗi phần đánh dấu bằng Content-Type tương ứng. Định dạng này hữu ích khi cần thực thi nhiều loại tác vụ phối hợp trong cùng lúc khởi động.
* Cloud-config archive: Định dạng YAML mở rộng của MIME multi-part. Tập tin bắt đầu bằng #cloud-config-archive và định nghĩa một danh sách các mục có trường type: và content: theo dạng YAML. Cho phép chỉ định nhiều phần cấu hình (ví dụ phần boot, phần config) trong một file dễ đọc hơn.
* Jinja template: Mẫu với header ## template: jinja có thể là cloud-config hoặc shell script, dùng cú pháp Jinja2 để chèn biến động từ metadata vào nội dung cấu hình. Dành cho trường hợp cần sinh nội dung linh động dựa trên thông tin instance.
* Include file: Tập tin bắt đầu bằng #include theo sau là danh sách URL. Cloud-Init sẽ tải nội dung từ các URL này và gộp vào cấu hình chính. Thích hợp khi muốn chia nhỏ cấu hình ra nhiều module hoặc tái sử dụng.
* Part handler: Tập tin bắt đầu bằng #part-handler chứa mã Python tùy chỉnh để xử lý các định dạng MIME không chuẩn. Cho phép mở rộng khả năng xử lý user-data bằng mã lập trình riêng.
* Gzip compressed content: Cloud-Init hỗ trợ tự động giải nén nội dung user-data nén gzip trước khi xử lý. Điều này hữu ích khi có giới hạn kích thước dữ liệu trên một số nền tảng đám mây.

### Dữ liệu đầu vào



Hình quy trình khởi tạo và cấu hình một cloud instance

Cloud-Init sử dụng ba nguồn dữ liệu chính từ môi trường đám mây để cấu hình máy chủ:

* User-data: Là dữ liệu cấu hình do người dùng cung cấp, bao gồm các định dạng cloud-config, shell scripts, YAML.. Nội dung này chỉ rõ các tác vụ cần thực hiện (cài gói, tạo user, thêm SSH key, chạy script…) và được Cloud-Init đọc để thiết lập hệ thống.
* Metadata: Là thông tin định danh của hệ thống do nền tảng đám mây cung cấp (qua metadata service hoặc config-drive). Metadata có thể gồm instance ID, hostname, địa chỉ IP, cấu hình mạng, và các thông tin riêng của nhà cung cấp. Cloud-Init sử dụng metadata để biết thông tin bản thân máy ảo và cách truy cập các nguồn dữ liệu khác.
* Vendor-data: Là dữ liệu tùy chỉnh do nhà cung cấp (hoặc tổ chức) cung cấp thêm cho máy ảo. Vendor-data thường chứa các cấu hình tối ưu hóa riêng cho hạ tầng (ví dụ, cài đặt phần cứng, tham số đặc thù) và được áp dụng sau metadata. Theo mặc định, vendor-data chỉ chạy ở lần boot đầu tiên của instance.

### Các giai đoạn thực thi

Cloud-Init được tích hợp chặt chẽ vào quy trình khởi động của hệ điều hành, chạy theo năm giai đoạn tuần tự (các stages) để thu thập dữ liệu và thiết lập cấu hình. Cụ thể [7]:

* Generator stage: Dùng systemd generator để quyết định có kích hoạt các service của cloud-init hay không. Nếu không có file /etc/cloud/cloud-init.disabled và dòng lệnh kernel không chứa cloud-init=disabled, thì cloud-init sẽ được bật để chạy.
* Local stage: Chạy rất sớm với phân vùng gốc đã mount sẵn, nhiệm vụ tìm nguồn dữ liệu (datasource) cục bộ và áp dụng cấu hình mạng ban đầu. Cloud-Init sẽ tìm thông tin mạng trong metadata hoặc dùng cơ chế DHCP fallback nếu không có metadata. Giai đoạn này đảm bảo rằng thiết bị mạng được cấu hình ngay từ đầu (xóa cấu hình cũ nếu có) trước khi tiếp tục khởi động.
* Network stage: Sau khi mạng cơ bản đã hoạt động, giai đoạn này chạy các module được liệt kê trong cloud\_init\_modules (trong /etc/cloud/cloud.cfg). Tại đây, Cloud-Init xử lý mọi user-data: bao gồm nạp các file bổ sung qua #include, giải nén nếu nén gzip, và chạy các phần handle (part-handler). Các module như disk\_setup và mounts cũng chạy trong giai đoạn này để phân vùng, format và mount ổ đĩa phụ ngay khi nhận được cấu hình từ metadata hoặc user-data.
* Config stage: Giai đoạn này chạy các module trong cloud\_config\_modules. Các tác vụ chủ yếu như cài đặt gói phần mềm, tạo người dùng, ghi file cấu hình được thực thi ở đây. Ví dụ, module runcmd sẽ thực hiện các lệnh do người dùng chỉ định trong cloud-config.
* Final stage: Cuối cùng, khi hệ thống cơ bản đã ổn định, cloud-init chạy cloud\_init\_modules và cloud\_final\_modules. Đây là giai đoạn chạy muộn nhất trong boot (tương đương sau rc.local truyền thống). Cloud-Init sẽ cài đặt các công cụ quản lý cấu hình (như Ansible, Puppet, Salt), hoặc thực thi các script của người dùng (shell script trong user-data). Đến giai đoạn này, mọi tác vụ khởi tạo đã hoàn tất, và máy ảo sẵn sàng cho hoạt động bình thường. (Cloud-Init cũng có thể tự khởi động lại hệ thống nếu được cấu hình).

### Các pha thực thi

A diagram of a cloud process

AI-generated content may be incorrect.

