



Terraform + AWS

Infrastructure as Code

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
5. **Provisioning tools**: công cụ dự toán tài nguyên

Ad hoc scripts

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 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

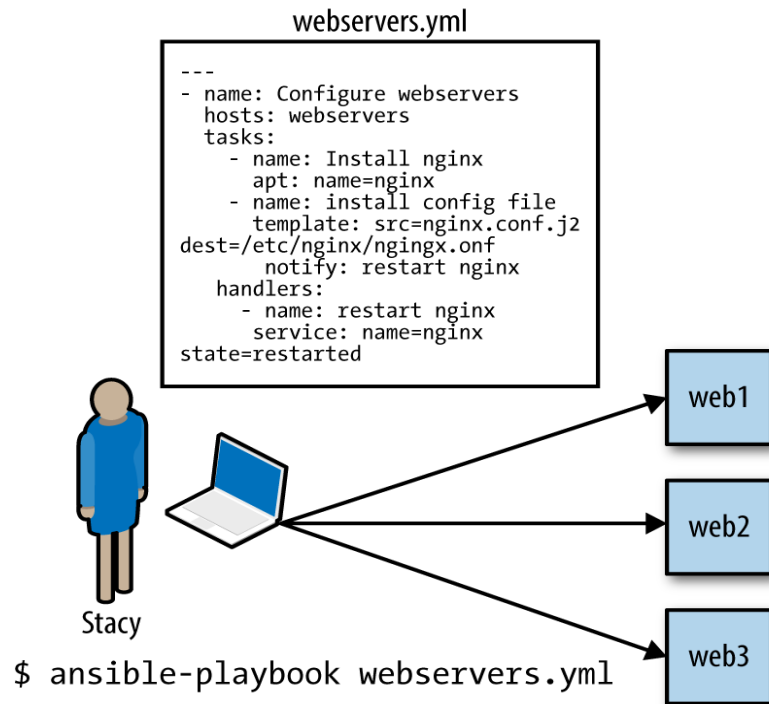
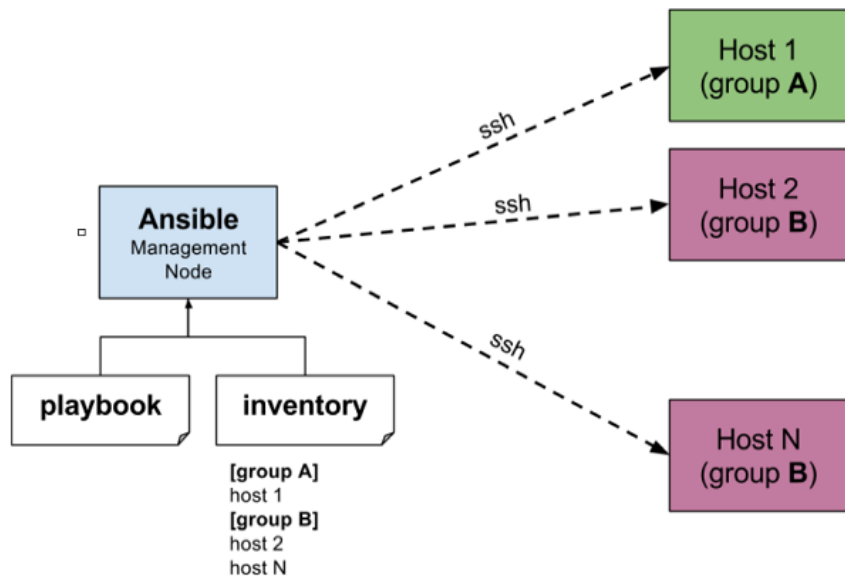
- `name`: Copy the code from the repository

 - `git`: repo=https://github.com/brikis98/php-app.git dest=/var/www/html/app

- `name`: Start Apache

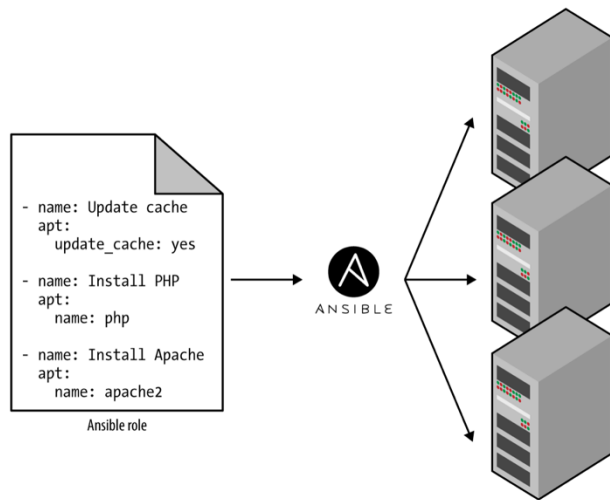
 - `service`: name=apache2 state=started enabled=yes

