

Sử dụng Terraform và Ansible để triển khai các dịch vụ AWS



Infrastructure As Code với Terraform và Ansible



Terraform



Được thiết kế chuyên biệt cho tự động hóa cơ sở hạ tầng (infrastructure automation) và không phụ thuộc vào một nhà cung cấp dịch vụ đám mây cụ thể (có nghĩa là nó không bị ràng buộc với một nhà cung cấp đám mây nào). Làm việc với tất cả các nhà cung cấp đám mây phổ biến (AWS/GCP/Azure).



Cung cấp các hàm tích hợp sẵn (built-in functions), và một kho tàng modules và nhà cung cấp dịch vụ đám mây (providers) để hỗ trợ làm việc với các hệ thống đám mây và hệ thống on-prem



Dễ dàng viết code sử dụng ngôn ngữ HCL (Hashicorp Configuration Language) dưới dạng các mẫu khai báo (declarative templates)



Không cần agent, chỉ cần cài đặt Terraform trong hệ điều hành của bạn



Theo dõi cơ sở hạ tầng thông qua các file trạng thái (state files) được lưu trữ cục bộ hoặc từ xa. Người dùng phải sao lưu state file.

Ansible



Ansible là công cụ quản lý cấu hình và triển khai cơ sở hạ tầng, là công cụ phổ biến cho quản lý cấu hình



Ansible sử dụng ngôn ngữ YAML đơn giản và dễ hiểu



Ansible không theo dõi trạng thái việc triển khai cơ sở hạ tầng như CloudFormation và Terraform



Cung cấp cả hai phương pháp tiếp cận, một là thông qua các tác vụ thủ tục (như ad-hoc tasks hoặc playbooks), và hai là thông qua các module khai báo (như AWS) để quản lý hạ tầng.

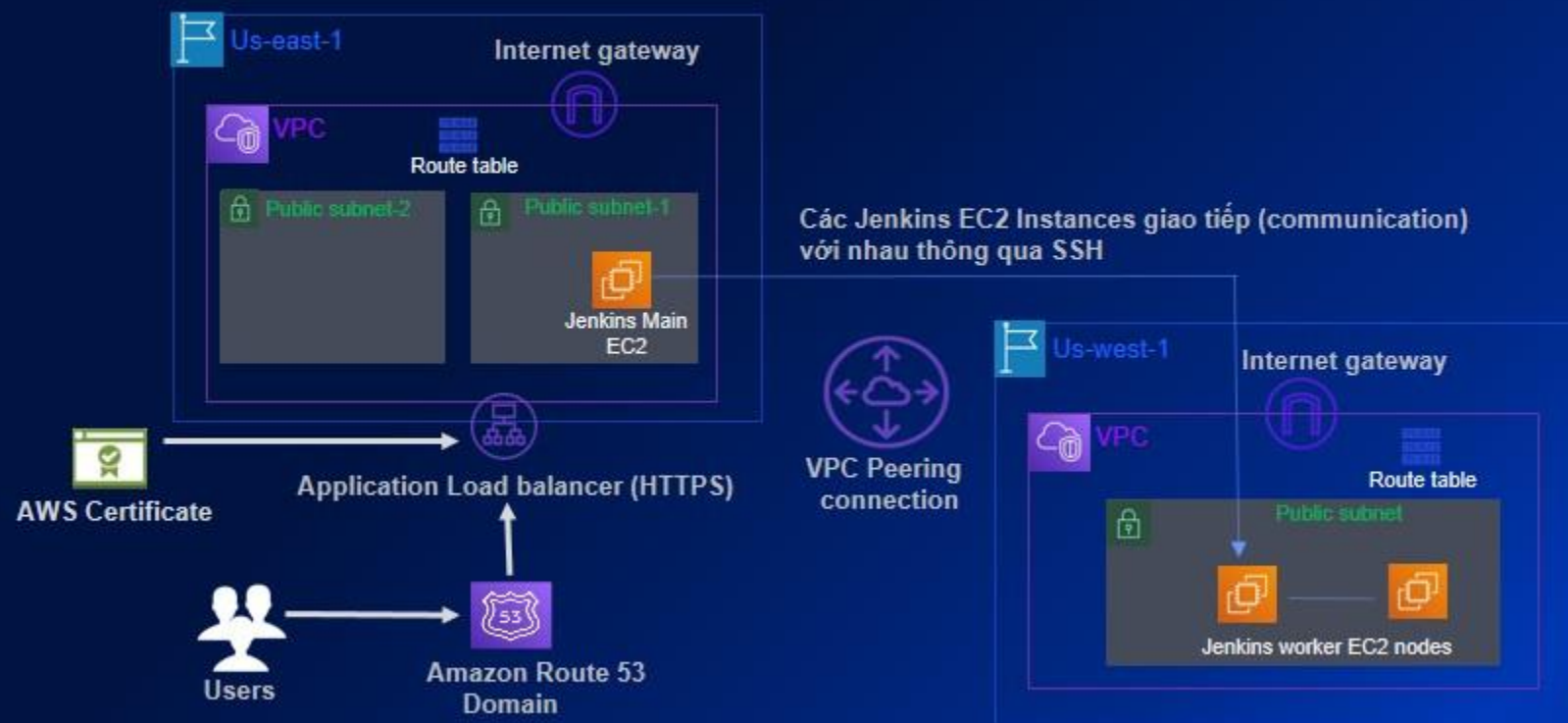


Cung cấp một kho tàng modules cho hầu hết các tiện ích cấp hệ điều hành cũng như nhà cung cấp cơ sở hạ tầng đám mây và on-prem.



Không sử dụng agent, truy cập vào API của các nhà cung cấp (providers) cơ sở hạ tầng để triển khai cơ sở hạ tầng và SSH để quản lý cấu hình của hệ điều hành.




Triển khai hệ thống Multi-Region, Jenkins CI/CD sử dụng Terraform và Ansible



Chú ý quan trọng về điều kiện để tham gia khóa học



Sử dụng Terraform và Ansible để triển khai các dịch vụ AWS

- 1 Để tham gia khóa học này, bạn cần có hiểu biết cơ bản về các dịch vụ AWS như EC2, Route 53, VPC, AWS CLI, biết một chút về Ansible và Terraform thì là một điểm cộng. 
- 2 Cần có một kiến thức cơ bản về các lệnh Linux, và Linux Bash Shell, hoặc Python. 
- 3 Nếu có kiến thức cơ bản về CI/CD hoặc Jenkins sẽ cũng rất tốt để tham gia khóa học. 
- 4 Học viên cần kiên nhẫn để học và làm theo các bài giảng. Vì chúng ta sẽ xây dựng từng bước một các Terraform Blocks và Ansible Playbooks cho dự án này từ đầu đến cuối.



Setup môi trường Terraform (Terraform Environment)



Download Terraform binary (File cài đặt Terraform)
cho OS của bạn từ Hashicorp website.

