

Terraform on AWS with IaC DevOps & SRE

Kalyan Reddy Daida

AWS EKS
Kubernetes

Azure AKS
Kubernetes

AWS ECS
Docker on AWS

AWS
CloudFormation

Google GKE
Kubernetes

AWS
Lambda & Serverless

STACKSIMPLIFY

DevOps & SRE Roadmap

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

Terraform on AWS with IaC DevOps & SRE

Kalyan Reddy Daida

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

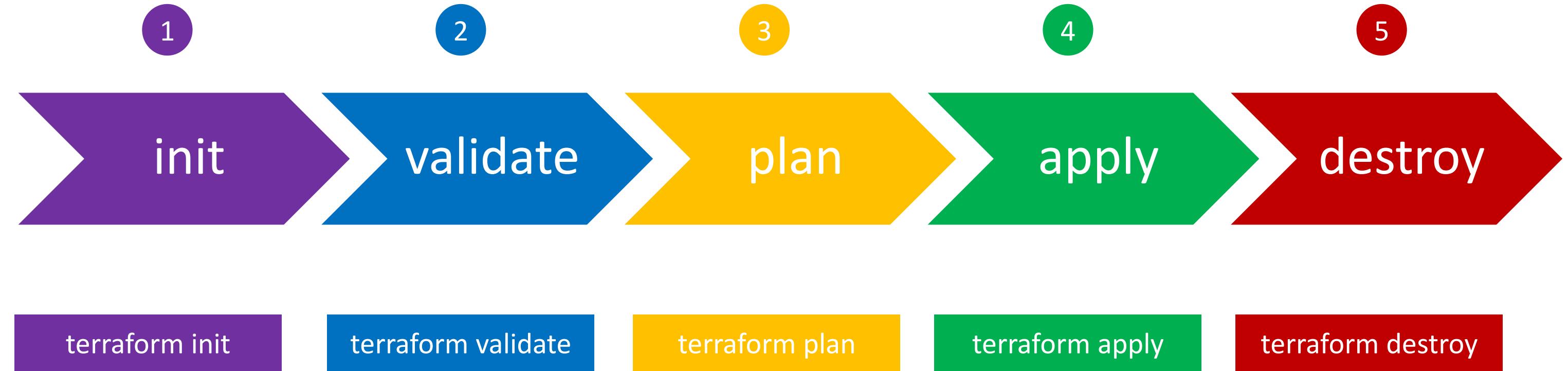
Terraform Local Modules – Build from scratch

Terraform Remote State Storage

Terraform Remote State Datasource

IaC DevOps with AWS CodePipeline for Terraform Project

Terraform Workflow



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.

Terraform Block

Providers Block

Resources Block

Fundamental Blocks

Input Variables Block

Output Values Block

Local Values Block

Variable Blocks

Data Sources Block

Modules Block

Calling / Referencing Blocks

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

Utility Project
with
Datasources

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

AWS VPC 3-Tier Architecture

Build **manually** using AWS Mgmt Console

Build same using **Terraform**

Terraform Modules

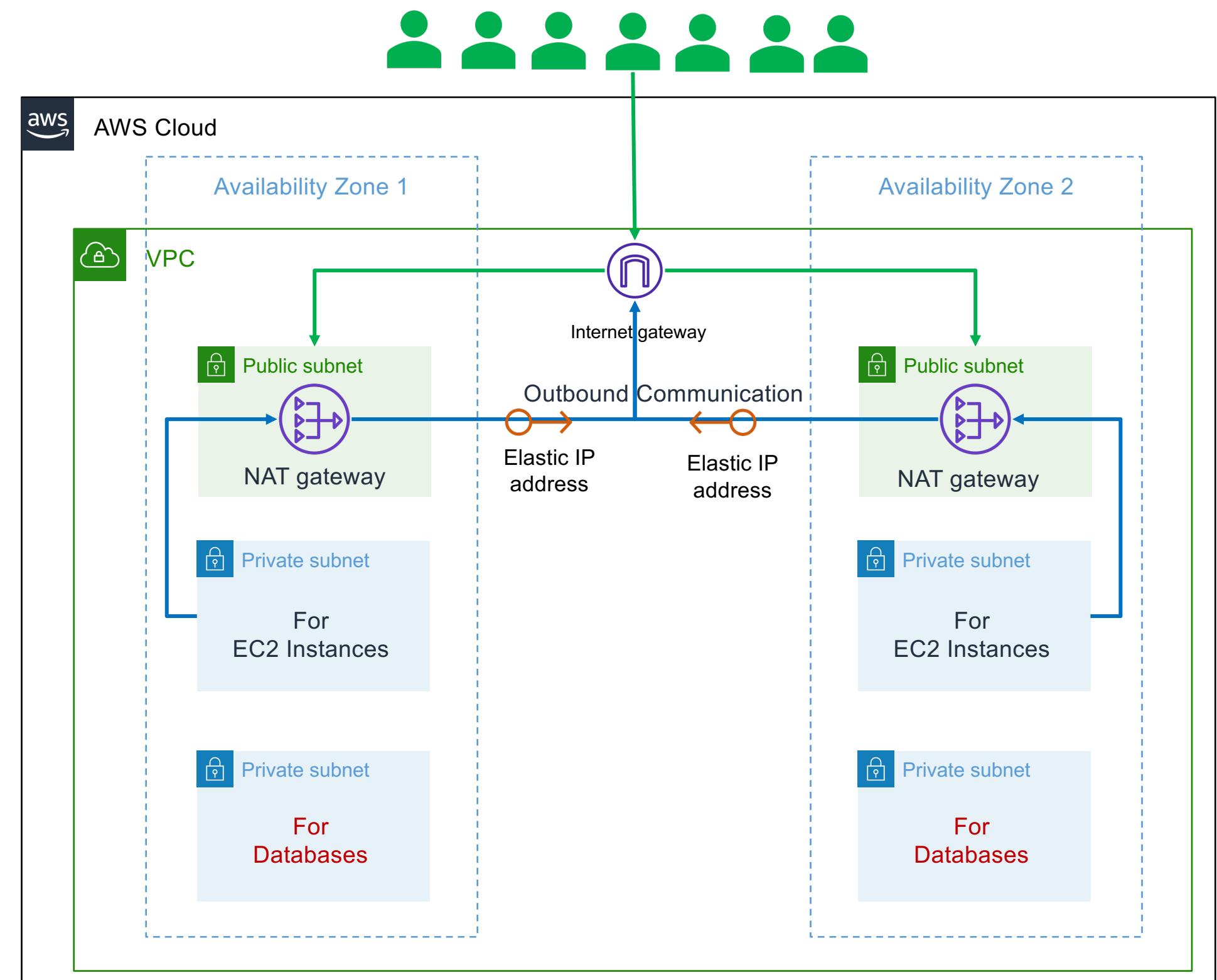
Terraform Local Values

Terraform Variables – `terraform.tfvars`

Terraform Variables – `vpc.auto.tfvars`

Terraform Version Constraints

Terraform Code Organizing – **Production Grade style**



AWS VPC + EC2 Instance + Security Groups



Terraform & AWS Concepts

Terraform Module: VPC

Terraform Module: Security Group

Terraform Module: AWS EC2 Instance

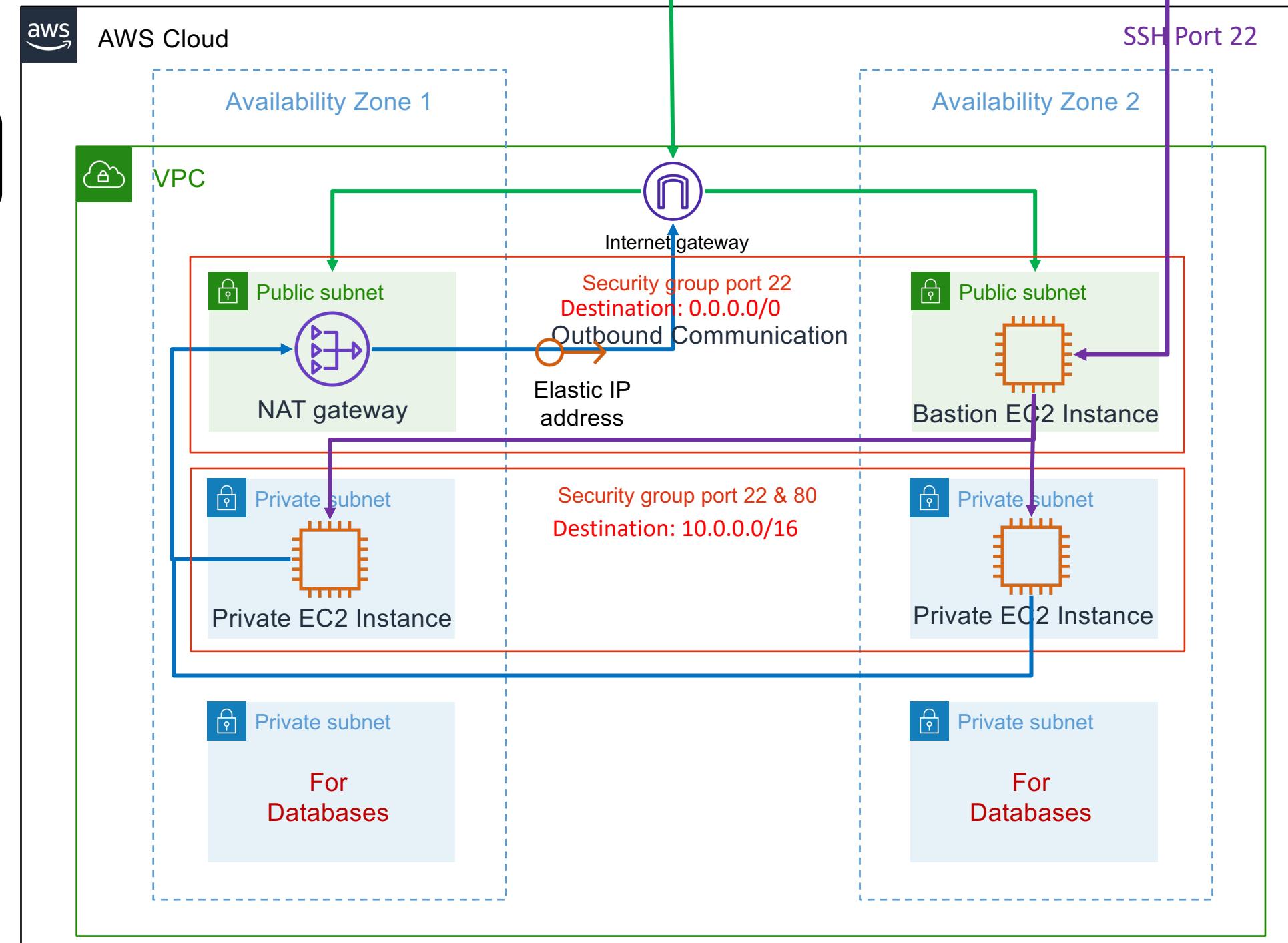
Meta-Argument: `depends_on`

Terraform `null_resource`

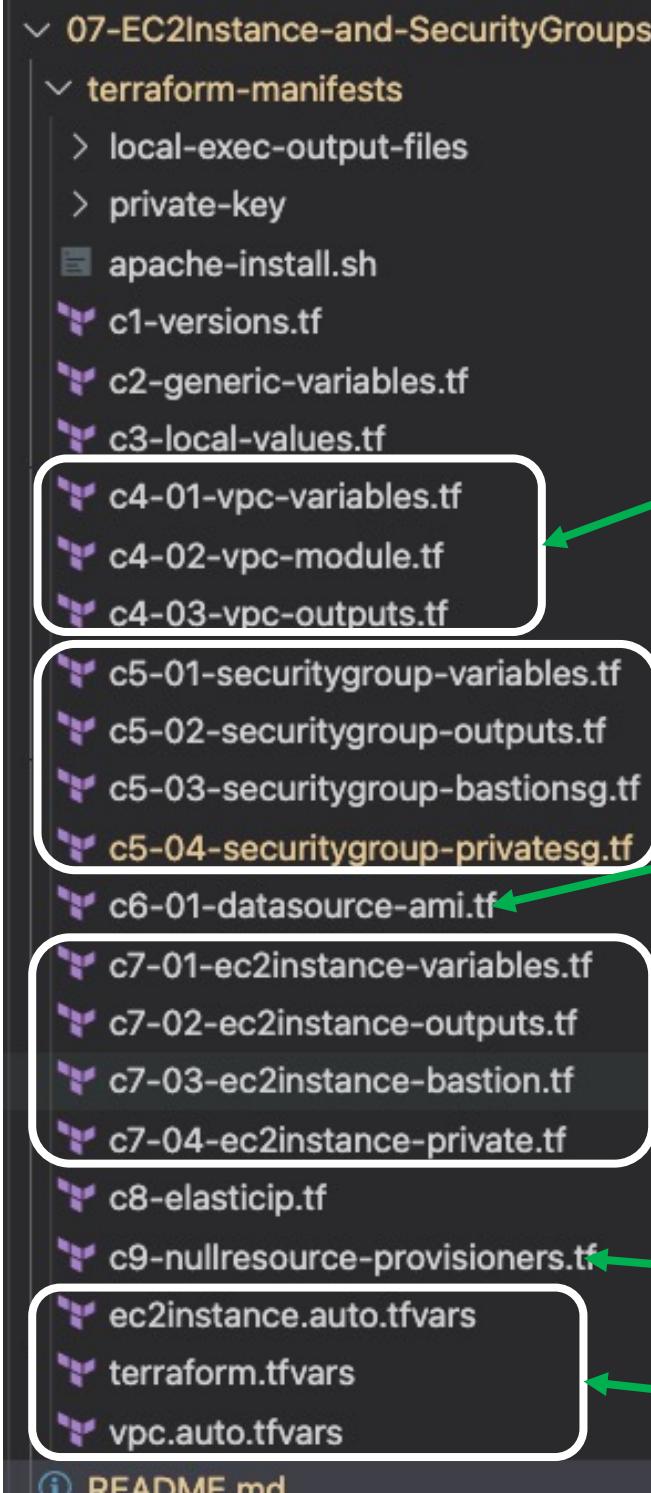
Terraform `File` Provisioner

Terraform `Remote-exec` Provisioner

Terraform `Local-exec` Provisioner



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

Variables

File Provisioner

Remote-exec
Provisioner

Local-exec
Provisioner

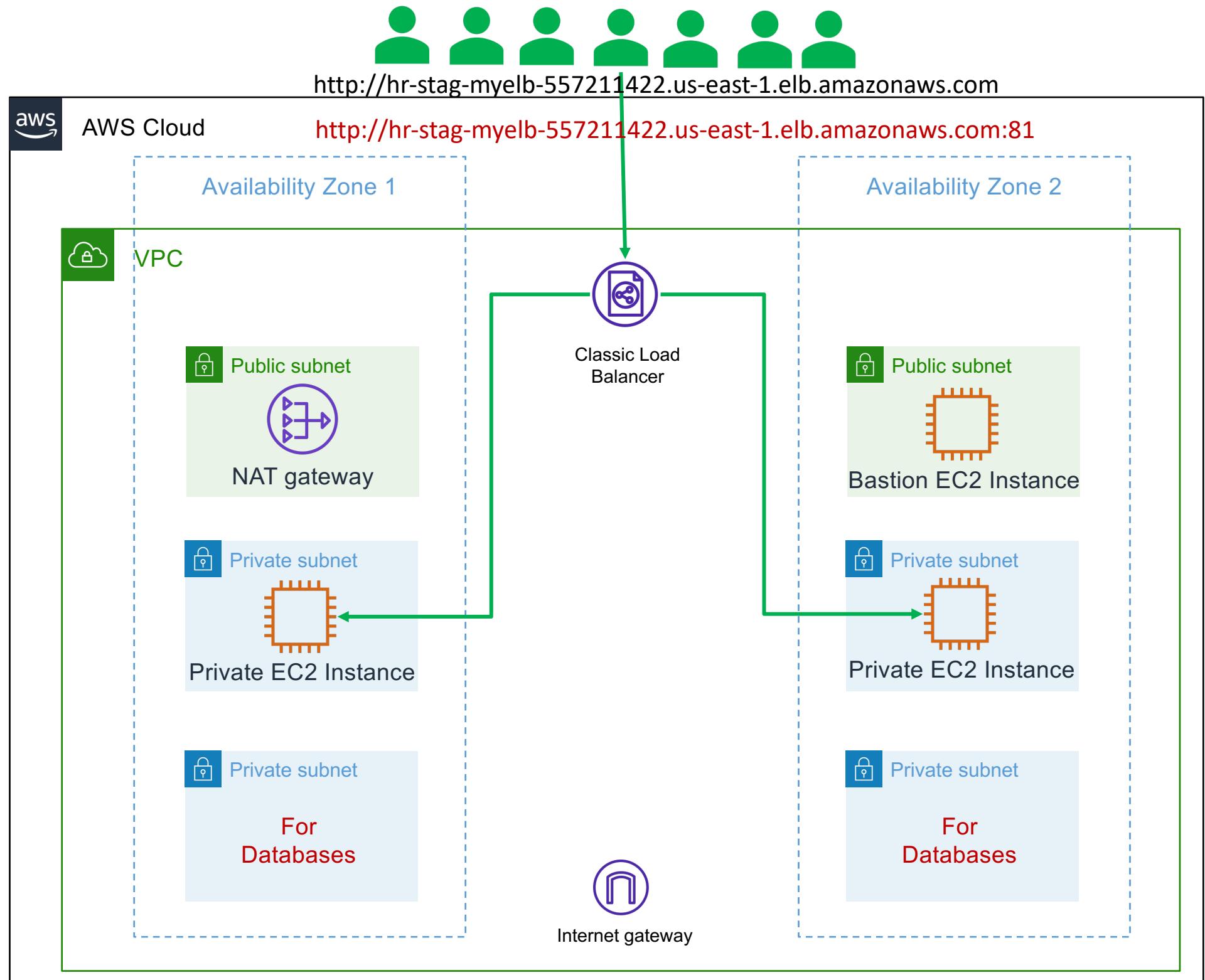
AWS VPC + EC2 Instance + Security Groups + AWS ELB Classic Load Balancer

Terraform & AWS Concepts

Terraform Module: ELB Classic LB

Terraform Module: Security Group

Add Custom SG Rule for Inbound Port 81



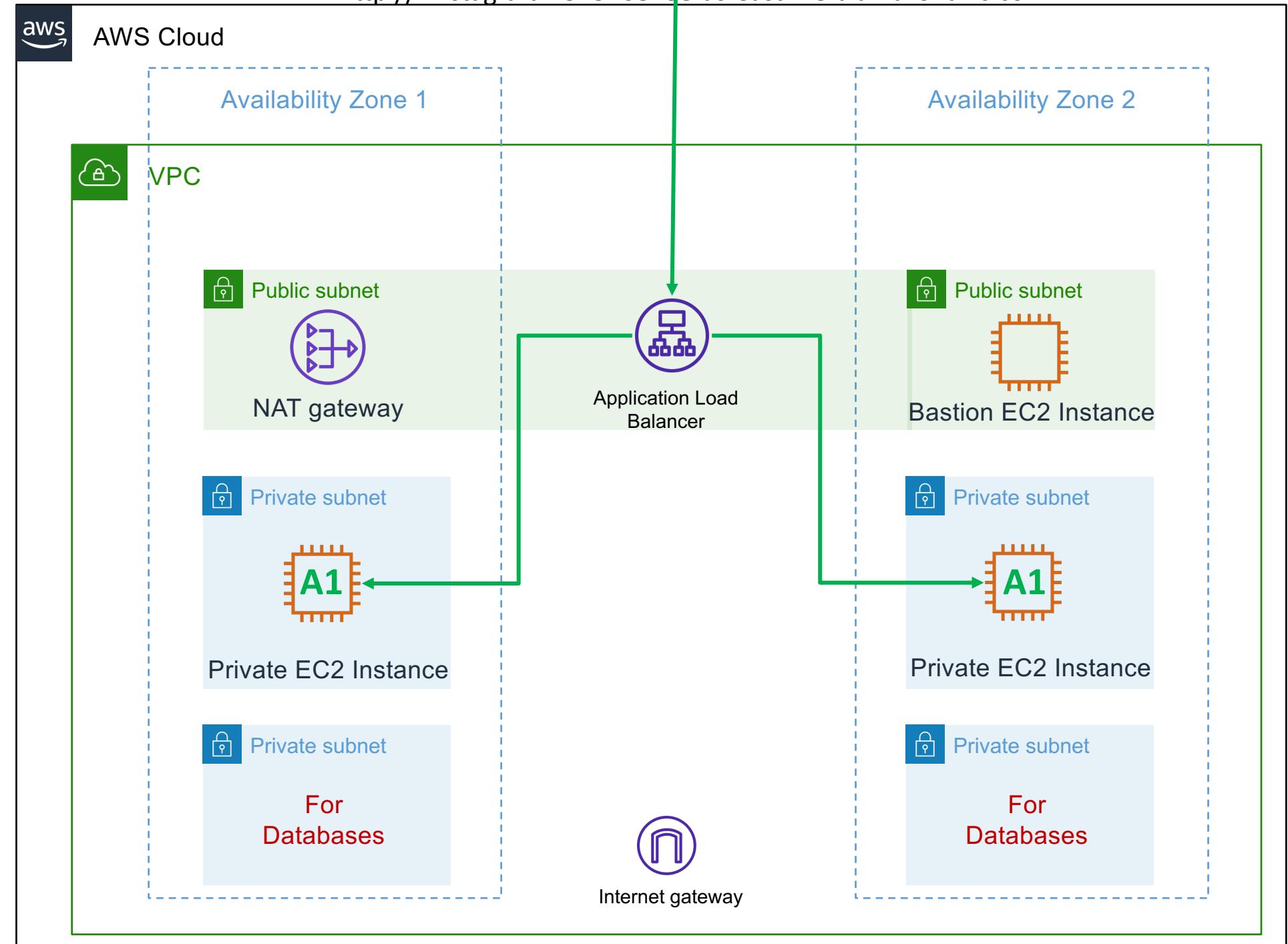
AWS VPC + EC2 Instance + Security Groups + AWS ALB Application Load Balancer

Terraform & AWS Concepts

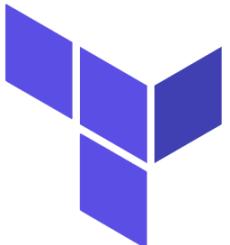
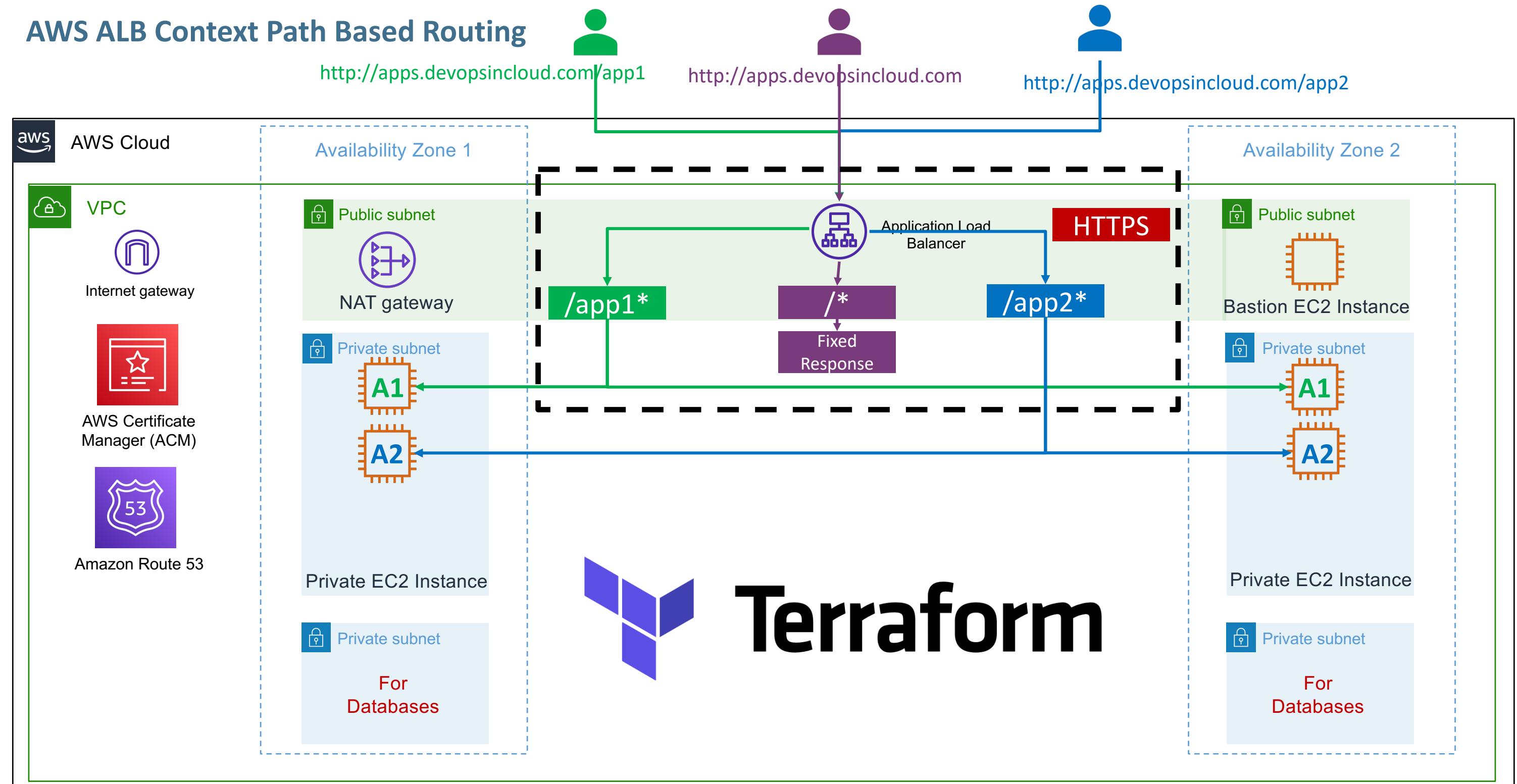
Terraform Module: AWS ALB



