LÀM VIỆC VỚI TERRAFORM TRÊN AWS

Trong chương này chúng ta sẽ làm quen với một công cụ ở một tầm cao hơn: **Terraform**. Học cách sử dụng Terraform để tự động cung cấp và quản lý các tài nguyên trên Amazon Web Services (AWS), một trong những nền tảng điện toán đám mây lớn nhất thế giới.

Các bạn sẽ được hướng dẫn từng bước để tạo ra máy chủ EC2, thiết lập tường lửa (Security Group), quản lý khóa truy cập (SSH Key), tạo cơ sở dữ liệu, và quản lý quyền truy cập của người dùng (IAM), tất cả chỉ bằng việc viết code.

Mục Lục

PHẦN 1: CẤU HÌNH BAN ĐẦU

1.1. Hướng dẫn: Cấu hình Terraform để làm việc với AWS

- **Mục tiêu:** Kết nối Terraform với tài khoản AWS của bạn bằng cách cung cấp các thông tin xác thực (Access Keys).
- **Yêu cầu:**
    1. Đã cài đặt Terraform.
    2. Có một tài khoản AWS và đã tạo cặp Access Key (ID và Secret Key).
- **Các bước thực hiện:**
    1. Tạo một thư mục dự án mới.
    2. Trong thư mục đó, tạo 3 file sau:

a. `variables.tf` - File để khai báo các biến sẽ sử dụng

```
variable "aws_access_key" {
  description = "AWS Access Key ID."
}

variable "aws_secret_key" {
  description = "AWS Secret Access Key."
}

variable "aws_region" {
  description = "The AWS region to deploy to."
  default     = "eu-west-1" // Bạn có thể đổi sang region gần bạn, ví dụ "ap-southeast-1" cho Singapore
}
```

b. `terraform.tfvars` - File để gán giá trị cho các biến. **File này chứa thông tin nhạy cảm, tuyệt đối không đưa lên Git**

```
aws_access_key = "YOUR_ACCESS_KEY_ID"
aws_secret_key = "YOUR_SECRET_ACCESS_KEY"
```

c. `provider.tf` - File để khai báo nhà cung cấp dịch vụ (provider), trong trường hợp này là AWS

```
provider "aws" {
  access_key = var.aws_access_key
  secret_key = var.aws_secret_key
  region     = var.aws_region
}
```

3. Mở terminal trong thư mục dự án và chạy lệnh sau để Terraform tải provider về:

```
terraform init
```

- **Kết quả mong đợi:** Terraform sẽ tạo một thư mục `.terraform` chứa các file cần thiết của provider AWS. Môi trường của bạn đã sẵn sàng để tạo tài nguyên trên AWS.

1.2. Hướng dẫn: Tạo và sử dụng cặp khóa SSH trên AWS

- **Mục tiêu:** Tạo một cặp khóa SSH (public/private) trên máy của bạn và dùng Terraform để tải public key lên AWS, giúp bạn có thể truy cập vào các máy chủ sau này.
- **Yêu cầu:**
  1. Đã hoàn thành bài 1.1.
- **Các bước thực hiện:**
  1. Trong terminal, tạo một thư mục `keys` và tạo cặp khóa SSH:

```
mkdir keys
ssh-keygen -f keys/my_key -t rsa -N ""
# Lệnh này tạo ra 2 file: my_key (private) và my_key.pub (public)
```

2. Trong file `variables.tf`, khai báo một biến để lưu đường dẫn đến file public key:

```
variable "ssh_public_key_path" {
  description = "Path to the SSH public key."
  default     = "keys/my_key.pub"
}
```

3. Tạo file mới `ssh_key.tf` và định nghĩa tài nguyên `aws_key_pair`. Terraform sẽ đọc nội dung file public key và tải nó lên AWS.

```
resource "aws_key_pair" "my_key" {
  key_name   = "my-aws-key"
  public_key = file(var.ssh_public_key_path)
}
```

Giải thích: Hàm `file()` của Terraform có chức năng đọc nội dung của một file tại đường dẫn được chỉ định.

