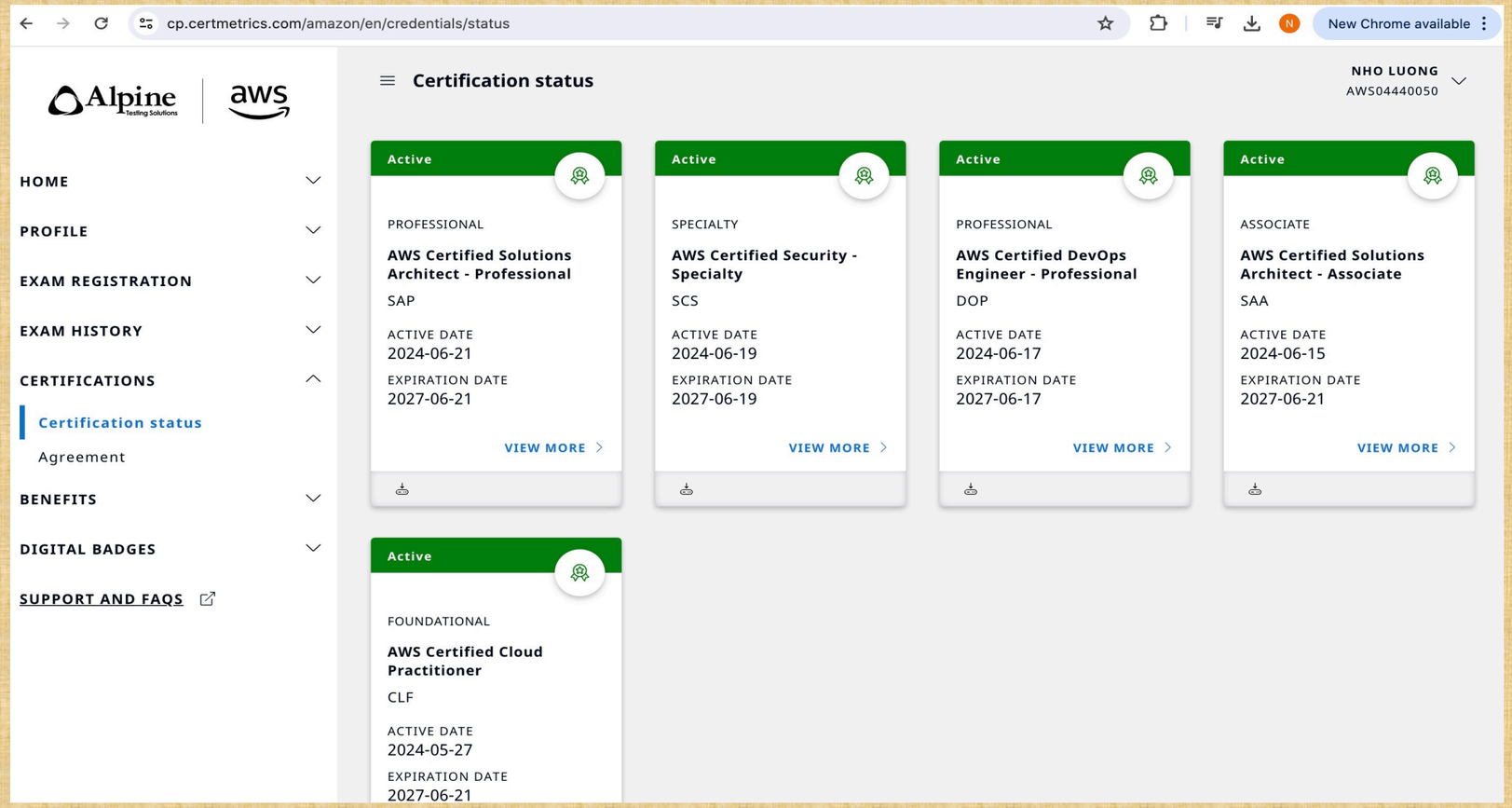
# Terraform on AWS Cloud EC2

Author: Nho Luong

Skill: DevOps Engineer Lead


# CLOUDSTACK

# DevOps & SRE Roadmap

**AWS EKS**
Kubernetes

**Azure AKS**
Kubernetes

**AWS ECS**
Docker on AWS

**AWS**
CloudFormation

**Google GKE**
Kubernetes

**AWS**
Lambda & Serverless

**HashiCorp Certified**
Terraform Associate

**HashiCorp Certified**
Vault & Consul

**CKA** Certified Kubernetes
Administrator

**DevOps on AWS**
EC2, ECS, EKS, Lambda, VPC

**DevOps on Azure**
VMs, ACI, AKS, Functions, App Services

**SRE with Terraform on AWS**
EC2, ECS, EKS, Lambda, VPC

**SRE with Terraform on Azure**
VMs, ACI, AKS, Functions, App Services

**CKAD** Certified Kubernetes
Application Developer

**CKS** Certified Kubernetes
Security

Author: Nho Luong
Skill: DevOps Engineer Lead

Terraform Fundamentals (Commands, Language, Settings, Providers, Resources, Variables, Datasources, Meta-Arguments)

AWS VPC 3-Tier Architecture

AWS EC2 Instances & Security Groups

AWS Classic Load Balancer

AWS Application Load Balancer ALB

ALB Context-Path based Routing

ALB Host-Header based Routing

ALB Custom HTTP Header & Query String Redirects

AWS DNS to DB Implementation

Terraform On AWS

Real-World Approach

Step by Step Documentation On GitHub

Incremental way to Build Complex Infra

AWS Autoscaling with Launch Configuration

AWS Autoscaling with Launch Templates

AWS Network Load Balancer with TCP & TLS

AWS CloudWatch Alarms for ALB, ASG and CIS

Terraform Local Modules – Leverage Public Registry

Terraform Local Modules – Build from scratch

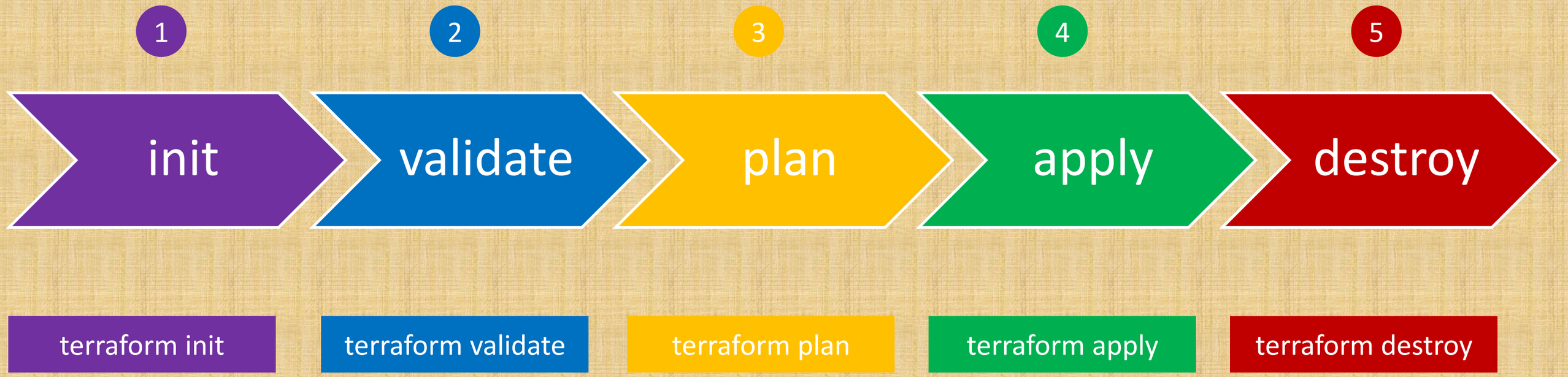
Terraform Remote State Storage

Terraform Remote State Datasource

IaC DevOps with AWS CodePipeline for Terraform Project

Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Workflow




Author: Nho Luong
Skill: DevOps Engineer Lead

**Terraform** language uses a limited number of **top-level block** types, which are **blocks** that can appear outside of any other **block** in a TF configuration file.

# Terraform Top-Level Blocks

Most of **Terraform's features** are implemented as **top-level** blocks.

## Fundamental Blocks

- Terraform Block
- Providers Block
- Resources Block

## Variable Blocks

- Input Variables Block
- Output Values Block
- Local Values Block

## Calling / Referencing Blocks

- Data Sources Block
- Modules Block

Author: Nho Luong
Skill: DevOps Engineer Lead

## Section-1

- Meta-Argument: count
- Variables: Lists & Maps
- For Loop with Lists
- For Loop with Maps
- For Loops with Advanced Maps
- Legacy Splat Operator (.*.)
- Latest Splat Operator [*]

## Section-2

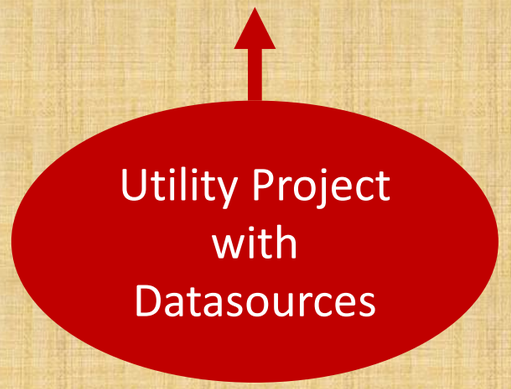
- Meta-Argument: for_each
- Function: toset
- Function: tomap
- Datasource: aws_availability_zones

## Section-4

- Fix issues in Section-2 with section-3
- Final Output

## Section-3

- Datasource: aws_ec2_instance_type_offerings
- Datasource: aws_availability_zones
- For Loop with Maps
- For Loop with if
- Function: keys




Author: Nho Luong
Skill: DevOps Engineer Lead

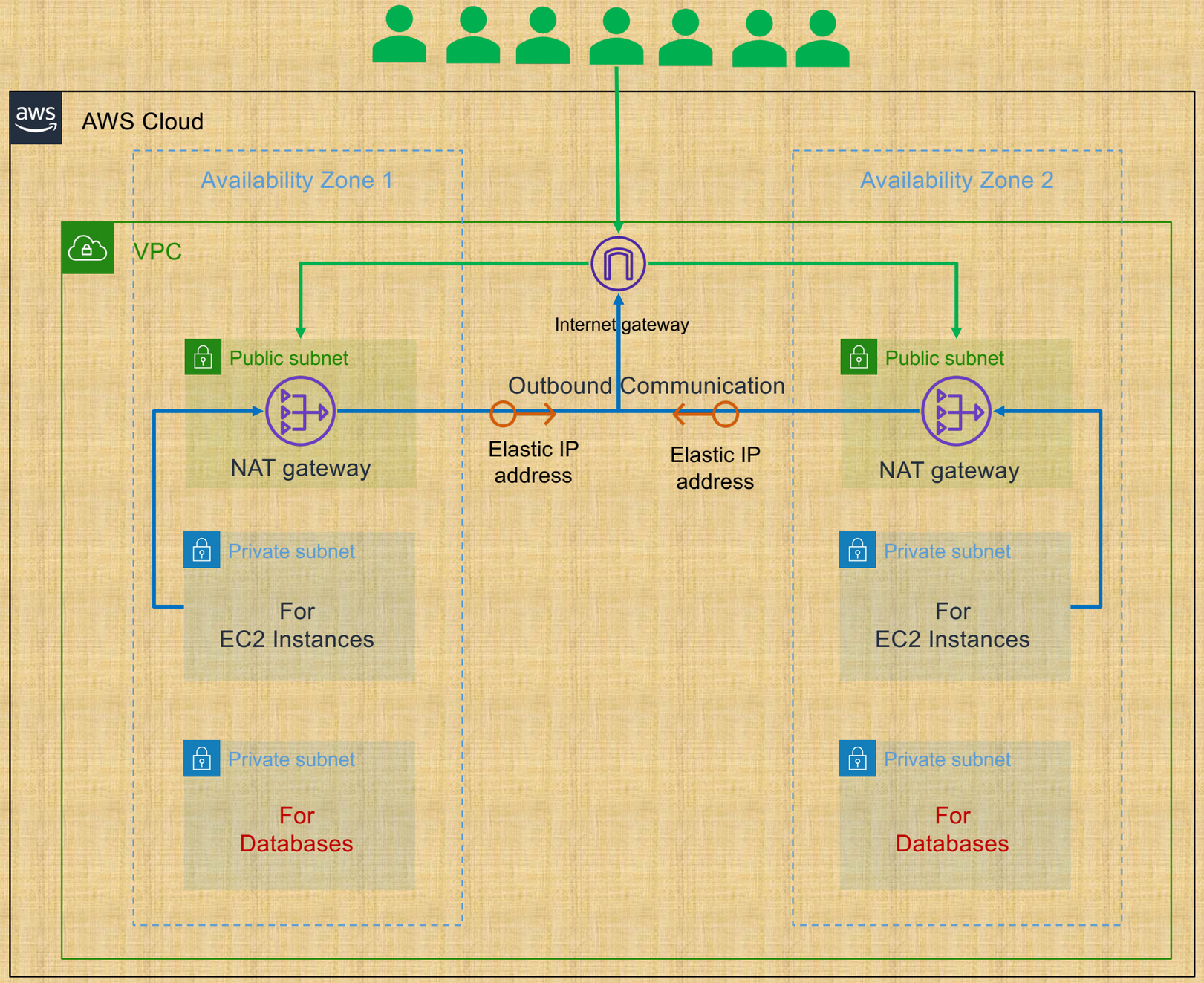
# Terraform Fundamentals

| | |
|---|---|
| **Terraform Basics** | 12 lectures • 1hr 6min |
| **Terraform Settings, Providers and Resources** | 8 lectures • 1hr 10min |
| **Terraform Input Variables, Datasources and Output Values** | 8 lectures • 53min |
| **Terraform Loops, MetaArguments, Splat Operator and Functions** | 9 lectures • 1hr 24min |

Author: Nho Luong
Skill: DevOps Engineer Lead

# AWS VPC 3-Tier Architecture

- Build manually using AWS Mgmt Console
- Build same using Terraform
- Terraform Modules
- Terraform Local Values
- Terraform Variables – terraform.tfvars
- Terraform Variables – vpc.auto.tfvars
- Terraform Version Constraints
- Terraform Code Organizing – Production Grade style

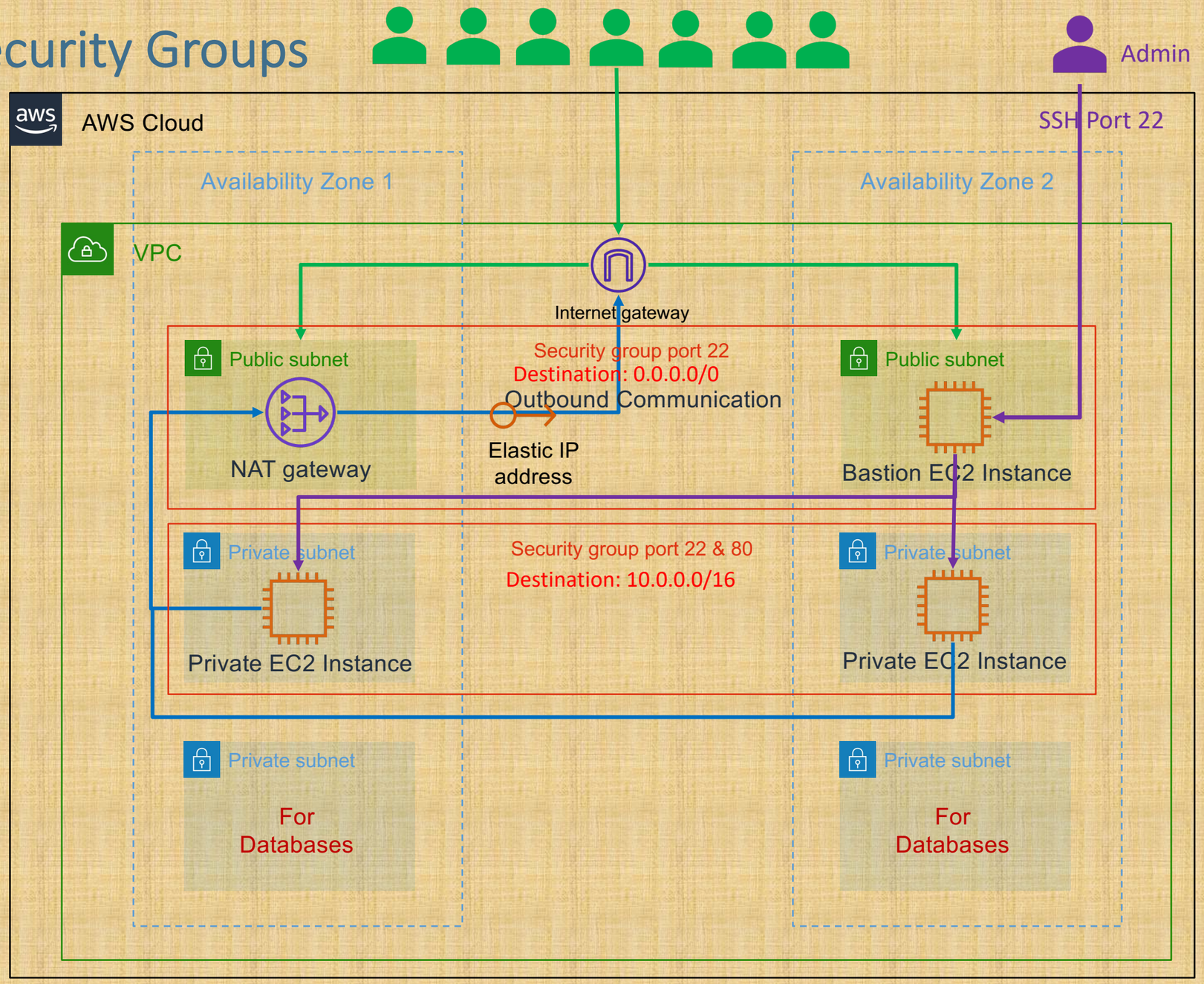



Author: Nho Luong
Skill: DevOps Engineer Lead

# AWS VPC + EC2 Instance + Security Groups

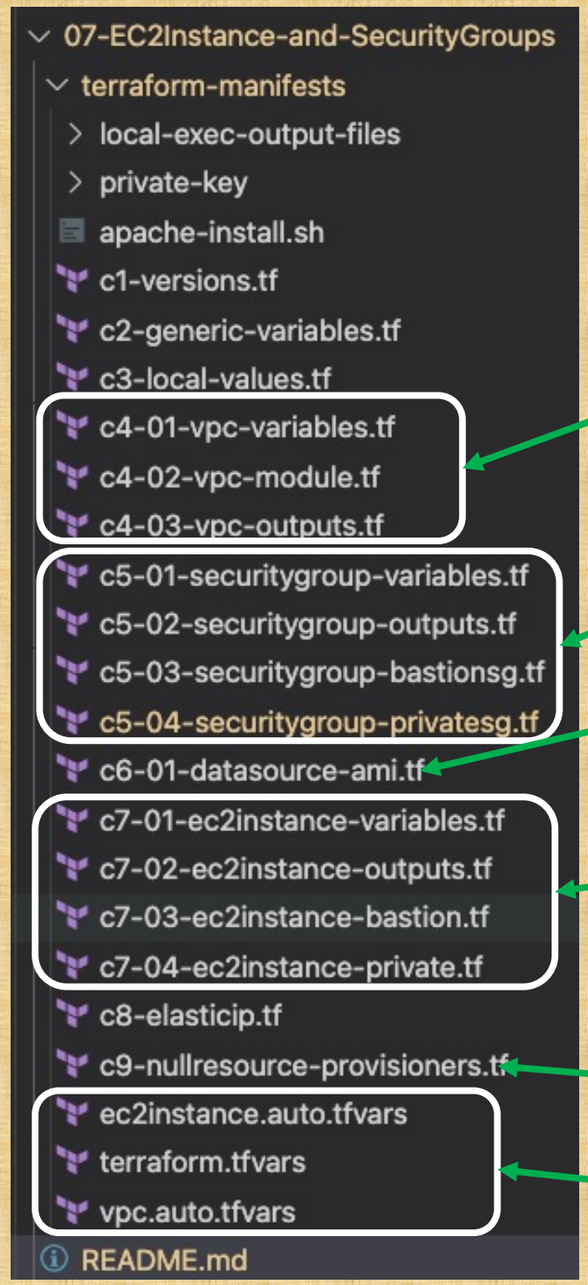
## Terraform & AWS Concepts

- Terraform Module: VPC
- Terraform Module: Security Group
- Terraform Module: AWS EC2 Instance
- Meta-Argument: depends_on
- Terraform null_resource
- Terraform File Provisioner
- Terraform Remote-exec Provisioner
- Terraform Local-exec Provisioner




Author: Nho Luong
Skill: DevOps Engineer Lead

# What are we going to learn ?




Create AWS VPC using Terraform Modules

Create AWS Security Groups using Terraform Modules

Create AWS AMI Datasource to get latest AMI ID dynamically

Create AWS EC2 Instance using Terraform Modules

Null Resource

- File Provisioner
- Remote-exec Provisioner
- Local-exec Provisioner

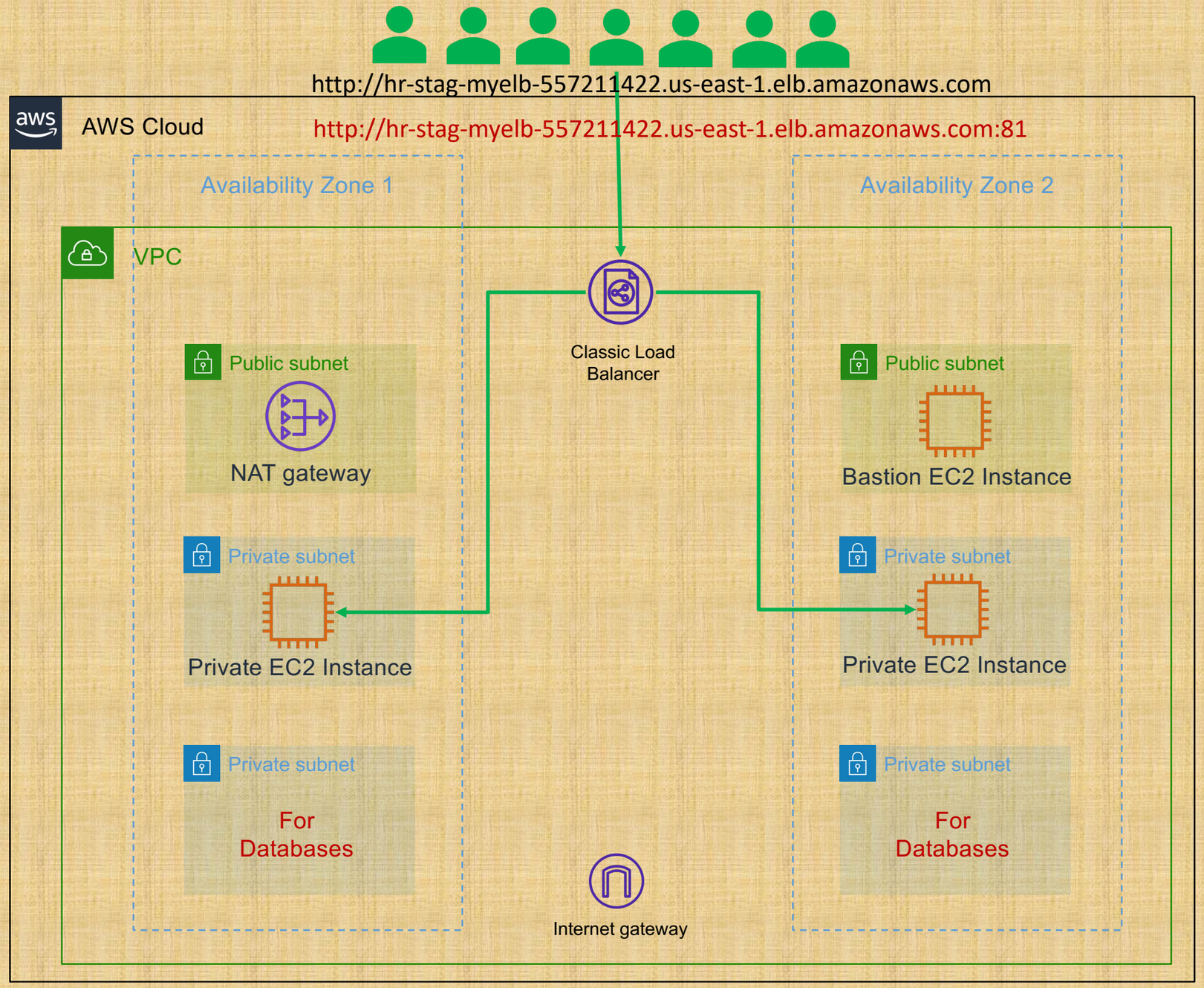
Variables

Author: Nho Luong
Skill: DevOps Engineer Lead

# AWS VPC + EC2 Instance + Security Groups + AWS ELB Classic Load Balancer

**Terraform & AWS Concepts**

- Terraform Module: ELB Classic LB
- Terraform Module: Security Group
- Add Custom SG Rule for Inbound Port 81



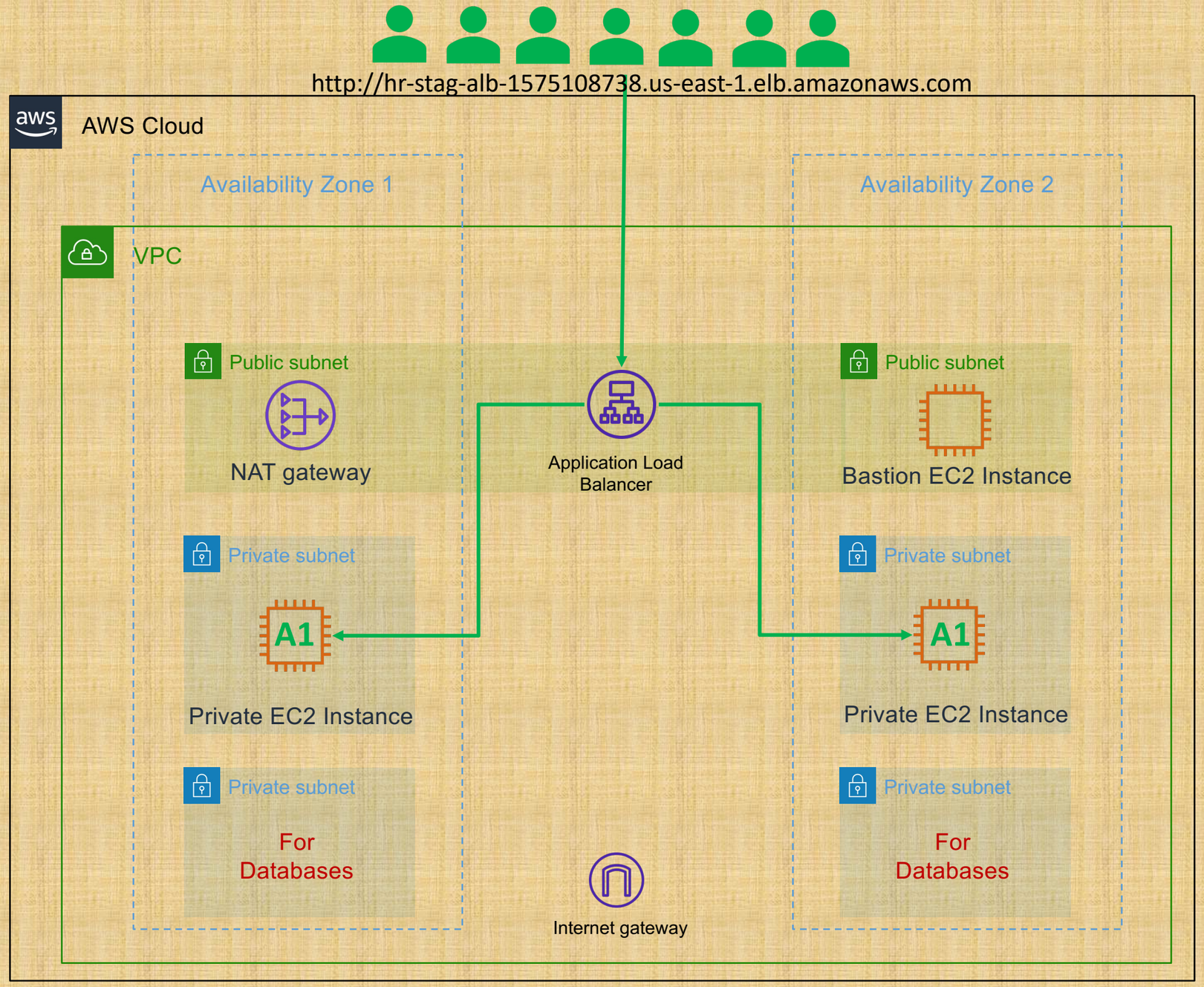


Author: Nho Luong
Skill: DevOps Engineer Lead

# AWS VPC + EC2 Instance + Security Groups + AWS ALB Application Load Balancer

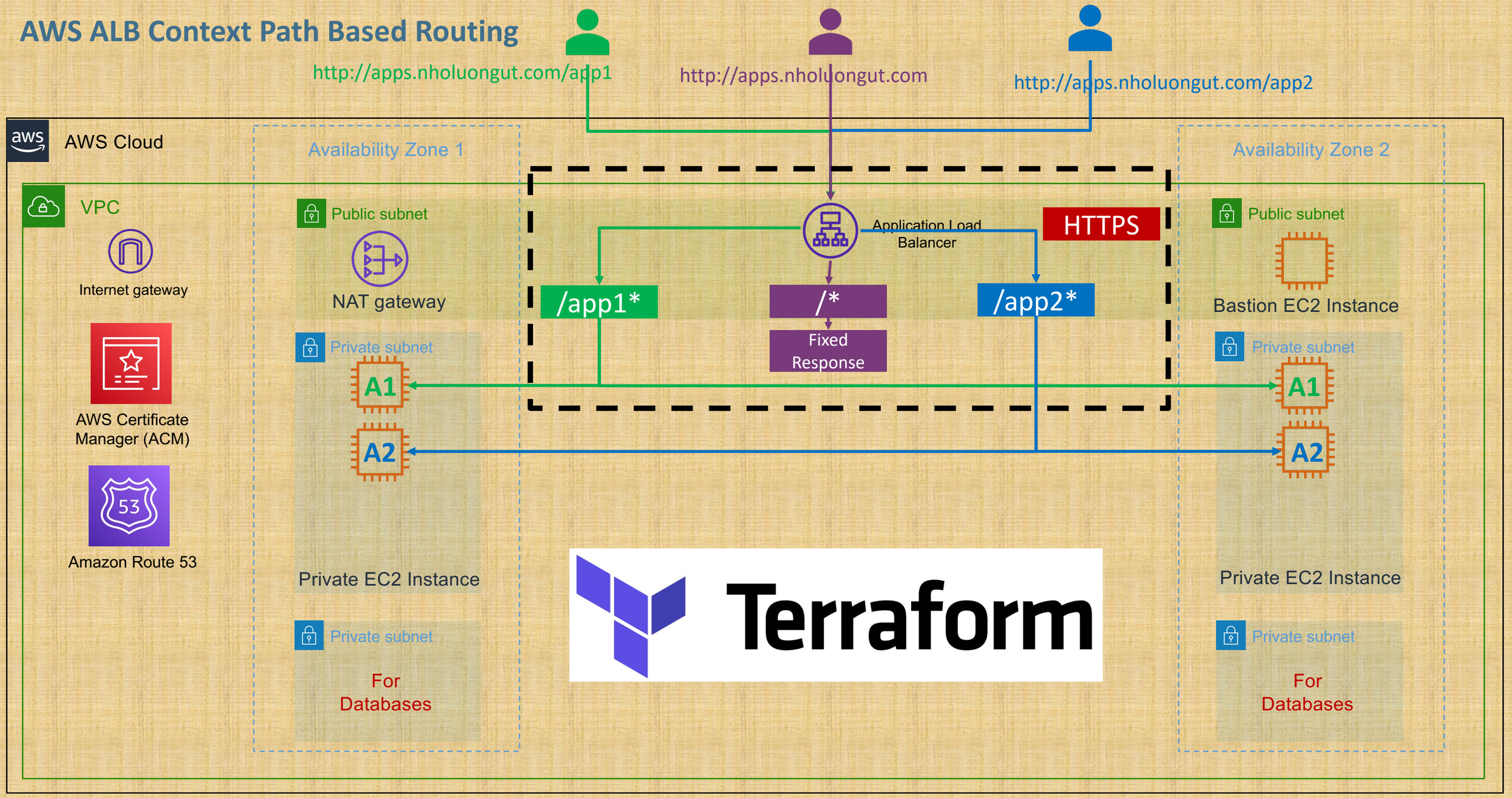
Terraform & AWS Concepts

Terraform Module: AWS ALB




Author: Nho Luong
Skill: DevOps Engineer Lead

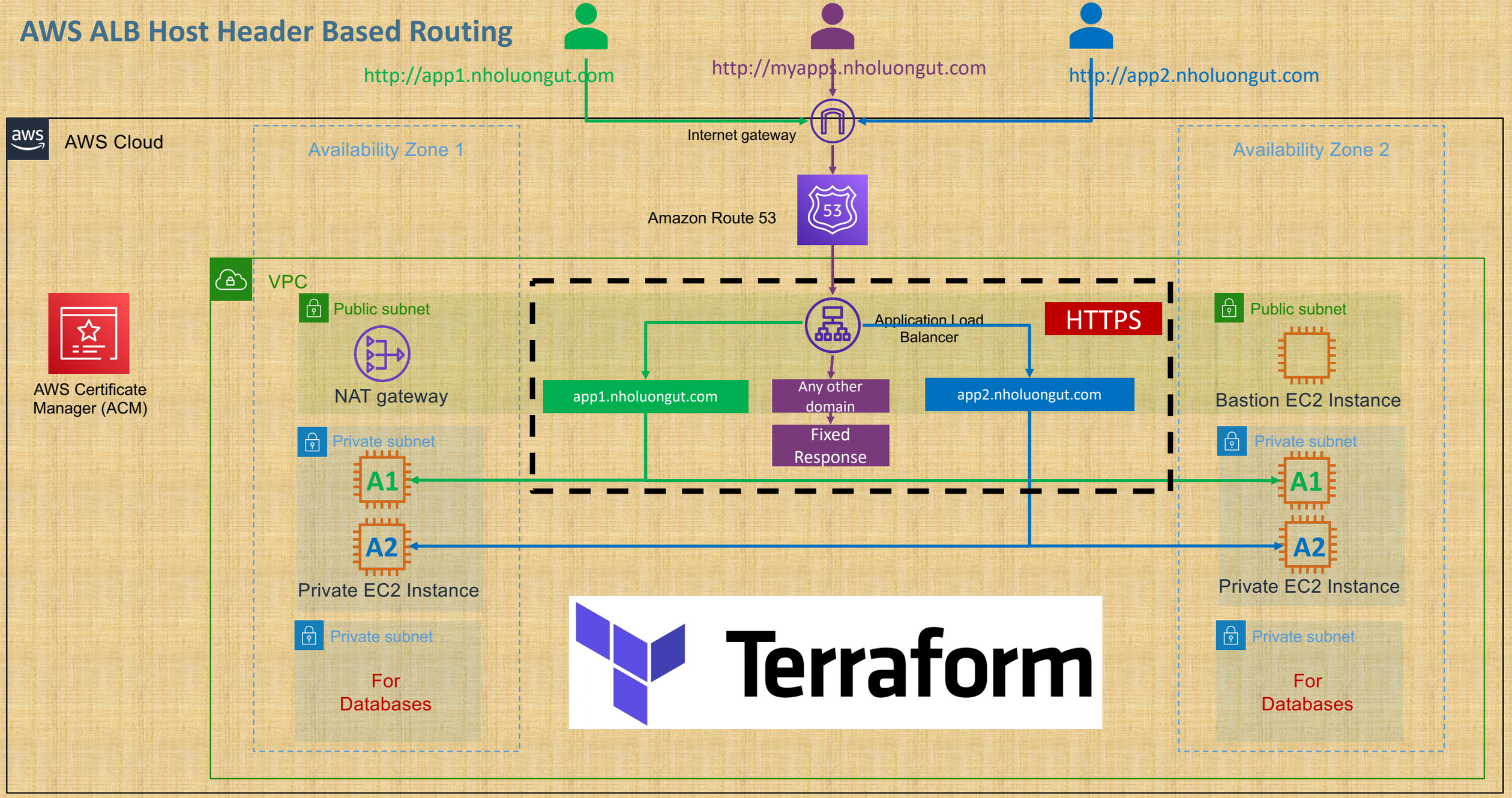
# AWS ALB Context Path Based Routing





Author: Nho Luong
Skill: DevOps Engineer Lead

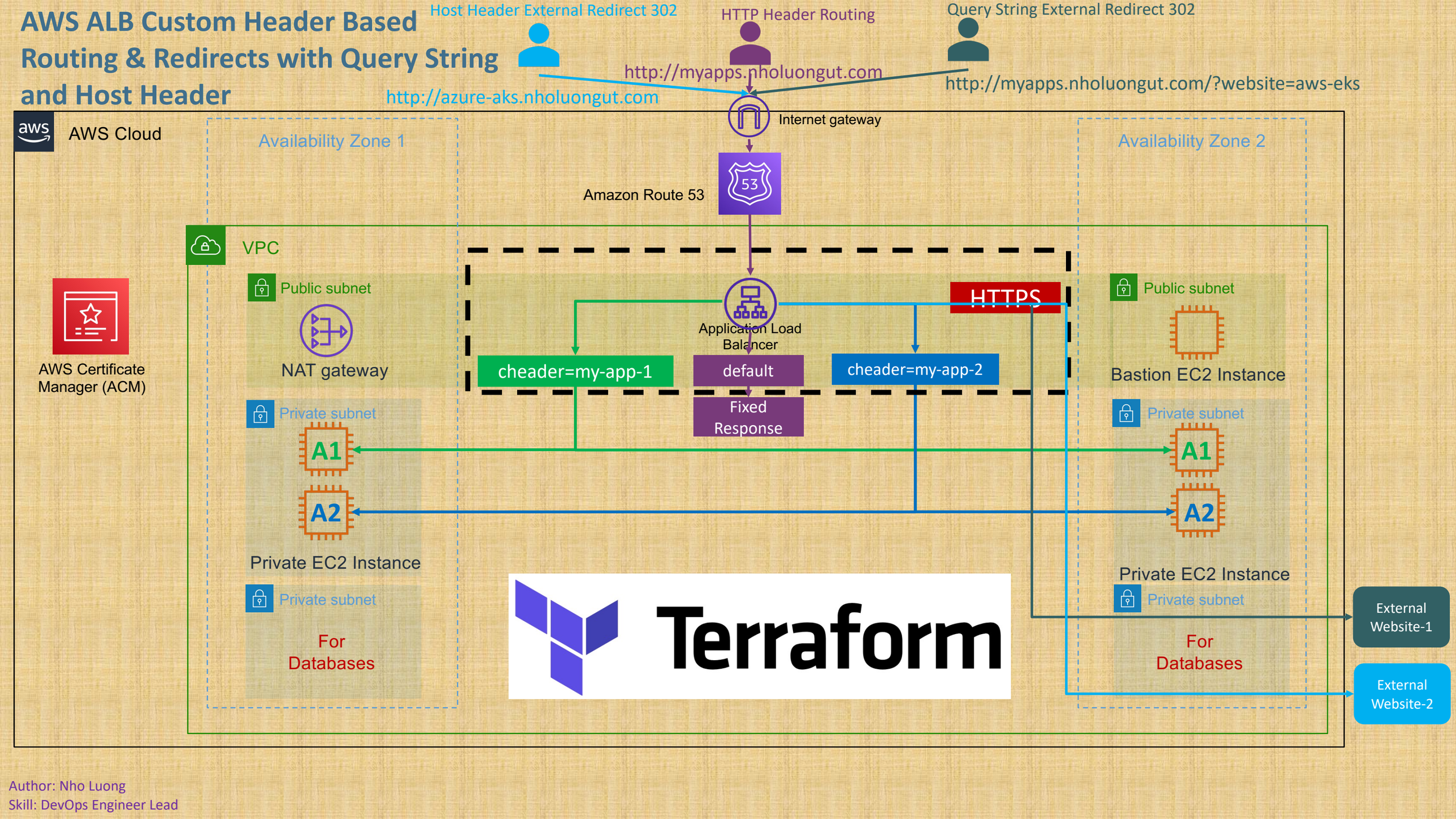
# AWS ALB Host Header Based Routing





Author: Nho Luong
Skill: DevOps Engineer Lead

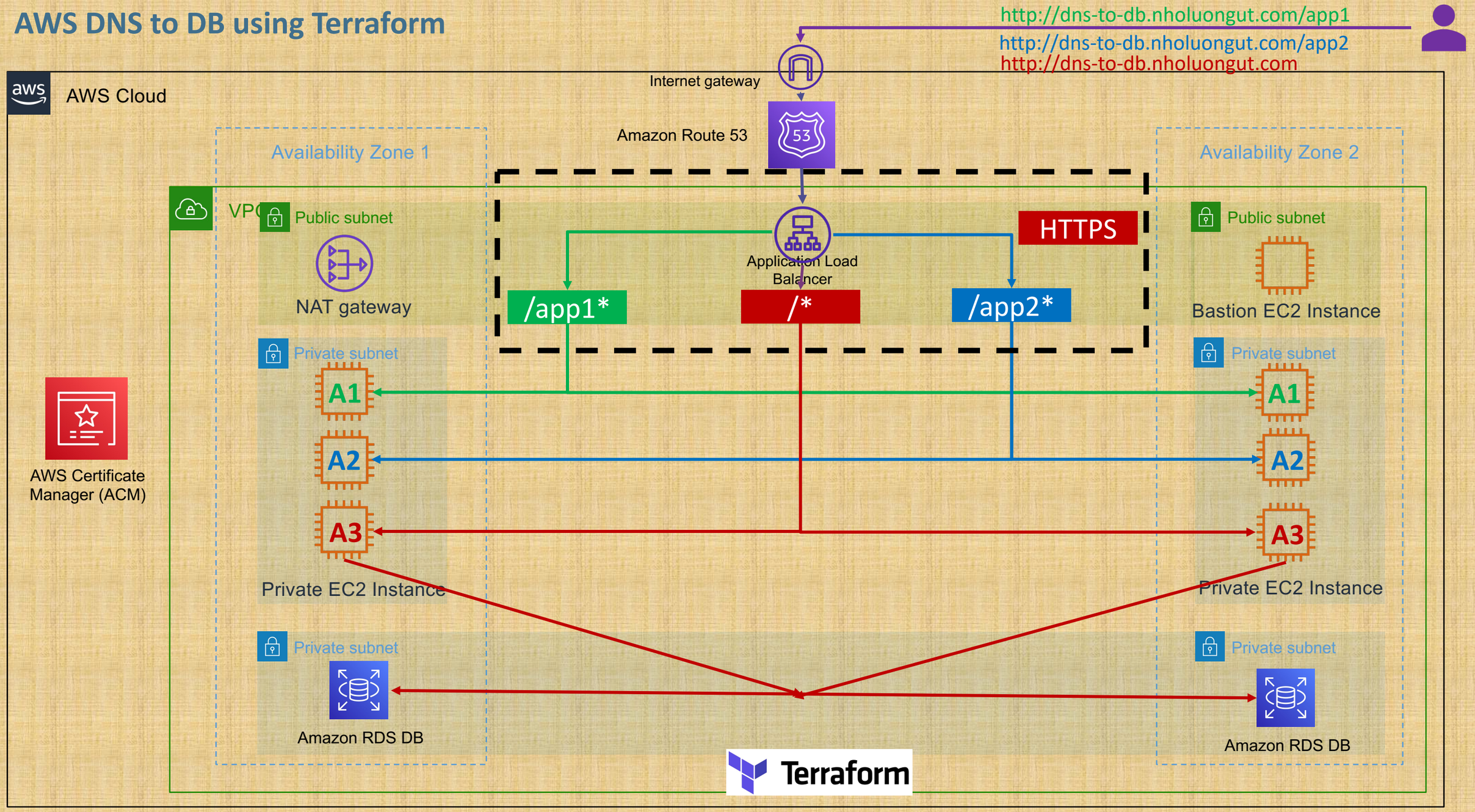
# AWS ALB Custom Header Based Routing & Redirects with Query String and Host Header





Author: Nho Luong
Skill: DevOps Engineer Lead

# AWS DNS to DB using Terraform





Author: Nho Luong
Skill: DevOps Engineer Lead

# Upgrade Terraform Modules

| Module Type | Previous Version | New Version | Impact Analysis |
|---|---|---|---|
| VPC Terraform Module | v2.78.0 | v3.0.0 | No Impact |
| Security Group Terraform Module | v3.18.0 | v4.0.0 | High Impact |
| Application Load Balancer Terraform Module | v5.16.0 | v6.0.0 | High Impact |
| ACM Certificate Manager Terraform Module | v2.14.0 | v3.0.0 | High Impact |

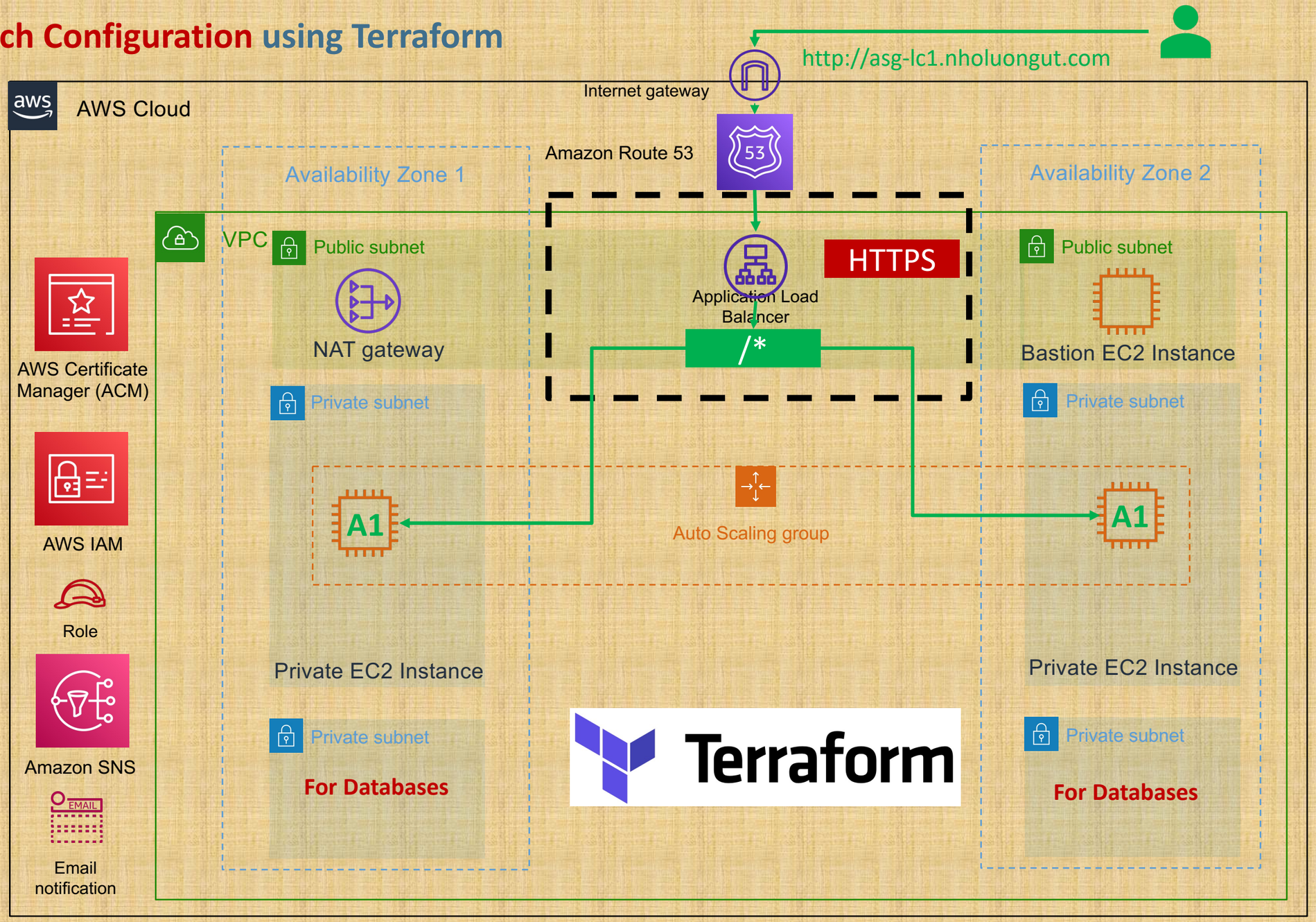
There is a major change in all the modules (except vpc) related to output names of these modules which impacts all these module references wherever you use.