Hình Mối quan hệ giữa các giai đoạn khởi động (Stages) và các pha xử lý module (Phases)

Khi cloud-init chạy, nó sẽ thực thi các mô-đun trong cloud.cfg theo thứ tự trong ba giai đoạn:

* Pha mạng (Network Phase) – Các module trong cloud\_init\_modules: thực hiện các thiết lập cơ bản như thêm SSH key, đặt hostname ….
* Pha cấu hình (Configuration Phase) – Các module trong

cloud\_config\_modules: chạy các tác vụ như tạo người dùng, cài phần mềm, ghi file cấu hình từ user-data.

* Pha cuối (Final Phase) – Các module trong cloud\_final\_modules: giao tiếp với công cụ quản lý cấu hình bên ngoài (Ansible, Puppet...), và chạy các script do người dùng cung cấp.

Mỗi module có thể được thiết lập tần suất chạy:

* per-instance: chỉ chạy ở lần boot đầu tiên của instance.
* per-once: chỉ chạy một lần duy nhất kể cả khi clone instance.
* per-always: chạy mỗi lần boot.

### Ứng dụng trong đồ án

Trong đồ án này, Cloud init được sử dụng như một công cụ chủ lực để thiết lập môi trường hệ điều hành ban đầu cho từng máy ảo, đảm bảo rằng mọi instance sau khi được khởi tạo đều sẵn sàng hoạt động mà không cần can thiệp thủ công. Cụ thể, Cloud init được dùng để:

* **Tạo người dùng tùy chỉnh** có quyền truy cập từ xa thông qua SSH, phục vụ cho mục đích quản trị và triển khai.
* **Thêm khóa SSH công khai** vào máy ảo, đảm bảo kết nối an toàn và không cần mật khẩu.
* **Cấu hình các tham số hệ thống** như bật mô-đun IP forwarding, vô hiệu hóa swap, cấu hình sysctl, hoặc tinh chỉnh tường lửa, nhằm đáp ứng các yêu cầu kỹ thuật đặc thù của Kubernetes.
* **Cài đặt các gói phần mềm cần thiết**, chẳng hạn như Docker, containerd và các tiện ích mạng.
* **Thực thi script tham gia cụm Kubernetes**, giúp node sẵn sàng tích hợp vào hệ thống phân tán ngay sau khi khởi tạo.

# Phân tích thiết kế hệ thống

## Kiến trúc

A diagram of a process

AI-generated content may be incorrect.

Hình Kiến trúc Terraform Generator

Terraform Generator là một framework tự động hóa việc sinh mã và triển khai hạ tầng cloud. Hệ thống được thiết kế theo kiến trúc thành phần (Component - Based), bao gồm 4 thành phần chính hoạt động phối hợp để chuyển đổi từ mô tả topology JSON thành hạ tầng cloud thực tế. Chi tiết các thành phần kiến trúc:

**Dashboard (Giao diện người dùng)**

Là giao diện trực quan giúp người dùng dễ dàng thao tác, quản lý toàn bộ hệ thống triển khai hạ tầng. Được xây dựng bằng ReactJS, Dashboard tập trung vào chức năng triển khai và quản lý các dự án.

**Code Generation (Sinh mã tự động)**

Đóng vai trò là bộ xử lý trung tâm, chịu trách nhiệm chuyển đổi file topology JSON thành mã HCL, tự động thực thi quá trình triển khai. Thành phần này bao gồm các file chính như generate.py, terraform\_templates.py và run\_terraform.py.

**Validation and Mapping (Kiểm tra và ánh xạ)**

Thực hiện kiểm tra tính hợp lệ của cấu trúc topology, xác thực logic mạng và đối chiếu tài nguyên khai báo với các cloud provider. Các tác vụ được xử lý thông qua các script: validate\_json.py, validate\_aws.py và validate\_openstack.py.

**State Management (Quản lý trạng thái)**

Theo dõi, lưu trữ và khôi phục trạng thái hạ tầng sau mỗi lần triển khai. Đồng thời, hệ thống cũng thực hiện khóa trạng thái để tránh xung đột truy cập khi có nhiều quá trình hoạt động song song. Trạng thái được lưu qua các tệp như .tfstate, .tfstate.backup và .terraform.lock.hcl.

## File json mô tả hệ thống

Để đảm bảo khả năng sinh mã Terraform tự động, công cụ được thiết kế để nhận đầu vào là một **file JSON mô tả toàn bộ topology hệ thống**. Cấu trúc JSON này thể hiện rõ các thành phần hạ tầng máy ảo (instances), mạng (networks) và router, cho phép trích xuất dữ liệu và ánh xạ sang các resource tương ứng trên nền tảng AWS hoặc OpenStack. Dưới đây là chi tiết:

**Instances:** Danh sách các máy ảo cần triển khai. Mỗi máy ảo bao gồm các thuộc tính sau:

|  |  |
| --- | --- |
| Trường | Ý nghĩa |
| name | Tên định danh duy nhất cho mỗi máy ảo. |
| image | Hệ điều hành để khởi tạo máy ảo, ví dụ: ubuntu-jammy. |
| cpu, ram | Số lượng vCPU và dung lượng RAM (GB) cấp phát cho máy ảo. |
| disk | Dung lượng ổ cứng gắn kèm máy ảo (GB). |
| cloud\_init | Tên file cấu hình cloud init. |
| networks | Danh sách mạng mà máy ảo kết nối tới. Mỗi mạng có tên và địa chỉ IP tĩnh tương ứng. |
| keypair | (Tuỳ chọn) Tên keypair được gán vào máy ảo. |
| security\_groups | (Tuỳ chọn) Danh sách Security Group áp dụng cho máy ảo. |
| floating\_ip | (Tuỳ chọn) Ip hoặc Boolean, địa chỉ ip public cho máy ảo . |