4. Áp dụng thay đổi:

```
terraform apply
```

- **Kết quả mong đợi:** Sau khi chạy `apply`, một key pair tên là `my-aws-key` sẽ xuất hiện trong giao diện EC2 Dashboard của AWS (phần Key Pairs).

1.3. Hướng dẫn: Quản lý tường lửa (Security Group) với Terraform

- **Mục tiêu:** Tạo một Security Group - hoạt động như một "tường lửa ảo" - để kiểm soát luồng truy cập vào/ra các máy chủ EC2. Trong bài này, chúng ta sẽ chỉ cho phép truy cập SSH (cổng 22) từ địa chỉ IP của chính bạn.
- **Yêu cầu:**
    1. Đã hoàn thành các bài trước.
- **Các bước thực hiện:**
    1. Tạo file mới `security_group.tf`.
    2. Định nghĩa tài nguyên `aws_security_group` với các quy tắc sau:

```
resource "aws_security_group" "ssh_allowed" {
  name        = "ssh-allowed-group"
  description = "Allow SSH inbound traffic"

  // Quy tắc truy cập vào (ingress)
  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    // Thay "0.0.0.0/0" bằng IP của bạn, ví dụ "1.2.3.4/32"
    // Để tìm IP của bạn, gõ "what is my ip" trên Google
    cidr_blocks = ["0.0.0.0/0"]
  }

  // Quy tắc truy cập ra (egress)
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1" // -1 nghĩa là cho phép mọi giao thức
    cidr_blocks = ["0.0.0.0/0"] // Cho phép đi ra bất cứ đâu
  }

  tags = {
    Name = "SSH Allowed"
  }
}
```

3. Áp dụng thay đổi:

```
terraform apply
```

- **Kết quả mong đợi:** Một Security Group mới tên là `ssh-allowed-group` sẽ được tạo trên AWS, sẵn sàng để gán vào các máy chủ EC2.

PHẦN 2: TẠO VÀ QUẢN LÝ MÁY CHỦ

2.1. Hướng dẫn: Tạo một máy chủ (EC2 Instance) Ubuntu trên AWS

- **Mục tiêu:** Tạo một máy chủ ảo Ubuntu hoàn chỉnh trên AWS, sử dụng SSH key và Security Group đã tạo ở các bài trước.
- **Yêu cầu:**
    1. Đã hoàn thành các bài trong Phần 1.
-
- **Các bước thực hiện:**
    1. Tạo file mới `ec2_instance.tf`.
    2. Định nghĩa tài nguyên `aws_instance` và tham chiếu đến các tài nguyên đã tạo:

```
resource "aws_instance" "web_server" {
  // AMI là ID của "bản mẫu hệ điều hành". AMI này là cho Ubuntu 20.04 ở region us-east-1.
  // Bạn cần tìm AMI phù hợp với region của mình.
  ami           = "ami-0c55b159cbfafe1f0"

  instance_type = "t2.micro" // Đây là loại máy miễn phí trong Free Tier của AWS

  // Tham chiếu đến key pair đã tạo
  key_name      = aws_key_pair.my_key.key_name

  // Tham chiếu đến security group đã tạo
  vpc_security_group_ids = [aws_security_group.ssh_allowed.id]

  tags = {
    Name = "My Web Server"
  }
}
```

Giải thích: Terraform cho phép bạn dùng `resource_type.resource_name.attribute` để lấy thông tin từ một tài nguyên đã được định nghĩa ở nơi khác. Ví dụ: `aws_key_pair.my_key.key_name`.

3. Áp dụng thay đổi:

```
terraform apply
```

- **Kết quả mong đợi:** Một máy chủ EC2 Ubuntu sẽ được tạo và khởi chạy trên tài khoản AWS của bạn.