<http://hr-stag-alb-1575108738.us-east-1.elb.amazonaws.com>

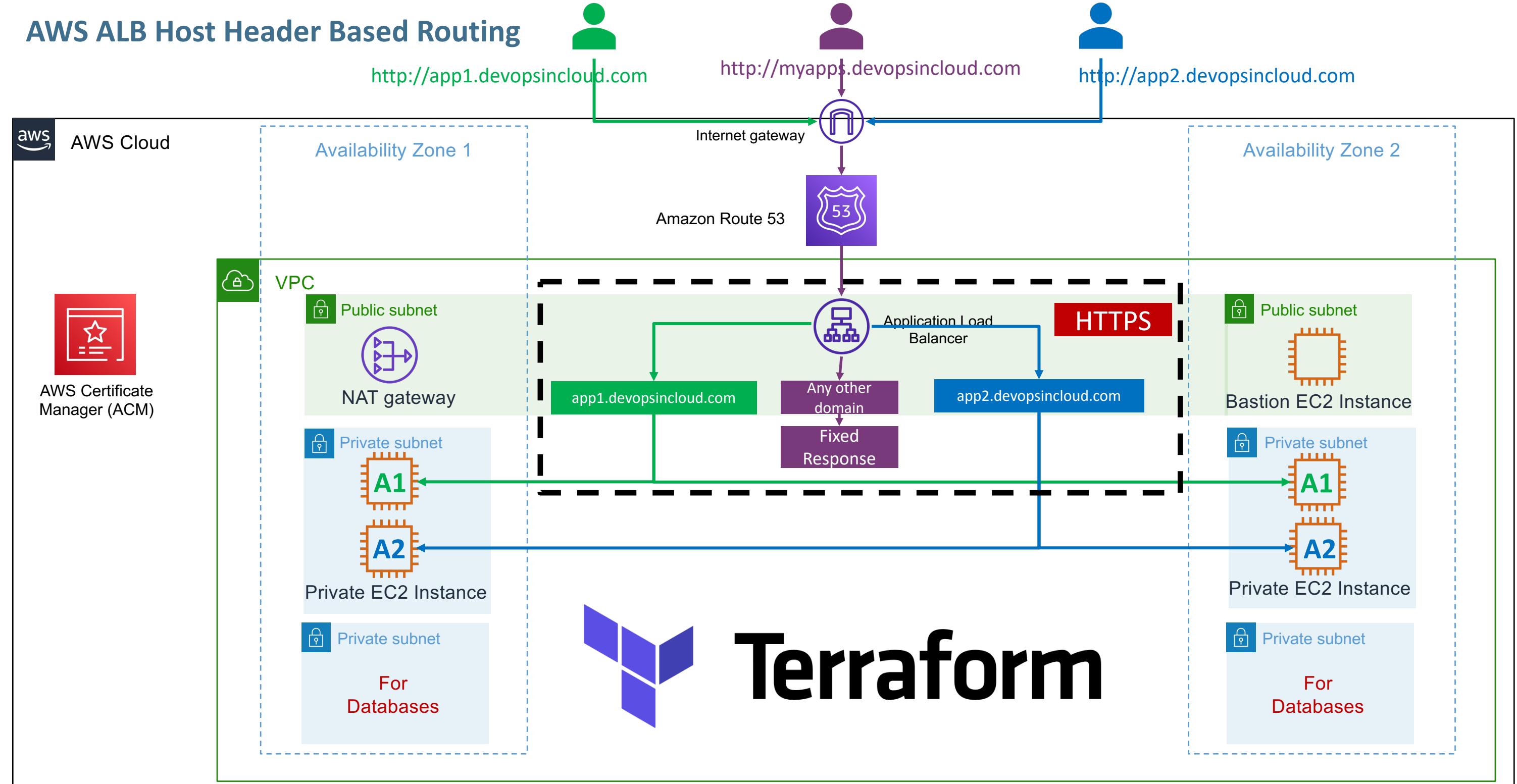


AWS ALB Context Path Based Routing



Terraform

AWS ALB Host Header Based Routing



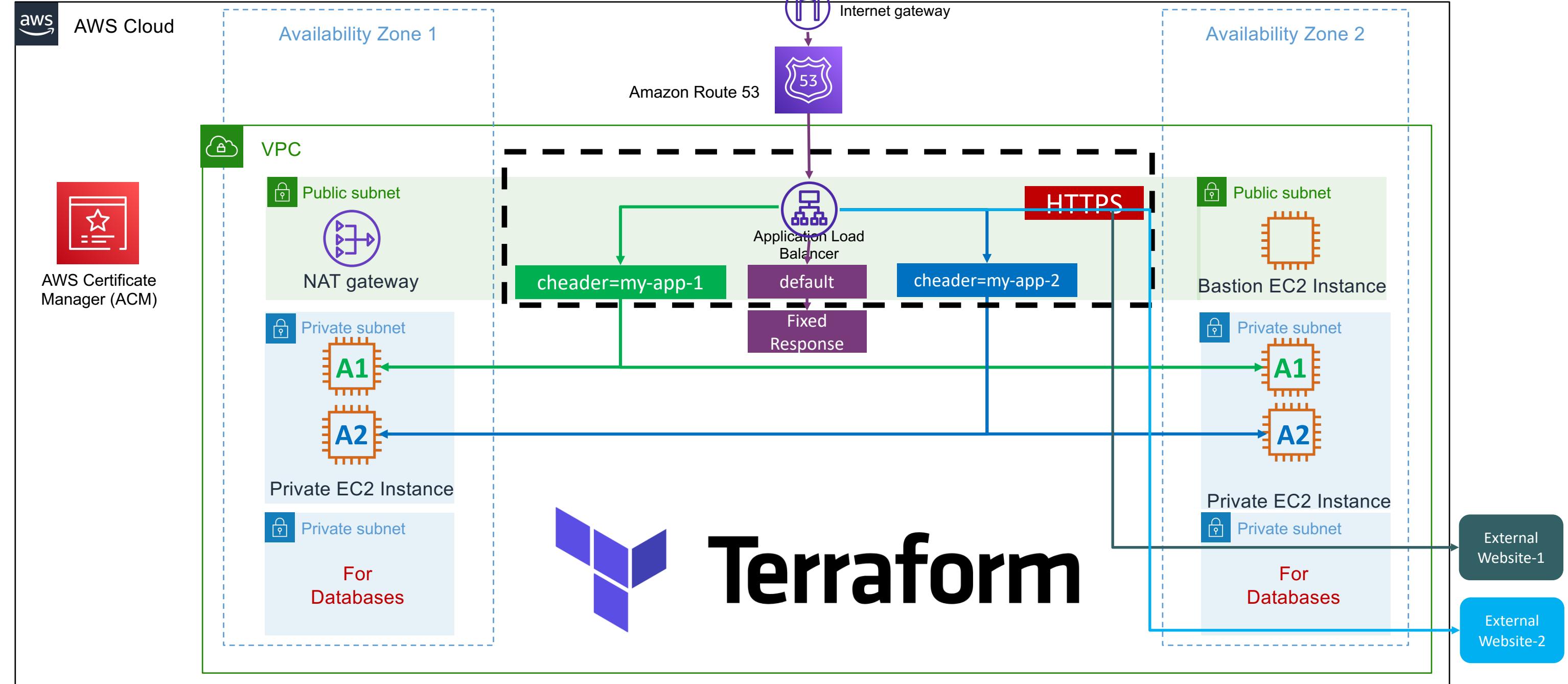
Terraform

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

Host Header External Redirect 302

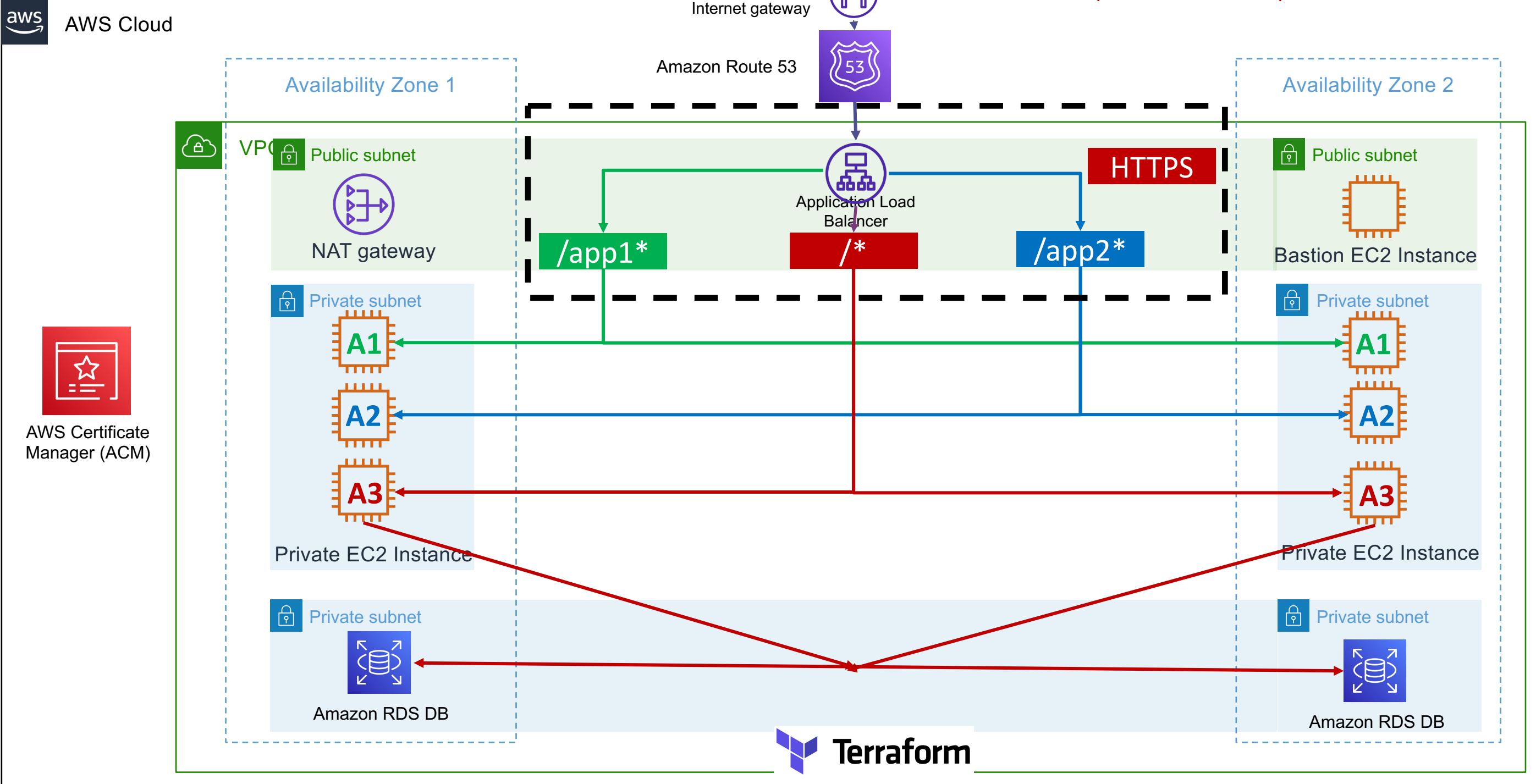
HTTP Header Routing

Query String External Redirect 302



AWS DNS to DB using Terraform

<http://dns-to-db.devopsincloud.com/app1>
<http://dns-to-db.devopsincloud.com/app2>
<http://dns-to-db.devopsincloud.com>



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

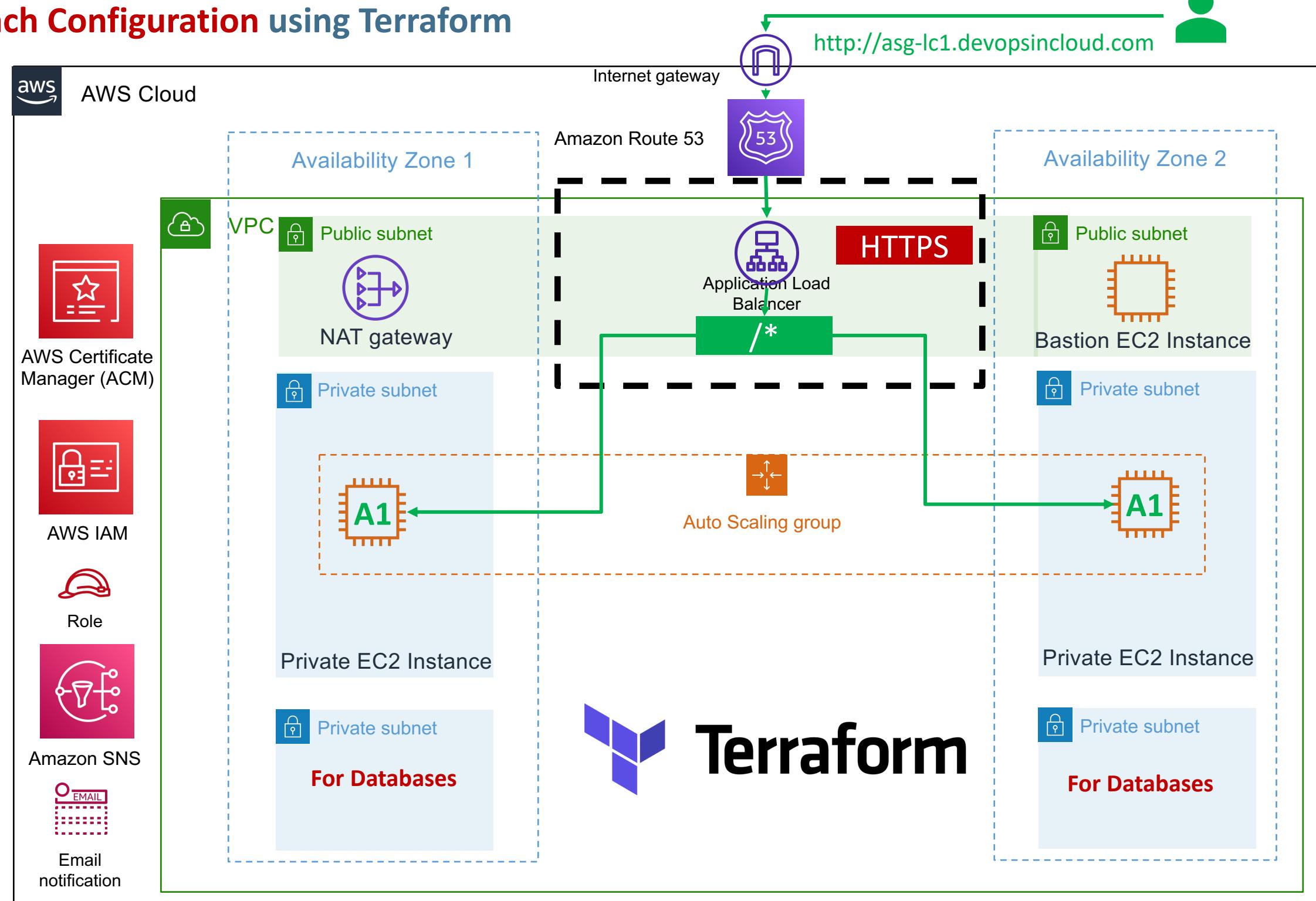
There is a major change in all the modules (except vpc) related to output names of these modules which impacts all these module references wherever you use.

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

AWS Autoscaling with Launch Configuration using Terraform



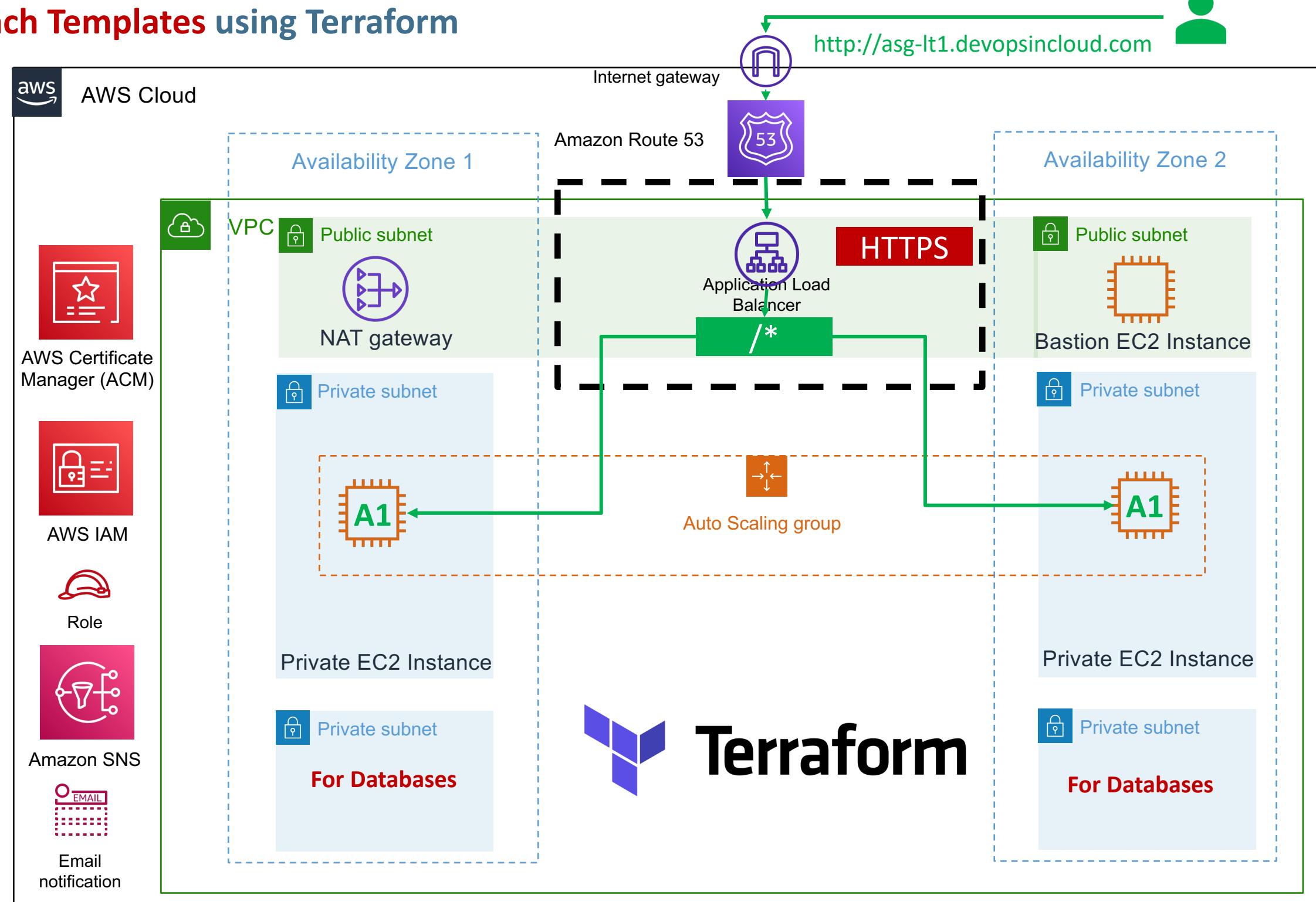
<http://asg-lc1.devopsincloud.com>



AWS Autoscaling with **Launch Templates** using Terraform



<http://asg-lt1.devopsincloud.com>



Terraform ASG Resource

ASG with Launch Templates

ASG Instance Refresh

ASG Lifecycle Hooks

ASG TTSP

ASG Scheduled Actions

ASG Notifications

ASG Autoscaling Tests

AWS Network Load Balancer with TCP & TLS Listeners using Terraform

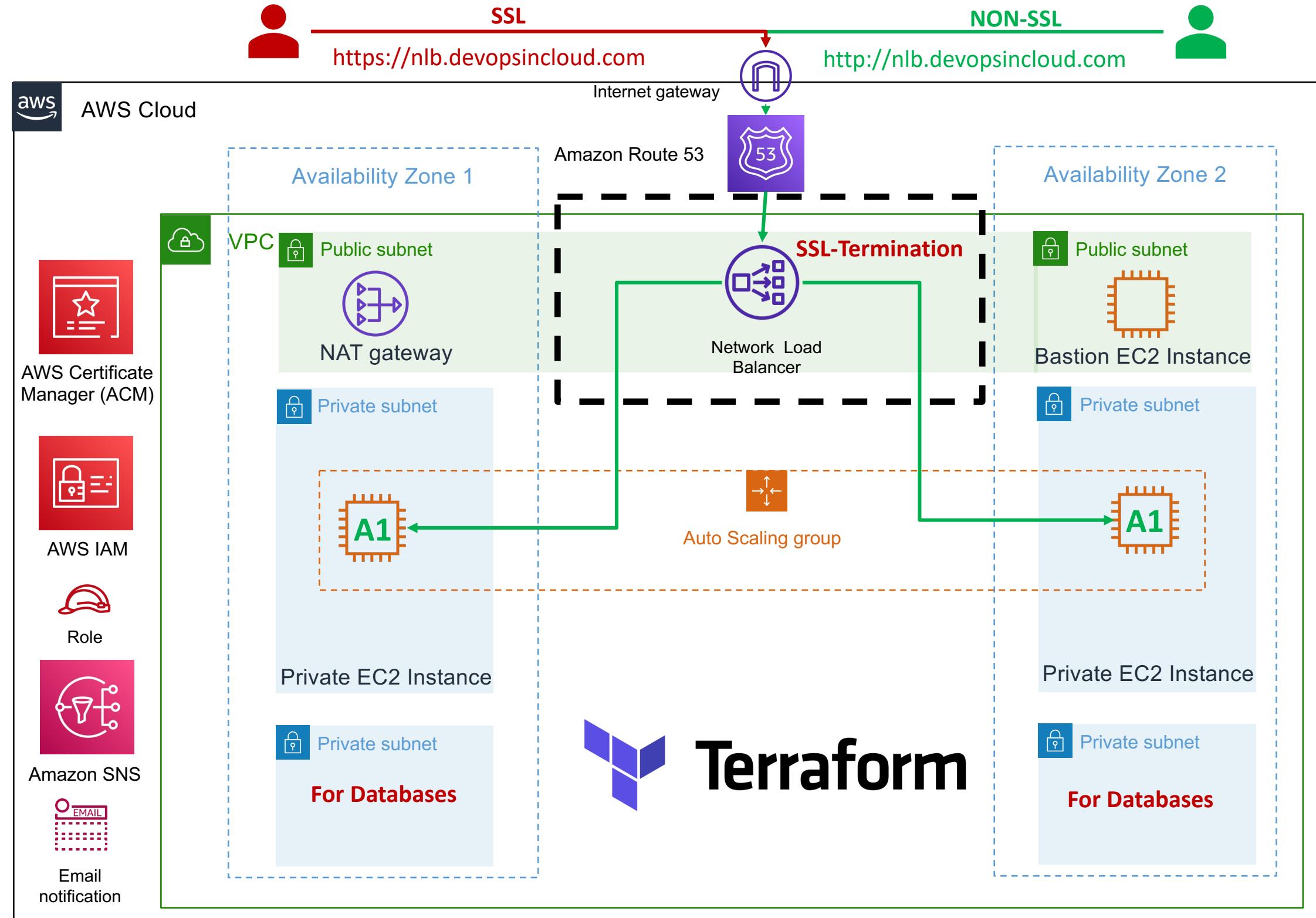
Terraform NLB Module

Terraform NLB TCP Listeners

Terraform NLB TLS Listeners

Terraform NLB Target Groups

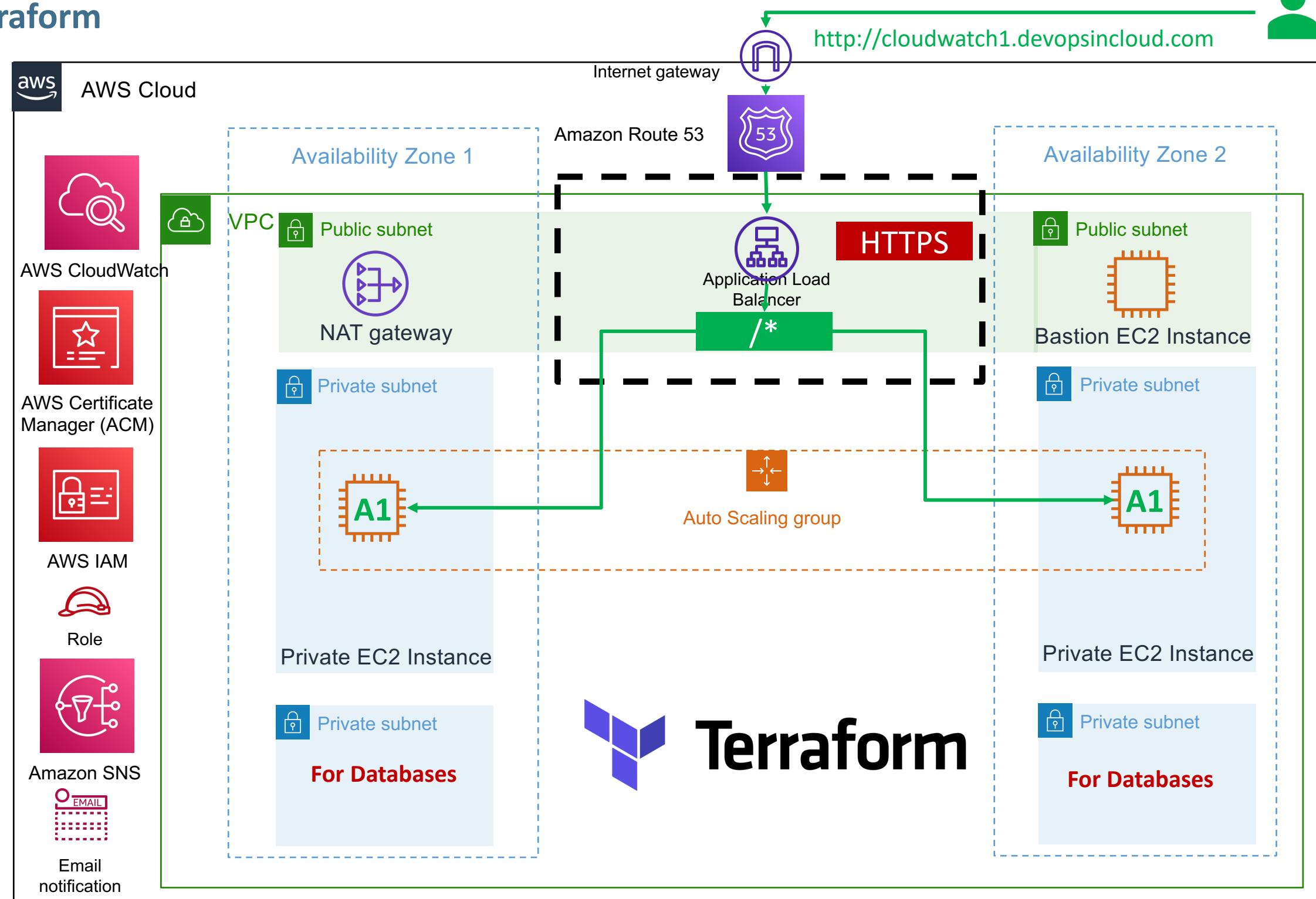
Terraform NLB with EC2 Autoscaling Groups



AWS CloudWatch using Terraform



<http://cloudwatch1.devopsincloud.com>



Build Terraform Modules Locally

Build from Scratch

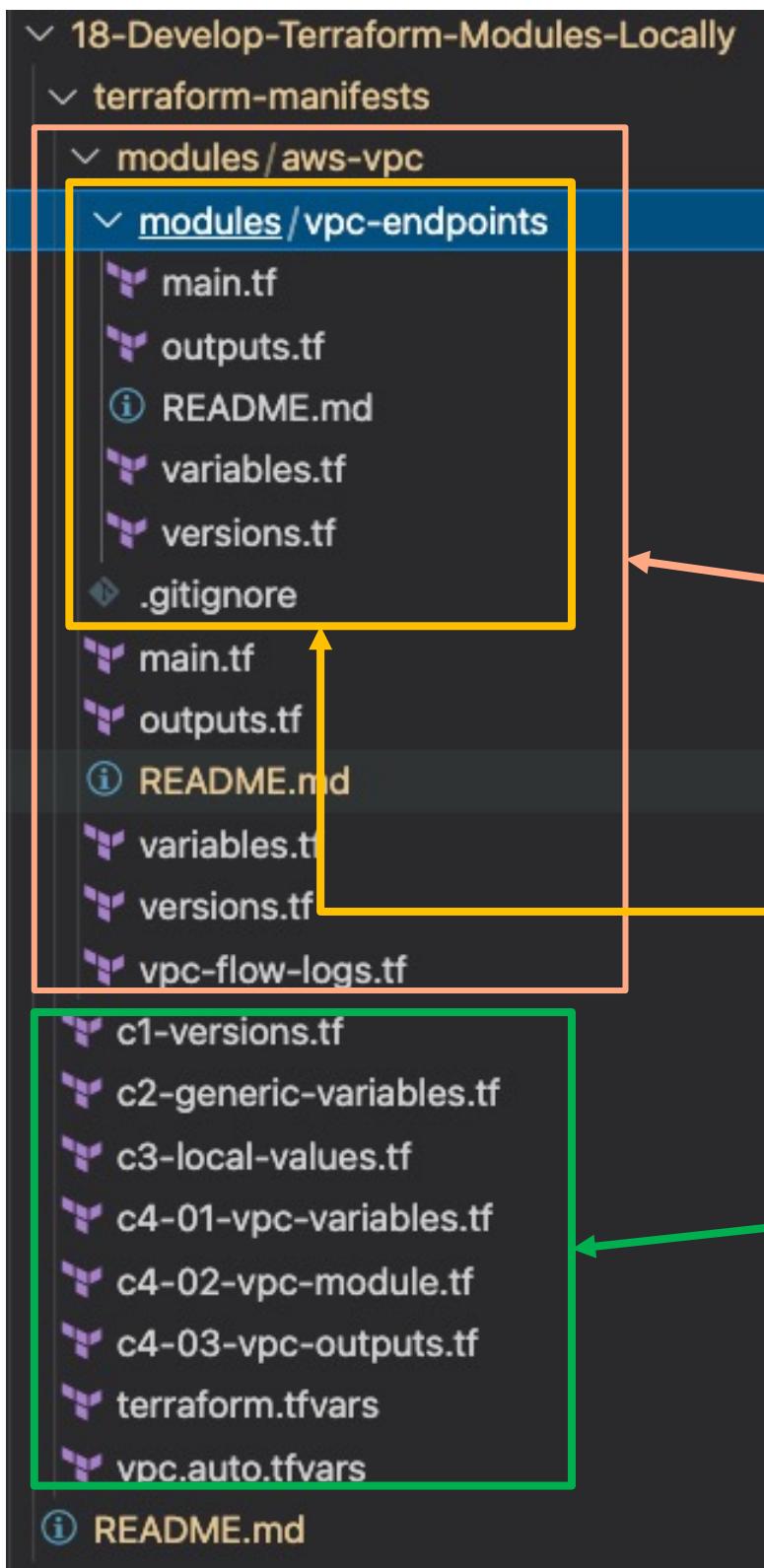
If already not available, then use this approach

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



What are we going to learn ?

