
Terraform + AWS

Infrastructure as Code

Tại sao cần dựng hệ thống bằng code (IAC)?

1. Code thì lưu phiên bản được
2. Code có thể cải tiến, chia tách module, tái sử dụng được
3. Code có thể xem lại được
4. Code có thể chạy tự động cho dù nó rất phức tạp và nhiều bước

Ngược lại:

1. Gõ lệnh thủ công hay thao tác giao diện tốn thời gian và có nhiều sai sót
2. Giao diện AWS console liên tục cải tiến, khiến hướng dẫn thường xuyên bị lạc hậu

5 loại Infrastructure As Code

1. **Ad hoc scripts**: Lệnh cấu hình tự viết thủ công
2. **Configuration management tools**: công cụ quản lý cấu hình
3. **Server templating tools**: công cụ tạo máy chủ mẫu
4. **Orchestration tools**: công cụ điều phối các tài nguyên, dịch vụ
5. **Provisioning tools**: công cụ dự toán tài nguyên

Ad hoc scripts – mã cấu hình tự viết

Lệnh tự viết thủ công không cần sử dụng thư viện, giải quyết từng trường hợp cụ thể.
Cần kinh nghiệm, tốn công sức, khó debug và dễ lỗi

```
# Update the apt-get cache  
sudo apt-get update
```

```
# Install PHP and Apache  
sudo apt-get install -y php apache2
```

```
# Copy the code from the repository  
sudo git clone https://github.com/brikis98/php-app.git /var/www/html/app
```

```
# Start Apache  
sudo service apache2 start"
```

Configuration management tools – công cụ cấu hình

Công cụ quản lý cấu hình: Chef, Puppet, Ansible, Salt Stack. Viết mã và tự động cấu hình trên nhiều máy.

```
- name: Update the apt-get cache
  apt:
    update_cache: yes
Code Ansible cài đặt Apache, PHP và clone git repo

- name: Install PHP
  apt:
    name: php

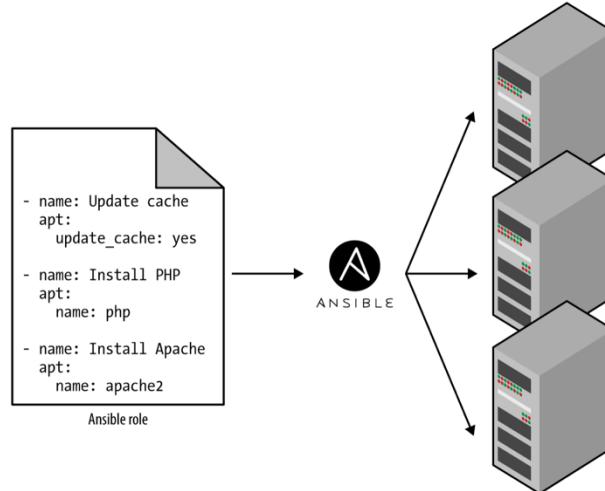
- name: Install Apache
  apt:
    name: apache2

- name: Copy the code from the repository
  git: repo=https://github.com/brikis98/php-app.git dest=/var/www/html/app

- name: Start Apache
  service: name=apache2 state=started enabled=yes
```

Ưu điểm Configuration Management Tools (CMT)

- Cung cấp sẵn rất nhiều hàm được kiểm thử kỹ lưỡng. Lập trình viên chỉ viết kịch bản (script), CMT sẽ tự chuyển đổi và thực thi
- Nhất quán trong mỗi lần chạy (**idempotence**)
- Viết một lần, chạy tự động trên nhiều máy



Server Templating Tools

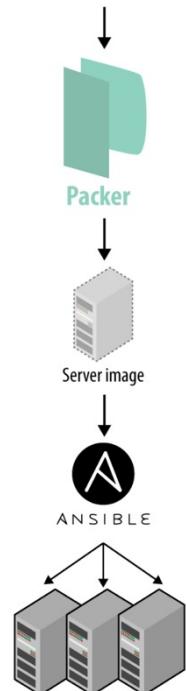
Công cụ dựng máy chủ từ mẫu: Docker, Packer, Vagrant.
Giúp lập trình viên không phải cài đặt, cấu hình máy chủ thủ công nữa.

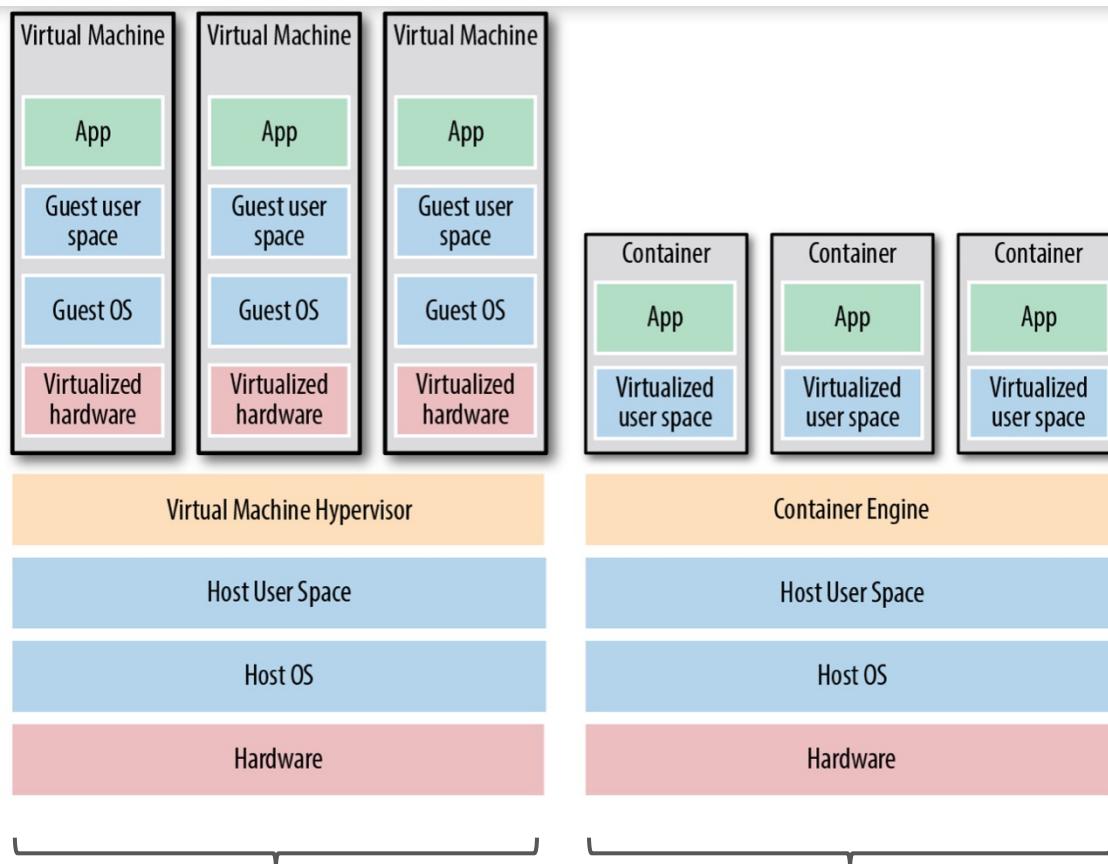
Docker tạo ra docker image từ các docker image gốc và tạo docker container từ docker image.

Vagrant tạo ra máy ảo trên các provider: virtual box, vmware, ...

```
"provisioners": [{  
    "type": "shell",  
    "inline": [  
        "apt-get update",  
        "apt-get install  
-y php",  
        "apt-get install  
-y apache2",  
    ]  
}]
```

Packer Template





Packer, Vagrant

Docker

```

Vagrant.configure("2") do |config|
  config.vm.provision "shell", inline: "echo Hello"

  config.vm.define "web" do |web|
    web.vm.box = "ubuntu/trusty64"
    web.vm.network "private_network", ip: "192.168.33.20"
    web.vm.synced_folder "code/", "/app/code"
    web.vm.provider "virtualbox" do |vb|
      vb.memory = 1048
      vb.cpus = 1
    end
  end

  config.vm.define "db" do |db|
    db.vm.box = "ubuntu/trusty64"
    db.vm.network "private_network", ip: "192.168.33.30"
    db.vm.synced_folder "data/", "/db/data"
    db.vm.provider "virtualbox" do |vb|
      vb.memory = 2048
      vb.cpus = 1
    end
  end
end

```



Kịch bản Vagrant dùng
2 máy ảo web và db

Mã Packer tạo EC2 instance sau đó cài đặt php, apache2

```
{  
  "builders": [{  
    "ami_name": "packer-example",  
    "instance_type": "t2.micro",  
    "region": "us-east-2",  
    "type": "amazon-ebs",  
    "source_ami": "ami-0c55b159cbfafe1f0",  
    "ssh_username": "ubuntu"  
  }],  
  "provisioners": [{  
    "type": "shell",  
    "inline": [  
      "sudo apt-get update",  
      "sudo apt-get install -y php apache2",  
      "sudo git clone https://github.com/brikis98/php-app.git /var/www/html/app"  
    ],  
    "environment_vars": [  
      "DEBIAN_FRONTEND=noninteractive"  
    ]  
  }]  
}
```

Terraform là provisioning và orchestration tool

- Provisioning: triển khai tài nguyên – deploy EC2,
- Orchestration: phối hợp giữa các tài nguyên trên các hệ thống / nền tảng khác nhau.



Loạt bài Terraform của thày Quân Huỳnh

[Bài 1 - Infrastructure as Code và Terraform](#)

[Bài 2 - Life cycle của một resource trong Terraform](#)

[Bài 3 - Terraform functional programming](#)

[Bài 4 - Terraform Module: Create Virtual Private Cloud on AWS](#)

[Bài 5 - Module In Depth: Create Multi-Tier Application](#)

[Bài 6 - Terraform Backend: Understand Backend](#)

[Bài 7 - Terraform Backend: S3 Standard Backend](#)

[Bài 8 - Terraform Backend: Remote Backend with Terraform Cloud](#)

[Bài 9 - CI/CD with Terraform Cloud and Zero-downtime deployments](#)

[Bài 10 - Terraform Blue/Green deployments](#)

[Bài 11 - Terraform A/B Testing Deployment](#)

[Bài 12 - Ansible with Terraform](#)

[Bài 13 - Automating Terraform with Gitlab CI](#)

[Bài 14 - Automating Terraform with Jenkins](#)

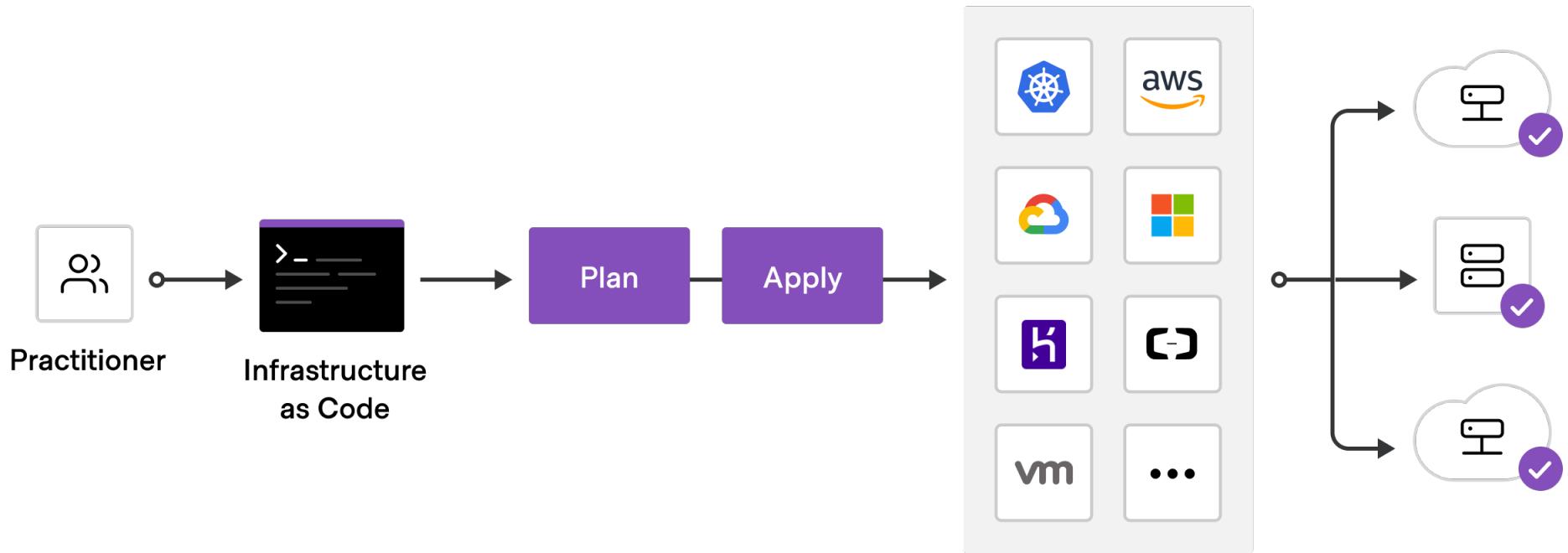
[Bài 15 - Multi-cloud environment](#)

[Bài 16 - Security - Securing Logs and Securing State File](#)

[Bài 17 - Security - Managing Secrets with Vault](#)

[Bài 18 - Handle the difference between Terraform State and Real Infrastructure](#)