2.2. Hướng dẫn: Tự động tạo nhiều máy chủ giống nhau

- **Mục tiêu:** Học cách sử dụng tham số `count` để tạo ra nhiều bản sao của một tài nguyên một cách nhanh chóng.
- **Yêu cầu:**
    1. Đã hoàn thành bài 2.1.
- **Các bước thực hiện:**
    1. Mở file `ec2_instance.tf`.
    2. Thêm tham số `count` vào tài nguyên `aws_instance`. Ví dụ, để tạo 2 máy chủ:

```
resource "aws_instance" "web_server" {
  count         = 2 // Tạo 2 máy chủ

  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = "t2.micro"
  key_name      = aws_key_pair.my_key.key_name
  vpc_security_group_ids = [aws_security_group.ssh_allowed.id]

  tags = {
    // Dùng count.index để đặt tên khác nhau cho mỗi máy
    Name = "Web Server ${count.index + 1}"
  }
}
```

3. Áp dụng thay đổi:

```
terraform apply
```

- **Kết quả mong đợi:** Terraform sẽ tạo ra 2 máy chủ EC2 giống hệt nhau, với tên là "Web Server 1" và "Web Server 2".

2.3. Hướng dẫn: Hiển thị thông tin hữu ích sau khi tạo tài nguyên

- **Mục tiêu:** Sử dụng khối `output` để Terraform in ra các thông tin quan trọng (như địa chỉ IP công cộng) sau khi chạy xong, giúp bạn dễ dàng sử dụng.
- **Yêu cầu:**
    1. Đã hoàn thành các bài trước.
- **Các bước thực hiện:**
    1. Tạo file mới `outputs.tf`.
    2. Định nghĩa một `output` để lấy địa chỉ IP của máy chủ đã tạo.

```
output "web_server_public_ip" {
  description = "The public IP address of the web server."
  // Nếu bạn dùng count, hãy dùng dấu * để lấy IP của tất cả các máy
  value       = aws_instance.web_server.*.public_ip
}

output "ssh_command" {
  description = "Command to SSH into the first server."
  // Lấy IP của máy đầu tiên (index 0)
  value       = "ssh -i keys/my_key ubuntu@${aws_instance.web_server[0].public_ip}"
}
```

3. Áp dụng thay đổi:

```
terraform apply
```

Kết quả mong đợi: Sau khi chạy xong, terminal sẽ hiển thị các giá trị output mà bạn đã định nghĩa, ví dụ:

```
Outputs:

ssh_command = "ssh -i keys/my_key ubuntu@54.12.34.56"
web_server_public_ip = [
  "54.12.34.56",
  "54.12.34.57",
]
```

PHẦN 3: CÁC DỊCH VỤ AWS NÂNG CAO

3.1. Hướng dẫn: Tạo và quản lý bộ nhớ S3

- **Mục tiêu:** Amazon S3 (Simple Storage Service) là dịch vụ lưu trữ đối tượng (object storage) cực kỳ phổ biến. Bạn sẽ học cách dùng Terraform để tạo một "xô" (bucket) S3, đưa một file lên đó, và truy cập file này qua trình duyệt.
- **Yêu cầu:**
    1. Đã hoàn thành các bài trong Phần 1.
- **Các bước thực hiện:**
    1. Tạo một file mới `s3.tf`.
    2. Trong file, định nghĩa tài nguyên `aws_s3_bucket` để tạo một bucket mới. Tên bucket phải là duy nhất trên toàn cầu.

```
resource "aws_s3_bucket" "my_website_bucket" {
  // Tên bucket phải là duy nhất trên toàn cầu, hãy thay đổi nó!
  bucket = "my-unique-handout-bucket-12345"

  tags = {
    Name = "My Website Bucket"
  }
}
```

3. Tiếp theo, định nghĩa tài nguyên `aws_s3_object` để tải một file `index.html` lên bucket vừa tạo.

```
resource "aws_s3_object" "index_file" {
  bucket       = aws_s3_bucket.my_website_bucket.id
  key          = "index.html" // Tên file trên S3
  source       = "index.html" // Đường dẫn đến file trên máy của bạn
  acl          = "public-read" // Cho phép mọi người đọc file này
  content_type = "text/html"
}
```

Lưu ý: Bạn cần tạo một file `index.html` đơn giản trong cùng thư mục dự án với nội dung ví dụ: `<h1>Hello from S3!</h1>`.