Ansible hoạt động như thế nào



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",  
  ]  
}]
```

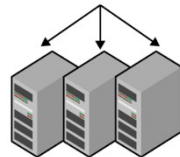
Packer Template

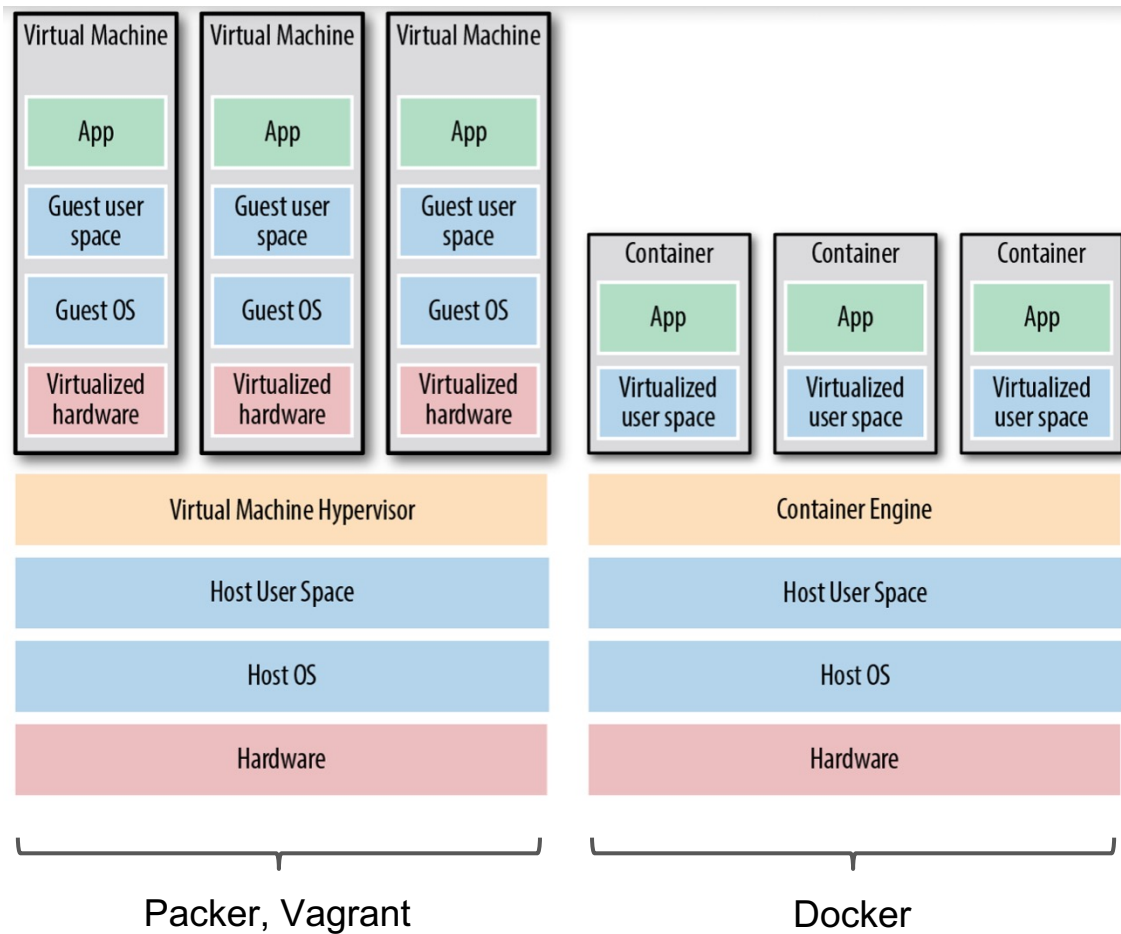


Server image



ANSIBLE

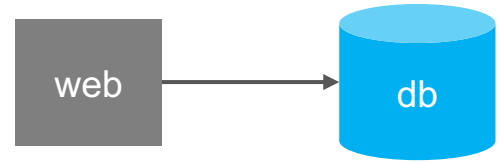




```
Vagrant.configure("2") do |config|
  config.vm.provision "shell", inline: "echo Hello"