Terraform + AWS

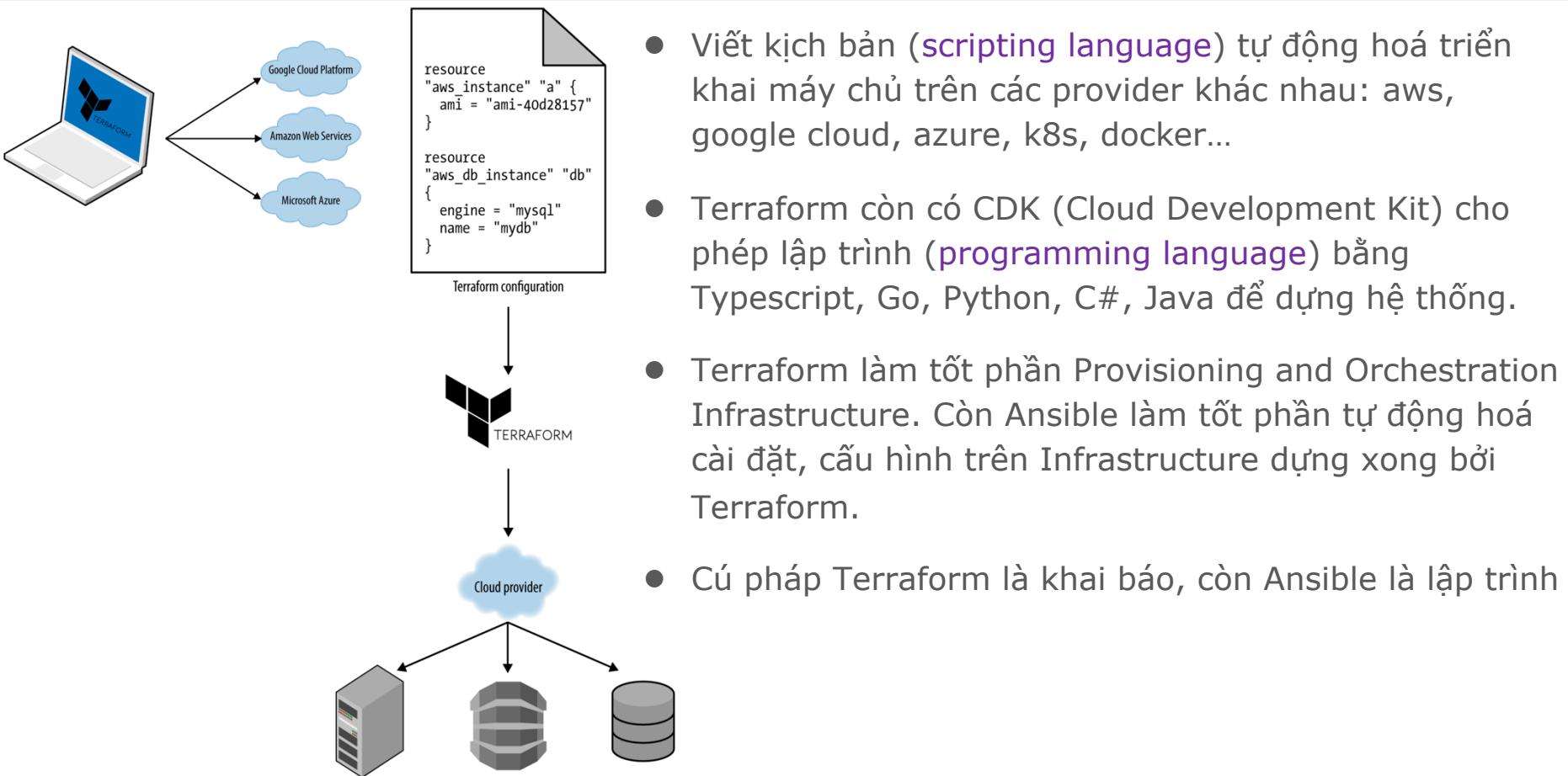


Ứng dụng của Terraform

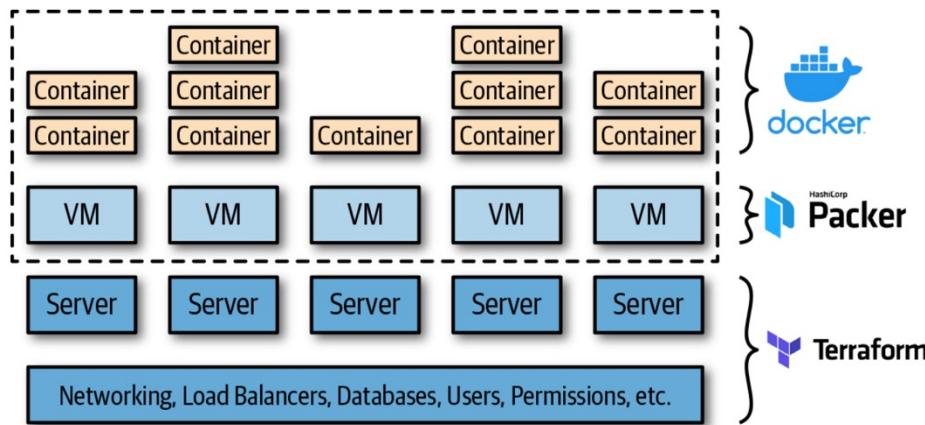
- Infrastructure As Code
- Multicloud Deployment
- Manage Kubernetes
- Manage Network Infrastructure
- Manage Virtual Machine Images
- Integrate with existing workflows
- Enforce policy as code
- Inject secrets into Terraform

- 
- Terraform mã nguồn mở, dùng miễn phí.
 - Hỗ trợ nhiều providers
 - Viết kịch bản (scripting) hoặc lập trình (programming)

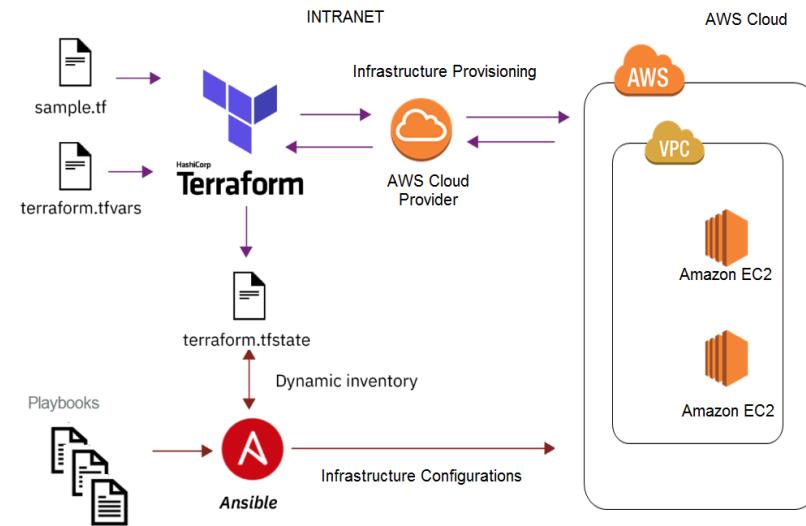
Terraform khác gì với các công cụ IAC?



Terraform phối hợp với các công cụ IAC khác



Kết hợp với Packer, Docker



Kết hợp với Ansible

Terraform

- Terraform dùng AWS SDK trong provider AWS
- Dùng ngôn ngữ HCL (gần giống JSON) để khai báo cách triển khai tài nguyên (provisioning)
- Terraform không dùng để lập trình sử dụng các tài nguyên mà chỉ dùng để triển khai và điều phối tài nguyên

AWS CLI, AWS SDK

- AWS CLI chạy lệnh trên terminal
- Kết hợp AWS CLI và Bash script để deploy (khó viết, dễ lỗi, khó debug)
- AWS SDK không những tạo được các resource, còn lập trình được
- AWS SDK lập trình logic phức tạp sử dụng nhiều ngôn ngữ khác nhau: Python, Java, Go, C++, C#, PHP, Ruby...

Terraform

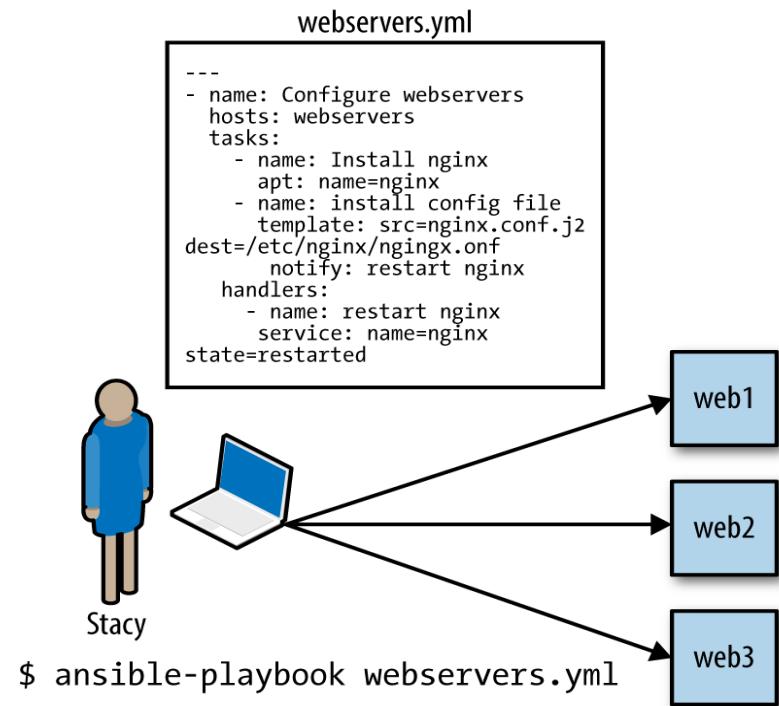
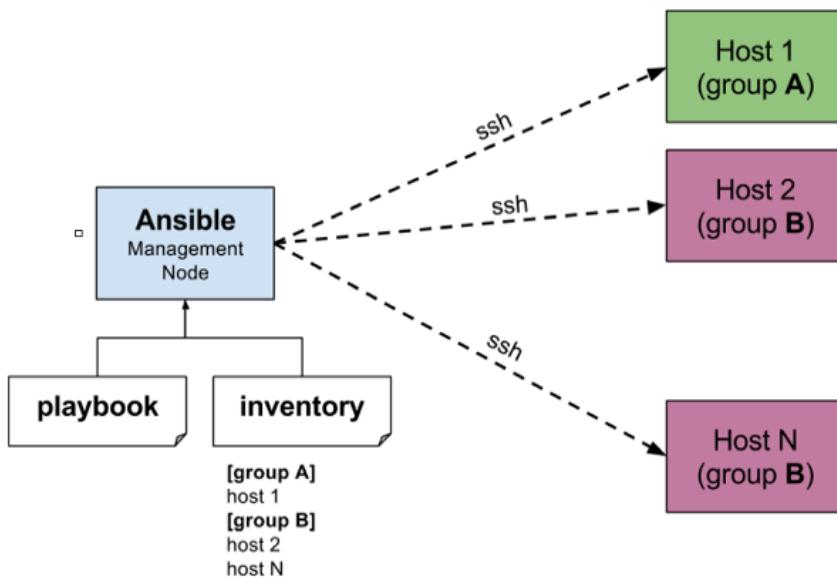
- Terraform là công cụ provisioning và orchestration
- Ngôn ngữ HCL + Golang
- Terraform có provider Ansible để thực thi lệnh trong play book

Ansible

- Ansible là công cụ cấu hình (configuration)
- Ngôn ngữ YAML + Python

Xem thêm bài này
<https://spacelift.io/blog/ansible-vs-terraform>

Ansible hoạt động như thế nào



Terraform

- Do Hashicorp open source
- HCL
- Hỗ trợ nhiều provider
- Provisioning + Orchestration
- Phổ biến hơn, dùng trong thực tế dự án.
- Không dùng để thi chứng chỉ AWS
- Có số lượng module dùng sẵn vượt trội

Cloud Formation

- Do Amazon phát triển, kín, độc quyền
- YAML
- Chỉ hỗ trợ AWS
- Provisioning
- Dễ học, nhưng áp dụng trong thực tế dự án ngày càng ít
- Cần học để thi chứng chỉ

Terraform hoạt động như thế nào? #1

- Terraform Client và Provider được viết bằng Golang
- Client + Provider sẽ đọc file *.tf để chuyển thành danh sách lệnh thực thi ở local PC và môi trường đích (AWS, Azure, Google Cloud)



Cài đặt Terraform

<https://learn.hashicorp.com/tutorials/terraform/install-cli>

- Mac

```
$ brew install terraform
```

- Windows

```
$ choco install terraform
```

- Linux

```
$ sudo apt-get install terraform
```

Tạo alias trong Linux/Mac để gõ lệnh Terraform nhanh

```
$ alias t='terraform'
```

```
$ t -version
```

```
Terraform v1.2.7  
on darwin_arm64
```

Từ bây giờ mọi lệnh **terraform**

tôi sẽ viết tắt thành **t** nhé



Lab 1: khởi tạo docker container

```
terraform {  
    required_providers {  
        docker = {  
            source  = "kreuzwerker/docker"  
            version = "~> 2.13.0"  
        }  
    }  
  
    provider "docker" {}  
  
    resource "docker_image" "nginx" {  
        name      = "nginx:latest"  
        keep_locally = false  
    }  
  
    resource "docker_container" "nginx" {  
        image = docker_image.nginx.latest  
        name  = "tutorial"  
        ports {  
            internal = 80  
            external = 8111  
        }  
    }  
}
```

\$ t init
\$ t plan
\$ t apply



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to [nginx.org](#). Commercial support is available at [nginx.com](#).

Thank you for using nginx.

Lab 2: Tạo EC2

```
terraform {  
    required_providers {  
        aws = {  
            source  = "hashicorp/aws"  
            version = "~> 4.25"  
        }  
    }  
    required_version = ">= 1.2.5"  
}  
  
provider "aws" {  
    region  = "ap-southeast-1"  
}  
  
resource "aws_instance" "app_server" {  
    ami           = "ami-0ff89c4ce7de192ea"  
    instance_type = "t2.micro"  
  
    tags = {  
        Name = "ExampleAppServerInstance"  
    }  
}
```



Cần cấu hình \$ aws configure
để cài đặt Access Key

resource định nghĩa tài nguyên cần khởi tạo

- Type: được định nghĩa sẵn tùy thuộc từng provider
- Name: do dev đặt để gọi, tham chiếu chỉ trong mã terraform. Khác với thuộc tính Name của tài nguyên
- Thuộc tính / logic (for loop, condition, function): dev tự khai báo

```
resource "aws_instance" "web" {
    ami           = "ami-0ff89c4ce7de192ea"
    instance_type = "t2.micro"
    security_groups = ["ingress_rules"]
    tags = {
        Name = "phpserver"
    }
}
```

Type

Name

Thuộc tính

Bạn nên chủ động xem tài liệu
của Terraform sẽ thấy rất nhiều ví dụ
khởi tạo resource



AWS DOCUMENTATION

Filter

EC2 (Elastic Compute Cloud)

- Resources
 - aws_ami
 - aws_ami_copy
 - aws_ami_from_instance
 - aws_ami_launch_permission
 - aws_ec2_availability_zone_group
 - aws_ec2_capacity_reservation
 - aws_ec2_fleet
 - aws_ec2_host
 - aws_ec2_serial_console_access
 - aws_ec2_tag
 - aws_eip
 - aws_eip_association
 - aws_instance**
 - aws_key_pair
 - aws_launch_template
 - aws_placement_group
 - aws_spot_datafeed_subscription
 - aws_spot_fleet_request
 - aws_spot_instance_request
- Data Sources
- EC2 Image Builder

Resource: aws_instance

Provides an EC2 instance resource. This allows instances to be created, updated, and deleted. Instances also support [provisioning](#).