Link: <https://developer.hashicorp.com/terraform/install>



Thêm đường dẫn tới thư mục chứa các files thực thi của
Terraform vào biến môi trường PATH để chúng ta có thể dễ
dàng truy cập và sử dụng Terraform tại tất cả các vị trí trong
hệ điều hành Linux.

Test và xác nhận có thể sử dụng được Terraform



Cấp quyền truy cập cho Terraform



Terraform sẽ cần phải có các quyền (Permissions) để tạo, cập nhật (update), và xóa (delete) các tài nguyên AWS.



Bạn có thể thực hiện theo 02 cách sau, tùy thuộc vào cách mà bạn muốn triển khai:

1. Tạo một IAM user có các quyền cần thiết. Ví dụ như admin user.
2. Tạo một EC2 (IAM role) instance profile có các quyền cần thiết và gán (attach) nó cho EC2.



Setup AWS CLI và Ansible



Tùy thuộc vào hệ điều hành (OS), sử dụng lệnh yum, apt-get, dnf để setup python pip (Python Package Installer).



- Cài đặt Ansible và download Ansible configure file (Link trong phần tài nguyên).
- Cài đặt AWS CLI, và cấu hình AWS CLI (aws configure)



Tìm hiểu về Teraform fmt, validate, plan và apply

Terraform init (initialize)

- 1 **Khởi tạo thư mục làm việc**
Tải xuống và bao gồm tất cả các module và Provider (ngoại trừ bên thứ ba) trong Terraform file.
- 2 **Cần được chạy trước khi triển khai cơ sở hạ tầng**
Vì các giai đoạn khác của việc triển khai Terraform yêu cầu provider, plugins và modules, lệnh này cần phải được chạy đầu tiên.
- 3 **Đồng bộ hóa cấu hình, an toàn để chạy**
Lệnh **terraform init** sẽ thiết lập cấu hình backend để lưu trữ trạng thái của cơ sở hạ tầng. Nó sẽ không làm thay đổi hoặc xóa bất cứ cài đặt cấu hình hoặc trạng thái hiện tại nào, mà chỉ đồng bộ hóa cấu hình để chuẩn bị cho việc triển khai.



Terraform fmt (format)

- 1 **Định dạng các templates**
Làm các Terraform templates trông dễ nhìn và dễ đọc.
- 2 **Giúp duy trì code một cách nhất quán**
Duy trì nhất quán định dạng code, đặc biệt là nếu các nhóm phát triển đang cùng làm việc và theo dõi Terraform code thông qua việc kiểm soát phiên bản.
- 3 **An toàn để chạy tại mọi thời điểm**
không thay đổi hoặc thêm bất kỳ code mới nào vào file Terraform hiện tại. Thay vào đó, nó chỉ đơn giản làm cho code hiện tại trông đẹp hơn, dễ đọc hơn và tuân thủ theo một định dạng chuẩn.



Terraform validate

- 1 **Kiểm tra các files cấu hình (config files)**
Kiểm tra lỗi cú pháp và tính nhất quán bên trong (ví dụ: Lỗi chính tả và lỗi cấu hình tài nguyên sai).
- 2 **Cần phải chạy terraform init trước**
Yêu cầu một thư mục làm việc được khởi tạo, do đó lệnh init cần được chạy trước khi validate có thể chạy.
- 3 **An toàn để chạy bất cứ lúc nào**
Trường hợp sử dụng là chạy lệnh này để kiểm tra các vấn đề trong TF code trước khi commit vào version control.



Terraform plan

- 1 **Tạo ra kế hoạch thực thi**
Terraform tính toán sự khác biệt giữa trạng thái yêu cầu và trạng thái hiện tại để tạo ra một kế hoạch thực thi.
- 2 **An toàn trước khi triển khai thật**
Đây là một bước kiểm tra để xem liệu kế hoạch thực thi phù hợp với kế hoạch mà chúng ta muốn, trước khi tạo ra hoặc sửa đổi cơ sở hạ tầng thật.
- 3 **Kế hoạch thực thi có thể được lưu lại bằng cách sử dụng tham số **-out****
Tuy nhiên, cần phải nhận thức là các cấu hình nhạy cảm cũng sẽ được lưu lại trong một file dưới dạng văn bản.



Terraform apply (Deploy)

- 1 **Triển khai kế hoạch thực thi!**
Áp dụng các thay đổi cần thiết để đạt được trạng thái mong muốn của Terraform code.
- 2 **Mặc định sẽ hiển thị thông báo trước khi triển khai** Theo mặc định, người dùng sẽ cần phải gõ "yes" một cách rõ ràng trước khi một cơ sở hạ tầng được triển khai.
- 3 **Sẽ hiển thị kế hoạch thực thi một lần nữa**
Terraform apply sẽ hiển thị kế hoạch (plan) thực thi một lần nữa trước khi yêu cầu triển khai thật các tài nguyên.



Lưu trữ trạng thái (State) Terraform trong S3 Backend

Terraform Backends

- 1 Xác định cách lưu trữ trạng thái
- 2 Mặc định, Terraform state được lưu trữ trong Local disk
- 3 Các biến không thể được sử dụng trong cấu hình Backends

Ví dụ backend trong một Terraform block sử dụng S3

```
terraform
{
  required_version = ">=0.12.0"
  backend "s3"
  {
    region = "us-east-1"
    profile = "default"
    key    = "<arbitrary-state-file-name>"
    bucket = "<name-of-already-created-bucket>"
  }
}
```


Set up nhiều AWS Providers trong Terraform

Set up nhiều AWS Providers trong Terraform

- ✓ Các providers thực hiện các tương tác với API của nhà cung cấp như AWS và Azure. Chúng cũng cung cấp logic để quản lý, cập nhật và tạo các tài nguyên trong Terraform.

Khai báo nhiều nhà cung cấp AWS (AWS Providers)

```
Provider "aws"  
{  
  profile = var.profile  
  region = var.region-master  
  alias = "region-master"  
}
```

```
Provider "aws"  
{  
  profile = var.profile  
  region = var.region-worker  
  alias = "region-worker"  
}
```

**Network Setup Phần 1:
Triển khai VPCs, Internet GWs
và Subnets**

Mục đích của bài học:

Triển khai hệ thống mạng



Chú ý

Setup S3 Backend và nhiều AWS Providers

Đừng quên setup S3 backend và hai AWS Providers, như đã được học trong các videos của các bài học trước.