  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
```

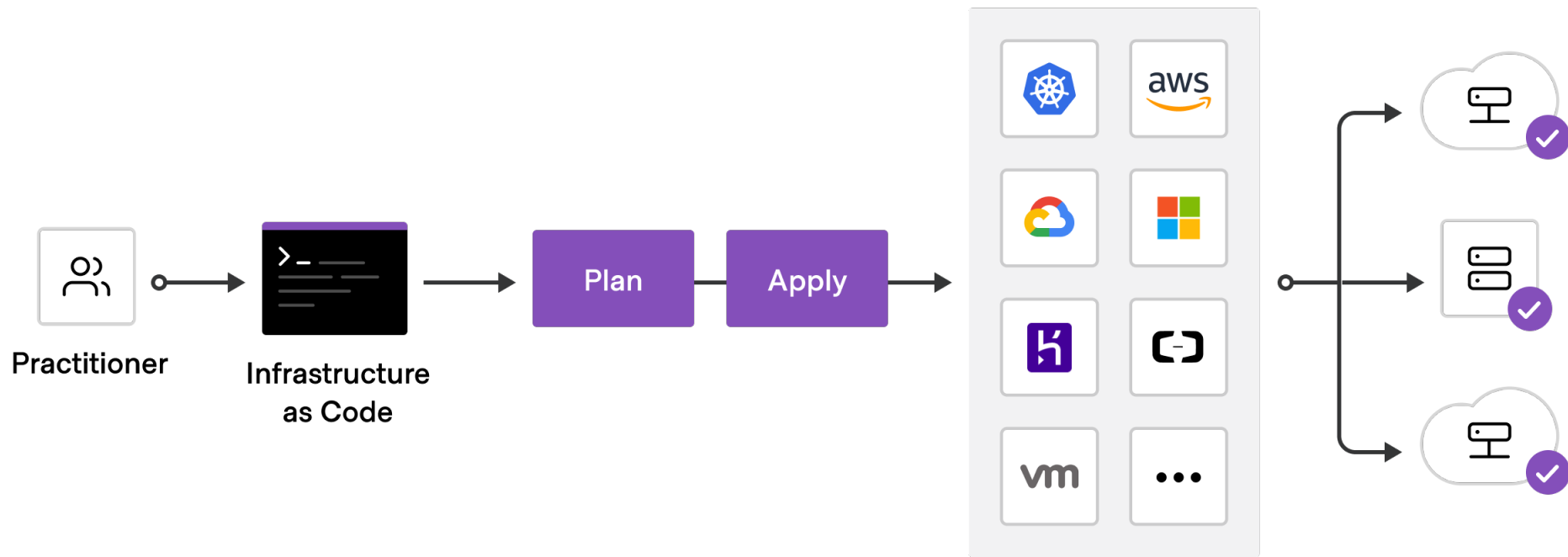


Kịch bản Vagrant dựng
2 máy ảo web và db

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-0c55b159cbfafa1f0",
    "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 + AWS



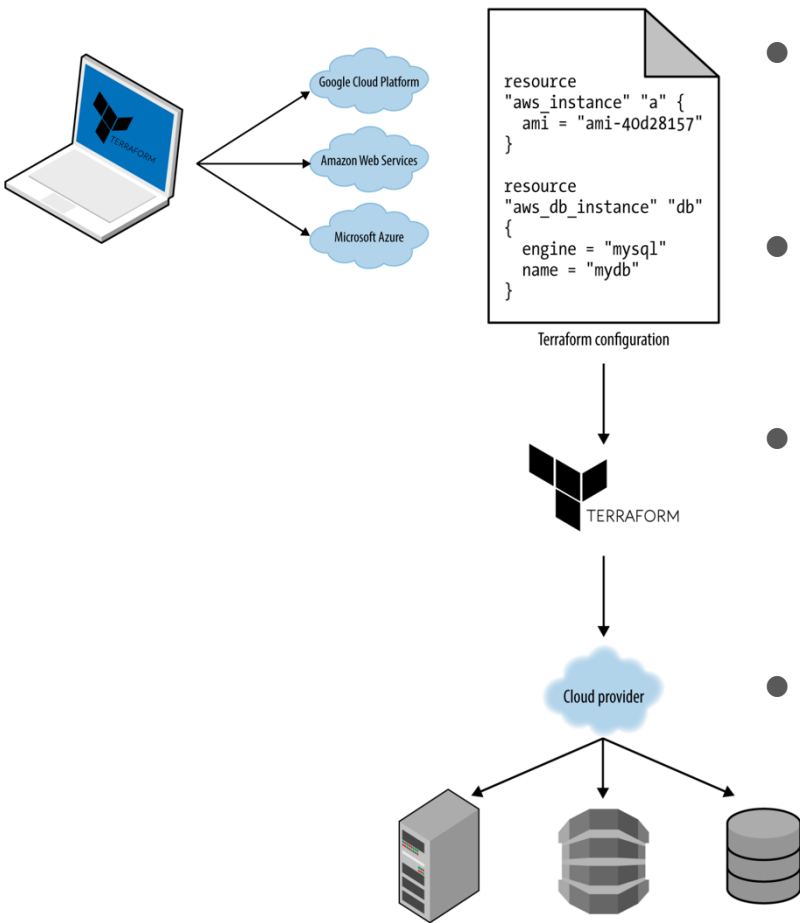
Ứng dụng của Terraform

- 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



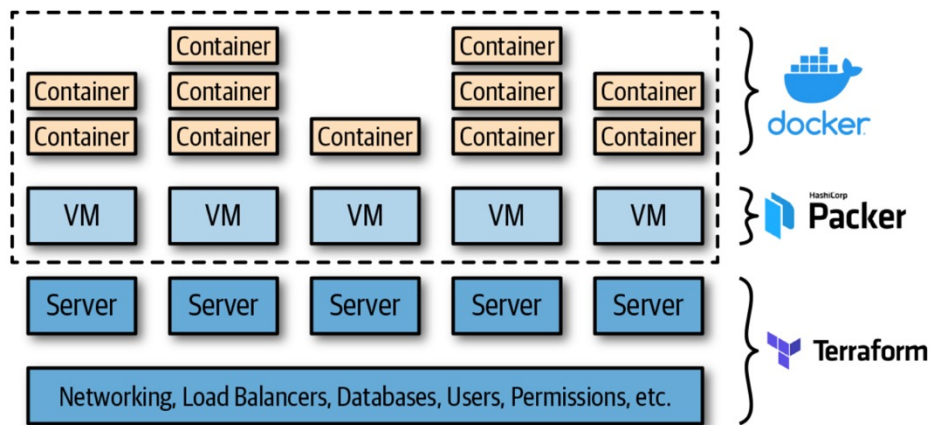
- Terraform mã nguồn mở, dùng miễn phí.
- Hỗ trợ nhiều providers
- Viết kịch bản (scripting) hoặc lập trình (programming)

Terraform khác gì với các công cụ IAC?

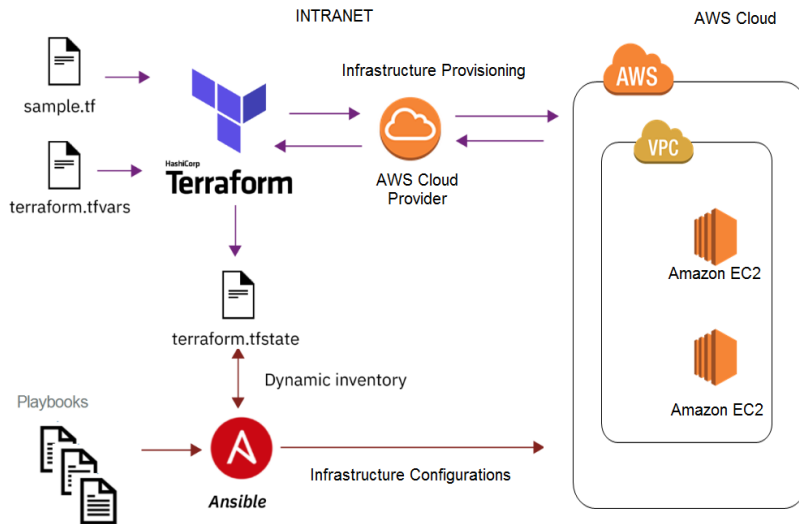