**Actual Change:** `this_` for all outputs was removed for these modules.

Author: Nho Luong
Skill: DevOps Engineer Lead

# AWS Autoscaling with Launch Configuration using Terraform

- Terraform ASG Module
- ASG with Launch Configuration
- ASG Instance Refresh
- ASG Lifecycle Hooks
- ASG TTSP
- ASG Scheduled Actions
- ASG Notifications
- ASG Autoscaling Tests

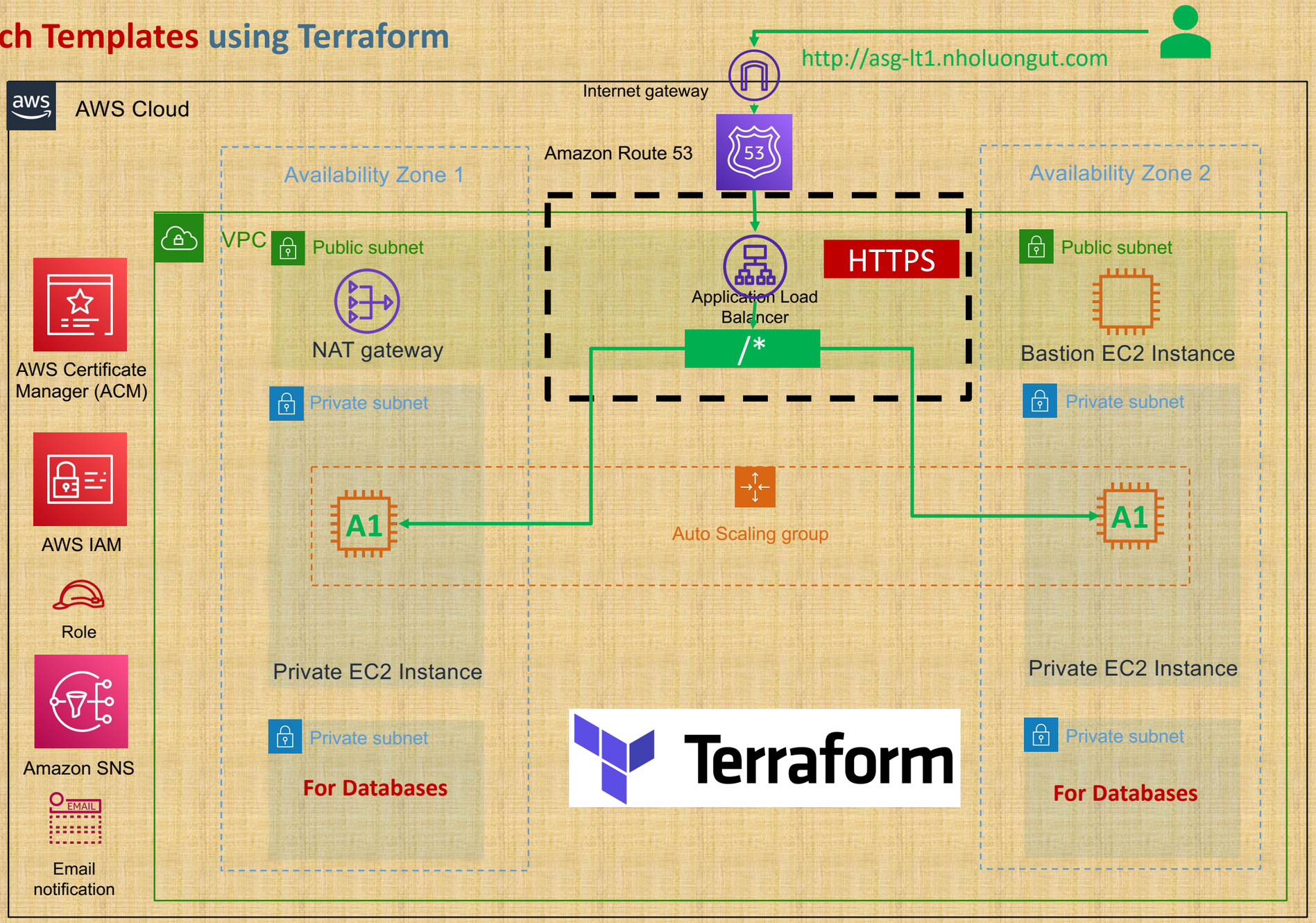




Author: Nho Luong
Skill: DevOps Engineer Lead

# AWS Autoscaling with Launch Templates using Terraform

- Terraform ASG Resource
- ASG with Launch Templates
- ASG Instance Refresh
- ASG Lifecycle Hooks
- ASG TTSP
- ASG Scheduled Actions
- ASG Notifications
- ASG Autoscaling Tests

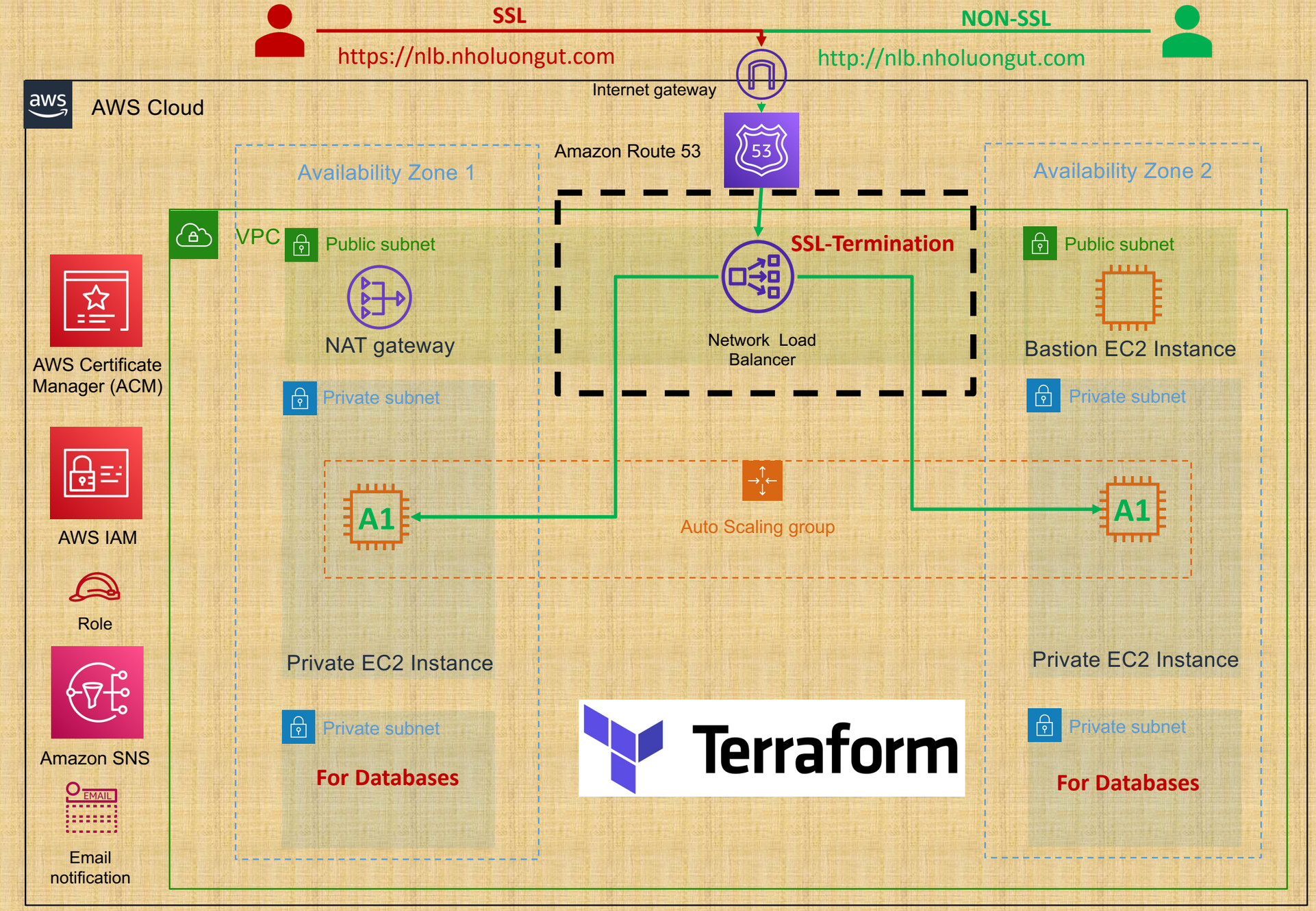




Author: Nho Luong
Skill: DevOps Engineer Lead

# AWS Network Load Balancer with TCP & TLS Listeners using Terraform

- Terraform NLB Module
- Terraform NLB TCP Listeners
- Terraform NLB TLS Listeners
- Terraform NLB Target Groups
- Terraform NLB with EC2 Autoscaling Groups





Author: Nho Luong
Skill: DevOps Engineer Lead

# AWS CloudWatch using Terraform

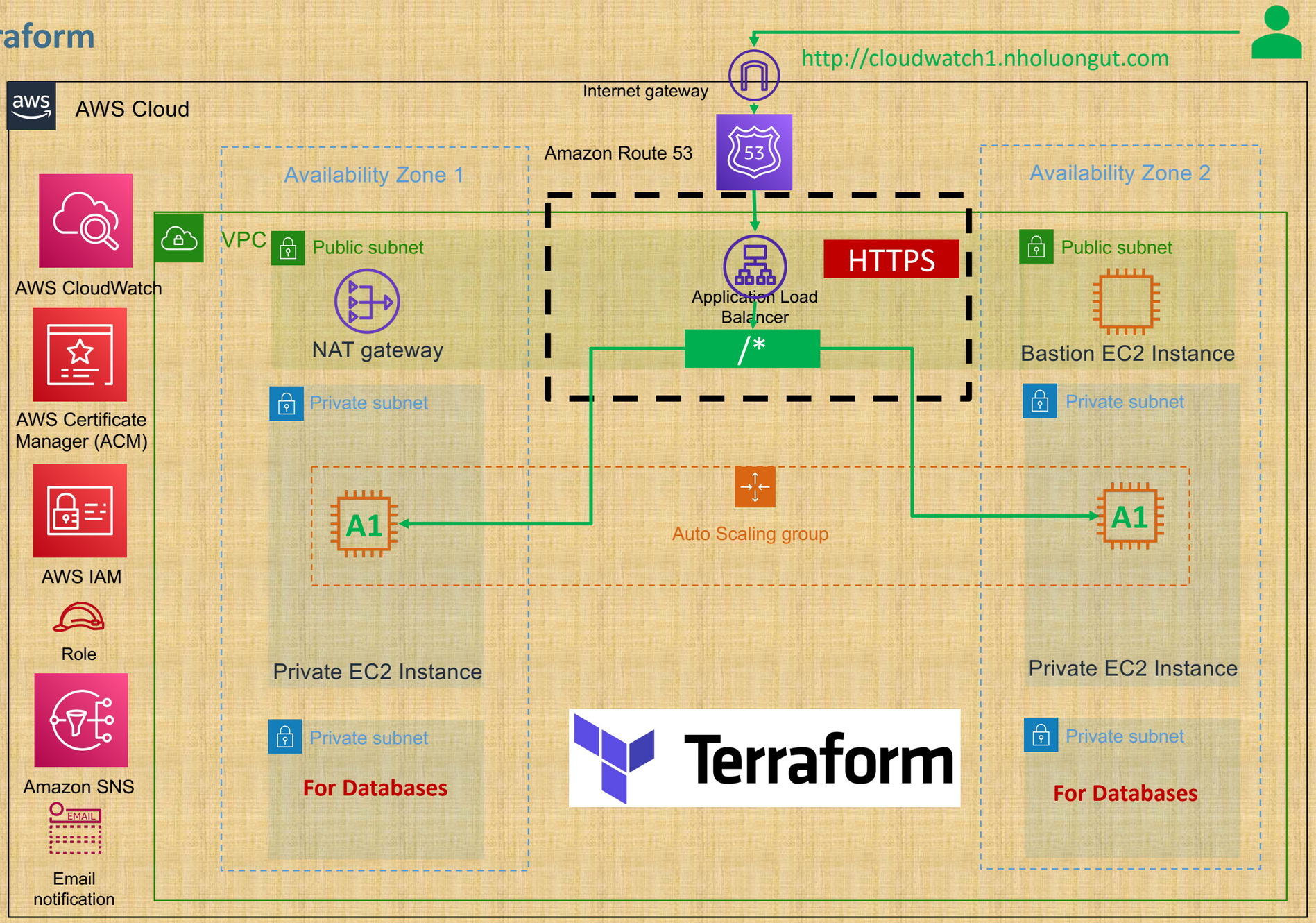
AWS CloudWatch

ASG Alarms

ALB Alarms

CIS Alarms

CloudWatch Synthetics





Author: Nho Luong
Skill: DevOps Engineer Lead

# Build Terraform Modules Locally

## Build from Scratch

If already not available, then use this approach

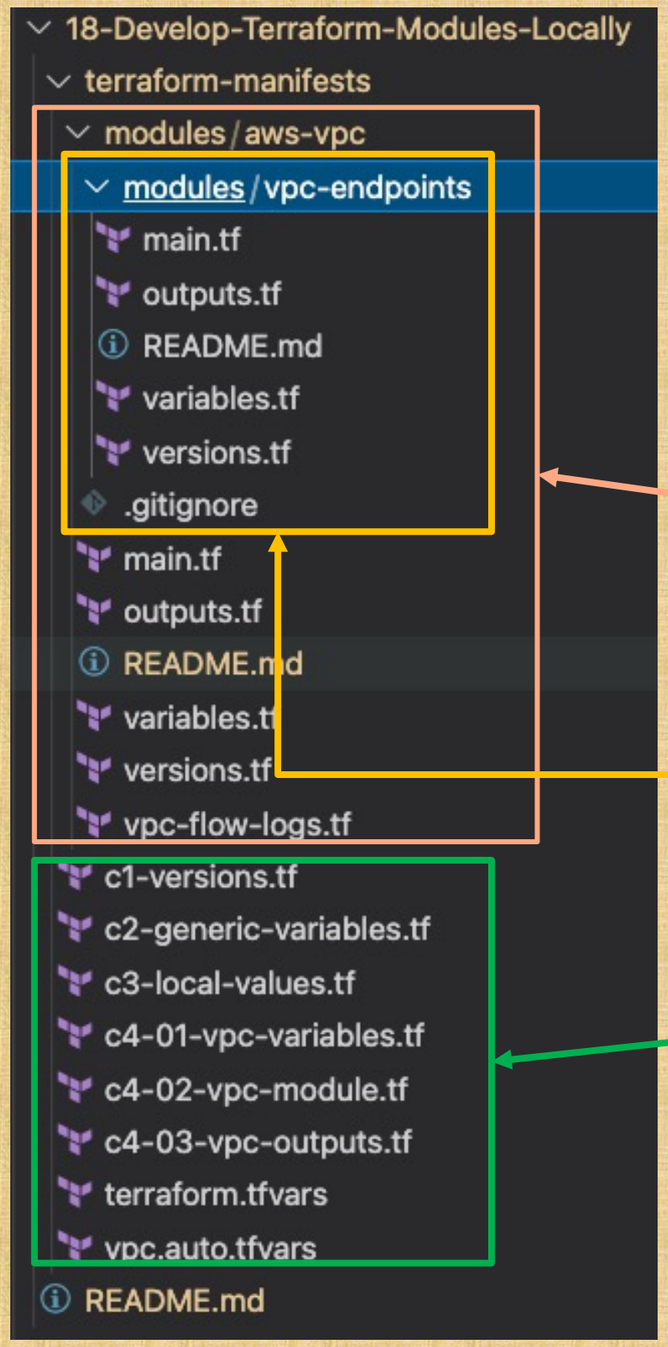
Review existing modules in Terraform Public Registry (Minimum 3 modules to get complete idea)

## Leverage existing Terraform Modules and change the code as per your need

If already available, use the code and modify as per your need

Less effort and you can start using them in your private infrastructure spaces without public internet access

Author: Nho Luong
Skill: DevOps Engineer Lead




# What are we going to learn ?

AWS VPC Module

AWS VPC Modules - VPC Endpoints Sub Module

Root Module calling the VPC Local Module

Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform State

Terraform
Local
State
Storage

Terraform
Remote
State
Storage

Author: Nho Luong
Skill: DevOps Engineer Lead

# What is Terraform Backend ?

Backends are responsible for storing state and providing an API for state locking.

Terraform
State Storage

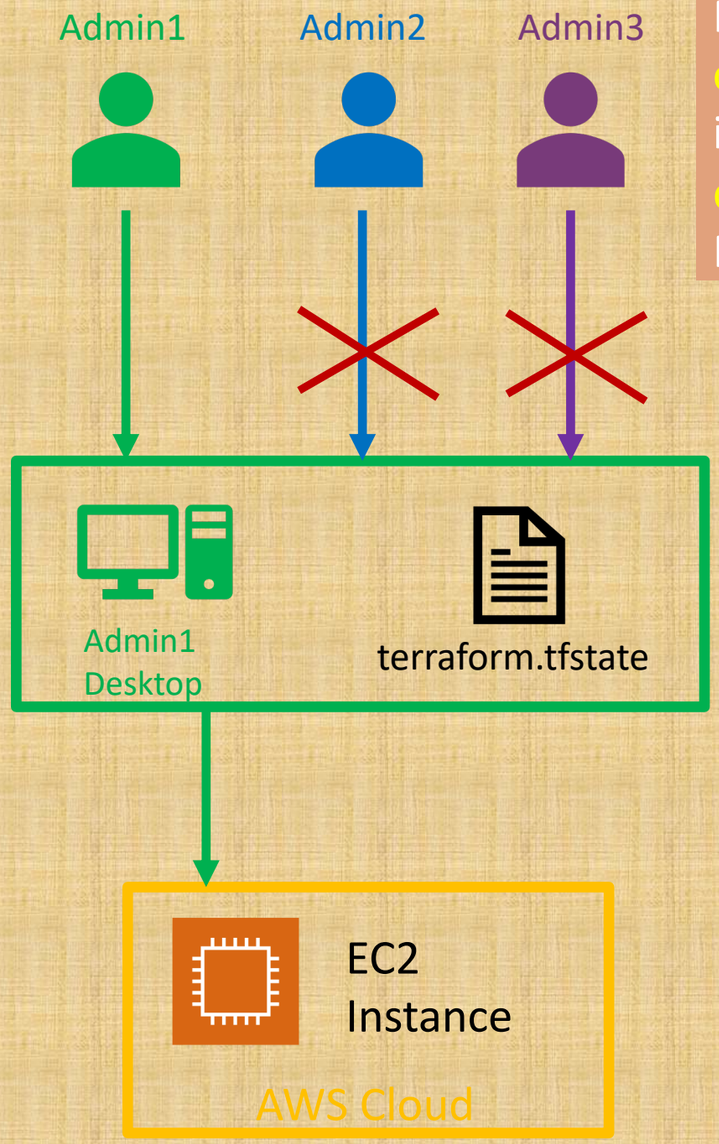
AWS S3 Bucket

Terraform
State Locking

AWS DynamoDB

Author: Nho Luong
Skill: DevOps Engineer Lead

# Local State File




Multiple Team members cannot update the infrastructure as they don't have access to State File

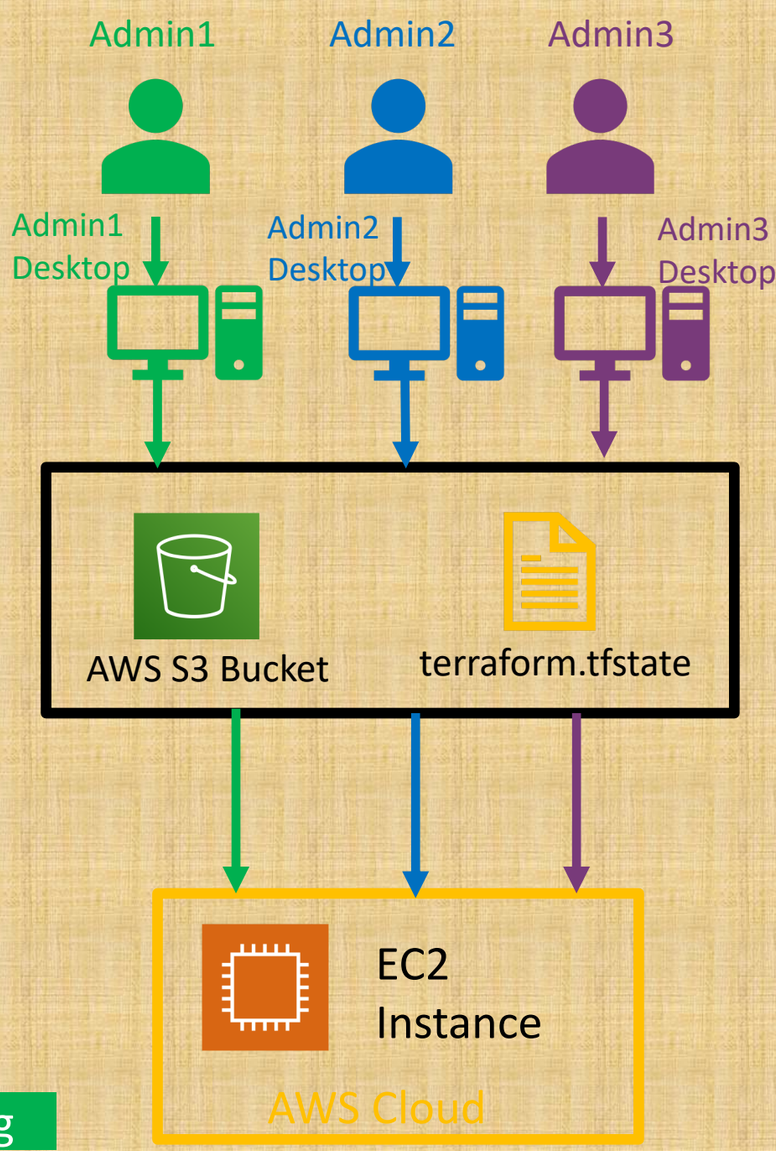
This means we need store the state file in a shared location.

# Remote State File

Using Terraform Backend concept we can use AWS S3 as the shared storage for State Files

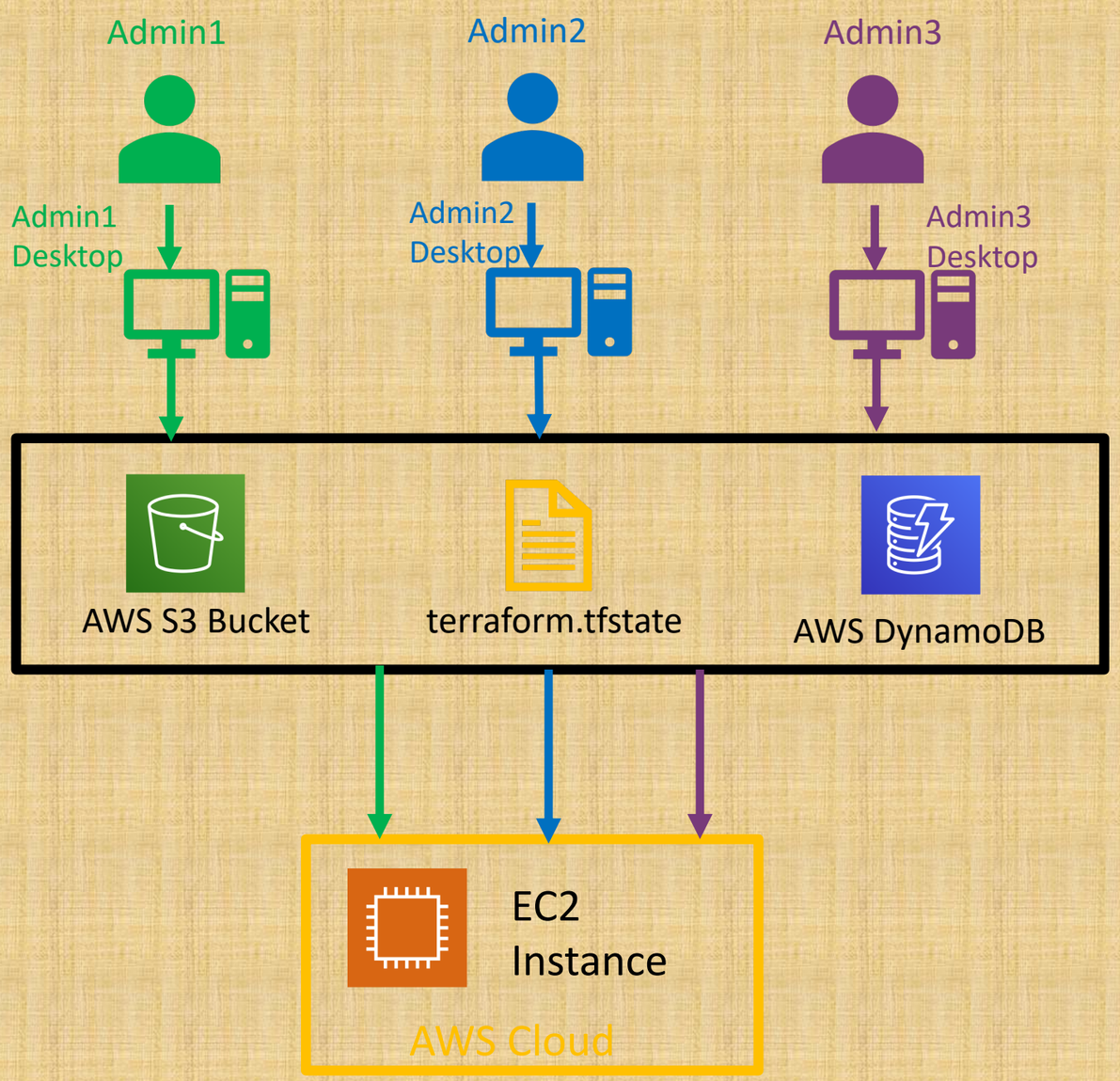
If two team members are running Terraform at the same time, you may run into race conditions as multiple Terraform processes make concurrent updates to the state files, leading to conflicts, data loss, and state file corruption.

Implement State Locking




Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Remote State File with State Locking




Not all backends support State Locking. AWS S3 supports State Locking

State locking happens automatically on all operations that could write state.

If state locking fails, Terraform will not continue.

You can disable state locking for most commands with the -lock flag but it is not recommended.

If acquiring the lock is taking longer than expected, Terraform will output a status message.

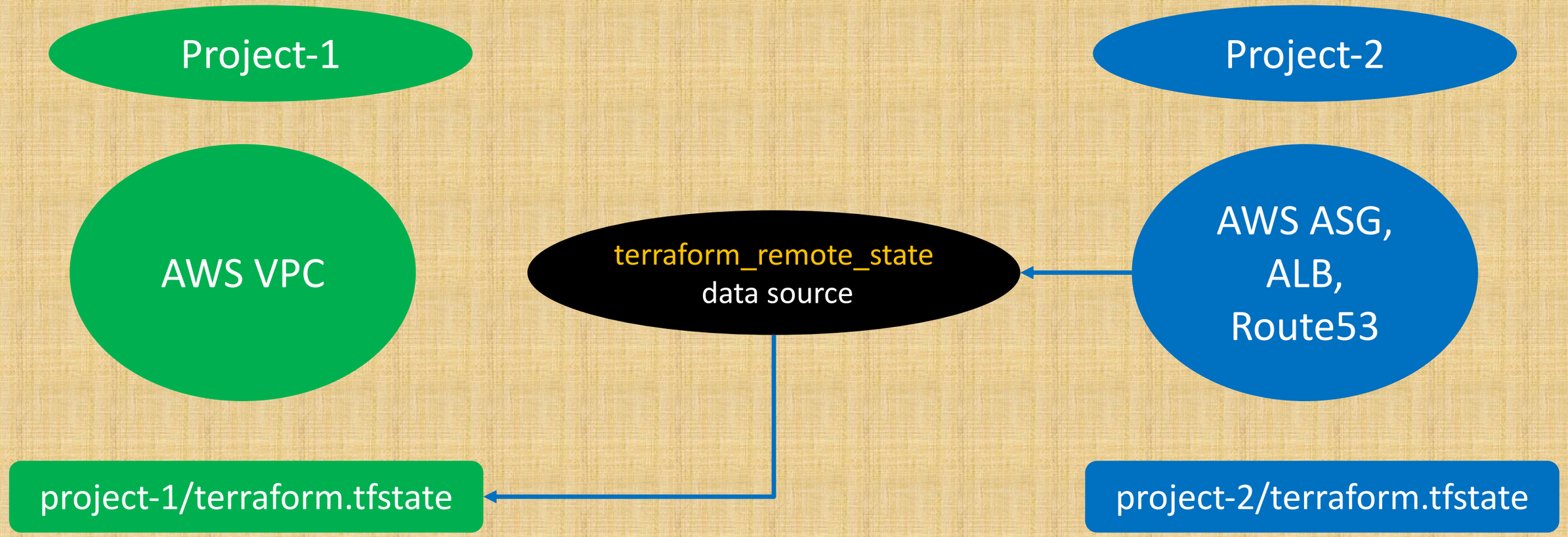
If Terraform doesn't output a message, state locking is still occurring if your backend supports it.

Terraform has a force-unlock command to manually unlock the state if unlocking failed.

Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Remote State Datasource

The terraform_remote_state data source retrieves the root module output values from some other Terraform configuration, using the latest state snapshot from the remote backend.




Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Remote State Datasource

Project-1

VPC Resources

Project-2

Autoscaling Groups

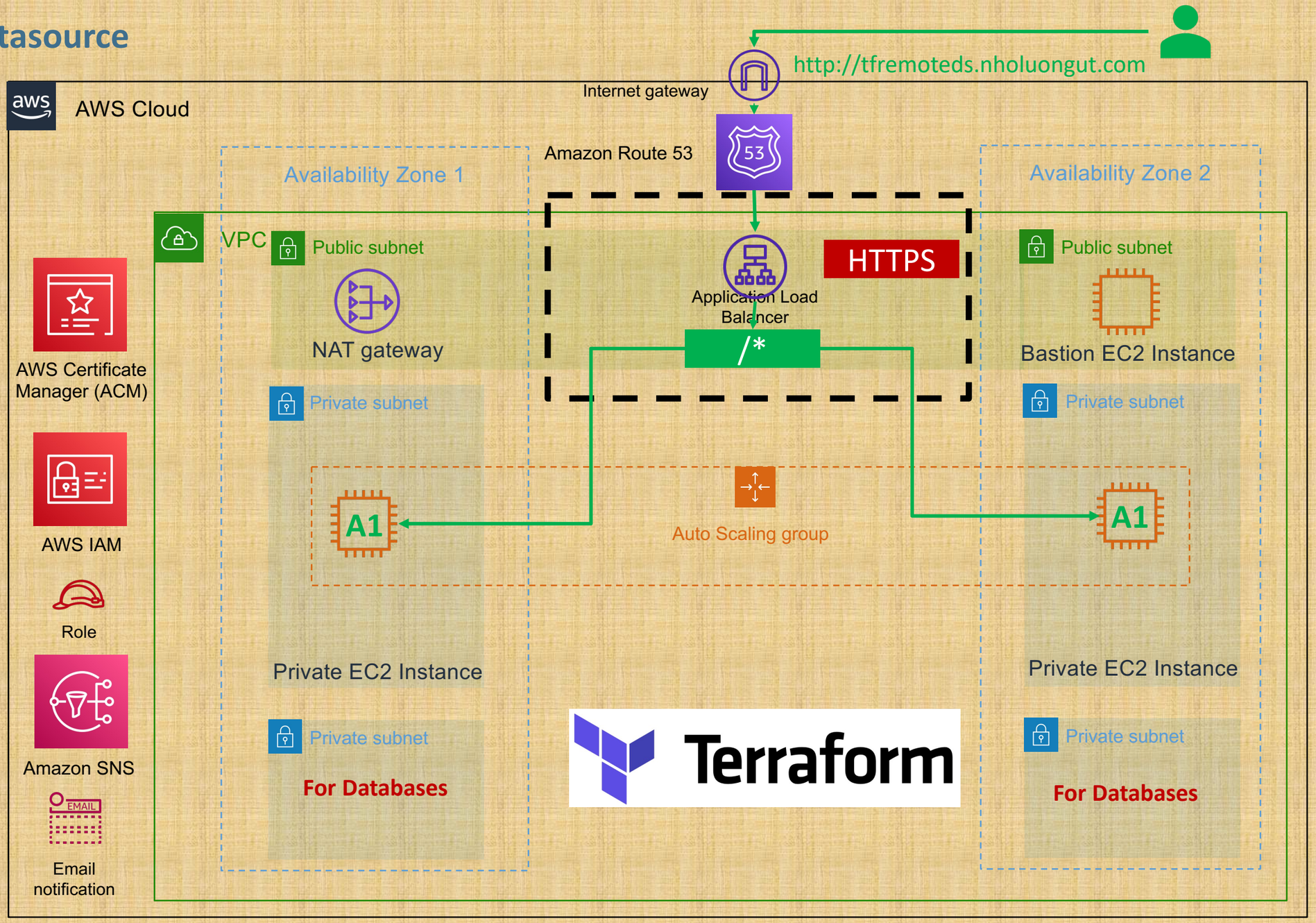
Application Load Balancers

AWS Route53

AWS Certificate Manager

AWS SNS

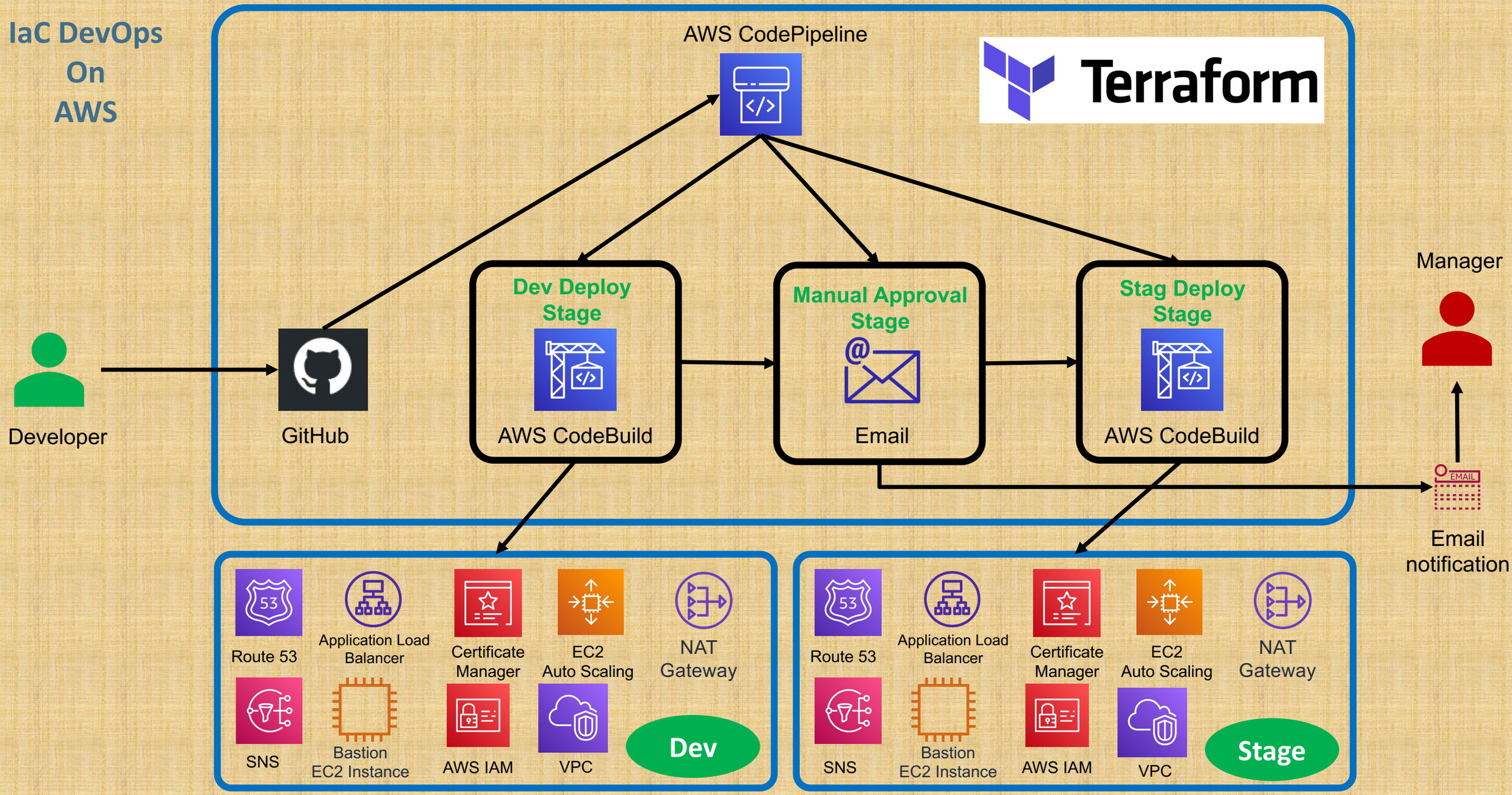
AWS IAM



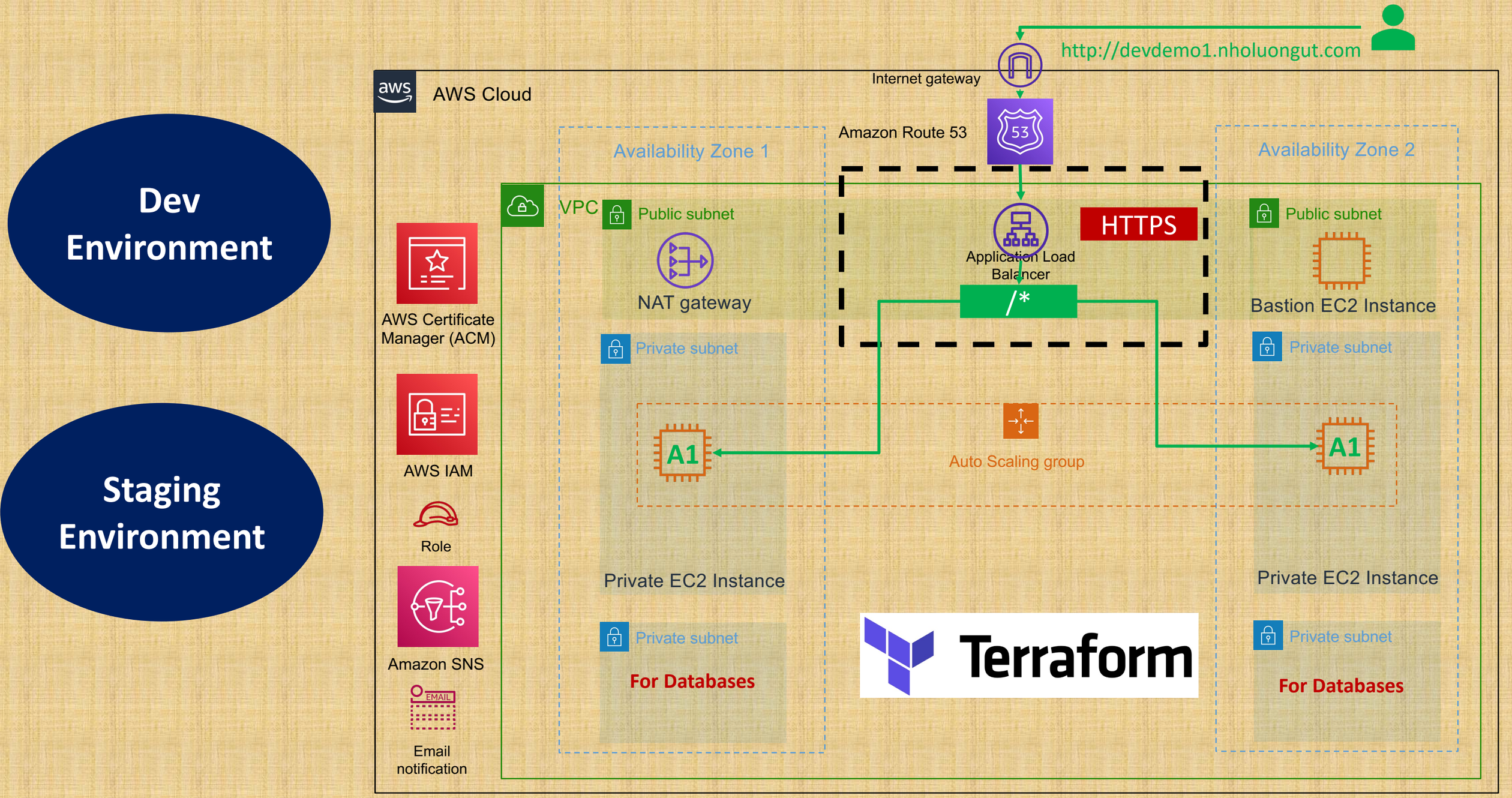


Author: Nho Luong
Skill: DevOps Engineer Lead

# IaC DevOps On AWS




Author: Nho Luong
Skill: DevOps Engineer Lead

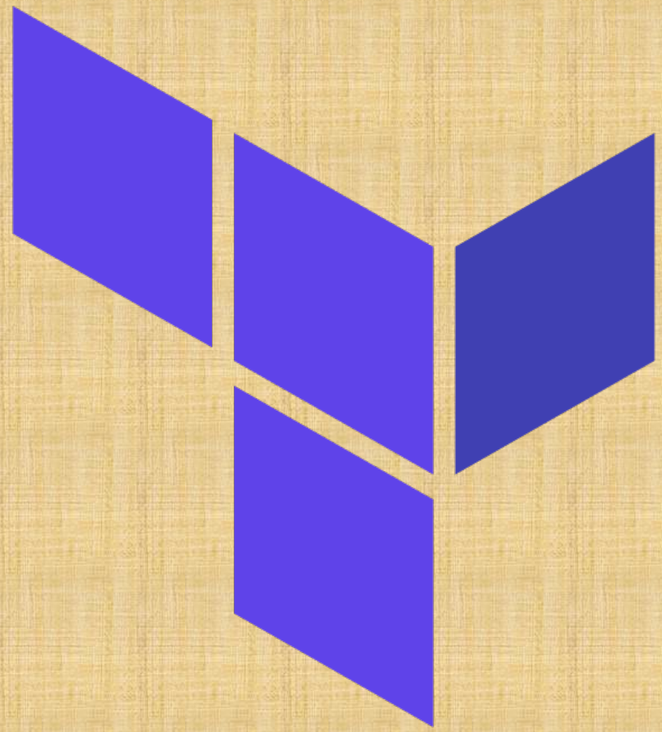



Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform

# Infrastructure as Code

# IaC

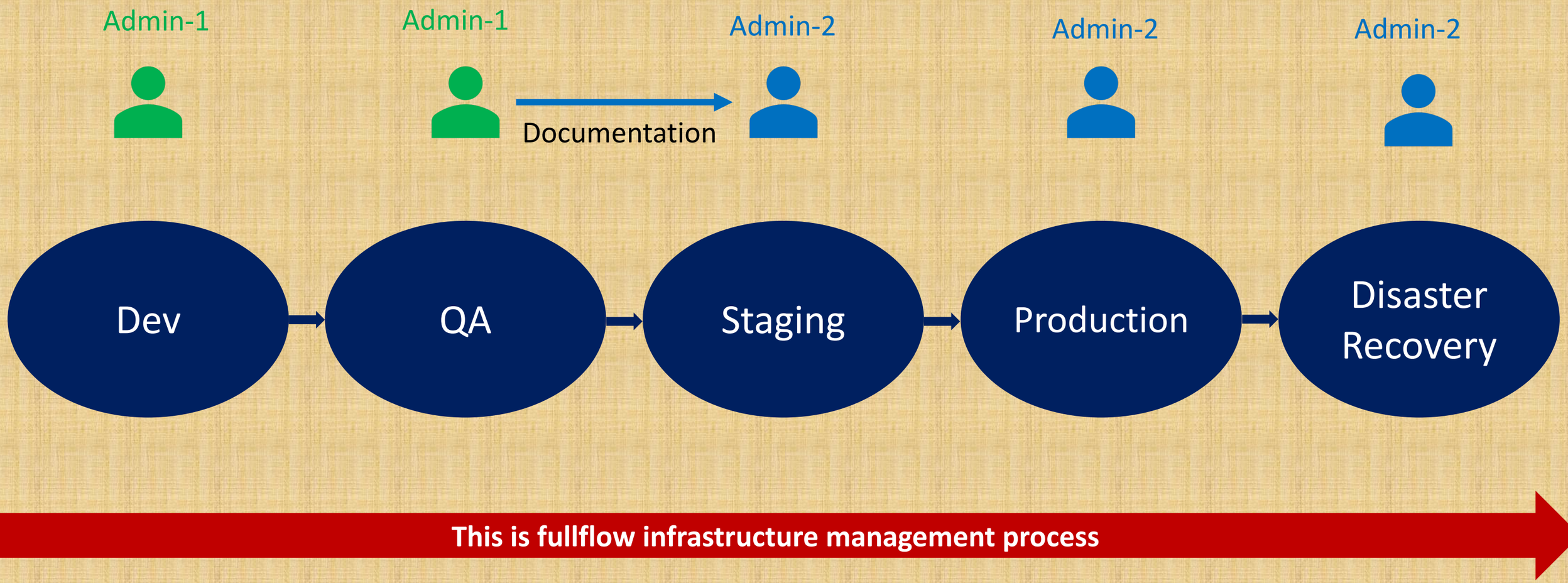


Author: Nho Luong
Skill: DevOps Engineer Lead

# What is Infrastructure as Code ?

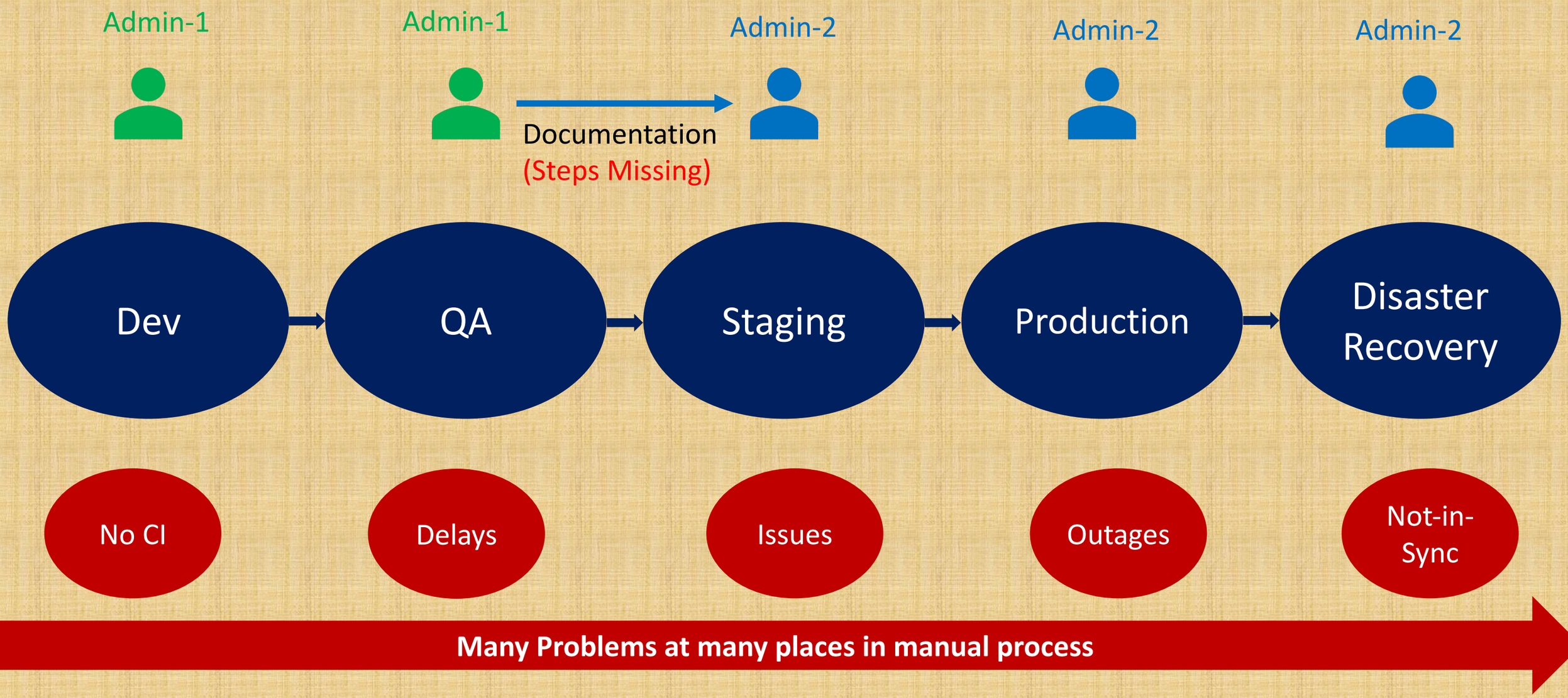
Author: Nho Luong
Skill: DevOps Engineer Lead

# Traditional Way of Managing Infrastructure



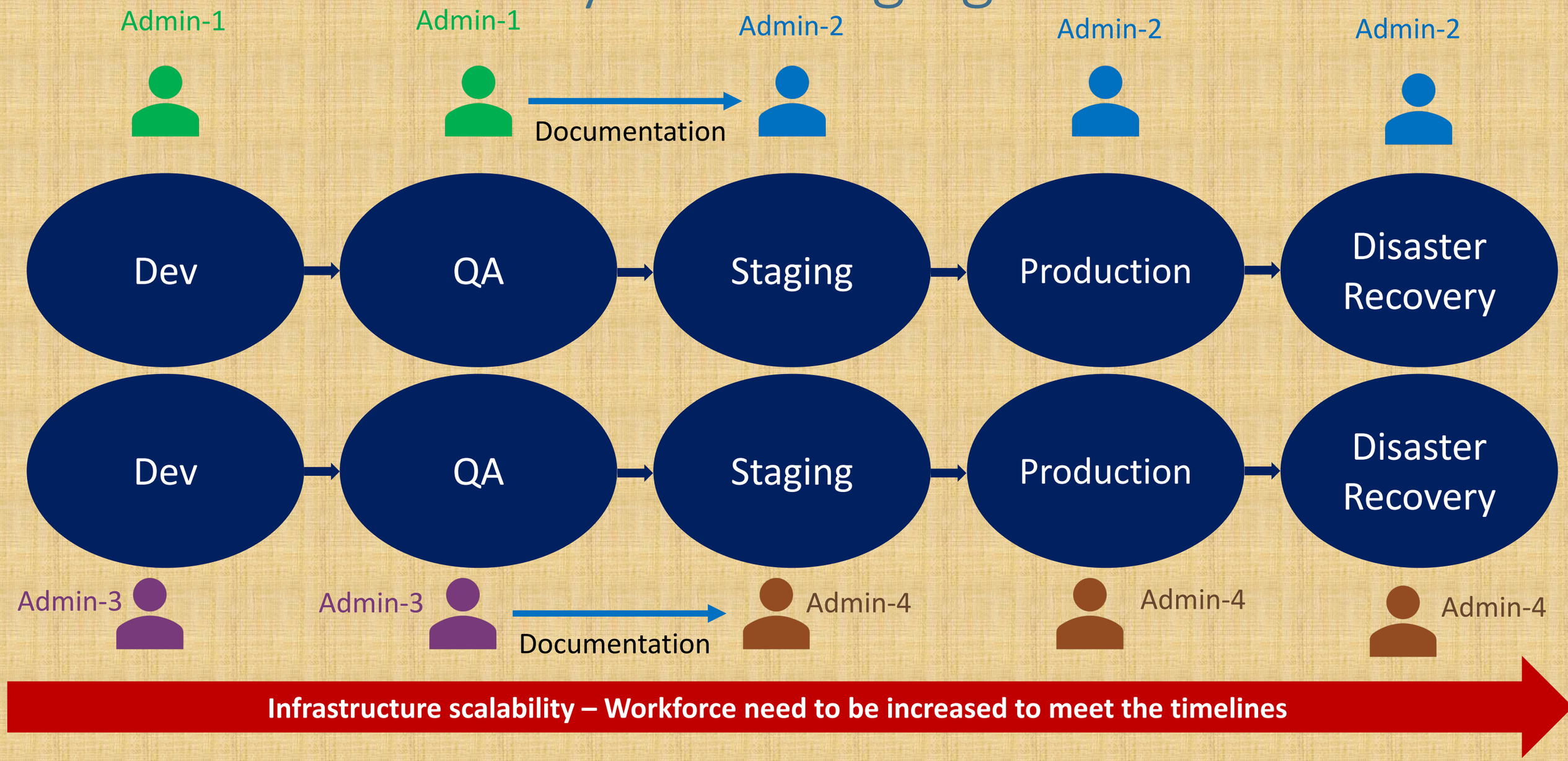

Author: Nho Luong
Skill: DevOps Engineer Lead

# Traditional Way of Managing Infrastructure



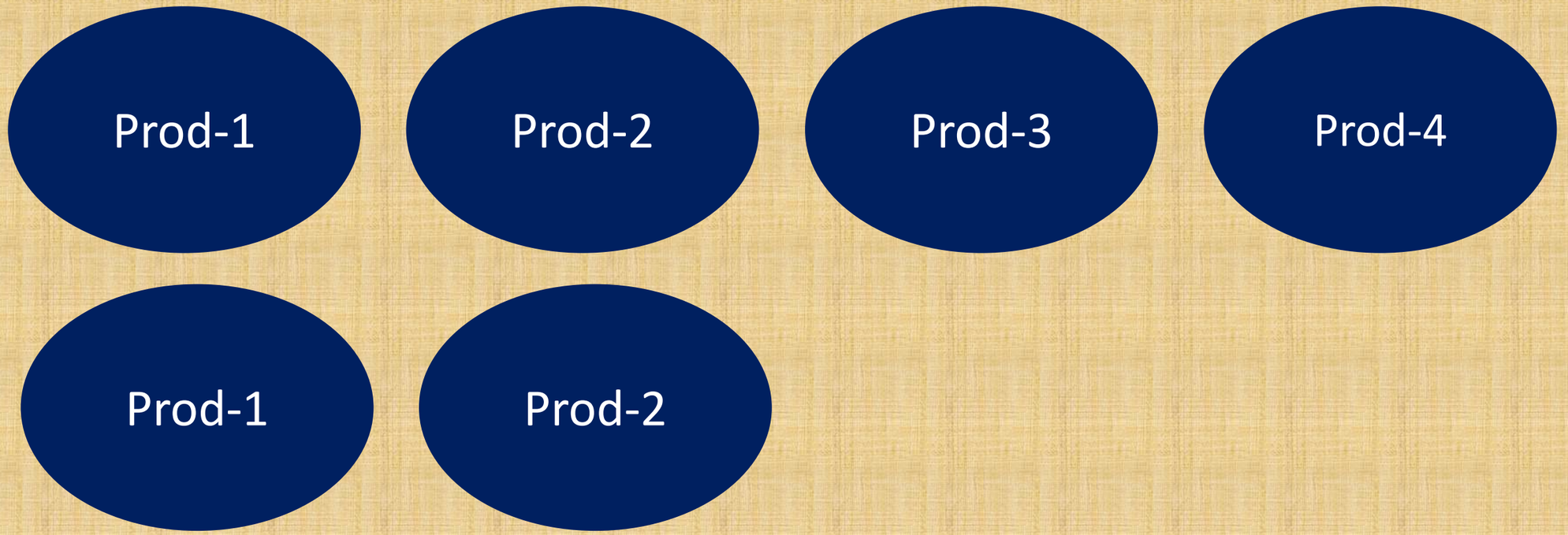

Author: Nho Luong
Skill: DevOps Engineer Lead

# Traditional Way of Managing Infrastructure




Author: Nho Luong
Skill: DevOps Engineer Lead

# Traditional Way of Managing Infrastructure




Scale Up

Scale Down

**On-Demand Scale-Up and Scale-Down is not an option**

Author: Nho Luong
Skill: DevOps Engineer Lead

# Manage using IaC with Terraform

5 Days

Admin-1

Check-In TF Code

Github

Triggers TF Runs

Terraform Cloud

DevOps / CI CD for IaC

Scale-Up and Scale-Down On-Demand

Creates Infra

Dev → QA → Staging → Production → Disaster Recovery

One-Time Work

Re-Use Templates

Quick & Fast

Reliable

Tracked for Audit

**Total Time: 25 Days reduced to 5 days, Provisioning environments will be in minutes or seconds**

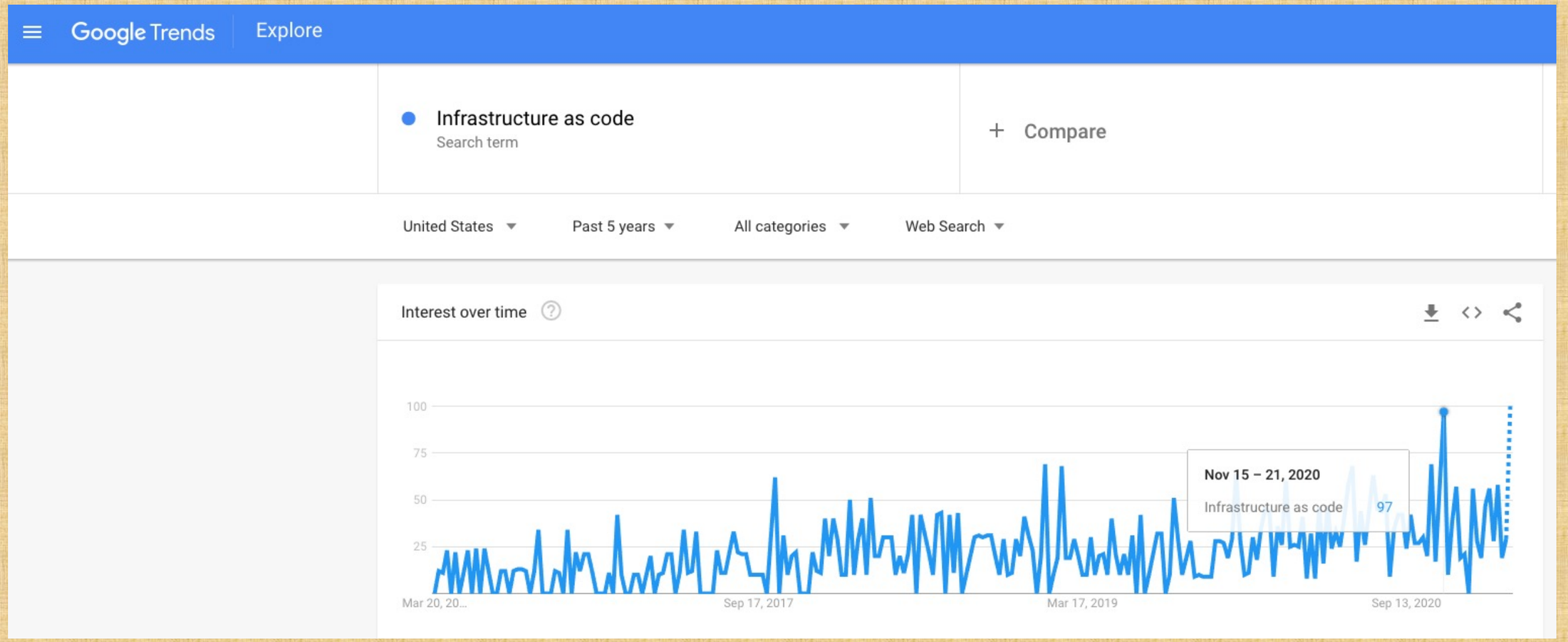
Author: Nho Luong
Skill: DevOps Engineer Lead

# Manage using IaC with Terraform

| | |
|---|---|
| Visibility | IaC serves as a very clear reference of what resources we created, and what their settings are. We don't have to navigate to the web console to check the parameters. |
| Stability | If you accidentally change the wrong setting or delete the wrong resource in the web console you can break things. IaC helps solve this, especially when it is combined with version control, such as Git. |
| Scalability | With IaC we can write it once and then reuse it many times. This means that one well written template can be used as the basis for multiple services, in multiple regions around the world, making it much easier to horizontally scale. |
| Security | Once again IaC gives you a unified template for how to deploy our architecture. If we create one well secured architecture we can reuse it multiple times, and know that each deployed version is following the same settings. |
| Audit | Terraform not only creates resources it also maintains the record of what is created in real world cloud environments using its State files. |

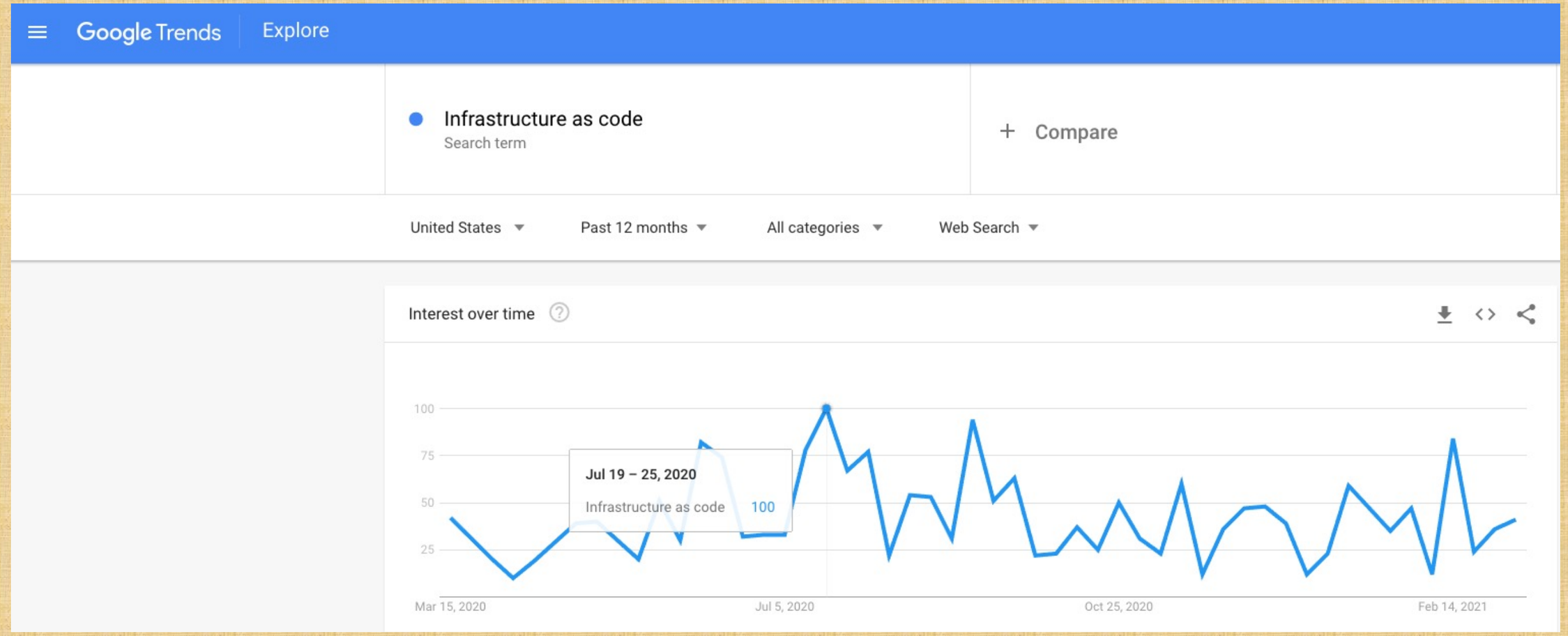
Author: Nho Luong
Skill: DevOps Engineer Lead

# Google Trends – Past 5 Years



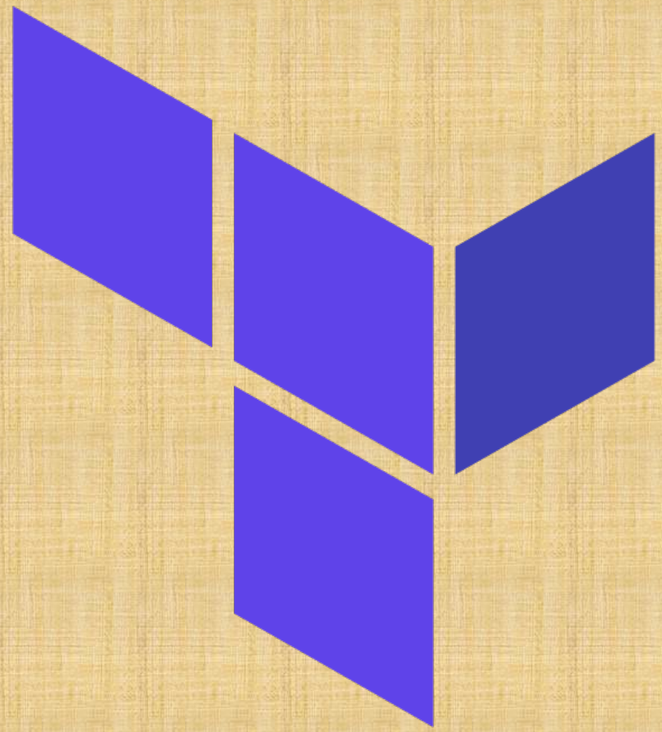

Author: Nho Luong
Skill: DevOps Engineer Lead

# Google Trends – Past 1 Year




Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform
# Installation



Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Installation

Terraform CLI

AWS CLI

VS Code Editor

Terraform plugin for VS Code

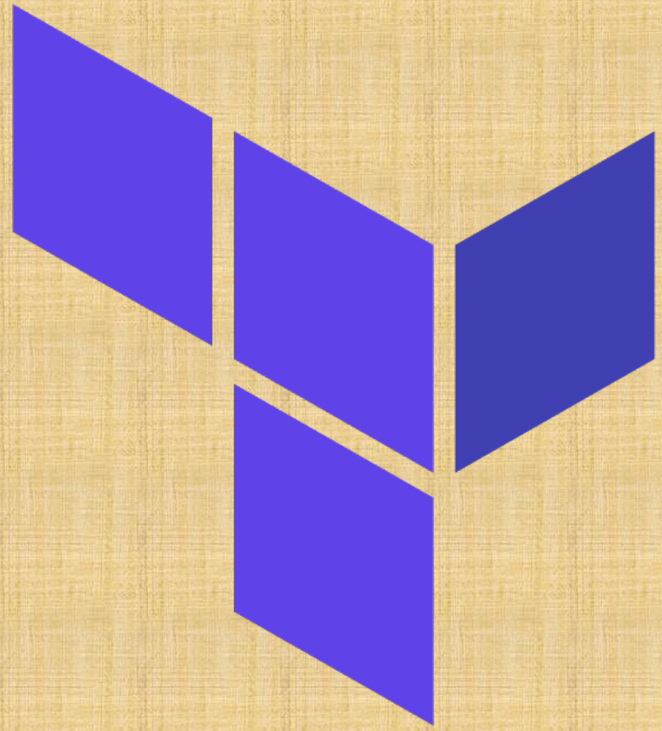
Mac OS

Windows OS

Linux OS

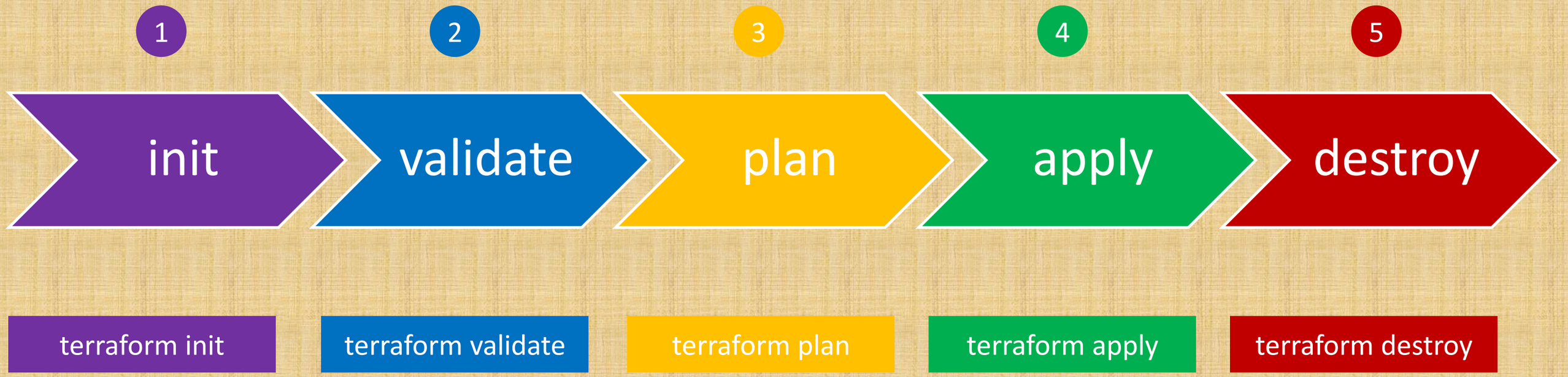
Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform
# Command Basics



Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Workflow




Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Workflow

1 2 3 4 5

## init

- Used to Initialize a working directory containing terraform config files
- This is the first command that should be run after writing a new Terraform configuration
- Downloads Providers

## validate

- Validates the terraform configurations files in that respective directory to ensure they are syntactically valid and internally consistent.

## plan

- Creates an execution plan
- Terraform performs a refresh and determines what actions are necessary to achieve the desired state specified in configuration files

## apply

- Used to apply the changes required to reach the desired state of the configuration.
- By default, apply scan s the current directory for the configuration and applies the changes appropriately.

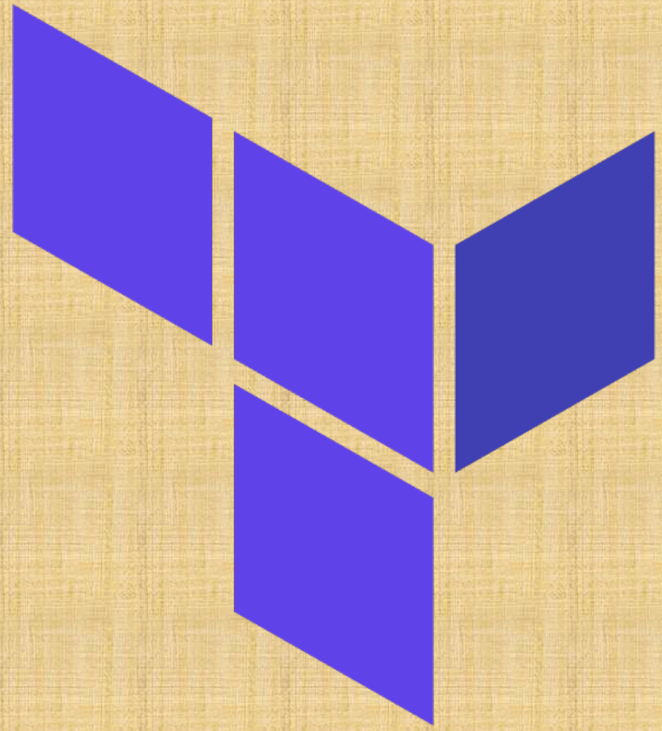
## destroy

- Used to destroy the Terraform-managed infrastructure
- This will ask for confirmation before destroying.

Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform

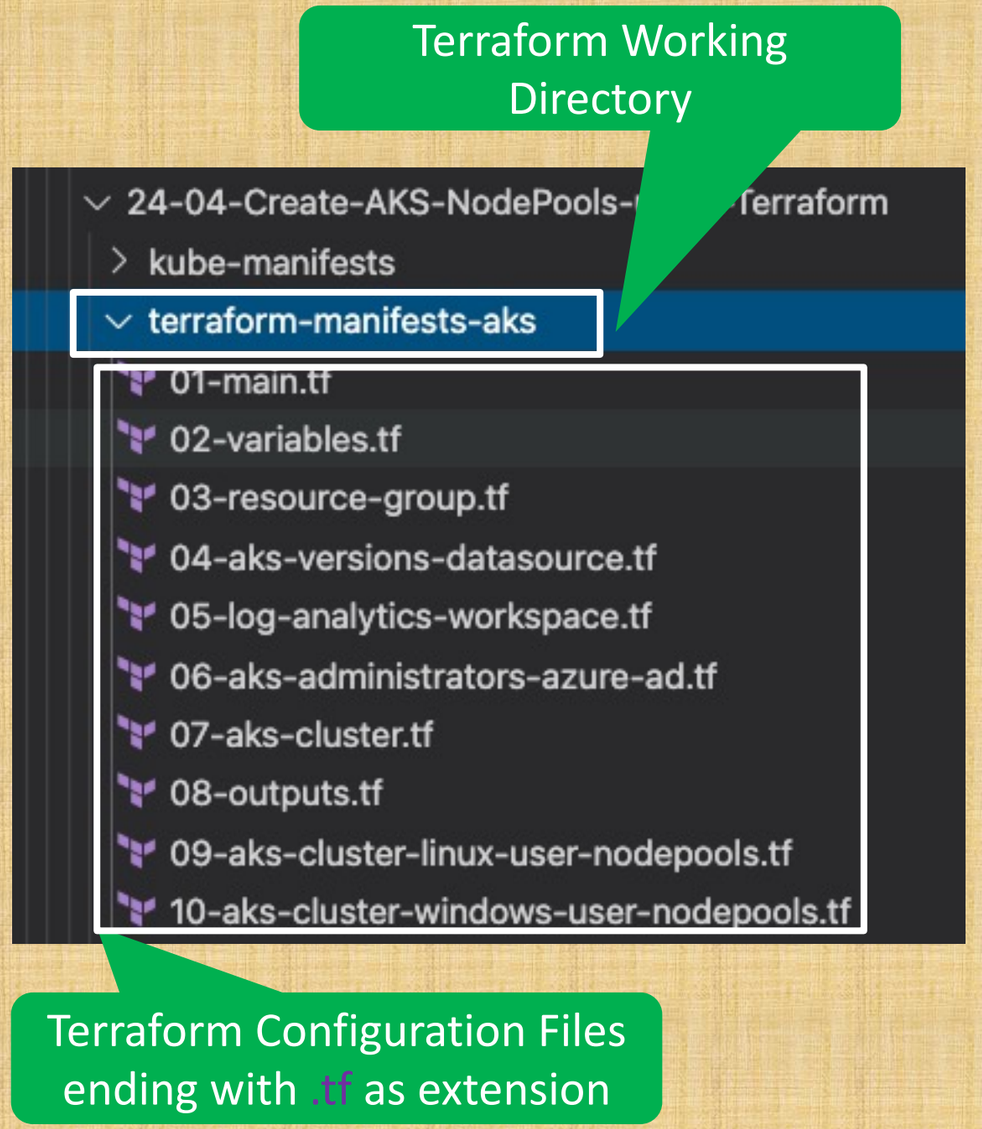
# Language Basics



Author: Nho Luong
Skill: DevOps Engineer Lead

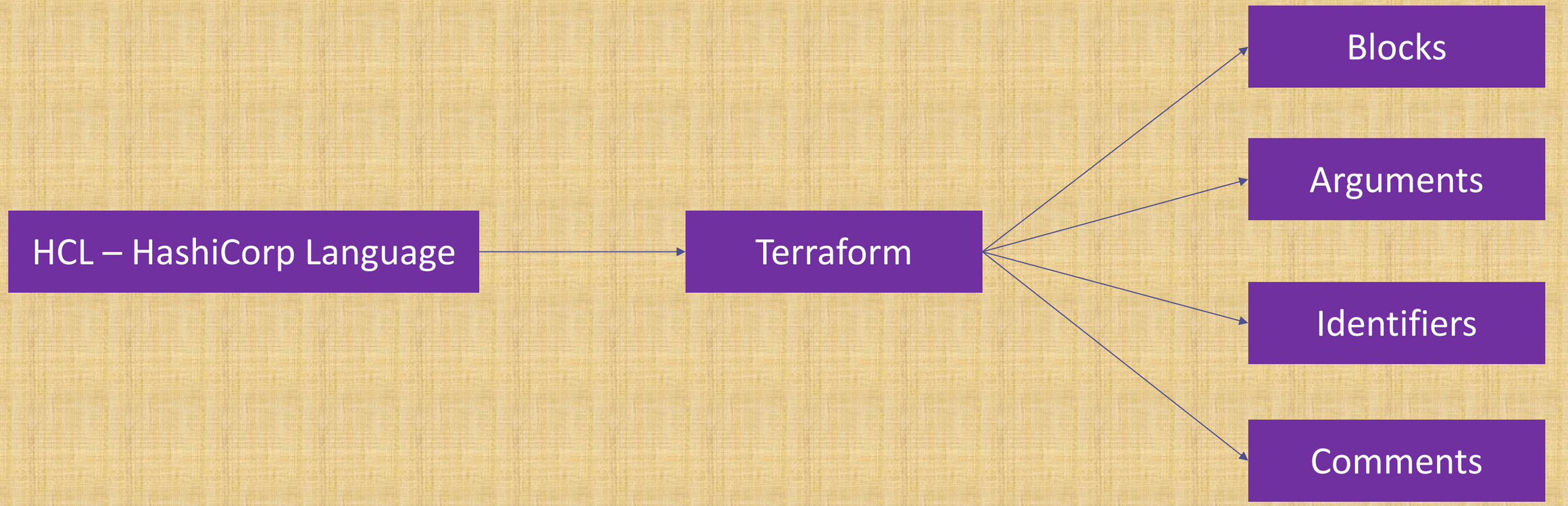
# Terraform Language Basics – Files

- Code in the Terraform language is stored in plain text files with the .tf file extension.
- There is also a JSON-based variant of the language that is named with the .tf.json file extension.
- We can call the files containing terraform code as Terraform Configuration Files or Terraform Manifests




Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Language Basics – Configuration Syntax




Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Language Basics – Configuration Syntax

```
# Template
<BLOCK TYPE> "<BLOCK LABEL>" "<BLOCK LABEL>"   {
  # Block body
  <IDENTIFIER> = <EXPRESSION> # Argument
}

# AWS Example
resource "aws_instance" "ec2demo" {
  ami           = "ami-04d29b6f966df1537"
  instance_type = "t2.micro"
}
```

Block Labels

Block Type

Top Level & Block inside Blocks

**Top Level Blocks:** resource, provider

**Block Inside Block:** provisioners, resource specific blocks like tags

Arguments

Based on Block Type block labels will be 1 or 2
**Example:**
Resource – 2 labels
Variables – 1 label

Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Language Basics – Configuration Syntax

```
# Template
<BLOCK TYPE> "<BLOCK LABEL>" "<BLOCK LABEL>"   {
  # Block body
  <IDENTIFIER> = <EXPRESSION> # Argument
}

# AWS Example
resource "aws_instance" "ec2demo" {
  ami           = "ami-04d29b6f966df1537"
  instance_type = "t2.micro"
}
```

Argument Name [or] Identifier

Argument Value [or] Expression

Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Language Basics – Configuration Syntax

Single Line Comments with # or //

Multi-line comment

```
# EC2 Instance Resource
resource "aws_instance" "ec2demo" {
  ami           = "ami-0885b1f6bd170450c" // Ubuntu 20.04 LTS
  instance_type = "t2.micro"
  /*
  Multi-line comments
  Line-1
  Line-2
  */
}
```

Author: Nho Luong
Skill: DevOps Engineer Lead

**Terraform** language uses a limited number of **top-level block** types, which are **blocks** that can appear outside of any other **block** in a TF configuration file.

# Terraform Top-Level Blocks

Most of **Terraform's features** are implemented as **top-level** blocks.

## Fundamental Blocks

- Terraform Block
- Providers Block
- Resources Block

## Variable Blocks

- Input Variables Block
- Output Values Block
- Local Values Block

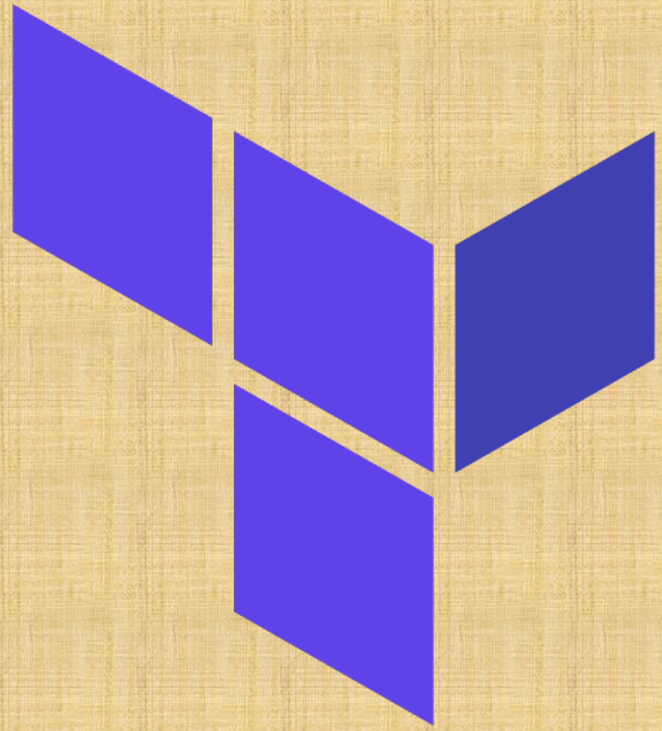
## Calling / Referencing Blocks

- Data Sources Block
- Modules Block

Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform

# Fundamental Blocks



Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Basic Blocks

## Terraform Block

Special block used to configure some behaviors

Required Terraform Version

List Required Providers

Terraform Backend

## Provider Block

HEART of Terraform

Terraform relies on providers to interact with Remote Systems

Declare providers for Terraform to install providers & use them

Provider configurations belong to Root Module

## Resource Block

Each Resource Block describes one or more Infrastructure Objects

Resource Syntax: How to declare Resources?

Resource Behavior: How Terraform handles resource declarations?

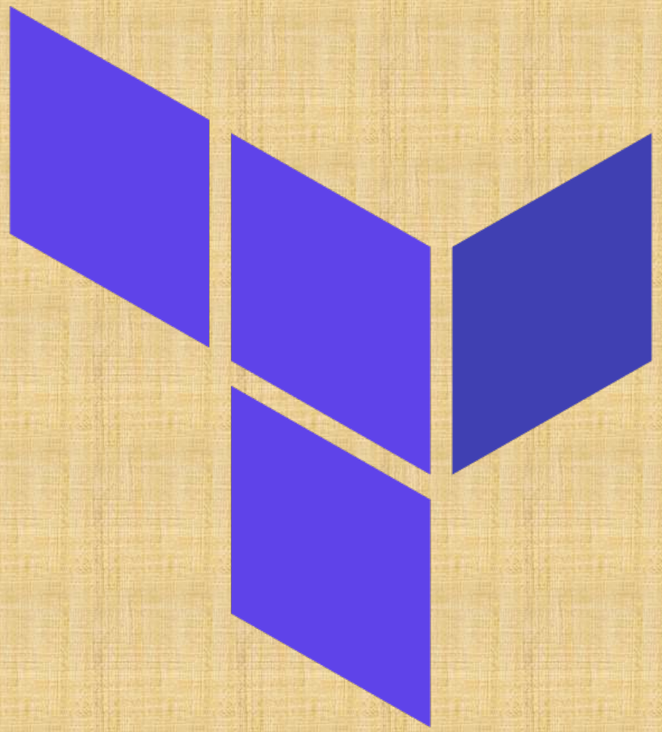
Provisioners: We can configure Resource post-creation actions

c1-versions.tf

c2-resource-name.tf

Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Block



Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Block

- This block can be called in 3 ways. All means the same.
  - Terraform Block
  - Terraform Settings Block
  - Terraform Configuration Block
- Each terraform block can contain a number of settings related to Terraform's behavior.
- **VERY VERY IMPORTANT TO MEMORIZE**
  - Within a terraform block, only constant values can be used; arguments may not refer to named objects such as resources, input variables, etc, and may not use any of the Terraform language built-in functions.

Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Block from 0.13 onwards

Terraform 0.12 and earlier:

```
# Configure the AWS Provider
provider "aws" {
  version = "~> 3.0"
  region  = "us-east-1"
}

# Create a VPC
resource "aws_vpc" "example" {
  cidr_block = "10.0.0.0/16"
}
```

Terraform 0.13 and later:

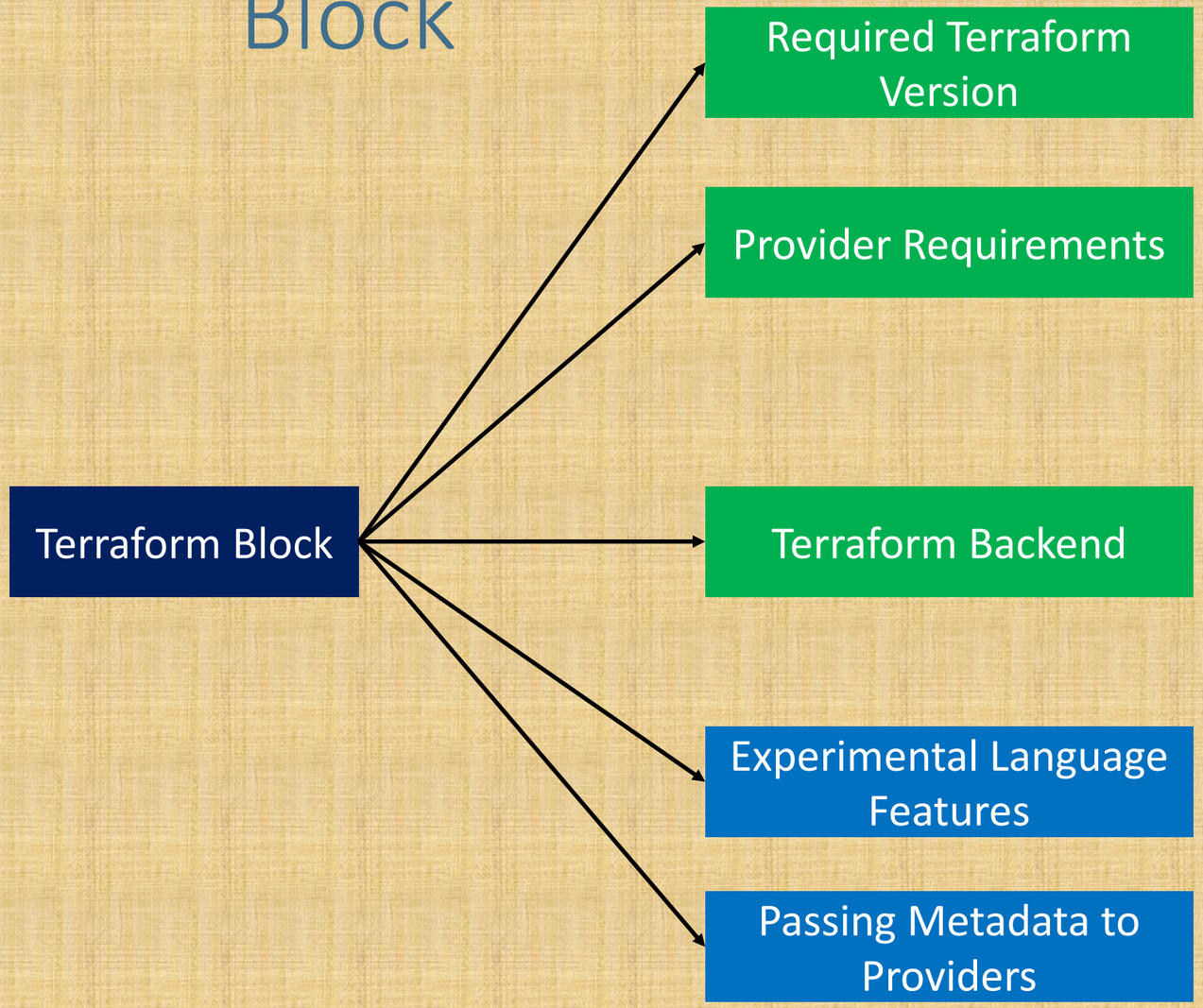
```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
}

# Configure the AWS Provider
provider "aws" {
  region = "us-east-1"
}

# Create a VPC
resource "aws_vpc" "example" {
  cidr_block = "10.0.0.0/16"
}
```

Author: Nho Luong
Skill: DevOps Engineer Lead

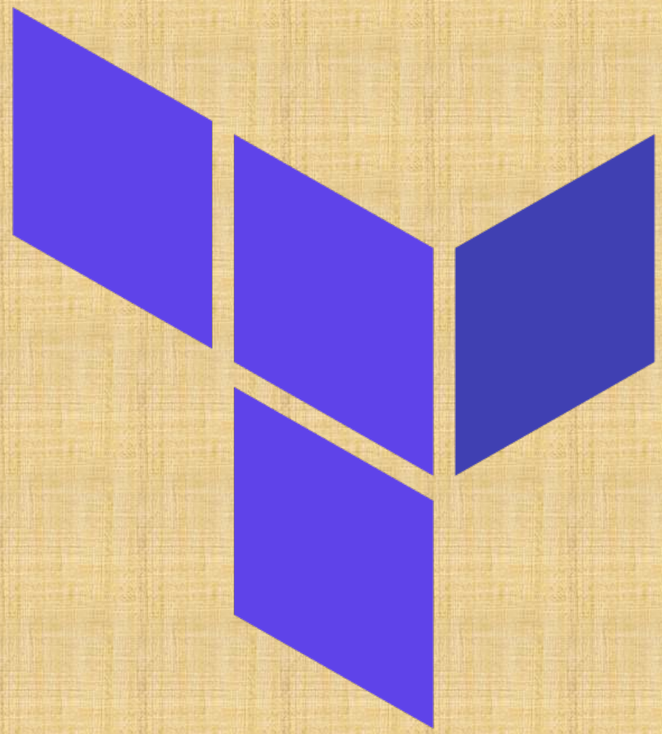
# Terraform Block




```hcl
terraform {
  # Required Terraform Version
  required_version = "~> 0.14.3"
  # Required Providers and their Versions
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.21" # Optional but recommended
    }
  }
  # Remote Backend for storing Terraform State in S3 bucket
  backend "s3" {
    bucket = "mybucket"
    key    = "path/to/my/key"
    region = "us-east-1"
  }
  # Experimental Features (Not required)
  experiments = [ example ]
  # Passing Metadata to Providers (Super Advanced - Terrafor
  provider_meta "my-provider" {
    hello = "world"
  }
}
```

Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform
# Providers



Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Providers

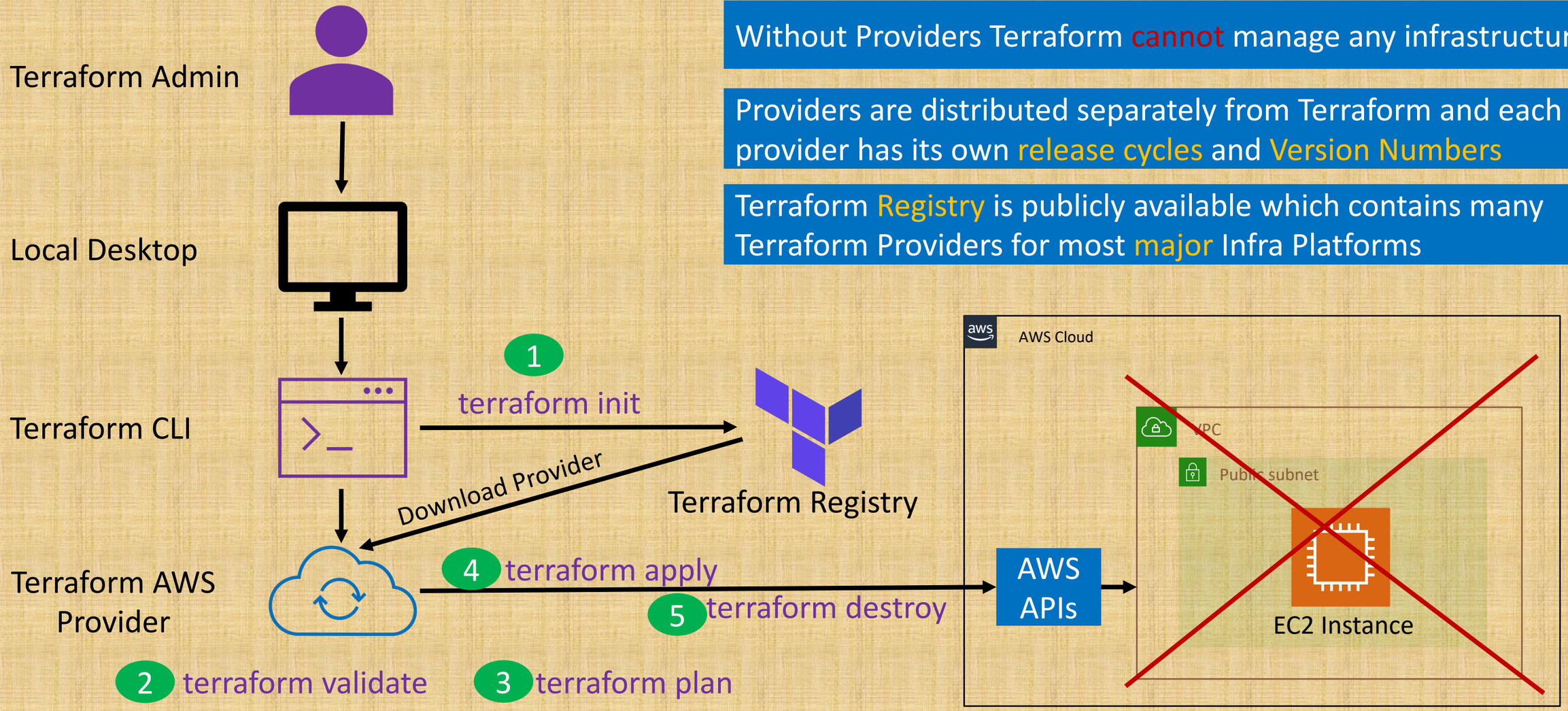
Providers are HEART of Terraform

Every Resource Type (example: EC2 Instance), is implemented by a Provider

Without Providers Terraform cannot manage any infrastructure.

Providers are distributed separately from Terraform and each provider has its own release cycles and Version Numbers

Terraform Registry is publicly available which contains many Terraform Providers for most major Infra Platforms




Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Providers

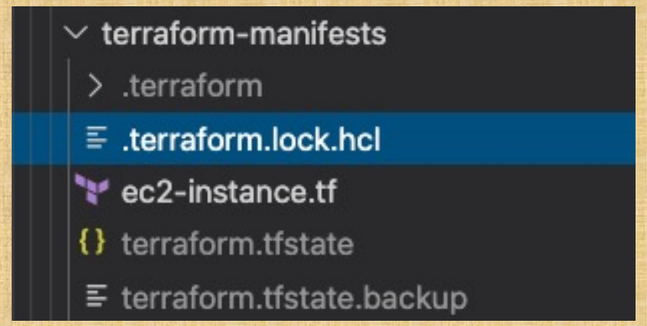
## Provider Requirements

```
# Terraform Block
terraform {
  required_version = "~> 0.14.3"
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
}
```

## Provider Configuration

```
# Provider Block
provider "aws" {
  profile = "default"
  region  = "us-east-1"
}
```

## Dependency Lock File




```
# This file is maintained automatically by "terraform init".
# Manual edits may be lost in future updates.

provider "registry.terraform.io/hashicorp/aws" {
  version = "3.22.0"
  hashes = [
    "h1:f/Tz8zv1Zb78ZaiyJkQ0MGIViZwbYrLuQk3kojPM91c=",
    "zh:4a9a66caf1964cdd3b61fb3ebb0da417195a5529cb8e496f266b0778335d11c8",
    "zh:514f2f006ae68db715d86781673faf9483292deab235c7402ff306e0e92ea11a",
    "zh:5277b61109fddb9011728f6650ef01a639a0590aeffe34ed7de7ba10d0c31803",
    "zh:67784dc8c8375ab37103eea1258c3334ee92be6de033c2b37e3a2a65d0005142",
    "zh:76d4c8be2ca4a3294fb51fb58de1fe03361d3bc403820270cc8e71a04c5fa806",
    "zh:8f90b1cfdcf6e8fb1a9d0382ecaa5056a3a84c94e313fbf9e92c89de271cdede",
    "zh:d0ac346519d0df124df89be2d803eb53f373434890f6ee3fb37112802f9eac59",
    "zh:d6256feedada82cbfb3b1dd6dd9ad02048f23120ab50e6146a541cb11a108cc1",
    "zh:db2fe0d2e77c02e9a74e1ed694aa352295a50283f9a1cf896e5be252af14e9f4",
    "zh:eda61e889b579bd90046939a5b40cf5dc9031fb5a819fc3e4667a78bd432bdb2",
  ]
}
```

Author: Nho Luong
Skill: DevOps Engineer Lead

# Dependency Lock File

```
# This file is maintained automatically by "terraform init".
# Manual edits may be lost in future updates.

provider "registry.terraform.io/hashicorp/aws" {
  version = "3.22.0"
```

Author: Nho Luong
Skill: DevOps Engineer Lead

# Required Providers

```
# Terraform Block
terraform {
  required_version = "~> 0.14.3"
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
}

# Provider Block
provider "aws" {
  profile = "default"
  region  = "us-east-1"
}
```

## Local Names

Local Names are Module specific and should be unique per-module

Terraform configurations always refer to local name of provider outside required_provider block

Users of a provider can choose any local name for it (myaws, aws1, aws2).

Recommended way of choosing local name is to use preferred local name of that provider (For AWS Provider: hashicorp/aws, preferred local name is aws)

## Source

It is the primary location where we can download the Terraform Provider

Source addresses consist of three parts delimited by slashes (/)

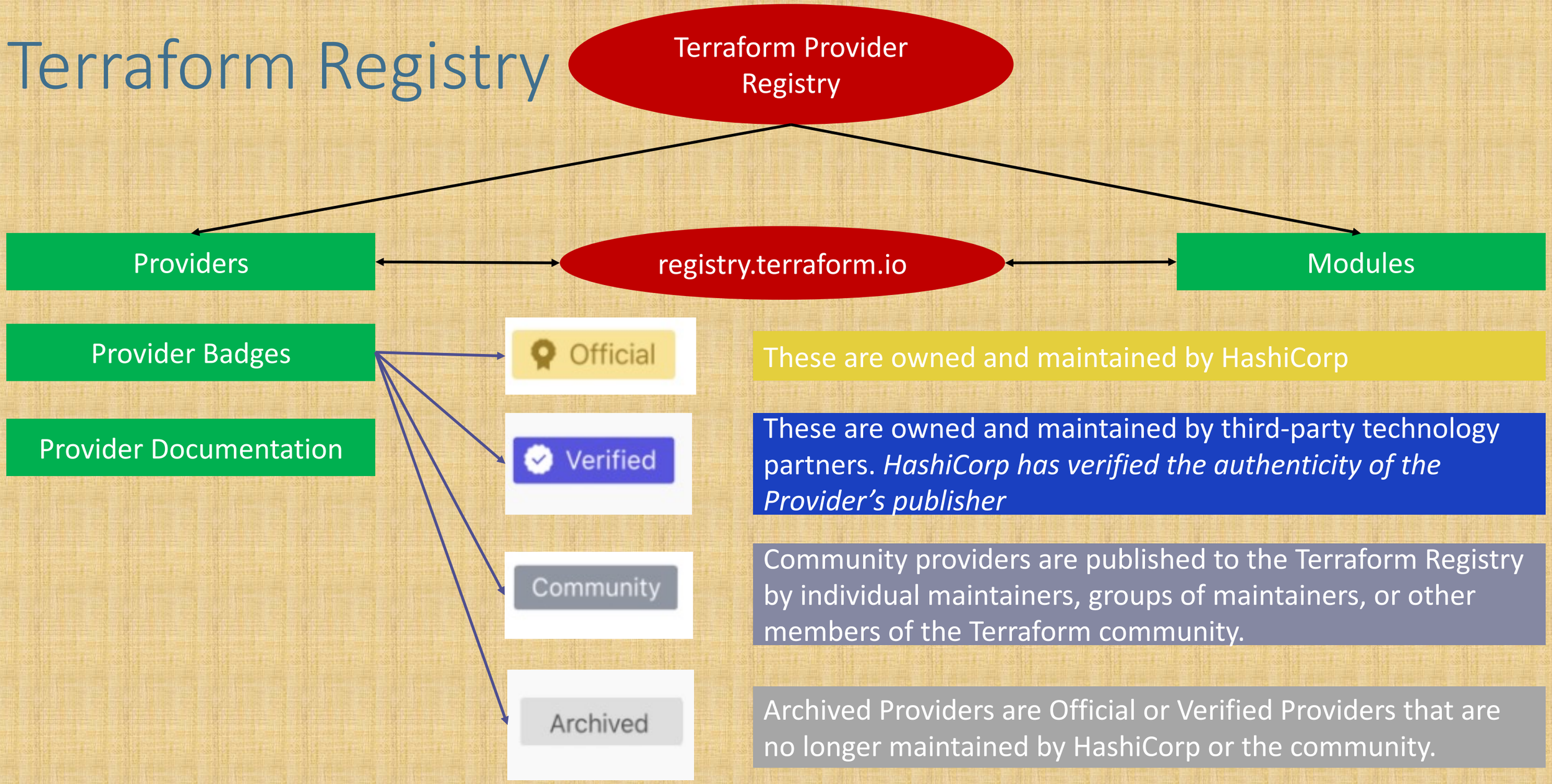
[<HOSTNAME>/]<NAMESPACE>/<TYPE>

registry.terraform.io/hashicorp/aws

Registry Name is optional as default is going to be Terraform Public Registry

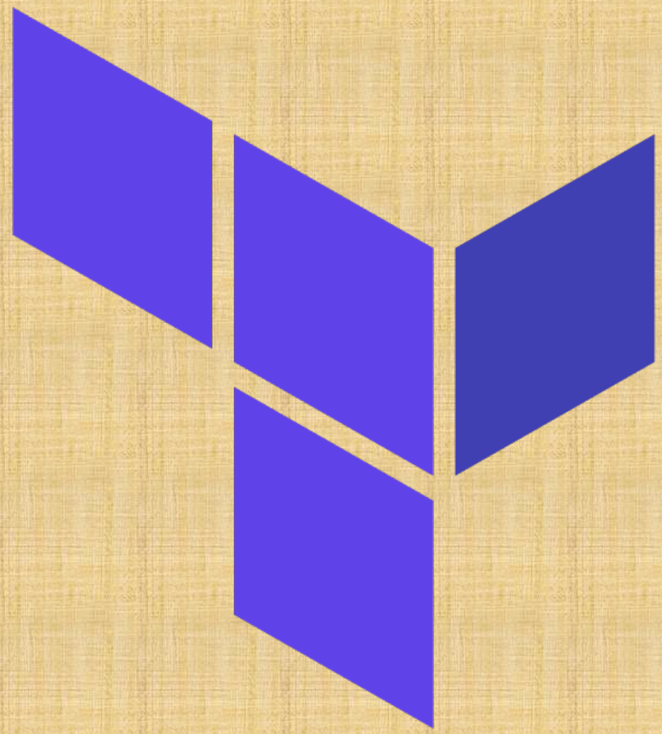
Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Registry




Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform
# Resources
# Introduction



Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Language Basics – Configuration Syntax

```
# Template
<BLOCK TYPE> "<BLOCK LABEL>" "<BLOCK LABEL>"   {
  # Block body
  <IDENTIFIER> = <EXPRESSION> # Argument
}

# AWS Example
resource "aws_instance" "ec2demo" {
  ami           = "ami-04d29b6f966df1537"
  instance_type = "t2.micro"
}
```

Block Labels

Block Type

Top Level & Block inside Blocks

**Top Level Blocks:** resource, provider

**Block Inside Block:** provisioners, resource specific blocks like tags

Arguments

Based on Block Type block labels will be 1 or 2
**Example:**
Resource – 2 labels
Variables – 1 label

Author: Nho Luong
Skill: DevOps Engineer Lead

# Resource Syntax

**Resource Type:** It determines the kind of infrastructure object it manages and what arguments and other attributes the resource supports.

**Resource Local Name:** It is used to refer to this resource from elsewhere in the same Terraform module, but has no significance outside that module's scope.
The resource type and name together serve as an identifier for a given resource and so must be unique within a module

**Meta-Arguments:** Can be used with any resource to change the behavior of resources

**Resource Arguments:** Will be specific to resource type. Argument Values can make use of Expressions or other Terraform Dynamic Language Features

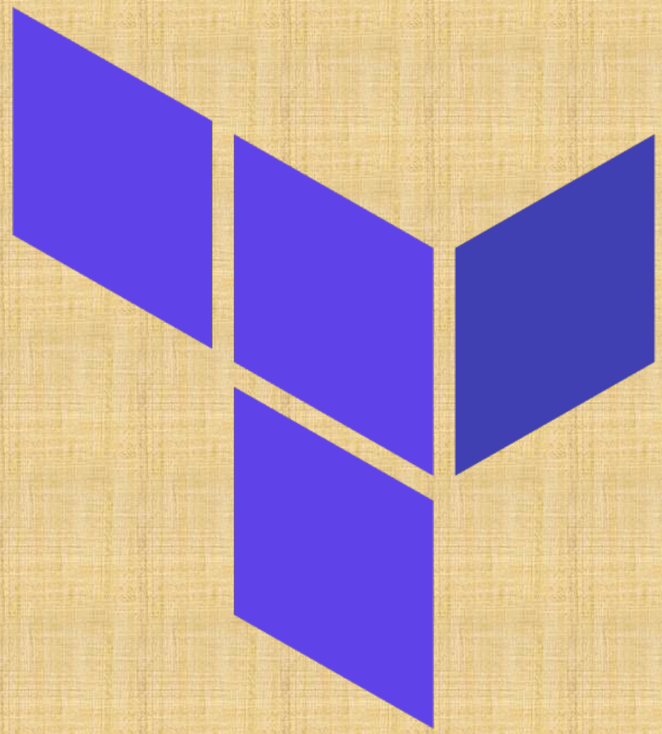
```
# Provider-2 for us-west-1
provider "aws" {
  region = "us-west-1"
  profile = "default"
  alias = "aws-west-1"
}

# Resource Block to Create VPC
resource "aws_vpc" "vpc_us-west-1" {
  provider = aws.aws-west-1
  cidr_block = "10.2.0.0/16"
  tags = {
    "Name" = "vpc-1"
  }
}
```

Author: Nho Luong
Skill: DevOps Engineer Lead

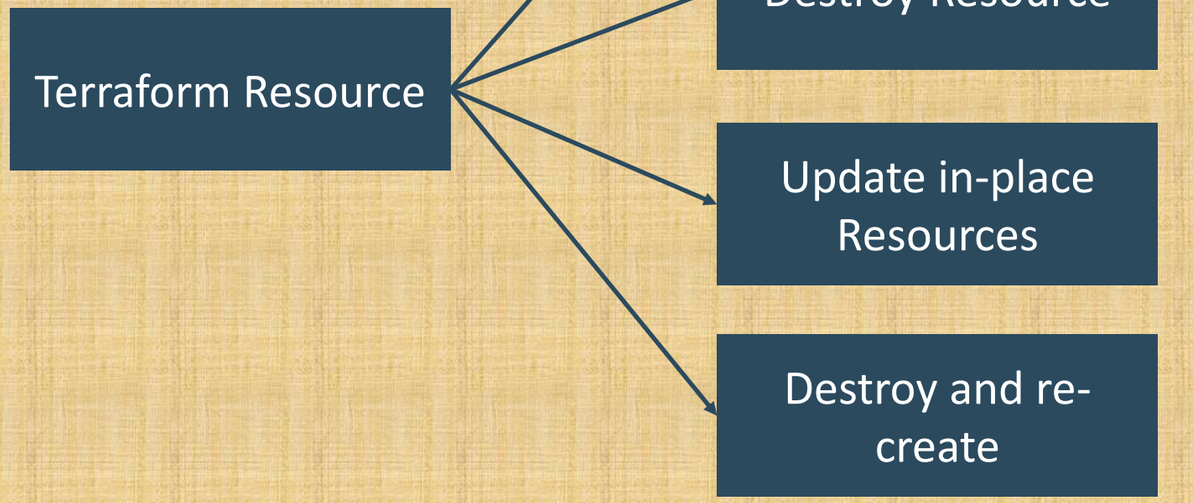
# Terraform

# State



Author: Nho Luong
Skill: DevOps Engineer Lead

# Resource Behavior




- **Create Resource**: Create resources that exist in the configuration but are not associated with a real infrastructure object in the state.
- **Destroy Resource**: Destroy resources that exist in the state but no longer exist in the configuration.
- **Update in-place Resources**: Update in-place resources whose arguments have changed.
- **Destroy and re-create**: Destroy and re-create resources whose arguments have changed but which cannot be updated in-place due to remote API limitations.

**Terraform State**

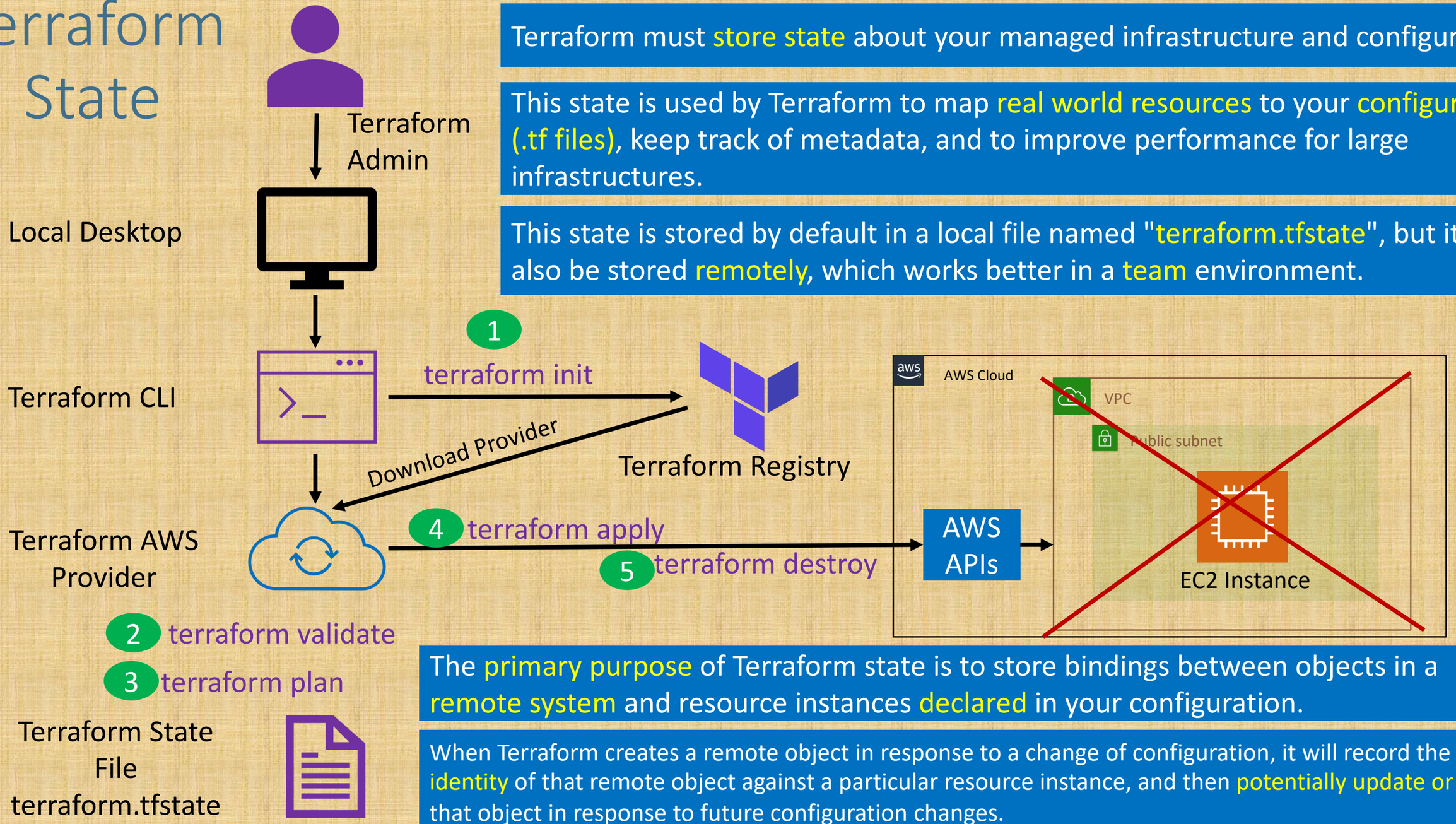
Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform State

Terraform must store state about your managed infrastructure and configuration

This state is used by Terraform to map real world resources to your configuration (.tf files), keep track of metadata, and to improve performance for large infrastructures.

This state is stored by default in a local file named "terraform.tfstate", but it can also be stored remotely, which works better in a team environment.




The primary purpose of Terraform state is to store bindings between objects in a remote system and resource instances declared in your configuration.

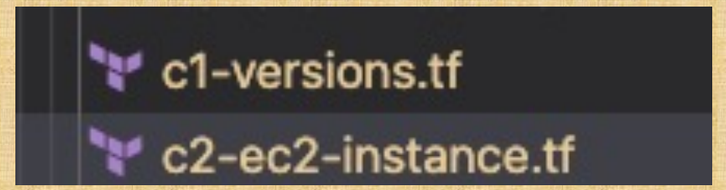
When Terraform creates a remote object in response to a change of configuration, it will record the identity of that remote object against a particular resource instance, and then potentially update or delete that object in response to future configuration changes.


Author: Nho Luong
Skill: DevOps Engineer Lead

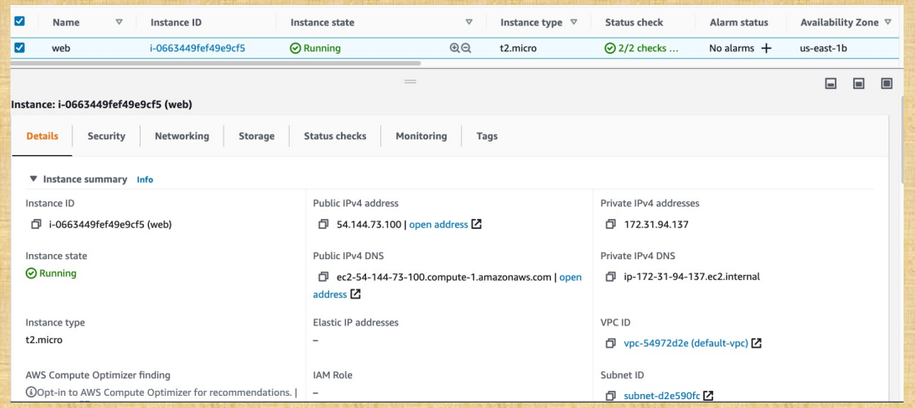
# Desired & Current Terraform States

Terraform Configuration Files

Real World Resource – EC2 Instance








Desired State

Current State

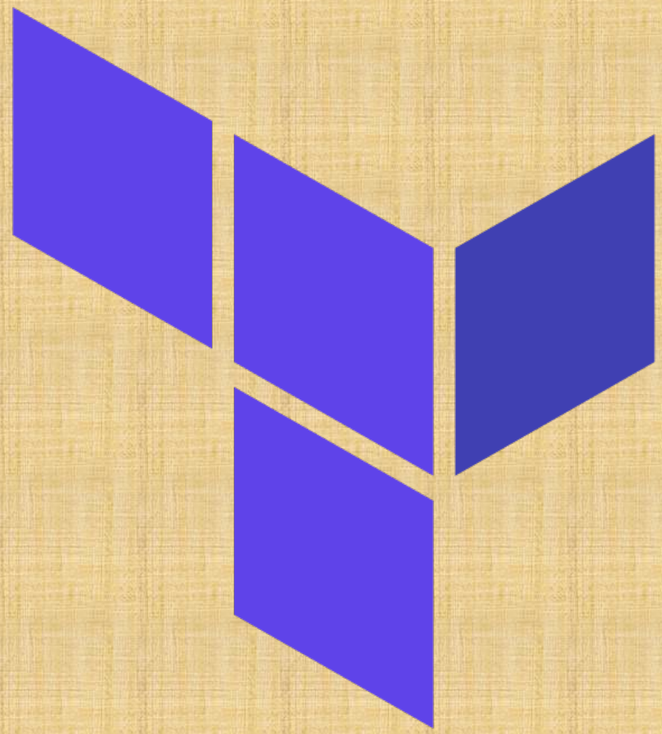
Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform

# Input Variables

# Datasources

# Outputs



Author: Nho Luong
Skill: DevOps Engineer Lead

# What are we going to learn ?

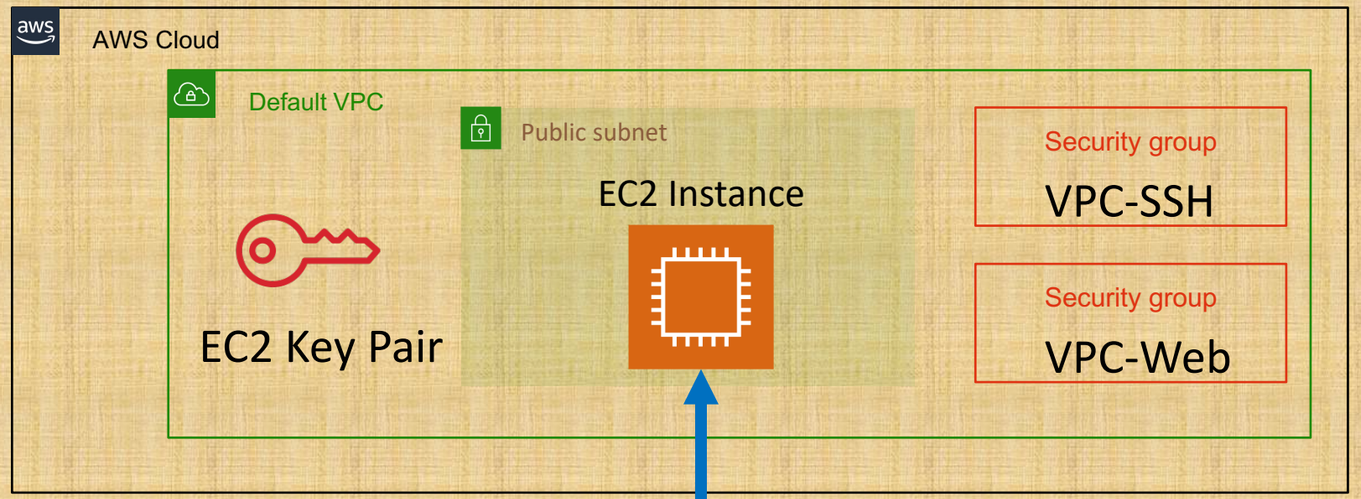
Terraform Concepts

AWS Services

Terraform Input Variables

Terraform Output Values

Terraform Datasources




**Dynamically get latest AMI ID**

Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Variables

## Terraform Input Variables

## Terraform Output Values

## Terraform Local Values

Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Input Variables

Input variables serve as parameters for a Terraform module, allowing aspects of the module to be customized without altering the module's own source code, and allowing modules to be shared between different configurations.

Terraform Input Variables

1. Input Variables - Basics
2. Provide Input Variables when prompted during terraform plan or apply
3. Override default variable values using CLI argument -var
4. Override default variable values using Environment Variables (TF_var_aa)
5. Provide Input Variables using terraform.tfvars files
6. Provide Input Variables using <any-name>.tfvars file with CLI argument -var-file
7. Provide Input Variables using auto.tfvars files
8. Implement complex type constructors like List & Map in Input Variables
9. Implement Custom Validation Rules in Variables
10. Protect Sensitive Input Variables

Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Datasources

*Data sources* allow data to be fetched or computed for use elsewhere in Terraform configuration.

Use of data sources allows a Terraform configuration to make use of information defined outside of Terraform, or defined by another separate Terraform configuration.

A data source is accessed via a special kind of resource known as a *data resource*, declared using a data block

Each data resource is associated with a single data source, which determines the kind of object (or objects) it reads and what query constraint arguments are available

Data resources have the same dependency resolution behavior as defined for managed resources. Setting the depends_on meta-argument within data blocks defers reading of the data source until after all changes to the dependencies have been applied.

```
# Get latest AMI ID for Amazon Linux2 OS
data "aws_ami" "amzlinux" {
  most_recent      = true
  owners           = ["amazon"]
  filter {
    name   = "name"
    values = ["amzn2-ami-hvm-*"]
  }
  filter {
    name   = "root-device-type"
    values = ["ebs"]
  }
  filter {
    name   = "virtualization-type"
    values = ["hvm"]
  }
  filter {
    name   = "architecture"
    values = ["x86_64"]
  }
}
```

Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Datasources

We can refer the data resource in a resource as depicted

```
# Create EC2 Instance - Amazon Linux
resource "aws_instance" "my-ec2-vm" {
  ami                    = data.aws_ami.amzlinux.id
  instance_type = var.ec2_instance_type
  key_name       = "terraform-key"
  user_data = file("apache-install.sh")
  vpc_security_group_ids = [aws_security_group
  tags = {
    "Name" = "amz-linux-vm"
  }
}
```

## Meta-Arguments for Datasources

Data resources support the provider meta-argument as defined for managed resources, with the same syntax and behavior.

Data resources do not currently have any customization settings available for their lifecycle, but the lifecycle nested block is reserved in case any are added in future versions.

Data resources support count and for_each meta-arguments as defined for managed resources, with the same syntax and behavior.
Each instance will separately read from its data source with its own variant of the constraint arguments, producing an indexed result.

Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Variables – Output Values

Output values are like the return values of a Terraform module and have several uses

Terraform Variables Outputs

1

A root module can use outputs to print certain values in the CLI output after running terraform apply.

2

A child module can use outputs to expose a subset of its resource attributes to a parent module.

Advanced

When using remote state, root module outputs can be accessed by other configurations via a terraform_remote_state d ata source.

3

Author: Nho Luong
Skill: DevOps Engineer Lead

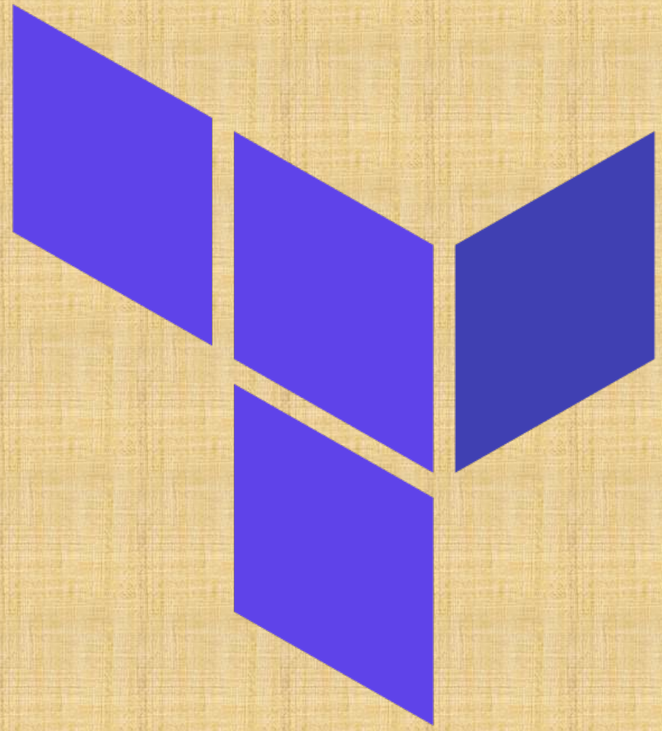
# Terraform

# For Loops

# Lists & Maps

# Meta-Argument

# Count & for_each



Author: Nho Luong
Skill: DevOps Engineer Lead

## Section-1

- Meta-Argument: count
- Variables: Lists & Maps
- For Loop with Lists
- For Loop with Maps
- For Loops with Advanced Maps
- Legacy Splat Operator (.*.)
- Latest Splat Operator [*]

## Section-2

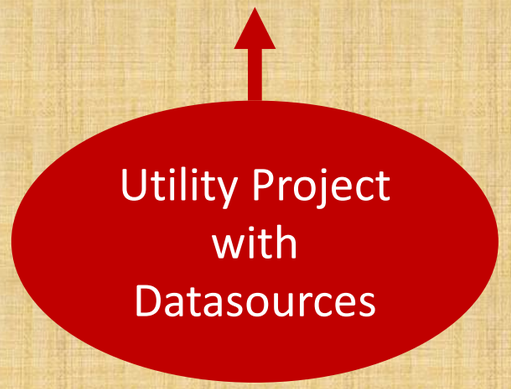
- Meta-Argument: for_each
- Function: toset
- Function: tomap
- Datasource: aws_availability_zones

## Section-4

- Fix issues in Section-2 with section-3
- Final Output

## Section-3

- Datasource: aws_ec2_instance_type_offerings
- Datasource: aws_availability_zones
- For Loop with Maps
- For Loop with if
- Function: keys




Author: Nho Luong
Skill: DevOps Engineer Lead

Amazon Virtual Private Cloud (Amazon VPC)

Internet gateway

NAT gateway

Elastic IP address


Route table

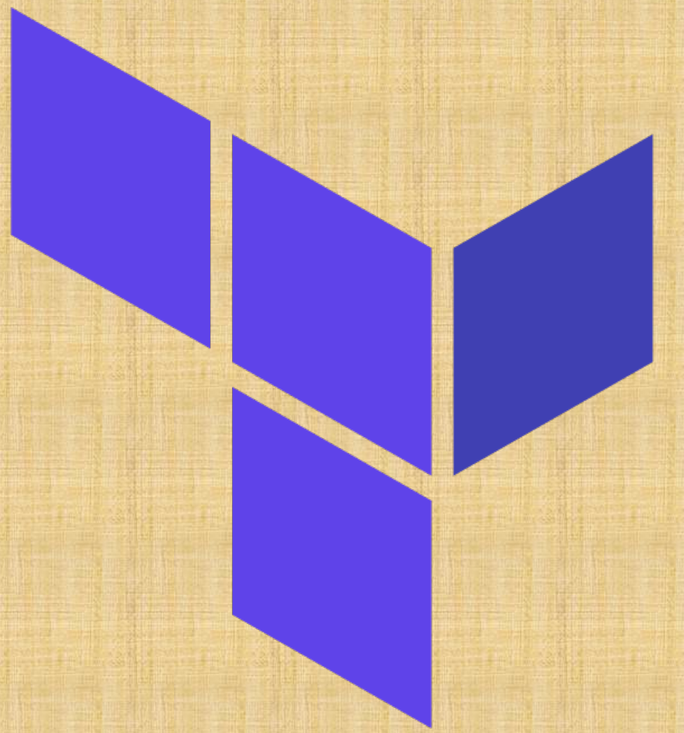
Public Subnet

Private Subnet

# AWS VPC

# 3-Tier

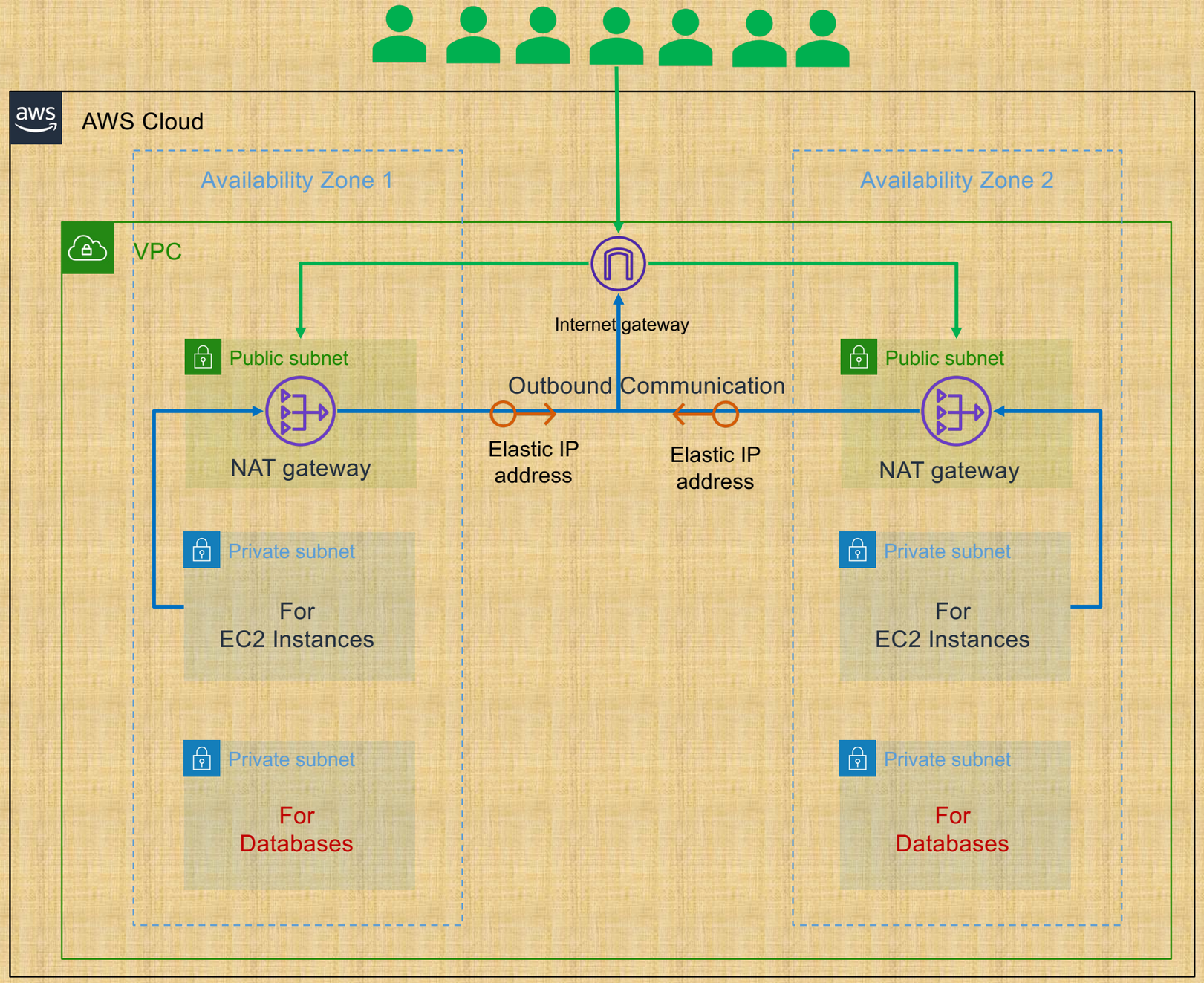
# Web, App and DB



Author: Nho Luong
Skill: DevOps Engineer Lead

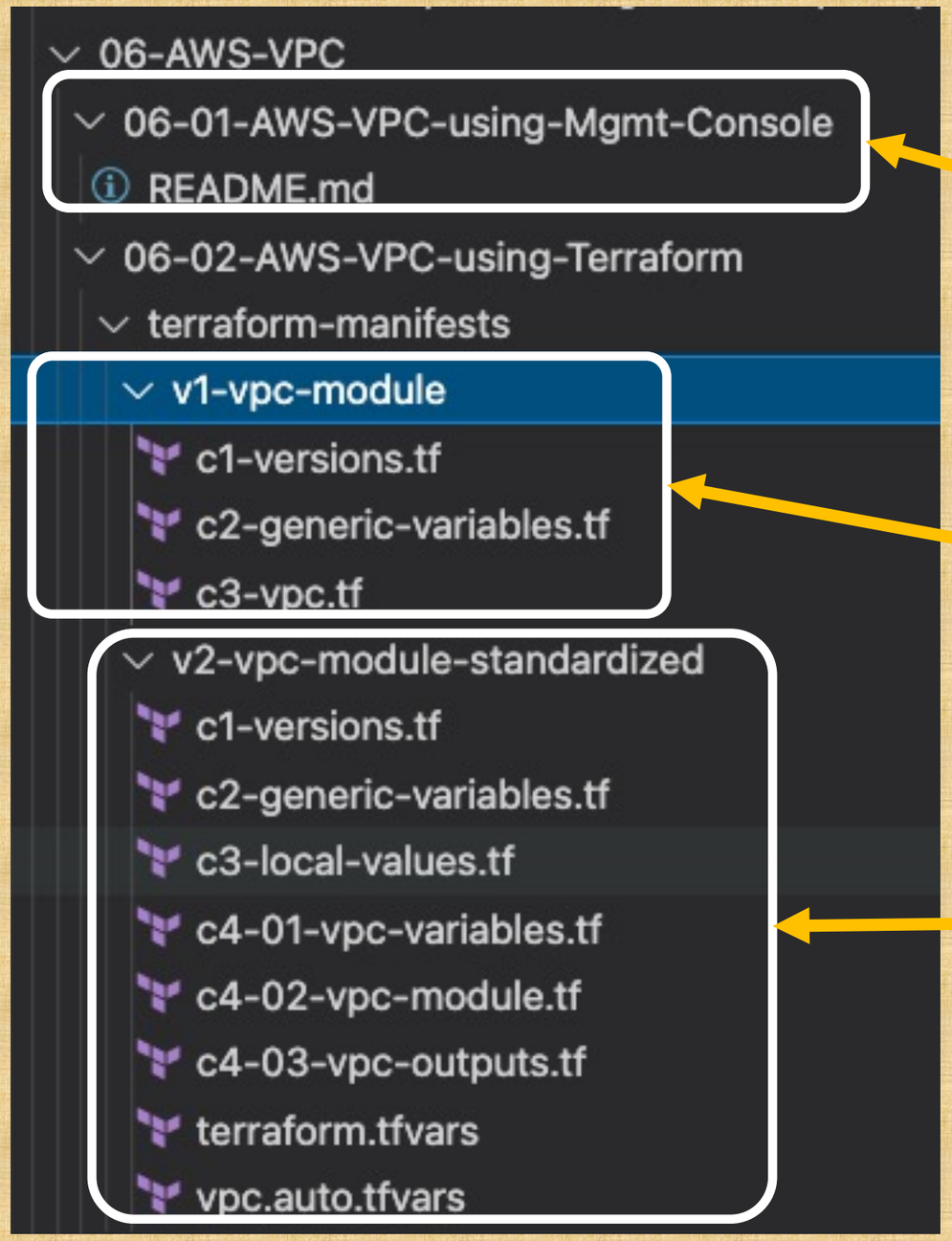
# AWS VPC 3-Tier Architecture

- Build manually using AWS Mgmt Console
- Build same using Terraform
- Terraform Modules
- Terraform Local Values
- Terraform Variables – terraform.tfvars
- Terraform Variables – vpc.auto.tfvars
- Terraform Version Constraints
- Terraform Code Organizing – Production Grade style




Author: Nho Luong
Skill: DevOps Engineer Lead

# What are we going to Learn ?




Build VPC manually using AWS Management Console

Build VPC using Terraform Modules

Standardize the Terraform Code at Production Grade

Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Modules

Modules are containers for multiple resources that are used together. A module consists of a collection of .tf files kept together in a directory.