Example Usage

Basic Example Using AMI Lookup

```
data "aws_ami" "ubuntu" {
  most_recent = true

  filter {
    name    = "name"
    values = ["ubuntu/images/hvm-ssd/ubuntu-focal-20.04-amd64-server-*"]
  }

  filter {
    name    = "virtualization-type"
    values = ["hvm"]
  }

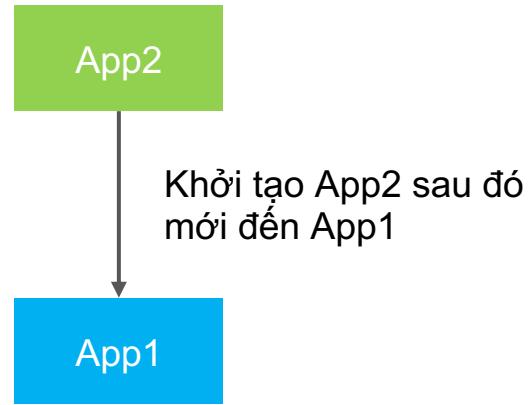
  owners = ["099720109477"] # Canonical
}

resource "aws_instance" "web" {
  ami          = data.aws_ami.ubuntu.id
  instance_type = "t3.micro"

  tags = {
    Name = "HelloWorld"
  }
}
```

Lab: 02_ec2_c_dependon

```
resource "aws_instance" "App1" {  
    ami           = "ami-0ff89c4ce7de192ea"  
    instance_type = "t2.micro"  
    depends_on = [  
        aws_instance.App2  
    ]  
    tags = {  
        Name = "App1"  
    }  
}  
  
resource "aws_instance" "App2" {  
    ami           = "ami-0ff89c4ce7de192ea"  
    instance_type = "t2.micro"  
    tags = {  
        Name = "App2"  
    }  
}
```

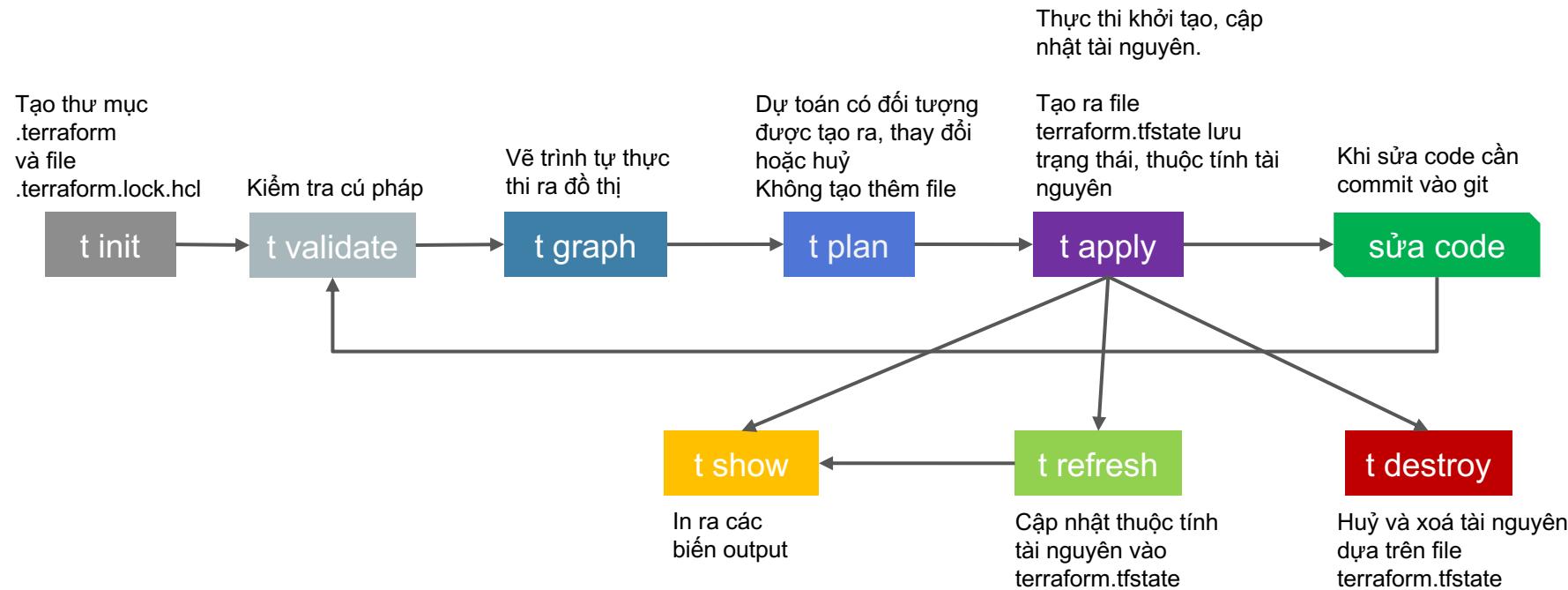


Chú ý khi dùng depends_on

- Tránh circular reference A depends_on B, B depends_on A
Error: Cycle: aws_instance.App1, aws_instance.App2
- Khi B tham chiếu đến A có nghĩa là B đã phụ thuộc A rồi, không cần khai báo thêm depends_on nữa

Tập lệnh terraform

Terraform hoạt động như thế nào? #2



Terraform có rollback như Cloud Formation không?

- Terraform không có rollback chỉ có destroy (xoá tài nguyên tạo trước đó hoặc tạo lỗi)
- Nên chia nhỏ project thành nhiều phần nhỏ, để khi cần có thể destroy và apply lại, không ảnh hưởng các tài nguyên khác
- Hoặc destroy một tài nguyên cụ thể → xem trang tiếp
- Một số tài nguyên, khi chỉnh sửa thuộc tính bắt buộc phải destroy rồi tạo lại chứ không đơn thuần gán thuộc tính được

terraform destroy -target restype.resname

- Quay lại bài lab 02_ec2_c_dependon, sau đó bỏ phụ thuộc depends_on
- Sau đó thử chạy lệnh

```
  t destroy -target aws_instance.App1 --autoapprove
```

- Đặt lại phụ thuộc depends_on để thấy khi xoá 1 tài nguyên X, mà tài nguyên Y, Z phụ thuộc, terraform sẽ xoá Y, Z trước rồi mới xoá X

terraform apply -target restype.resname

Chúng ta cũng có thể chọn để triển khai 1 tài nguyên cụ thể.

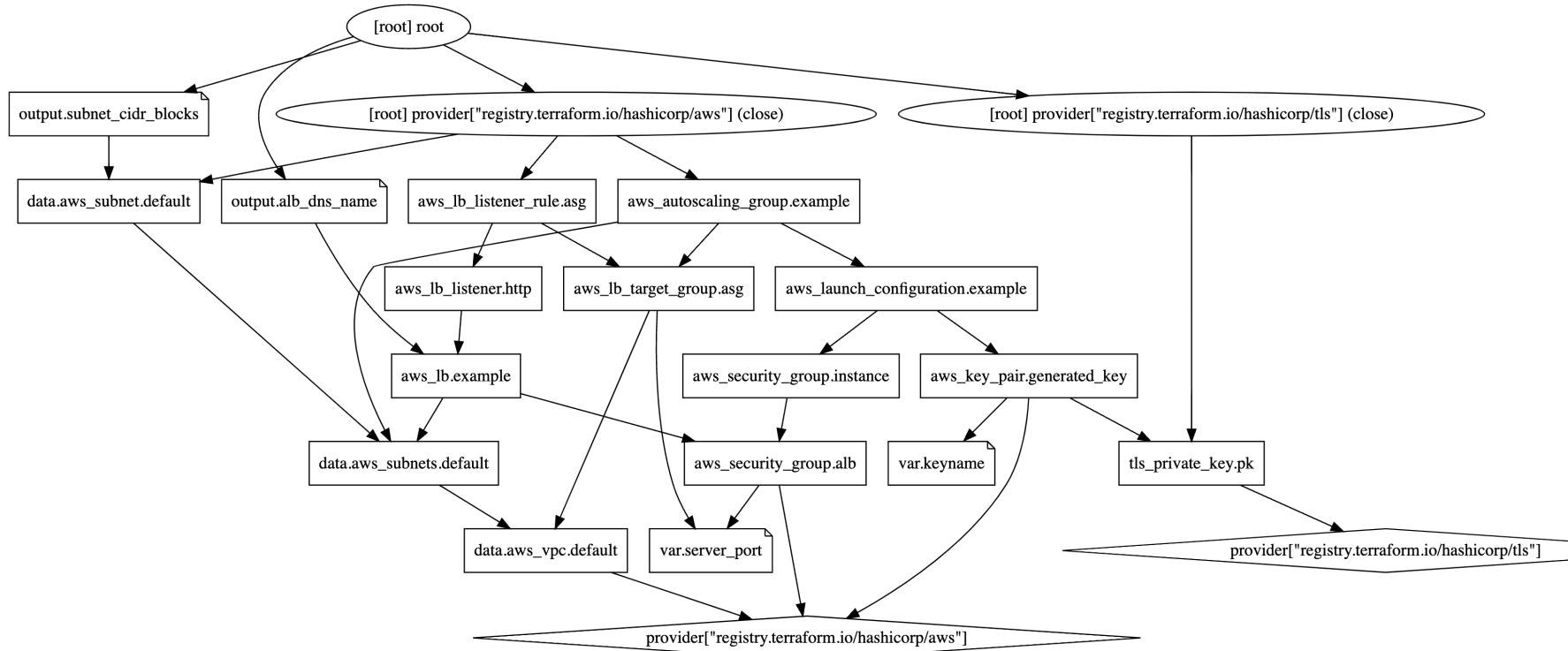
Tuy nhiên nếu tài nguyên này phụ thuộc/tham chiếu đến các tài nguyên khác thì terraform cũng triển khai các tài nguyên phụ thuộc ràng buộc lẫn nhau.

terraform state list

Liệt kê danh sách các resource được tạo ra sau lệnh **t apply**

```
~/aws/08_Terraform/07_b_module main !12 ?18 ➔ t state list
aws_instance.appA
aws_instance.appB
aws_security_group.ping_ssh
module.keypair["keyA"].aws_key_pair.generated_key
module.keypair["keyA"].tls_private_key.pk
module.keypair["keyB"].aws_key_pair.generated_key
module.keypair["keyB"].tls_private_key.pk
```

terraform graph



Hãy sử dụng terraform graph để quan sát sự phụ thuộc giữa các tài nguyên

A → B có nghĩa là tài nguyên A dùng, tham chiếu đến B

Hãy giải thích ý nghĩa thư mục terraform

- Thư mục .terraform
- File .terraform.lock.hcl
- File terraform.tfstate



Lab 3: Tạo security group gắn vào EC2

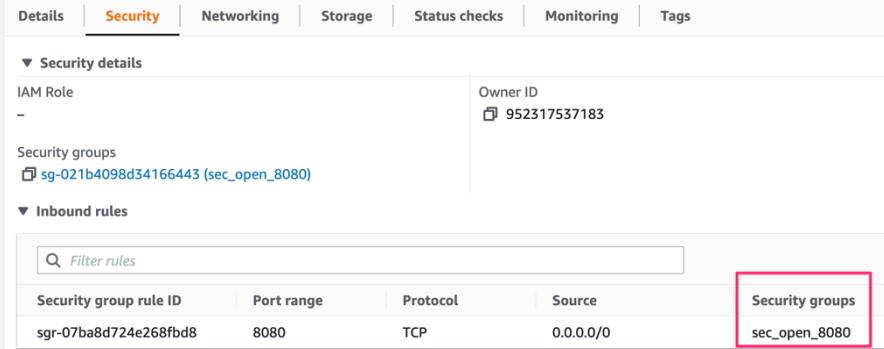
```
resource "aws_security_group" "instance" {
  name = "sec_open_8080"

  ingress {
    from_port   = 8080
    to_port     = 8080
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

resource "aws_instance" "app_server" {
  ami           = "ami-0ff89c4ce7de192ea"
  instance_type = "t2.micro"
  vpc_security_group_ids = [aws_security_group.instance.id]

  user_data = <<-EOF
    #!/bin/bash
    echo "Hello Terraform!" > index.html
    sudo python3 -m http.server 8080 &
  EOF

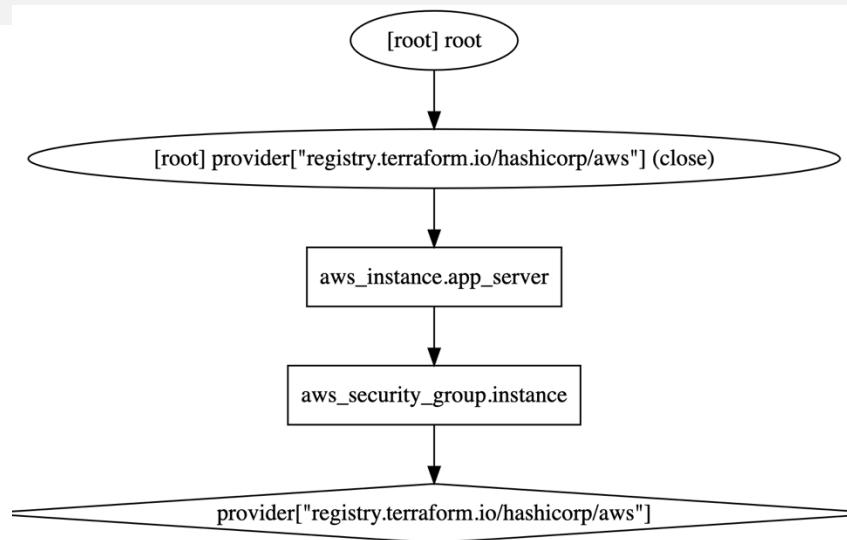
  tags = {
    Name = "PythonWebServer"
  }
}
```



terraform graph

```
digraph {
    compound = "true"
    newrank = "true"
    subgraph "root" {
        "[root] aws_instance.app_server (expand)" [label = "aws_instance.app_server", shape = "box"]
        "[root] aws_security_group.instance (expand)" [label = "aws_security_group.instance", shape = "box"]
        "[root] provider[\"registry.terraform.io/hashicorp/aws\"]" [label = "provider[\"registry.terraform.io/hashicorp/aws\"]", shape = "diamond"]
        "[root] aws_instance.app_server (expand)" -> "[root] aws_security_group.instance (expand)"
        "[root] aws_security_group.instance (expand)" -> "[root] provider[\"registry.terraform.io/hashicorp/aws\"]"
        "[root] provider[\"registry.terraform.io/hashicorp/aws\"] (close)" -> "[root] aws_instance.app_server (expand)"
        "[root] root" -> "[root] provider[\"registry.terraform.io/hashicorp/aws\"] (close)"
    }
}
```