- Viết kịch bản (**scripting language**) tự động hoá triển khai máy chủ trên các provider khác nhau: aws, google cloud, azure, k8s, docker...
- Terraform còn có CDK (Cloud Development Kit) cho phép lập trình (**programming language**) bằng Typescript, Go, Python, C#, Java để dựng hệ thống.
- Terraform làm tốt phần Provisioning and Orchestration Infrastructure. Còn Ansible làm tốt phần tự động hoá cài đặt, cấu hình trên Infrastructure dựng xong bởi Terraform.
- Cú pháp Terraform là khai báo, còn Ansible là lập trình

Terraform phối hợp với các công cụ IAC khác



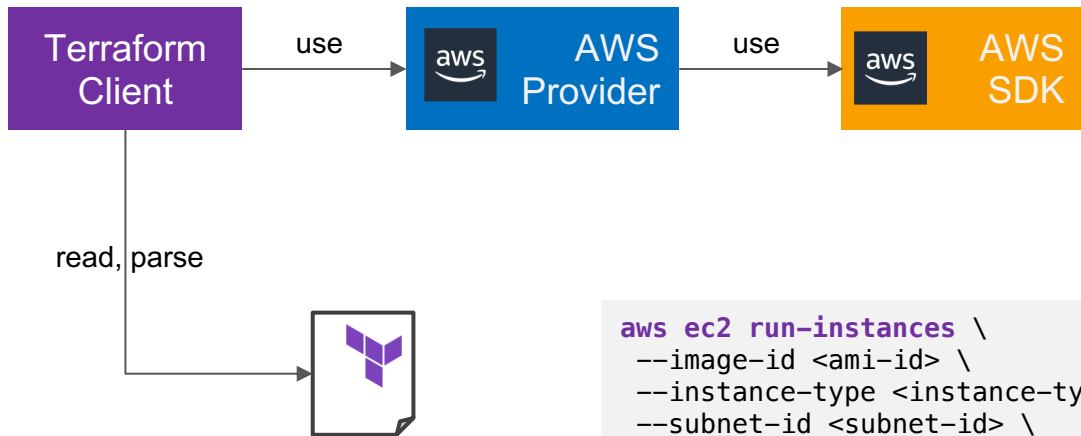
Kết hợp với Packer, Docker



Kết hợp với Ansible

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)