Modules are the main way to package and reuse resource configurations with Terraform.

A module that has been called by another module is often referred to as a child module.

Every Terraform configuration has at least one module, known as its root module, which consists of the resources defined in the .tf files in the main working directory

Child modules can be called multiple times within the same configuration, and multiple configurations can use the same child module.

A Terraform module (usually the root module of a configuration) can call other modules to include their resources into the configuration.

In addition to modules from the local filesystem, Terraform can load modules from a public or private registry.

This makes it possible to publish modules for others to use, and to use modules that others have published.

Author: Nho Luong
Skill: DevOps Engineer Lead

Amazon Virtual Private Cloud (Amazon VPC)

Internet gateway

NAT gateway

Elastic IP address


Route table

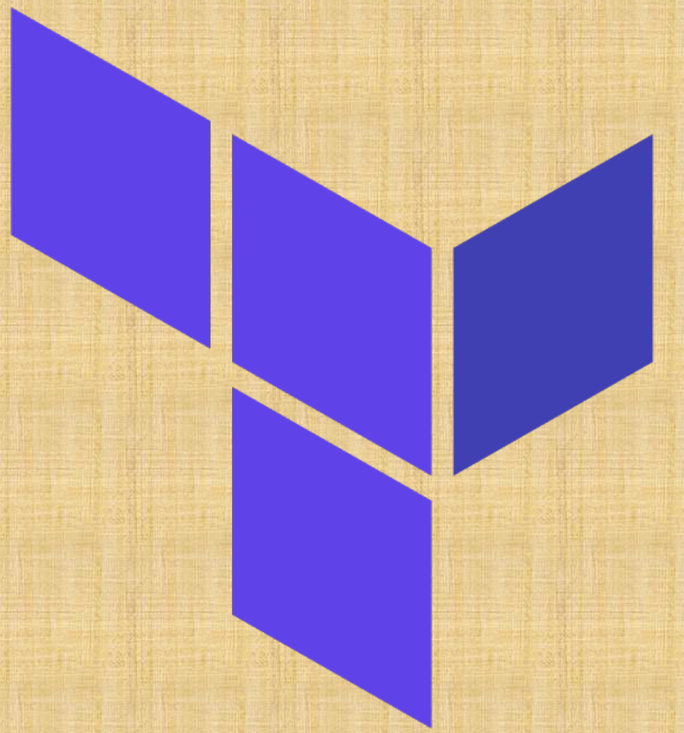
Public Subnet

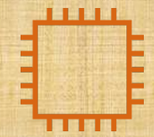
Private Subnet

# AWS

## VPC with Web, App and DB Tiers

## EC2 Instances, Bastion Host and Security Groups





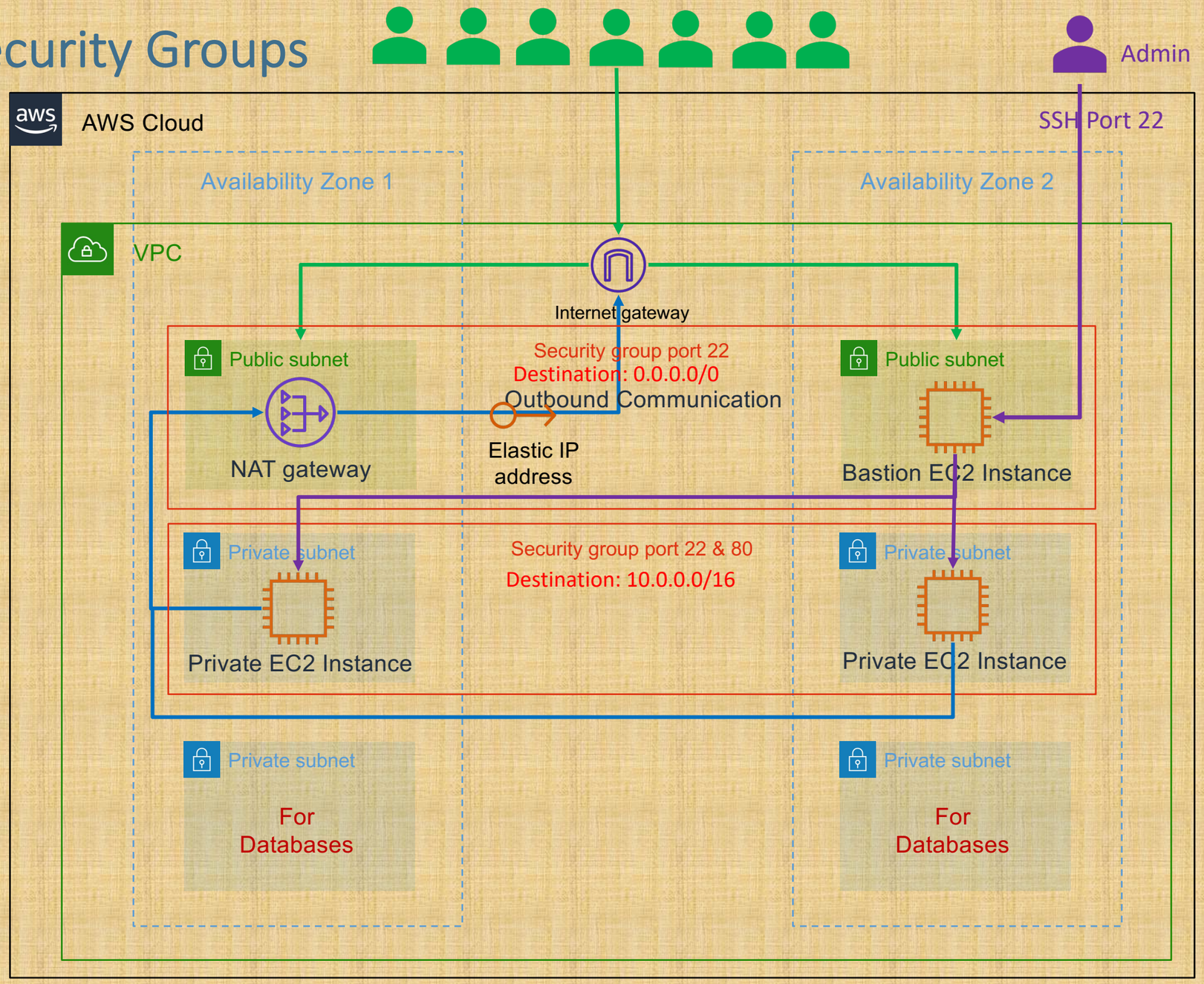
EC2 Instance


Author: Nho Luong
Skill: DevOps Engineer Lead

# AWS VPC + EC2 Instance + Security Groups

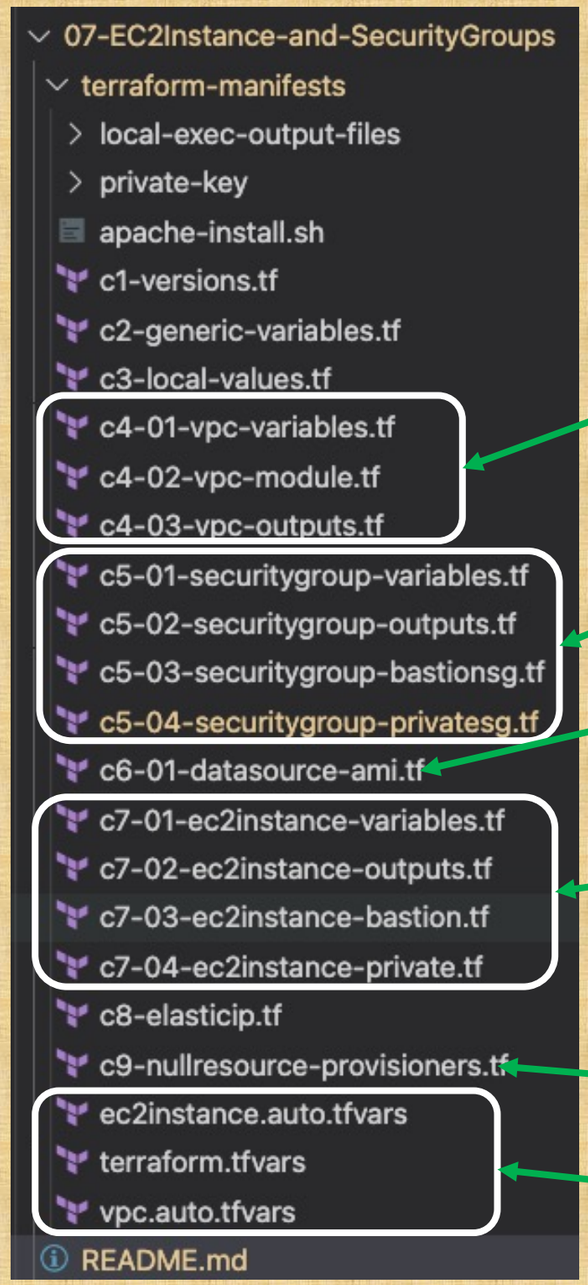
## Terraform & AWS Concepts

- Terraform Module: VPC
- Terraform Module: Security Group
- Terraform Module: AWS EC2 Instance
- Meta-Argument: depends_on
- Terraform null_resource
- Terraform File Provisioner
- Terraform Remote-exec Provisioner
- Terraform Local-exec Provisioner




Author: Nho Luong
Skill: DevOps Engineer Lead

# What are we going to learn ?




Create AWS VPC using Terraform Modules

Create AWS Security Groups using Terraform Modules

Create AWS AMI Datasource to get latest AMI ID dynamically

Create AWS EC2 Instance using Terraform Modules

Null Resource

- File Provisioner
- Remote-exec Provisioner
- Local-exec Provisioner

Variables

Author: Nho Luong
Skill: DevOps Engineer Lead

Amazon Virtual Private Cloud
(Amazon VPC)

Internet gateway

NAT gateway

Elastic IP
address


Route table

Public Subnet

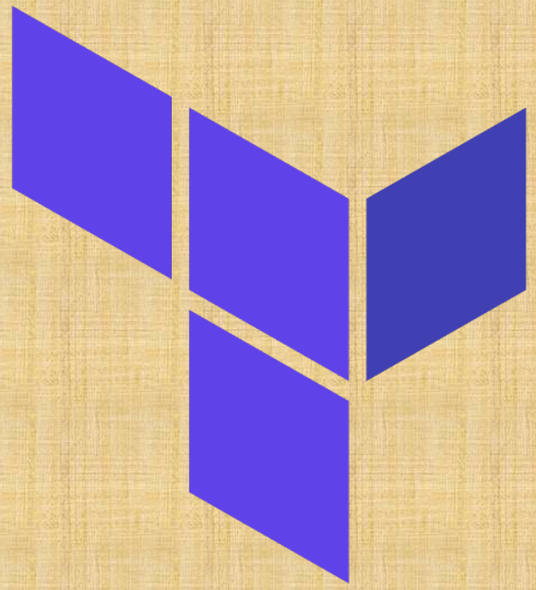
Private Subnet

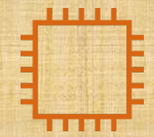
# AWS

## VPC with Web, App and DB Tiers

## EC2 Instances, Bastion Host and Security Groups

## Classic Load Balancer





EC2 Instance


Security Groups

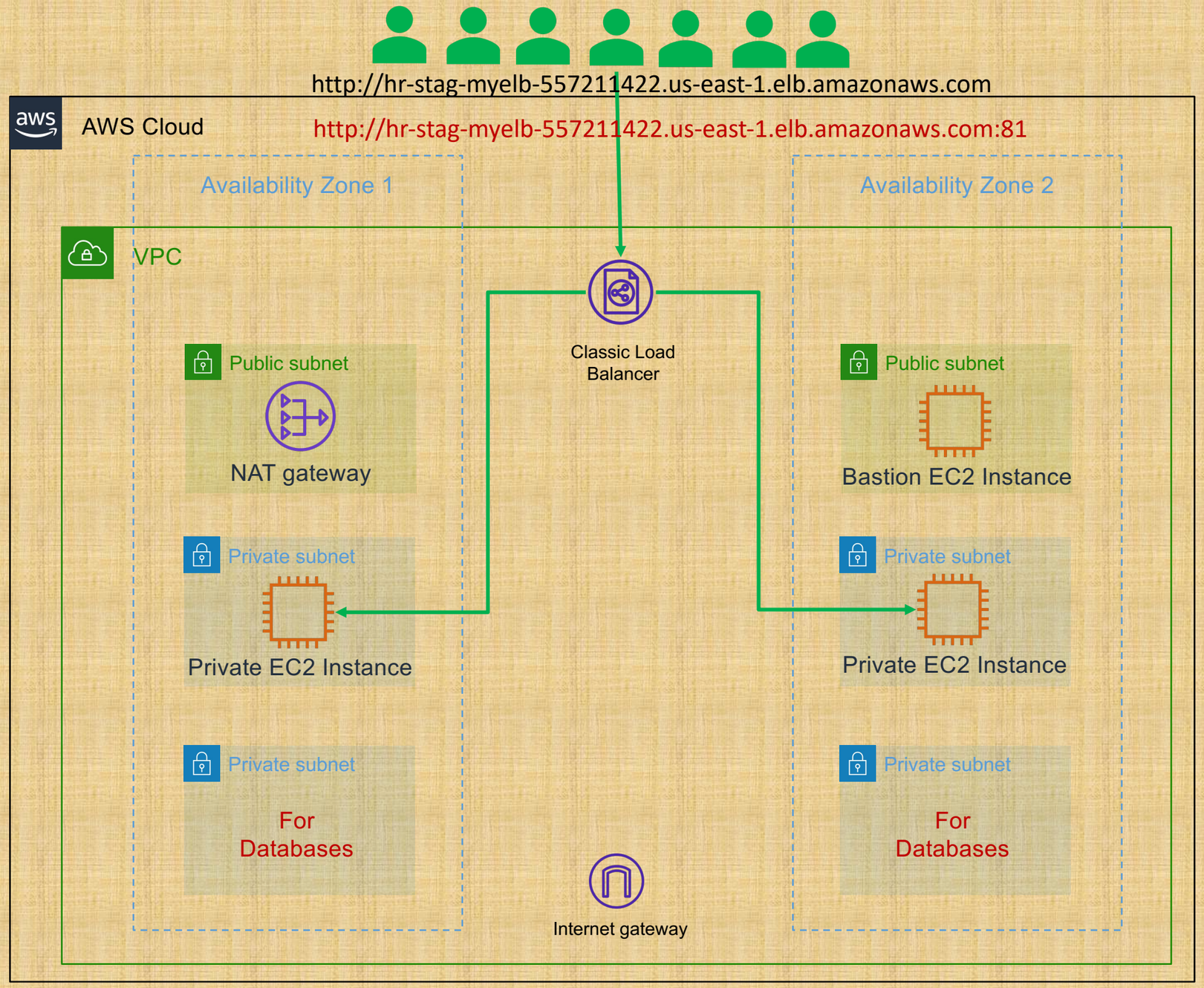
Classic Load
Balancer

Author: Nho Luong
Skill: DevOps Engineer Lead

# AWS VPC + EC2 Instance + Security Groups + AWS ELB Classic Load Balancer

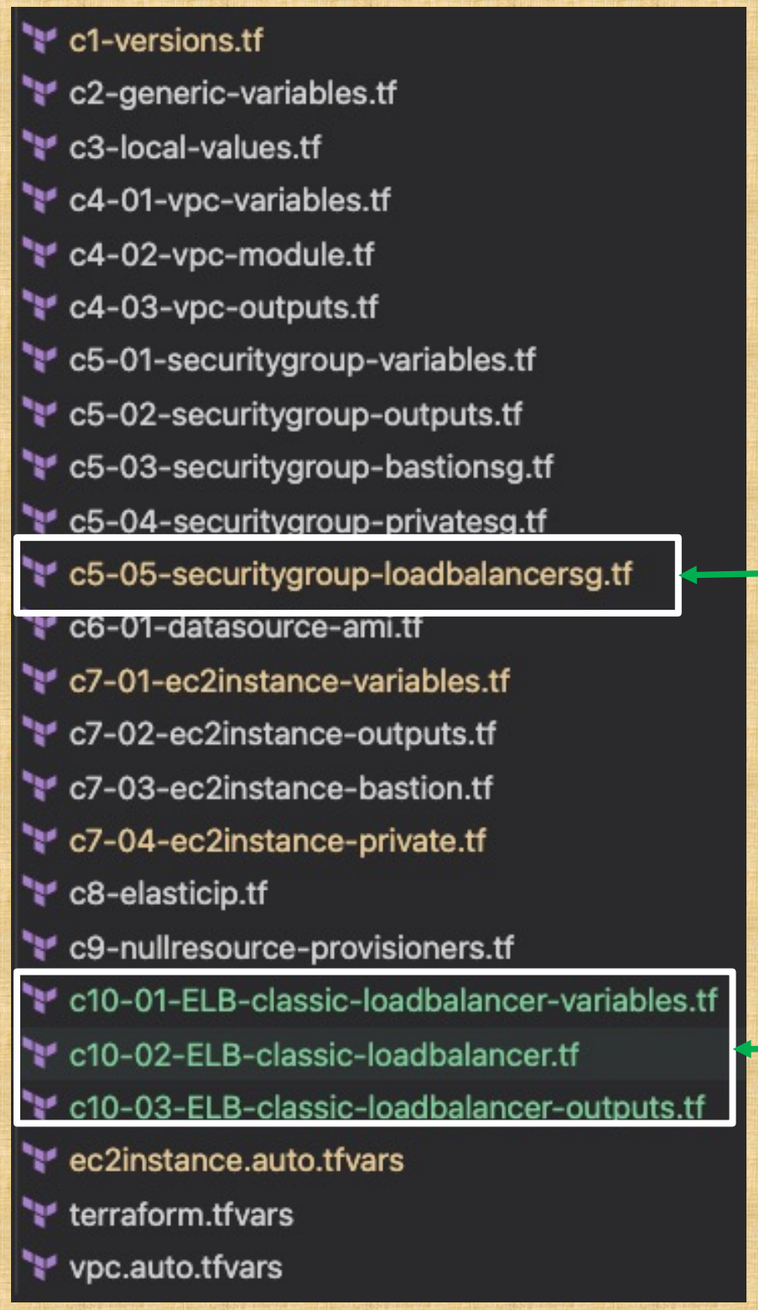
**Terraform & AWS Concepts**

- Terraform Module: ELB Classic LB
- Terraform Module: Security Group
- Add Custom SG Rule for Inbound Port 81





Author: Nho Luong
Skill: DevOps Engineer Lead




# What are we going to learn ?

Create Security Group to allow access from Internet to Load Balancer using AWS Security Group Terraform Module

Create AWS ELB Classic Load Balancer using Terraform Modules

Author: Nho Luong
Skill: DevOps Engineer Lead

Amazon Virtual Private Cloud (Amazon VPC)

Internet gateway

NAT gateway

Elastic IP address


Route table

Public Subnet

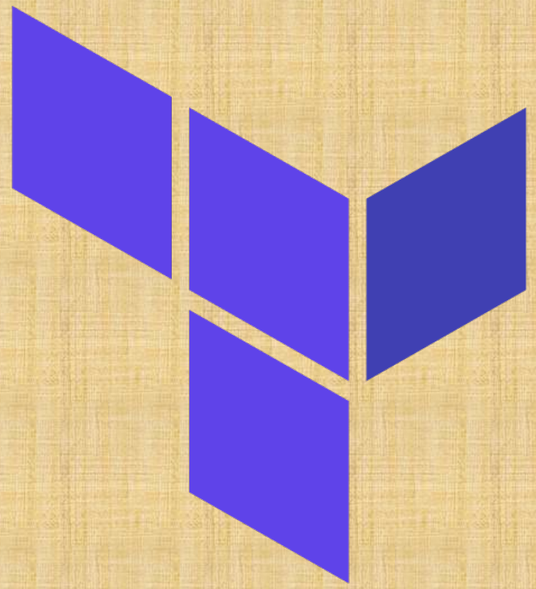
Private Subnet

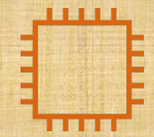
# AWS

## VPC with Web, App and DB Tiers

## EC2 Instances, Bastion Host and Security Groups

## Application Load Balancer - Basic





EC2 Instance


Security Groups

Classic Load Balancer

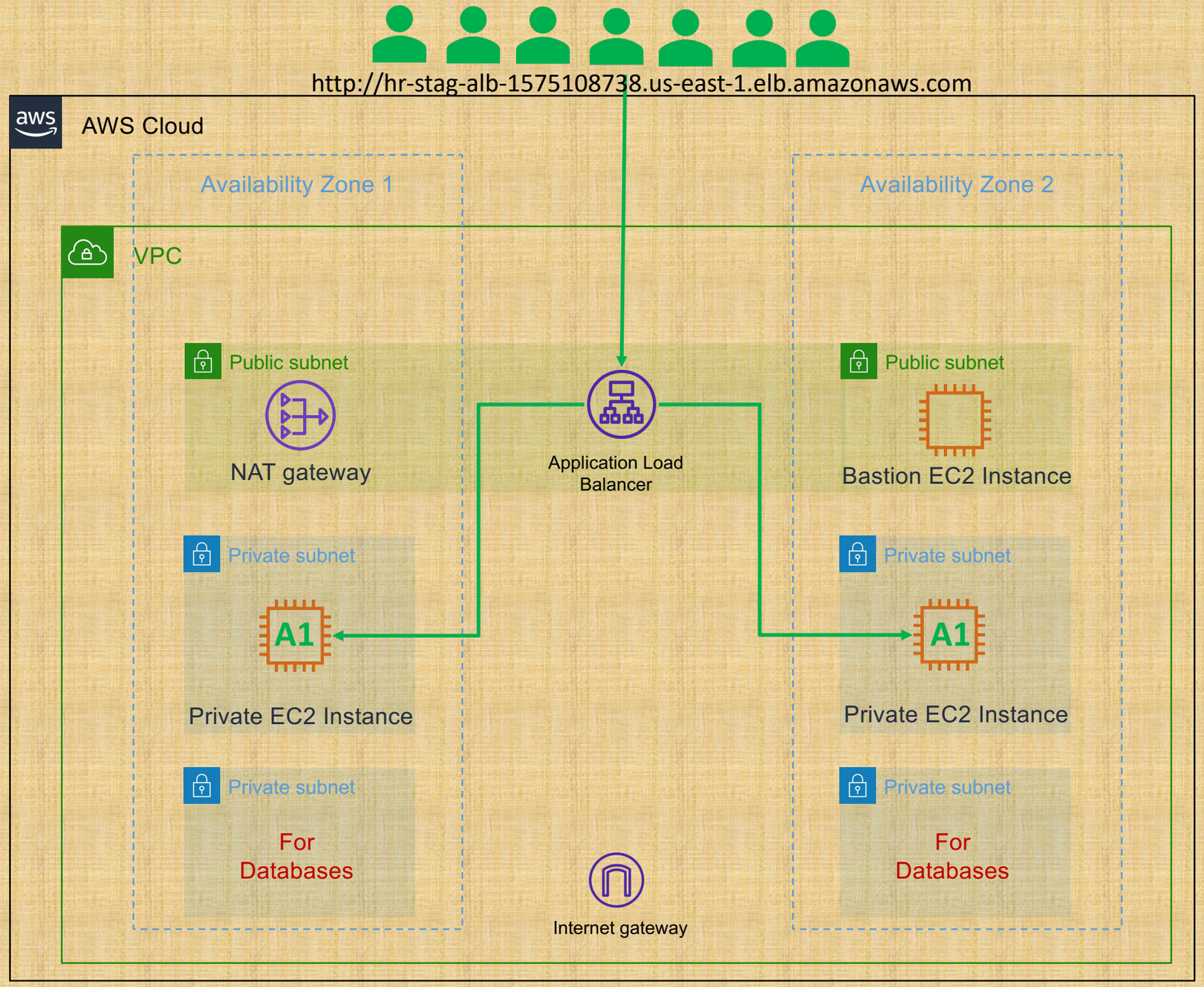
Application Load Balancer

Author: Nho Luong
Skill: DevOps Engineer Lead

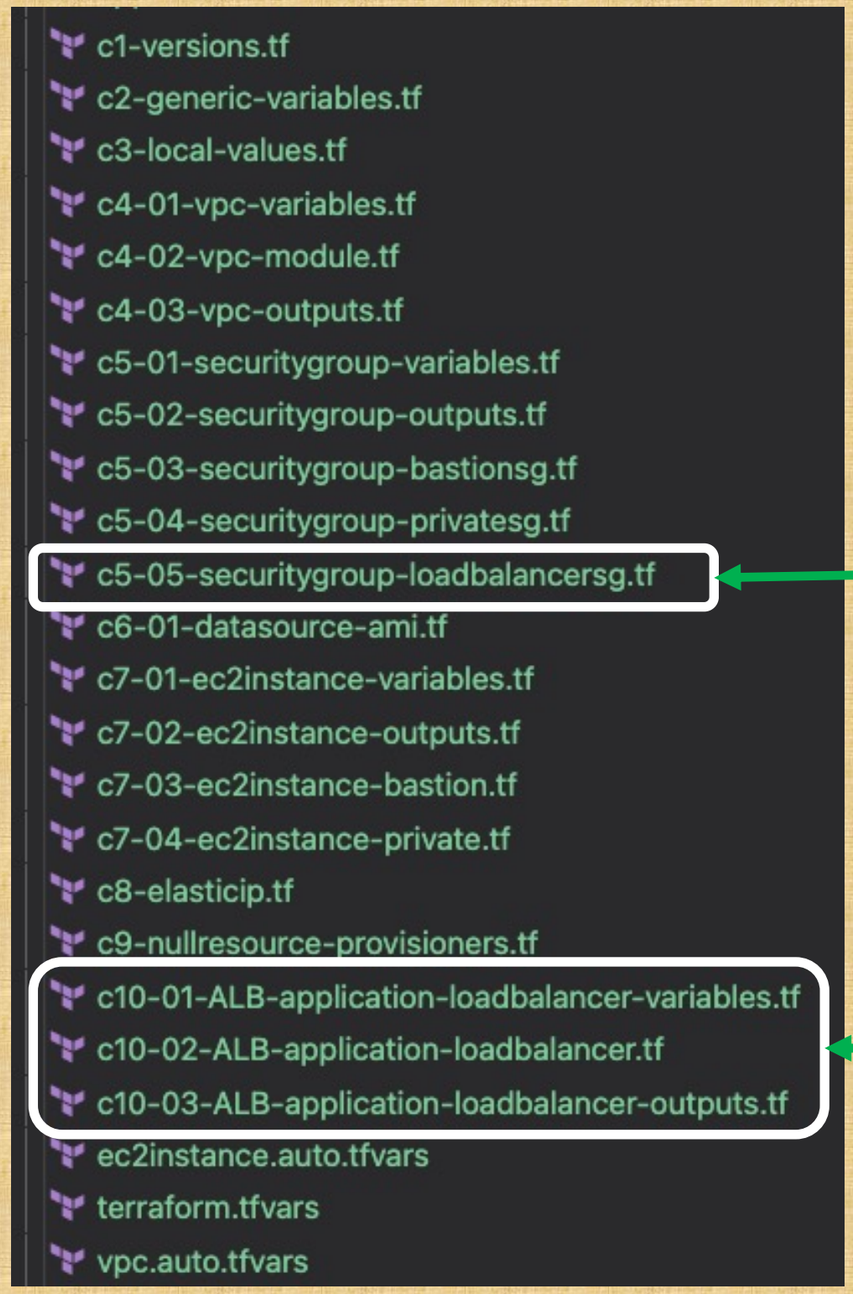
# AWS VPC + EC2 Instance + Security Groups + AWS ALB Application Load Balancer

Terraform & AWS Concepts

Terraform Module: AWS ALB




Author: Nho Luong
Skill: DevOps Engineer Lead




# What are we going to learn ?

Create Security Group to allow access from Internet to Load Balancer using AWS Security Group Terraform Module

Create AWS ALB Application Load Balancer using Terraform Modules

Author: Nho Luong
Skill: DevOps Engineer Lead

Amazon Virtual Private Cloud (Amazon VPC)

Internet gateway

NAT gateway

Elastic IP address


Route table

Public Subnet

Private Subnet

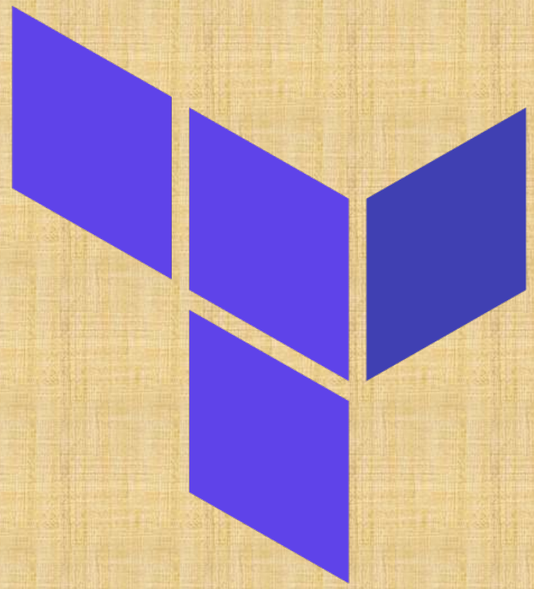
# AWS

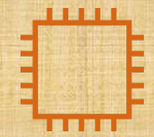
## VPC with Web, App and DB Tiers

## EC2 Instances, Bastion Host and Security Groups

## Application Load Balancer

## Context Path Based Routing





EC2 Instance


Security Groups

Classic Load Balancer

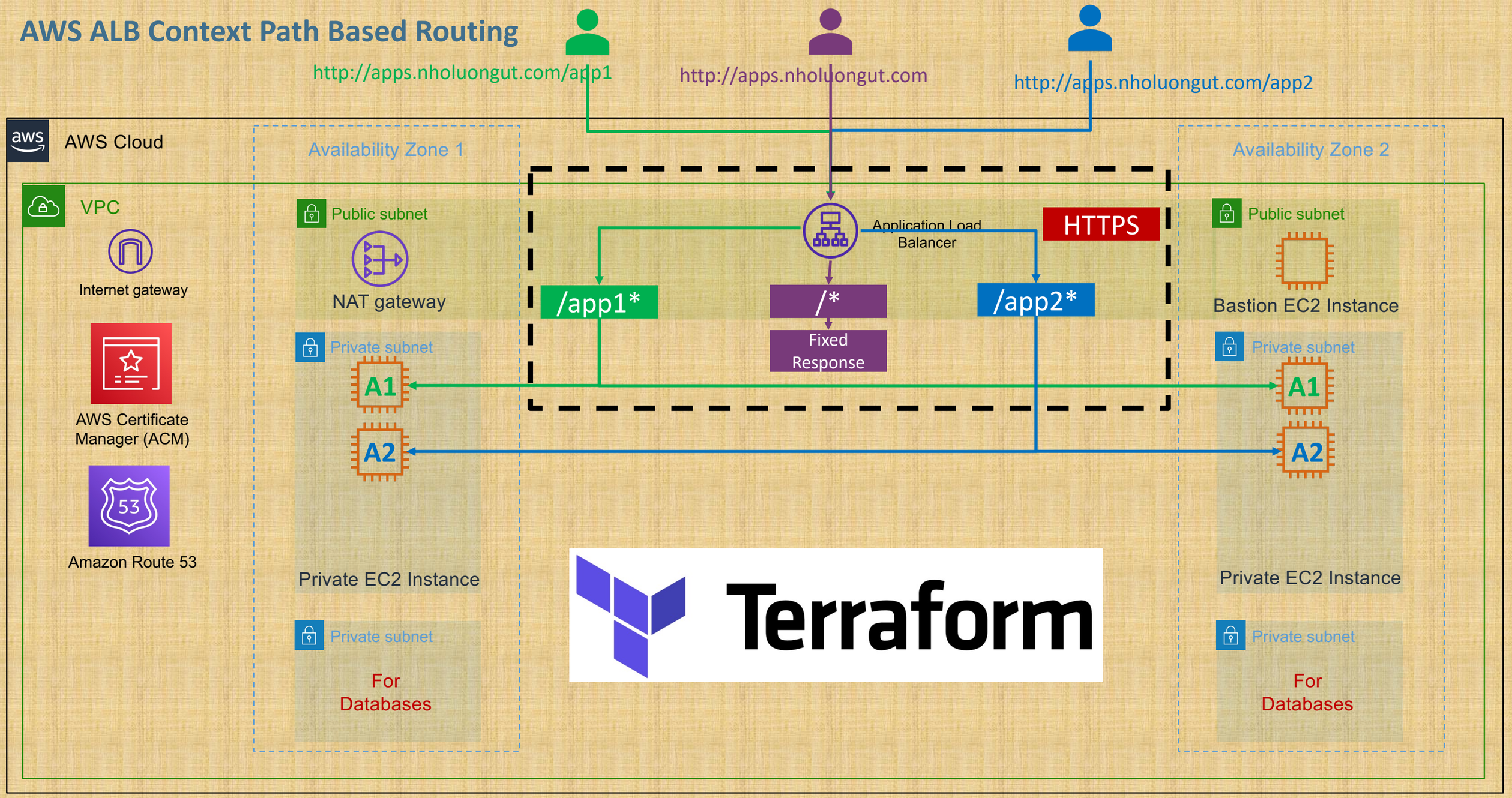
Application Load Balancer

AWS Certificate Manager (ACM)


Amazon Route 53

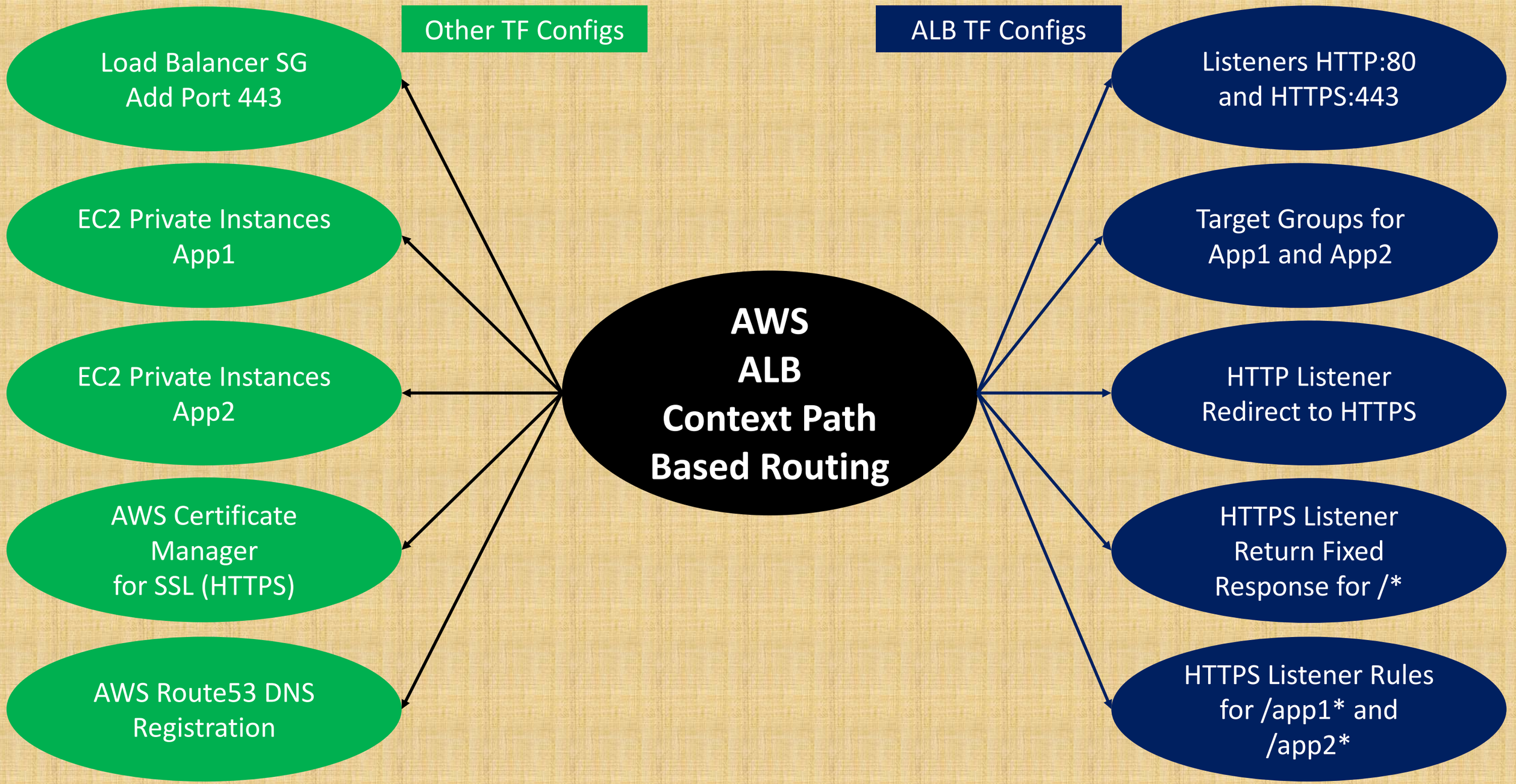
Author: Nho Luong
Skill: DevOps Engineer Lead

# AWS ALB Context Path Based Routing



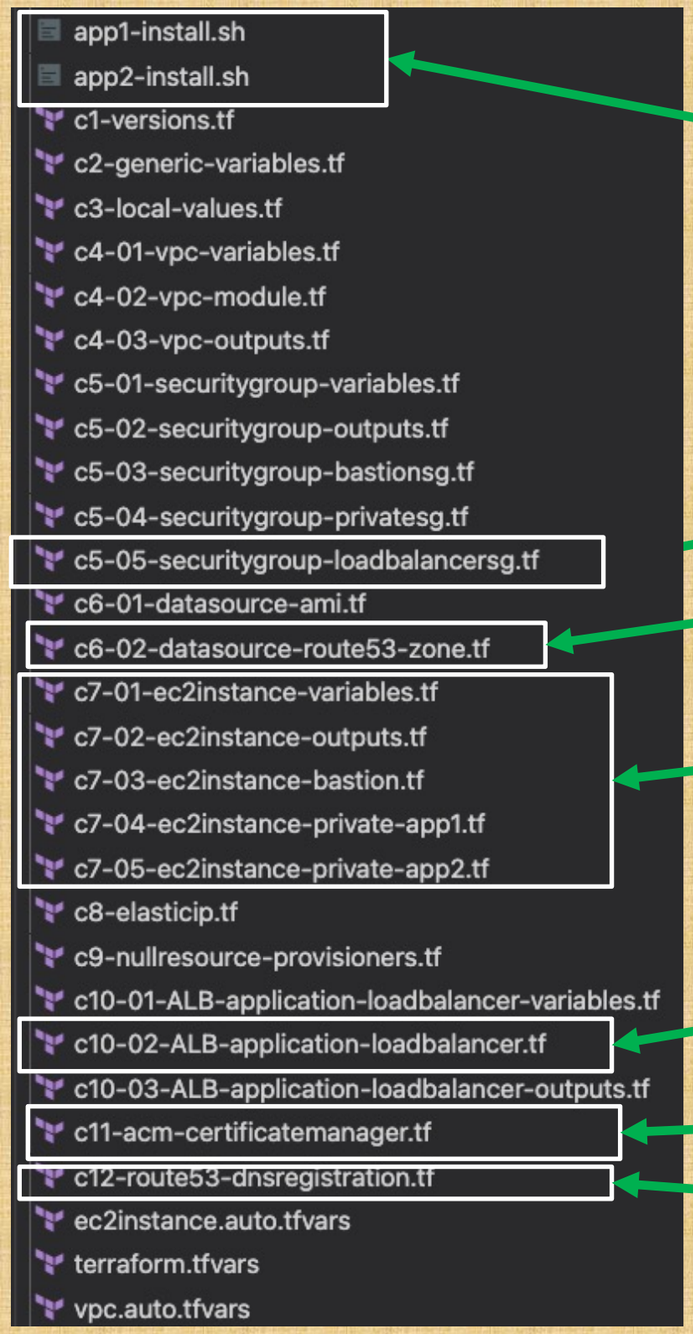


Author: Nho Luong
Skill: DevOps Engineer Lead




Author: Nho Luong
Skill: DevOps Engineer Lead

# What are we going to learn ?




- App1 and App2 Userdata
- Allow HTTPS 443 in Load Balancer SG
- Route53 Hosted Zone Datasource
- Define App1 and App2 EC2 Instances and also configure their equivalent Outputs
- Define ALB HTTP and HTTPS Listeners, App1 and 2 Target Groups, Listener Rules, Redirects etc
- Define SSL Certificate using AWS Certificate Manager Service
- Create Route53 DNS Record

Author: Nho Luong
Skill: DevOps Engineer Lead

Amazon Virtual Private Cloud (Amazon VPC)