Bảng Thuộc tính của instance

**Networks:** Xác định cấu trúc mạng cho hệ thống, gồm các subnet và thông tin cấu hình chi tiết:

|  |  |
| --- | --- |
| Trường | Ý nghĩa |
| name | Tên định danh mạng. |
| cidr | Dải địa chỉ IP theo định dạng CIDR (ví dụ: 192.168.1.0/24) |
| gateway\_ip | Địa chỉ IP cổng mặc định của mạng. |
| enable\_dhcp | Boolean, cấp phát cấu hình mạng tự động cho máy ảo. |
| pool | Dải IP của cidr nhé |

Bảng Thuộc tính của network

**Routers** Thiết lập các router để định tuyến giữa các mạng nội bộ và kết nối ra ngoài:

|  |  |
| --- | --- |
| Trường | Ý nghĩa |
| name | Tên định danh router. |
| networks | Danh sách các mạng router kết nối, mỗi mục có name và địa chỉ IP gateway của mạng đó. |
| external | Boolean, router kết nối ra ngoài internet hoặc không. |
| routes | Danh sách các route tĩnh bổ sung. |

Bảng Thuộc tính của router

## Thư mục mẫu cấu hình hạ tầng

|  |  |
| --- | --- |
| A screenshot of a computer  AI-generated content may be incorrect. |  |

Hình Cấu trúc module xử lý của AWS và Openstack

Để tổ chức mã nguồn rõ ràng, dễ bảo trì và mở rộng, hệ thống sinh mã Terraform được chia thành các tệp và thư mục chức năng. Mỗi thành phần trong dự án đảm nhiệm một vai trò cụ thể, từ xử lý input đầu vào, khai báo tài nguyên đến lưu trữ trạng thái triển khai.

A screen shot of a computer

AI-generated content may be incorrect.

Hình Vị trí thư mục mẫu trong terraform generatỏ

Các thư mục như aws/ và openstack/ trong terraform-generator vừa đóng vai trò là **thư mục template mẫu** cho từng provider, vừa có cấu trúc giống hệt một thư mục kết quả sau khi sinh môi trường. Khi chạy script generator, nội dung của các thư mục này sẽ được copy sang thư mục kết quả, chỉ riêng file main.tf là được sinh động theo topology và resource thực tế.

**Các thành phần trong 1 thư mục cấu hình hạ tầng.**

**1. main.tf**

Đây là file cấu hình chính của Terraform, đóng vai trò trung tâm trong việc điều phối toàn bộ quá trình triển khai hạ tầng.

Trong tệp này:

* Provider (ví dụ: AWS) được khai báo để xác định nền tảng đám mây mục tiêu.
* Cấu trúc dữ liệu từ file topology.json được load vào và xử lý nhằm sinh mã Terraform tự động.
* Các module network, security\_groups, instance được gọi dùng dữ liệu từ topology.json để tạo các resource.

main.tf định nghĩa luồng triển khai từ network → security group → instance, đảm bảo tính tuần tự và chính xác về phụ thuộc tài nguyên.

**2. variables.tf**

Tệp này khai báo các biến được sử dụng trong toàn bộ dự án Terraform.

Các biến có thể bao gồm: vùng triển khai (region), CIDR block của VPC, tên subnet, danh sách security group, v.v.

Việc tách biến ra khỏi logic chính giúp tăng khả năng tái sử dụng và tùy biến linh hoạt.

**3. outputs.tf**

File này định nghĩa các giá trị đầu ra (output) sau khi quá trình triển khai hoàn tất.

Ví dụ: địa chỉ IP công khai (public IP), địa chỉ IP nội bộ, ID của các máy ảo…

Các output này giúp người dùng nhanh chóng lấy thông tin để truy cập hệ thống, tích hợp vào pipeline CI/CD, hoặc phục vụ mục đích giám sát.

**4. terraform.tfvars**

Là tệp chứa giá trị cụ thể của các biến đã định nghĩa trong variables.tf.

Dùng để cung cấp cấu hình thực tế cho môi trường triển khai (dev, staging, prod), tách biệt khỏi phần logic để dễ kiểm soát.

Nhờ đó, có thể triển khai cùng một mô hình hạ tầng trên nhiều môi trường khác nhau chỉ bằng cách thay đổi nội dung file này.

**5. terraform.tfstate và terraform.tfstate.backup**

Đây là các tệp trạng thái nội bộ của Terraform, được tự động tạo và cập nhật sau mỗi lần thực hiện apply.

Chúng ghi lại trạng thái hiện tại của toàn bộ hạ tầng đã được triển khai, giúp Terraform tính toán sự khác biệt khi có thay đổi mới.

**6. topology.json**

Là file JSON đầu vào mô tả kiến trúc hệ thống, bao gồm thông tin về máy ảo, mạng, router, bảo mật, địa chỉ IP, keypair, v.v.

Main.tf sẽ đọc file này, ánh xạ các trường tương ứng để sinh mã Terraform phù hợp cho từng nền tảng.

**7. cloud\_init/ (thư mục)**

Chứa các tệp cloud init script ở định dạng YAML hoặc shell script được instance sử dụng.

**8. modules/ (thư mục)**

Thư mục modules/ chứa các module con, mỗi module đại diện cho một nhóm tài nguyên hạ tầng cụ thể. Việc phân chia theo module giúp tổ chức mã rõ ràng, dễ mở rộng và tái sử dụng trong nhiều môi trường triển khai khác nhau.

Với nền tảng **AWS**:

* network/: Tạo và cấu hình VPC, subnet, route table, Internet Gateway, NAT Gateway và Elastic IP.
* instance/: Khởi tạo EC2 instance, gán subnet, keypair, cloud init và security group tương ứng.
* security\_groups/: Định nghĩa các security group và các rule cho phép hoặc chặn lưu lượng mạng.
* keypair/: (Tùy chọn) Tạo SSH keypair sử dụng để truy cập EC2 thông qua giao thức SSH.