```
aws ec2 run-instances \
  --image-id <ami-id> \
  --instance-type <instance-type> \
  --subnet-id <subnet-id> \
  --security-group-ids <security-group-id> <security-group-id> ... \
  --key-name <ec2-key-pair-name>
```

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 trong 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"  
    }  
}
```

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



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

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"]
  }

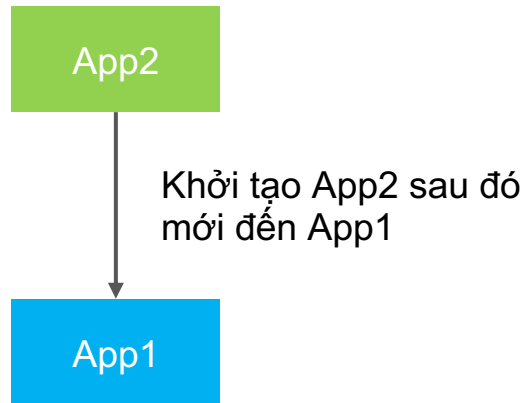
  owners = ["099720109477"] # Canonical
}

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

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

depends_on : tạo phụ thuộc giữa các resource

```
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"  
  }  
}
```

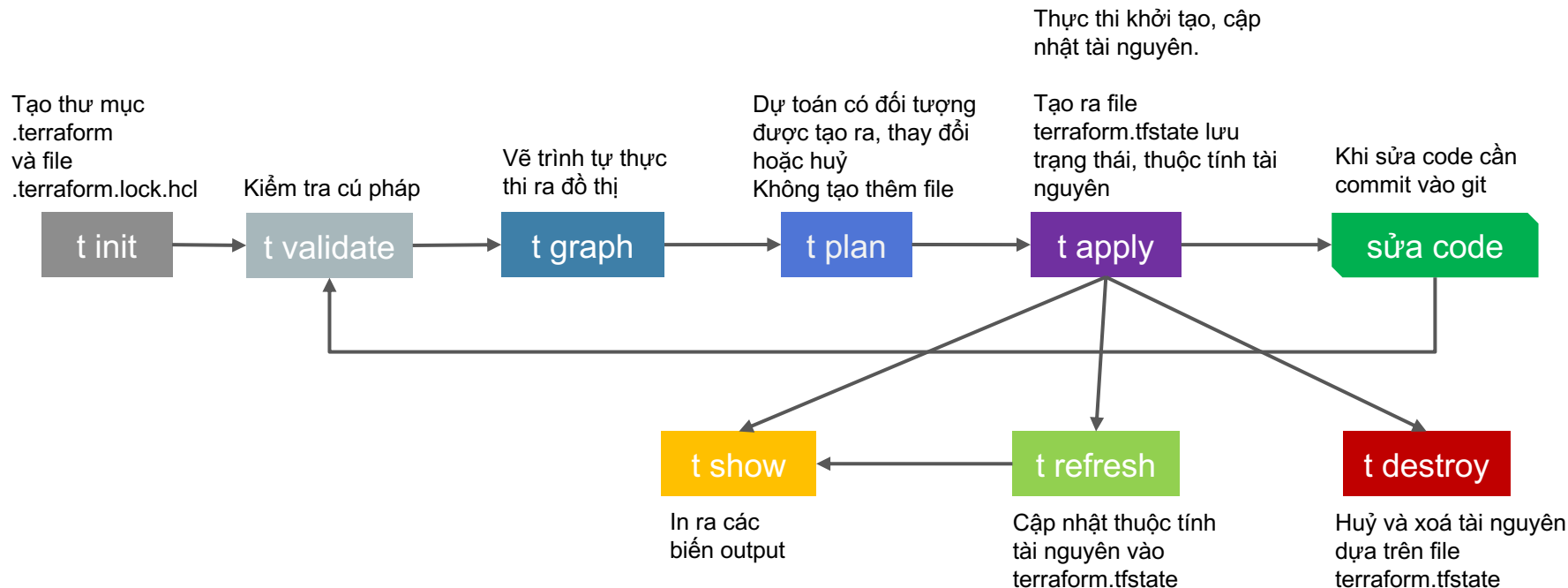


Chú ý khi dùng depends_on

- Tránh circular reference A depends_on B, B depends_on A
- 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