Internet gateway

NAT gateway

Elastic IP address


Route table

Public Subnet

Private Subnet

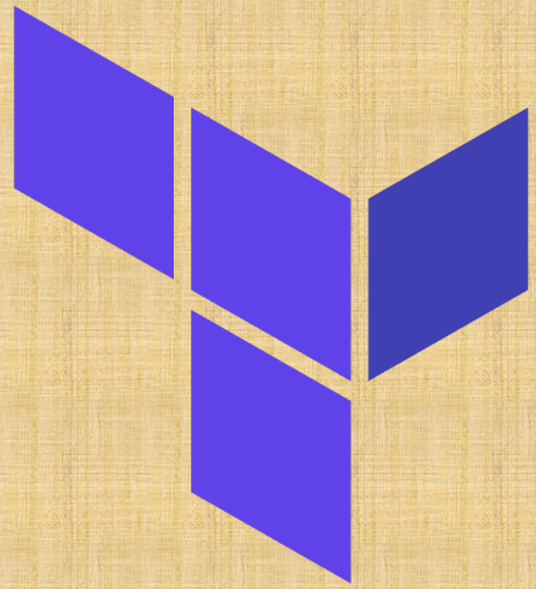
# AWS

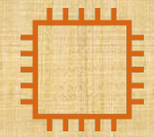
## VPC with Web, App and DB Tiers

## EC2 Instances, Bastion Host and Security Groups

## Application Load Balancer

## Host Header Based Routing





EC2 Instance


Security Groups

Classic Load Balancer

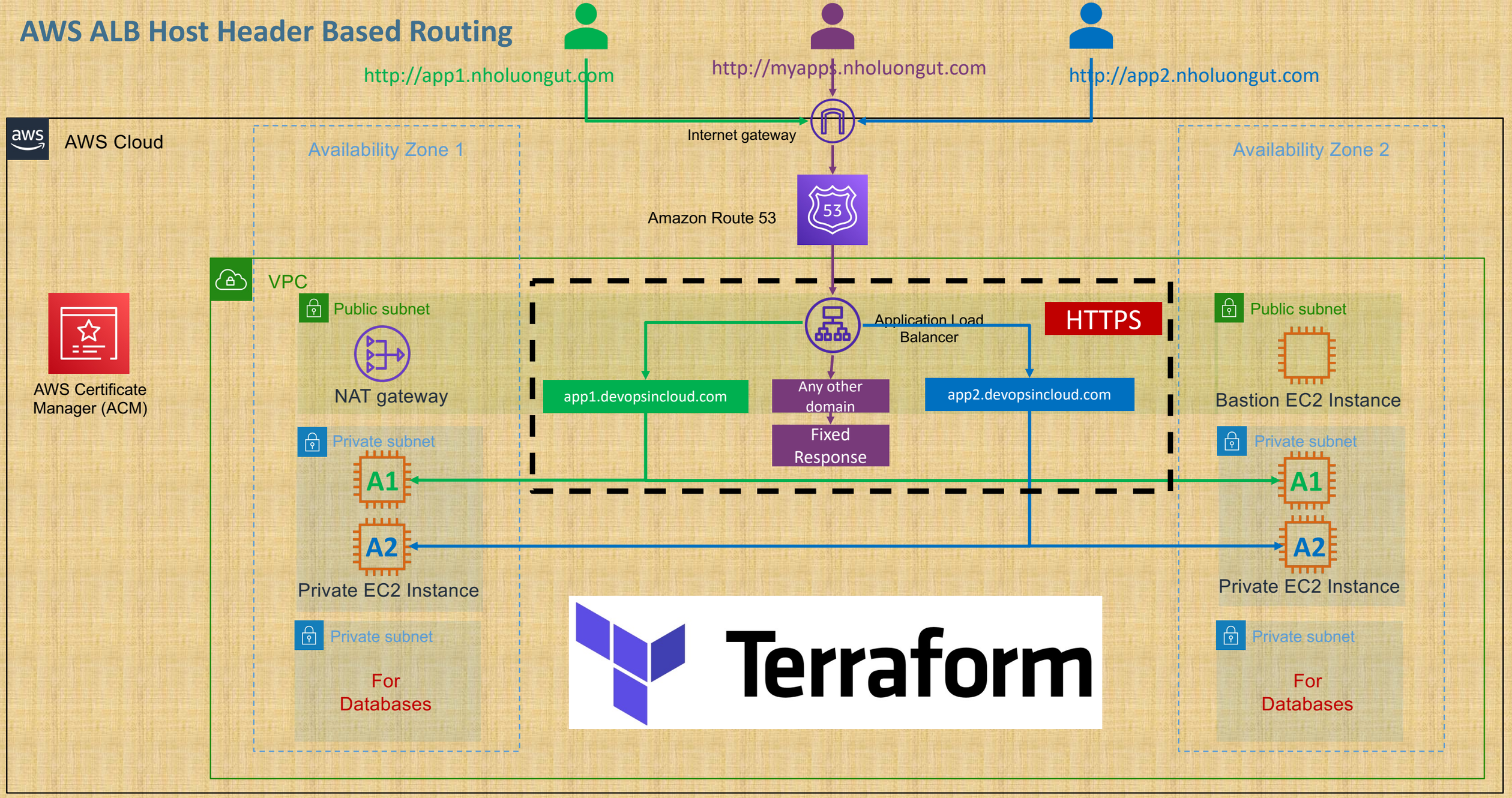
Application Load Balancer

AWS Certificate Manager (ACM)


Amazon Route 53

Author: Nho Luong
Skill: DevOps Engineer Lead

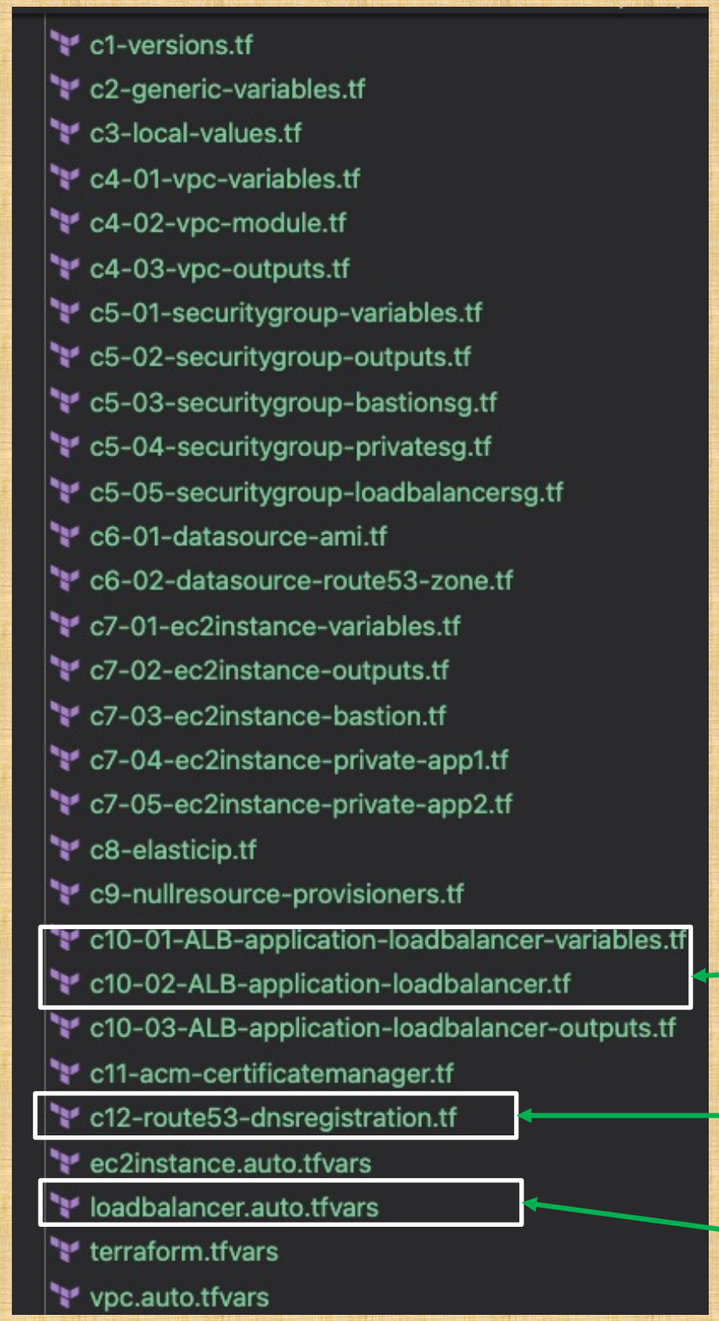
# AWS ALB Host Header Based Routing





Author: Nho Luong
Skill: DevOps Engineer Lead

# What are we going to learn ?




Changes related to ALB Variables and ALB Host Header settings

Changes related to Multiple DNS Names in Route53

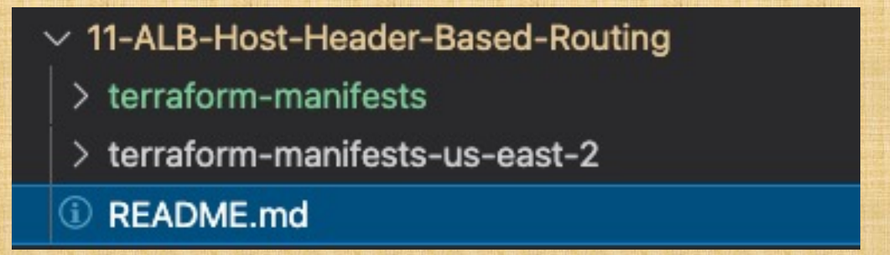
Multiple DNS Names reference in ALB and Route53 DNS – So created variables for DNS Names

Author: Nho Luong
Skill: DevOps Engineer Lead

# AWS ACM Certificate Limit Issue

```
aws_route53_record.app1_dns: Creation complete after 5m20s [id=Z10098343FYUXIH4G2MGF_a
pp18.devopsincloud.com_A]

| Error: Error requesting certificate: LimitExceededException: Error: you have reached
 your limit of 20 certificates in the last year.
|
|   on .terraform/modules/acm/main.tf line 11, in resource "aws_acm_certificate" "this
":
|   11: resource "aws_acm_certificate" "this" {
|
|
```




Author: Nho Luong
Skill: DevOps Engineer Lead

Amazon Virtual Private Cloud (Amazon VPC)

Internet gateway

NAT gateway

Elastic IP address


Route table

Public Subnet

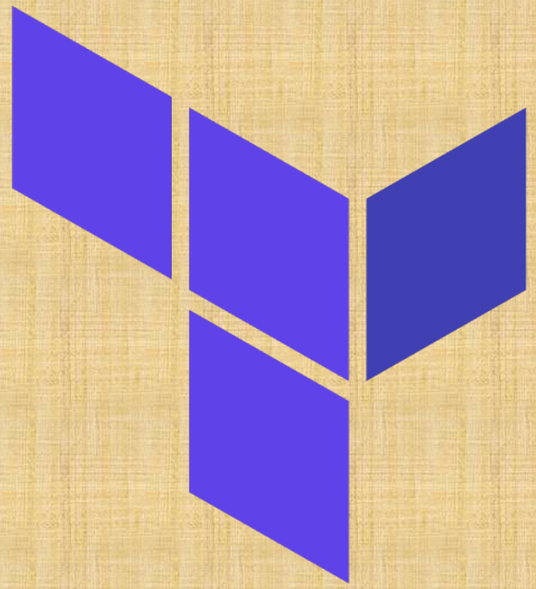
Private Subnet

# AWS

## VPC with Web, App and DB Tiers

## EC2 Instances, Bastion Host and Security Groups

## Application Load Balancer



AWS Certificate Manager (ACM)


Amazon Route 53

### Rule-1: Custom HTTP Header Routing to App1 TG

### Rule-2: Custom HTTP Header Routing to App2 TG

### Rule-3: Query String with 302 Redirect

### Rule-4: Host Header with 302 Redirect

EC2 Instance

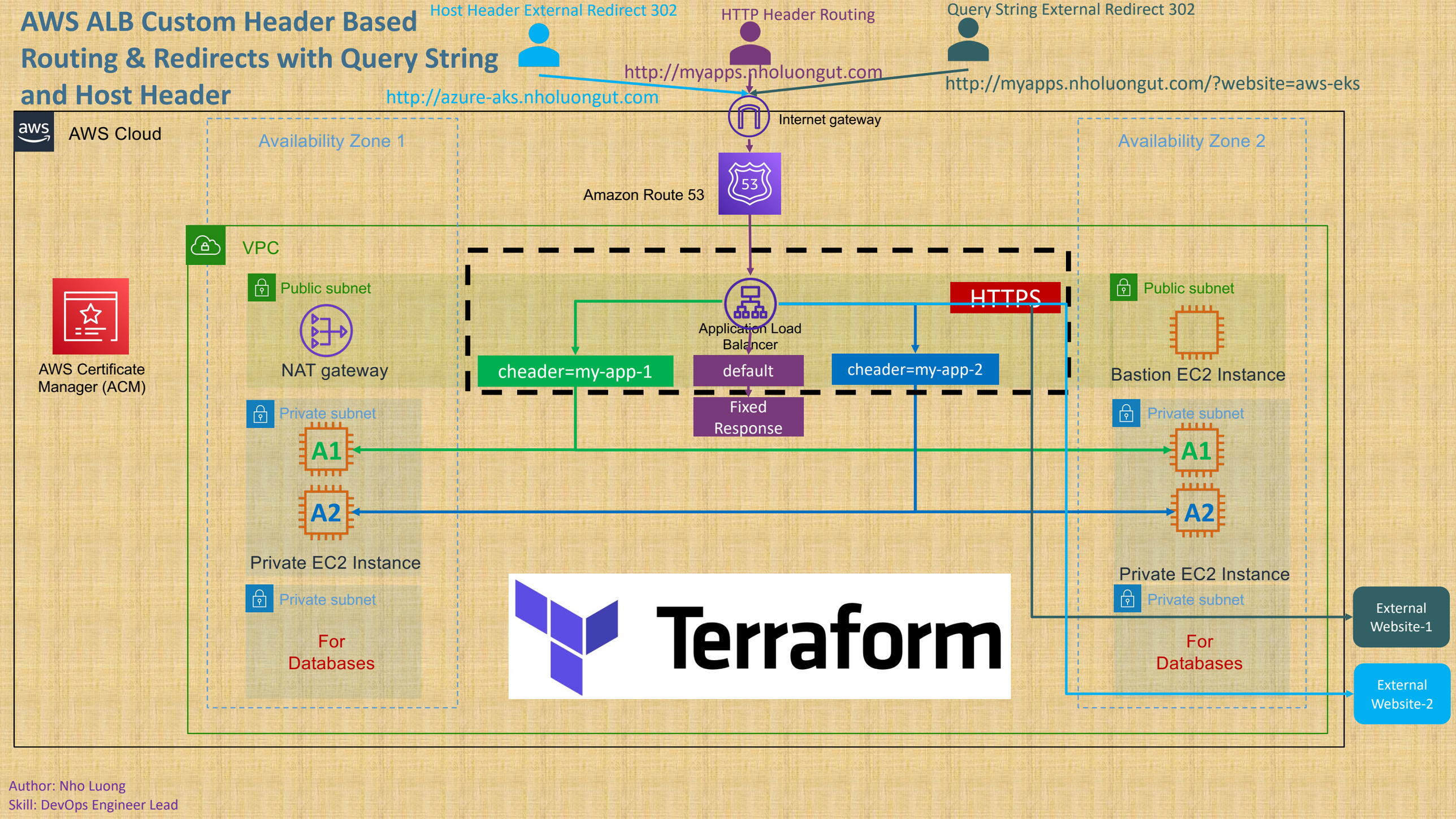

Security Groups

Classic Load Balancer

Application Load Balancer

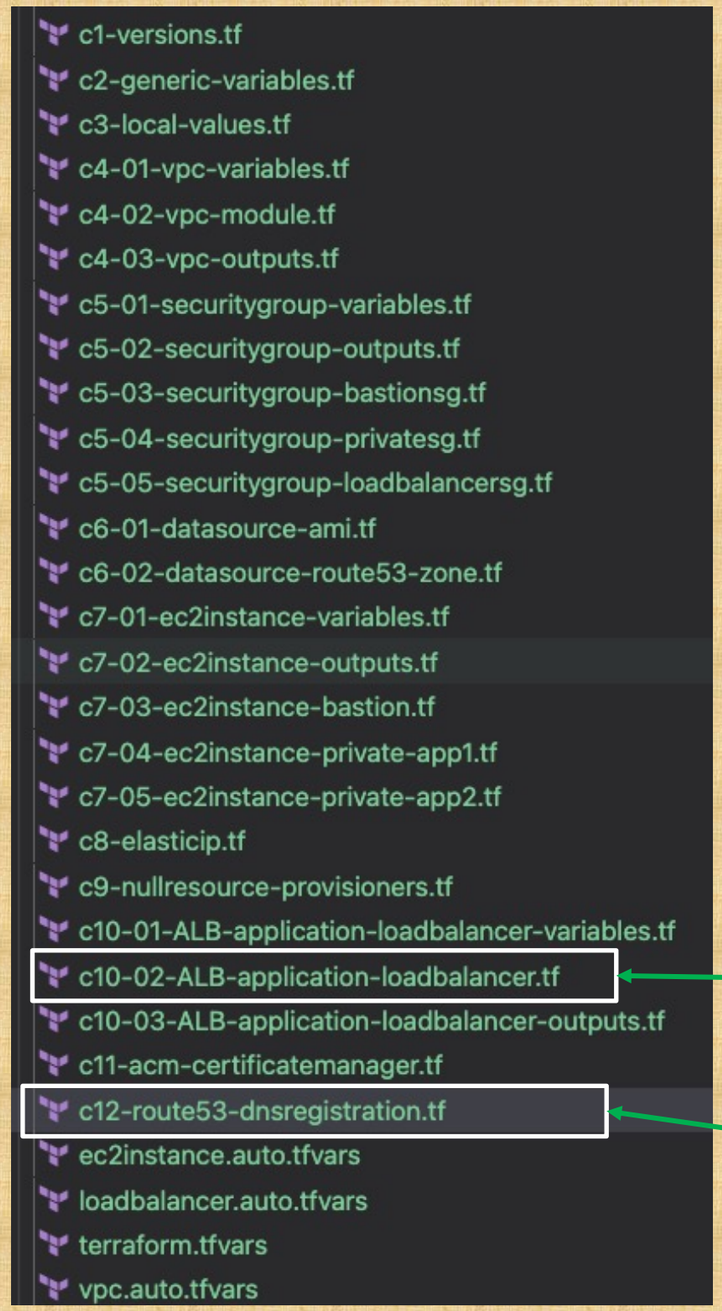
Author: Nho Luong
Skill: DevOps Engineer Lead

# AWS ALB Custom Header Based Routing & Redirects with Query String and Host Header





Author: Nho Luong
Skill: DevOps Engineer Lead




# What are we going to learn ?

Added new Rules 1 to 4 outlining Custom HTTP Header and Redirects with Query String and Host Header

Added DNS Records as per need for this usecase

Author: Nho Luong
Skill: DevOps Engineer Lead

Amazon Virtual Private Cloud (Amazon VPC)

Internet gateway

NAT gateway

Elastic IP address


Route table

Public Subnet

Private Subnet

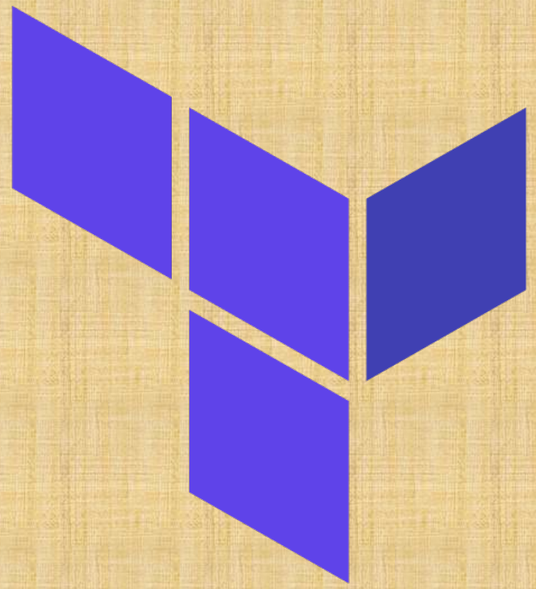
# AWS

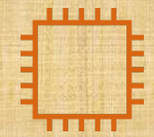
## VPC with Web, App and DB Tiers

## EC2 Instances, Bastion Host and Security Groups

## Application Load Balancer

## DNS to DB





EC2 Instance


Security Groups

Classic Load Balancer

Application Load Balancer

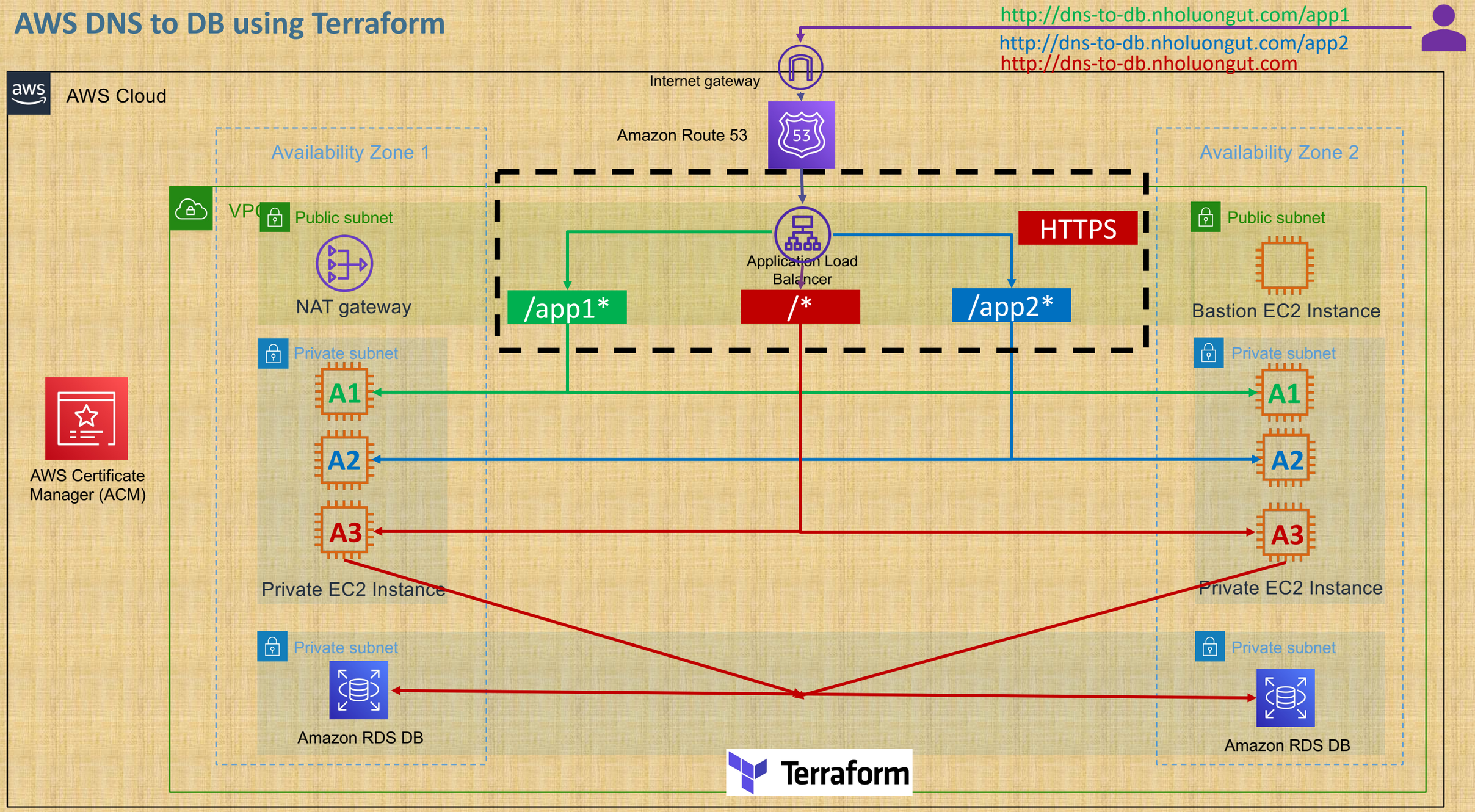
AWS Certificate Manager (ACM)


Amazon Route 53

Amazon RDS DB

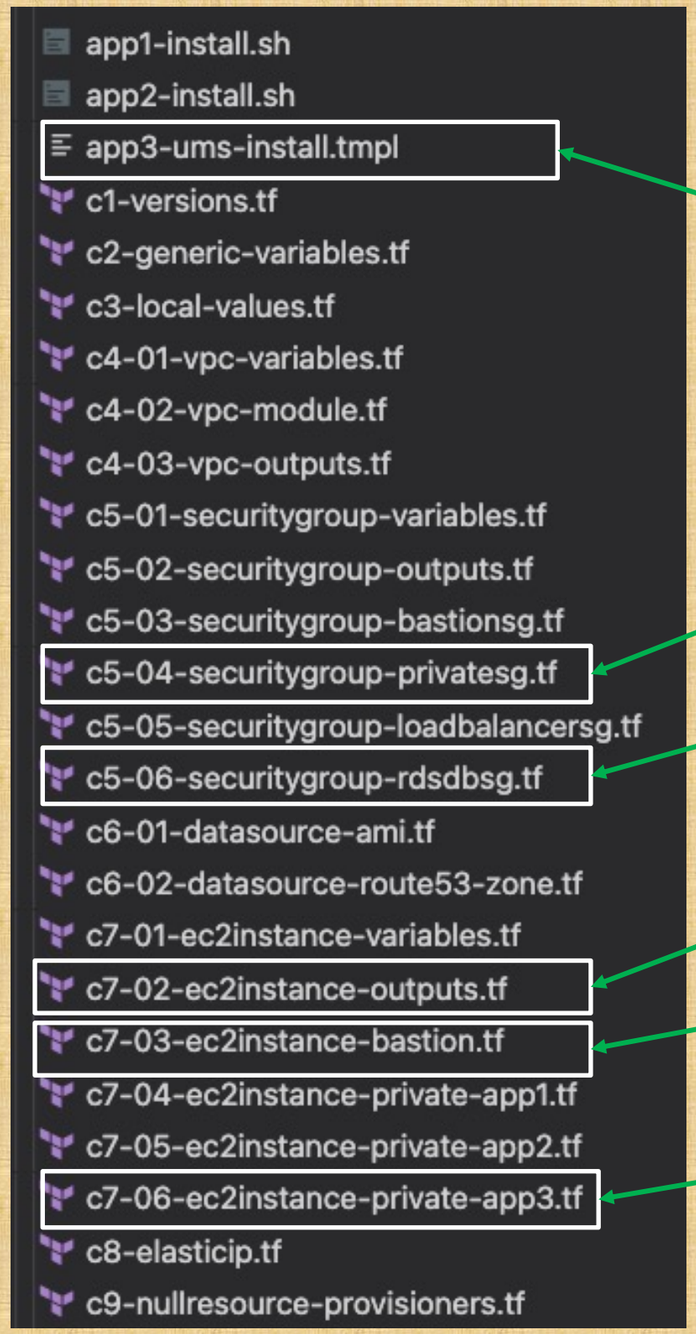
Author: Nho Luong
Skill: DevOps Engineer Lead

# AWS DNS to DB using Terraform





Author: Nho Luong
Skill: DevOps Engineer Lead




# What are we going to learn ?

App3 User Management Application (UMS) which needs Database Userdata for App3 EC2 Instances

Add HTTP 8080 for Private Instances Security Group which is an UMS App Listen Port

RDS Database Security Group

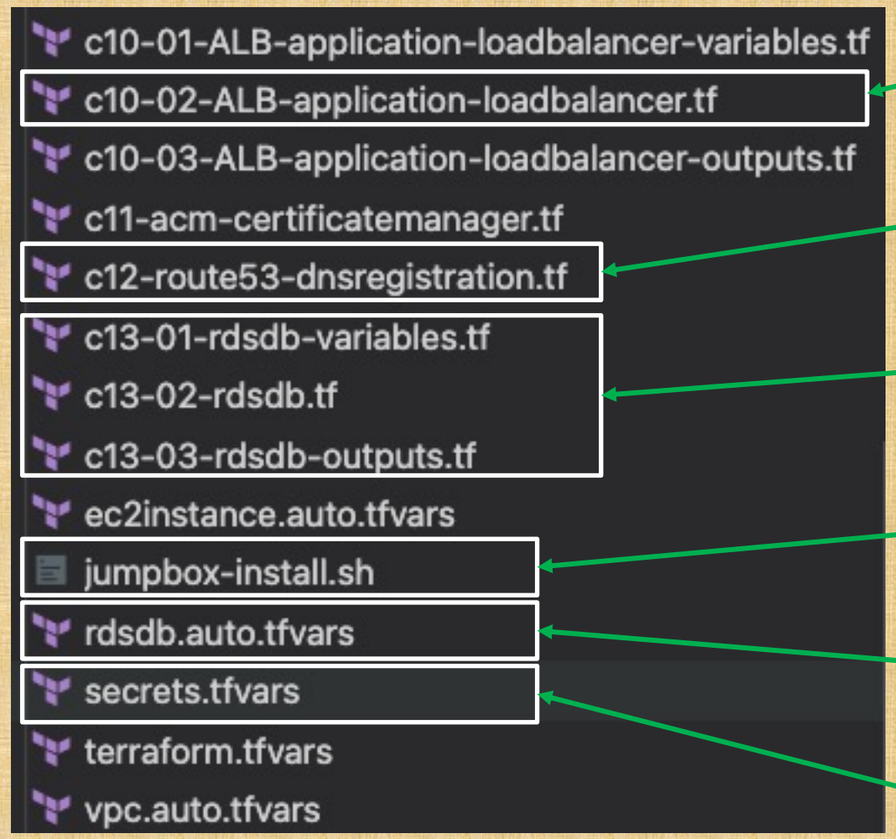
Define Outputs for App3 EC2 Instances

Update Bastion EC2 Instance to have Userdata containing automatic install of MySQL Client to check RDS DB Connection

Create Private EC2 Instances for App3

Author: Nho Luong
Skill: DevOps Engineer Lead

# What are we going to learn ?




- Create App3 Target Groups and Listener Rules
- Create dns-to-db Route53 Record
- Create RDS Database Variables, RDS Module and RDS Outputs
- Create Userdata for Bastion Host to install MYSQL Client during creation of EC2 Instance using userdata
- Define RDS Database Variables
- Create  RDS DB related password as secure variable in secret.tfvars

Author: Nho Luong
Skill: DevOps Engineer Lead

# Upgrade Terraform Modules

| Module Type | Previous Version | New Version | Impact Analysis |
|---|---|---|---|
| VPC Terraform Module | v2.78.0 | v3.0.0 | No Impact |
| Security Group Terraform Module | v3.18.0 | v4.0.0 | High Impact |
| Application Load Balancer Terraform Module | v5.16.0 | v6.0.0 | High Impact |
| ACM Certificate Manager Terraform Module | v2.14.0 | v3.0.0 | High Impact |

There is a major change in all the modules (except vpc) related to output names of these modules which impacts all these module references wherever you use.

**Actual Change:** `this_` for all outputs was removed for these modules.

Author: Nho Luong
Skill: DevOps Engineer Lead

Amazon Virtual Private Cloud (Amazon VPC)

Internet gateway

NAT gateway

Elastic IP address


Route table

Public Subnet

Private Subnet

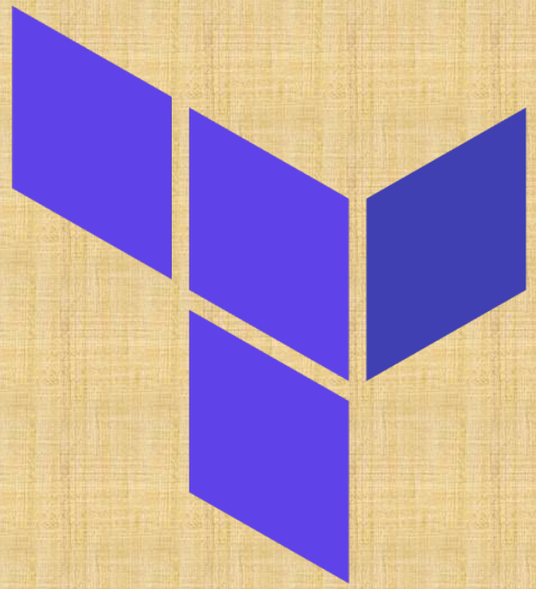
# AWS

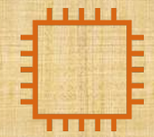
## VPC with Web, App and DB Tiers

## EC2 Instances, Bastion Host and Security Groups

## Application Load Balancer

## Autoscaling with Launch Configurations





EC2 Instance


Security Groups

Classic Load Balancer

Application Load Balancer

AWS Certificate Manager (ACM)


Amazon Route 53

Amazon RDS DB

Autoscaling

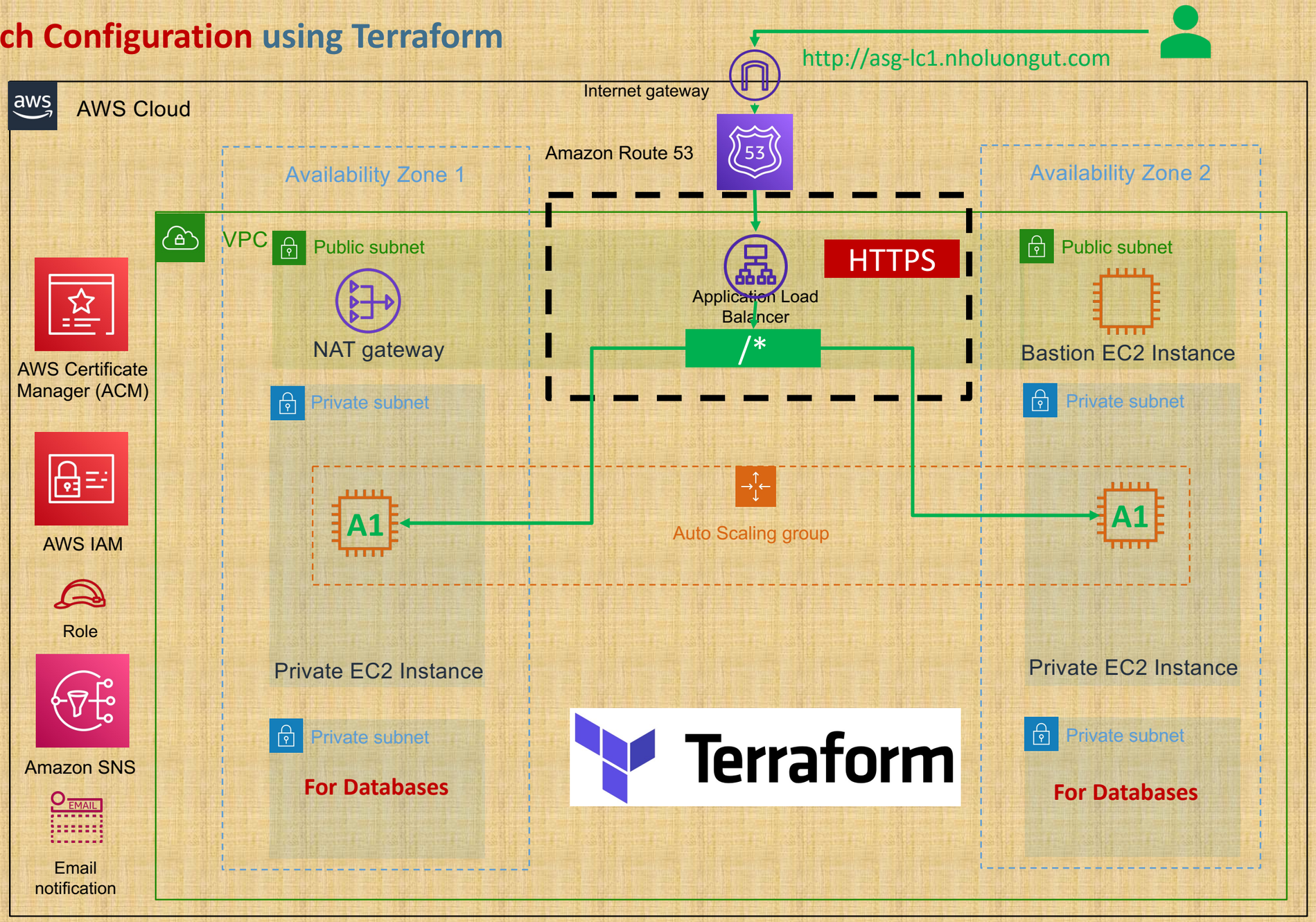
AWS IAM

Amazon SNS

Author: Nho Luong
Skill: DevOps Engineer Lead

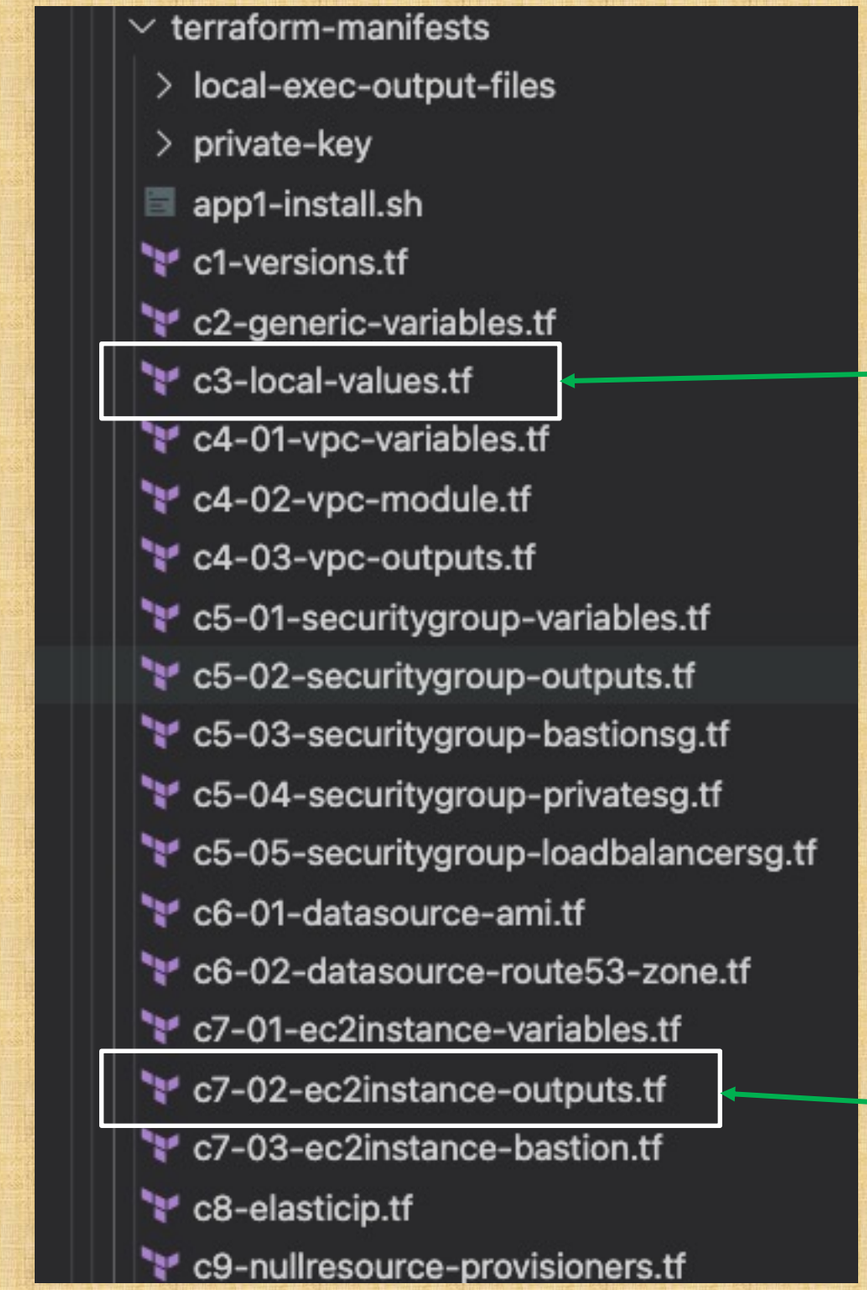
# AWS Autoscaling with Launch Configuration using Terraform

- Terraform ASG Module
- ASG with Launch Configuration
- ASG Instance Refresh
- ASG Lifecycle Hooks
- ASG TTSP
- ASG Scheduled Actions
- ASG Notifications
- ASG Autoscaling Tests





Author: Nho Luong
Skill: DevOps Engineer Lead



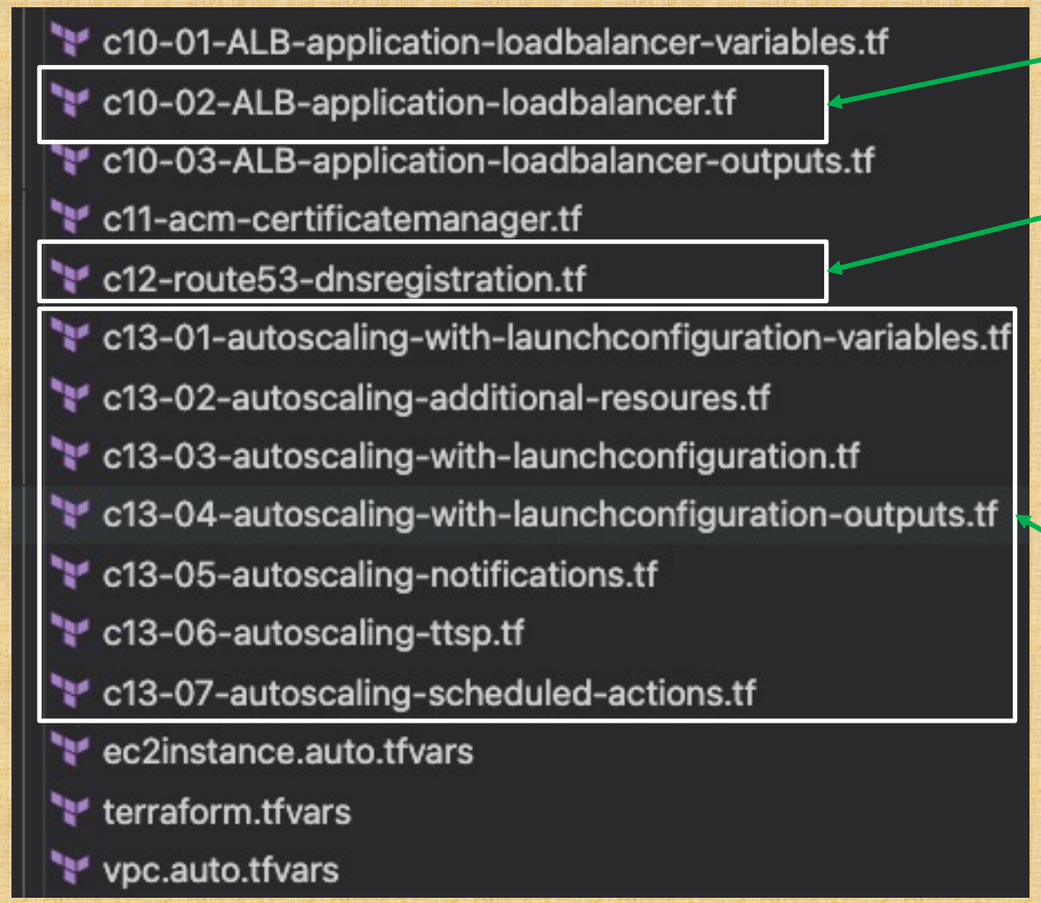

# What are we going to learn ?

Added Autoscaling related tags

Removed all private EC2 Instances (App1, App2) related configurations as we are going to create EC2 Instances using Autoscaling

Author: Nho Luong
Skill: DevOps Engineer Lead

# What are we going to learn ?




- ALB Load Balancer – Remove App1 Targets as we are going to link ALB with Autoscaling Group
- Update ASG Demo related DNS Name
- Terraform Autoscaling Module
- Autoscaling Service Linked Role
- Autoscaling Group with Launch Configuration
- Autoscaling Group Terraform Outputs
- Autoscaling Group Notifications
- Autoscaling Group TTSP and Scheduled Actions

Author: Nho Luong
Skill: DevOps Engineer Lead

Amazon Virtual Private Cloud (Amazon VPC)

Internet gateway

NAT gateway

Elastic IP address


Route table

Public Subnet

Private Subnet

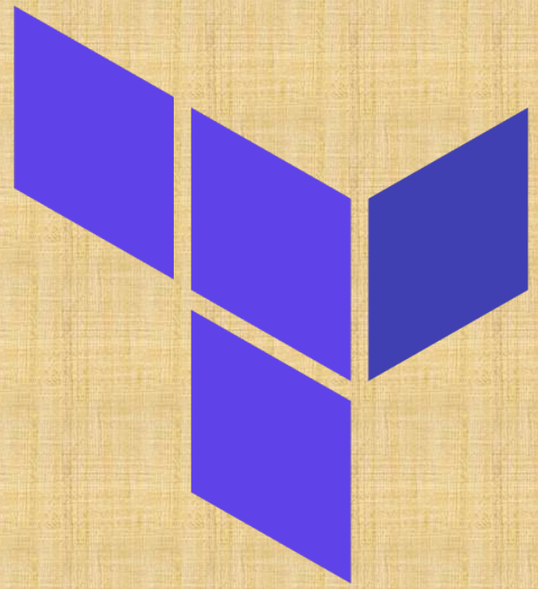
# AWS

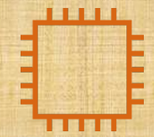
## VPC with Web, App and DB Tiers

## EC2 Instances, Bastion Host and Security Groups

## Application Load Balancer

## Autoscaling with Launch Templates





EC2 Instance


Security Groups

Classic Load Balancer

Application Load Balancer

AWS Certificate Manager (ACM)


Amazon Route 53

Amazon RDS DB

Autoscaling

AWS IAM

Amazon SNS

Author: Nho Luong
Skill: DevOps Engineer Lead

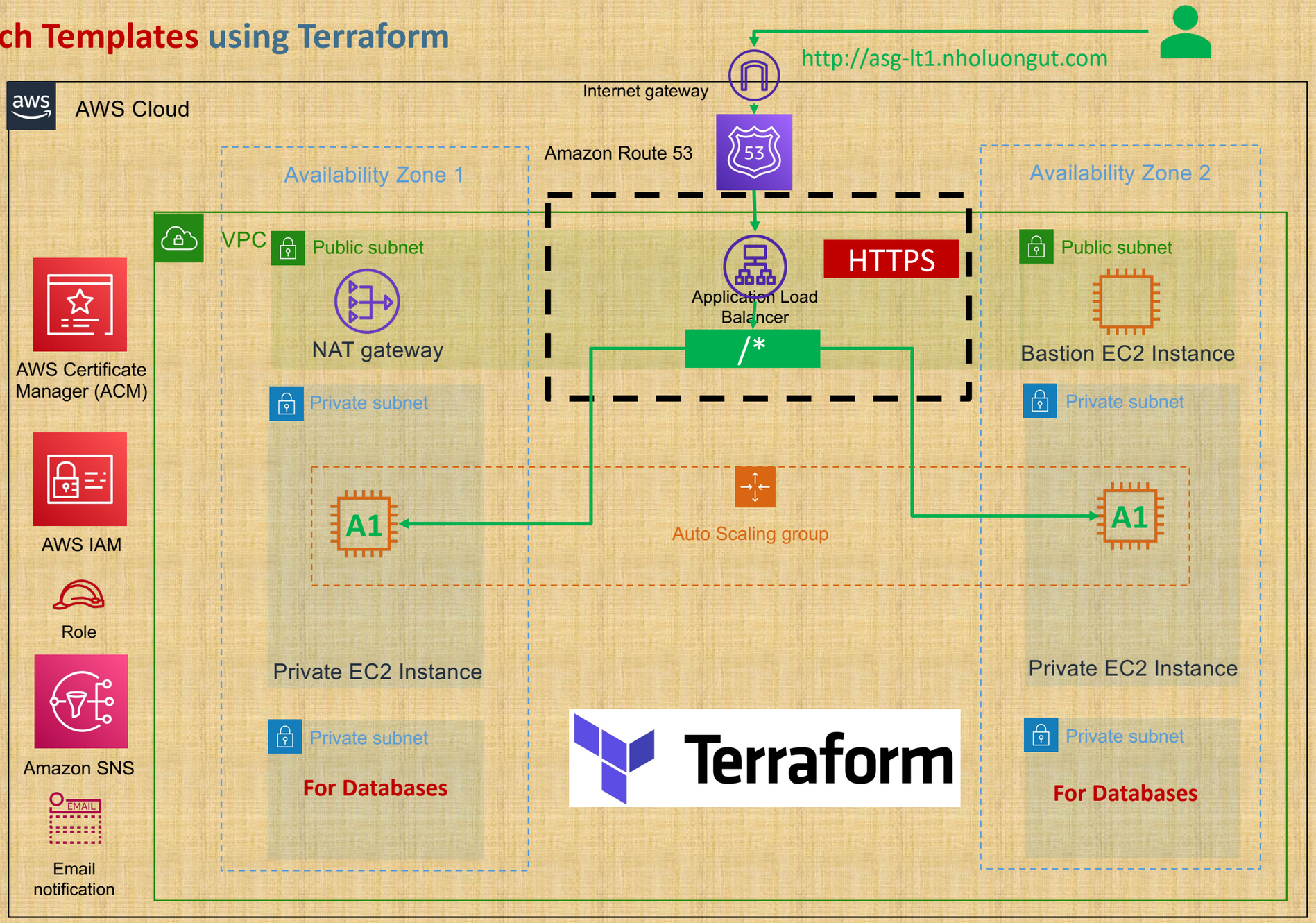
# AWS Autoscaling Launch Templates

- A launch template is similar to a launch configuration with latest and advanced features
- Template Versioning
  - We can create multiple versions of same template and use it. Tracking changes to LT by versioning them
- Template Re-Usability
  - Create base template with important features (Standardization of Templates)
  - Create app specific template by loading base template features and adding additional features to it.
- Going forward, start using Launch Templates for all your Autoscaling groups. This is latest and greatest available on AWS from ASG perspective.