Paste vào đây <https://dreampuf.github.io/GraphvizOnline>
để hình dung terraform đã chạy như thế nào



Lab 3: đọc từ file gán vào user_data

Nên dùng cách đọc file để giúp code terraform ngắn gọn, tách biệt khỏi những định dạng khác như JSON, bash, python...

```
resource "aws_instance" "app_server" {  
    ami           = "ami-0ff89c4ce7de192ea"  
    instance_type = "t2.micro"  
    vpc_security_group_ids = [aws_security_group.instance.id]  
  
    user_data = file("script.bash")  
    tags = {  
        Name = "PythonWebServer"  
    }  
}
```

main.tf

```
└── main.tf  
└── script.bash
```

Câu hỏi ôn tập, sinh viên tự trả lời

Giải thích các từ khoá:

- required_providers
- provider
- resource

Kiểu, Biển



Kiểu trong Terraform

- **string**: một dòng hoặc nhiều dòng
- **number**: 8080, 3.14
- **bool**: true hoặc false
- **list** (or tuple): ["us-west-1a", "us-west-1c"]
- **map** (or object): {name = "Mabel", age = 52}

3 loại biến trong Terraform

- Input: tham số đầu vào, nên viết tách ra khỏi logic chính để dễ bảo trì

```
variable "keyname" {
  description = "Name of keypair"
  type        = string
  default     = "demokey"
}
```

Cách để thay giá trị mặc định trong input variable

```
$ t apply -var 'keyname=product_key'
```

- Local: biến nội bộ

```
locals {
  instance_ip = aws_instance.web.public_ip
}
```

- Output: trả về thuộc tính tài nguyên sau khi chạy lệnh `t refresh` và `t output`

```
output "ssh_command" {
  value = "ssh -i '${var.keyname}.pem' ec2-user@${local.instance_ip}"
}
```

EOF, EOT, file đọc nhiều dòng

Khi cần nhập text nhiều dòng vào file *.tf, sẽ có 3 lựa chọn

1. Heredoc EOF

```
<<EOF  
line 1  
...  
line N  
EOF
```

2. Heredoc EOT tương tự như EOF

```
<<EOT  
line 1  
...  
line N  
EOT
```

3. Đọc từ file

```
inline_policy {  
    policy = file("write_convert_photo.json")  
}
```

Lab 4: sử dụng biến

```
variable "server_port" {
  description = "The port of web server"
  type        = number
  default     = 8080
}

variable "message" {
  description = "Message show on web site"
  type        = string
  default     = "Welcome to Terraform"
}

resource "aws_security_group" "instance" {
  name      = "sec_open"
  description = "Open inbound port to EC2"
  ingress {
    from_port    = var.server_port
    to_port      = var.server_port
    protocol     = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

```
resource "aws_instance" "app_server" {
  ami           = "ami-0ff89c4ce7de192ea"
  instance_type = "t2.micro"
  vpc_security_group_ids = [aws_security_group.instance.id]

  user_data = <<-EOF
    #!/bin/bash
    echo ${var.message} > index.html
    sudo python3 -m http.server 8080 &
  EOF

  tags = {
    Name = "PythonWebServer"
  }
}
```

Lab 5: output

```
resource "aws_instance" "app_server" {  
...  
}  
  
output "instance_ips" {  
    value = aws_instance.app_server.*.public_ip  
}
```

```
$ terraform output  
instance_ips = [  
    "13.228.30.115",  
]
```



Outputs:

```
instance_ips = [  
    "13.228.30.115",  
]
```

Lab 6: list

```
provider "aws" {  
    region = "ap-southeast-1"  
}  
  
resource "aws_iam_user" "users" {  
    count = length(var.user_names)  
    name  = var.user_names[count.index]  
}
```

main.tf

\$ **terraform apply**

\$ **terraform output**

\$ **terraform destroy**

```
variable "user_names" {  
    description = "Create IAM users"  
    type        = list(string)  
    default     = ["Paul", "John", "Hai"]
```

var.tf

Danh sách IAM user sẽ tạo

```
output "dirac_arn" {  
    value      = aws_iam_user.users[0].arn  
    description = "The ARN for user Paul"  
}
```

out.tf

```
output "all_arns" {  
    value      = aws_iam_user.users[*].arn  
    description = "The ARNs for all users"  
}
```

Lab 6: map for each

Khai báo biến kiểu map dạng {key, value}

```
variable "user_names" {
  type = map(object({
    path = string,
    tags = map(string)
  }))
  default = {
    "Paul" = {
      path = "/sales/"
      tags = {
        "email" = "paul@acme.com"
        "mobile" = "0902209011"
      }
    }
    "John" = {
      path = "/marketing/"
      tags = {
        "email" = "john@acme.com"
        "mobile" = "0902209012" }
    }
  }
}
```

Duyệt biến kiểu map bằng for_each

```
resource "aws_iam_user" "users" {
  for_each = var.user_names
  name     = each.key
  path     = each.value.path
  tags     = each.value.tags
}
```

Lab 07_a: tạo keypair #1

```
variable "keyname" {  
  description = "Name of keypair"  
  type        = string  
  default     = "demokey"  
}
```

var.tf

Khai báo biến keyname

```
resource "tls_private_key" "pk" {  
  algorithm = "RSA"  
  rsa_bits  = 4096  
}
```

main.tf

```
resource "aws_key_pair" "generated_key" {  
  key_name    = var.keyname  
  public_key  = "${tls_private_key.pk.public_key_openssh}"  
}
```

Định nghĩa resource aws_key_pair

```
provisioner "local-exec" { # Create keypair to your computer!!  
  command = <<EOT  
    echo '${tls_private_key.pk.private_key_pem}' > ./${var.keyname}.pem  
    chmod 400 ${var.keyname}.pem  
  EOT  
}
```

Ghi ra file

Giới hạn lại quyền

Lab 07_a: tạo keypair #1

```
resource "aws_instance" "web" {
    ami          = "ami-0ff89c4ce7de192ea"
    instance_type = "t2.micro"
    key_name      = "${aws_key_pair.generated_key.key_name}"      Gắn keypair vào EC2

    tags = {
        Name = "HelloWorld"
    }
}

locals {
    instance_ip = aws_instance.web.public_ip      Khởi tạo một biến cục bộ instance_ip
}

output "ssh_command" {
    value = "ssh -i '${var.keyname}.pem' ec2-user@${local.instance_ip}"      In ra câu lệnh để SSH vào EC2
}
```

↓

```
ssh_command = "ssh -i 'demokey.pem' ec2-user@54.254.244.129"
```

Trong bài lab này chúng ta đã học thêm kỹ thuật gì?

- Tạo key pair
- Chạy lệnh local

```
provisioner "local-exec" { # Create keypair to your computer!!
  command = <<EOT
    echo '${tls_private_key.pk.private_key_pem}' > ./${var.keyname}.pem
    chmod 400 ${var.keyname}.pem
  EOT
}
```

- Khai báo biến cục bộ `locals`

```
locals {
  instance_ip = aws_instance.web.public_ip
}
```

- Tạo một string từ nhiều biến ghép lại

```
value = "ssh -i '${var.keyname}.pem' ec2-user@${local.instance_ip}"
```

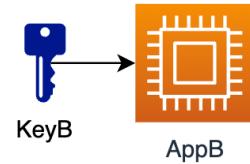
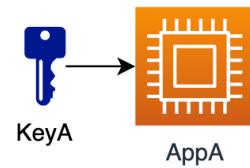
Module

Module để tái sử dụng khai báo tài nguyên

- Module giúp code tái sử dụng được và gọn hơn
- Module có thể nhận vào tham số
- Module có thể là local hoặc import từ trên mạng
- Module có thể tạo 1 hoặc nhiều resource tuy theo tham số

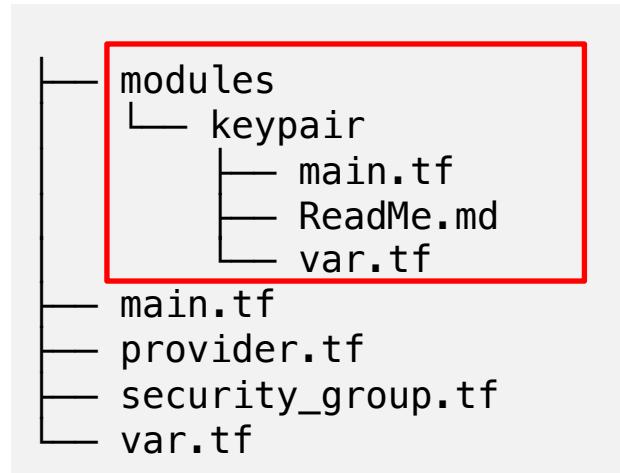
Lab 07_b: module

Yêu cầu: Cần tạo ra 2 EC2 instance, mỗi instance có một keypair để kết nối SSH riêng. Hãy tái sử dụng chức năng tạo keypair bằng khai báo module



Tạo modules trong dự án

Module có thể tạo local trong thư mục modules hoặc kéo về từ Internet



Sử dụng module

```
module "keypairA" {
  source = "./modules/keypair"
  keyname = var.keys.keyA
}

module "keypairB" {
  source = "./modules/keypair"
  keyname = var.keys.keyB
}
```

Tạo 2 keypair khác nhau từ một module
khai báo local

```
module "vpc" {
  source  = "terraform-aws-modules/vpc/aws"
  version = "2.77.0"

  name = "main-vpc"
  cidr = "10.0.0.0/16"

  azs           = data.aws_availability_zones.available.names
  public_subnets = ["10.0.4.0/24", "10.0.5.0/24", "10.0.6.0/24"]
  enable_dns_hostnames = true
  enable_dns_support    = true
}
```

Tạo 1 vpc từ module trên Internet

Một module thường có gì?

- var.tf định nghĩa các tham số truyền vào
- out.tf định nghĩa các biến sẽ in ra console
- main.tf hoặc resource.tf hoặc xyz.tf là nơi sẽ tạo các tài nguyên

Để học cách viết module hãy <https://github.com/terraform-aws-modules>
xem cách module chuyên nghiệp viết như thế nào



Terraform có tới 5659 modules cho AWS !!!

<https://registry.terraform.io/browse/modules?provider=aws>

The screenshot shows the Terraform Registry interface. At the top, there's a search bar with the URL "registry.terraform.io/browse/modules?provider=aws". Below the header, there are tabs for "Providers", "Modules" (which is selected), and "Policy Libraries". On the left, there's a sidebar with a "FILTERS" section containing a "Provider" dropdown set to "aws", which is highlighted with a red box. A "Clear Filters" button is also present. The main content area is titled "Modules" and contains two listed items:

- terraform-aws-modules / vpc**
Terraform module which creates VPC resources on AWS
- terraform-aws-modules / iam**
Terraform module which creates IAM resources on AWS

Each item has a timestamp ("25 days ago" or "5 days ago"), a file size ("26.3M" or "16.5M"), and an "aws provider" badge.

Hãy tìm module trước khi
bạn định viết một đoạn code rất
dài và phức tạp

Lab 07c: sử dụng module có sẵn

```
module "ssh_key_pair" {
  source = "cloudposse/key-pair/aws"
  version          = "0.18.3"
  stage            = var.stage
  name             = var.keyname
  ssh_public_key_path = "."
  generate_ssh_key   = "true"
  private_key_extension = ".pem"
  public_key_extension = ".pub"
}
```

Dùng module của CloudPose để tạo key pair

```
resource "aws_instance" "appA" {
  ami           = "ami-0ff89c4ce7de192ea"
  instance_type = "t2.micro"
  key_name      = "${var.stage}-${var.keyname}"
  security_groups = ["ping_ssh"]
  tags = {
    Name = "App A"
  }
}
```

Xuất key pair ra file

```
output "ssh_command" {
  value = "ssh -i '${var.stage}-${var.keyname}.pem' ec2-user@${aws_instance.appA.public_ip}"
```

Gắn key pair vào EC2 instance

Học được gì từ Terraform Module

- Module giúp tái sử dụng code
- Hãy tận dụng thư viện module có sẵn, nó giúp code nhanh hơn, bao được nhiều trường hợp ngoại lệ hơn.
- Với hơn 5600 modules cho AWS, bạn có thể dựng giải pháp AWS bằng Terraform nhanh, an toàn, tiện bằng IAC Terraform

Lab 8: Gắn EBS vào EC2 #1

```
resource "aws_instance" "web" {
    ami           = "ami-0ff89c4ce7de192ea"
    instance_type = "t2.micro"

    tags = {
        Name = "HelloWorld"
    }
}

resource "aws_ebs_volume" "example" {
    availability_zone = var.az
    size              = 1
    type              = "gp2"
    tags              = {name: "example volume", type: "gp2"}
}
```

main.tf



Lab 8: Gắn EBS vào EC2 #2

```
resource "aws_volume_attachment" "ebs_att" { main.tf
  device_name = "/dev/sdh"
  volume_id   = aws_ebs_volume.example.id
  instance_id = aws_instance.web.id
}
output "storages" {
  value = aws_instance.web.ebs_block_device
}
```



```
storages = toset([
{
  "delete_on_termination" = false
  "device_name" = "/dev/sdh"
  "encrypted" = true
  "iops" = 100
  "kms_key_id" = "arn:aws:kms:ap-southeast-1:952317537183:key/3e9b0905-5371-4340-91e5-df2ea4467f7c"
  "snapshot_id" = ""
  "tags" = tomap({
    "name" = "example volume"
    "type" = "gp2"
  })
  "throughput" = 0
  "volume_id" = "vol-0c9014f1cdb9d01bb"
  "volume_size" = 1
  "volume_type" = "gp2"
},
])
```

Lab 9: Tạo EC2, ssh để cài đặt phần mềm

Yêu cầu: khởi tạo EC2, kết nối SSH để cài đặt nginx, sau đó thay đổi file index.html

Bài lab này sử dụng **provisioner "remote-exec"** để kết nối SSH vào EC2