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"
  }
}
```



Details

Security

Networking

Storage

Status checks

Monitoring


Tags

▼ Security details


IAM Role

-

Owner ID

 952317537183

Security groups

 [sg-021b4098d34166443 \(sec_open_8080\)](#)

▼ Inbound rules

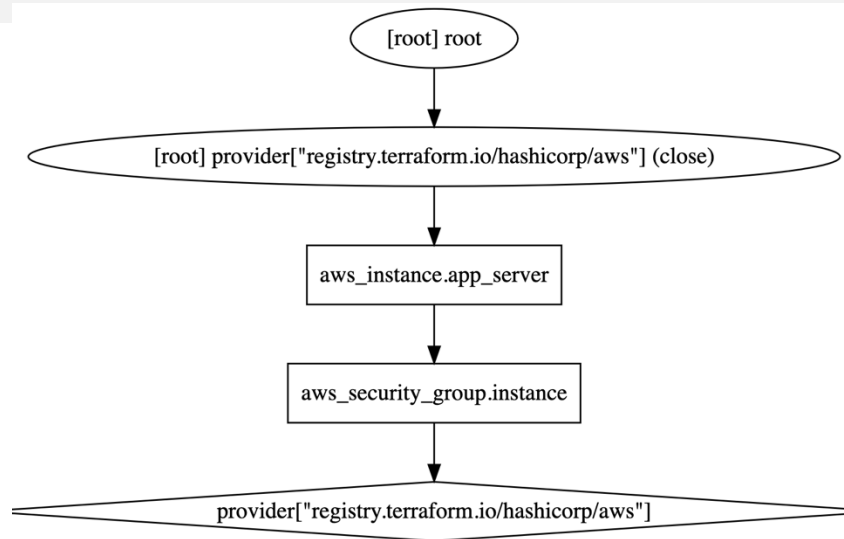
Filter rules

Security group rule ID	Port range	Protocol	Source	Security groups
sgr-07ba8d724e268fbd8	8080	TCP	0.0.0.0/0	sec_open_8080

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, Biến

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  
}
```



Outputs:

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

```
$ terraform output  
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 7: 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 7: tạo keypair #1

```
resource "aws_instance" "web" {  
  ami          = "ami-0ff89c4ce7de192ea"  
  instance_type = "t2.micro"  
  key_name      = "${aws_key_pair.generated_key.key_name}"  
  
  tags = {  
    Name = "HelloWorld"  
  }  
}  
  
locals {  
  instance_ip = aws_instance.web.public_ip  
}  
  
output "ssh_command" {  
  value = "ssh -i '${var.keyname}.pem' ec2-user@${local.instance_ip}"  
}
```

Gắn keypair vào EC2

Khởi tạo một biến cục bộ 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}"
```

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" {  
  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  
}
```

main.tf



```
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
```

◀ ▶ ↻ 🔖 ⚠ 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 exgress (outbound)
- from_port ... to_port: dải 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

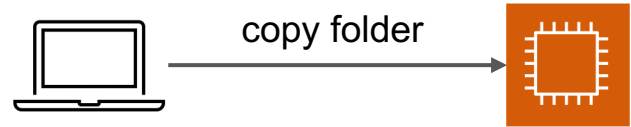
Lab 9b: triển khai web site nginx

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"  
}
```

security_group.tf

```
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  
  protocol  = "tcp"  
  cidr_blocks = ["0.0.0.0/0"]  
  description = var.ports[count.index].description  
  security_group_id = aws_security_group.app_secgroup.id  
}
```