4. Để truy cập file này như một trang web, chúng ta cần bật tính năng "Static Website Hosting" cho bucket.

```
resource "aws_s3_bucket_website_configuration" "website_config" {
  bucket = aws_s3_bucket.my_website_bucket.id

  index_document {
    suffix = "index.html"
  }
}
```

5. Cuối cùng, tạo một `output` trong file `outputs.tf` để lấy đường dẫn trang web.

```
output "website_url" {
  description = "URL for the static website hosted on S3."
  value       = aws_s3_bucket_website_configuration.website_config.website_endpoint
}
```

6. Áp dụng thay đổi:

```
terraform apply
```

- **Kết quả mong đợi:** Terraform sẽ tạo một bucket S3, tải file `index.html` lên, và cấu hình nó như một trang web tĩnh. Terminal sẽ hiển thị một URL, bạn có thể dán vào trình duyệt để xem thành quả.

3.2. Hướng dẫn: Tạo kho chứa Docker (ECR) cá nhân

- **Mục tiêu:** Amazon ECR (Elastic Container Registry) là dịch vụ của AWS để lưu trữ các Docker image của bạn một cách riêng tư và an toàn. Bạn sẽ học cách tạo một kho chứa (repository) trên ECR.
- **Yêu cầu:**
    1. Đã hoàn thành các bài trong Phần 1.
    2. Đã cài đặt Docker và AWS CLI trên máy tính.
- **Các bước thực hiện:**
    1. Tạo file mới `ecr.tf`.
    2. Định nghĩa tài nguyên `aws_ecr_repository` để tạo một kho chứa mới.

```
resource "aws_ecr_repository" "my_app_repo" {
  name = "my-awesome-app" // Tên kho chứa của bạn

  image_tag_mutability = "MUTABLE" // Cho phép ghi đè tag (ví dụ: `latest`)
}
```

3. Thêm một `output` trong `outputs.tf` để lấy URL của kho chứa. URL này rất quan trọng để bạn có thể `docker push` sau này.

```
output "ecr_repository_url" {
  description = "URL of the ECR repository."
  value       = aws_ecr_repository.my_app_repo.repository_url
}
```

4. Áp dụng thay đổi:

```
terraform apply
```

5. **Bước quan trọng: Xác thực Docker với ECR.** Sau khi kho chứa được tạo, bạn cần chạy lệnh sau trên terminal để Docker có quyền đẩy image lên:

```
aws ecr get-login-password --region eu-west-1 | docker login --username AWS --password-stdin YOUR_ECR_REPOSITORY_URL
```

Thay `eu-west-1` bằng region của bạn và `YOUR_ECR_REPOSITORY_URL` bằng URL bạn nhận được từ output ở trên.

- **Kết quả mong đợi:** Một kho chứa Docker riêng tư đã sẵn sàng trên AWS. Bạn có thể build, tag, và đẩy Docker image của mình lên đó.

3.3. Hướng dẫn: Tạo cơ sở dữ liệu PostgreSQL (RDS)

- **Mục tiêu:** Amazon RDS (Relational Database Service) giúp bạn dễ dàng thiết lập, vận hành và mở rộng một cơ sở dữ liệu quan hệ trên đám mây. Bạn sẽ học cách tạo một CSDL PostgreSQL.
- **Yêu cầu:**
    1. Đã hoàn thành các bài trong Phần 1.
- **Các bước thực hiện:**
    1. Tạo file mới `rds.tf`.
    2. Đầu tiên, tạo một Security Group riêng cho RDS, chỉ cho phép truy cập vào cổng 5432 (cổng mặc định của PostgreSQL) từ IP của bạn.

```
resource "aws_security_group" "rds_access" {
  name        = "rds-access-group"
  description = "Allow access to PostgreSQL RDS"
  ingress {
    from_port   = 5432
    to_port     = 5432
    protocol    = "tcp"
    cidr_blocks = ["YOUR_IP/32"] // Thay bằng IP của bạn
  }
}
```