```
└── demokey.pem <- Keypair
└── main.tf <- file logic chính
└── provider.tf <- cấu hình provider, region
└── remote.tf <- kết nối SSH thực thi
└── security_group.tf <- cấu hình security group
└── var.tf <- các biến
```

◀ ▶ C ⌂ ▲ Not Secure | 13.212.116.75

Terraform is Cool

Kết nối SSH vào EC2 dùng key pair

```
resource "null_resource" "remote" {
  connection {
    type        = "ssh"
    user        = "ec2-user"
    private_key = file("./${var.keyname}.pem")
    host        = aws_instance.web.public_ip
  }

  provisioner "remote-exec" {
    inline = [
      "sudo amazon-linux-extras install nginx1 -y",
      "echo -e '<h1>Terraform is Cool</h1>' | sudo tee /usr/share/nginx/html/index.html",
      "sudo service nginx start"
    ]
  }
}
```

remote.tf

Security Group

- Có thể bổ xung nhiều ingress (inbound) và express (outbound)
- from_port ... to_port: dài port nào đến port nào
- protocol: "tcp". Nếu ""all" hoặc "-1" dùng để mở hết các port

```
resource "aws_security_group" "ingress_rules" {  
    name = "ingress_rules"  
    ingress { //SSH  
        from_port    = 22  
        to_port      = 22  
        protocol     = "tcp"  
        cidr_blocks = ["0.0.0.0/0"]  
    }  
  
    ingress { //HTTP  
        from_port    = 80  
        to_port      = 80  
        protocol     = "tcp"  
        cidr_blocks = ["0.0.0.0/0"]  
    }  
}
```

securitygroup.tf

Học được gì?

1. Chia nhỏ các file *.tf ra. Mỗi file chuyên trách một nhiệm vụ
2. Có thể dùng EC2 user data thay thế cho lệnh SSH
3. Đọc nội dung file vào một biến

```
private_key = file("./${var.keyname}.pem")
```

4. Có 2 loại thực thi lệnh:
 - `provisioner "local-exec"` dùng command, chạy trên laptop của dev
 - `provisioner "remote-exec"` dùng inline, chạy trên tài nguyên terraform khởi tạo
5. Nên kết hợp Ansible thì sẽ tốt hơn

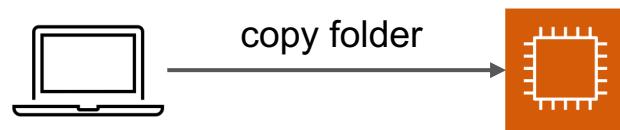
Lab 9b: triển khai web site nginx

Cải tiến từ bài lab 9, cần copy thư mục travel vào EC2, sau đó move vào /usr/share/nginx/html

Mã nguồn https://github.com/TechMaster/terraform_aws/tree/main/09_remote_ssh_cp_folder

Cần chia nhỏ các file *.tf ra theo chức năng: ec2.tf, keypair.tf, out.tf, provider.tf

```
resource "aws_instance" "web" {
  ...
  //Copy folder travel vào thư mục /home/ec2-user/
  provisioner "file" {
    source      = "./travel"
    destination = "/home/ec2-user/"
    connection {
      type      = "ssh"
      user      = "ec2-user"
      private_key = file("./${var.keyname}.pem")
      host      = aws_instance.web.public_ip
    }
  }
}
```



Lab 10: Cấu hình security group

- Cần tạo security group mở nhiều inbound ports 22, 80, 3306, 5432 và 6379 rồi gắn vào một EC2 instance
- Hướng xử lý tạo một biến kiểu list, chứa các đối tượng port, description

```
variable "ports" {
  type = list(object({
    port   = number
    description = string
  }))
  default = [
    {
      port      = 22
      description = "SSH"
    },
    {
      port      = 80
      description = "HTTP"
    },
    {
      port      = 3306
      description = "MySQL"
    },
    {
      port      = 5432
      description = "Postgresql"
    },
    {
      port      = 6379
      description = "Redis"
    }
  ]
}
```

var.tf

```
resource "aws_security_group" "app_secgroup" {  
    name      = "app_secgroup"  
}
```

```
resource "aws_security_group_rule" "ingress_rules" {  
    count      = length(var.ports)   count = số lượng phần tử trong list  
    type       = "ingress"  
    from_port  = var.ports[count.index].port  
    to_port    = var.ports[count.index].port  Lấy giá trị từng phần tử trong list gán vào  
    protocol   = "tcp"  
    cidr_blocks = ["0.0.0.0/0"]  
    description = var.ports[count.index].description  
    security_group_id = aws_security_group.app_secgroup.id Trỏ lên resource khai báo phía trên  
}
```

```
output "ingress_rules_simple" {
    value = [for i, v in aws_security_group_rule.ingress_rules :
              {desc = v.description, port = v.from_port}]
}
```



```
ingress_rules_simple = [
  {
    "desc" = "SSH"
    "port" = 22
  },
  {
    "desc" = "HTTP"
    "port" = 80
  },
  {
    "desc" = "MySQL"
    "port" = 3306
  },
  {
    "desc" = "Postgresql"
    "port" = 5432
  },
  {
    "desc" = "Redis"
    "port" = 6379
  },
]
```

Sử dụng vòng lặp **for** để duyệt một biến mảng

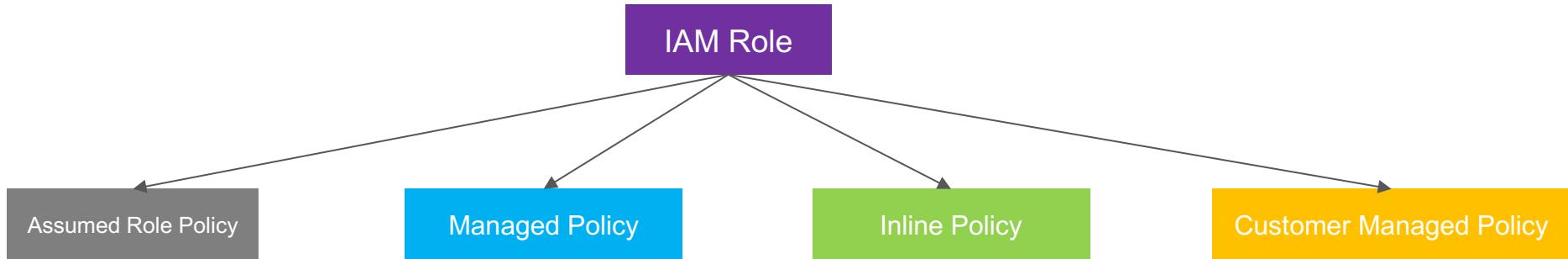
Xem thêm ví dụ ở đây

<https://www.terraform.io/language/expressions/for>

IAM Role



Những điểm cần nhớ về IAM Role



Role này sẽ được
dịch vụ nào đó dùng
EC2, Lambda...

Policy do AWS định nghĩa
sẵn, chỉ việc gắn vào

Policy định nghĩa trực tiếp
trong role, không tái sử dụng
ở nơi khác

Policy có thể gắn vào nhiều
role khác nhau

Assume Role Policy



```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "",  
      "Effect": "Allow",  
      "Principal": {  
        "Service": "ec2.amazonaws.com"  
      },  
      "Action": "sts:AssumeRole"  
    }  
  ]  
}
```



```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Principal": {  
        "Service": "lambda.amazonaws.com"  
      },  
      "Action": "sts:AssumeRole"  
    }  
  ]  
}
```

Managed Policy

Do AWS định nghĩa sẵn, chỉ việc gắn vào dùng

 AWSDirectConnectReadOnlyAccess	AWS managed	None
 AmazonGlacierReadOnlyAccess	AWS managed	None
 AWSMarketplaceFullAccess	AWS managed	None
 ClientVPNServiceRolePolicy	AWS managed	None
 AWSSSOAdministrator	AWS managed	None
 AWSIoT1ClickReadOnlyAccess	AWS managed	None
 AutoScalingConsoleReadOnlyAccess	AWS managed	None
 AmazonDMSRedshiftS3Role	AWS managed	None
 AWSQuickSightListIAM	AWS managed	None
 AWSHealthFullAccess	AWS managed	None
 AlexaForBusinessGatewayExecution	AWS managed	None

Lab 11: IAM Role Basic

Ví dụ tạo một role chứa 4 loại policy khác nhau

```
resource "aws_iam_role" "LambdaConvertPhotoRole2" {
    name          = "LambdaConvertPhotoRole2"
    assume_role_policy = file("assume_role.json")
    managed_policy_arns = ["arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole"]

    inline_policy {
        name      = "write_convert_photo"
        policy = file("write_convert_photo.json")
    }
    tags = {
        type = "Photo processing"
    }
}

resource "aws_iam_role_policy_attachment" "attach_LambdaConvertPhoto" {
    role      = aws_iam_role.LambdaConvertPhotoRole2.name
    policy_arn = aws_iam_policy.AccessS3Bucket.arn
}
```

Lab 12: gắn IAM role vào EC2

Hãy tạo một EC2 gắn với IAM Role chứa 2 policy AmazonS3FullAccess và AmazonRDSFullAccess

```
└── assume_role_ec2.json  
└── ec2.tf  
└── provider.tf  
└── role_rds_s3.tf
```

```
resource "aws_iam_role" "s3_rds_role" {  
    name = "S3_RDS_Role"  
    assume_role_policy = file("assume_role_ec2.json")  
    managed_policy_arns =  
        ["arn:aws:iam::aws:policy/AmazonS3FullAccess",  
         "arn:aws:iam::aws:policy/AmazonRDSFullAccess"]  
}
```

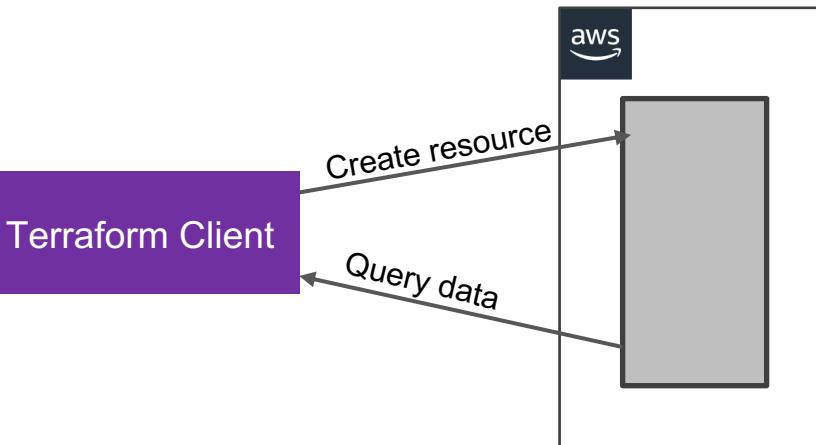
```
resource "aws_iam_instance_profile" "s3_rds_profile" {  
    name = "S3_RDS_Profile"  
    role = aws_iam_role.s3_rds_role.name  
}  
  
resource "aws_instance" "app_server" {  
    ami           = "ami-0ff89c4ce7de192ea"  
    instance_type = "t2.micro"  
    iam_instance_profile =  
        "${aws_iam_instance_profile.s3_rds_profile.name}"  
    tags = {  
        Name = "DemoIAMRole"  
    }  
}
```

Chú ý: `aws_iam_role`, `aws_iam_instance_profile`, `iam_instance_profile`

Data

Quan hệ Resource - Data

- **resource** là khối để tạo tài nguyên hạ tầng
- **data** là khối để truy vấn tài nguyên



A screenshot of the AWS Documentation website. The search bar contains "aws". The main navigation menu has "AWS DOCUMENTATION" at the top. Under "AWS SERVICES", "EC2 (Elastic Compute Cloud)" is selected. In the "Resources" section, "Data Sources" is expanded, showing a list of resources including "aws_instances" which is highlighted with a red border.

Data Source: aws_instances

Use this data source to get IDs or IPs of Amazon EC2 instances to be referenced elsewhere, e.g., to allow easier migration from another management solution or to make easier for an operator to connect through bastion host(s).

Note:

It's a best practice to expose instance details via `outputs` and `remote state` and use `terraform_remote_state` data source instead if you manage referenced instances via Terraform.

Note:

It's strongly discouraged to use this data source for querying ephemeral instances (e.g., managed via autoscaling group), as the output may change at any time and you'd need to re-run `apply` every time an instance comes up or dies.

Example Usage

```
data "aws_instances" "test" {
  instance_tags = {
    Role = "HardWorker"
  }

  filter {
    name = "instance_group_id"
  }
}
```

Lab 13: Định nghĩa data block để truy vấn thông tin

```
//danh sách AZ trong region
data "aws_availability_zones" "available" {
    state = "available"
}

//danh sách running EC2 instances
data "aws_instances" "running_instances" {
    instance_state_names = ["running"]
}

//Danh sách AMI
data "aws_ami" "amazon-linux" {
    most_recent = true
    owners      = ["amazon"]

    filter {
        name    = "name"
        values = ["amzn-ami-hvm-*-x86_64-ebs"]
    }
}
```

data.tf