Lấy giá trị từng phần tử trong list gán vào

Trở lên resource khai báo phía trên

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

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

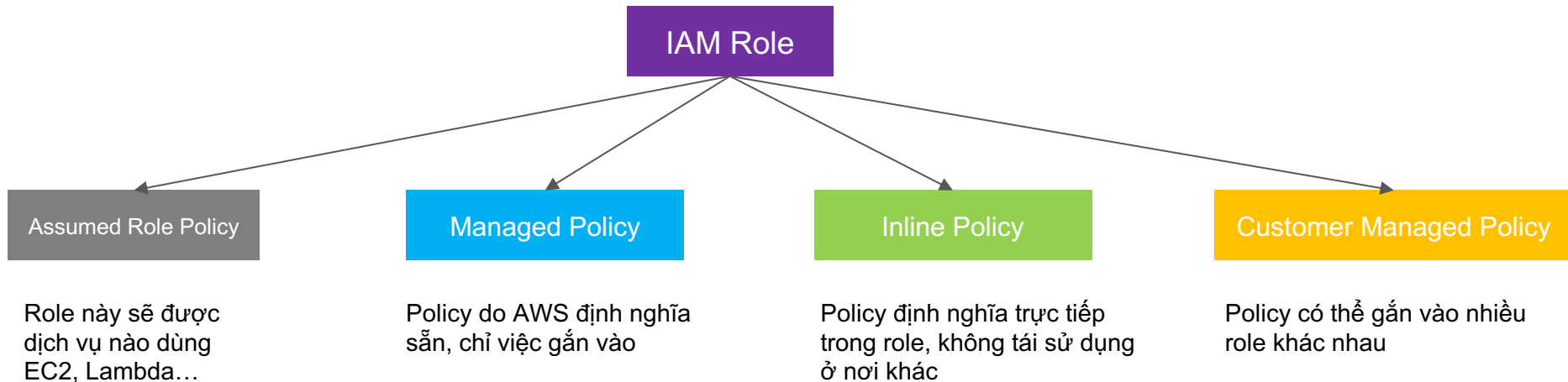


Xem thêm ví dụ ở đây
<https://www.terraform.io/language/expressions/for>

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

IAM Role

Những điểm cần nhớ về IAM Role



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
  AWSSSODirectoryAdministrator	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: gán 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"]  
}
```

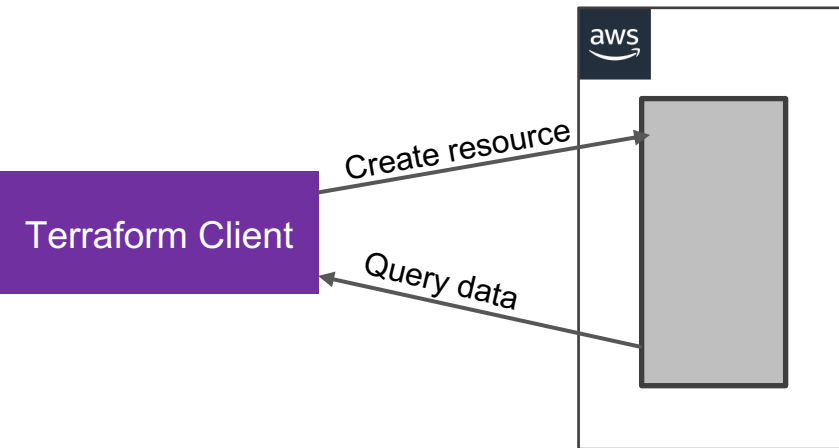
```
resource "aws_iam_instance_profile" "s3_rds_profile" {  
  name = "S3_RDS_Profile"  
  role = aws_iam_role.s3_rds_role.name  
}  
  
resource "aws_instance" "app_server" {  
  ami           = "ami-0ff89c4ce7de192ea"  
  instance_type = "t2.micro"  
  iam_instance_profile =  
    "${aws_iam_instance_profile.s3_rds_profile.name}"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



aws ⓘ

AWS DOCUMENTATION

Q Filter

> EBS (EC2)

▼ EC2 (Elastic Compute Cloud)

> Resources

▼ Data Sources

- aws_ami
- aws_ami_ids
- aws_availability_zone
- aws_availability_zones
- aws_ec2_host
- aws_ec2_instance_type
- aws_ec2_instance_type_offering
- aws_ec2_instance_type_offerings
- aws_ec2_instance_types
- aws_ec2_serial_console_access
- aws_ec2_spot_price
- aws_eip
- aws_eips
- aws_instance
- [aws_instances](#)
- aws_key_pair
- aws_launch_template

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  
}
```

out.tf

```
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  
  }  
}
```



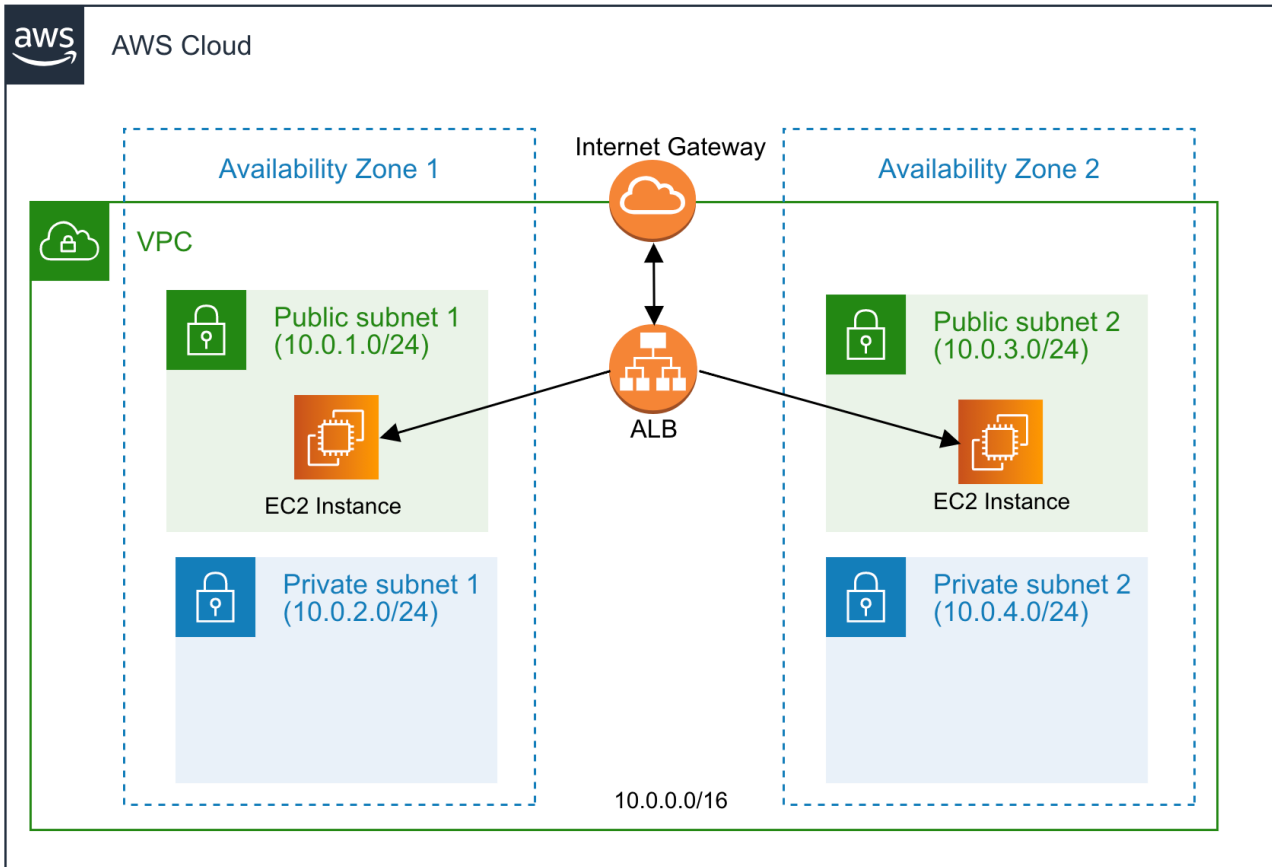
```
amazon_ami = {  
  "desc" = "Amazon Linux AMI 2018.03.0.20220705.1 x86_64 HVM ebs"  
  "id" = "ami-083689f4a841ee07a"  
  "name" = "amzn-ami-hvm-2018.03.0.20220705.1-x86_64-ebs"  
}  
availability_zones = tolist([  
  "ap-southeast-1a",  
  "ap-southeast-1b",  
  "ap-southeast-1c",  
)  
running_instances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