AWS VPC Module

AWS VPC Modules - VPC Endpoints Sub Module

Root Module calling the VPC Local Module

Terraform State

Terraform
Local
State
Storage

Terraform
Remote
State
Storage

What is Terraform Backend ?

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

Terraform
State Storage



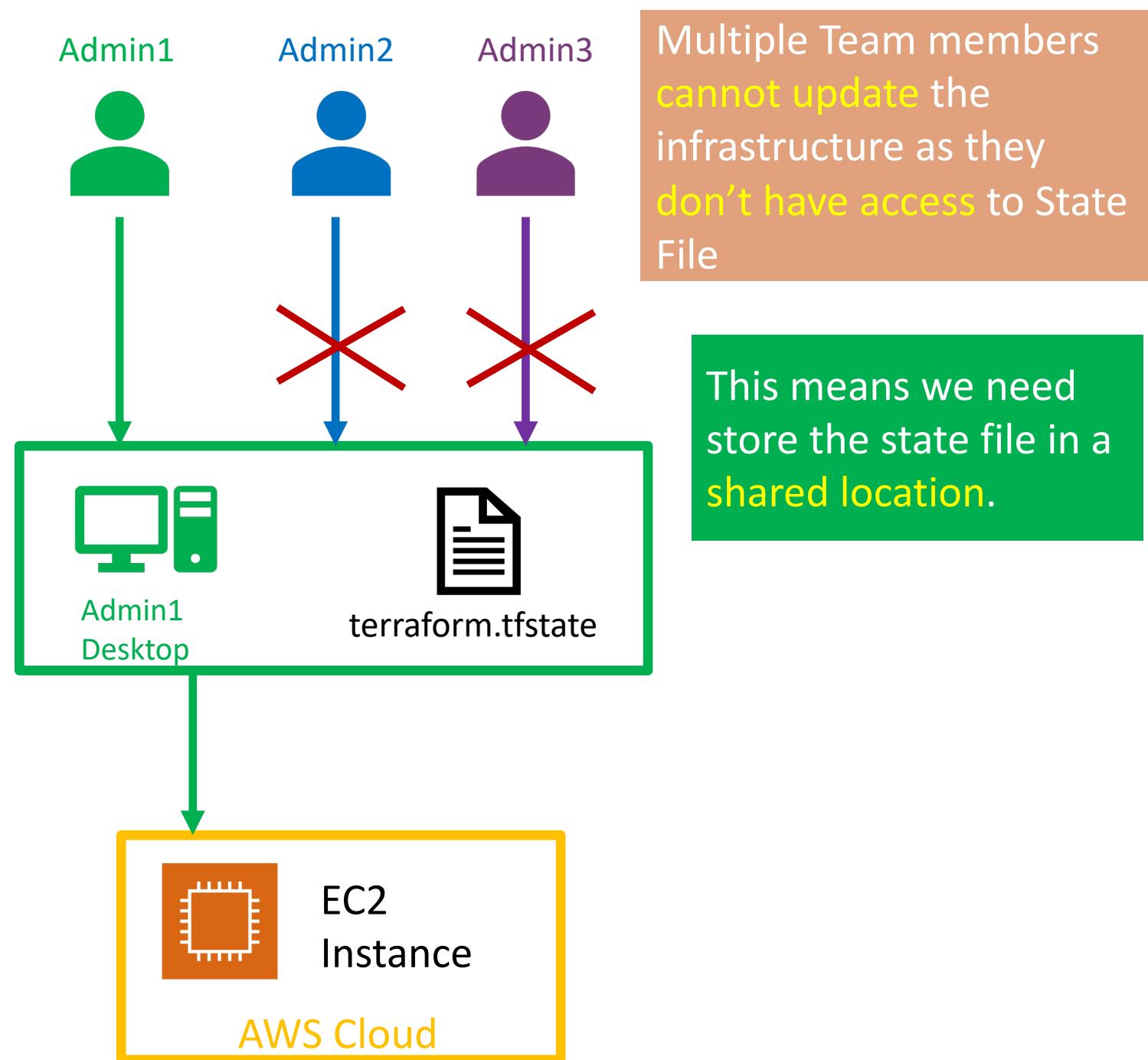
AWS S3 Bucket

Terraform
State Locking

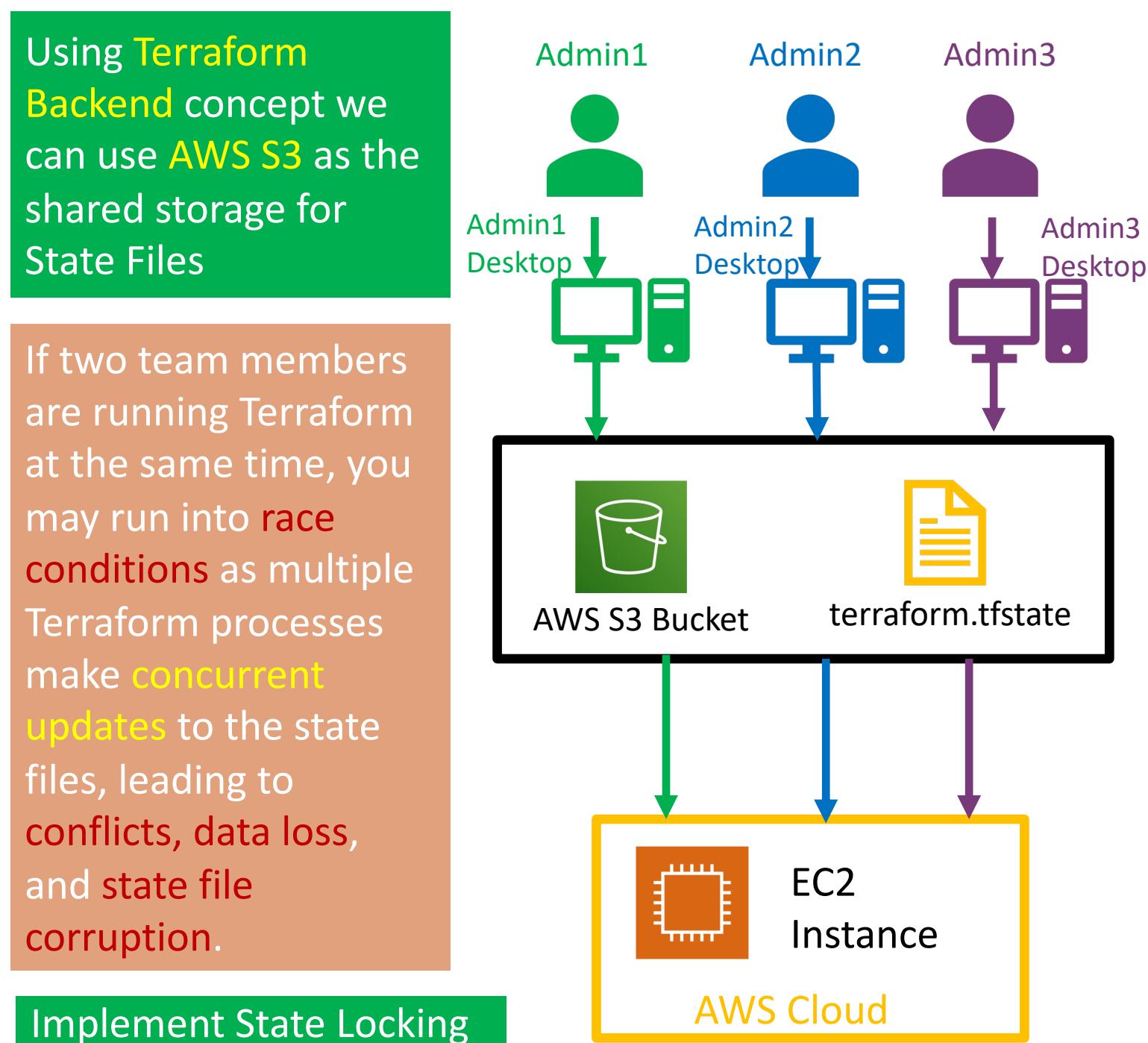


AWS DynamoDB

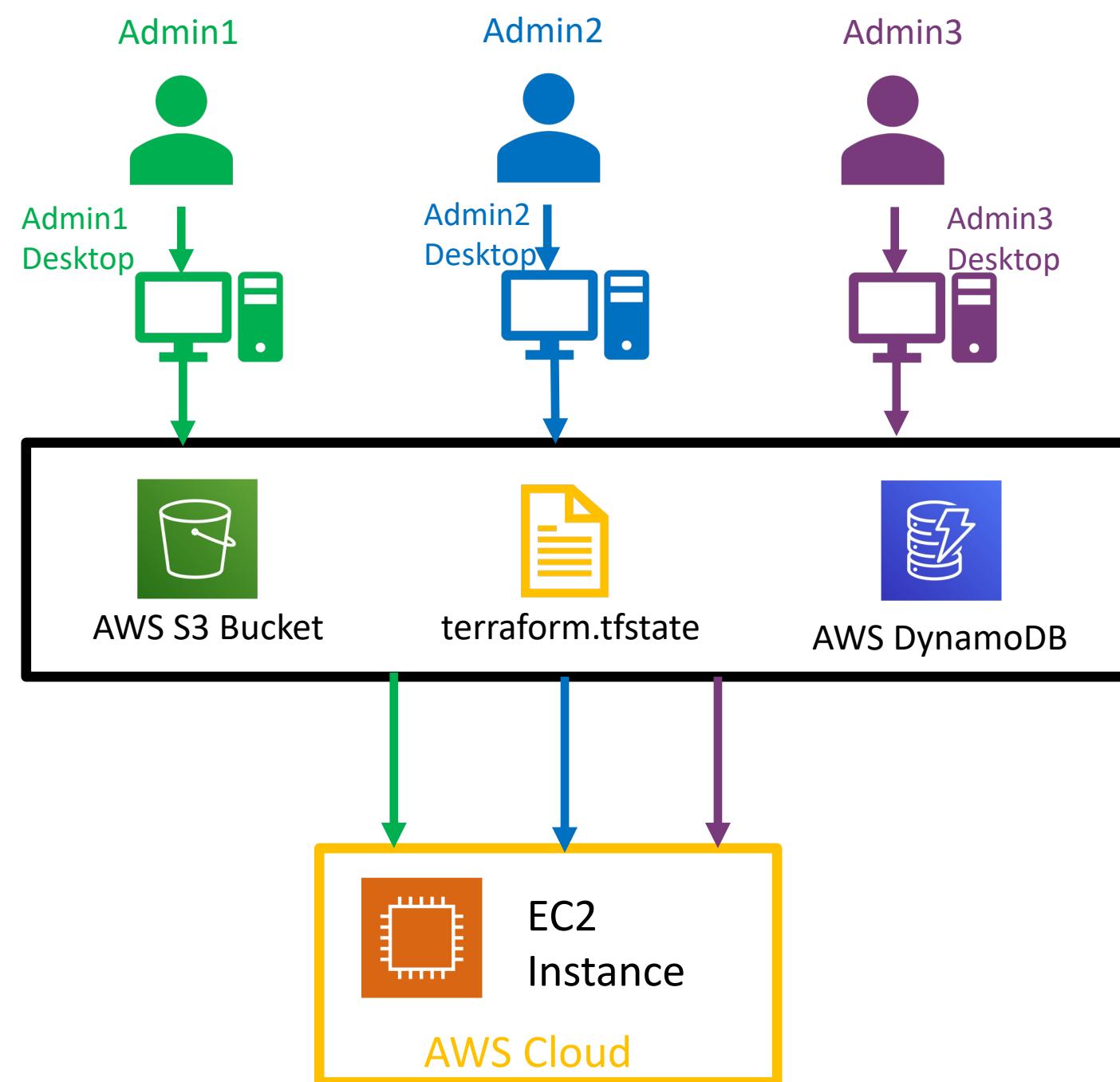
Local State File



Remote State File



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**.

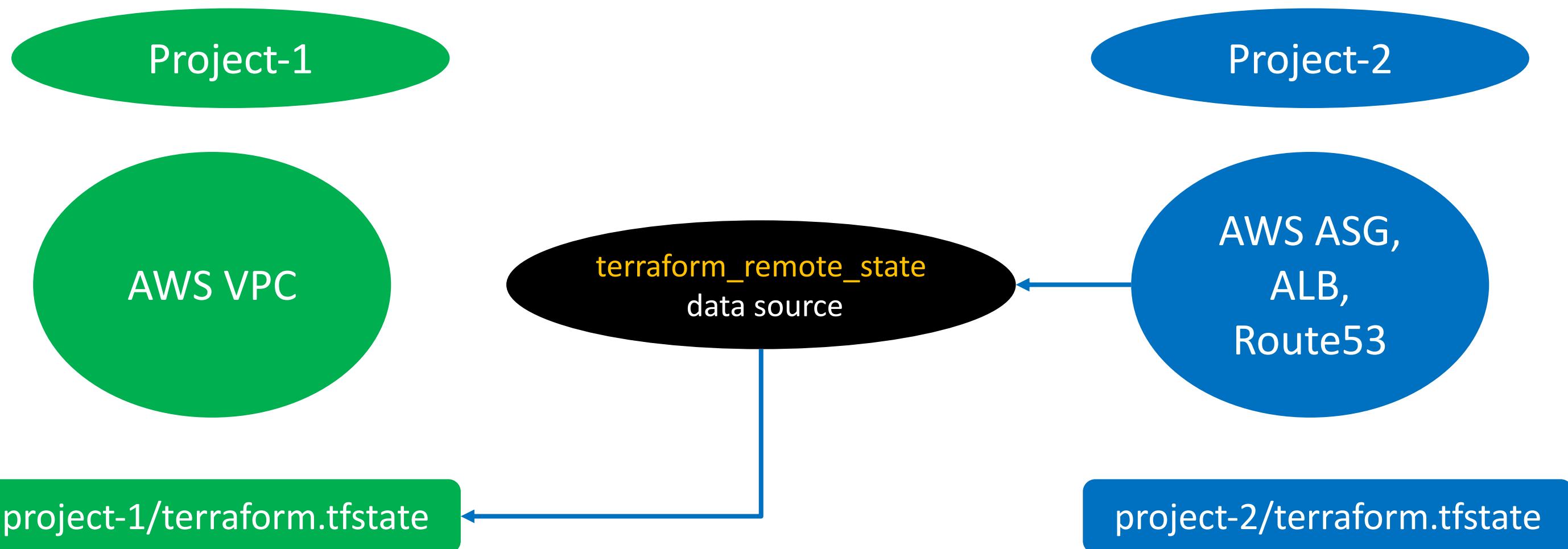
If acquiring the lock is taking **longer** than expected, Terraform will output a **status message**.

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.

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.