```
output "availability_zones" {
  value = data.aws_availability_zones.available.names
}

output "running_instances" {
  value = data.aws_instances.running_instances.ids
}

output "amazon_ami" {
  value = {
    name = data.aws_ami.amazon-linux.name
    id = data.aws_ami.amazon-linux.id
    desc = data.aws_ami.amazon-linux.description
  }
}
```

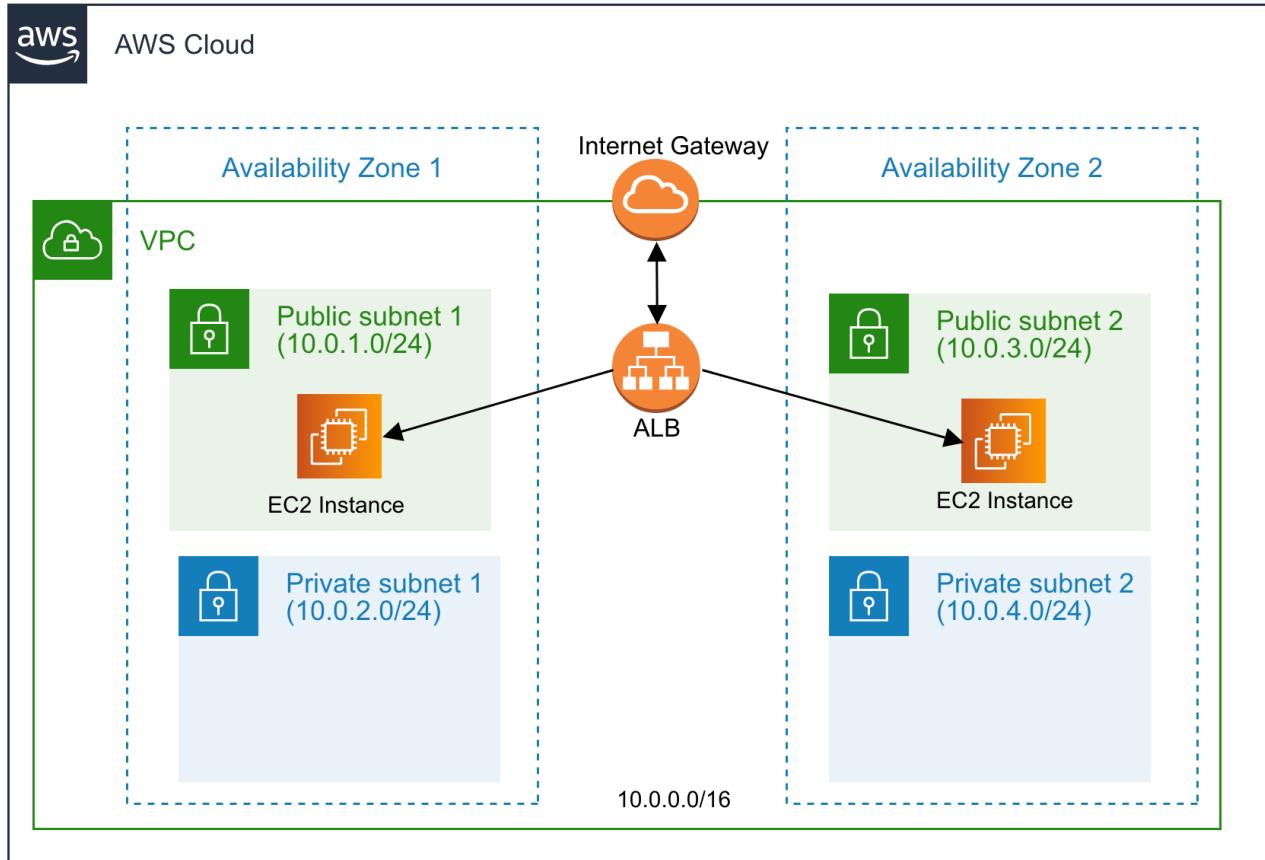
out.tf



```
amazon_ami = {
  "desc" = "Amazon Linux AMI 2018.03.0.20220705.1 x86_64 HVM ebs"
  "id" = "ami-083689f4a841ee07a"
  "name" = "amzn-ami-hvm-2018.03.0.20220705.1-x86_64-ebs"
}
availability_zones = tolist([
  "ap-southeast-1a",
  "ap-southeast-1b",
  "ap-southeast-1c",
])
running_instances = tolist([
  "i-093821fe22149d950",
])
```

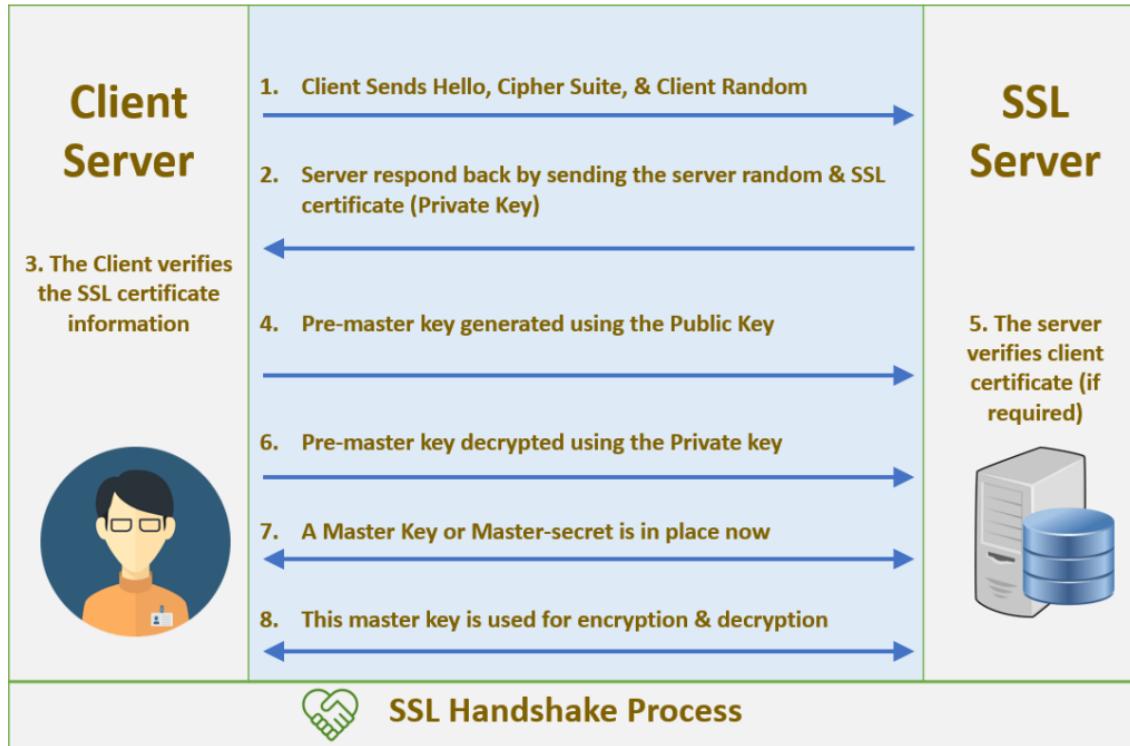
Autoscaling

Bài tập



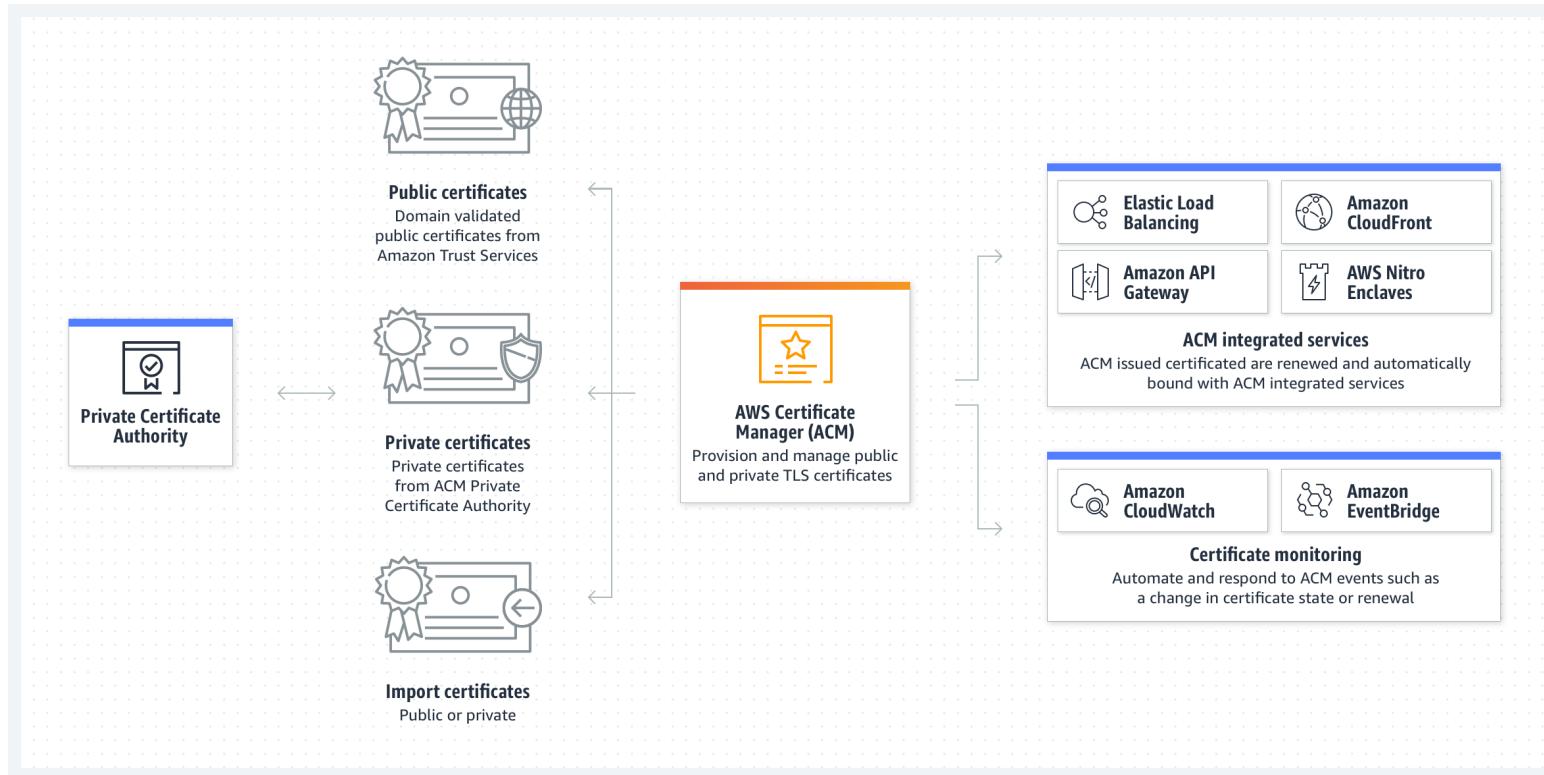
Sử dụng SSL

SSL certification



AWS Certificate Manager

Easily provision, manage, and deploy public and private SSL/TLS certificates for use with AWS services and your internal connected resources



- Từ 1-1000: 0.75\$/month
- Chỉ áp dụng cho ELB, Cloud Front, Beanstalk, API Gateway, Nitro Enclaves không áp dụng cho EC2



114



Q: Can I use certificates on Amazon EC2 instances or on my own servers?

No. At this time, certificates provided by ACM can only be used with specific AWS services.

Q: With which AWS services can I use certificates provided by ACM?

You can use ACM with the following AWS services:

- Elastic Load Balancing
- Amazon CloudFront
- AWS Elastic Beanstalk
- Amazon API Gateway

<https://aws.amazon.com/certificate-manager/faqs/>

You can't install the certificates created by [Amazon Certificate Manager \(ACM\)](#) on resources you have direct low-level access to, like EC2 or servers outside of AWS, because you aren't provided with access to the private keys. These certs can only be deployed on resources managed by the AWS infrastructure -- ELB and CloudFront -- because the AWS infrastructure holds the only copies of the private keys for the certificates that it generates, and maintains

Để web trong EC2 phục SSL connection

- Cách 1: Import SSL certificate được cấp phát
- Cách 2: dùng Certbot để gửi yêu cầu đến Letsencrypt cấp phát miễn phí SSL certificate. Sau một khoảng thời gian cần renew (yêu cầu cấp lại) certificate mới.
- <https://certbot.eff.org/>

Để 1 EC2 phục vụ SSL connection có thể dùn

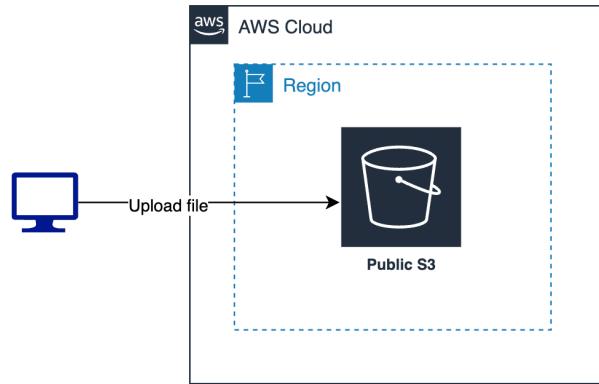
- You can only use ACM SSL certificates with AWS Load Balancers, CloudFront and API Gateway. it is not possible obtain the certificate from ACM and install it directly on a server.
- You can attach certificates issued with ACM to the AWS Load balancer and hide your instance behind the load balancer, more on this [here](#)
- If you want to manage ssl directly on your Nginx you will need to issue certificate with another tool i.e [letsencrypt](#).
- [Using Free Let's Encrypt SSL/TLS Certificates with NGINX](#)

<https://stackoverflow.com/questions/61502474/adding-aws-public-certificate-with-nginx>

S3



Lab 15: Hãy dựng một web site sử dụng S3



Upload file lên S3 bucket cần chọn đúng content type

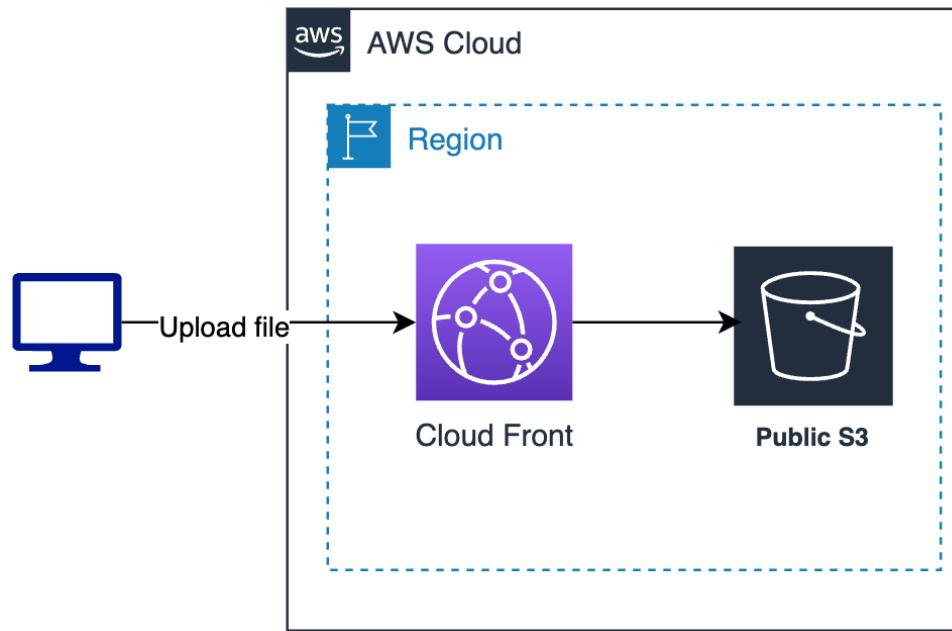
```
resource "aws_s3_object" "object" {
  for_each      = fileset(var.website_root, "**") //Duyệt tất cả các file trong thư mục
var.website_root
  bucket        = aws_s3_bucket.website.id //Gắn object vào s3 bucket
  key          = each.key
  source        = "${var.website_root}/${each.key}"
  source_hash   = filemd5("${var.website_root}/${each.key}")
  acl           = "public-read"
  //Gán đúng mime type
  content_type = lookup(local.mime_types, regex("\\.\\.[^.]+$", each.key), null)
}
```



Look up mime type theo file extension