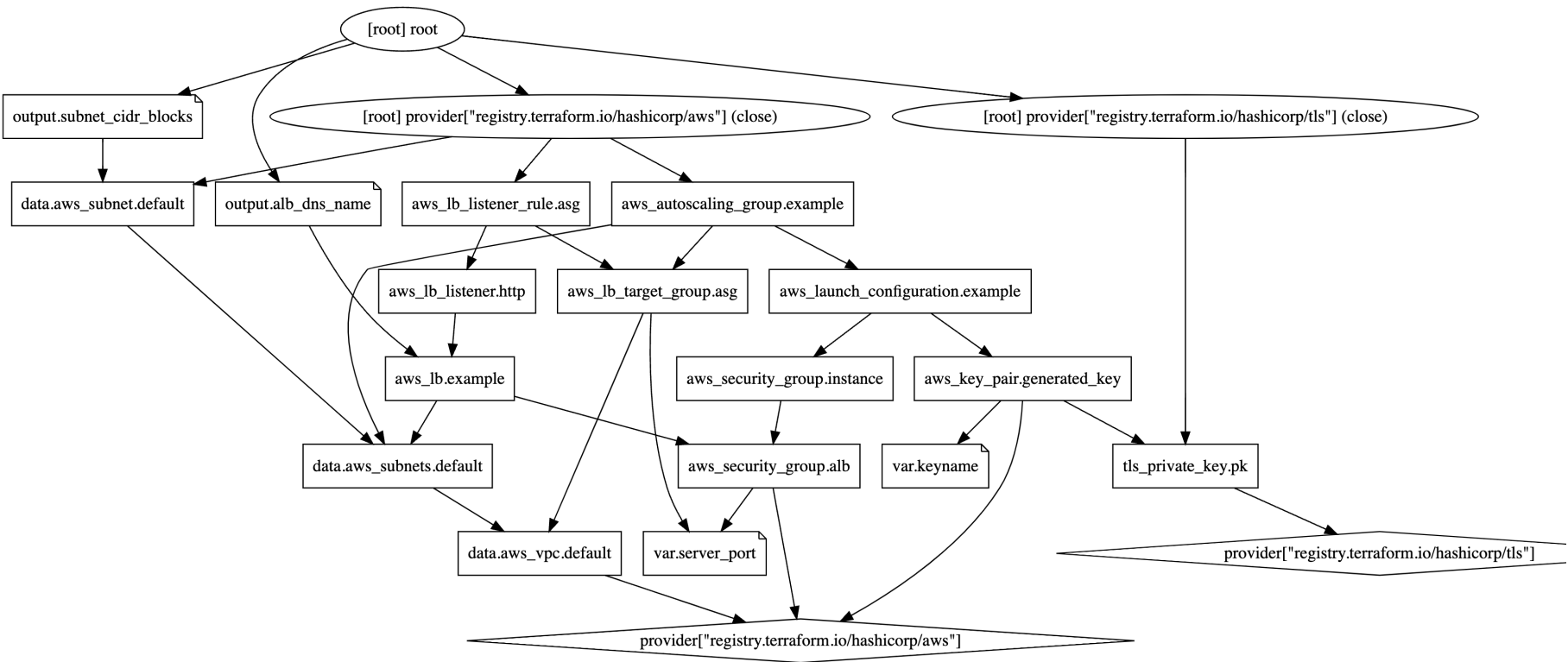
VPC

- <https://adamtheautomator.com/terraform-vpc/>
- <https://www.howtoforge.com/create-a-vpc-on-aws-using-terraform/>

Autoscaling

Bài tập





Gắn SSL vào Load Balancer

Sử dụng ACM SSL

- 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>

SS

S3

RDS

DynamoDB