Terraform Remote State Datasource



<http://tfremoteds.devopsincloud.com>

Project-1

VPC Resources

Project-2

Autoscaling Groups

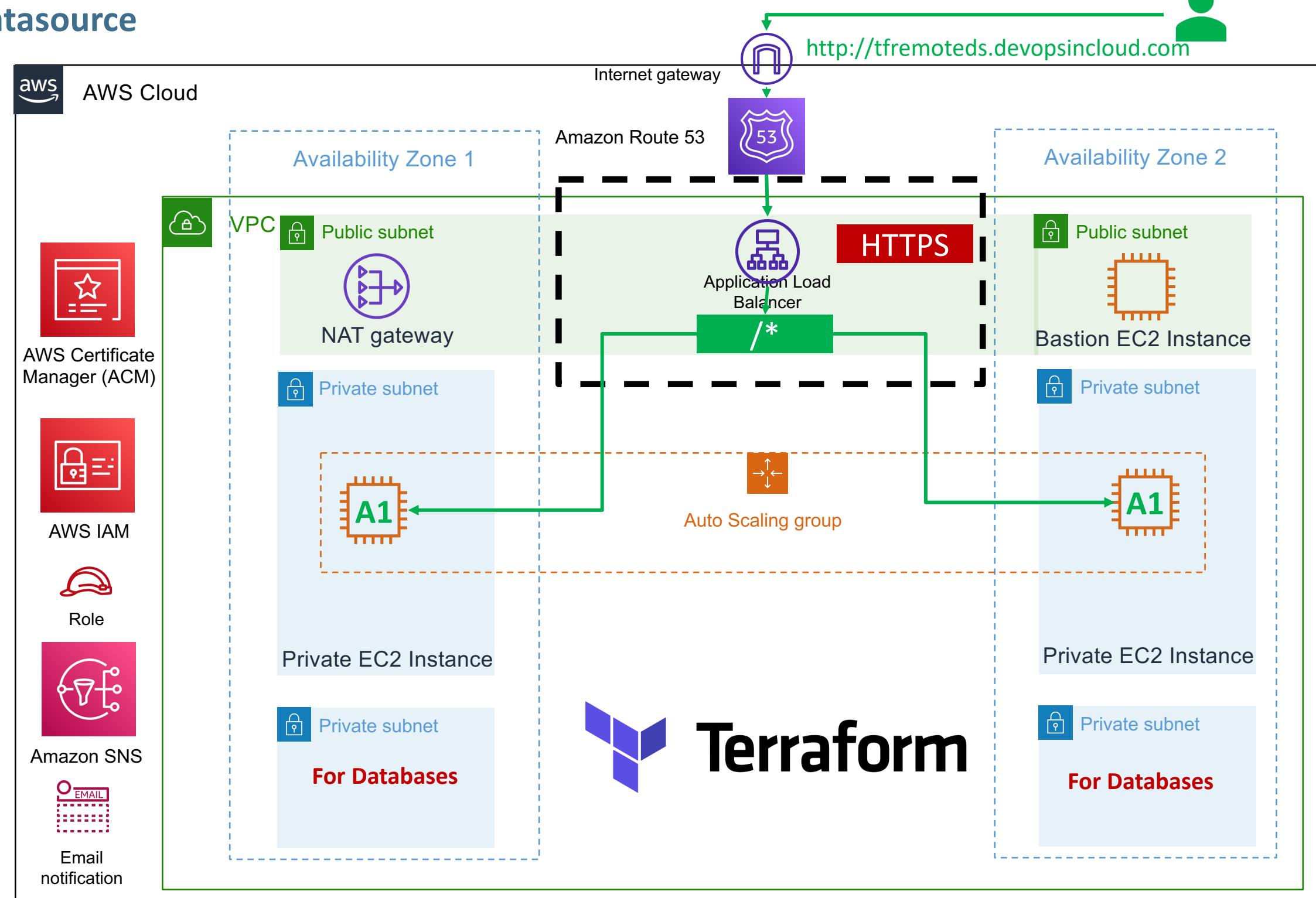
Application Load Balancers

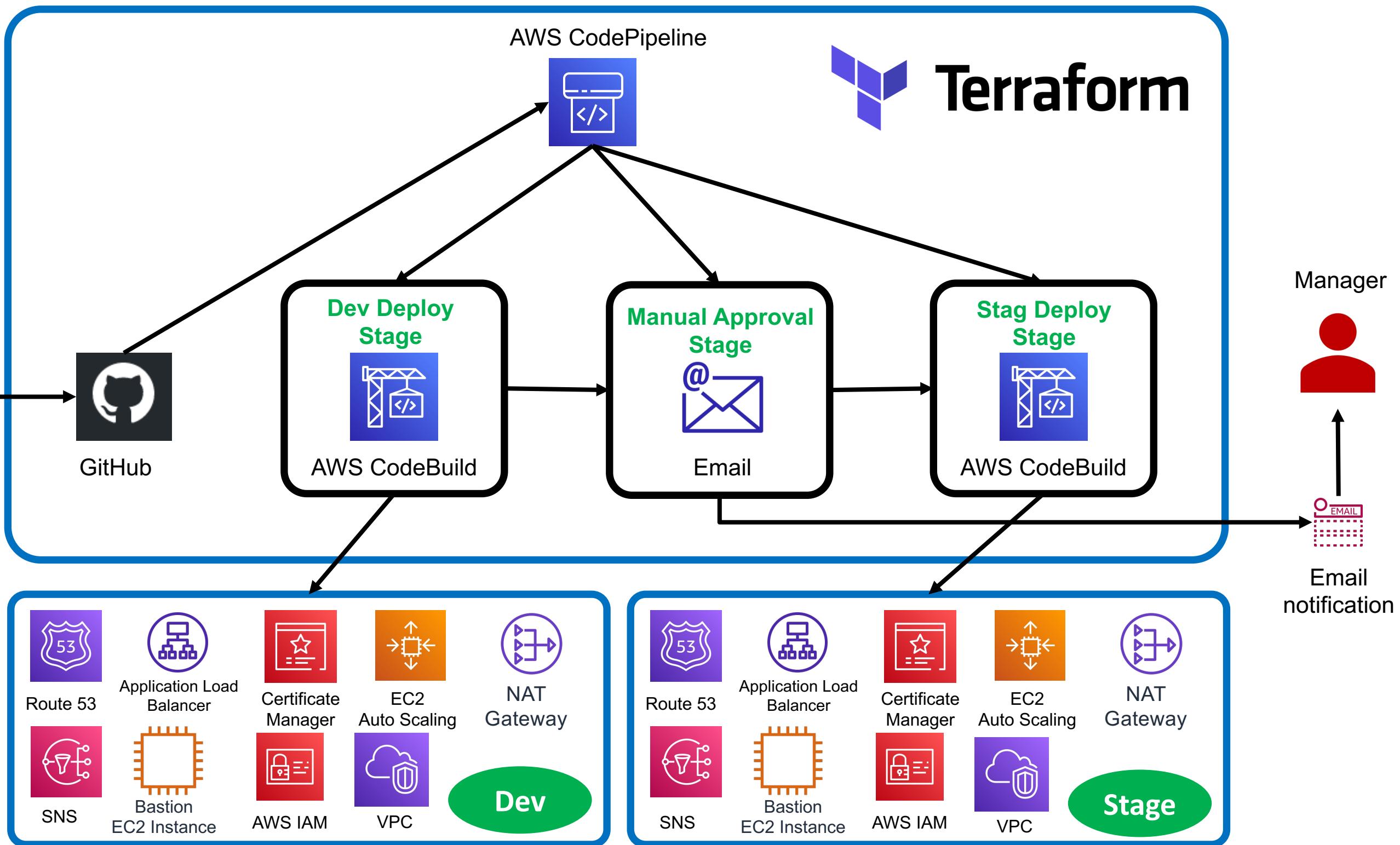
AWS Route53

AWS Certificate Manager

AWS SNS

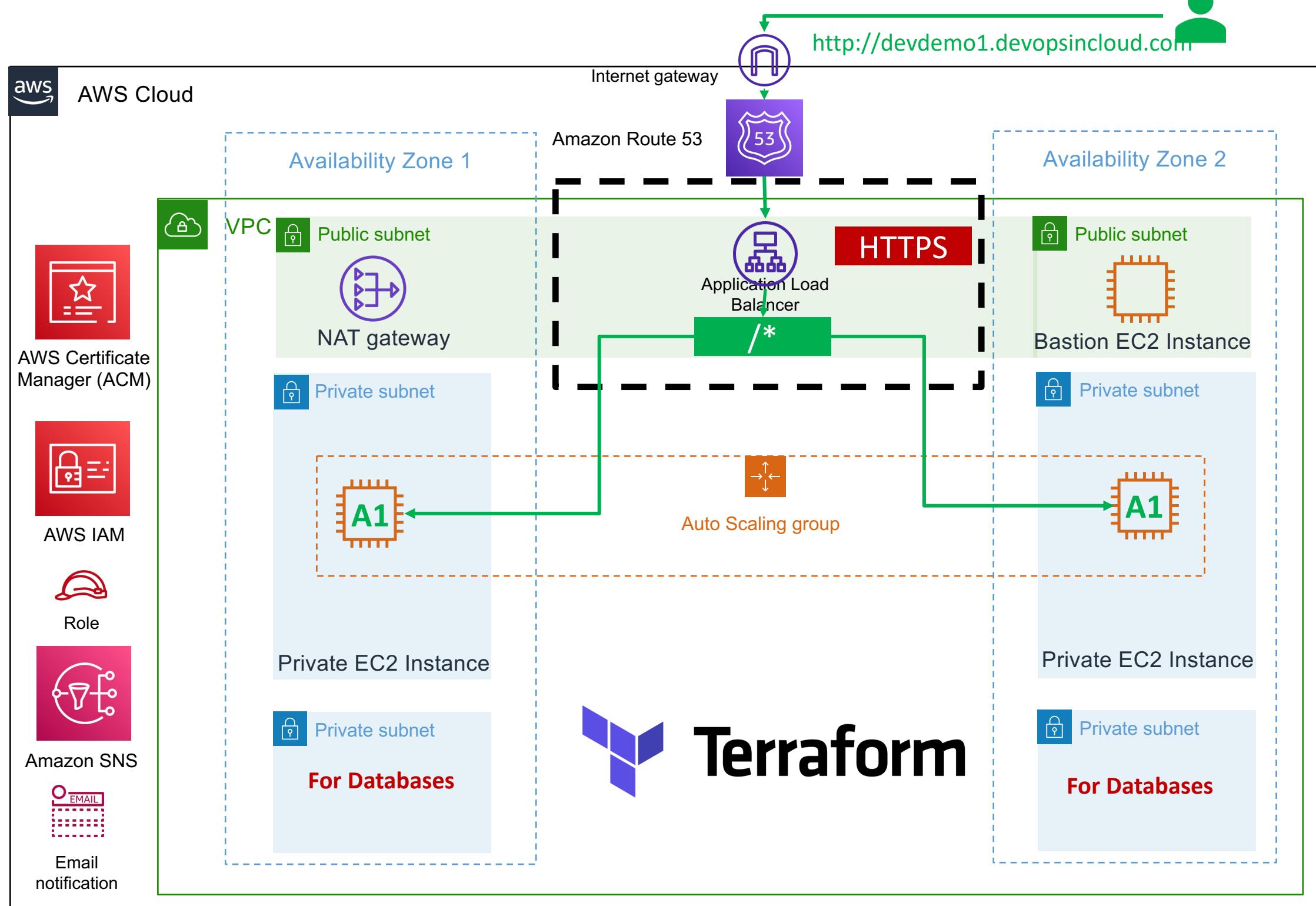
AWS IAM





Dev Environment

Staging Environment



Terraform

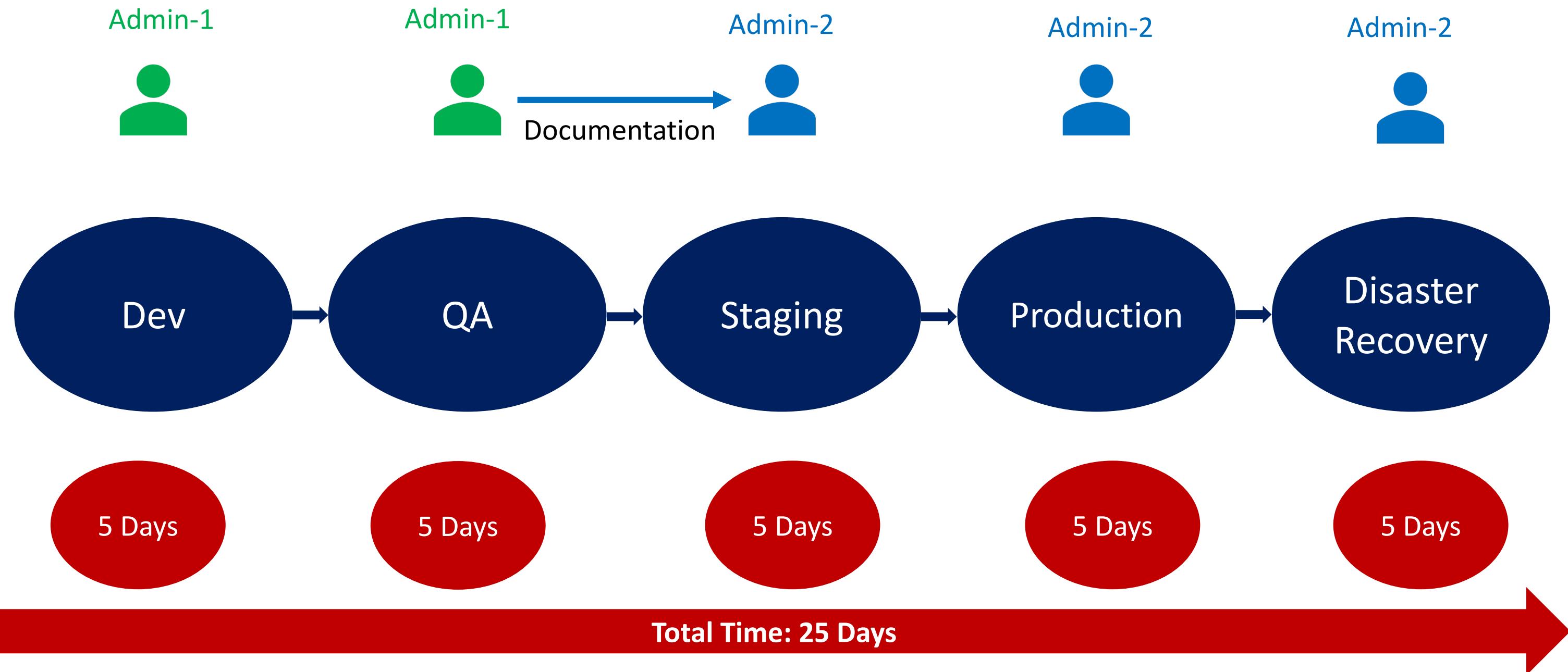
Infrastructure as Code

IaC

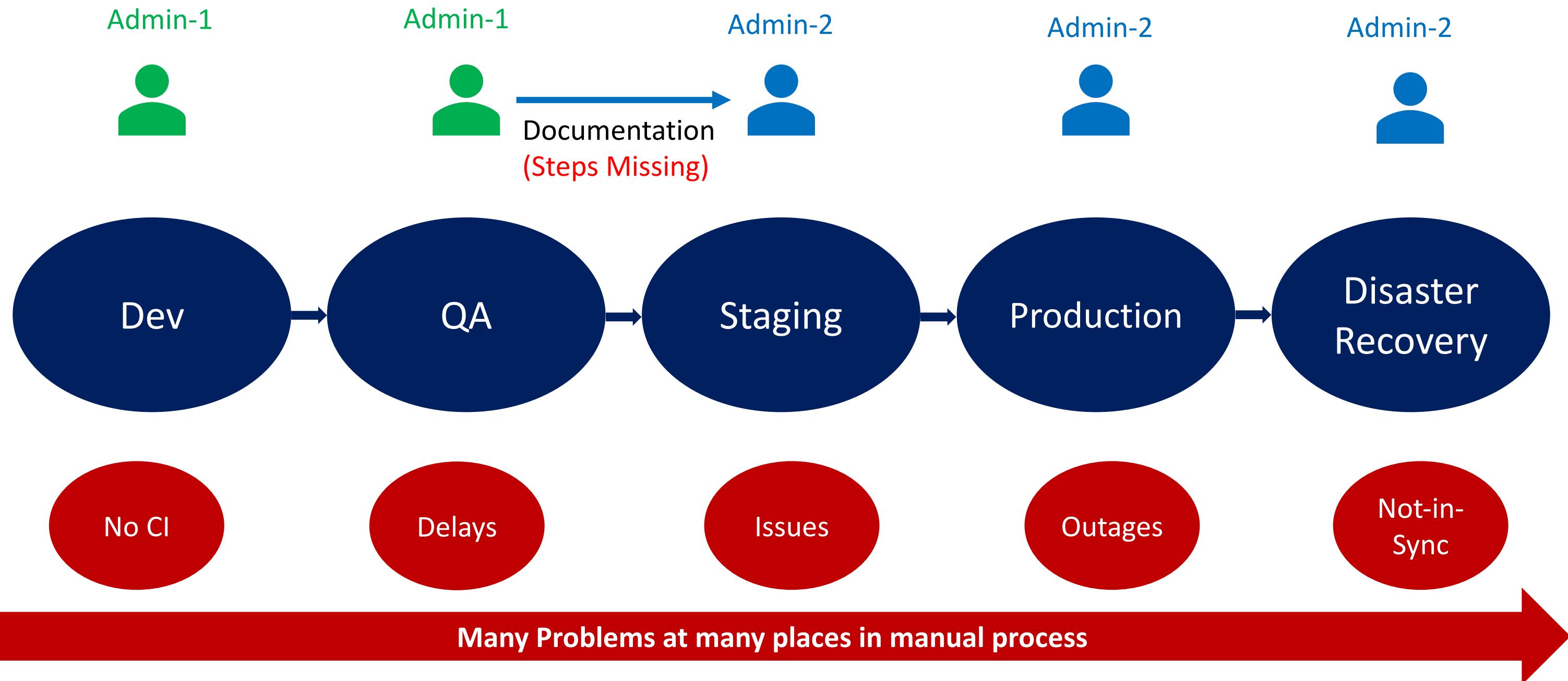


What is Infrastructure as Code ?

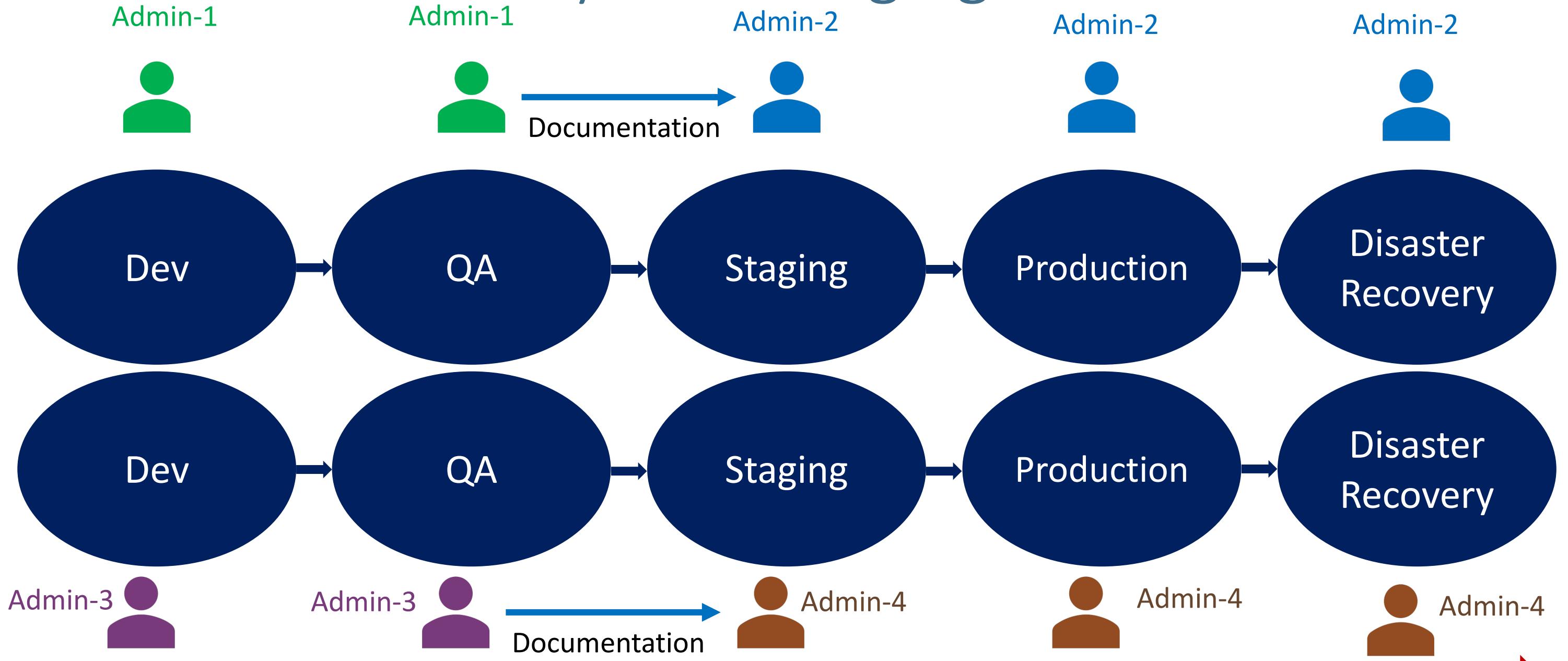
Traditional Way of Managing Infrastructure



Traditional Way of Managing Infrastructure



Traditional Way of Managing Infrastructure



Infrastructure scalability – Workforce need to be increased to meet the timelines

Traditional Way of Managing Infrastructure



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

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
Template
s

Quick &
Fast

Reliable

Tracked
for Audit

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

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.

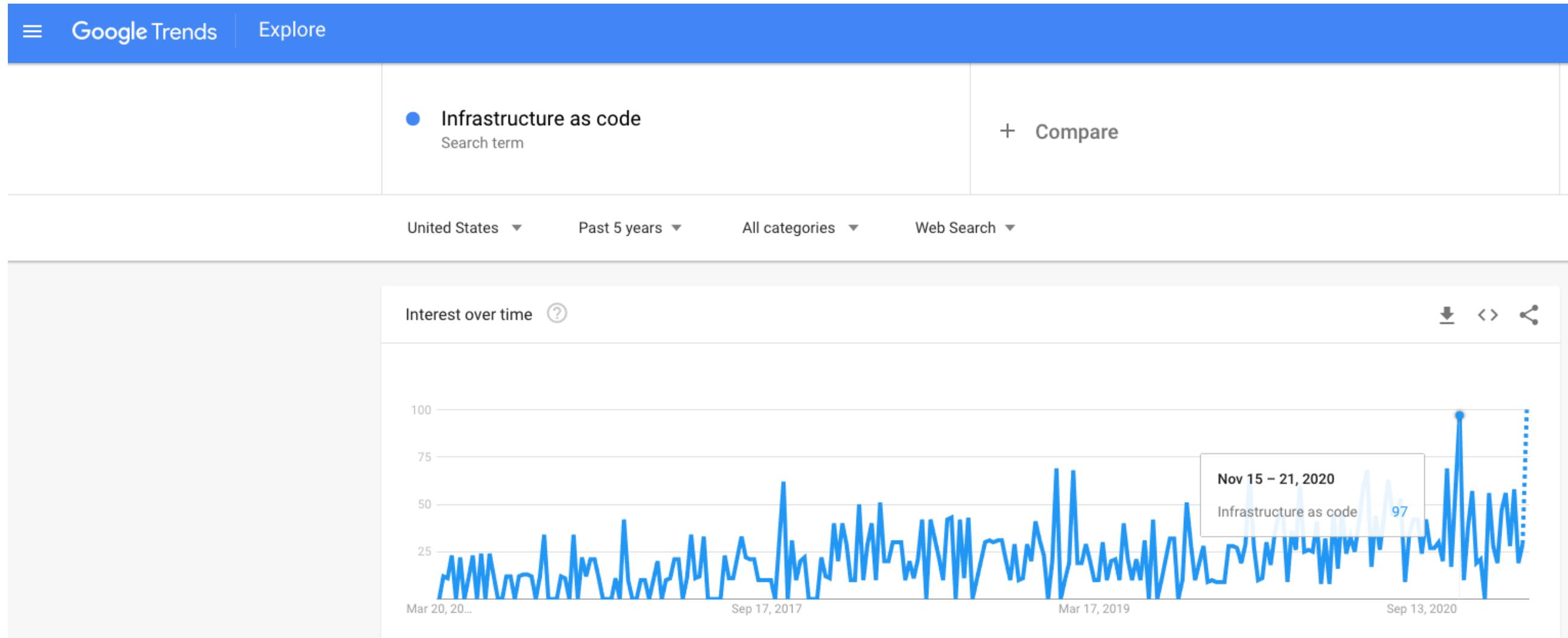
Security

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.

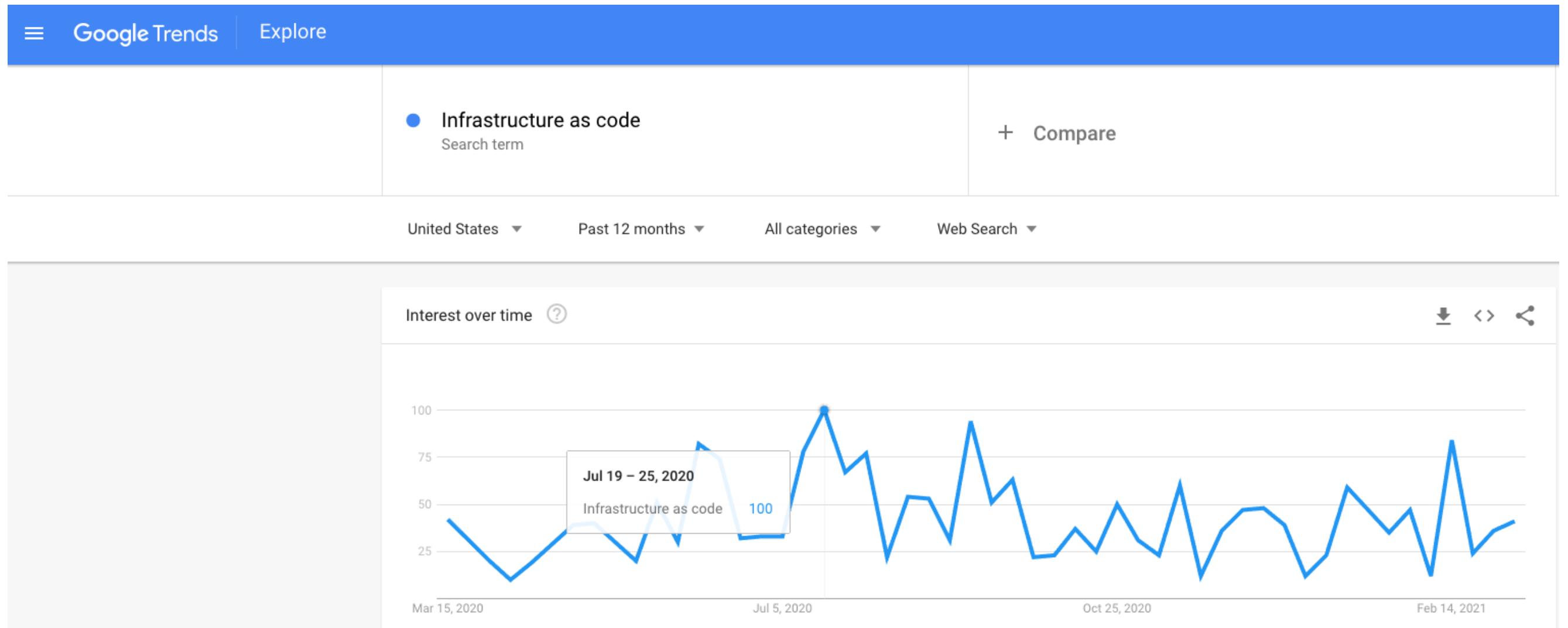
Audit

Terraform not only creates resources it also **maintains the record** of what is created in real world cloud environments using its State files.

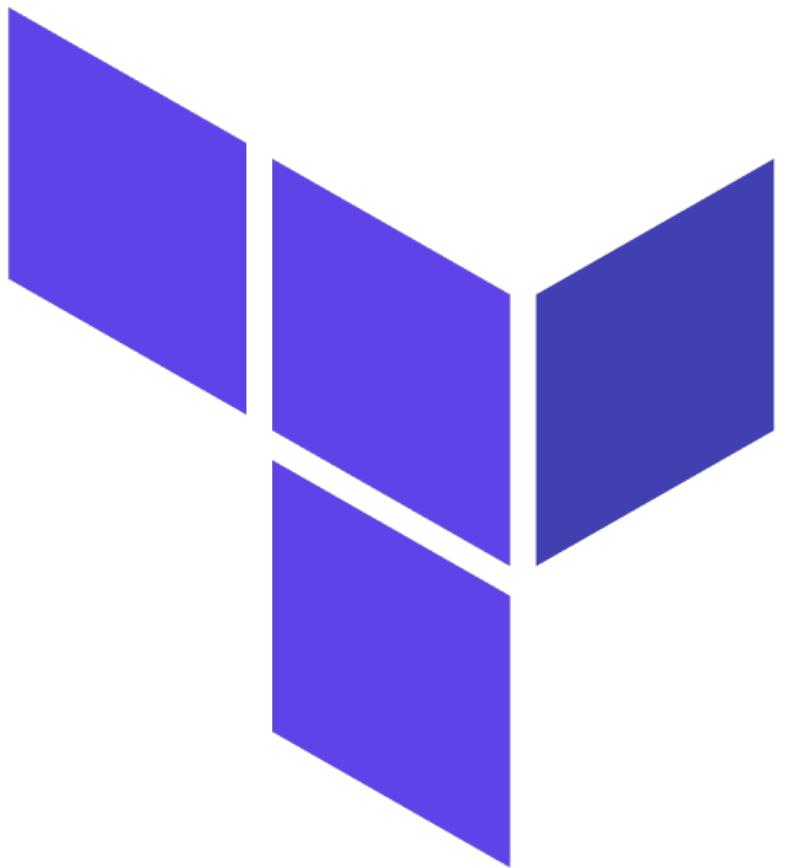
Google Trends – Past 5 Years



Google Trends – Past 1 Year



Terraform Installation



Terraform Installation

Terraform CLI

AWS CLI

VS Code Editor

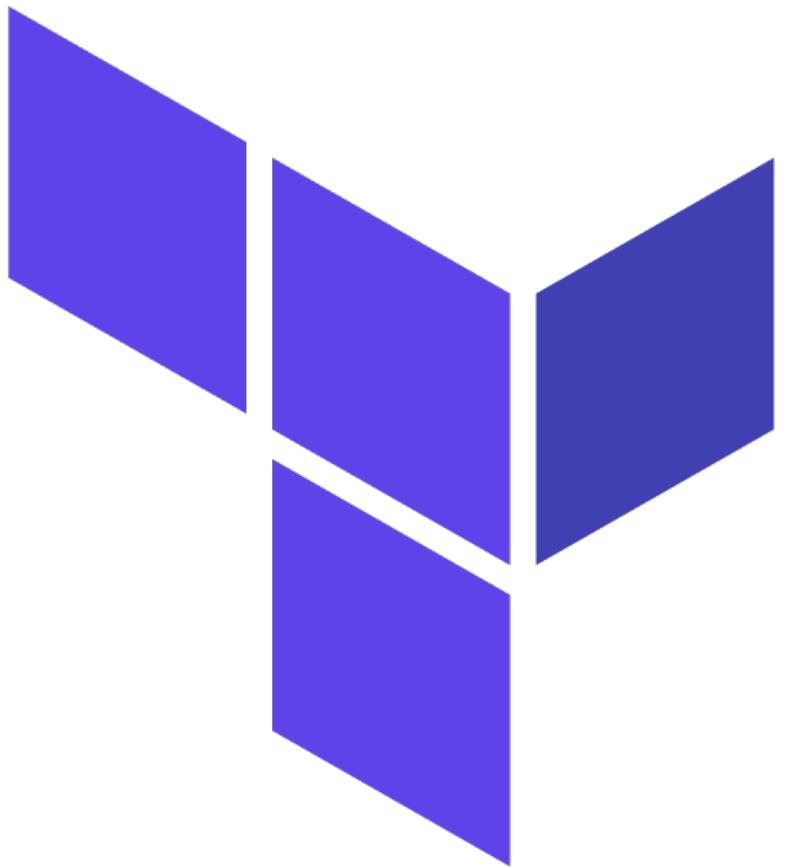
Terraform plugin
for VS Code

Mac OS

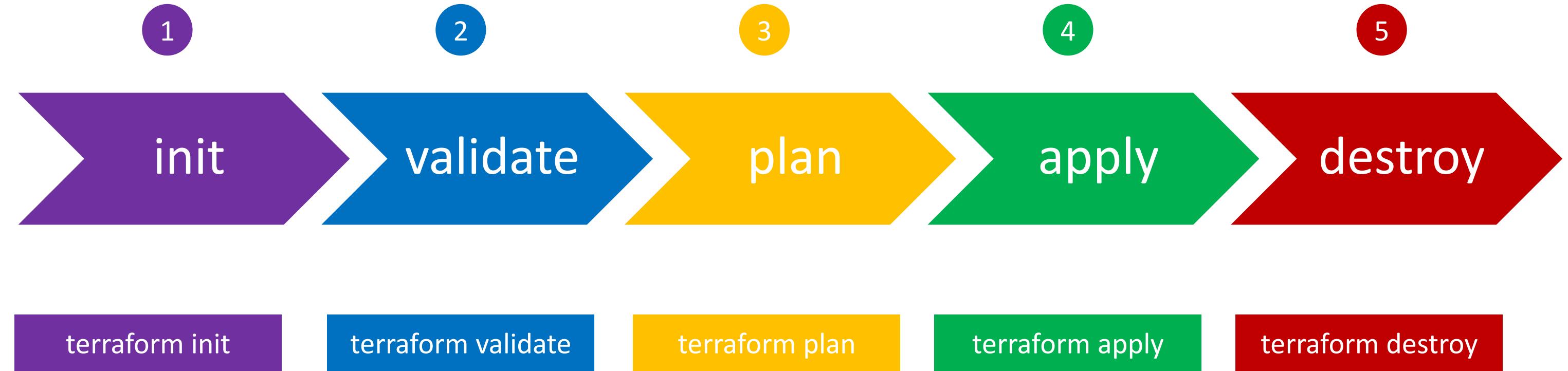
Windows OS

Linux OS

Terraform Command Basics



Terraform Workflow



Terraform Workflow

1

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

2

validate

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

3

plan

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

4

apply

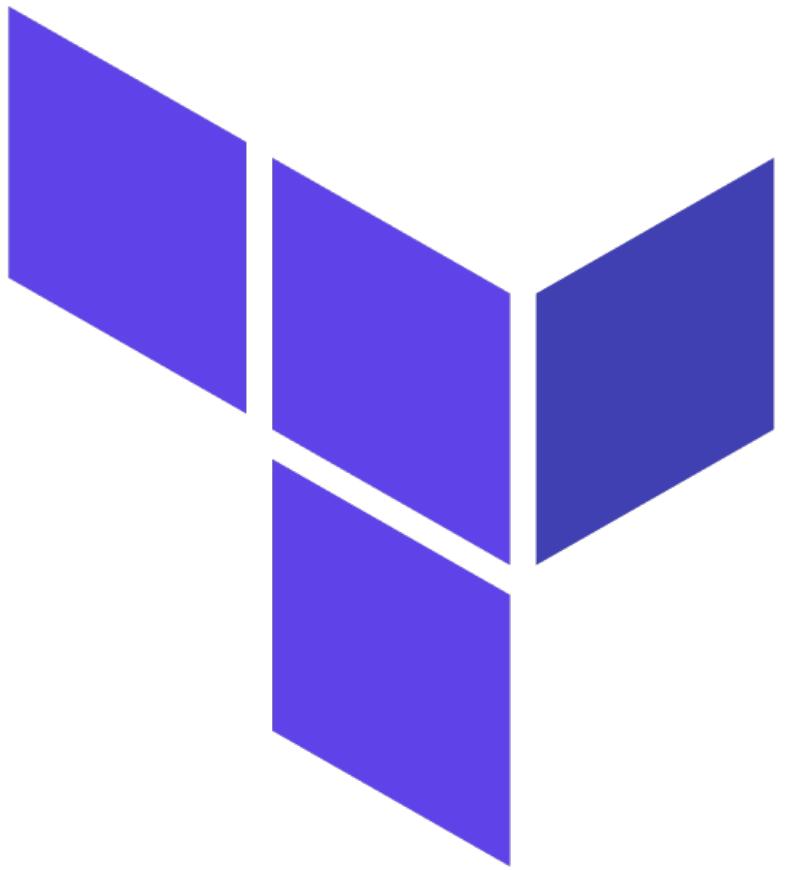
- Used to apply the changes required to reach the desired state of the configuration.
- By default, apply scans the current directory for the configuration and applies the changes appropriately.

5

destroy

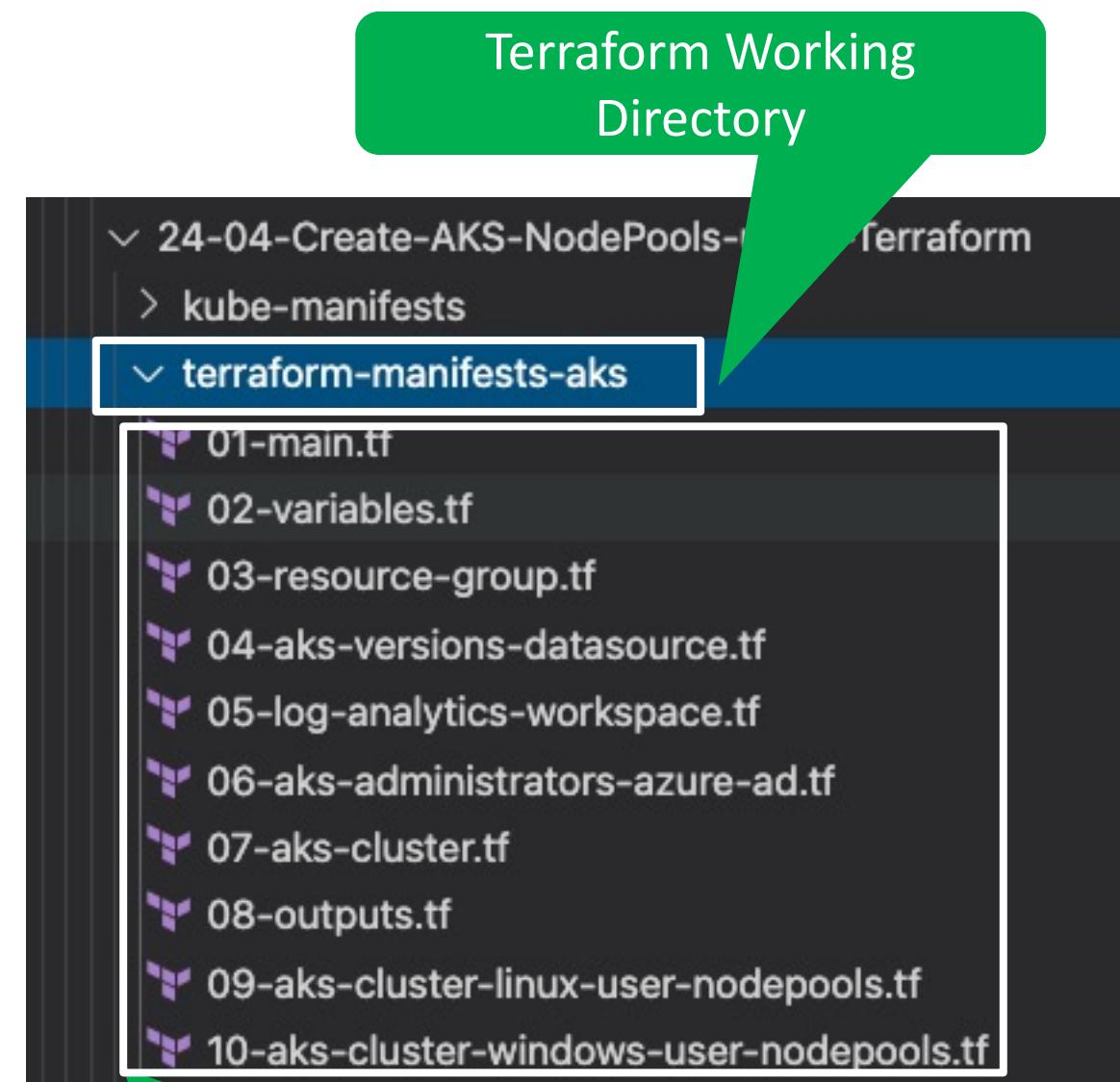
- Used to destroy the Terraform-managed infrastructure
- This will ask for confirmation before destroying.