Author: Nho Luong
Skill: DevOps Engineer Lead

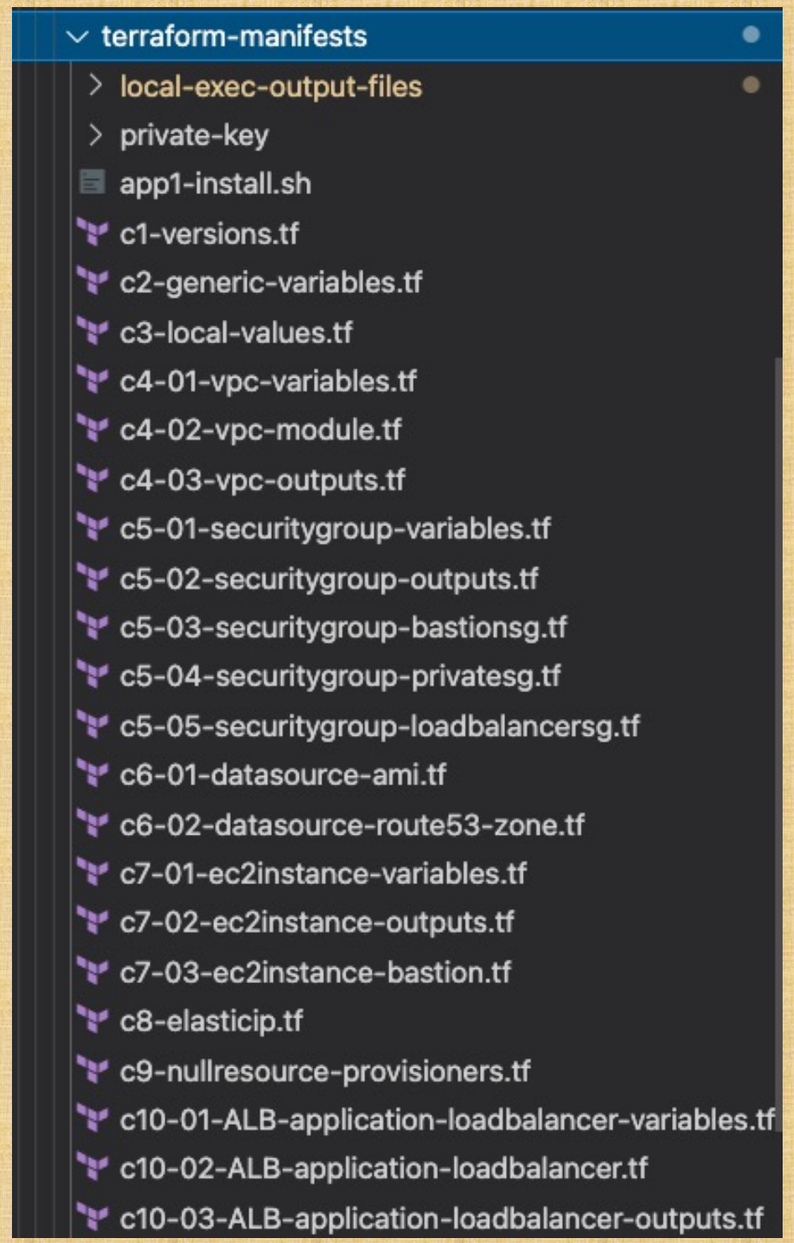
# AWS Autoscaling with Launch Templates using Terraform

- Terraform ASG Resource
- ASG with Launch Templates
- ASG Instance Refresh
- ASG Lifecycle Hooks
- ASG TTSP
- ASG Scheduled Actions
- ASG Notifications
- ASG Autoscaling Tests





Author: Nho Luong
Skill: DevOps Engineer Lead

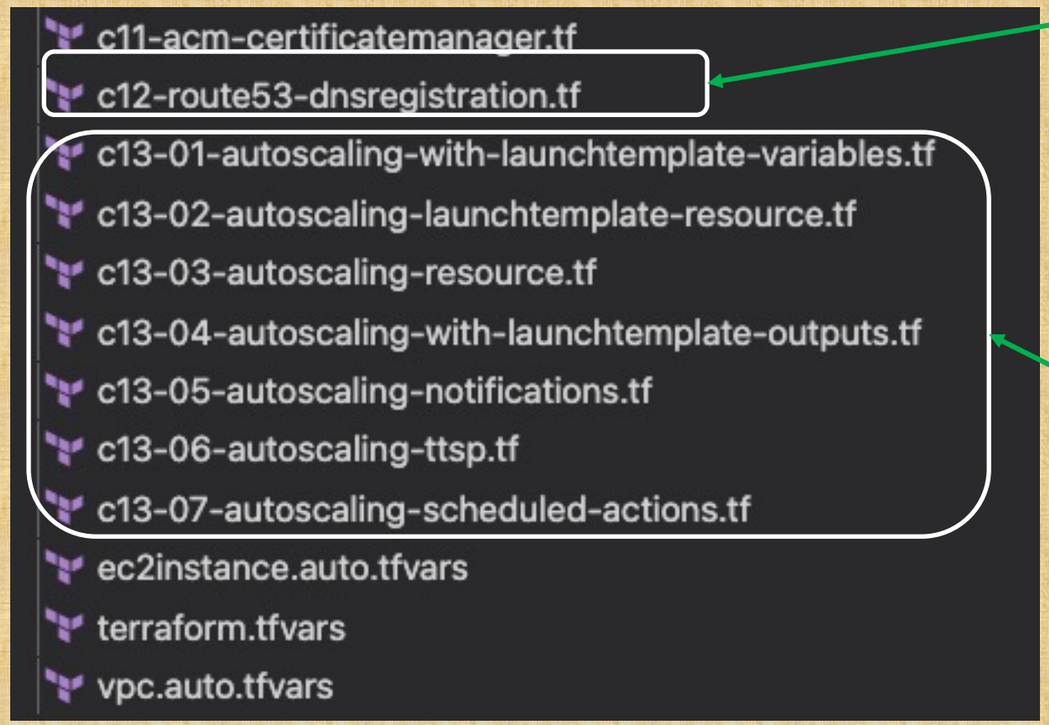



# What are we going to learn ?

No changes from previous section as these templates are copied from Autoscaling with Launch Configurations

Author: Nho Luong
Skill: DevOps Engineer Lead

# What are we going to learn ?




- DNS name changes relevant to demo
- Launch Template using Terraform Resource
- Autoscaling Group using Terraform Resource
- Launch Template and Autoscaling Group Terraform Outputs
- Autoscaling Notifications
- Autoscaling TTSP
- Autoscaling Scheduled Actions

Author: Nho Luong
Skill: DevOps Engineer Lead

Amazon Virtual Private Cloud (Amazon VPC)

Internet gateway

NAT gateway

Elastic IP address


Route table

Public Subnet

Private Subnet

# AWS

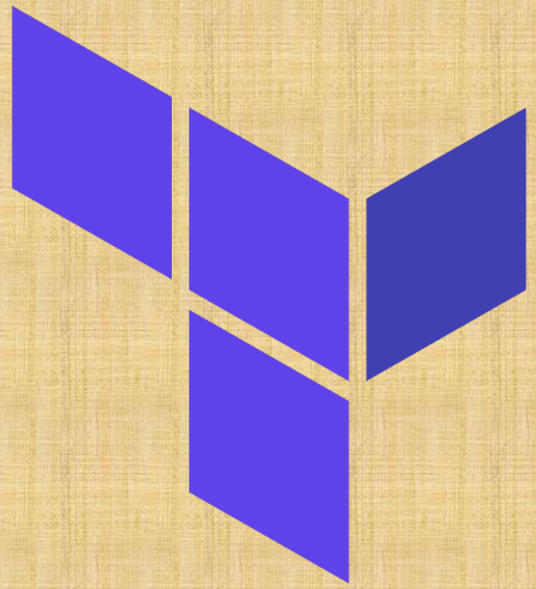
## VPC with Web, App and DB Tiers

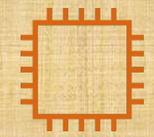
## EC2 Instances, Bastion Host and Security Groups

## Autoscaling with Launch Templates

## Network Load Balancer

## TCP & TLS Listeners





EC2 Instance


Security Groups

Classic Load Balancer

Application Load Balancer

AWS Certificate Manager (ACM)


Amazon Route 53

Amazon RDS DB

Autoscaling

AWS IAM

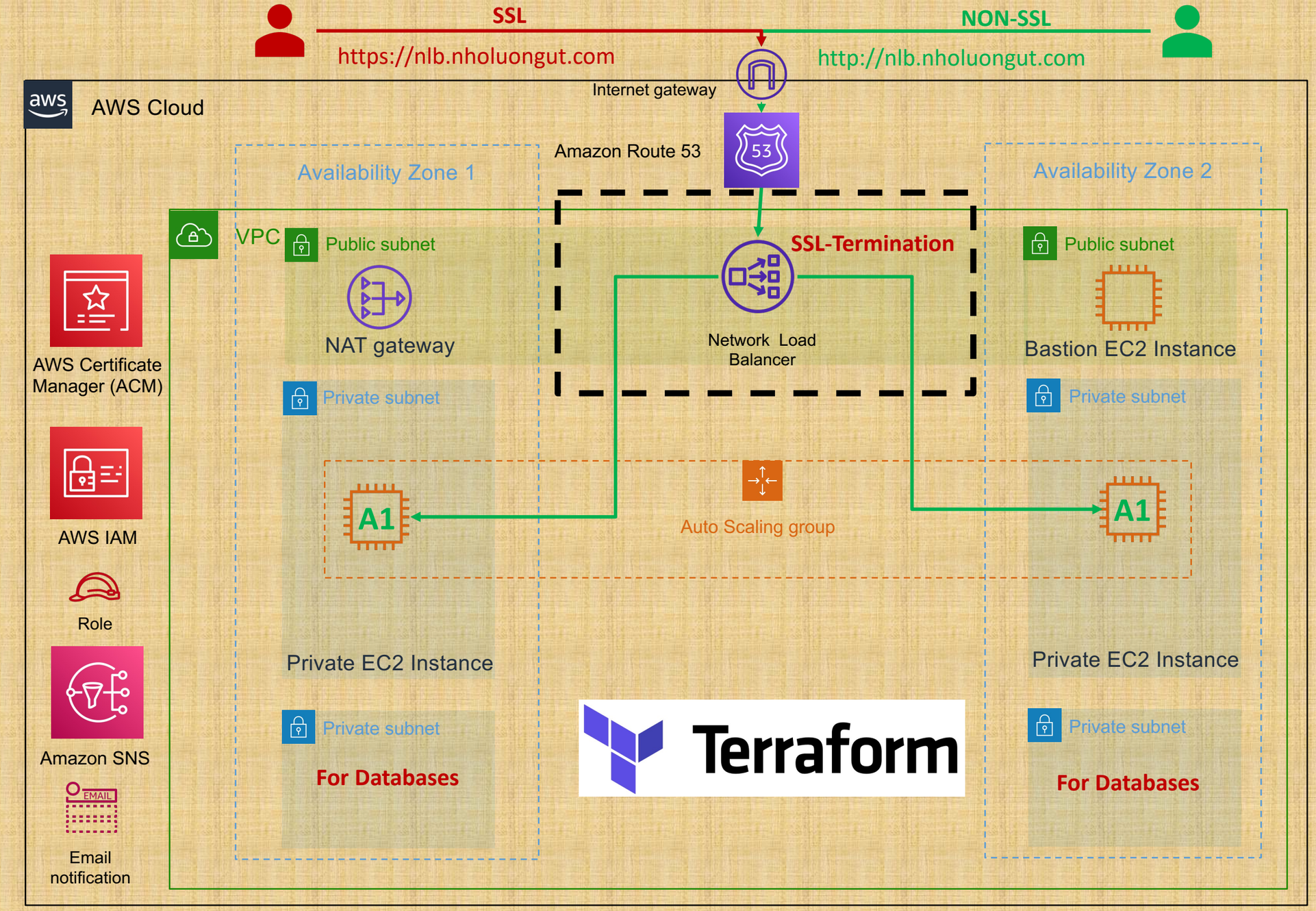
Amazon SNS

Network Load Balancer

Author: Nho Luong
Skill: DevOps Engineer Lead

# AWS Network Load Balancer with TCP & TLS Listeners using Terraform

- Terraform NLB Module
- Terraform NLB TCP Listeners
- Terraform NLB TLS Listeners
- Terraform NLB Target Groups
- Terraform NLB with EC2 Autoscaling Groups





Author: Nho Luong
Skill: DevOps Engineer Lead

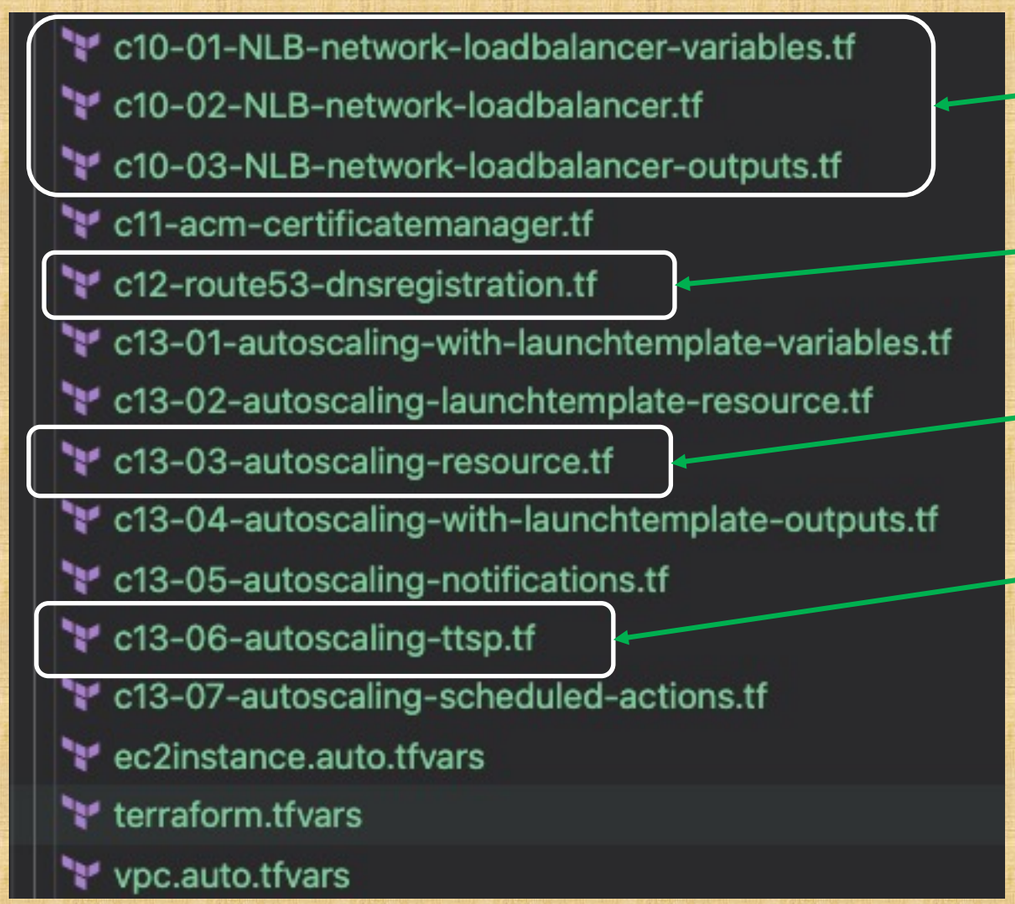
# What are we going to learn ?


Changes related to NLB for Ingress CIDR Block to be 0.0.0.0/0

No changes from c1 to c9 except c5-04

Author: Nho Luong
Skill: DevOps Engineer Lead

# What are we going to learn ?




Network Load Balancer using Terraform Module with TCP and TLS Listeners

Update Route53 related DNS Name and NLB References

Integrate Autoscaling Group and NLB to attach Autoscaling Group EC2 Instances to NLB Target Group

Comment TTSP ALB Policy which is not applicable for NLB usecase

Author: Nho Luong
Skill: DevOps Engineer Lead

Amazon Virtual Private Cloud (Amazon VPC)

Internet gateway

NAT gateway

Elastic IP address


Route table

Public Subnet

Private Subnet

# AWS

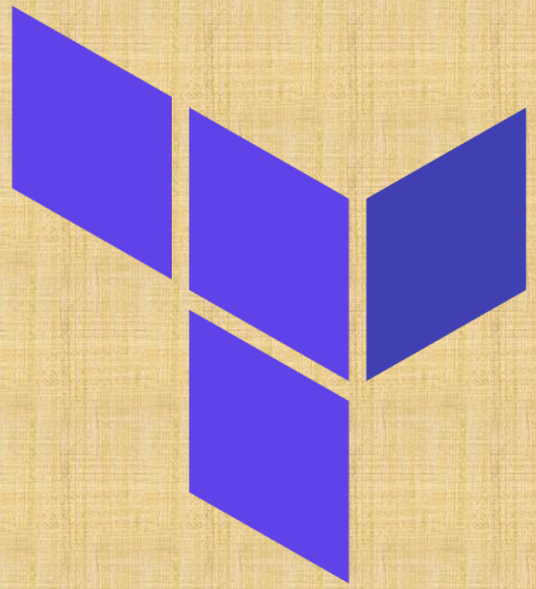
## VPC with Web, App and DB Tiers

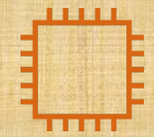
## EC2 Instances, Bastion Host and Security Groups

## Application Load Balancer

## Autoscaling with Launch Templates

## CloudWatch Alarms





EC2 Instance


Security Groups

Classic Load Balancer

Application Load Balancer

AWS Certificate Manager (ACM)


Amazon Route 53

Amazon RDS DB

Autoscaling

AWS IAM

Amazon SNS

CloudWatch

Author: Nho Luong
Skill: DevOps Engineer Lead

# AWS CloudWatch using Terraform

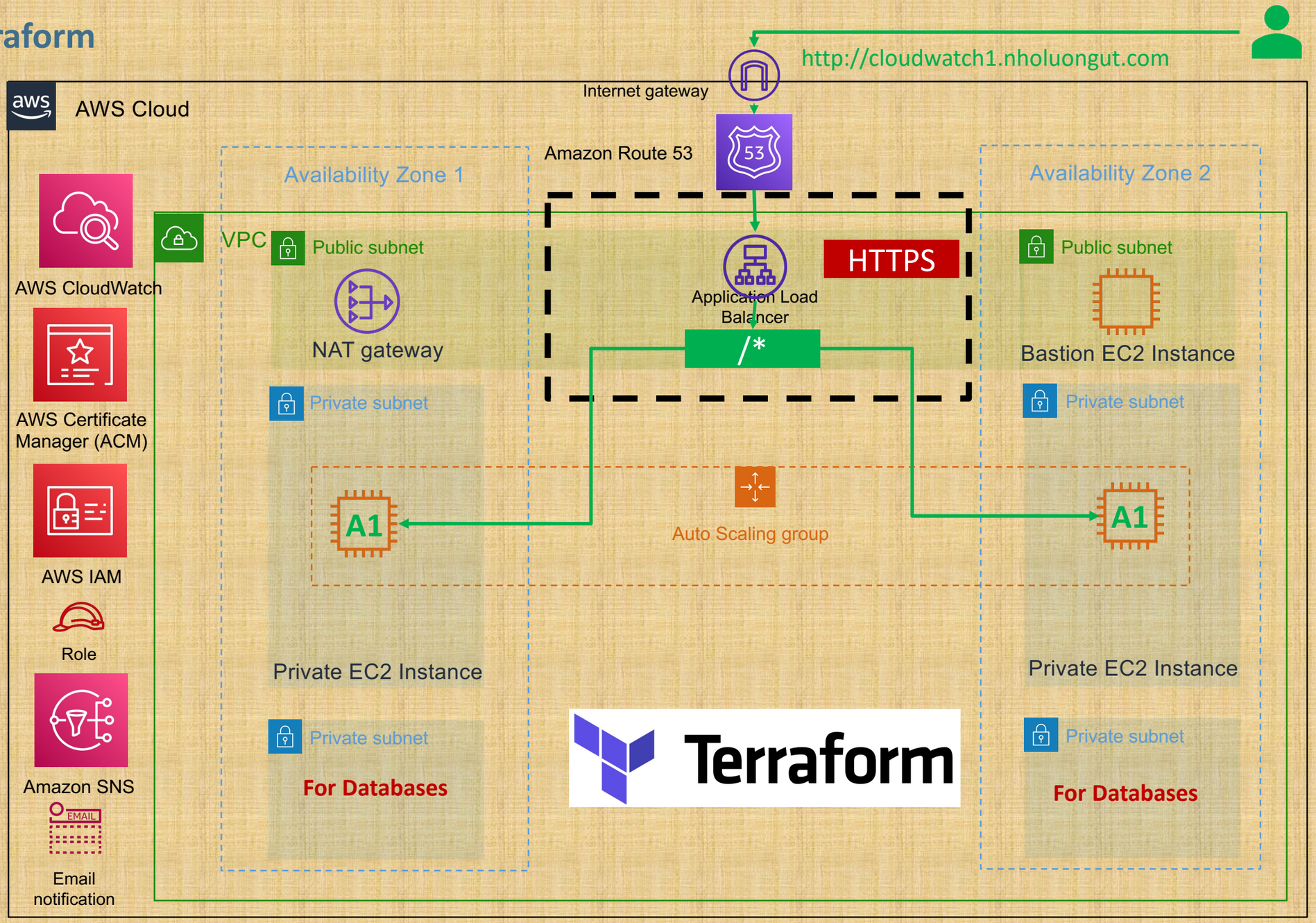
AWS CloudWatch

ASG Alarms

ALB Alarms

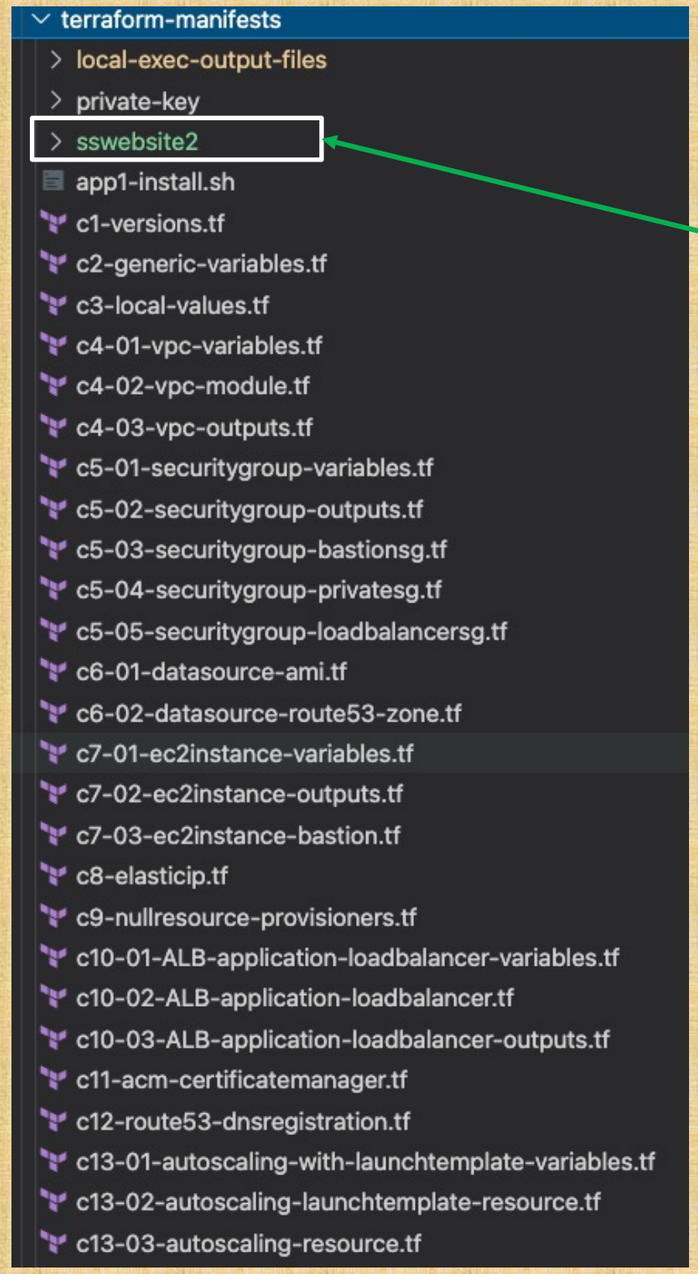
CIS Alarms

CloudWatch Synthetics





Author: Nho Luong
Skill: DevOps Engineer Lead



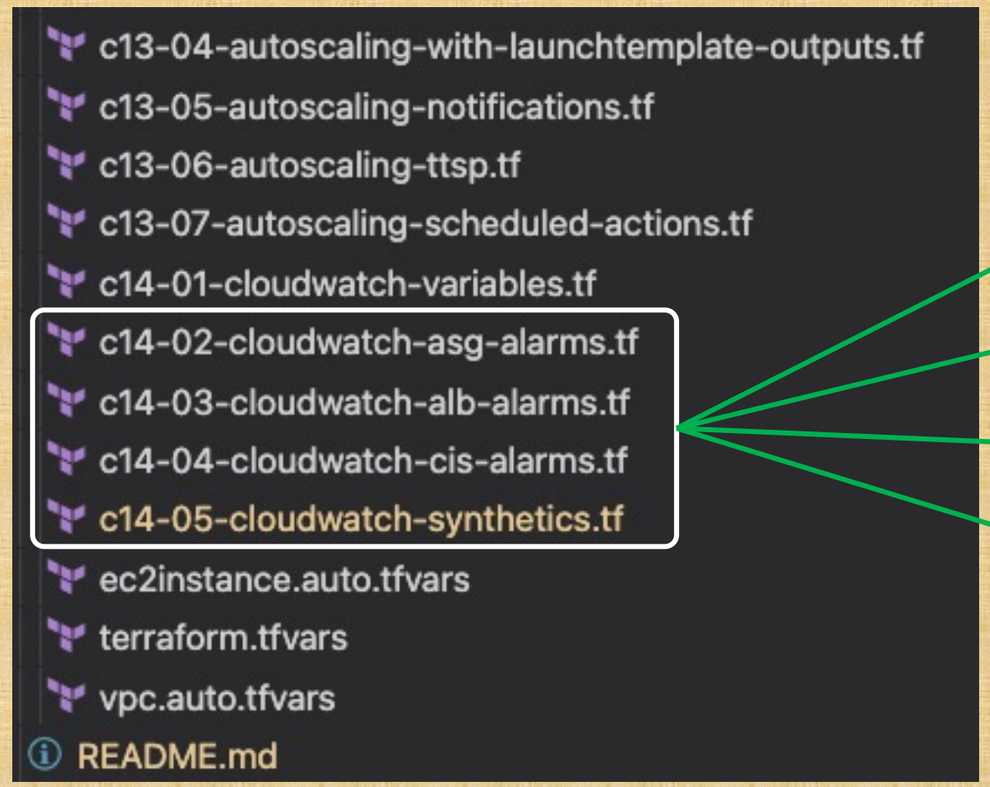

# What are we going to learn ?

AWS CloudWatch Synthetics related Node JS Files for Heart Beat Monitor

No changes from c1 to c13

Author: Nho Luong
Skill: DevOps Engineer Lead

# What are we going to learn ?




- AWS CloudWatch Autoscaling Alarms
- AWS CloudWatch Application Load Balancer Alarms
- AWS CloudWatch CIS Alarms
- AWS CloudWatch Synthetics

Author: Nho Luong
Skill: DevOps Engineer Lead

Amazon Virtual Private Cloud (Amazon VPC)

Internet gateway

NAT gateway

Elastic IP address


Route table

Public Subnet

Private Subnet

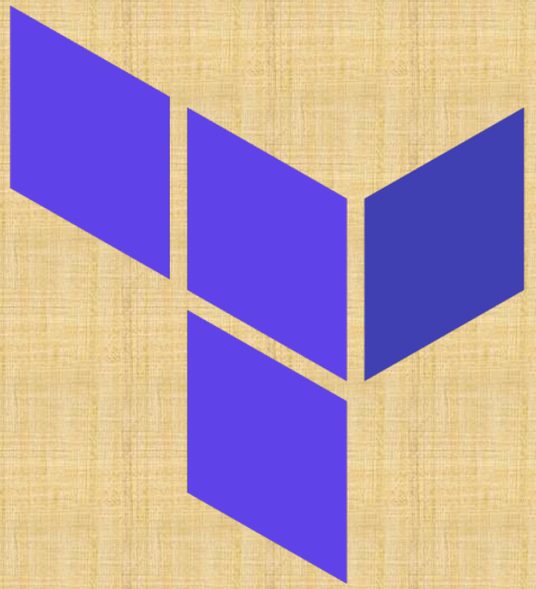
Amazon Simple Storage Service (Amazon S3)

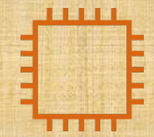
Amazon DynamoDB

# AWS

## VPC with Web, App and DB Tiers

## Build Terraform Modules Locally





EC2 Instance


Security Groups

Classic Load Balancer

Application Load Balancer

AWS Certificate Manager (ACM)


Amazon Route 53

Amazon RDS DB

Autoscaling

AWS IAM

Amazon SNS

CloudWatch

Author: Nho Luong
Skill: DevOps Engineer Lead

# Build Terraform Modules Locally

## Build from Scratch

If already not available, then use this approach

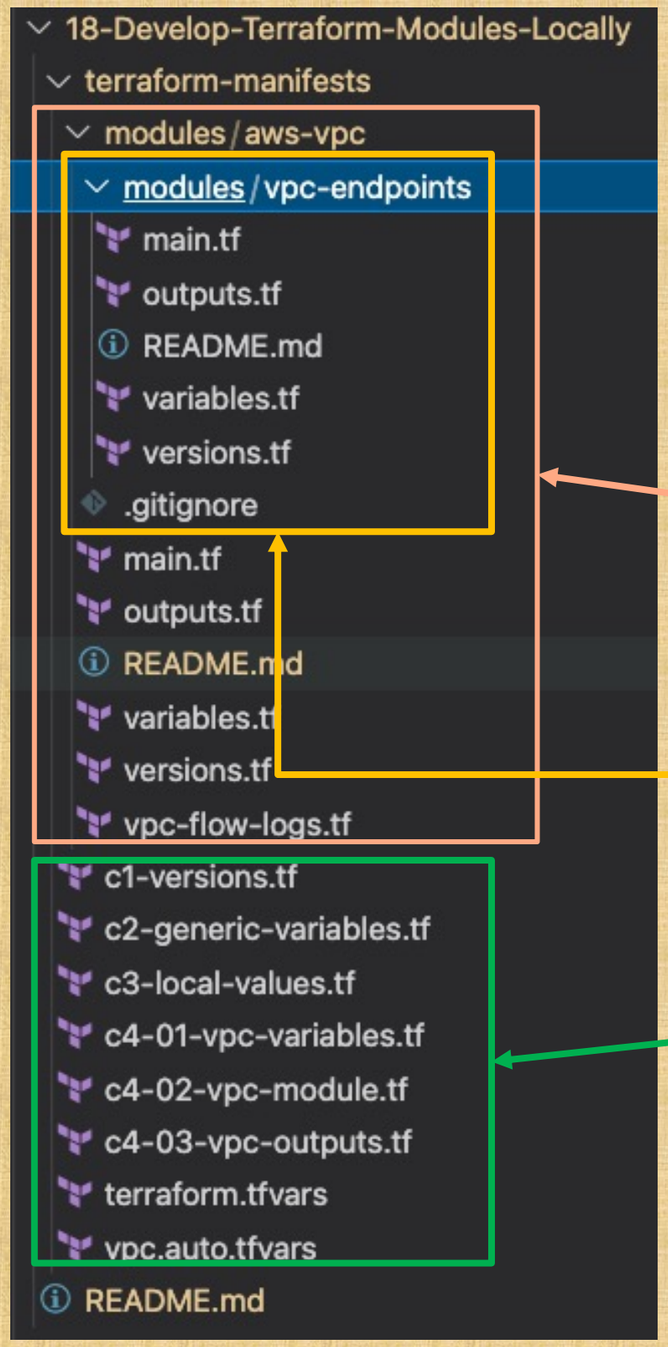
Review existing modules in Terraform Public Registry (Minimum 3 modules to get complete idea)

## Leverage existing Terraform Modules and change the code as per your need

If already available, use the code and modify as per your need

Less effort and you can start using them in your private infrastructure spaces without public internet access

Author: Nho Luong
Skill: DevOps Engineer Lead




# What are we going to learn ?

AWS VPC Module

AWS VPC Modules - VPC Endpoints Sub Module

Root Module calling the VPC Local Module

Author: Nho Luong
Skill: DevOps Engineer Lead

Amazon Virtual Private Cloud (Amazon VPC)

Internet gateway

NAT gateway

Elastic IP address


Route table

Public Subnet

Private Subnet

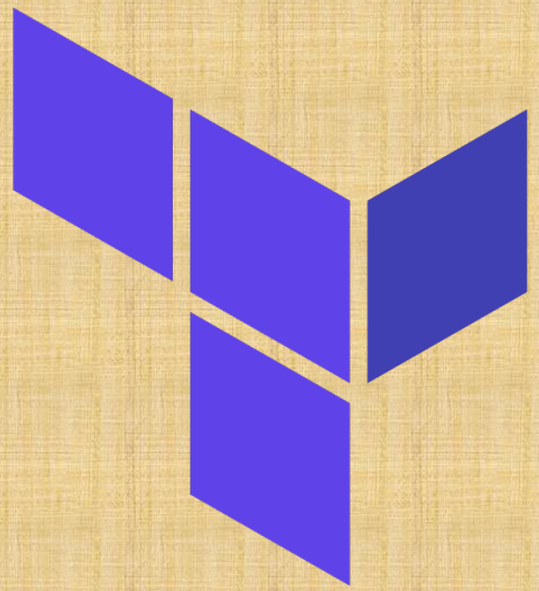
Amazon Simple Storage Service (Amazon S3)

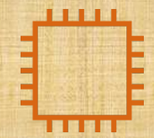
Amazon DynamoDB

# AWS

## VPC with Web, App and DB Tiers

## Remote State Storage with AWS S3 & AWS DynamoDB





EC2 Instance


Security Groups

Classic Load Balancer

Application Load Balancer

AWS Certificate Manager (ACM)


Amazon Route 53

Amazon RDS DB

Autoscaling

AWS IAM

Amazon SNS

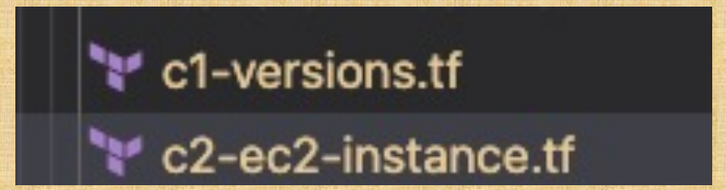
CloudWatch

Author: Nho Luong
Skill: DevOps Engineer Lead

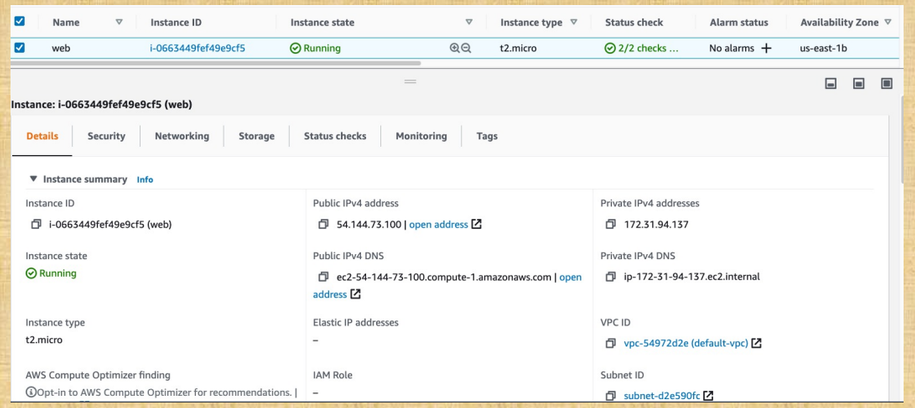
# Desired & Current Terraform States

Terraform Configuration Files

Real World Resource – EC2 Instance








Desired State

Current State

Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform State

Terraform
Local
State
Storage

Terraform
Remote
State
Storage

Author: Nho Luong
Skill: DevOps Engineer Lead

# What is Terraform Backend ?

Backends are responsible for storing state and providing an API for state locking.

Terraform
State Storage

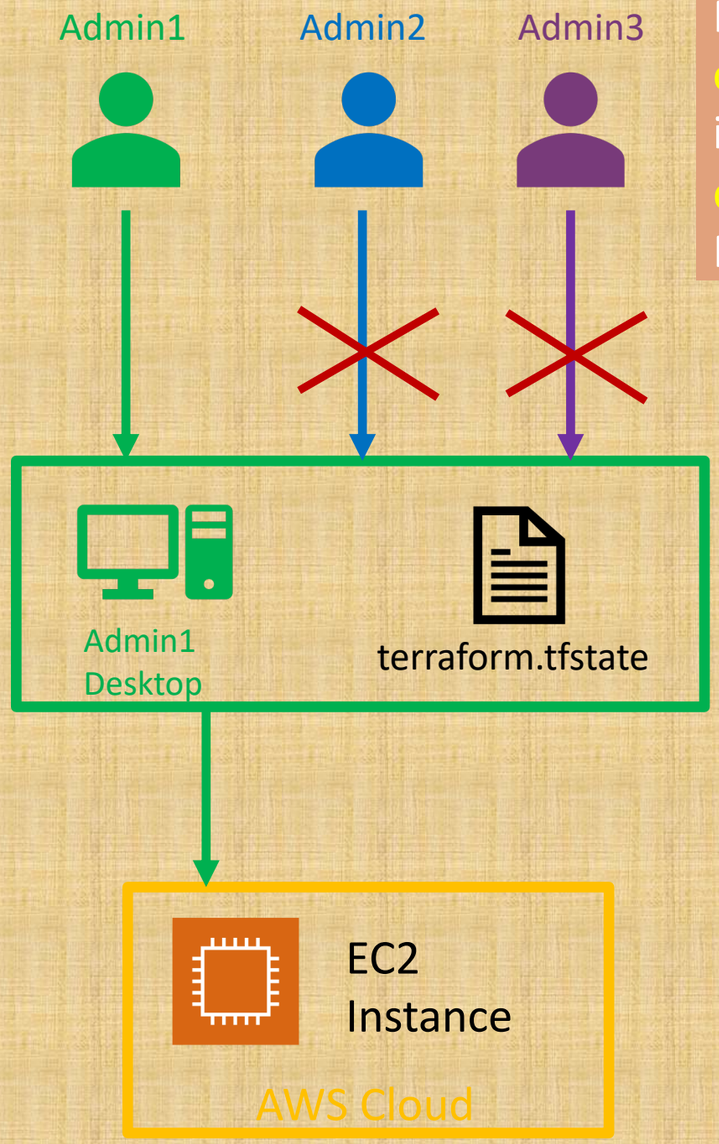
AWS S3 Bucket

Terraform
State Locking

AWS DynamoDB

Author: Nho Luong
Skill: DevOps Engineer Lead

# Local State File




Multiple Team members cannot update the infrastructure as they don't have access to State File

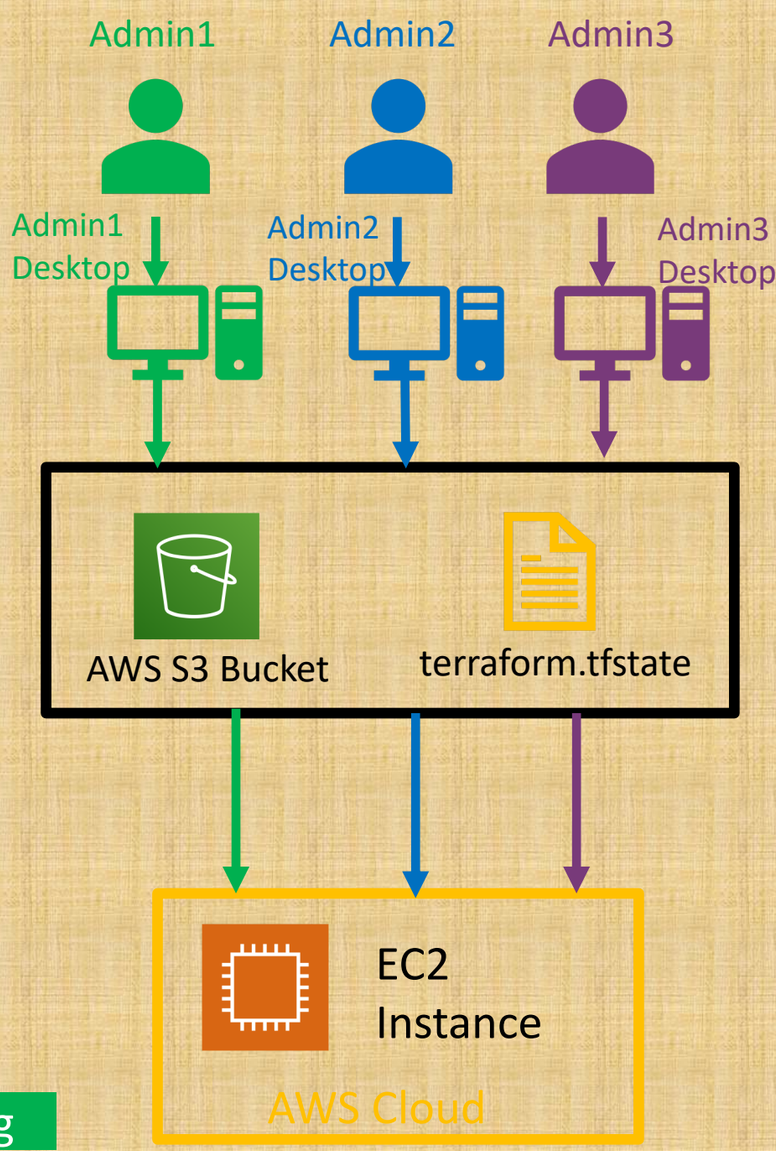
This means we need store the state file in a shared location.