3. Định nghĩa tài nguyên `aws_db_instance` để tạo CSDL.

```
resource "aws_db_instance" "my_database" {
  allocated_storage    = 10            // Dung lượng lưu trữ (GB)
  engine               = "postgres"
  engine_version       = "13.7"
  instance_class       = "db.t3.micro"  // Loại máy (có trong Free Tier)
  db_name              = "mydatabase"
  username             = "myadmin"
  password             = "MySuperSecretPassword123" // **Nên dùng biến cho password!**
  skip_final_snapshot  = true
  vpc_security_group_ids = [aws_security_group.rds_access.id]
}
```

Lưu ý: Trong thực tế, không bao giờ viết password trực tiếp vào code. Hãy dùng biến và file `.tfvars` như đã học ở bài 1.1.

4. Tạo `output` trong `outputs.tf` để lấy địa chỉ kết nối (endpoint) của CSDL.

```
output "db_endpoint" {
  description = "Endpoint address for the RDS instance."
  value       = aws_db_instance.my_database.address
}
```

5. Áp dụng thay đổi:

```
terraform apply
```

- **Kết quả mong đợi:** Một CSDL PostgreSQL sẽ được tạo và khởi chạy trên AWS. Bạn sẽ nhận được địa chỉ endpoint để có thể kết nối bằng các công cụ như DBeaver, DataGrip, hoặc `psql`.

3.4. Hướng dẫn: Quản lý người dùng và phân quyền (IAM)

- **Mục tiêu:** Amazon IAM (Identity and Access Management) cho phép bạn quản lý quyền truy cập vào các tài nguyên AWS một cách an toàn. Bạn sẽ học cách tạo một người dùng mới với quyền hạn bị giới hạn (ví dụ: chỉ được đọc S3).
- **Yêu cầu:**
    1. Đã hoàn thành các bài trong Phần 1.
- **Các bước thực hiện:**
    1. Tạo file mới `iam.tf`.
    2. Định nghĩa tài nguyên `aws_iam_user` để tạo một người dùng mới tên là `student`.

```
resource "aws_iam_user" "student_user" {
  name = "student"
  path = "/students/"
}
```

3. Để gán quyền, chúng ta sẽ sử dụng một "chính sách" (policy) có sẵn của AWS là `AmazonS3ReadOnlyAccess`. Dùng `aws_iam_user_policy_attachment` để gắn policy này vào user.

```
resource "aws_iam_user_policy_attachment" "s3_read_only" {
  user       = aws_iam_user.student_user.name
  policy_arn = "arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess"
}
```

4. Tạo một cặp Access Key cho người dùng này để họ có thể truy cập thông qua các công cụ dòng lệnh (CLI).

```
resource "aws_iam_access_key" "student_key" {
  user = aws_iam_user.student_user.name
}
```

5. Tạo `output` trong `outputs.tf` để hiển thị Access Key vừa tạo.

```
output "student_access_key" {
  description = "Access key for the student user."
  value       = aws_iam_access_key.student_key.id
  sensitive   = true // Đánh dấu là thông tin nhạy cảm
}

output "student_secret_key" {
  description = "Secret key for the student user."
  value       = aws_iam_access_key.student_key.secret
  sensitive   = true
}
```

6. Áp dụng thay đổi:

```
terraform apply
```

- **Kết quả mong đợi:** Một người dùng IAM mới tên `student` được tạo. Người này chỉ có quyền đọc dữ liệu trên S3. Terraform sẽ hiển thị ra Access Key và Secret Key để bạn có thể cung cấp cho người dùng đó.