Terraform Language Basics



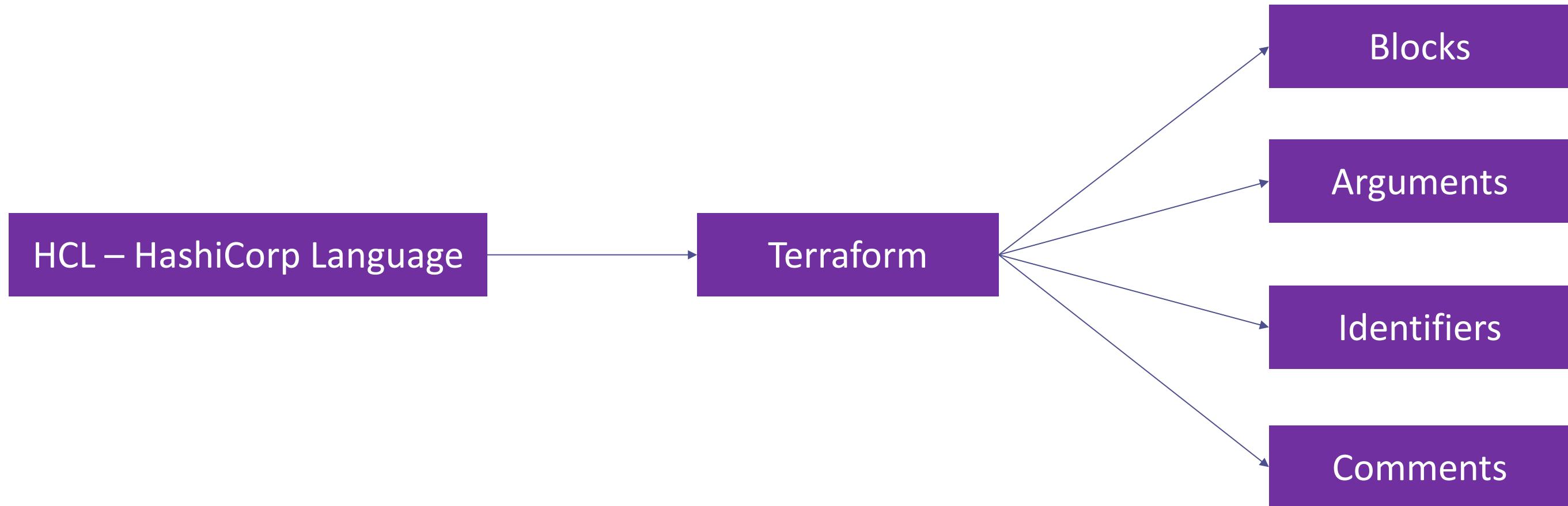
Terraform Language Basics – Files

- 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**



Terraform Configuration Files ending with **.tf** as extension

Terraform Language Basics – Configuration Syntax



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 Type

Top Level &
Block inside
Blocks

Top Level Blocks: resource, provider

Block Inside Block: provisioners,
resource specific blocks like tags

Arguments

Block Labels

Based on Block
Type block labels
will be 1 or 2

Example:
Resource – 2
labels

Variables – 1 label

Terraform Language Basics – Configuration Syntax

Argument
Name
[or]
Identifier

Argument
Value
[or]
Expression

```
# 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"
}
```

Terraform Language Basics – Configuration Syntax

Single Line Comments with # or //

```
# 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
    */
}
```

Multi-line comment



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.

Terraform Block

Providers Block

Resources Block

Fundamental Blocks

Input Variables Block

Output Values Block

Local Values Block

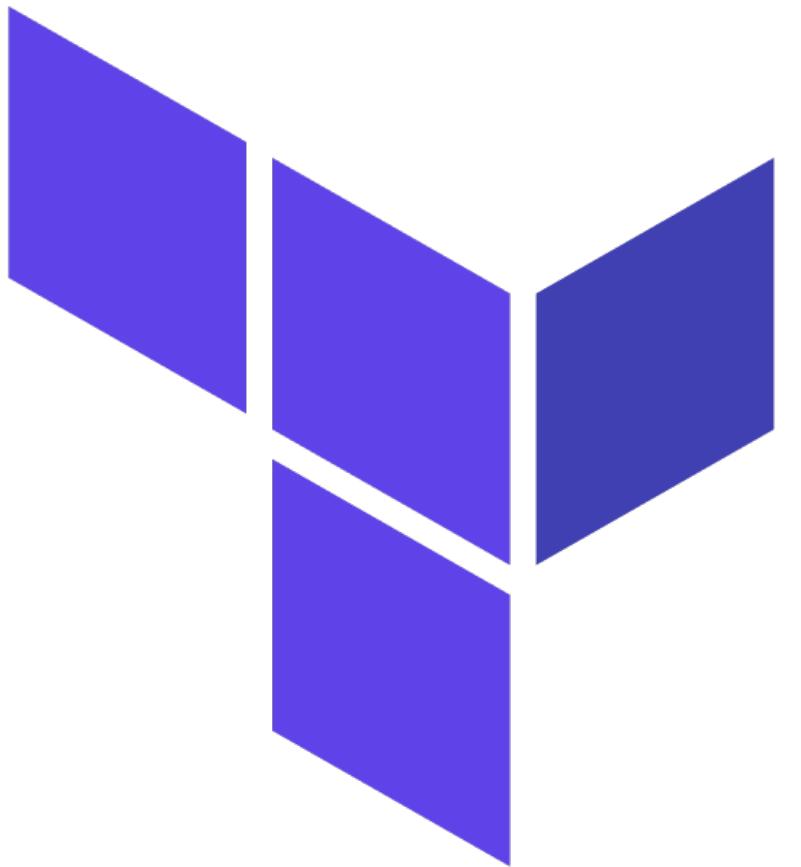
Variable Blocks

Data Sources Block

Modules Block

Calling / Referencing Blocks

Terraform Fundamental Blocks



Terraform Basic Blocks

Terraform Block

Special block used to configure some **behaviors**

Required Terraform Version

List Required Providers

Terraform Backend

c1-versions.tf

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**

c2-resource-name.tf

Resource Block

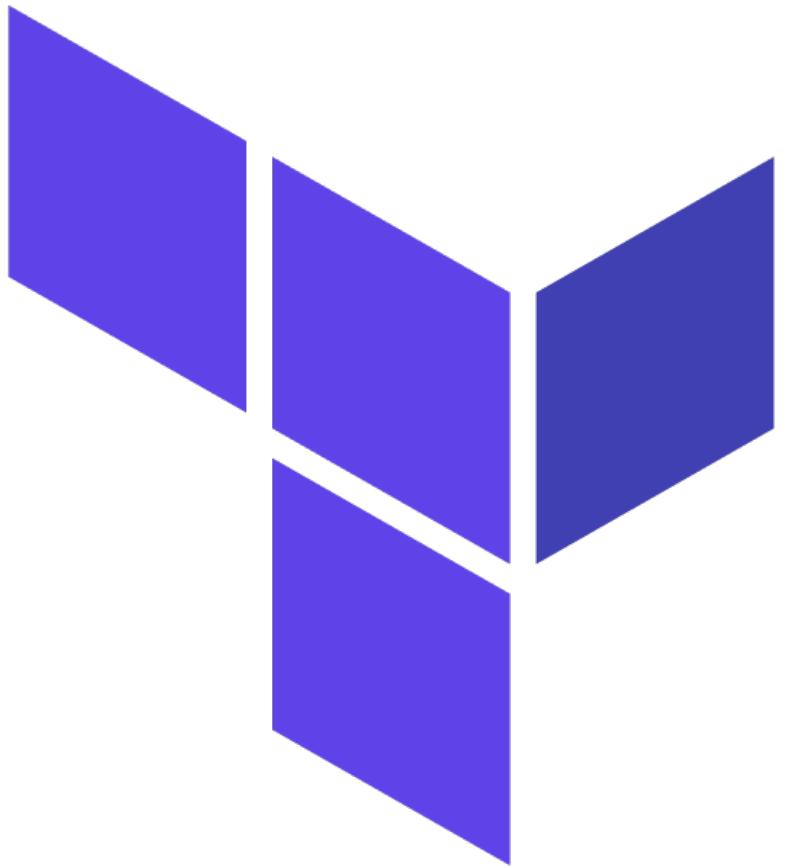
Each Resource Block describes one or more Infrastructure Objects

Resource Syntax: How to declare Resources?

Resource Behavior: How Terraform handles resource declarations?

Provisioners: We can configure Resource post-creation actions

Terraform Block



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.

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"
}
```

Terraform Block

Terraform Block

Required Terraform Version

Provider Requirements

Terraform Backend

Experimental Language Features

Passing Metadata to Providers

```
terraform {
  # Required Terraform Version
  required_version = "~> 0.14.3"

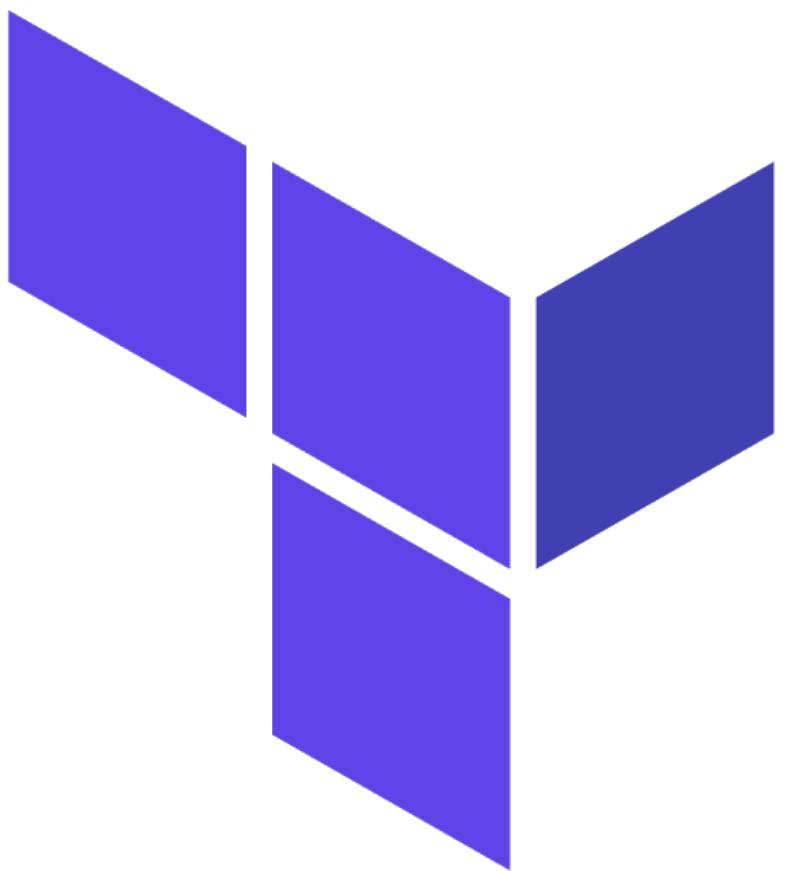
  # Required Providers and their Versions
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.21" # Optional but recommended
    }
  }

  # 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 ]

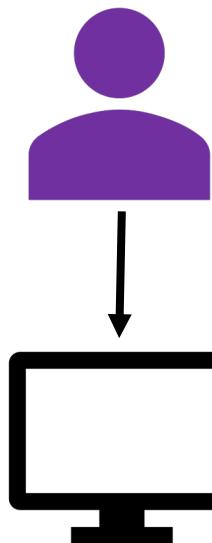
  # Passing Metadata to Providers (Super Advanced – Terraform)
  provider_meta "my-provider" {
    hello = "world"
  }
}
```

Terraform Providers



Terraform Providers

Terraform Admin



Local Desktop

Terraform CLI

Terraform AWS Provider

2 terraform validate

3 terraform plan

1 terraform init

Download Provider

4 terraform apply

5 terraform destroy

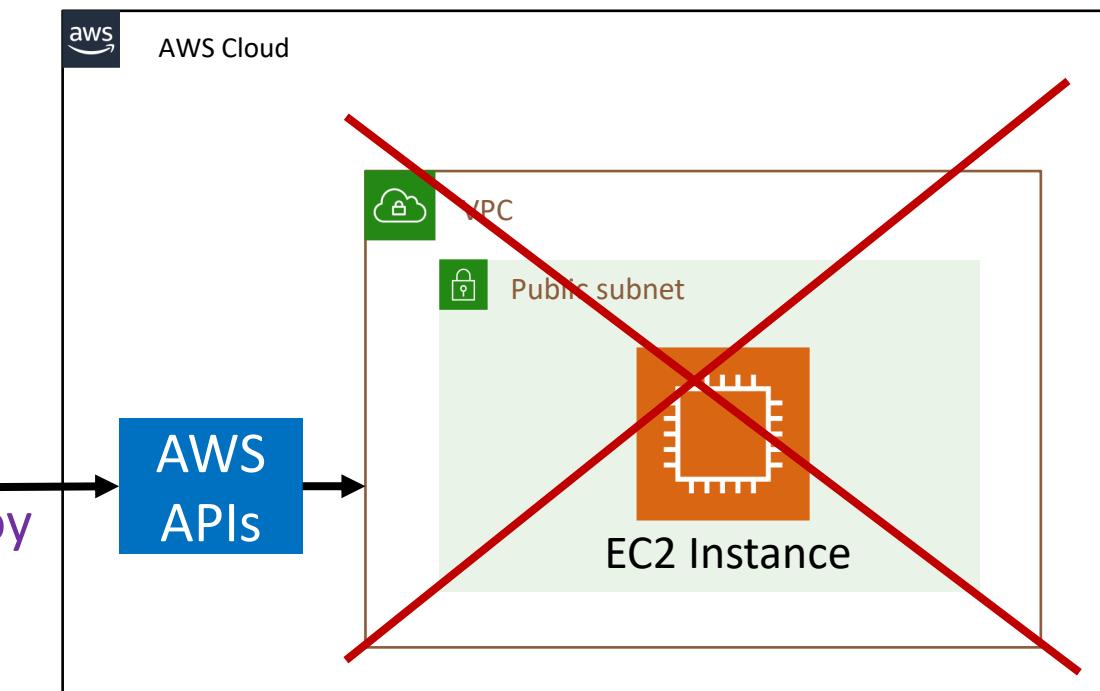
Providers are **HEART** of Terraform

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



Provider Requirements

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

Terraform Providers

Provider Configuration

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

Dependency Lock File

```
└── terraform-manifests
    ├── .terraform
    └── .terraform.lock.hcl
        ├── ec2-instance.tf
        └── terraform.tfstate
        └── terraform.tfstate.backup

# 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 = [
    "hh:f/Tz8zv1zb78ZaiyJk00MGIViZwbYrLuQk3kojPM91c=",
    "zh:4a9a66caf1964cd3b61fb3eb0da417195a5529ccb8e496f266b0778335d11c8",
    "zh:514f2f006ae68db715d86781673faf9483292deab235c7402ff306e0e92ea11a",
    "zh:5277b61109fdbb9011728f6650ef01a639a0590effe34ed7de7ba10d0c31803",
    "zh:67784dc8c8375ab37103eea1258c334ee92be6de033c2b37e3a2a65d0005142",
    "zh:76d4c8be2ca4a3294fb51fb58de1fe03361d3bc403820270cc8e71a04c5fa06",
    "zh:8f900b1cfdf6e8fb1a9d0382ecaa5056a3a84c94e313fb9e92c89de271cde",
    "zh:d0ac346519d0df124df89be2d803eb53f373434890f6ee3fb27112802f9eac59",
    "zh:d6256feedada82cbfb3b1d6dd9ad02048f23120ab50e6146a541cb11a108cc1",
    "zh:db2fe0d2e77c02e9a74e1ed694aa352295a50283f9a1cf896e5be252af14e9f4",
    "zh:eda61e889b579bd90046939a5b40cf5dc9031fb5a819fc3e4667a78bd432bdb2"
  ]
}
```

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:8f90b1cfdf6e8fb1a9d0382ecaa5056a3a84c94e313fbf9e92c89de271cdede",
    "zh:d0ac346519d0df124df89be2d803eb53f373434890f6ee3fb37112802f9eac59",
    "zh:d6256feedada82cbfb3b1dd6dd9ad02048f23120ab50e6146a541cb11a108cc1",
    "zh:db2fe0d2e77c02e9a74e1ed694aa352295a50283f9a1cf896e5be252af14e9f4",
    "zh:eda61e889b579bd90046939a5b40cf5dc9031fb5a819fc3e4667a78bd432bdb2",
  ]
}
```

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

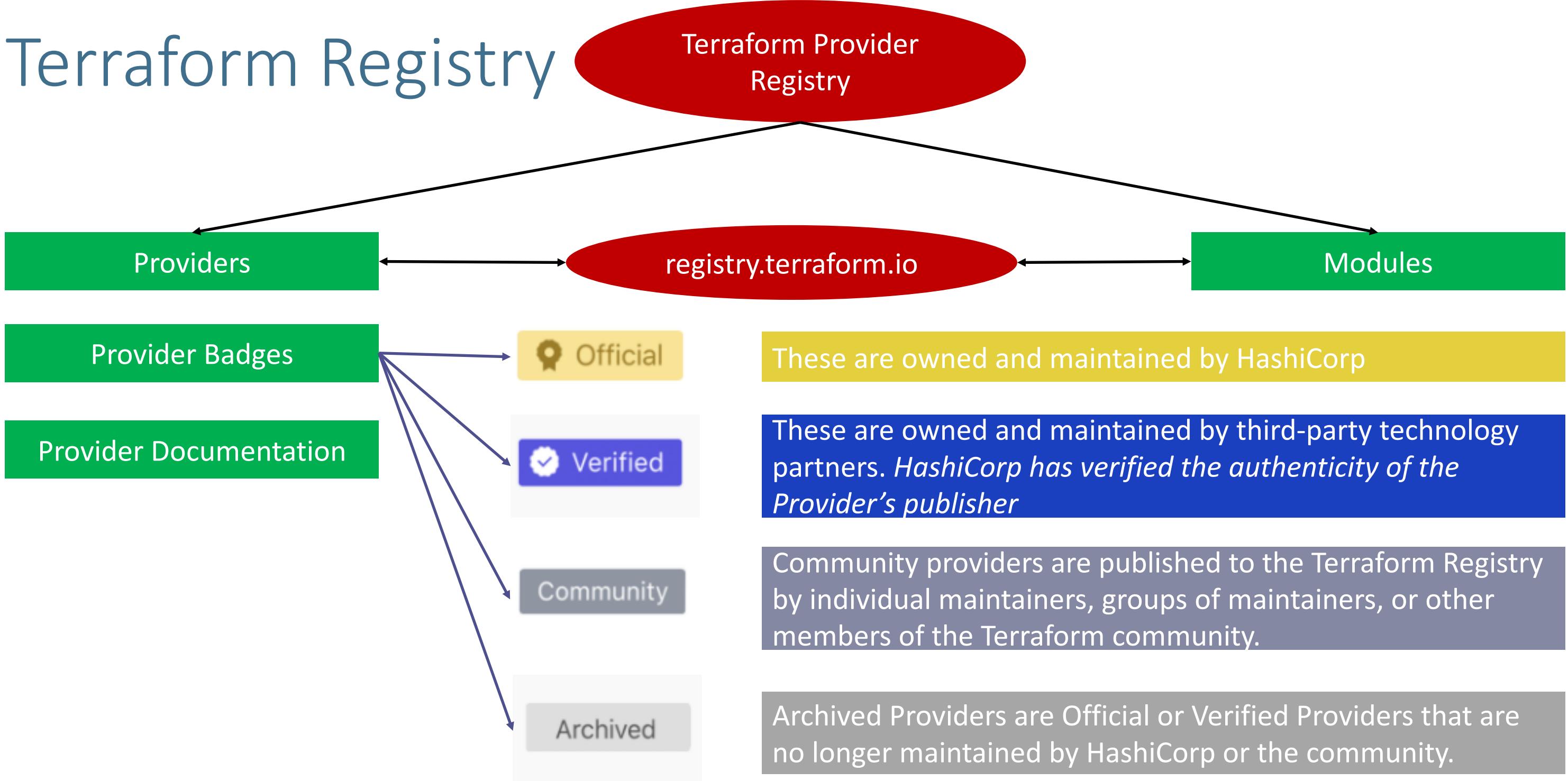
Source addresses consist of **three parts** delimited by **slashes (/)**

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

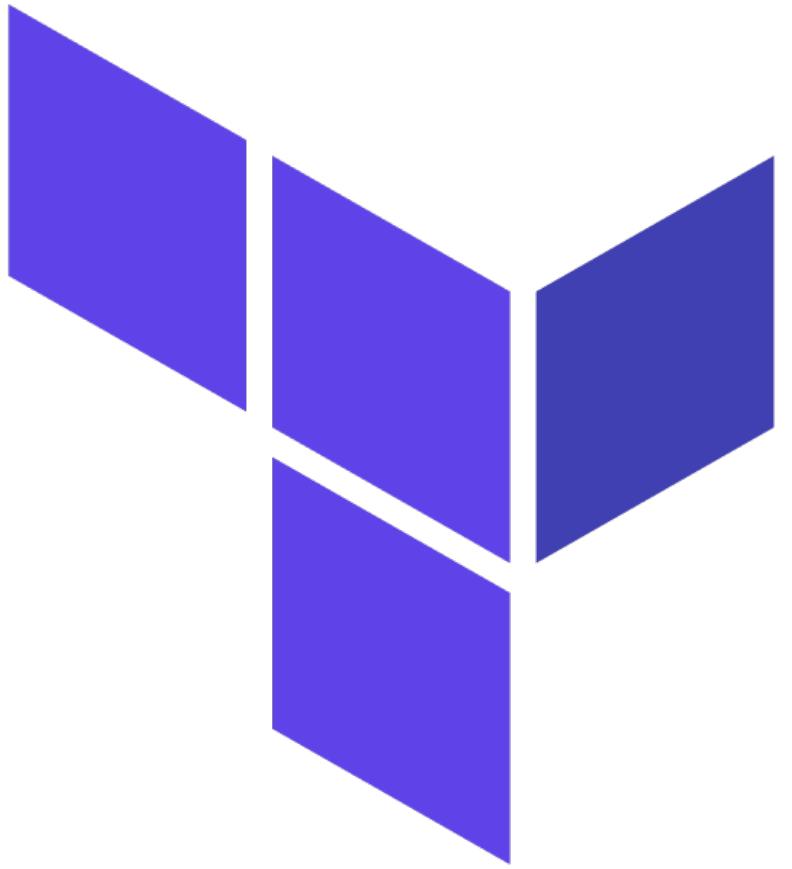
registry.terraform.io/hashicorp/aws

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

Terraform Registry



Terraform Resources Introduction



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 Type

Top Level &
Block inside
Blocks

Top Level Blocks: resource, provider

Block Inside Block: provisioners,
resource specific blocks like tags

Arguments

Block Labels

Based on Block
Type block labels
will be 1 or 2

Example:
Resource – 2
labels

Variables – 1 label

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

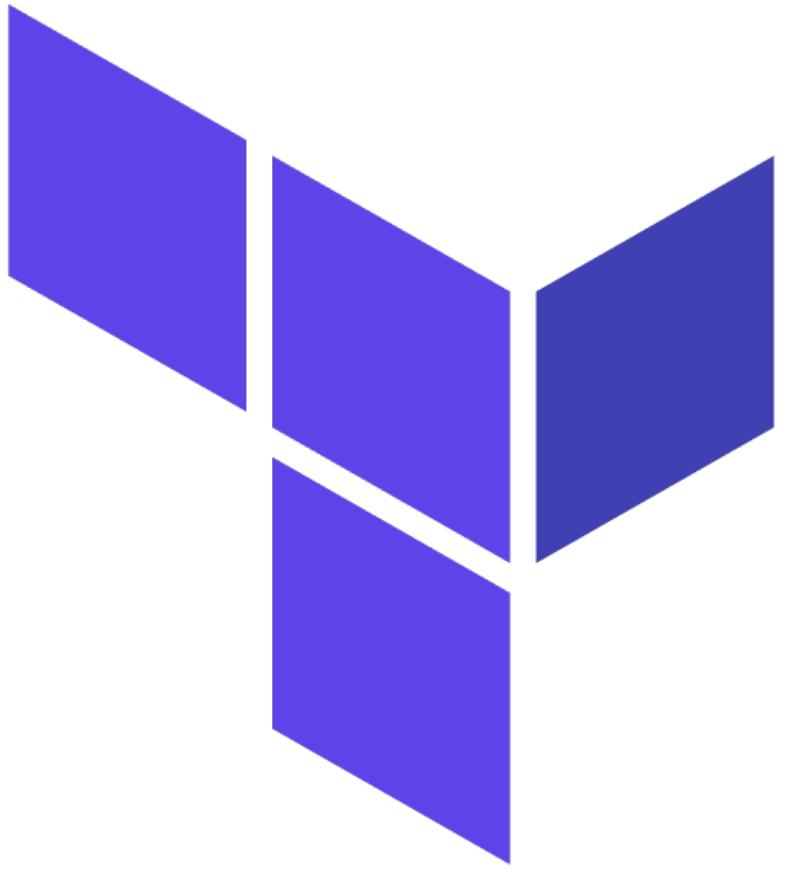
Meta-Arguments: Can be used with any resource to change the behavior of resources