```
{
  ".html": "text/html",
  ".css": "text/css",
  ".png": "image/png",
  ".jpg": "image/jpeg",
  ".js": "text/javascript",
  ".eot": "application/vnd.ms-fontobject",
  ".svg": "image/svg+xml",
  ".php": "application/x-httpd-php"
}
```

Lab 16: kết hợp S3 và Cloud Front



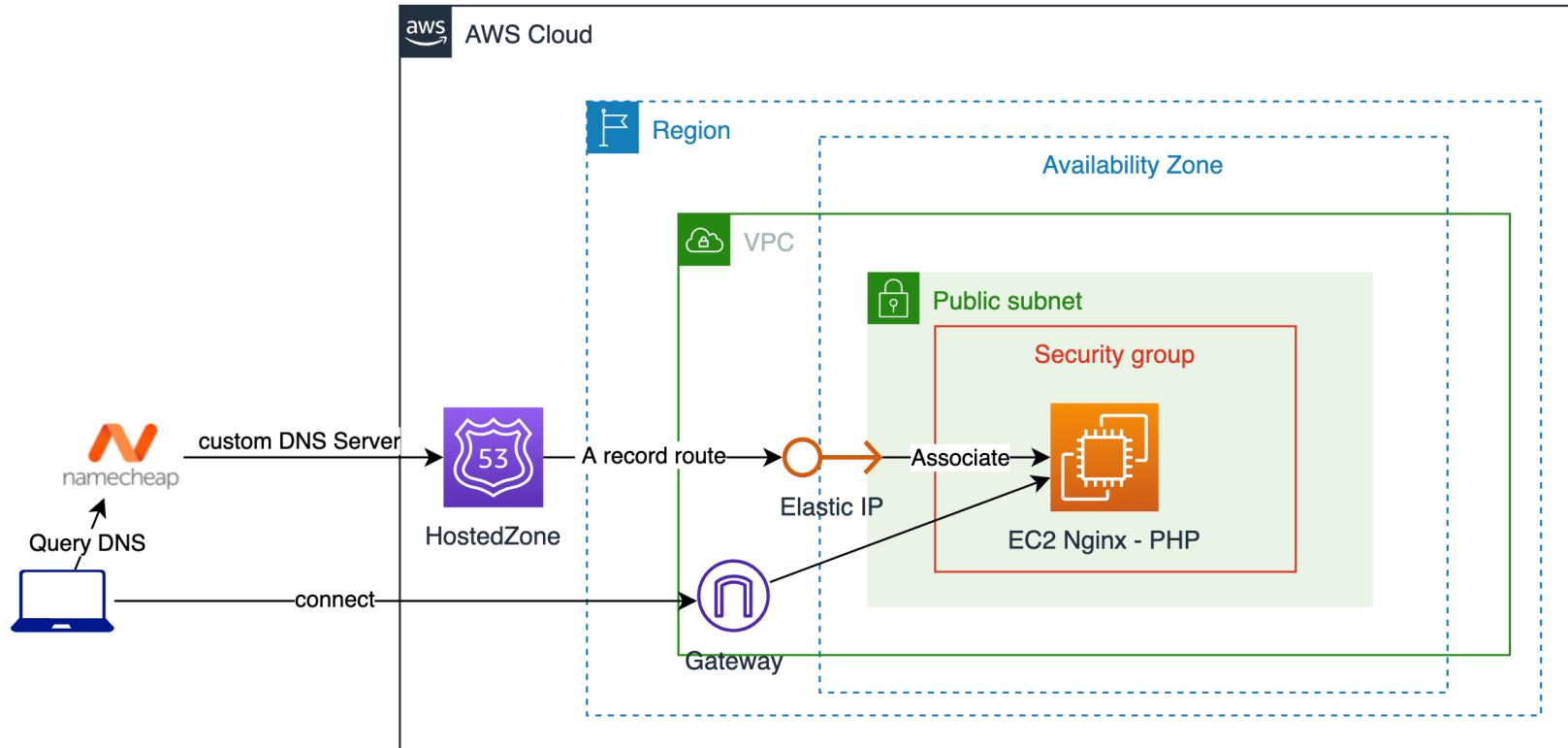
Cấu hình Domain, SSL, Elastic IP

Lab 18: nginx php ssl

Yêu cầu: hãy dựng một web site có tên miền, có SSL chạy trên Nginx và PHP

Cách thực hiện:

1. Tên miền mua ở aws 12\$/năm, tên miền mua ở NameCheap chỉ có 1.75\$/năm loại rẻ nhất
2. Cấu hình từ NameCheap trả về DNS do dịch vụ hosting zone AWS Route53 quản lý
3. Khởi tạo EC2 cài đặt Nginx – PHP
4. Cài đặt CertBot trong EC2 để tạo SSL certificate và chạy cron job tự động renew
5. Gắn Elastic IP vào EC2
6. Tạo A record trong hosting zone Route53 để trả vào Elastic IP



Mua tên miền ở NameCheap vì nó rẻ !

Namecheap Order Summary

Date: Aug 23, 2022 10:07:32 AM EST

Dear Cuong,

Thank you for choosing Namecheap. Here's a summary of your order.

Order Details

Order Date:	Aug 23, 2022 10:07:23 AM	Payment Source:	Credit Card ***(* * * ■■■■)
Order Number:	101958189	Initial Charge:	\$2.17
Transaction ID:	121343694	Final Cost:	\$2.17
User Name:	techmastervietnam	Total Refund:	N/A
Address:	No 14, lane 4, Nguyen Dinh Chieu Hai Ba Trung Dist Hanoi Hanoi,010000 VN	Refund Transaction ID:	N/A
		Refunded To:	N/A

TITLE	QTY	DURATION	PRICE	SUB TOTAL
Domain Registration hocphp.fun	1	1 year	\$1.99 ICANN Fee \$0.18	\$1.99
Free Domain Privacy	1	1 year	\$0.00 Setup \$0.00	\$0.00

Chỉ có 2.17 USD bạn đã có 1 tên miền

Sub Total \$0.00
TOTAL \$2.17

Trong R53 tạo hosting zone, copy DNS servers

Route 53 > Hosted zones > hocphp.fun

Public **hocphp.fun** [Info](#) [Delete zone](#) [Test record](#) [Configure query logging](#)

▶ Hosted zone details [Edit hosted zone](#)

[Records \(4\)](#) [DNSSEC signing](#) [Hosted zone tags \(0\)](#)

Records (4) [Info](#) [C](#) [Delete record](#) [Import zone file](#) [Create record](#)

Automatic mode is the current search behavior optimized for best filter results. [To change modes go to settings.](#)

[Filter records by property or value](#) [Type](#) [Routing policy](#) [Alias](#) [«](#) [1](#) [»](#) [⚙️](#)

<input type="checkbox"/>	Record name	Type	Routing	Differ...	Value/Route traffic to	TTL (seconds)	
<input type="checkbox"/>	hocphp.fun	A	Simple	-	18.143.81.70	300	
<input type="checkbox"/>	hocphp.fun	NS	Simple	-	ns-2.awsdns-00.com. ns-845.awsdns-41.net. ns-1909.awsdns-46.co.uk. ns-1107.awsdns-10.org.	172800	
<input type="checkbox"/>	hocphp.fun	SOA	Simple	-	ns-2.awsdns-00.com. awsdns-host	900	
<input type="checkbox"/>	www.hocphp.fun	A	Simple	-	18.143.81.70	300	

Cấu hình NameCheap chở DNS sang Route53

hocphp.fun

Domain Products Sharing & Transfer Advanced DNS

STATUS & VALIDITY ACTIVE Aug 23, 2022 - Aug 23, 2023 AUTO-RENEW

WithheldforPrivacy PROTECTION Aug 23, 2022 - Aug 23, 2023 AUTO-RENEW

PremiumDNS ? Enable PremiumDNS protection in order to switch your domain to our PremiumDNS platform. With our PremiumDNS platform, you get 100% DNS uptime and DDoS protection at the DNS level.

NAMESERVERS ? Custom DNS ns-1107.awsdns-10.org ns-1909.awsdns-46.co.uk ns-2.awsdns-00.com ns-845.awsdns-41.net ADD NAMESERVER

Copy DNS từ Route53 vào đây

Tạo E

Elastic IP addresses

Filter Elastic IP c

Name

hocphp.fun

Quick create record Info

[Switch to wizard](#)

Record 1

[Delete](#)

Record name Info

subdomain

hocphp.fun

Keep blank to create a record for the root domain.

Alias

Value Info

18.143.81.70

Tạo A record trong Hosting Zone trả vào Elastin IP

Record type Info

A – Routes traffic to an IPv4 address and some AWS resources

Enter multiple values on separate lines.

TTL (seconds) Info

300

1m

1h

1d

Recommended values: 60 to 172800 (two days)

Routing policy Info

Simple routing

Record 2

[Delete](#)

Record name Info

www

.hocphp.fun

Keep blank to create a record for the root domain.

Alias

Value Info

18.143.81.70

Record type Info

A – Routes traffic to an IPv4 address and some AWS resources

Kiểm tra tên miền đã trỏ đúng vào Elastic IP chưa

```
~ ➔ nslookup hocphp.fun
Server:  192.168.28.1
Address: 192.168.28.1#53

Non-authoritative answer:
Name:  hocphp.fun
Address: 18.143.81.70
```

Cấu hình cài đặt nginx, PHP-FPM và certbot

- Xem file script.sh
- Chú ý: Amazon Linux 2 hiện mới hỗ trợ PHP 8.0. Nếu cài PHP 8.1 thì cấu hình Nginx – PHP-FPM rất phức tạp

```
amazon-linux-extras install epel -y
```

Cài đặt Certbot để cập SSL certificate

```
yum -y install certbot python-certbot-nginx  
certbot --nginx --non-interactive --agree-tos -m cuong@techmaster.vn -d  
hocphp.fun -d www.hocphp.fun  
service nginx restart
```

```
# Auto renew Certbot
```

```
echo << EOF > /etc/cron.d/certbot
```

```
SHELL=/bin/sh
```

```
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
```

```
0 * * * 7 /usr/local/bin/certbot renew && systemctl restart nginx
```

```
EOF
```

```
systemctl restart crond
```

```
ingress { //ICMP Ping
    from_port    = -1
    to_port      = -1
    protocol     = "icmp"
    cidr_blocks = ["0.0.0.0/0"]
}
```

Cần để ping kiểm tra tên miền đã trả vào Elastic IP chưa

```
ingress { //SSH
    from_port    = 22
    to_port      = 22
    protocol     = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
}
```

Cần để SSH kết nối vào EC2

```
ingress { //HTTP
    from_port    = 80
    to_port      = 80
    protocol     = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
}
```

```
ingress { //HTTPS
    from_port    = 443
    to_port      = 443
    protocol     = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
}
```

Phục vụ SSL connection

Nginx server block

- Một Nginx server có thể host nhiều web site ứng với tên miền cụ thể
- Với mỗi web site, chúng ta sẽ định nghĩa khối server block trong thư mục /etc/nginx/conf.d

```
server {  
    listen      80;  
    listen      [::]:80;  
    server_name hocphp.fun www.hocphp.fun;  
    root        /var/www/phpsite;  
    index       index.html index.php;  
    charset     UTF-8;  
  
    location ~ \.php$ {  
        try_files $uri =404;  
  
        include fastcgi_params;  
        fastcgi_param SCRIPT_FILENAME    $document_root$fastcgi_script_name;  
        fastcgi_split_path_info ^(.+\.php)(/.+)$;  
        fastcgi_index index.php;  
        fastcgi_pass  unix:/var/run/php-fpm/www.sock;  
        fastcgi_cache_valid any 30m;  
    }  
}
```

Học được gì từ bài lab này

- Đăng ký tên miền, trỏ tên miền về Route 53 hosting zone
- Gắn Elastic IP vào EC2
- Viết Bashscript cài đặt Nginx, PHP-FPM, Certbox. Nếu được hãy chuyển qua Ansible

VPC – Subnet – Internet Gateway – Route – Route Table



AWS Cloud



Region



Gateway

Route

Route table
172.16.0.0
172.16.1.0
172.16.2.0

VPC

Availability Zone A

Availability Zone B



Public subnet

Security group



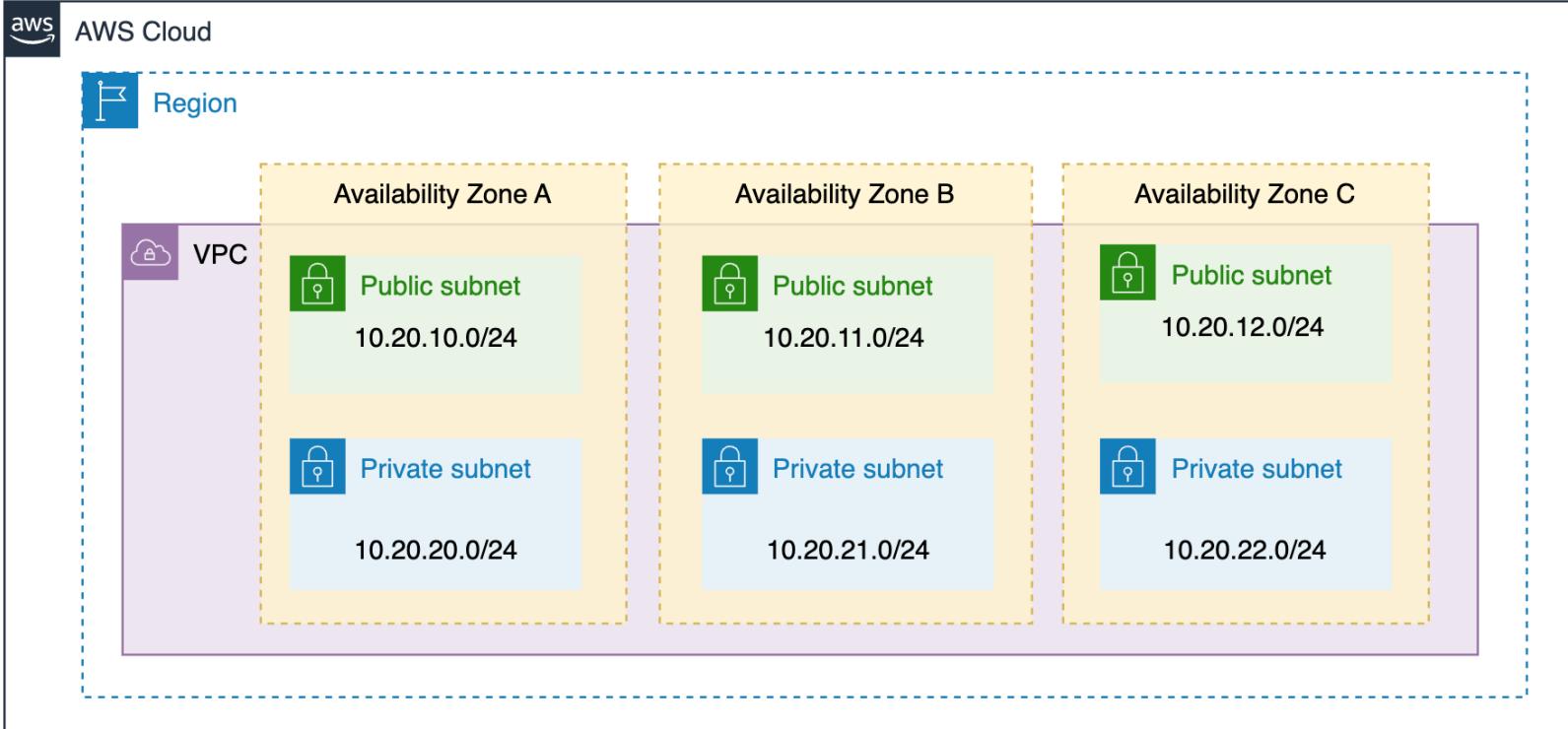
Public subnet



Private subnet



Private subnet



Tạo VPC với cidr_block
instance_tenancy "default" sẽ rẻ hơn "dedicated"