Với Openstack

* network/: Tạo và cấu hình các network, subnet, router, và router interface trên OpenStack. Đảm bảo các máy ảo có thể kết nối nội bộ và ra ngoài Internet thông qua router (external gateway).
* instance/: Khởi tạo các máy ảo trên OpenStack, gán vào các mạng đã tạo, cấu hình cloud init, gán keypair SSH, gán security group, và cấp phát floating IP nếu cần.
* security\_group/: Định nghĩa cácsecurity group và các rule ingress/egress để kiểm soát lưu lượng vào/ra các máy ảo, ví dụ: cho phép SSH, HTTP, HTTPS...
* keypair/: Quản lý việc tạo và gán SSH keypair cho các máy ảo, đảm bảo truy cập an toàn qua SSH.

## Cấu trúc mã nguồn hệ thống Terraform Generator

A screenshot of a computer

AI-generated content may be incorrect.

Hình Cấu trúc terraform generator

Hệ thống Terraform Generator được tổ chức thành nhiều thành phần riêng biệt, mỗi thành phần đảm nhận một chức năng cụ thể, tạo thành một quy trình tự động hóa triển khai hạ tầng hoàn chỉnh. Cấu trúc mã nguồn bao gồm:

**1. Script điều phối chính**

**generate.py**

* Đọc file topology.json.
* Gọi validate\_json.py để kiểm tra tính hợp lệ cấu trúc JSON.
* Gọi validate\_openstack.py hoặc validate\_aws.py để xác thực tài nguyên cloud.
* Gọi terraform\_templates.py để sinh file main.tf.
* Gọi run\_terraform.py để thực thi terraform apply triển khai hạ tầng.
* Hỗ trợ sinh nhiều bản sao cấu hình (theo số lượng N đầu vào).

**2. Các module kiểm tra và ánh xạ tài nguyên**

**validate\_json.py**

* Kiểm tra định dạng schema của topology.json.
* Đảm bảo đúng kiểu dữ liệu, dải IP hợp lệ, các trường bắt buộc.
* Trả lỗi cụ thể nếu có sai sót trong khai báo kiến trúc.

**validate\_openstack.py**

* Kiểm tra các tài nguyên thực tế trên OpenStack như: image, flavor, network,...
* Ánh xạ cpu, ram, disk → flavor; kiểm tra image tồn tại hay không.
* Trả về thông tin ánh xạ để dùng cho sinh mã Terraform.

**validate\_aws.py**

* Tương tự như OpenStack nhưng kiểm tra tài nguyên trên AWS: AMI, instance type,...
* Trả về các giá trị ánh xạ thực tế như ami\_id, instance\_type.

**3. Module sinh mã Terraform**

**terraform\_templates.py**

* Chứa các template để sinh các khối mã cấu hình Terraform như: provider, resource (network, subnet, instance, security group,...).
* Sinh nội dung file main.tf phù hợp với dữ liệu đã ánh xạ và validate.

**4. Module thực thi Terraform**

**run\_terraform.py**

* run\_terraform.py được thiết kế để duyệt qua tất cả các thư mục con (mỗi thư mục là một môi trường Terraform riêng biệt).
* Với mỗi thư mục con, script này sẽ thực thi lệnh Terraform tương ứng (terraform apply, terraform destroy, hoặc terraform plan) một cách tự động.
* Nhờ đó, chỉ cần chạy một lệnh duy nhất, toàn bộ các môi trường sẽ được triển khai/xóa/kiểm tra song song mà không cần thao tác thủ công từng thư mục.

## Luồng hoạt động

A diagram of a software process

AI-generated content may be incorrect.

Hình Luồng hoạt động của terraform generator

**Quy trình tạo và triển khai hạ tầng từ file kiến trúc JSON:**

**Chuẩn bị dữ liệu kiến trúc (topology.json)**

Người dùng cung cấp một tệp topology.json mô tả toàn bộ kiến trúc hạ tầng cần triển khai, bao gồm: máy ảo, mạng, router, địa chỉ IP, nhóm bảo mật, khóa SSH,   
Tệp JSON này đóng vai trò là nguồn dữ liệu duy nhất, cho phép mô hình hóa hệ thống một cách linh hoạt, thống nhất và có thể mở rộng.

**2. Thực thi script sinh mã Terraform**

Lệnh thực thi: python3 generate.py [OpenStack|AWS] [N]

Trong đó:

* [OpenStack|AWS] là nền tảng mục tiêu (cloud provider).
* [N] là số lượng bản sao (số lượng môi trường hạ tầng được tạo từ cùng một kiến trúc).

Script generate.py đảm nhận vai trò là **bộ sinh mã (Infrastructure Generator)**, điều phối toàn bộ quy trình kiểm tra, xác thực và sinh các tệp cấu hình Terraform.

**3. Kiểm tra hợp lệ kiến trúc đầu vào (Validation)**

Trước khi sinh mã, script tiến hành xác thực tệp topology.json theo các tiêu chí:

* Đúng định dạng (theo JSON Schema).
* Đúng kiểu dữ liệu cho từng trường.
* Logic hợp lệ (ví dụ: không trùng địa chỉ IP, subnet đúng với CIDR,...).  
  Nếu phát hiện sai sót, chương trình sẽ dừng và hiển thị thông báo lỗi chi tiết để người dùng hiệu chỉnh kịp thời, đảm bảo không sinh ra cấu hình sai.

**4. Kiểm tra tính khả dụng của tài nguyên (Mapping & Resource Validation)**

Sau khi xác thực cấu trúc tổng thể của file topology.json, script tiến hành kiểm tra và ánh xạ các thông tin phần cứng như cpu, ram, disk, image sang các tài nguyên thực tế trên nền tảng cloud tương ứng.