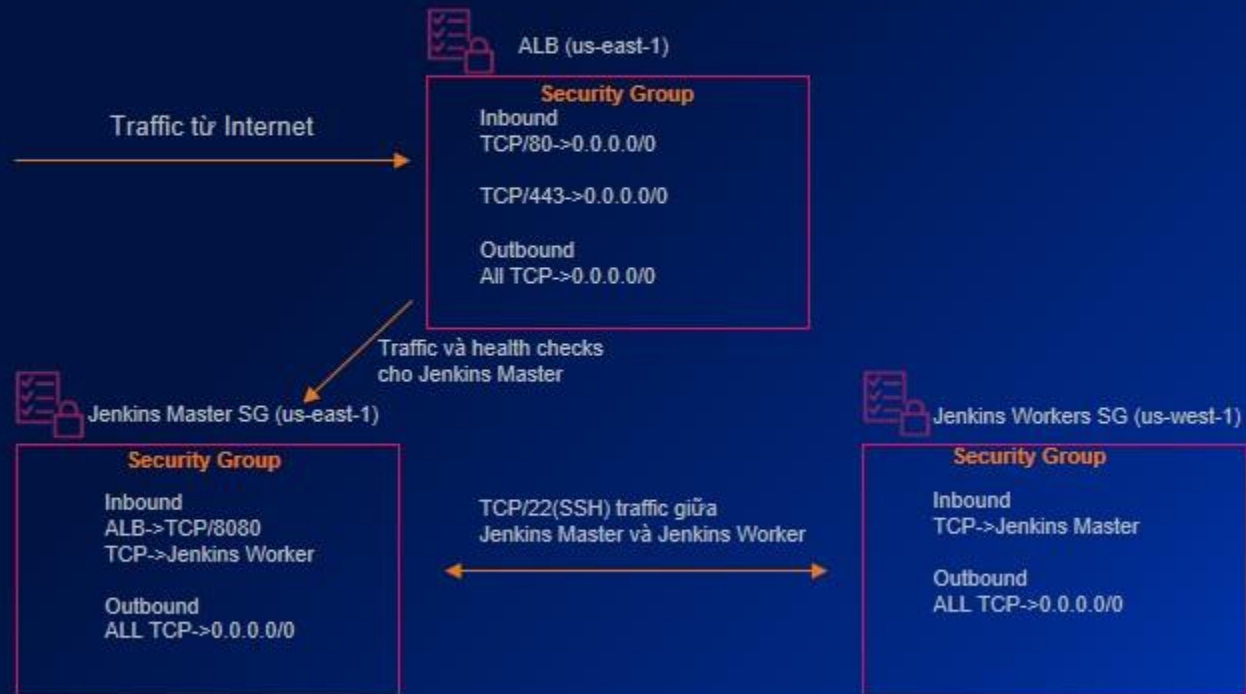
**Network Setup Phần 2:
Triển khai Multi-Region
VPC Peering**

Mục đích của bài học: Triển khai kết nối VPC Peering



Network Setup Phần 3: Triển khai Security Groups

Mục đích của bài học: Triển khai Security Groups

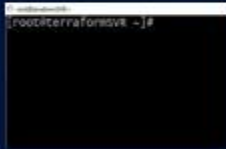


Tạo một Multi-Region Network với VPC Peering sử dụng SGs, IGW, và RTs

GIỚI THIỆU VỀ BÀI THỰC HÀNH NÀY

Việc theo dõi tất cả các thành phần định tuyến khác nhau của một hệ thống mạng có thể trở nên phức tạp, đặc biệt là trong môi trường mà các hoạt động IT thay đổi nhanh chóng như ngày nay. Bằng cách sử dụng Terraform để duy trì các tài nguyên AWS như VPC, SG và IGW giúp bạn quản lý và theo dõi tất cả các sự thay đổi của các tài nguyên hạ tầng hiệu quả hơn và giúp việc tự động hóa hệ thống dễ dàng hơn.

Trong bài thực hành này, bạn sẽ thực hiện tạo một thiết lập mạng hoàn chỉnh với VPCs, subnets, security group, internet gateway và VPC Peering trong AWS sử dụng Terraform. Bạn cần có một hiểu biết nhất định về VPC và các thành phần mạng cơ bản trong AWS để có thể vận dụng linh hoạt bài lab này.



Login vào Terraform
Controller Node

Giai đoạn 1



Clone GitHub Repo
cho Terraform Code

Giai đoạn 2



Triển khai
Terraform Code

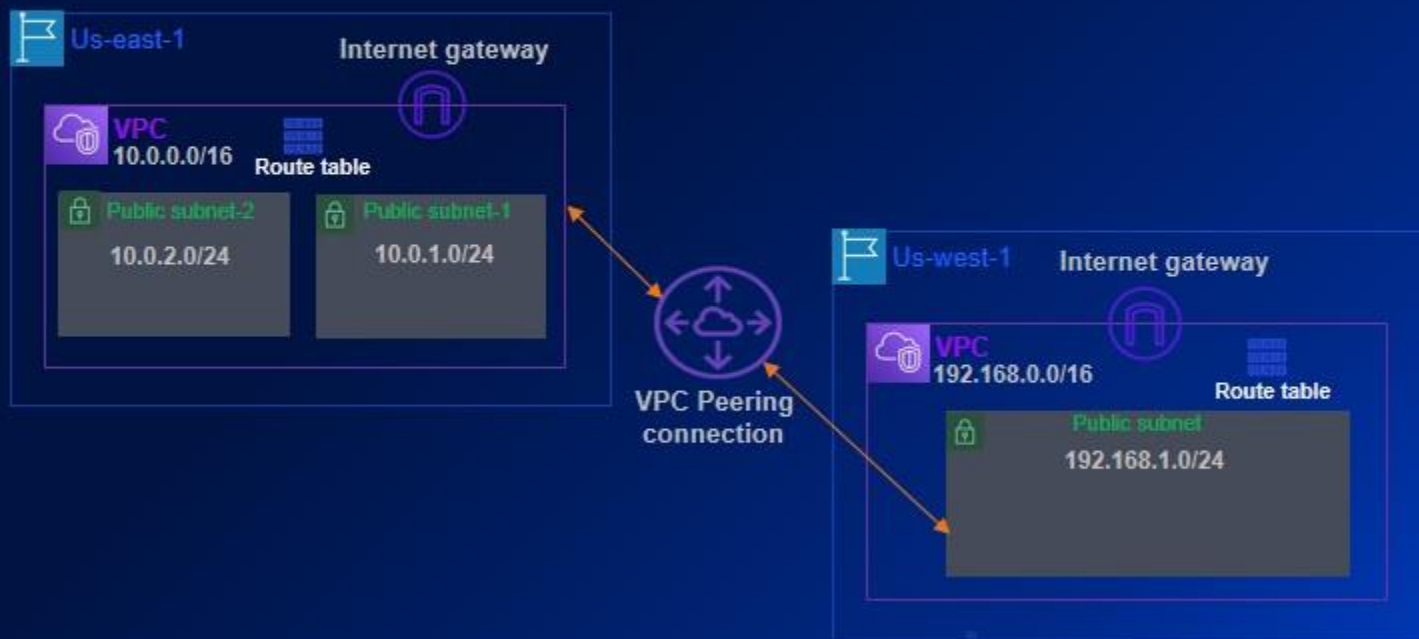
Giai đoạn 3



Kiểm tra các tài nguyên đã
được tạo trong AWS Console.
Hoàn thành bài Lab

Giai đoạn 4

Tạo một Multi-Region Network với VPC Peering sử dụng SGs, IGW, và RTs



**Triển khai App VM phần 1:
Sử dụng Data Source (SSM
Parameter Store) để lấy thông
tin AMI IDs**

Mục đích của bài học:

Lấy thông tin AMI IDs sử dụng SSM Parameter Store

Terraform Data Source cho SSM Parameter

```
data "aws_ssm_parameter" "AmazonOfficialAMI"
{
  Name = "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2"
}
```

SSM Parameter Store - Parameters cho Public AMI IDs

```
{
  "Parameters": [
    {
      "Name": "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
      "Type": "String",
      "Value": "ami-0cd5dfb4e35928968"
    }
  ]
}
```

Terraform SSM Data Source trả về AMI ID

```
data.aws_ssm_parameter.AmazonOfficialAMI.value == ami-0cd5dfb4e35928968
```

<https://docs.aws.amazon.com/systems-manager/latest/userguide/parameter-store-public-parameters.html>

**Triển khai App VM phần 2:
Triển khai Key Pairs cho
App Nodes**

Mục đích của bài học: Triển khai Key Pairs cho App Nodes

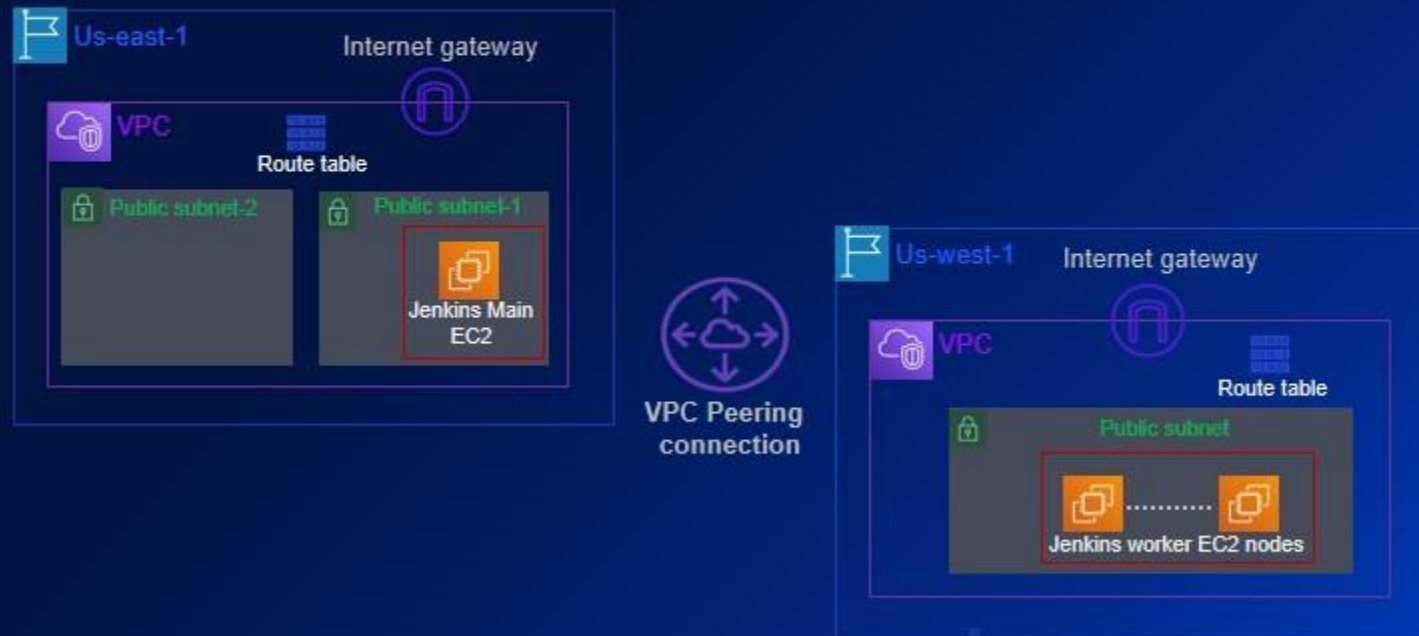


EC2 keypair cần được tạo ra và gắn vào một EC2 instance trước khi các EC2 instance đó được khởi chạy, vì keypair được tích hợp vào máy ảo trong quá trình khởi động.



**Triển khai App VM phần 3:
Triển khai Jenkins Master và
Worker Instances**

Mục đích của bài học:
Triển khai Jenkins Master và
Worker Instances



**Cấu hình Terraform
Provisioners để
Quản lý cấu hình qua Ansible**

Mục đích của bài học:

Bootstrapping cơ sở hạ tầng qua Terraform Provisioners

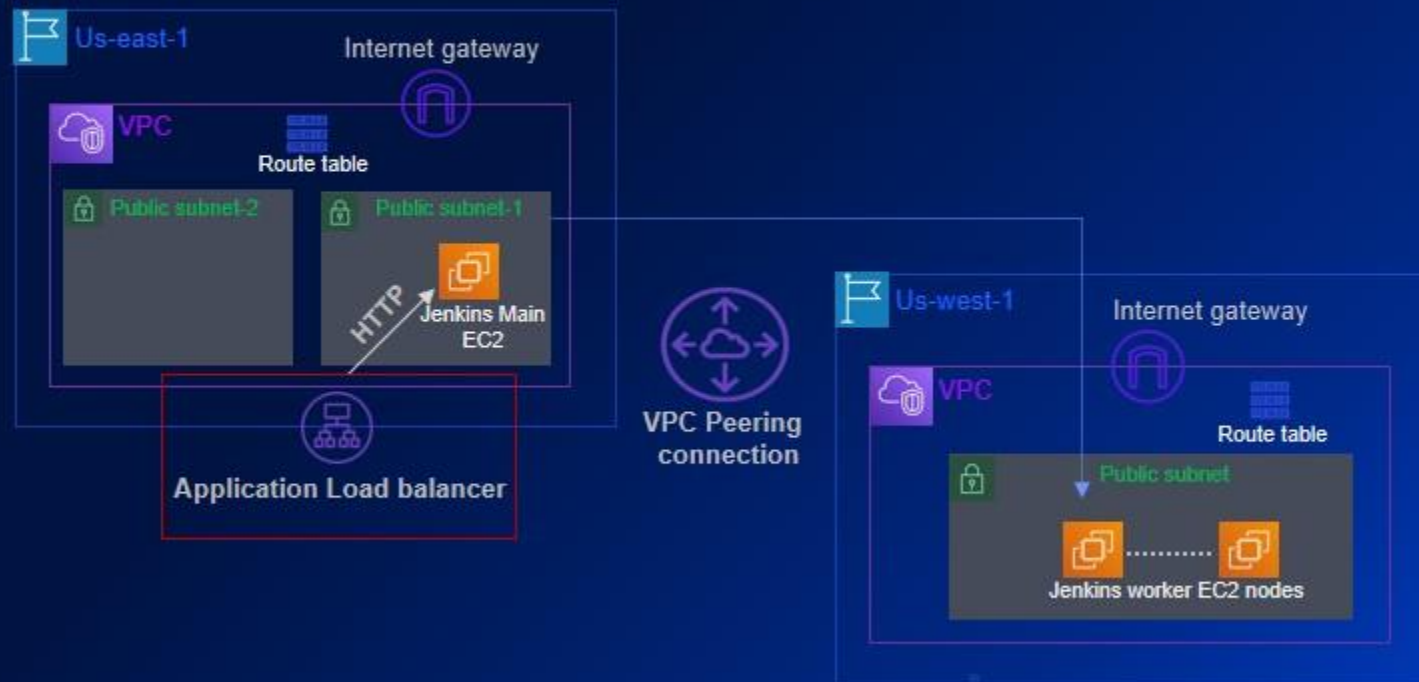
- 1 **Tạo (create) và hủy bỏ (destroy) time provisioners.**
Create và Destroy Time Provisioners giúp tự động hóa các tác vụ liên quan đến việc quản lý và duy trì các tài nguyên trong quá trình triển khai và hủy bỏ của hạ tầng của bạn.
- 2 **Nếu một provisioner thất bại khi chạy cho một tài nguyên trong Terraform, tài nguyên đó được đánh dấu là tainted (bị nhiễm)**
Một tài nguyên bị đánh dấu là tainted sẽ bị xóa bỏ (destroyed) trong lần tiếp theo thực hiện lệnh terraform apply và tạo lại tài nguyên.
- 3 **Provisioners có thể chạy lệnh cục bộ (locally) hoặc từ xa (sử dụng SSH hoặc WinRM)**
 - **Local Provisioners:** Chạy trên một máy tính mà các lệnh Terraform được gọi.
 - **Remote provisioners:** Chạy bên trong tài nguyên đang được tạo ra và yêu cầu một số biến kết nối như public keys. Chúng sử dụng các giao thức như SSH (Linux), WinRM (Windows).

Ví dụ sử dụng Provisioners bên trong một Terraform Resoure

```
resource "aws_instance" "jenkins-worker-cali"
{
  # ...
  provisioner "remote-exec"
  {
    when = destroy
    inline = [ " echo ' executing on the remote, provisioned instance' " ]
    connection {
      type      = "ssh"
      user      = "ec2-user"
      private_key = file("~/ssh/id_rsa")
      host      = self.public_ip
    }
  }
  provisioner "local-exec" {
    command = " echo 'Executing on the local server which is from the Linux Terraform Controller' "
  }
}
}
```

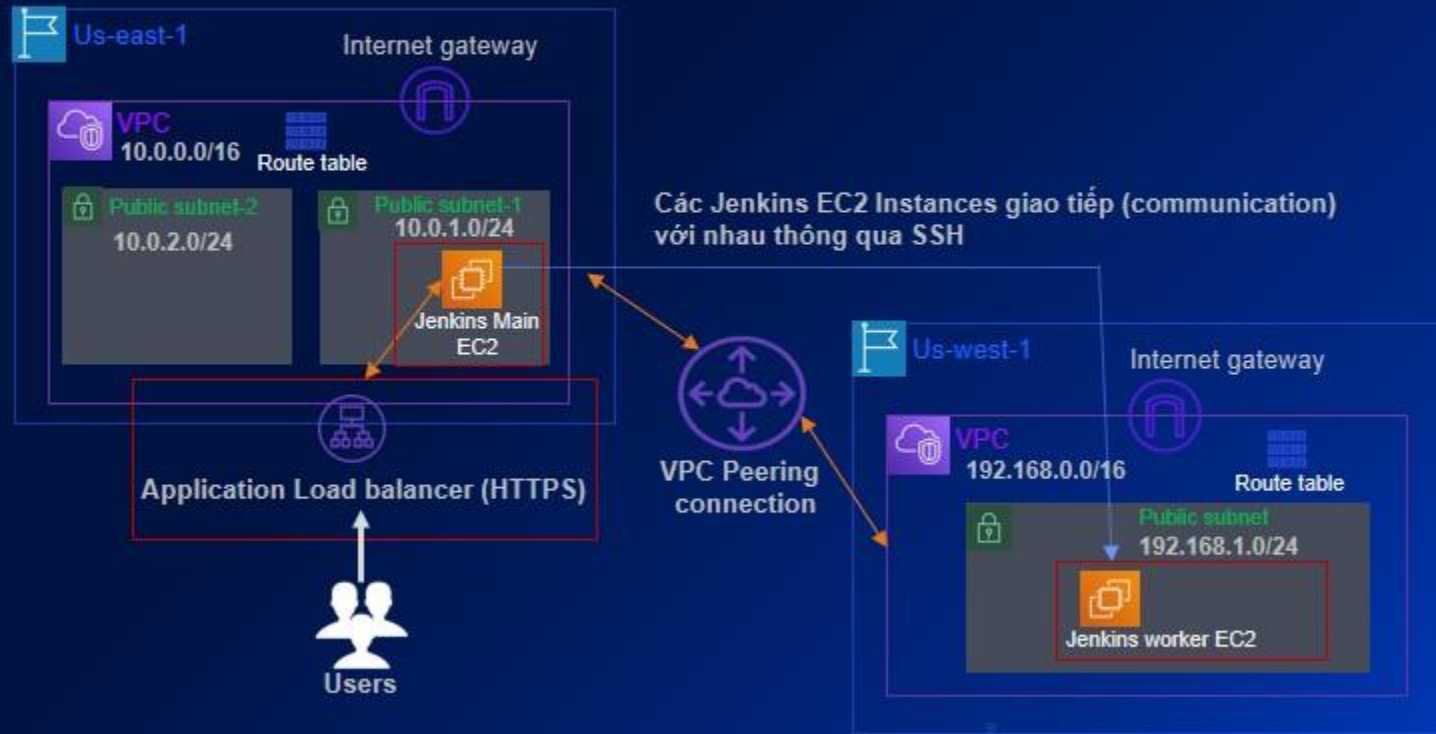
**Cấu hình một cân bằng tải
ALB và định tuyến traffic
tới EC2 App Node**

Mục đích của bài học:
Tạo một ALB và định tuyến traffic
tới EC2 node



**Sử dụng Terraform và Ansible để
triển khai Jenkins Master và
Worker Nodes đứng sau
một ALB**

Sử dụng Terraform và Ansible để triển khai Jenkins Master và Worker Nodes đứng sau một ALB



Sử dụng Terraform và Ansible để triển khai Jenkins Master và Worker Nodes đứng sau một ALB

GIỚI THIỆU VỀ BÀI THỰC HÀNH NÀY

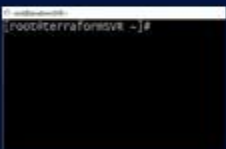
Trong bài thực hành này bạn sẽ sử dụng Terraform để triển khai Jenkins master và worker nodes trong các EC2 Instances(AWS) trong các regions, và sử dụng Ansible để quản lý ứng dụng và tích hợp giữa các Jenkins nodes.