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"
  }
}
```

Terraform State



Resource Behavior



Terraform State

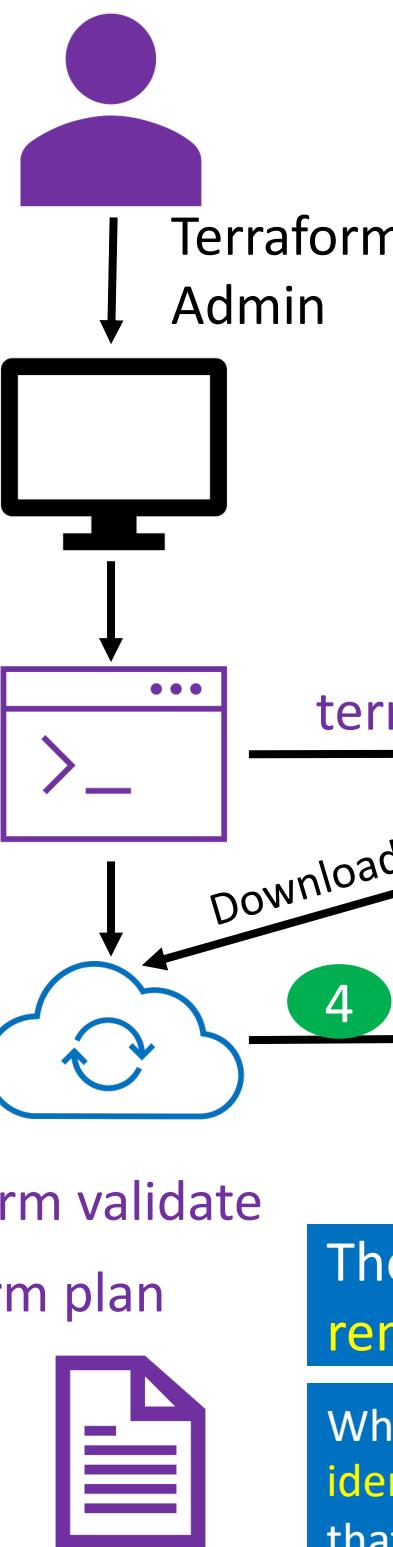
Terraform State

Local Desktop

Terraform CLI

Terraform AWS Provider

Terraform State File
terraform.tfstate



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.

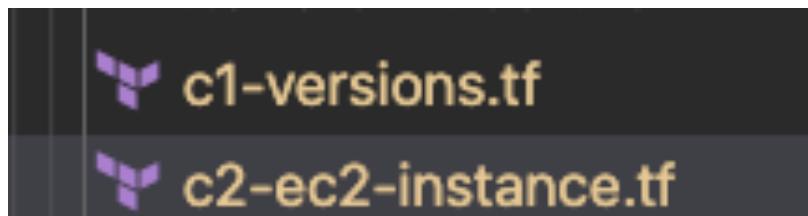
This state is stored by default in a local file named "**terraform.tfstate**", but it can also be stored **remotely**, which works better in a **team** environment.

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.

Desired & Current Terraform States

Terraform Configuration Files



Real World Resource – EC2 Instance

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
web	i-0663449fef49e9cf5	Running	t2.micro	2/2 checks ...	No alarms +	us-east-1b
Instance: i-0663449fef49e9cf5 (web)						
Details	Security	Networking	Storage	Status checks	Monitoring	Tags
Instance ID i-0663449fef49e9cf5 (web)	Public IPv4 address 54.144.73.100 open address	Private IPv4 addresses 172.31.94.137	Public IPv4 DNS ec2-54-144-73-100.compute-1.amazonaws.com open address	Private IPv4 DNS ip-172-31-94-137.ec2.internal	VPC ID vpc-54972d2e (default-vpc)	Subnet ID subnet-d2e590fc
Instance state Running	Elastic IP addresses -	AWS Compute Optimizer finding Opt-in to AWS Compute Optimizer for recommendations.	IAM Role -			

Desired State



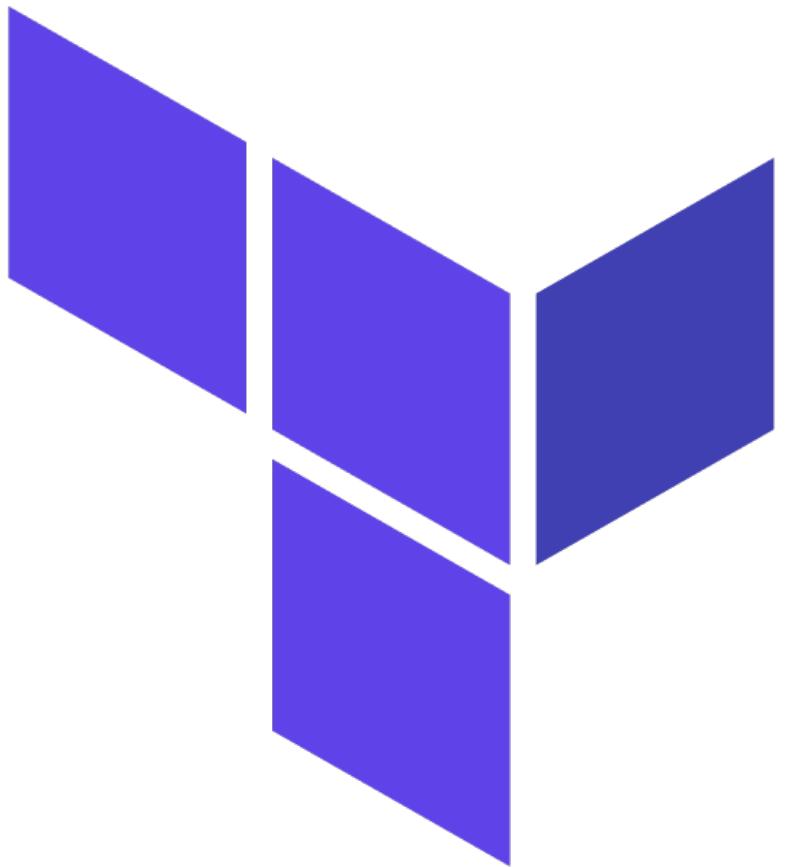
Current State

Terraform

Input Variables

Datasources

Outputs



What are we going to learn ?

Terraform
Concepts

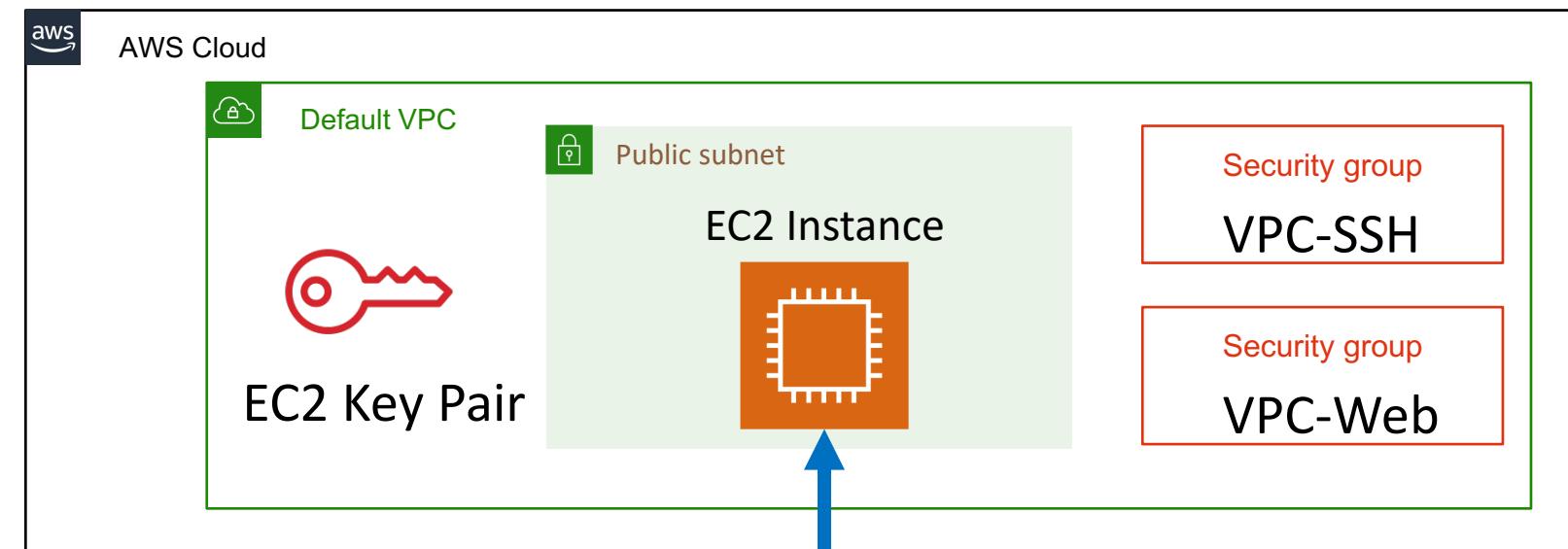
AWS Services

Terraform Input Variables

Terraform Output Values

Terraform Datasources

Dynamically get latest AMI ID



Terraform
Input
Variables

Terraform
Variables

Terraform
Output
Values

Terraform
Local
Values

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.

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

Terraform
Input
Variables

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"]
    }
}
```

Terraform Datasources

We can refer the data resource in a resource as depicted

Meta-Arguments for Datasources

```
# 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"
    }
}
```

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**.

Terraform Variables – Output Values

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

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**.

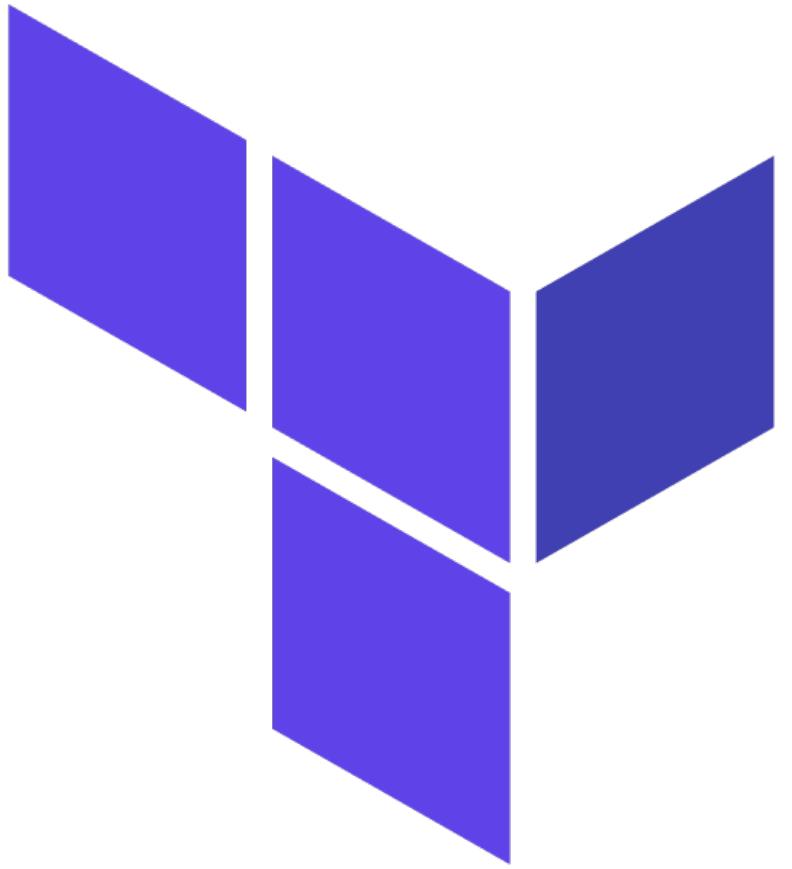
Terraform
Variables
Outputs

3

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

Advanced

Terraform
For Loops
Lists & Maps
Meta-Argument
Count & for_each



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

Utility Project
with
Datasources



Amazon Virtual Private Cloud
(Amazon VPC)



Internet gateway



NAT gateway



Elastic IP
address



Route table



Public Subnet

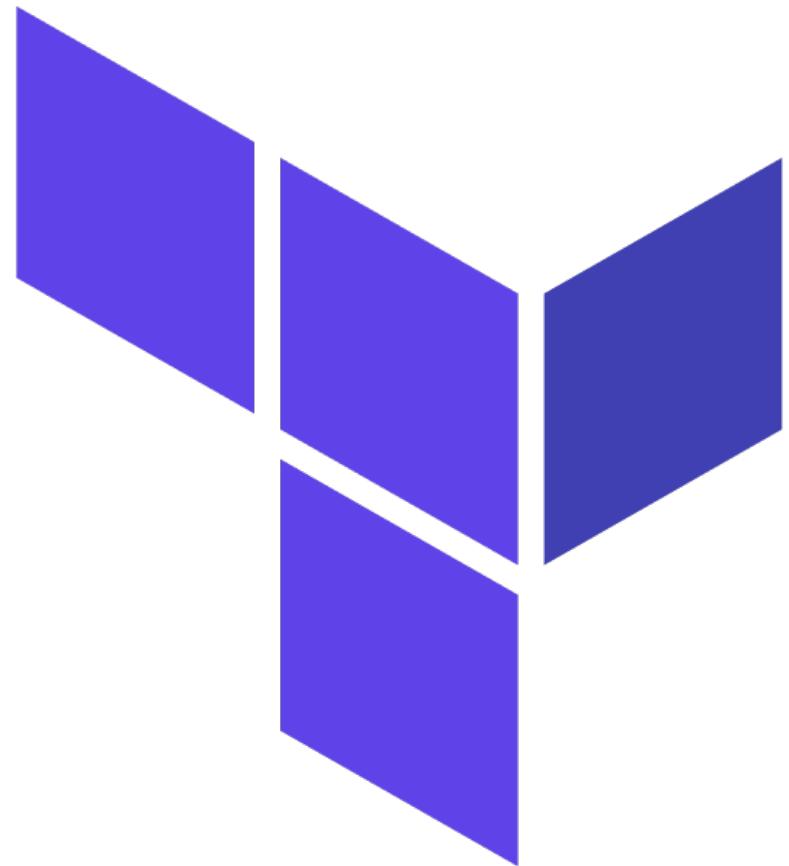


Private Subnet

AWS VPC

3-Tier

Web, App and DB



AWS VPC 3-Tier Architecture

Build **manually** using AWS Mgmt Console

Build same using **Terraform**

Terraform Modules

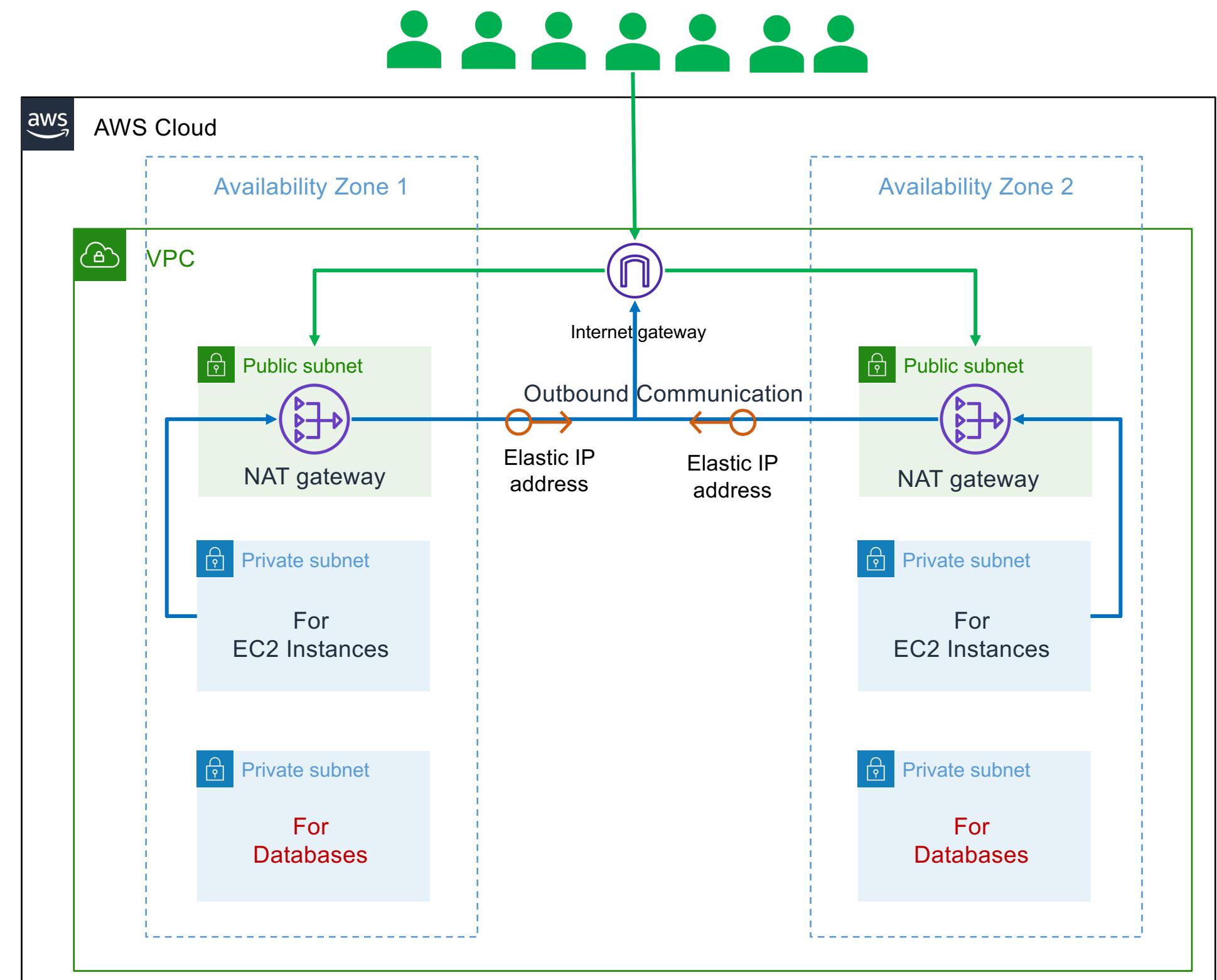
Terraform Local Values

Terraform Variables – `terraform.tfvars`

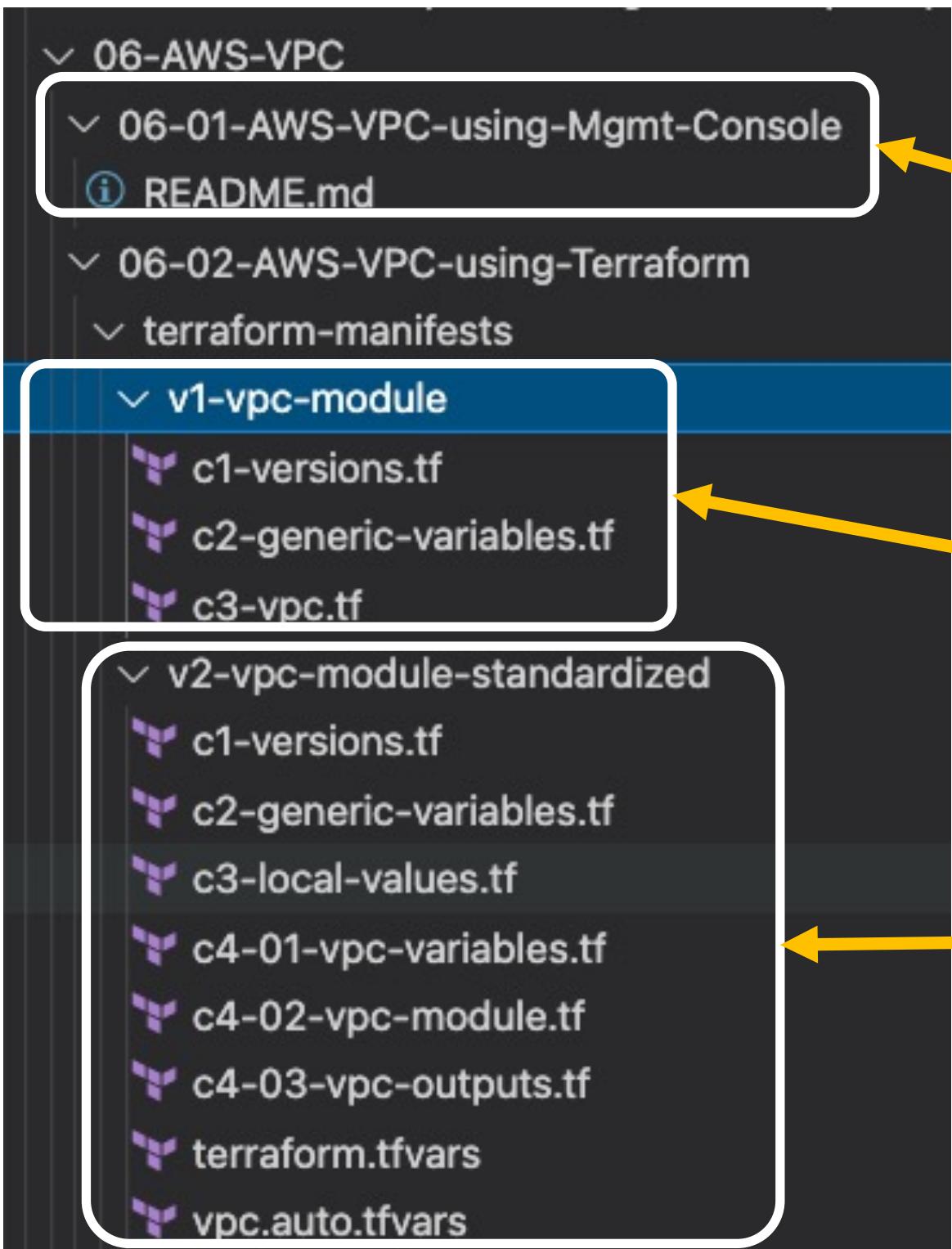
Terraform Variables – `vpc.auto.tfvars`

Terraform Version Constraints

Terraform Code Organizing – **Production Grade style**



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

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.

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**.

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

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

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

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

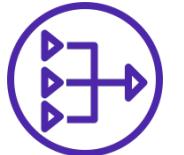
This makes it possible to **publish modules for others to use**, and to use modules that others have published.



Amazon Virtual Private Cloud
(Amazon VPC)



Internet gateway



NAT gateway



Elastic IP address



Route table



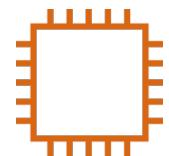
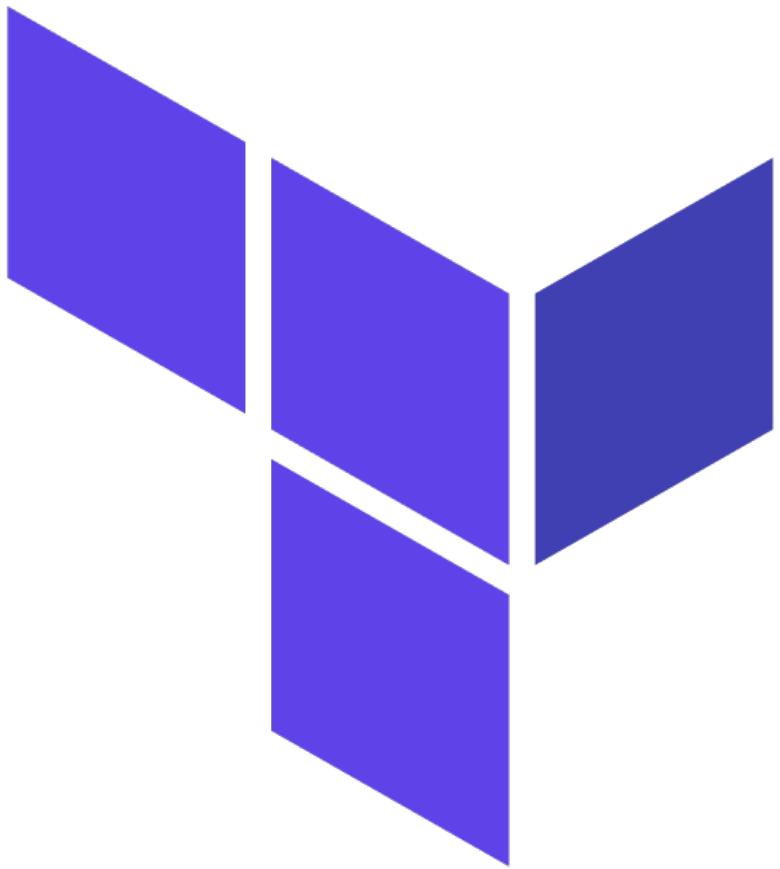
Public Subnet



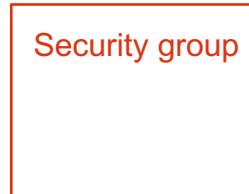
Private Subnet

AWS

VPC with Web, App and DB Tiers EC2 Instances, Bastion Host and Security Groups



EC2 Instance



Security group

AWS VPC + EC2 Instance + Security Groups



Terraform & AWS Concepts

Terraform Module: VPC

Terraform Module: Security Group

Terraform Module: AWS EC2 Instance

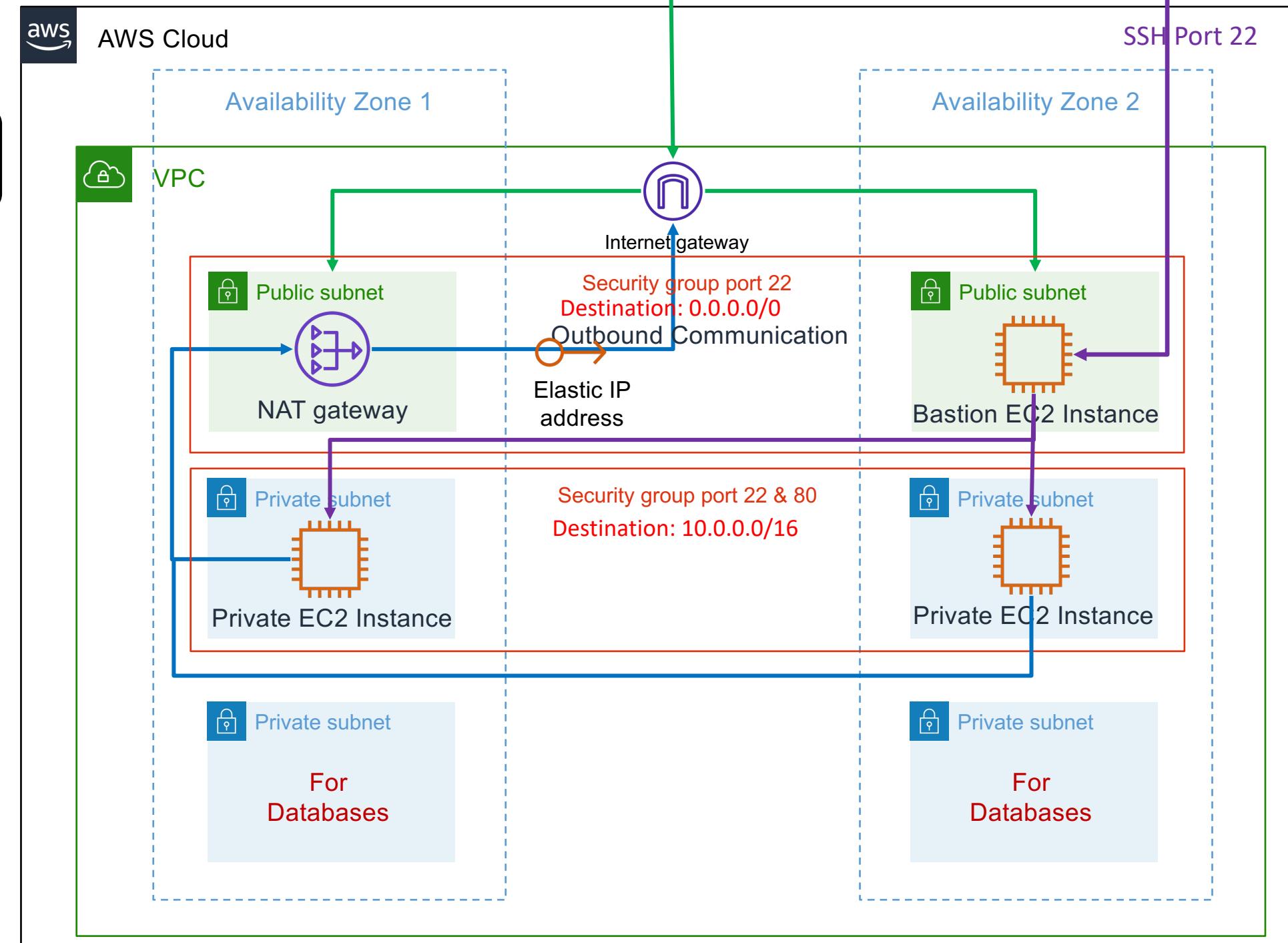
Meta-Argument: `depends_on`

Terraform `null_resource`

Terraform `File` Provisioner

Terraform `Remote-exec` Provisioner

Terraform `Local-exec` Provisioner



What are we going to learn ?