* **Đối với OpenStack**: Trường image được ánh xạ sang tên image thực tế đang tồn tại trên hệ thống. Đồng thời, cpu, ram và disk được dùng để lựa chọn flavor phù hợp, đảm bảo cấu hình máy ảo đúng theo yêu cầu người dùng.
* **Đối với AWS**: image được ánh xạ sang Amazon Machine Image (AMI), còn cpu và ram được sử dụng để xác định instance type tương ứng.

**5. Sinh mã cấu hình Terraform**

Sau khi xác nhận toàn bộ dữ liệu đầu vào hợp lệ, script sẽ tự động sử dụng các mẫu cấu hình (template) đã định nghĩa để sinh ra mã Terraform:

* Các tệp main.tf tương ứng với dữ liệu đầu vào và provider.
* Cấu trúc thư mục được tạo hoàn chỉnh, phân tách rõ ràng theo module.

**Cấu trúc thư mục được tạo:**

Thư mục cha: terraform-projects/{provider}\_{timestamp}/

* provider: Tên cloud provider (openstack/aws)
* timestamp: Thời gian tạo (YYYYMMDD\_HHMMSS)

Thư mục con: {provider}\_{random\_suffix}/

* Mỗi thư mục con chứa một môi trường hạ tầng hoàn chỉnh
* random\_suffix: Chuỗi 6 ký tự ngẫu nhiên đảm bảo tính duy nhất

Việc sinh mã hoàn toàn tự động và đảm bảo nội dung các tệp cấu hình phản ánh chính xác kiến trúc đã được định nghĩa trong topology.json.

**6. Triển khai hạ tầng (Terraform Apply)**

Sau khi sinh tệp, script sẽ tự động thực thi lệnh terraform apply để triển khai hạ tầng trên nền tảng đã chọn (AWS hoặc OpenStack). Quá trình này hoàn toàn tự động, không yêu cầu thêm bất kỳ thao tác thủ công nào từ người dùng.

# Hiện thực hệ thống

## Tải mã nguồn từ github

Mã nguồn của công cụ Terraform Generator được lưu trữ trên GitHub. Để bắt đầu sử dụng, người dùng thực hiện thao tác clone:

git clone <https://github.com/toannd021104/DACN.git>

## Cài đặt công cụ và thư viện cần thiết

Hệ thống yêu cầu môi trường Python, Terraform, và một số công cụ dòng lệnh phù hợp với nền tảng cloud đang sử dụng. Việc cài đặt được thực hiện theo các bước dưới đây:

**Python 3 và pip**

sudo apt update

sudo apt install python3 python3-pip -y

**Terraform CLI**

Wget https://releases.hashicorp.com/terraform/1.6.6/terraform\_1.6.6\_linux\_amd64.zip

unzip terraform\_1.6.6\_linux\_amd64.zip

sudo mv terraform /usr/local/bin/

**OpenStack CLI**

pip install python-openstackclient

**AWS CLI**

pip install awscli

## Chuẩn bị dữ liệu xác thực và cấu hình:

Để có thể triển khai hạ tầng trên nền tảng cloud, người dùng cần cung cấp thông tin đăng nhập, xác thực. Các thông tin này được khai báo trong file terraform.tfvars – đây là nơi lưu trữ biến thực tế cho provider.

A black rectangular object with white lines

AI-generated content may be incorrect.

Hình Cấu hình tfvar cho openstack

## Triển khai và thực thi script

Sau khi chuẩn bị đầy đủ các thành phần cần thiết, người dùng có thể thực hiện quy trình sinh mã và triển khai hạ tầng thực tế bằng cách sử dụng script chính của hệ thống là generate.py.

Script này cho phép người dùng khai báo nền tảng (AWS hoặc OpenStack) và số lượng bản sao cần tạo, sau đó tự động thực hiện toàn bộ quá trình kiểm tra, sinh file Terraform và chạy lệnh terraform apply.

python3 generate.py [provider] [num\_copies]

Trong đó:

* [OpenStack|AWS] là nền tảng mục tiêu (cloud provider).
* [N] là số lượng bản sao (số lượng môi trường được tạo từ cùng một kiến trúc).

# Thực nghiệm và đánh giá

## Kịch bản 1: Tạo topology openstack 2 bản sao

**Mục tiêu:** Kịch bản này minh họa khả năng triển khai một hạ tầng mạng đơn giản trên nền tảng OpenStack, bao gồm máy ảo, mạng ảo, cấu hình truy cập, định tuyến và kết nối ra Internet.

Hệ thống định nghĩa hai máy ảo:

* vm1: Máy ảo Ubuntu với 2 CPU, 4GB RAM, 20GB disk, kết nối mạng net1 (IP: 192.168.1.10), có floating IP tự động và security group ssh-sg
* s2: Máy ảo tương tự vm1 nhưng kết nối mạng net2 (IP: 192.168.2.10), không có floating IP

Cả hai máy đều sử dụng cloud-init.yaml để tự động cấu hình user, tải các gói phần mềm và keypair "toanndcloud-keypair" để truy cập SSH.

Hệ thống gồm ba mạng riêng biệt:

* net1: Dải IP 192.168.1.0/24, gateway 192.168.1.1
* net2: Dải IP 192.168.2.0/24, gateway 192.168.2.1
* net3: Dải IP 192.168.3.0/24, gateway 192.168.3.1

Tất cả đều bật DHCP và có pool IP trống.

Một router R1 kết nối tất cả ba mạng:

* Interface net1: 192.168.1.1
* Interface net2: 192.168.2.1
* Interface net3: 192.168.3.1

Router có kết nối external để truy cập Internet

A screenshot of a computer

AI-generated content may be incorrect.