✓ Chúng ta click vào đường link :

<https://raw.githubusercontent.com/phuongluuho/TerraformAnsible/main/Terraform-Lab1.json>

Sau đó sử dụng CloudFormation Stack. **Để tạo ra một môi trường cho bài thực hành gồm:**

01 máy ảo (EC2 instance & installed Terraform, Ansible, Python Boto3), 01 SecurityGroup, 01 VPC & Subnets, 01 Internet Gateway, 01 EC2InstanceProfile.



Login vào Terraform
Controller Node

Giai đoạn 1



Clone GitHub Repo
cho Terraform Code

Giai đoạn 2



Triển khai
Terraform Code

Giai đoạn 3

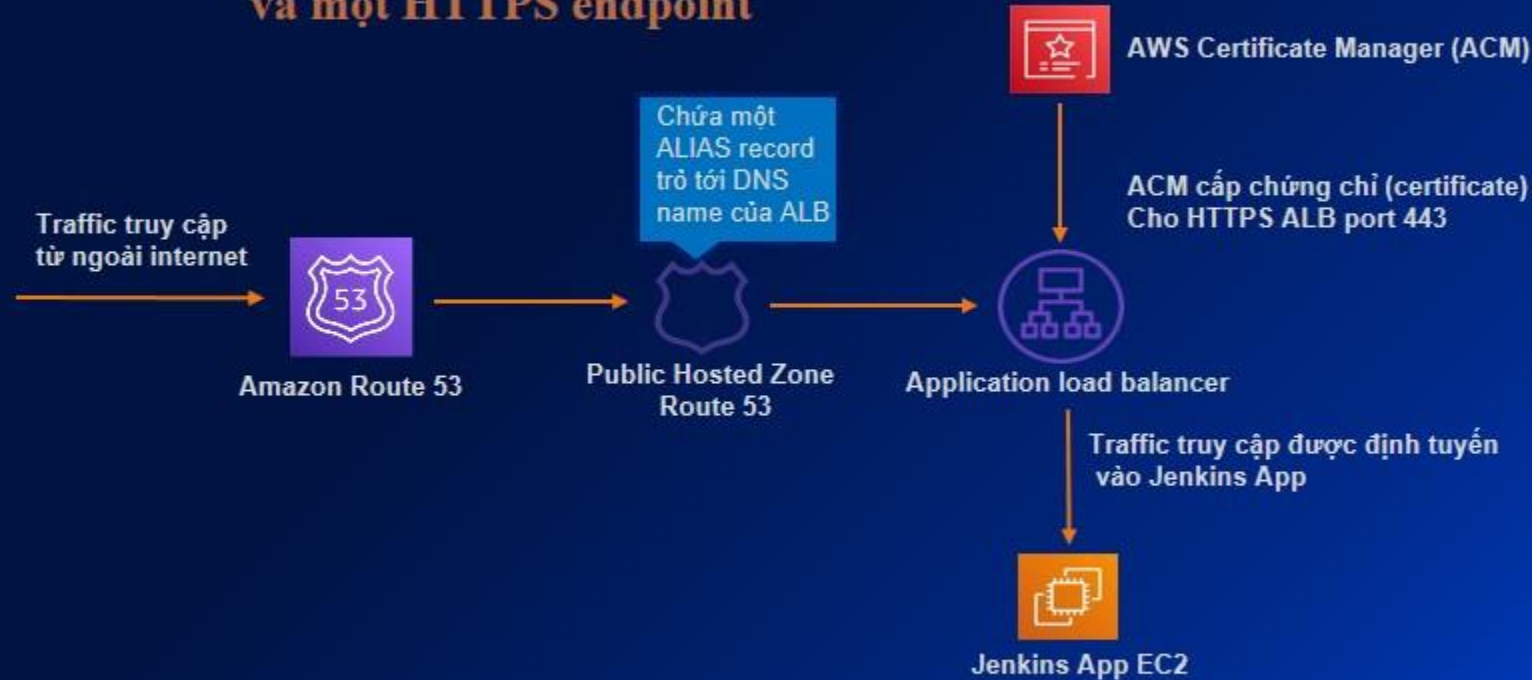


Truy cập vào Jenkins Master.
Hoàn thành bài Lab

Giai đoạn 4

Triển khai DNS, HTTPS và một Route 53 Record cho ALB

Mục đích của bài học:
Triển khai Route 53 Record (Public Hosted Zone)
và một HTTPS endpoint



Sử dụng Terraform tạo Route 53 Records (Alias) để định tuyến traffic truy cập vào ALB

GIỚI THIỆU VỀ BÀI THỰC HÀNH NÀY

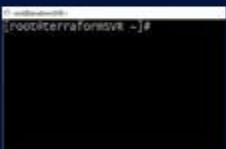
Trong bài thực hành này, bạn sẽ tạo một Route 53 alias record để định tuyến các traffic truy cập từ publicly hosted zone trong Route 53 (môi trường của bài thực hành này đã được tạo sử dụng file Terraform-Lab2.json) tới một cân bằng tải ALB sử dụng Terraform templates.

✓ Chúng ta click vào đường link :

<https://raw.githubusercontent.com/phuongluuho/TerraformAnsible/main/Terraform-Lab2.json>

Sau đó sử dụng CloudFormation Stack. **Để tạo ra một môi trường cho bài thực hành gồm:**

01 máy ảo (EC2 instance & installed Terraform), 02 máy ảo (EC2 instance & Installed Apache webserver), SecurityGroups, 01 VPC & Subnets, Internet Gateway, 01 EC2InstanceProfile.