# Remote State File

Using Terraform Backend concept we can use AWS S3 as the shared storage for State Files

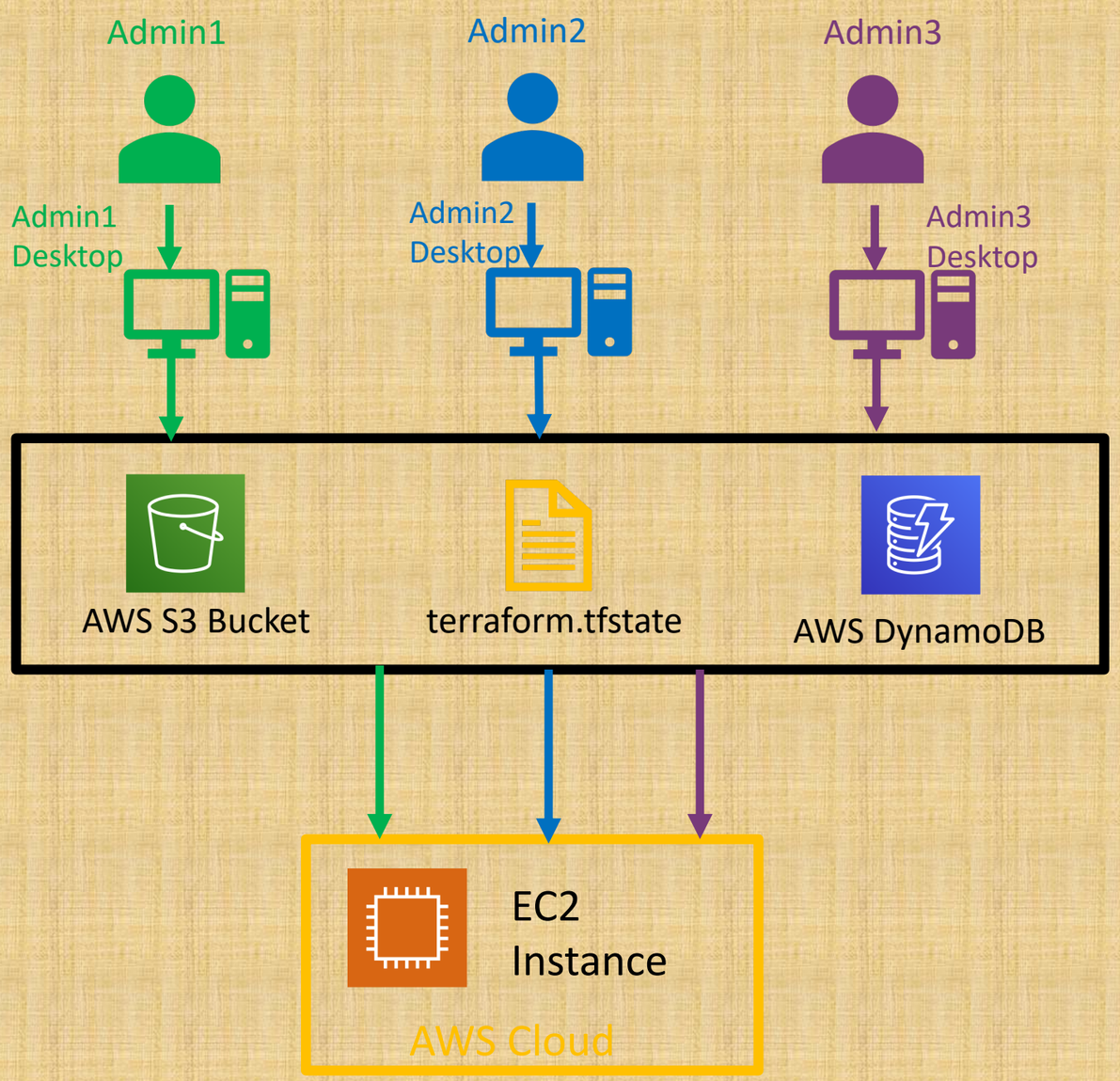
If two team members are running Terraform at the same time, you may run into race conditions as multiple Terraform processes make concurrent updates to the state files, leading to conflicts, data loss, and state file corruption.

Implement State Locking




Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Remote State File with State Locking




Not all backends support State Locking. AWS S3 supports State Locking

State locking happens automatically on all operations that could write state.

If state locking fails, Terraform will not continue.

You can disable state locking for most commands with the -lock flag but it is not recommended.

If acquiring the lock is taking longer than expected, Terraform will output a status message.

If Terraform doesn't output a message, state locking is still occurring if your backend supports it.

Terraform has a force-unlock command to manually unlock the state if unlocking failed.

Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Remote State File with State Locking

Terraform State Storage to Remote Backend

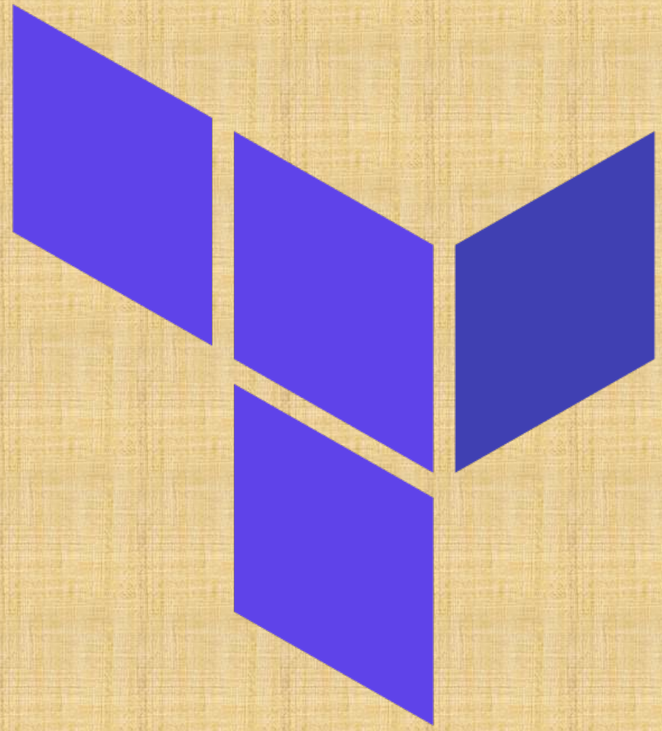
Terraform State Locking

```
# Terraform Block
terraform {
  required_version = "~> 0.14" # which means any ve
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
  # Adding Backend as S3 for Remote State Storage
  backend "s3" {
    bucket = "terraform-stacksimplify"
    key    = "dev/terraform.tfstate"
    region = "us-east-1"

    # Enable during Step-09
    # For State Locking
    dynamodb_table = "terraform-dev-state-table"
  }
}
```

Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform
# Backends



Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Backends

Each Terraform configuration can specify a backend, which defines where and how operations are performed, where state snapshots are stored, etc.

**Where Backends are Used**

Backend configuration is only used by Terraform CLI.

Terraform Cloud and Terraform Enterprise always use their own state storage when performing Terraform runs, so they ignore any backend block in the configuration.

For Terraform Cloud users also it is always recommended to use backend block in Terraform configuration  for commands like terraform taint which can be executed only using Terraform CLI

Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Backends

**What Backends Do**

There are two things backends will be used for

1. Where state is stored
2. Where operations are performed.

## Store State

Terraform uses persistent state data to keep track of the resources it manages.

Everyone working with a given collection of infrastructure resources must be able to access the same state data (shared state storage).

## State Locking

State Locking is to prevent conflicts and inconsistencies when the operations are being performed

What are Operations ?
terraform apply
terraform destroy

## Operations

"Operations" refers to performing API requests against infrastructure services in order to create, read, update, or destroy resources.

Not every terraform subcommand performs API operations; many of them only operate on state data.

Only two backends actually perform operations: local and remote.

The remote backend can perform API operations remotely, using Terraform Cloud or Terraform Enterprise.

Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Backends

## Backend Types

### Enhanced Backends

**Enhanced** backends can both store state and perform operations. There are only two enhanced backends: local and remote

Example for Remote Backend Performing Operations : Terraform Cloud, Terraform Enterprise

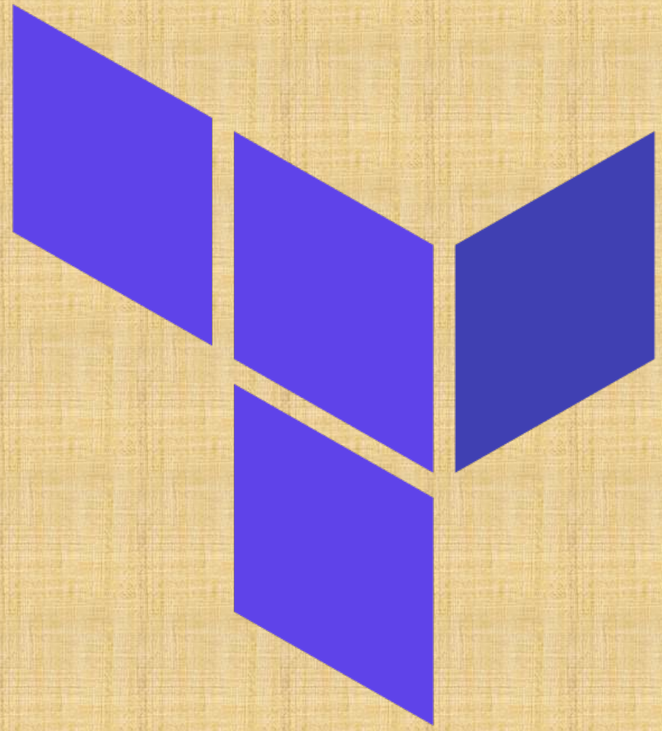
### Standard Backends

**Standard** backends only store state, and rely on the local backend for performing operations.

Example: AWS S3, Azure RM, Consul, etcd, gcs http and many more

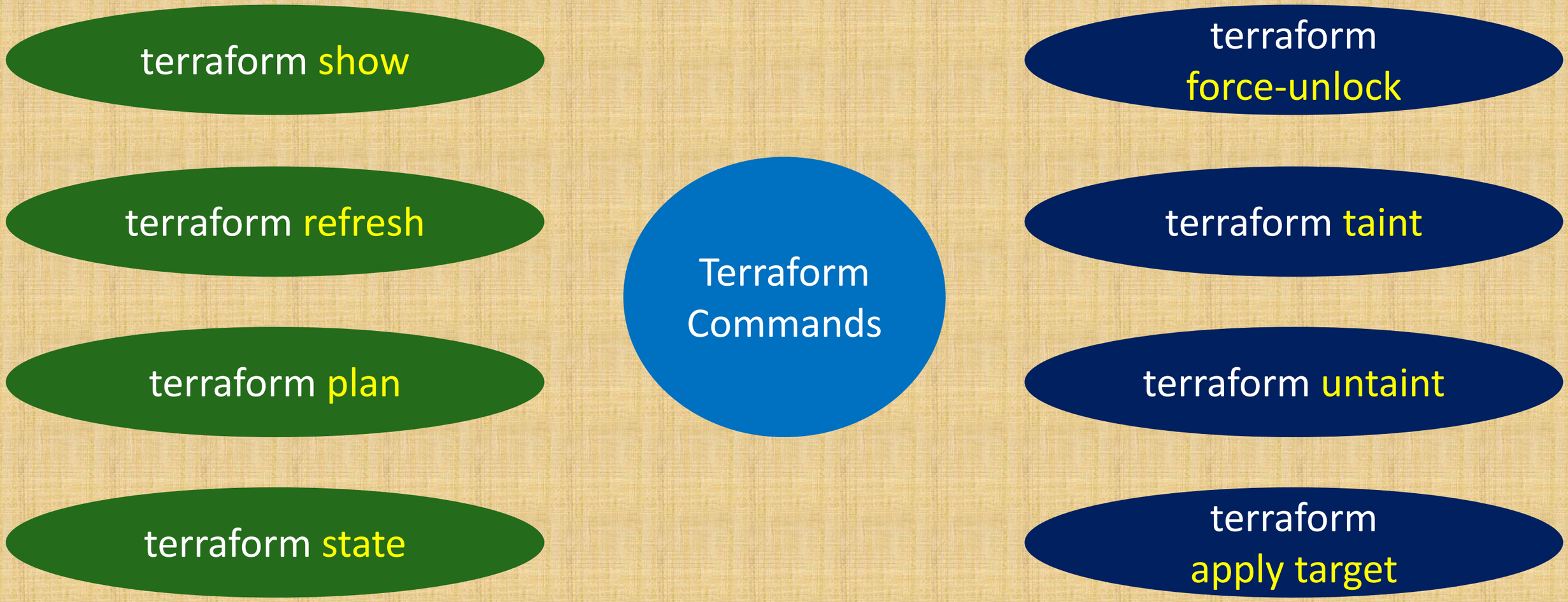
Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform
# State Commands



Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Commands – State Perspective




Author: Nho Luong
Skill: DevOps Engineer Lead

Amazon Virtual Private Cloud (Amazon VPC)

Internet gateway

NAT gateway

Elastic IP address


Route table

Public Subnet

Private Subnet

# AWS

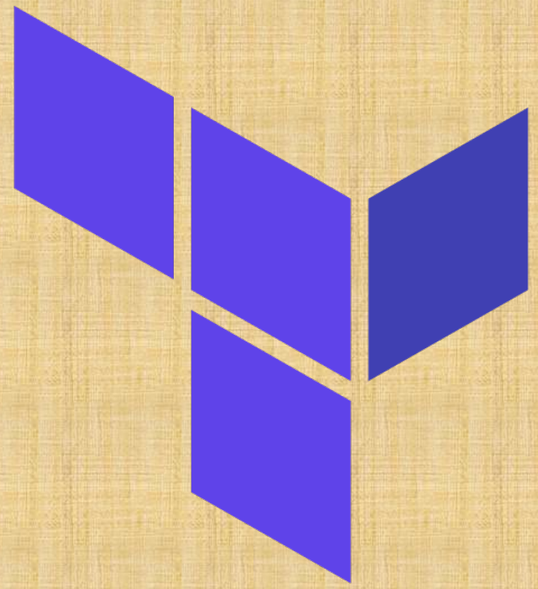
## VPC with Web, App and DB Tiers

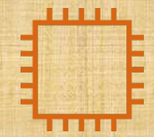
## EC2 Instances, Bastion Host and Security Groups

## Application Load Balancer

## Autoscaling with Launch Templates

## Terraform Remote State Datasource





EC2 Instance


Security Groups

Classic Load Balancer

Application Load Balancer

AWS Certificate Manager (ACM)


Amazon Route 53

Amazon RDS DB

Autoscaling

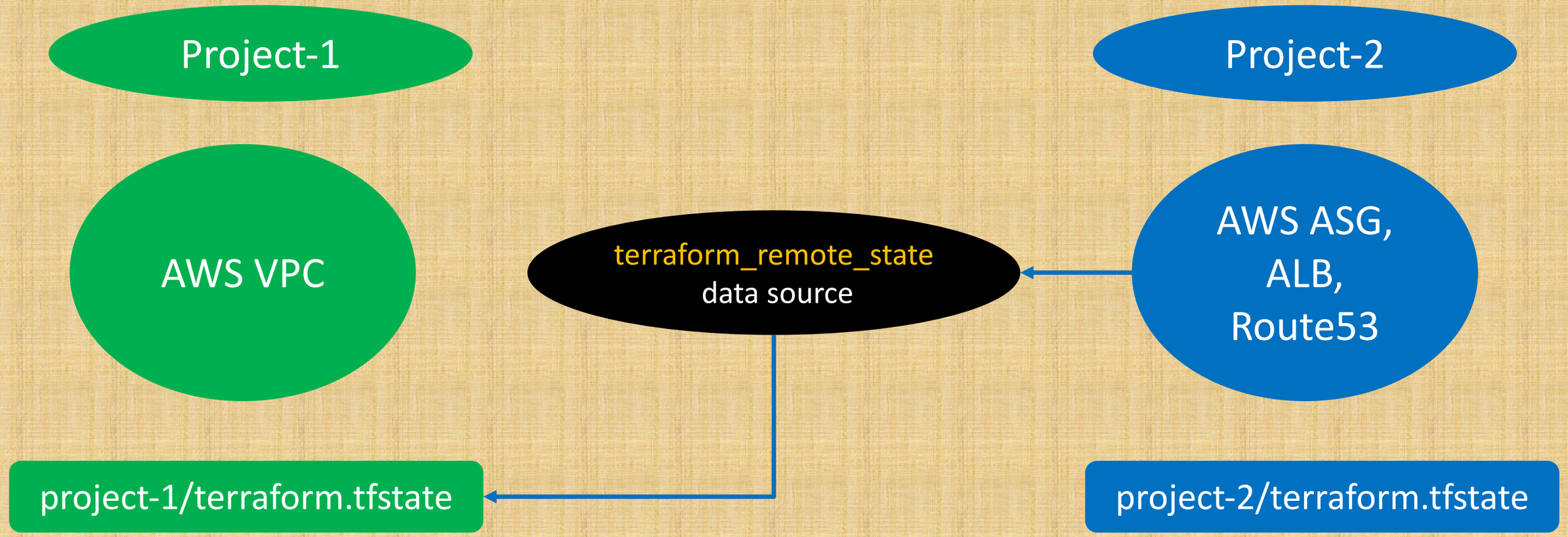
AWS IAM

Amazon SNS

Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Remote State Datasource

The terraform_remote_state data source retrieves the root module output values from some other Terraform configuration, using the latest state snapshot from the remote backend.




Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Remote State Datasource

Project-1

VPC Resources

Project-2

Autoscaling Groups

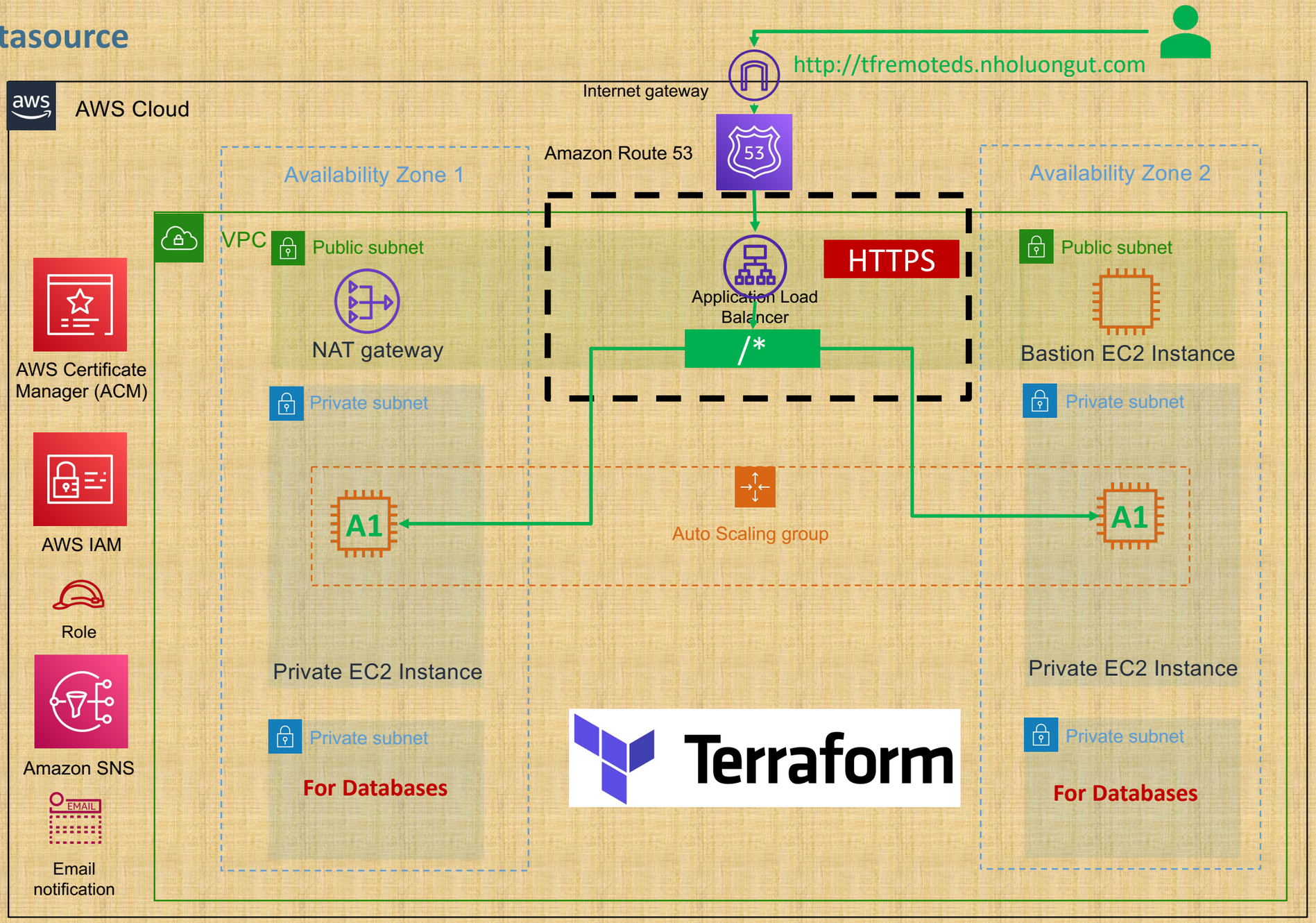
Application Load Balancers

AWS Route53

AWS Certificate Manager

AWS SNS

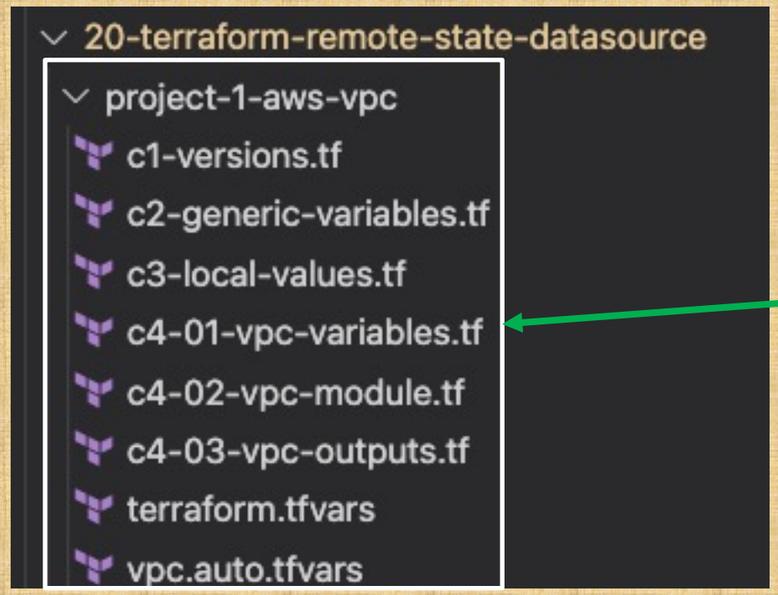
AWS IAM





Author: Nho Luong
Skill: DevOps Engineer Lead

# What are we going to learn ?




**Project-1**

Create VPC Resources with Terraform Backend as S3

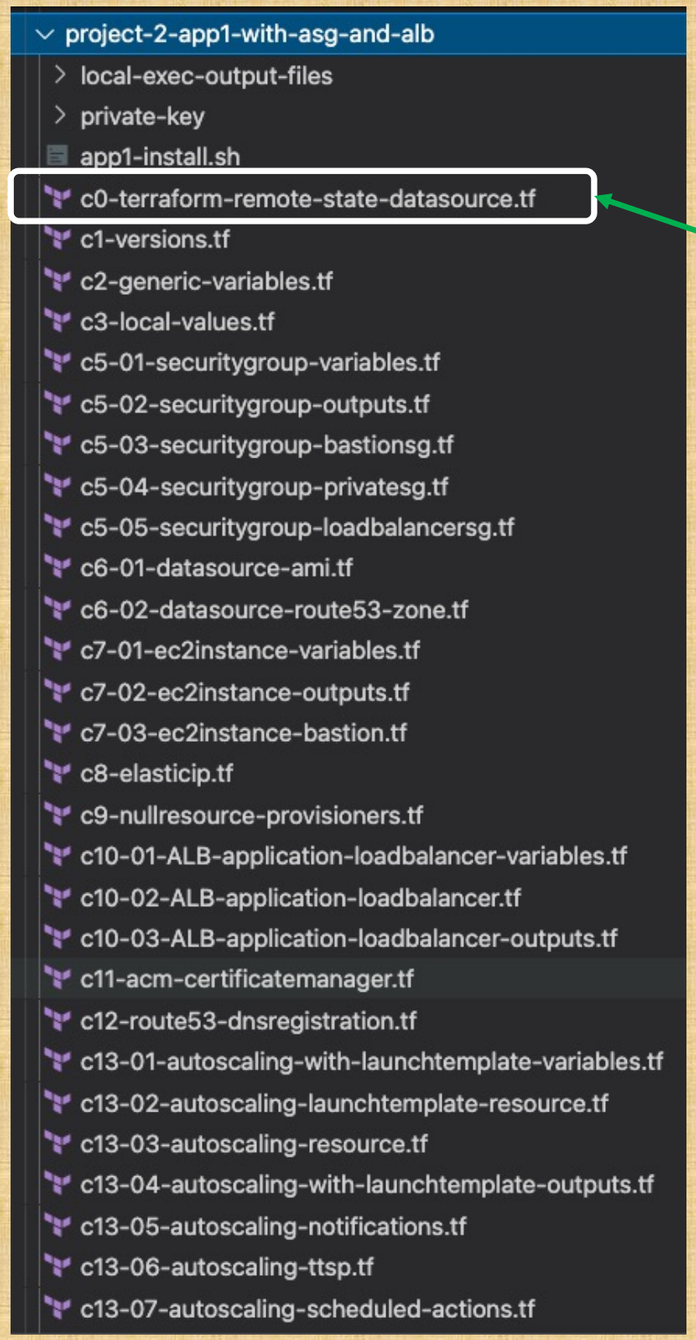
Author: Nho Luong
Skill: DevOps Engineer Lead

# Project-1: Terraform Backend

```
# Adding Backend as S3 for Remote State Storage
backend "s3" {
  bucket = "terraform-on-aws-for-ec2"
  key    = "dev/project1-vpc/terraform.tfstate"
  region = "us-east-1"

  # Enable during Step-09
  # For State Locking
  dynamodb_table = "dev-project1-vpc"
}
}
```

Author: Nho Luong
Skill: DevOps Engineer Lead

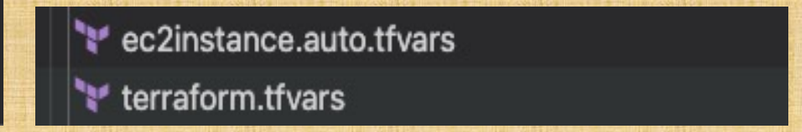



# What are we going to learn ?

Terraform Remote State Datasource Definition which will reference the Project-1 state file

**Project-2**

In this project, we will create Autoscaling Groups, Application Load Balancers in VPC created using Project-1 using Terraform Remote State Datasource.




Author: Nho Luong
Skill: DevOps Engineer Lead

# Project-2: Terraform Remote State Datasource

```
# Terraform Remote State Datasource
data "terraform_remote_state" "vpc" {
  backend = "s3"
  config = {
    bucket = "terraform-on-aws-for-ec2"
    key    = "dev/project1-vpc/terraform.tfstate"
    region = "us-east-1"
  }
}
```

Author: Nho Luong
Skill: DevOps Engineer Lead

# Project-2: Terraform Backend

```
  # Adding Backend as S3 for Remote State Storage
  backend "s3" {
    bucket = "terraform-on-aws-for-ec2"
    key    = "dev/project2-app1/terraform.tfstate"
    region = "us-east-1"

    # Enable during Step-09
    # For State Locking
    dynamodb_table = "dev-project2-app1"
  }
}
```

Author: Nho Luong
Skill: DevOps Engineer Lead

Amazon Virtual Private Cloud (Amazon VPC)

Internet gateway

NAT gateway

Elastic IP address


Route table

Public Subnet

Private Subnet

# AWS

## VPC with Web, App and DB Tiers

## EC2 Instances, Bastion Host and Security Groups

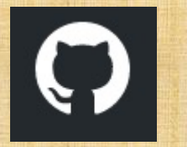
## Application Load Balancer

## Autoscaling with Launch Templates

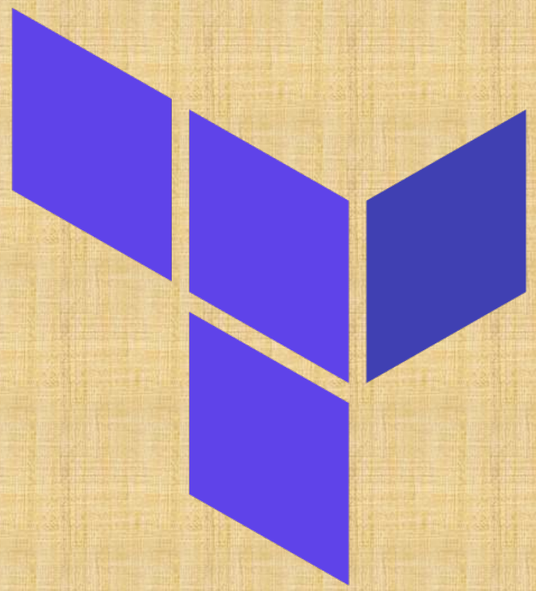
## IaC DevOps with AWS CodePipeline

AWS CodePipeline

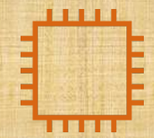
AWS CodeBuild



GitHub



Amazon Simple Storage Service (Amazon S3)



EC2 Instance


Security Groups

Classic Load Balancer

Application Load Balancer

AWS Certificate Manager (ACM)


Amazon Route 53

Amazon RDS DB

Autoscaling

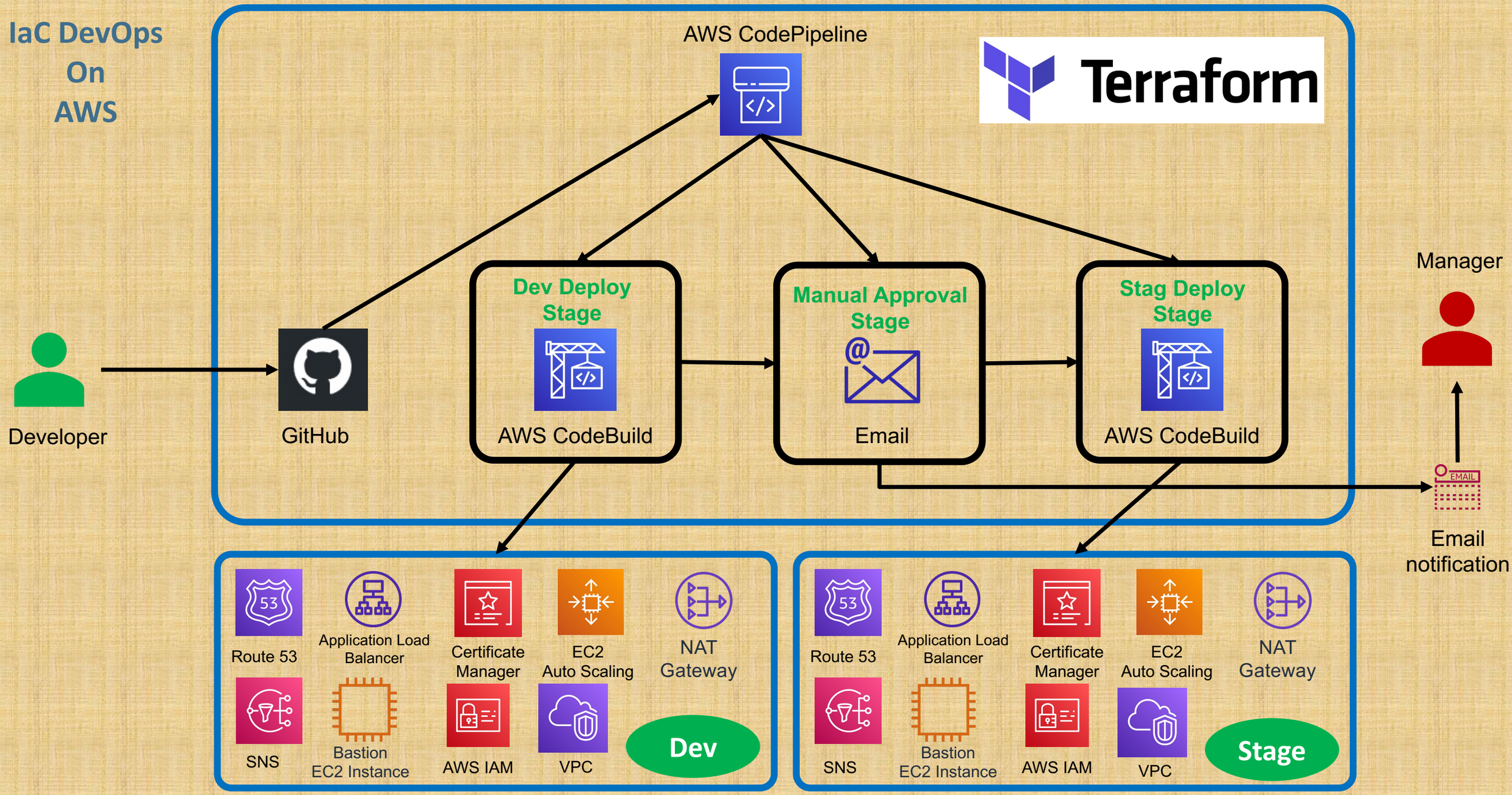
AWS IAM

Amazon SNS

Amazon DynamoDB

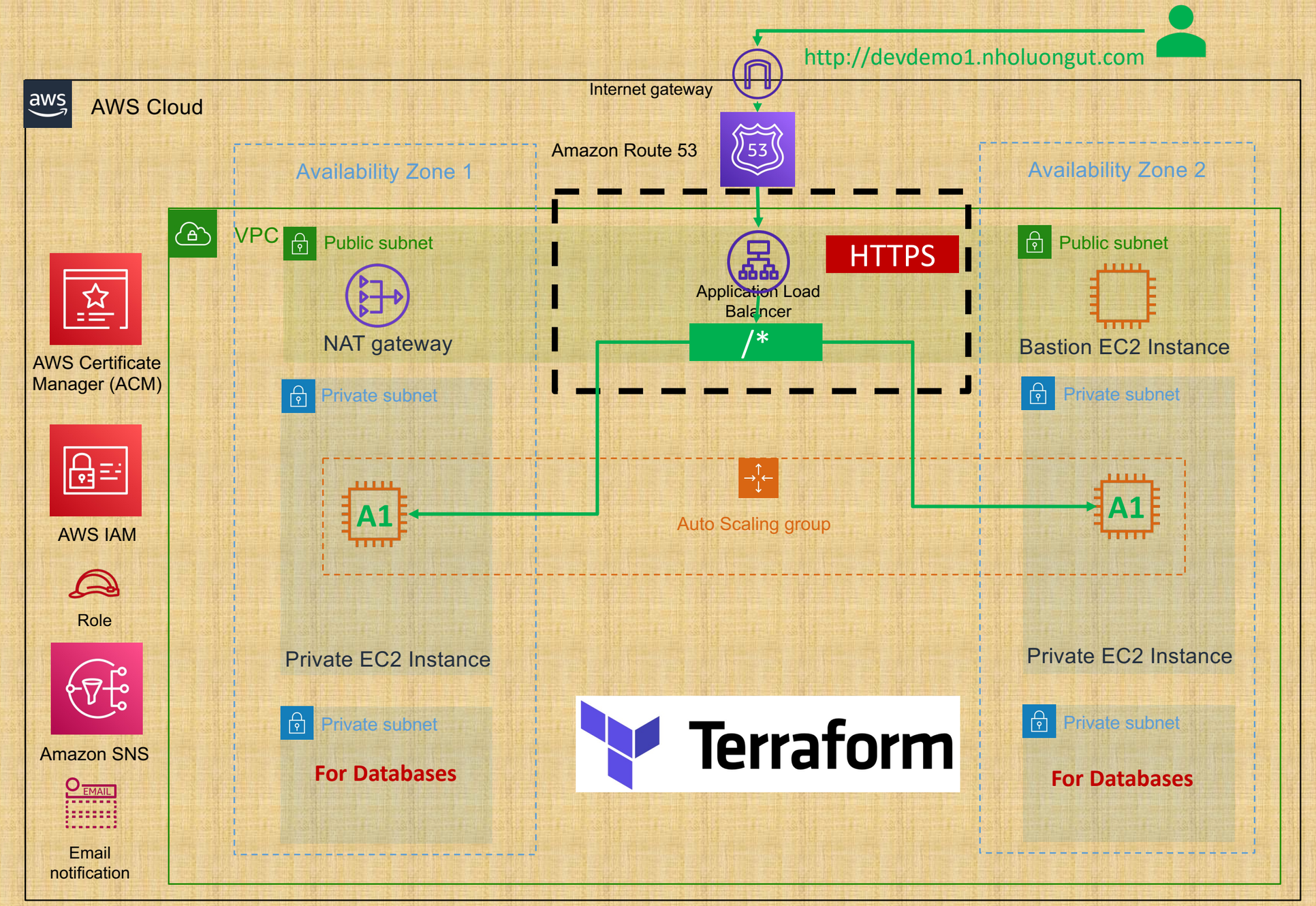
Author: Nho Luong
Skill: DevOps Engineer Lead

# IaC DevOps On AWS



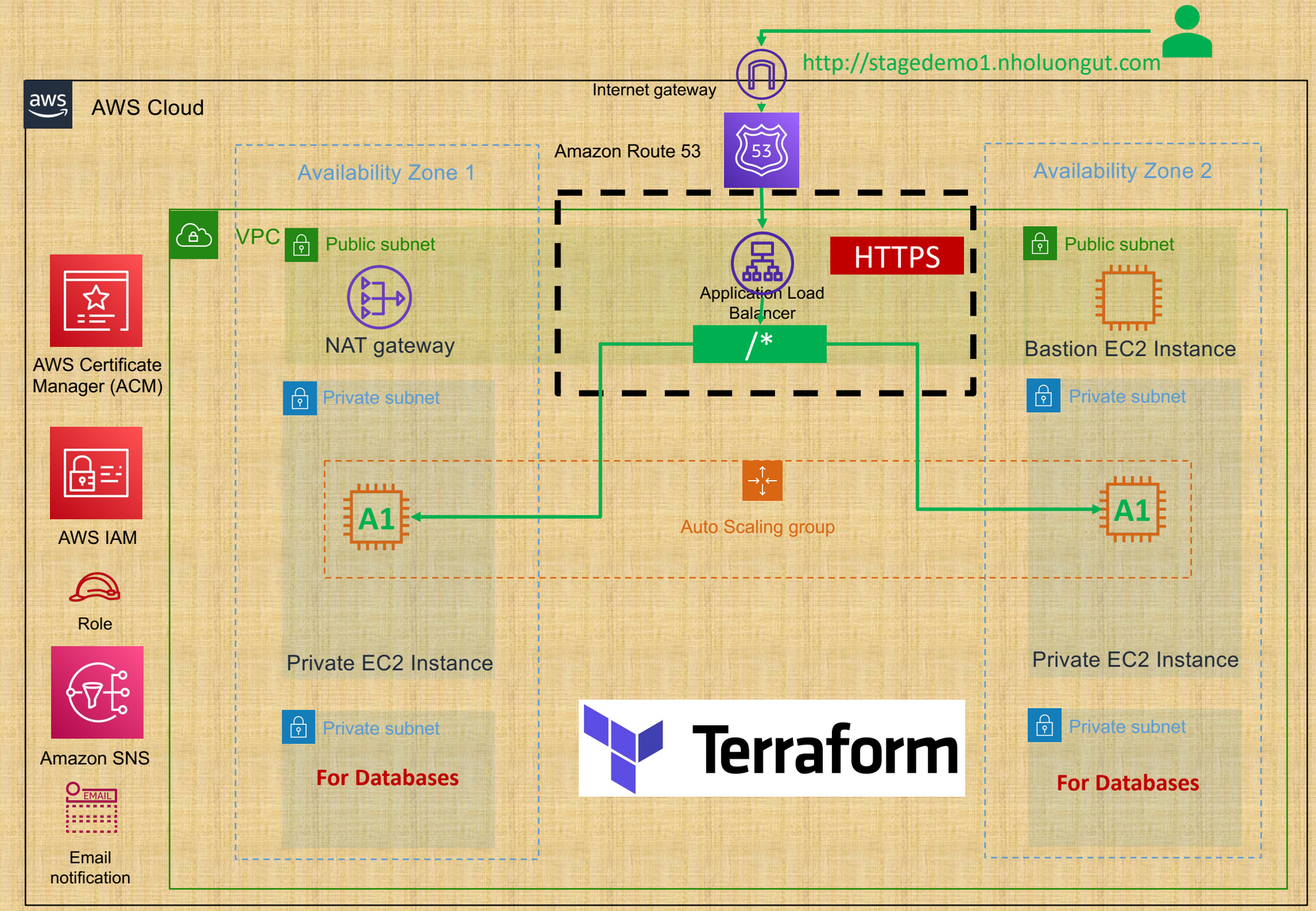

Author: Nho Luong
Skill: DevOps Engineer Lead

# Dev Environment



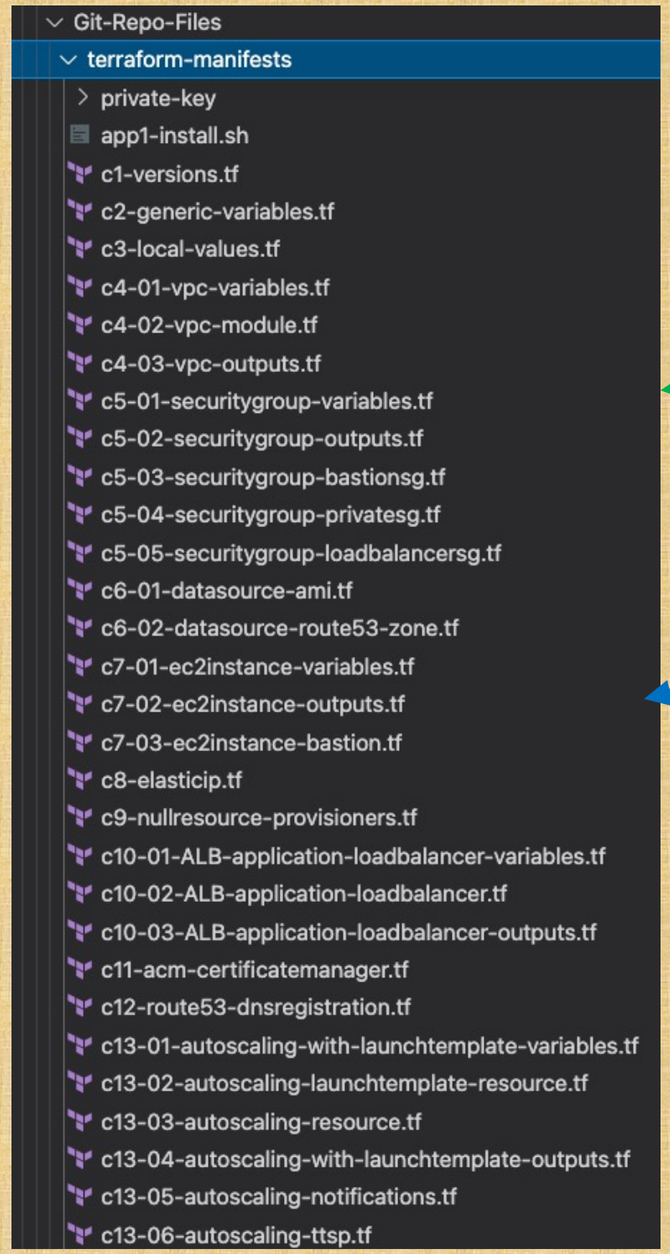

Author: Nho Luong
Skill: DevOps Engineer Lead

# Staging Environment




Author: Nho Luong
Skill: DevOps Engineer Lead

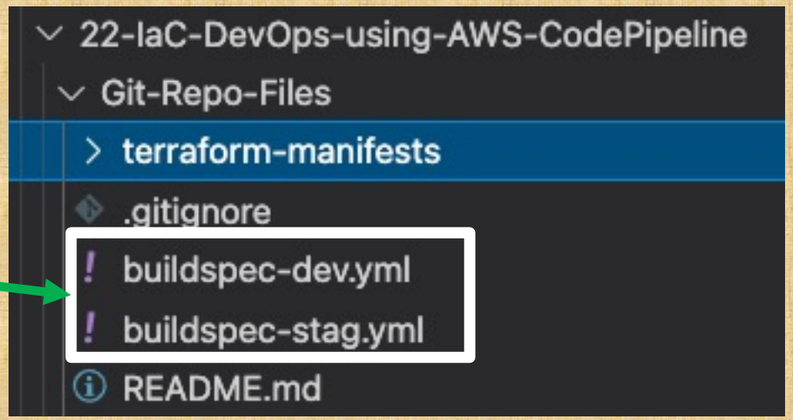
# What are we going to learn ?




We are going to use the TF Configs from Section-15 Autoscaling with Launch Templates.

We are going to make many changes to all these TF Configs to support multiple environments like Dev and Staging

Dev and Staging CodePipeline Build Specification Files




Author: Nho Luong
Skill: DevOps Engineer Lead

# Terraform Modules upgraded to Latest Version

| Module Type | Previous Version | New Version | Impact Analysis |
|---|---|---|---|
| AWS VPC Terraform Module | V2.78.0 | V5.4.0 | Low Impact |
| AWS Security Group Terraform Module | V3.18.0 | V5.1.0 | Low Impact |
| AWS EC2 Instance Module | V2.17.0 | V5.6.0 | Very High Impact |
| AWS Application Load Balancer Terraform Module | V5.16.0 | V9.4.0 | Very High Impact |
| AWS ACM Certificate Manager Terraform Module | V2.14.0 | V5.0.0 | Low Impact |
| AWS RDS Database Terraform Module | V3.0.0 | V6.0.0 | Low Impact |

Author: Nho Luong
Skill: DevOps Engineer Lead

# Thank You

Author: Nho Luong
Skill: DevOps Engineer Lead