Hình Kết quả generate

Chạy lệnh “python3 generate.py openstack 2”, tự động sinh ra 2 thư mục cấu hình hạ tầng OpenStack độc lập dựa trên file topology.json và tự động apply tài nguyên bằng Terraform. Hệ thống tạo ra một thư mục cha mới trong terraform-project/, Bên trong thư mục cha này có 2 thư mục con, mỗi thư mục con chứa đầy đủ cấu hình hạ tầng :

terraform-projects/openstack\_20250705\_154226/

openstack\_7cd446/

openstack\_acd4a4/

A screenshot of a computer

AI-generated content may be incorrect.

Hình Thư mục được tạo trong terraform project

Sau khi các thư mục cấu hình hạ tầng được sinh ra đầy đủ, script run\_terraform.py sẽ được tự động gọi để thực thi lệnh **terraform apply** trên từng thư mục con.A computer screen with text

AI-generated content may be incorrect.

Hình Apply tự động các cấu hình

Sau khi thực thi lệnh generate.py với provider là OpenStack, quá trình sinh mã và triển khai hoàn tất thành công. Hệ thống đã khởi tạo **4 máy ảo** với các thuộc tính đúng theo mô hình được khai báo trong file topology.json.

A screenshot of a computer

AI-generated content may be incorrect.

Hình Các máy ảo được tạo ra

A screenshot of a computer

AI-generated content may be incorrect.

Hình Thuộc tính của vm1

Cloud init đã thực thi thành công trên máy ảo. User đã được tạo đúng theo cấu hình trong file cloud init. Các gói phần mềm khai báo trong cloud init ( docker, git, ...) đã được cài đặt thành công.

A screen shot of a computer

AI-generated content may be incorrect.

Hình Cloud init thành công

**Tổng cộng 6 mạng ảo (network)** đã được tạo:

* Mỗi mạng được khai báo với tên riêng, dải địa chỉ IP (CIDR), gateway và cài đặt DHCP bật hoặc tắt.
* Các mạng được gắn đúng vào các máy ảo tương ứng theo cấu hình.
* Mỗi mạng kèm theo **subnet** tương ứng.

A screenshot of a cell phone

AI-generated content may be incorrect.

Hình Network được tạo

A screenshot of a computer

AI-generated content may be incorrect.

Hình Subnet tương ứng với network

**2 router ảo** cũng được triển khai:

* Mỗi router đảm nhận kết nối cho một nhóm mạng cụ thể.
* Tất cả router đều được cấu hình gateway ra ngoài v có trường external = true.
* Các kết nối giữa router và mạng được ánh xạ chính xác theo mô hình, đảm bảo khả năng truyền thông liên vùng.



Hình Router được tạo

A screenshot of a computer

AI-generated content may be incorrect.

Hình Router có interface tương ứng

A colorful lines with black text

AI-generated content may be incorrect.

Hình Topology tạo ra đúng với định nghĩa

## Kịch bản 2: Tạo topology aws 2 bản sao

**Mục tiêu:** Kịch bản này nhằm minh họa khả năng triển khai một hạ tầng đơn giản trên nền tảng AWS, sử dụng cùng phương pháp sinh mã tự động như với OpenStack. Mục tiêu chính là kiểm thử tính tương thích và khả năng sinh mã Terraform cho môi trường AWS.

Mô hình hạ tầng được thiết kế tương tự với kịch bản OpenStack, nhưng có một số điều chỉnh để phù hợp với đặc thù của AWS:

* Security Group đơn giản hóa: Sử dụng **allow-all-sg** mở toàn bộ các cổng mạng, giúp đơn giản hóa quá trình kiểm thử và đảm bảo truy cập dễ dàng trong môi trường lab.
* Tự động cấp phát Public IP: Thuộc tính floating\_ip được gán giá trị “**True**”, cho phép instance EC2 nhận địa chỉ IP công khai từ EIP.
* Cloud-init đơn giản: Tập tin cloud-init.yaml được sử dụng để cài đặt nhanh các tiện ích mạng và tạo các tệp xác nhận việc triển khai thành công.

A screenshot of a computer

AI-generated content may be incorrect.

Kết quả sau khi chạy lệnh “python3 generate.py aws 2”, hệ thống đã tiến hành sinh mã Terraform thành công 2 thư mục cấu hình hạ tầng và triển khai trên nền tảng Amazon Web Services (AWS).

Mỗi môi trường được tạo ra sẽ có cấu trúc hạ tầng giống nhau, bao gồm:

* **3 máy ảo (EC2 instance)**:
  + **2 máy ảo ứng dụng chính**, được định nghĩa từ file topology.json với các thuộc tính cụ thể như ami, instance type, IP tĩnh trong private subnet.
  + **1 máy bastion host tạo ở public subnet**, tự động được sinh ra để phục vụ mục đích truy cập SSH vào các máy ảo nằm trong private subnet.

A screenshot of a computer

AI-generated content may be incorrect.

Hình Máy ảo được tạo

A computer screen shot of a computer

AI-generated content may be incorrect.

Hình Cloud init trên vm1 được thực hiện thành công

* **3 private subnet và 1 public subnet**:
  + Mỗi private subnet tương ứng với một network đã được định nghĩa trong topology.
  + Public subnet được tự động tạo để Triển khai bastion host, cho phép SSH từ bên ngoài vào hệ thống cũng như Gắn NAT Gateway, giúp các máy ảo trong private subnet có thể truy cập Internet một cách an toàn.
  + Các subnet này nằm trong cùng một VPC, nhưng được phân tách IP rõ ràng, hỗ trợ cách ly giữa các thành phần.

A screenshot of a computer

AI-generated content may be incorrect.

Hình Các subnet được tạo

* **NAT Gateway**
  + Bởi vì router cho phép ra mạng nên NAT Gateway được cài đặt giúp máy ảo trong private subnet có thể ra ngoài mạng.
  + NAT Gateway hoạt động kết hợp với route table, cho phép các subnet riêng truy cập Internet một chiều.