Login vào Terraform
Controller Node

Giai đoạn 1



Clone GitHub Repo
cho Terraform Code

Giai đoạn 2



Triển khai
Terraform Code

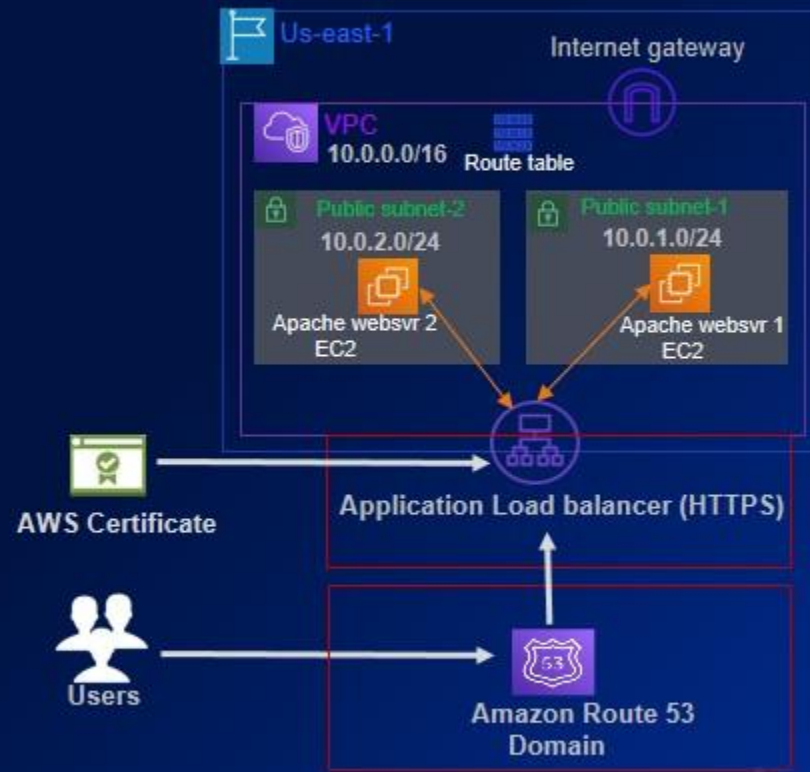
Giai đoạn 3



Sử dụng tên miền truy
cập vào webserver.
Hoàn thành bài Lab

Giai đoạn 4

Sử dụng Terraform tạo Route 53 Records (Alias) để định tuyến traffic truy cập vào ALB



Terraform Outputs và Terraform Graph

Mục đích của bài học:

Sử dụng các lệnh Terraform Output và Terraform Graph commands



Terraform Outputs

- Được sử dụng để hiển thị các giá trị tài nguyên nhất định trên CLI sau khi chạy lệnh terraform apply.
- Có thể được sử dụng để chuyển các giá trị từ module con tới module cha.
- Khi sử dụng remote state storage, các root module outputs có thể được các terraform configs khác sử dụng.



Terraform Graph

- Trình bày hình ảnh trực quan về cấu hình hoặc kế hoạch thực hiện.
- Cung cấp output trong định dạng chấm (DOT), có thể được hiển thị bằng GrapViz, hoặc các công cụ khác.

Các tham khảo hữu ích: Terraform Console

- ✓ Cung cấp một console tương tác để đánh giá các biểu thức trong Terraform.
- ✓ Tốt cho việc cho thử nghiệm và testing kiểm tra các phép nội suy.
- ✓ Nếu trạng thái triển khai hiện tại trống hoặc chưa được tạo, Terraform console có thể được sử dụng để kiểm tra cú pháp biểu thức và các hàm tích hợp sẵn(built-in functions) của Terraform.

Các tham khảo hữu ích: Terraform State

- ✓ Cho việc quản lý trạng thái (state) nâng cao của các tài nguyên(resources) Terraform.
- ✓ Lệnh "terraform state list" có thể được sử dụng để liệt kê các tài nguyên trong terraform state file.

Ví dụ sử dụng Terraform Outputs

```
Output "db_password" {  
  value      = aws_db_instance.db.password  
  description = "Database Password"  
  sensitive  = true  
}
```

Ansible Playbooks và kiểm tra cú pháp

Mục đích của bài học:

Ansible Playbooks và kiểm tra cú pháp

- ① Khai báo các cấu hình và tự động hóa các tác vụ, như cấu hình và cài đặt hệ thống thường được thực hiện một cách thủ công.
- ② Các playbooks chỉ được viết bằng ngôn ngữ YAML
- ③ Tham số dòng lệnh để kiểm tra cú pháp của một playbook trong Ansible là: `--syntax-check`

Ví dụ, để kiểm tra cú pháp một file playbook có tên là "sample.yaml". Sử dụng lệnh sau:

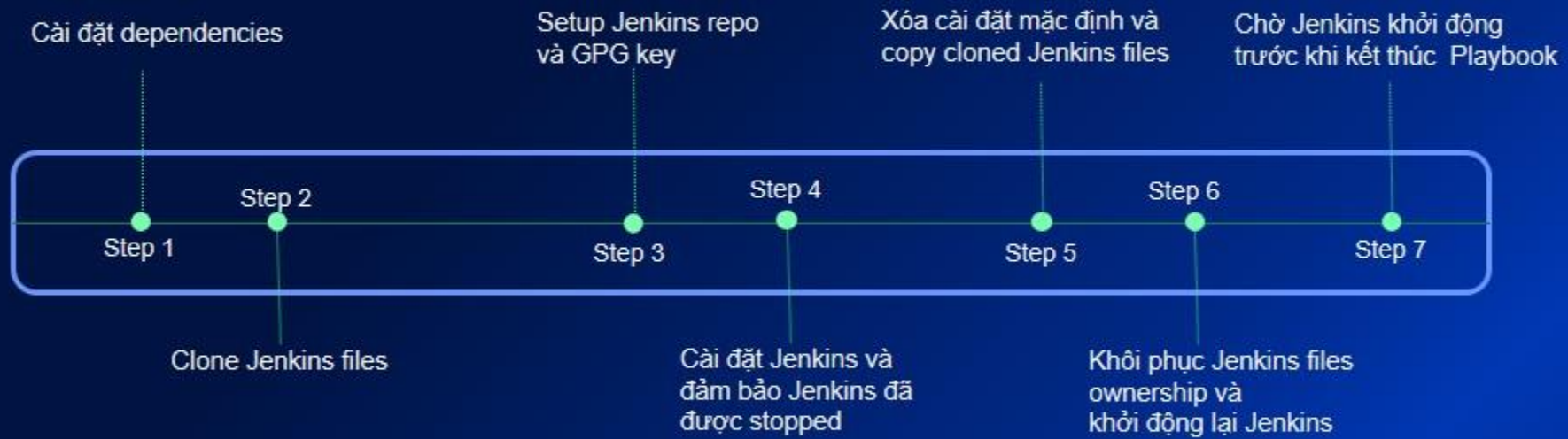
```
ansible-playbook --syntax-check sample.yaml
```

Sample Ansible Playbook cho Linux (RHEL 8/CentOS 8)

```
---  
- hosts: cloud-nodes  
  tasks:  
    - name: ensure apache is at the latest version  
      yum:  
        name: httpd  
        state: latest
```