Amazon Virtual Private Cloud
(Amazon VPC)



Internet gateway



NAT gateway



Elastic IP address



Public Subnet

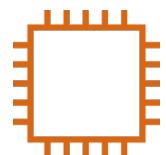
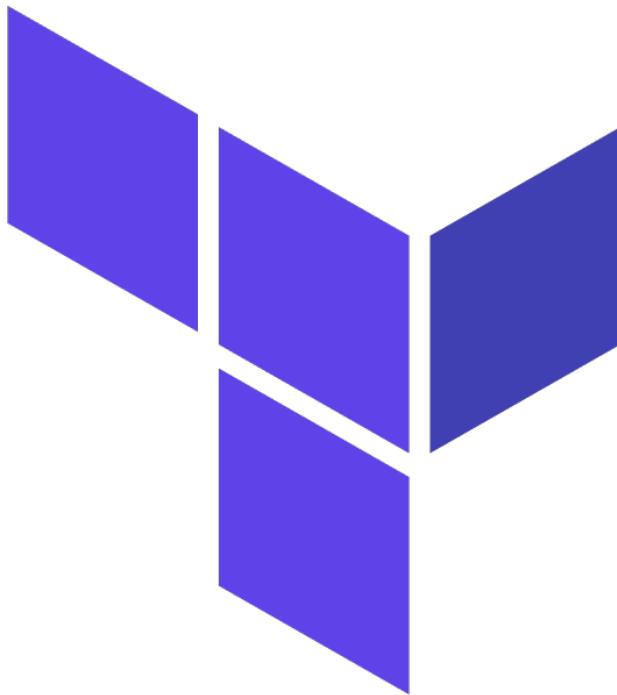


Private Subnet

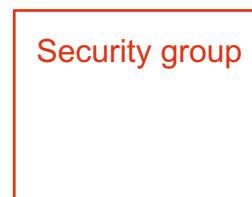
AWS

VPC with Web, App and DB Tiers
EC2 Instances, Bastion Host and Security Groups

Classic Load Balancer



EC2 Instance



Security Groups



Classic Load
Balancer

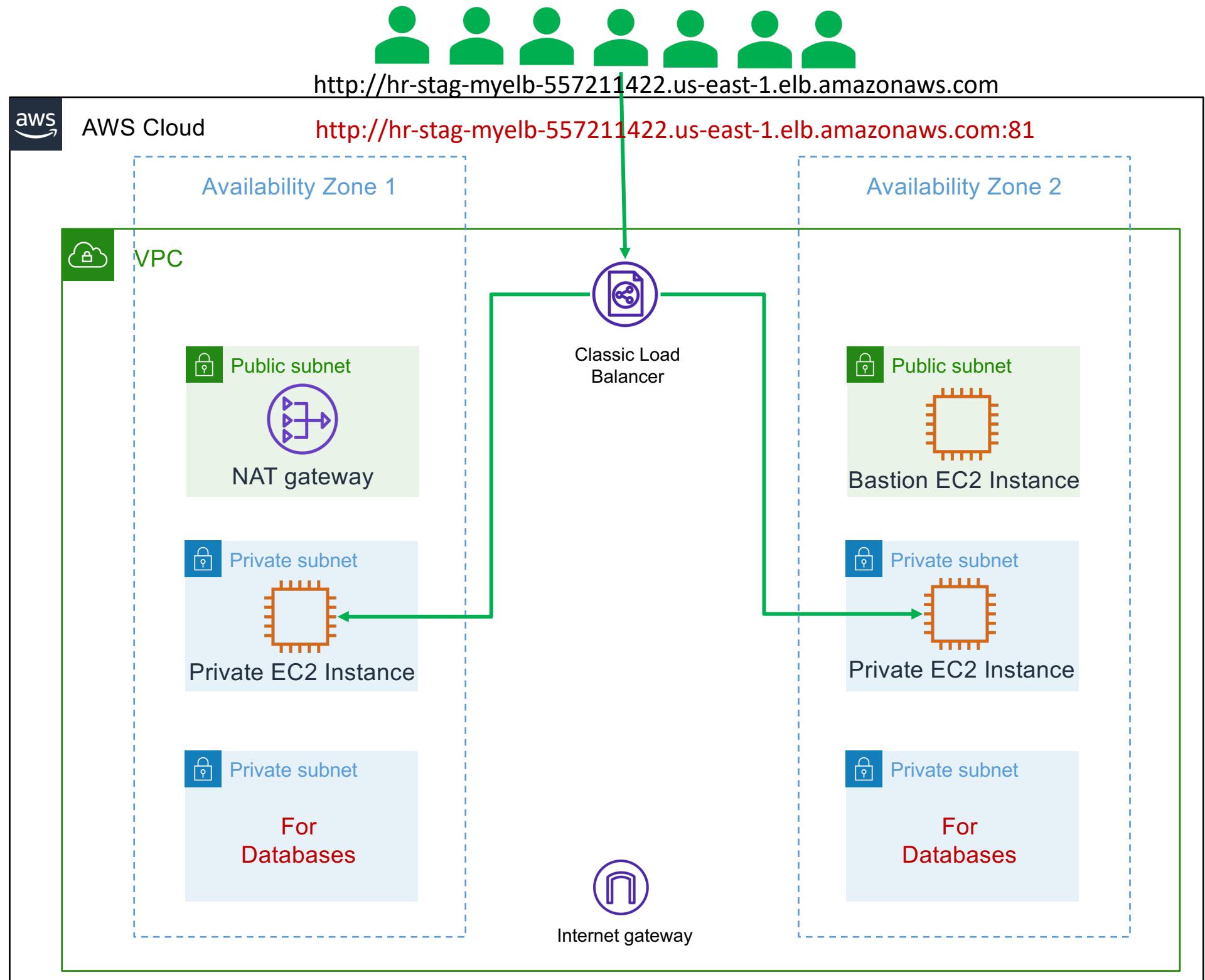
AWS VPC + EC2 Instance + Security Groups + AWS ELB Classic Load Balancer

Terraform & AWS Concepts

Terraform Module: ELB Classic LB

Terraform Module: Security Group

Add Custom SG Rule for Inbound Port 81



What are we going to learn ?

- └ c1-versions.tf
- └ c2-generic-variables.tf
- └ c3-local-values.tf
- └ c4-01-vpc-variables.tf
- └ c4-02-vpc-module.tf
- └ c4-03-vpc-outputs.tf
- └ c5-01-securitygroup-variables.tf
- └ c5-02-securitygroup-outputs.tf
- └ c5-03-securitygroup-bastionsg.tf
- └ c5-04-securitygroup-privatesg.tf
- └ **c5-05-securitygroup-loadbalancersg.tf**
- └ c6-01-datasource-ami.tf
- └ c7-01-ec2instance-variables.tf
- └ c7-02-ec2instance-outputs.tf
- └ c7-03-ec2instance-bastion.tf
- └ c7-04-ec2instance-private.tf
- └ c8-elasticip.tf
- └ c9-nullresource-provisioners.tf
- └ **c10-01-ELB-classic-loadbalancer-variables.tf**
- └ **c10-02-ELB-classic-loadbalancer.tf**
- └ **c10-03-ELB-classic-loadbalancer-outputs.tf**
- └ ec2instance.auto.tfvars
- └ terraform.tfvars
- └ vpc.auto.tfvars

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



Amazon Virtual Private Cloud
(Amazon VPC)



Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



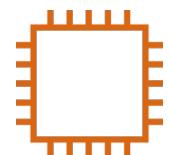
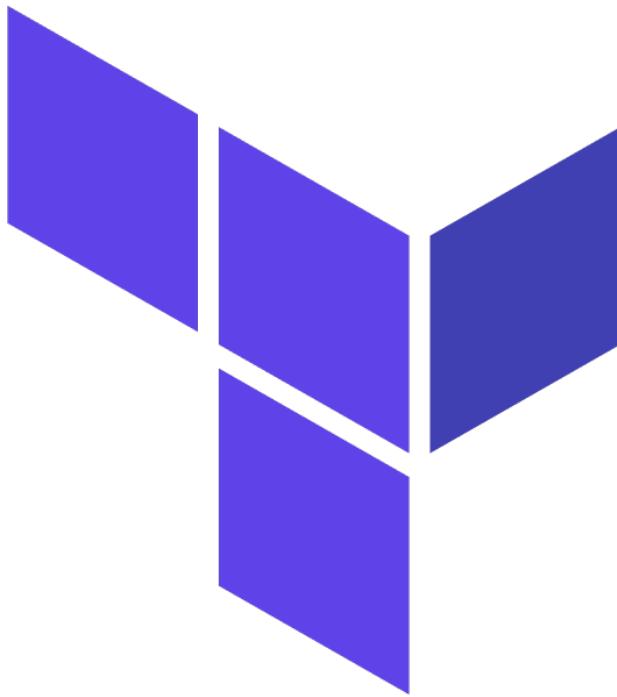
Private Subnet

AWS

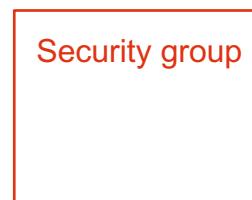
VPC with Web, App and DB Tiers

EC2 Instances, Bastion Host and Security Groups

Application Load Balancer - Basic



EC2 Instance



Security group



Classic Load
Balancer



Application Load
Balancer

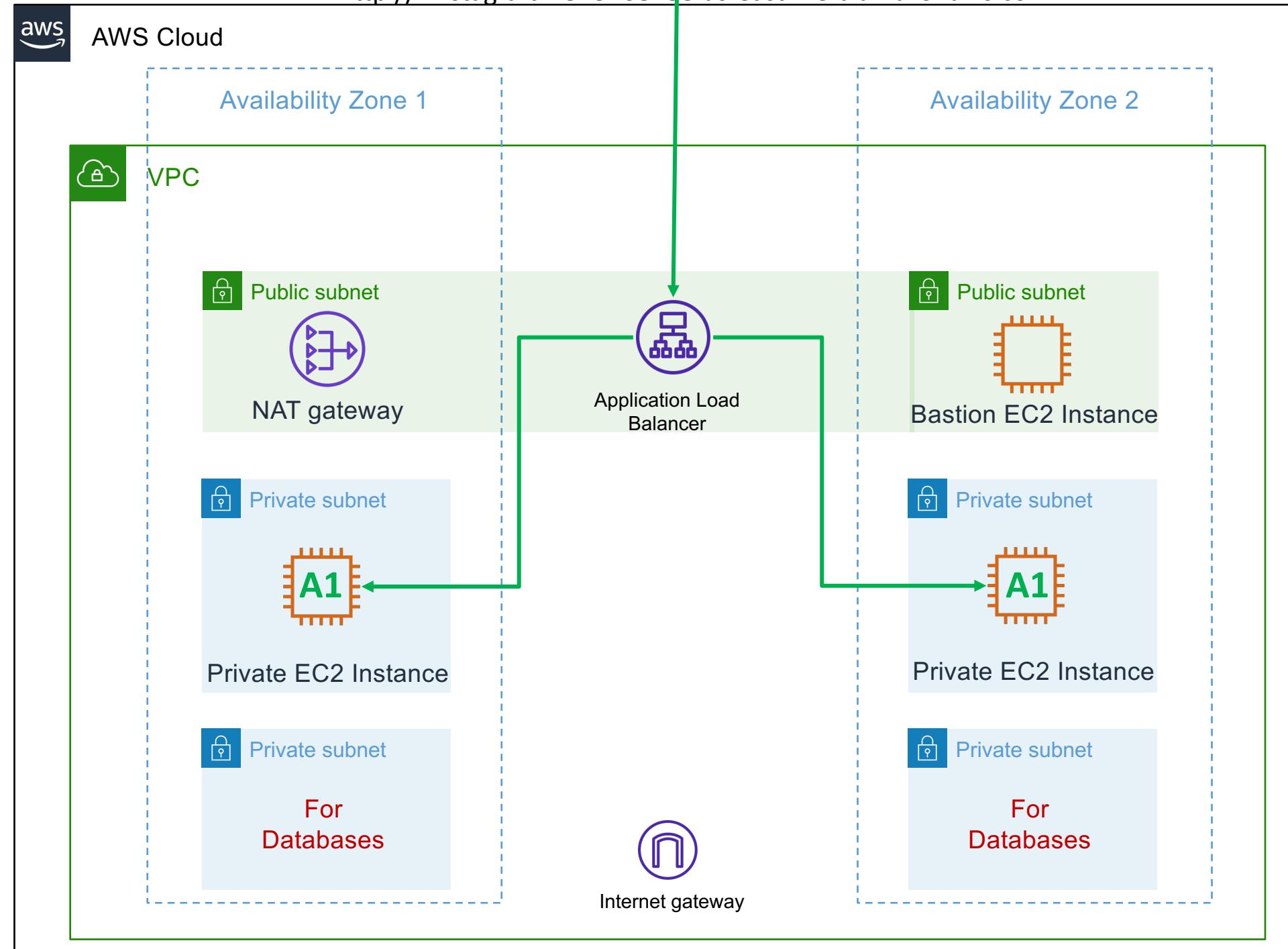
AWS VPC + EC2 Instance + Security Groups + AWS ALB Application Load Balancer

Terraform & AWS Concepts

Terraform Module: AWS ALB



<http://hr-stag-alb-1575108738.us-east-1.elb.amazonaws.com>



What are we going to learn ?

- ↳ c1-versions.tf
- ↳ c2-generic-variables.tf
- ↳ c3-local-values.tf
- ↳ c4-01-vpc-variables.tf
- ↳ c4-02-vpc-module.tf
- ↳ c4-03-vpc-outputs.tf
- ↳ c5-01-securitygroup-variables.tf
- ↳ c5-02-securitygroup-outputs.tf
- ↳ c5-03-securitygroup-bastionsg.tf
- ↳ c5-04-securitygroup-privatesg.tf
- ↳ **c5-05-securitygroup-loadbalancersg.tf**
- ↳ c6-01-datasource-ami.tf
- ↳ c7-01-ec2instance-variables.tf
- ↳ c7-02-ec2instance-outputs.tf
- ↳ c7-03-ec2instance-bastion.tf
- ↳ c7-04-ec2instance-private.tf
- ↳ c8-elasticip.tf
- ↳ c9-nullresource-provisioners.tf
- ↳ c10-01-ALB-application-loadbalancer-variables.tf
- ↳ c10-02-ALB-application-loadbalancer.tf
- ↳ **c10-03-ALB-application-loadbalancer-outputs.tf**
- ↳ ec2instance.auto.tfvars
- ↳ terraform.tfvars
- ↳ vpc.auto.tfvars

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



Amazon Virtual Private Cloud
(Amazon VPC)



Internet gateway



NAT gateway



Elastic IP address



Public Subnet



Private Subnet

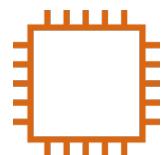
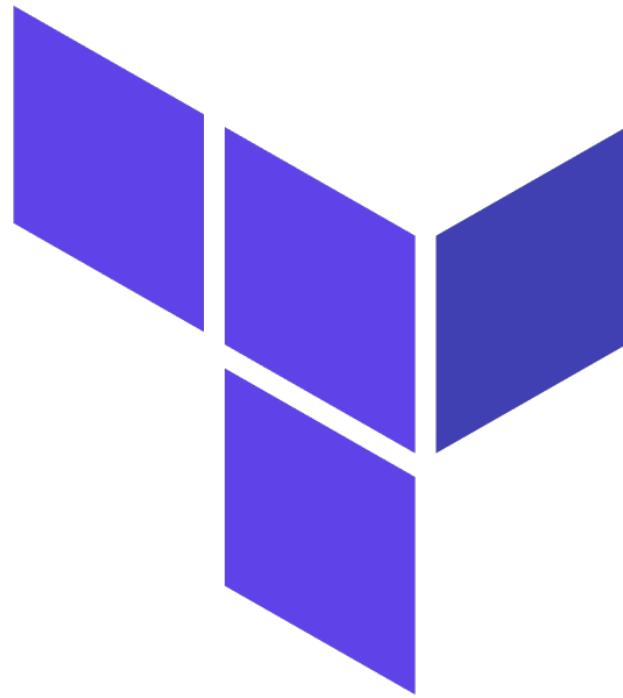
AWS

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

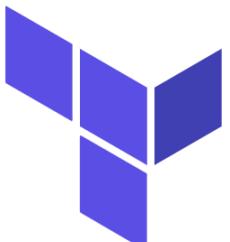
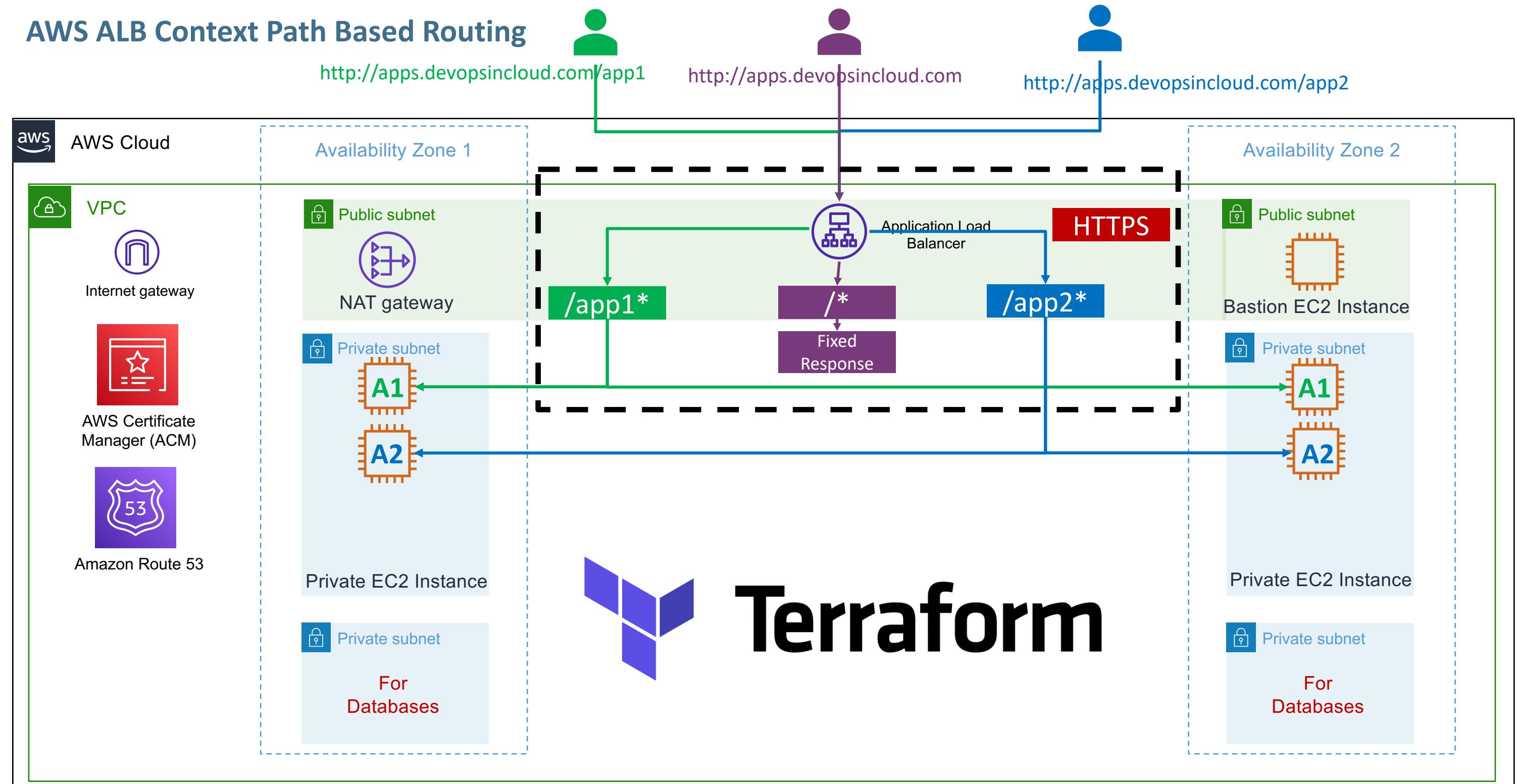


AWS Certificate Manager (ACM)

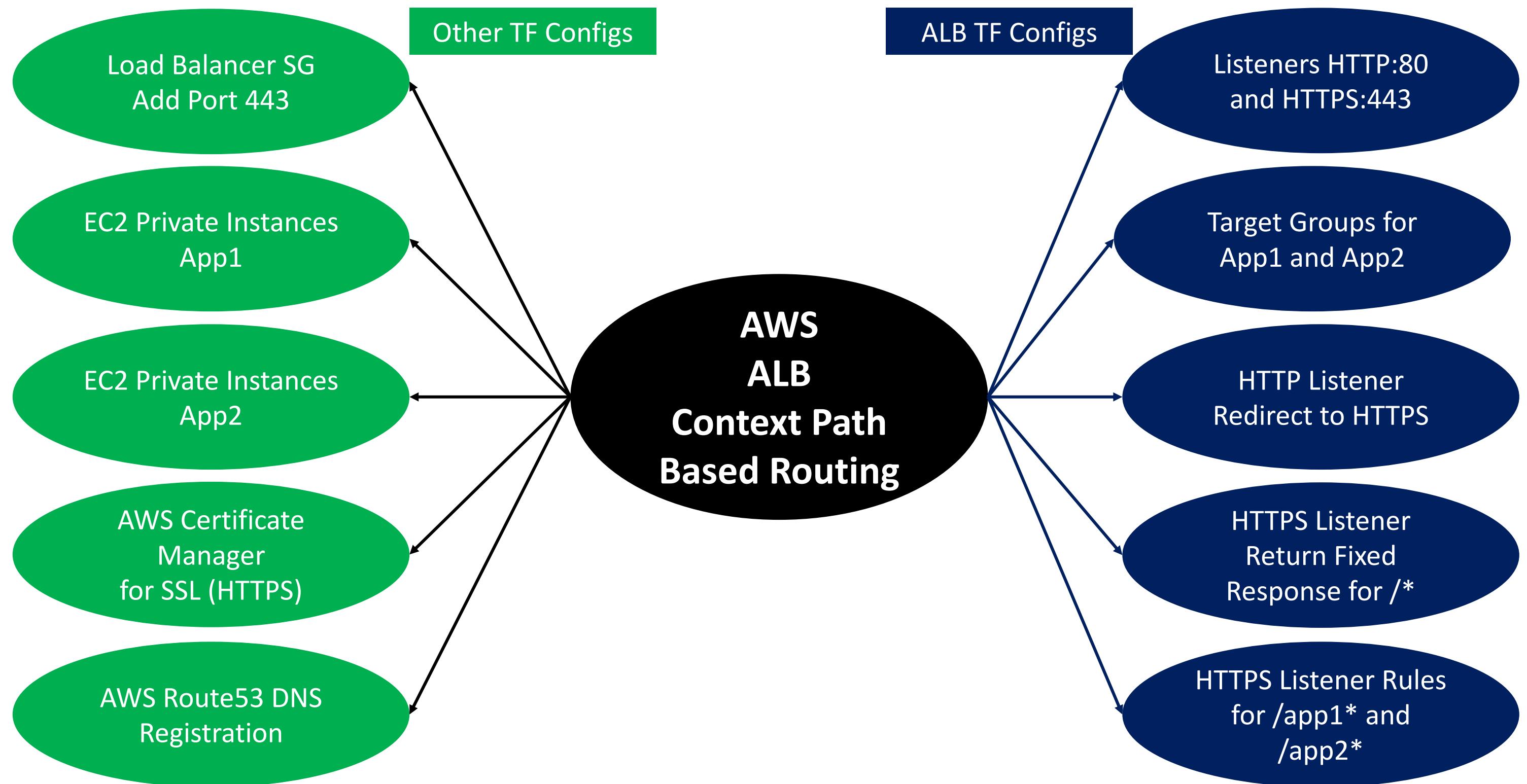


Amazon Route 53

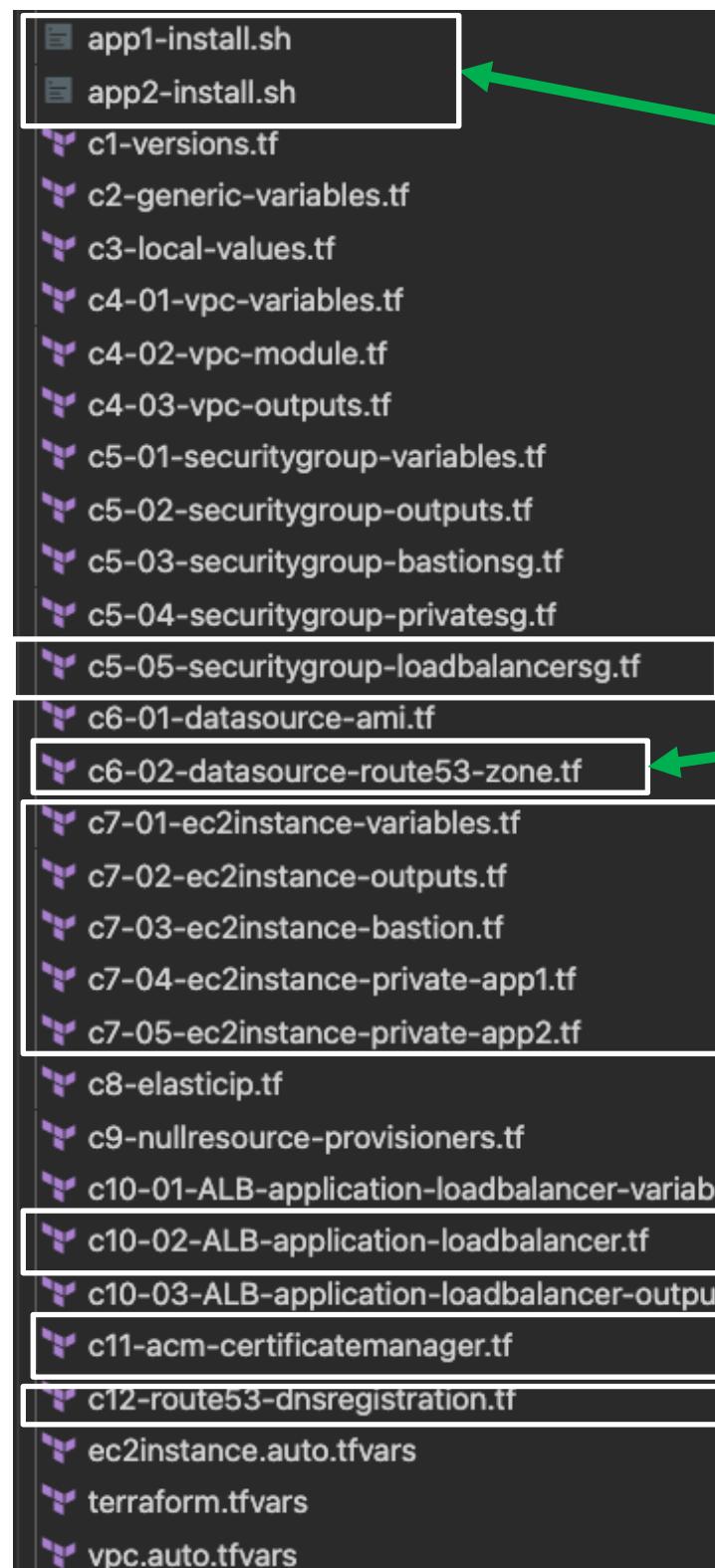
AWS ALB Context Path Based Routing



Terraform



What are we going to learn ?