```
resource "aws_vpc" "test-vpc" {  
    cidr_block          = "10.20.0.0/16"  
    enable_dns_hostnames = true  
    instance_tenancy     = "default"  
    tags = {  
        Name = "TestVPC"  
    }  
}
```

vpc.tf

Có bao nhiêu Availability Zone trong region thì tạo bấy nhiêu Public và Private Subnet

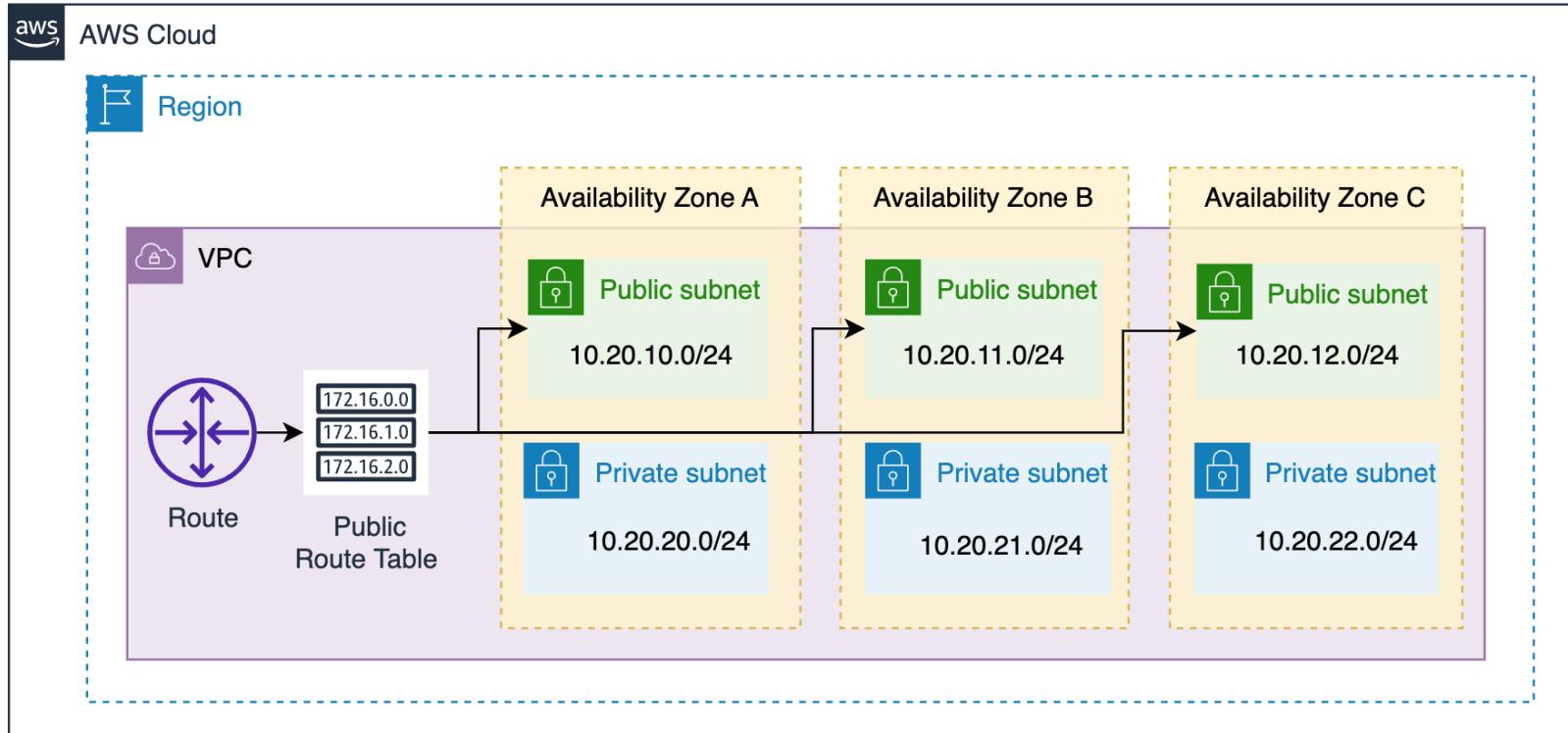
```
resource "aws_subnet" "public-subnet" {
    count                  = length(data.aws_availability_zones.available.names)
    vpc_id                 = aws_vpc.test-vpc.id
    cidr_block              = "10.20.${10 + count.index}.0/24"
    availability_zone       = data.aws_availability_zones.available.names[count.index]
    map_public_ip_on_launch = true
    tags = {
        Name = "public-subnet${count.index}"
    }
}
```

public_subnet.tf

```
resource "aws_subnet" "private-subnet" {
    count                  = length(data.aws_availability_zones.available.names)
    vpc_id                 = aws_vpc.test-vpc.id
    cidr_block              = "10.20.${20 + count.index}.0/24"
    availability_zone       = data.aws_availability_zones.available.names[count.index]
    map_public_ip_on_launch = false # Privat subnet
    tags = {
        Name = "private-subnet${count.index}"
    }
}
```

private_subnet.tf

Public route table



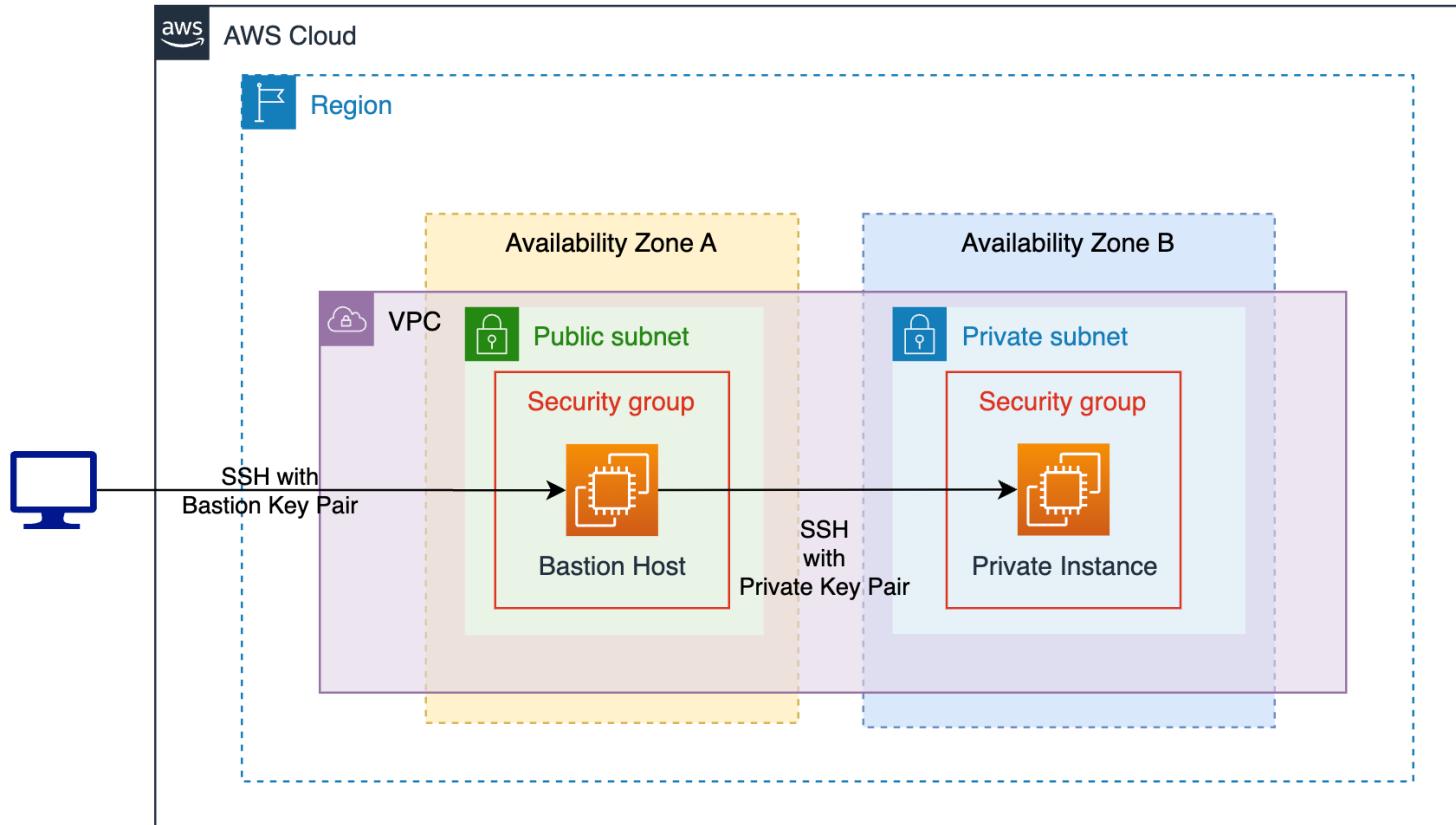
Bastion Host



Lab: Tạo Bastion Host

- Tận dụng code VPC, Subnet tạo ra ở bài lab trước, hãy tạo Bastion Host
- Một EC2 instance nằm trong private subnet sẽ không thể truy cập từ Internet. Bastion Host là một EC2 instance nằm trong public subnet. Nó dùng là nơi để truy xuất SSH vào các EC2 instance nằm trong private subnet.
- Hãy tạo 2 keypair: một cho bastion host và một cho private instance. Trong khi dựng Terraform, hãy copy keypair của private instance vào bastion host. Như vậy từ bastion host, user dễ dàng kết nối SSH vào private instance

Tham khảo bài lab <https://www.udemy.com/course/aws-certified-solutions-architect-associate-saa-c03/learn/lecture/28874510#overview>



```
modules
└── keypair
    ├── keypair.tf
    └── var.tf
bastion_host.tf
bastionkey.pem
data.tf
egress_gateway.tf
internet_gateway.tf
private_ec2.tf
private_route_table.tf
private_subnet.tf
privatekey.pem
provider.tf
public_route_table.tf
public_subnet.tf
security_group.tf
var.tf
vpc.tf
```

Định nghĩa module tạo KeyPair

Tạo bastion host trong public subnet

Key pair sinh ra để SSH vào bastion host

Tạo private instance trong private subnet

Key pair sinh ra copy vào bastion host rồi từ đó SSH vào private instance

Security group của bastion host và private instance

Nat Gateway

Yêu cầu

- Ở bài trước, EC2 instance trong private subnet không thể kết nối ra Internet, trong bài này, hãy bổ xung thêm NAT Gateway làm nhiệm vụ trung chuyển gói tin từ EC2 ra Internet và ngược lại.
- Code bài trước khá dài, hãy sử dụng VPC module để code gọn hơn
- Chú ý: NAT Gateway thu phí do đó thực hành xong cần destroy ngay

vpc.tf

```
module "vpc" {
    source = "terraform-aws-modules/vpc/aws"

    name = "my-vpc"
    cidr = "10.0.0.0/16"

    azs          = data.aws_availability_zones.az_available.names
    private_subnets = ["10.0.10.0/24", "10.0.11.0/24", "10.0.12.0/24"]
    public_subnets  = ["10.0.20.0/24", "10.0.21.0/24", "10.0.22.0/24"]

    enable_nat_gateway = true //Tạo NAT Gateway
    enable_vpn_gateway = true

    tags = {
        Terraform = "true"
        Environment = "dev"
    }
}
```

```
//Security group cho private EC2 instance nằm trong private subnet
resource "aws_security_group" "private_ssh_ping" {
    name      = "private_ssh_ping"
    vpc_id    = module.vpc.vpc_id

    ingress { //ICMP Ping
        from_port    = -1
        to_port      = -1
        protocol     = "icmp"
        security_groups = [aws_security_group.ssh_ping.id]
    }

    ingress { //SSH
        from_port    = 22
        to_port      = 22
        protocol     = "tcp"
        security_groups = [aws_security_group.ssh_ping.id]
    }

    //Cần bổ xung thêm để nối vào NAT Gateway
    egress { //Allow all outbound ports
        from_port    = 0
        to_port      = 0
        protocol     = "-1"
        cidr_blocks = ["0.0.0.0/0"]
    }
}
```



Mở cổng ra ngoài

Tổng kết