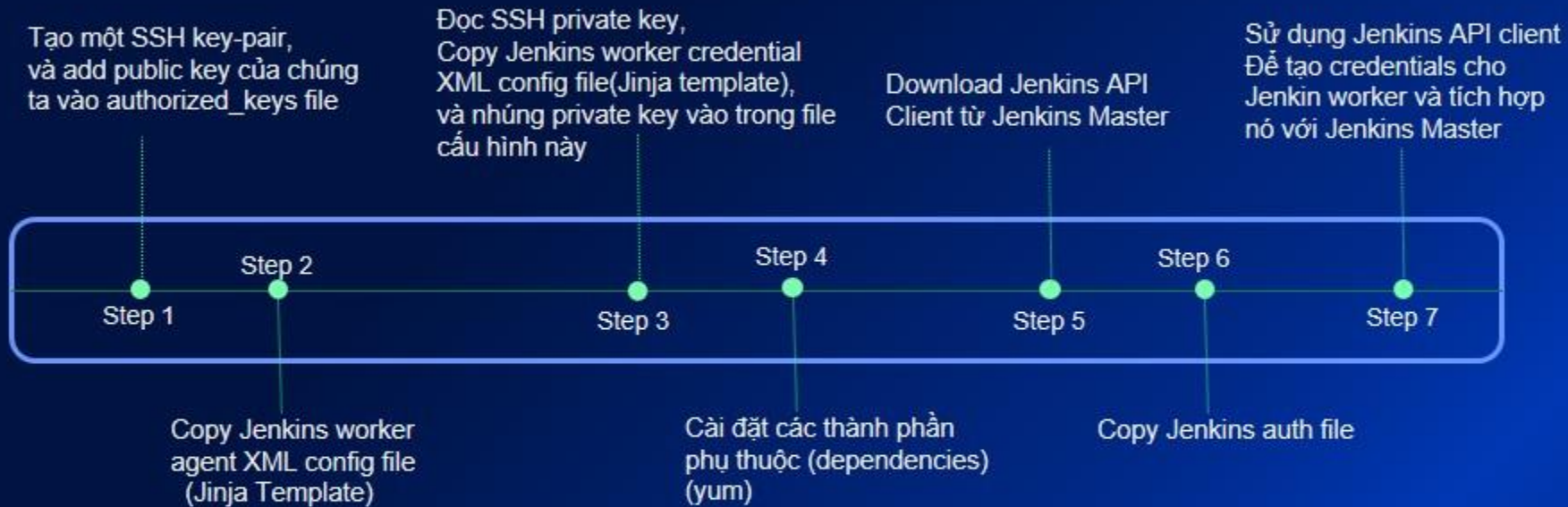

Xây dựng Playbooks cho Setup Jenkins Master

Trình tự thực thi của Ansible Playbook – Jenkins Master



Xây dựng Playbooks cho Setup Jenkins Workers

Trình tự thực thi của Ansible Playbook – Jenkins Workers



Xây dựng Jinja2 Templates cho Ansible

Mục đích của bài học:
**Xây dựng Jinja2 Templates cho Jenkins Config
thông qua Ansible**

- ✓ Jinja là một ngôn ngữ tạo template hiện đại, thân thiện với người dùng
- ✓ Ansible sử dụng Jinja2 để tạo template cho các biểu thức động và truy cập các biến
- ✓ Tất cả quá trình tạo template được diễn ra trên Ansible controller node trước khi task (tác vụ or công việc) được gửi đến các máy đích và được thực thi.

Việc này được thực hiện để giảm các yêu cầu về tài nguyên và ứng dụng phần mềm, đối với các máy đích.

Jinja2 chỉ cần được đòi hỏi có trong Ansible controller node

Ansible và Jinja trong tạo Template

Sample template cho DNS file trong Linux, resolv.j2

resolv.j2 – Jinja Template

```
nameserver  {{ ipv4 }}
```

✓ Đoạn code Ansible sử dụng Jinja Template

```
---
- hosts: target-nodes
  become: yes
  tasks:
    - name: Copy over the DNS resolution file to target node
      vars:
        ipv4: "{{ ansible_default_ipv4.address }}"
      template:
        src: resolv.j2
        dest: /etc/resolv.conf
        owner: ec2-user
        mode: '0644'
```

**Xác nhận IaC Code của chúng ta
và chạy lệnh Terraform Apply**

Checklist công việc

- ✓ Đảm bảo bạn cần có tất cả các dependencies: ansible, terraform, aws cli, boto3, đã tạo SSH key-pair cho user của bạn trong Linux Terraform Controller.
- ✓ Đảm bảo Terraform version là từ 12.0.0 trở lên.
- ✓ Đảm bảo bạn đã có AWS credentials trong thư mục ~/ .aws hoặc, nếu bạn sử dụng một EC2 instance để làm Linux Terraform Controller, server này cần có một role với các quyền cần thiết trong tài khoản AWS.
- ✓ **Chạy lệnh:** terraform fmt, terraform validate, terraform plan, và cuối cùng là: terraform apply.
- ✓ Đừng quá lo lắng nếu việc triển khai lỗi (fails), chúng ta cứ bình tĩnh và tìm cách sửa lỗi troubleshoot.

**Troubleshooting
Setup của chúng ta
(TF_LOG, ANSIBLE_DEBUG)**

Debugging Terraform – TF_LOG và TF_LOG_PATH

- ✓ **TF_LOG** là một biến môi trường dùng để bật (enable) ghi logs chi tiết trong terraform. Mặc định, nó sẽ gửi logs tới stderr (đầu ra lỗi tiêu chuẩn)
- ✓ Có thể sử dụng các mức ghi logs như: **TRACE**, **INFO**, **WARN**, và **ERROR**
- ✓ **TRACE** là mức ghi logs chi tiết nhất và cũng là mức đáng tin cậy nhất
- ✓ Để lưu log vào một file, sử dụng biến môi trường **TF_LOG_PATH**
- ✓ Setup các biến môi trường logs cho Terraform trong Linux:

```
export TF_LOG=TRACE  
export TF_LOG_PATH=./terraform.log
```

Debugging Ansible – **ANSIBLE_DEBUG** và tham số (-v)

- ✓ Biến môi trường **ANSIBLE_DEBUG** kích hoạt ghi logs rất chi tiết và có thể cản trở xử lý đa luồng.
- ✓ **ANSIBLE_DEBUG** cũng có thể hiển thị thông tin nhạy cảm và không được khuyến nghị sử dụng trong môi trường chạy thật (production)
- ✓ Biến môi trường **ANSIBLE_VERBOSITY** hoặc tùy chọn "-v" của các lệnh "ansible" và "ansible-playbook" trên giao diện dòng lệnh (CLI) cũng có thể xuất các debug logs ra stdout.
- ✓ Mức độ chi tiết (verbosity) khi sử dụng tùy chọn "-v" tỷ lệ thuận với số lượng chữ "v" trong tùy chọn (ví dụ: --vv thì chi tiết hơn -v).
- ✓ **ANSIBLE_VERBOSITY** chấp nhận một số làm đầu vào và đặt mức độ chi tiết mặc định tương ứng với số lượng chữ 'v' được truyền qua dòng lệnh cho các lệnh ansible hoặc ansible-playbook . VD:

Nếu bạn thiết lập **ANSIBLE_VERBOSITY=2**, điều này tương ứng với việc bạn thêm -vv vào lệnh ansible hoặc ansible-playbook.