A screenshot of a computer

AI-generated content may be incorrect.

Hình NAT Gateway được tạo

* **Elastic IP**
  + Giúp cấp phát địa chỉ ip public tĩnh tránh thay đổi mỗi khi khởi tạo lại. Elastic IP được tạo vì floating ip được gán giá trị true
  + Ngoài ra Elastic IP được cấp và gán cho NAT Gateway, đảm bảo có địa chỉ IP tĩnh để định tuyến ra ngoài.

A screenshot of a computer

AI-generated content may be incorrect.

Hình Elastic IP được tạo

## Kịch bản 3: Tạo cụm kubernetest 1 master 1 worker trên AWS

**Mục tiêu:** Kịch bản này minh họa khả năng triển khai một cụm Kubernetes hoàn chỉnh và tự động trên cloud, bao gồm việc cấu hình master node, worker node và thiết lập mạng container networking, tất cả được thực hiện thông qua cloud-init automation [8].

Cấu hình máy ảo:

* vm1 (Master node): Máy ảo Ubuntu với 2 CPU, 4GB RAM, 20GB disk, địa chỉ IP tĩnh 192.168.1.10, sử dụng cloud-init "kubernetes-master.yaml"
* worker: Máy ảo Worker với cấu hình tương tự master, địa chỉ IP tĩnh 192.168.1.11, sử dụng cloud-init "kubernetes-worker.yaml"

Cấu hình mạng:

* net1: Mạng nội bộ với dải IP 192.168.1.0/24, gateway 192.168.1.1, bật DHCP để nhận cấu hình mạng tự động.

Cấu hình router:

* Router R1: Là cầu nối giữa mạng nội bộ net1 và mạng external, đảm bảo kết nối Internet cho các máy ảo.

**Tự động hóa Master Node**: Quá trình cấu hình master node được thực hiện hoàn toàn tự động thông qua kubernetes-master.yaml:

* Thiết lập hostname và hosts file để nhận diện các node
* Cài đặt Docker engine và Kubernetes components (kubelet, kubeadm, kubectl)
* Cấu hình containerd với systemd cgroup driver
* Khởi tạo cụm Kubernetes với kubeadm init
* Thiết lập kubeconfig cho cả root và user devops
* Cài đặt Calico CNI để quản lý mạng pod
* Sinh lệnh join command và lưu vào file để worker node sử dụng

**Tự động hóa Worker Node**: Worker node được cấu hình tối giản nhưng đầy đủ để tham gia cluster:

* Cài đặt Docker và Kubernetes runtime components (kubelet, kubeadm)
* Cấu hình containerd tương tự master
* Chuẩn bị môi trường để nhận lệnh join từ master
* Tắt swap và load các kernel module cần thiết cho Kubernetes

Sau khi cloud init thực thi trên node master, toàn bộ quá trình cài đặt Docker, Kubernetes, cấu hình mạng, khởi tạo cluster với kubeadm init và cài đặt Calico CNI đều hoàn tất tự động. User devops được tạo sẵn với quyền sudo và SSH key, giúp dễ dàng quản lý. File /join-command.sh trên master chứa lệnh join dành cho các node worker.

A screenshot of a computer program

AI-generated content may be incorrect.

Hình Cloud init output trên master

Node worker cũng được cloud init tự động cài đặt Docker, Kubernetes, cấu hình containerd, tắt swap, nạp kernel module cần thiết. Sau khi hoàn tất, worker ở trạng thái sẵn sàng, chỉ cần thực hiện lệnh join do master cung cấp để tham gia vào cụm Kubernetes.

A computer screen with white text

AI-generated content may be incorrect.

Hình Node worker

Kiểm tra trạng thái cluster trên master

A screen shot of a computer program

AI-generated content may be incorrect.

Hình Worker join cluster thành công

## Kịch bản 4: Demo quản lý hạ tầng bằng giao diện

**Mục tiêu:** Trong kịch bản này, việc triển khai và quản lý hạ tầng AWS hoặc OpenStack được thực hiện hoàn toàn qua giao diện web thay vì dòng lệnh, nhằm đơn giản hóa thao tác và nâng cao trải nghiệm người dùng.

**Công nghệ sử dụng**

* **Frontend:** ReactJS
* **Backend:** FastAPI (Python), đóng gói bằng Docker Compose

**Chức năng chính**

* Hiển thị danh sách các cấu trúc hạ tầng đã sinh ra trong thư mục terraform-projects/
* Khởi tạo (init), triển khai (apply) hoặc huỷ (destroy) hạ tầng một cách nhanh chóng nhiều thư mục cấu hình thông qua giao diện.

A screenshot of a computer

AI-generated content may be incorrect.

Hình Thực hiện Destroy trên 1 thư mục cấu hình Open stack

## Kết quả triển khai

* Hệ thống cho phép tự động sinh mã Terraform từ file JSON, giúp định nghĩa nhanh toàn bộ kiến trúc hạ tầng (VM, mạng, router, IP, bảo mật...).
* Hỗ trợ triển khai thành công trên cả AWS và OpenStack, với khả năng ánh xạ chính xác các tài nguyên đặc thù như EC2, subnet, flavor…
* Mỗi môi trường được sinh trong thư mục riêng biệt, cấu trúc rõ ràng và dễ dàng tái sử dụng.
* Giao diện web hiện đại (React + FastAPI), cho phép người dùng thao tác nhanh chóng với các chức năng khởi tạo, apply, destroy.
* Toàn bộ quy trình triển khai được tự động hóa, không cần can thiệp dòng lệnh.
* Thiết kế mở, dễ tích hợp thêm provider hoặc mở rộng tính năng trong tương lai.

# Hạn chế và hướng phát triển.

## Một số hạn chế của hệ thống

Mặc dù hệ thống đã đạt được nhiều kết quả tích cực, vẫn còn tồn tại một số điểm yếu cần tiếp tục cải thiện:

* **Chưa hỗ trợ nhiều nhà cung cấp cloud:**

Hiện tại, hệ thống chỉ hỗ trợ AWS và OpenStack, chưa tích hợp với các nền tảng phổ biến khác như Microsoft Azure, Google Cloud Platform (GCP).

* **Số lượng loại tài nguyên hỗ trợ còn hạn chế:**

Hiện tại, hệ thống chỉ cho phép tạo và quản lý một số loại tài nguyên cơ bản; chưa hỗ trợ đầy đủ các dịch vụ, tài nguyên nâng cao hoặc đặc thù của từng nhà cung cấp cloud.

* **Khả năng tùy biến giao diện còn hạn chế:**

Giao diện web mới chỉ cung cấp các thao tác cơ bản như init, apply, destroy. Chưa có tính năng tạo kiến trúc hạ tầng từ json, không có dashboard tổng quan, biểu đồ hiển thị trạng thái hoặc báo cáo tài nguyên.

* **Chưa có cơ chế phân quyền người dùng:**

Hệ thống chưa hỗ trợ chức năng đăng nhập, phân quyền thao tác theo vai trò (admin, user, viewer), hoặc ghi log thao tác (audit log).

* **Thiếu tích hợp với pipeline CI/CD:**

Việc sinh mã và triển khai Terraform hiện vẫn thực hiện thủ công, chưa kết nối với quy trình DevOps tự động như GitLab CI, GitHub Actions hay Jenkins.

• **Chưa hỗ trợ tích hợp với hệ thống quản lý bí mật (Secrets Management):**

Việc quản lý thông tin nhạy cảm như mật khẩu, API key vẫn thực hiện thủ công, chưa tích hợp với các giải pháp như HashiCorp Vault, AWS Secrets Manager…

## Hướng phát triển trong tương lai

* **Mở rộng hỗ trợ đa cloud:**

Tích hợp thêm các nhà cung cấp cloud phổ biến như Microsoft Azure, Google Cloud Platform (GCP)… để đáp ứng nhu cầu đa dạng của người dùng.

* **Bổ sung nhiều loại tài nguyên và dịch vụ:**

Phát triển thêm khả năng quản lý các tài nguyên nâng cao (database, load balancer, storage, serverless, v.v.) và các dịch vụ đặc thù của từng nhà cung cấp.

* **Nâng cấp giao diện người dùng:**

Xây dựng dashboard tổng quan, biểu đồ trạng thái, báo cáo tài nguyên, và cho phép tạo kiến trúc hạ tầng trực tiếp từ file JSON hoặc kéo thả trực quan.

* **Bổ sung phân quyền và xác thực:**

Hỗ trợ đăng nhập, phân quyền thao tác theo vai trò (admin, user, viewer), ghi log thao tác (audit log) để tăng tính bảo mật và kiểm soát.

* **Tích hợp với pipeline CI/CD:**

Kết nối hệ thống với các công cụ DevOps như GitLab CI, GitHub Actions, Jenkins… để tự động hóa quy trình kiểm thử, triển khai và rollback hạ tầng.

* **Tích hợp hệ thống quản lý bí mật:**

Kết nối với các giải pháp quản lý secrets như HashiCorp Vault, AWS Secrets Manager… để bảo mật thông tin nhạy cảm và tự động hóa việc cấp phát secrets cho hạ tầng.

**TÀI LIỆU THAM KHẢO**

[1] 200lab.io, “Infrastructure as Code (IaC) là gì?,” *200lab.io*, 2024. [Trực tuyến]. Có sẵn: <https://200lab.io/blog/iac-la-gi>

[2] HashiCorp, “What is Terraform | Terraform | HashiCorp Developer,” *HashiCorp Developer*, 2024. [Trực tuyến]. Có sẵn: <https://developer.hashicorp.com/terraform/intro>

[3] Amazon Web Services, Inc., “AWS là gì,” *Amazon Web Services*. [Trực tuyến]. Có sẵn: <https://aws.amazon.com/vi/what-is-aws/>

[4] OpenStack Foundation, “OpenStack Docs: 2025.1 User Guides,” *OpenStack.org*, 2025. [Trực tuyến]. Có sẵn: <https://docs.openstack.org/2025.1/user/>

[5] Canonical Ltd., “Introduction to cloud-init - cloud-init 25.1.4 documentation,” *Cloud-init Documentation*, 2017. [Trực tuyến]. Có sẵn: <https://cloudinit.readthedocs.io/en/latest/explanation/introduction.html>

[6] Canonical Ltd., “User-data formats - cloud-init 25.1.4 documentation,” *Readthedocs.io*, 2017. [Trực tuyến]. Có sẵn: <https://cloudinit.readthedocs.io/en/latest/explanation/format.html>

[7] Red Hat Inc., “Chapter 2. Introduction to cloud-init | Configuring and managing cloud-init for RHEL 9,” *Red Hat Documentation*, 2025. [Trực tuyến]. Có sẵn: <https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/9/html/configuring_and_managing_cloud-init_for_rhel_9/introduction-to-cloud-init_cloud-content#cloud-init-operates-in-stages_introduction-to-cloud-init>

[8] DEVOPSEDU.VN, “Bài 5. Triển khai Kubernetes cluster trên On-Premise,” *DEVOPSEDU.VN*, 07-Feb-2025. [Trực tuyến]. Có sẵn: https://devopsedu.vn/courses/khoa-hoc-kubenetes-thuc-te/lesson/bai-5-trien-khai-kubernetes-cluster-tren-on-premise-2/

‌