Amazon Virtual Private Cloud
(Amazon VPC)



Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet

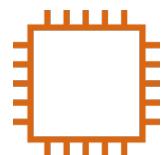
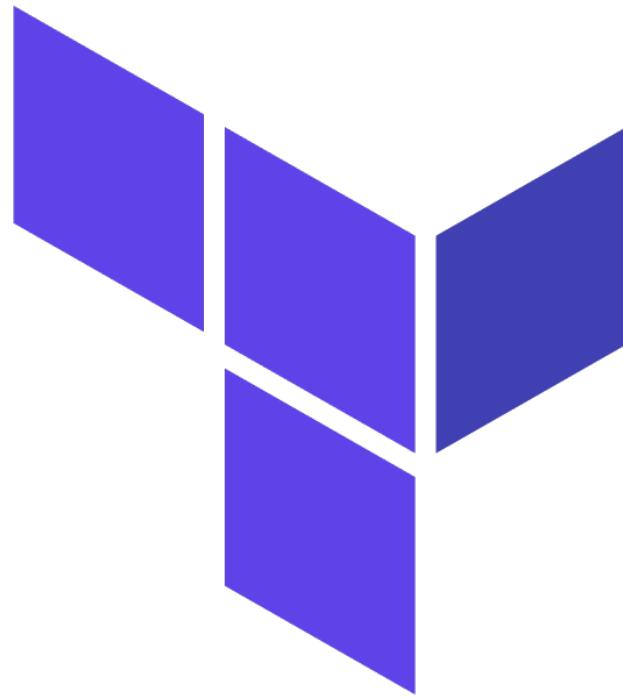
AWS

VPC with Web, App and DB Tiers

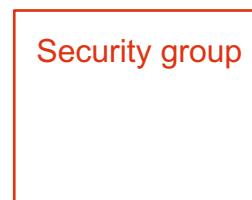
EC2 Instances, Bastion Host and Security Groups

Application Load Balancer

Host Header Based Routing



EC2 Instance



Security Groups



Classic Load Balancer



Application Load Balancer

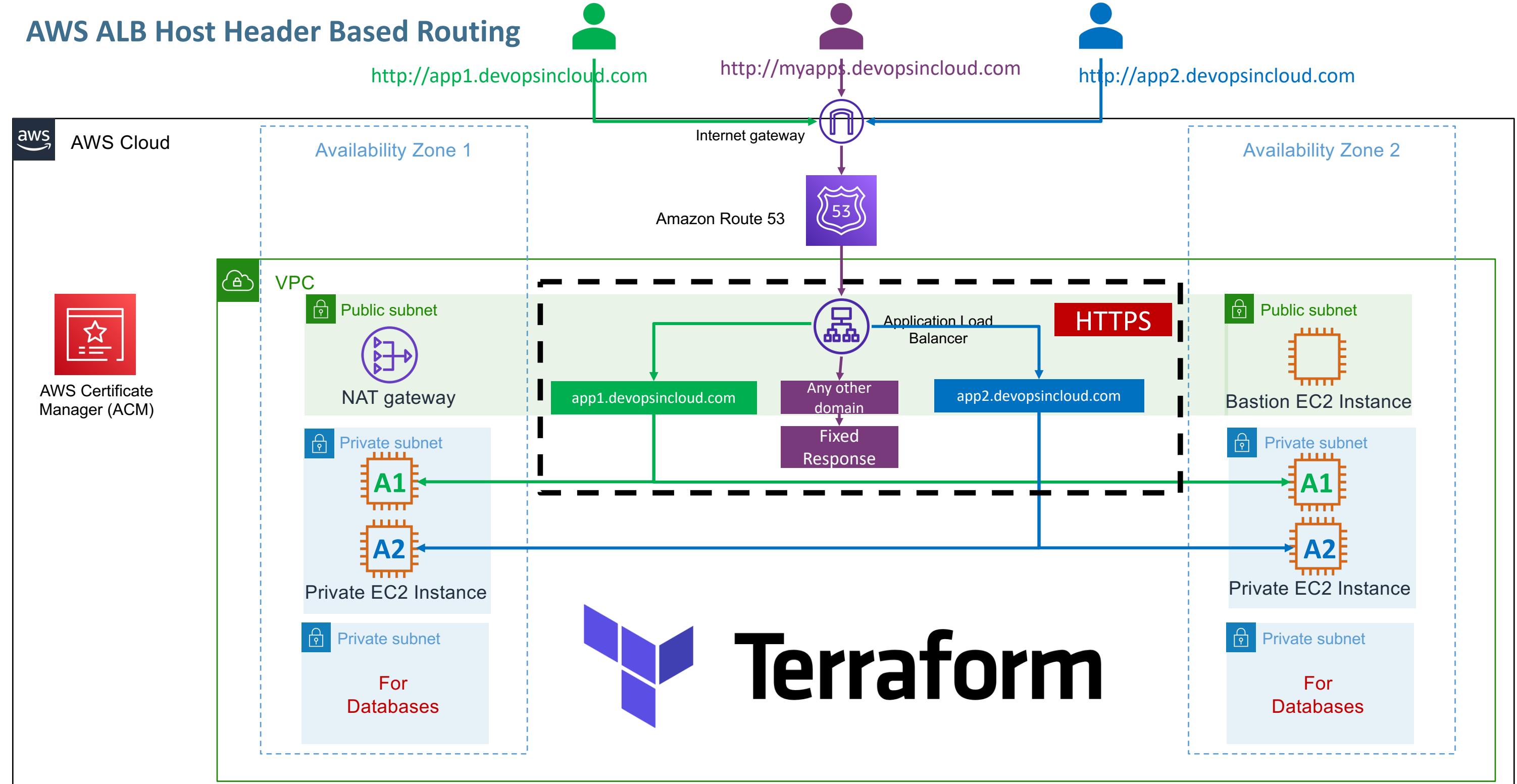


AWS Certificate Manager (ACM)



Amazon Route 53

AWS ALB Host Header Based Routing



What are we going to learn ?

- ↳ c1-versions.tf
- ↳ c2-generic-variables.tf
- ↳ c3-local-values.tf
- ↳ c4-01-vpc-variables.tf
- ↳ c4-02-vpc-module.tf
- ↳ c4-03-vpc-outputs.tf
- ↳ c5-01-securitygroup-variables.tf
- ↳ c5-02-securitygroup-outputs.tf
- ↳ c5-03-securitygroup-bastionssg.tf
- ↳ c5-04-securitygroup-privatesg.tf
- ↳ c5-05-securitygroup-loadbalancersg.tf
- ↳ c6-01-datasource-ami.tf
- ↳ c6-02-datasource-route53-zone.tf
- ↳ c7-01-ec2instance-variables.tf
- ↳ c7-02-ec2instance-outputs.tf
- ↳ c7-03-ec2instance-bastion.tf
- ↳ c7-04-ec2instance-private-app1.tf
- ↳ c7-05-ec2instance-private-app2.tf
- ↳ c8-elasticip.tf
- ↳ c9-nullresource-provisioners.tf
- ↳ c10-01-ALB-application-loadbalancer-variables.tf
- ↳ c10-02-ALB-application-loadbalancer.tf
- ↳ c10-03-ALB-application-loadbalancer-outputs.tf
- ↳ c11-acm-certificatemanager.tf
- ↳ c12-route53-dnsregistration.tf
- ↳ ec2instance.auto.tfvars
- ↳ loadbalancer.auto.tfvars
- ↳ terraform.tfvars
- ↳ vpc.auto.tfvars

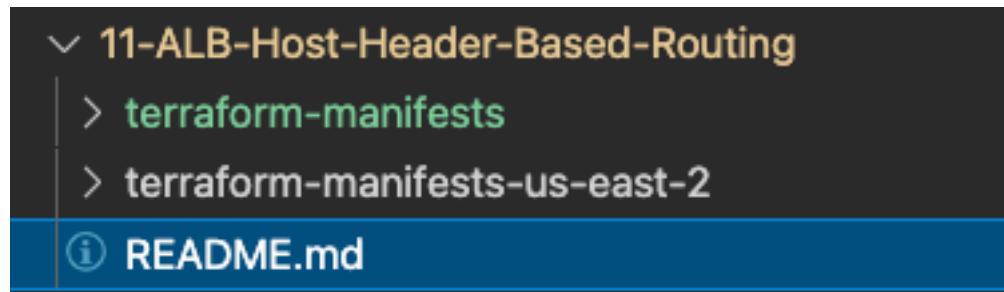
Changes related to ALB Variables and ALB Host Header settings

Changes related to Multiple DNS Names in Route53

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

AWS ACM Certificate Limit Issue

```
pp@pp-devopsincloud: ~]$ 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" {
```

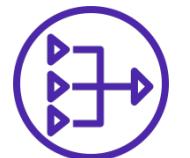




Amazon Virtual Private Cloud
(Amazon VPC)



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

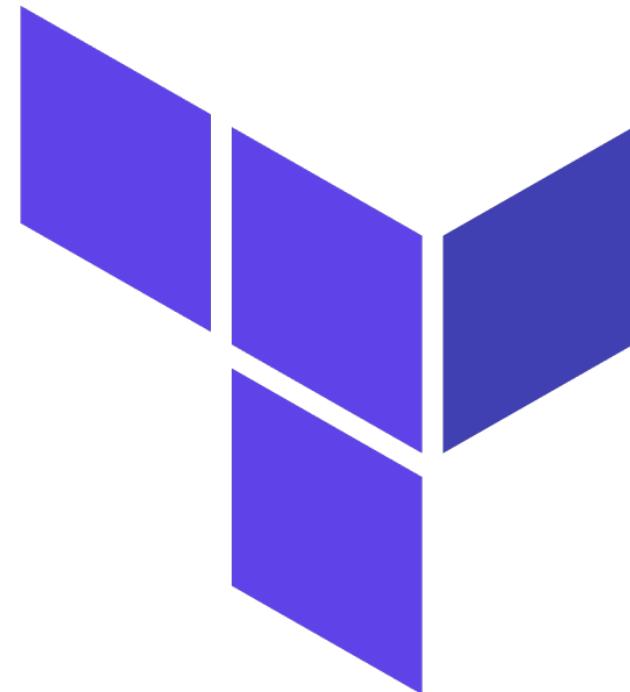
Application Load Balancer

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

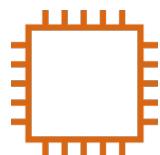
Rule-4: Host Header with 302 Redirect



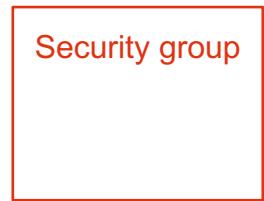
AWS Certificate
Manager (ACM)



Amazon Route 53



EC2 Instance



Security group



Classic Load
Balancer



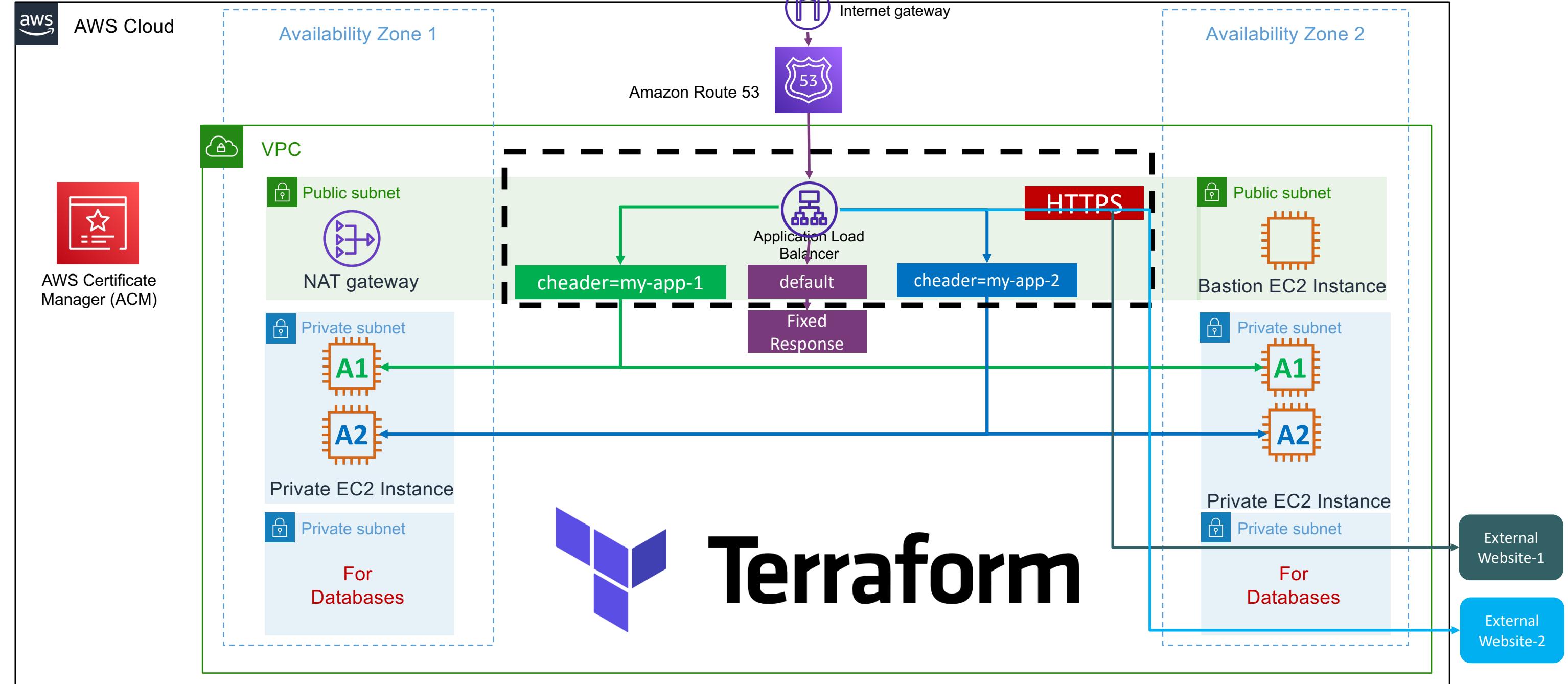
Application Load
Balancer

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

Host Header External Redirect 302

HTTP Header Routing

Query String External Redirect 302



What are we going to learn ?

- c1-versions.tf
- c2-generic-variables.tf
- c3-local-values.tf
- c4-01-vpc-variables.tf
- c4-02-vpc-module.tf
- c4-03-vpc-outputs.tf
- c5-01-securitygroup-variables.tf
- c5-02-securitygroup-outputs.tf
- c5-03-securitygroup-bastionssg.tf
- c5-04-securitygroup-privatesg.tf
- c5-05-securitygroup-loadbalancersg.tf
- c6-01-datasource-ami.tf
- c6-02-datasource-route53-zone.tf
- c7-01-ec2instance-variables.tf
- c7-02-ec2instance-outputs.tf
- c7-03-ec2instance-bastion.tf
- c7-04-ec2instance-private-app1.tf
- c7-05-ec2instance-private-app2.tf
- c8-elasticip.tf
- c9-nullresource-provisioners.tf
- c10-01-ALB-application-loadbalancer-variables.tf
- c10-02-ALB-application-loadbalancer.tf
- c10-03-ALB-application-loadbalancer-outputs.tf
- c11-acm-cryptomanager.tf
- c12-route53-dnsregistration.tf
- ec2instance.auto.tfvars
- loadbalancer.auto.tfvars
- terraform.tfvars
- vpc.auto.tfvars

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



Amazon Virtual Private Cloud
(Amazon VPC)



Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet

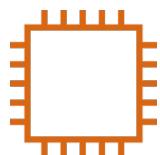
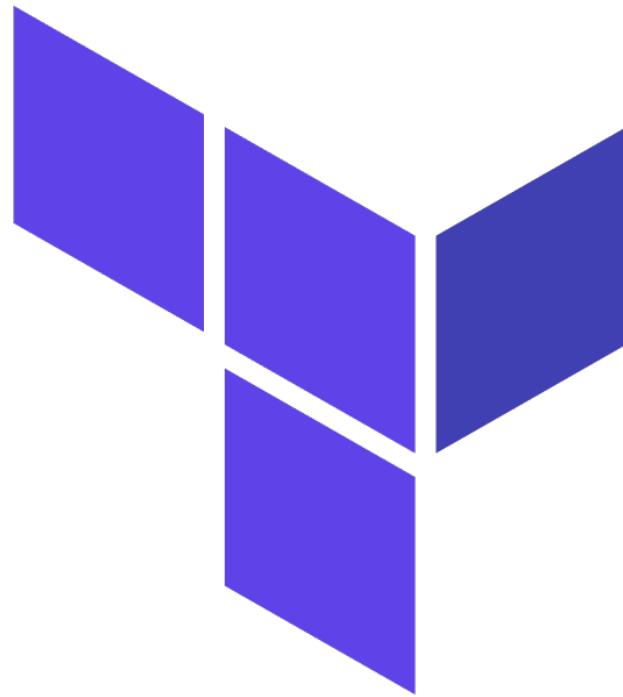
AWS

VPC with Web, App and DB Tiers

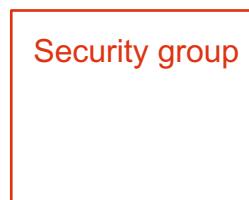
EC2 Instances, Bastion Host and Security Groups

Application Load Balancer

DNS to DB



EC2 Instance



Security Groups



Classic Load Balancer



Application Load Balancer



AWS Certificate Manager (ACM)



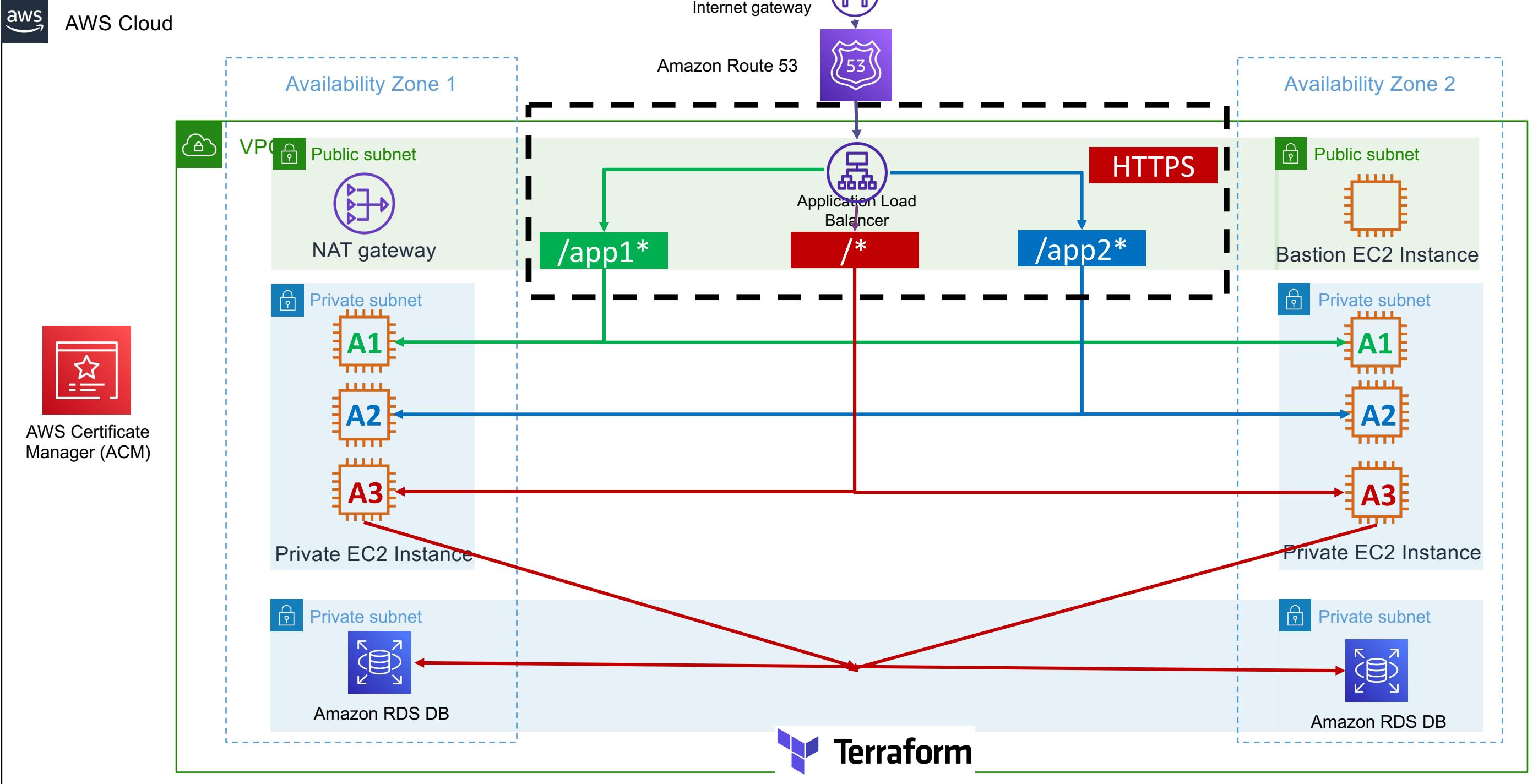
Amazon Route 53



Amazon RDS DB

AWS DNS to DB using Terraform

<http://dns-to-db.devopsincloud.com/app1>
<http://dns-to-db.devopsincloud.com/app2>
<http://dns-to-db.devopsincloud.com>



What are we going to learn ?

```
app1-install.sh  
app2-install.sh  
app3-ums-install.tpl  
c1-versions.tf  
c2-generic-variables.tf  
c3-local-values.tf  
c4-01-vpc-variables.tf  
c4-02-vpc-module.tf  
c4-03-vpc-outputs.tf  
c5-01-securitygroup-variables.tf  
c5-02-securitygroup-outputs.tf  
c5-03-securitygroup-bastionsg.tf  
c5-04-securitygroup-privatesg.tf  
c5-05-securitygroup-loadbalancersg.tf  
c5-06-securitygroup-rdsdbsg.tf  
c6-01-datasource-ami.tf  
c6-02-datasource-route53-zone.tf  
c7-01-ec2instance-variables.tf  
c7-02-ec2instance-outputs.tf  
c7-03-ec2instance-bastion.tf  
c7-04-ec2instance-private-app1.tf  
c7-05-ec2instance-private-app2.tf  
c7-06-ec2instance-private-app3.tf  
c8-elasticip.tf  
c9-nullresource-provisioners.tf
```

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

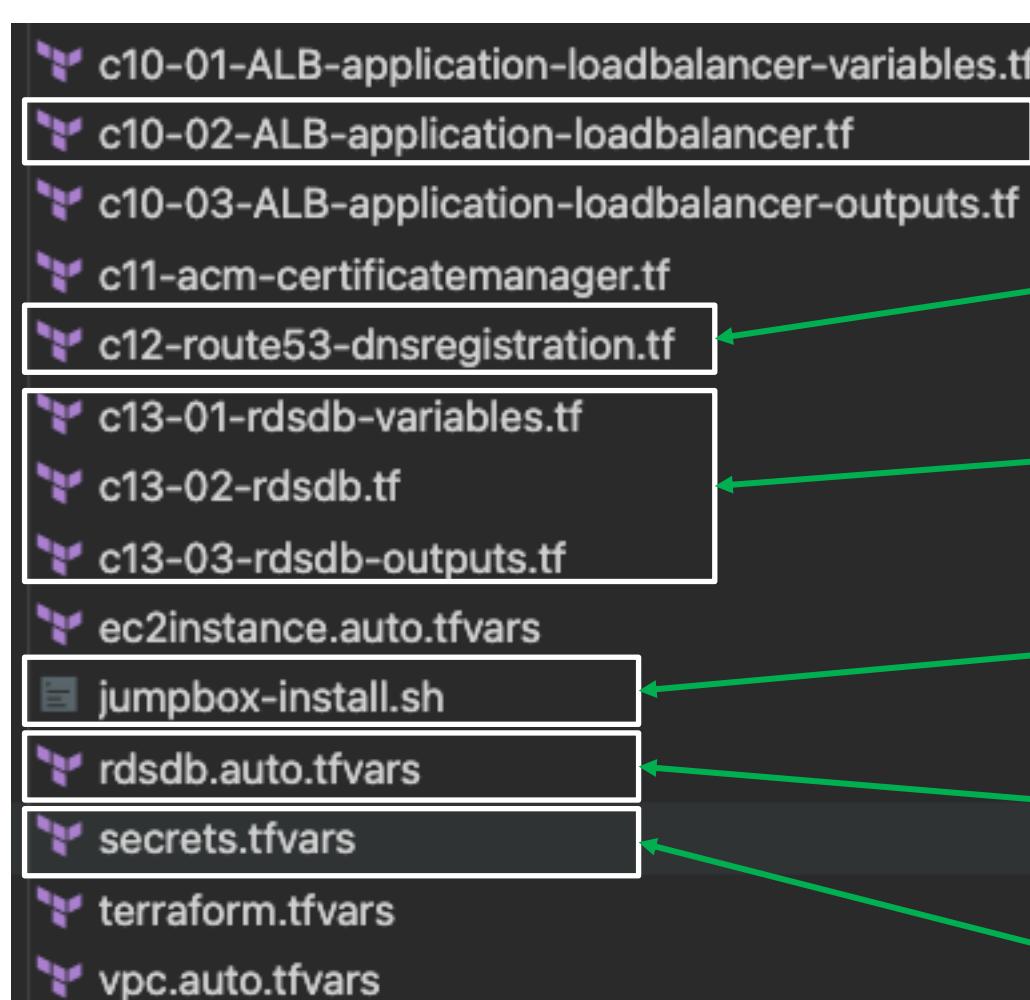
RDS Database Security Group

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

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

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.



Amazon Virtual Private Cloud
(Amazon VPC)



Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet

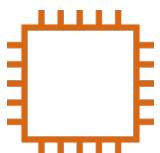
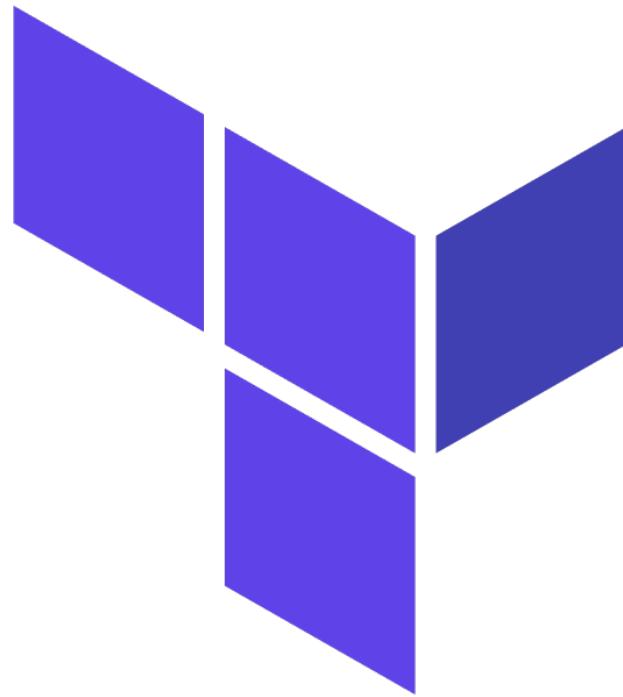
AWS

VPC with Web, App and DB Tiers

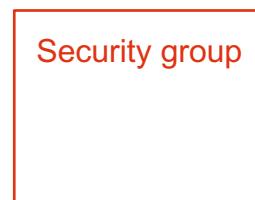
EC2 Instances, Bastion Host and Security Groups

Application Load Balancer

Autoscaling with Launch Configurations



EC2 Instance



Security Groups



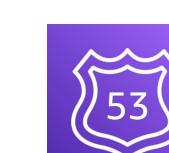
Classic Load Balancer



Application Load Balancer



AWS Certificate Manager (ACM)



Amazon Route 53



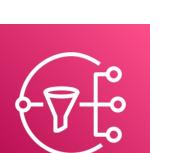
Amazon RDS DB



Autoscaling



AWS IAM

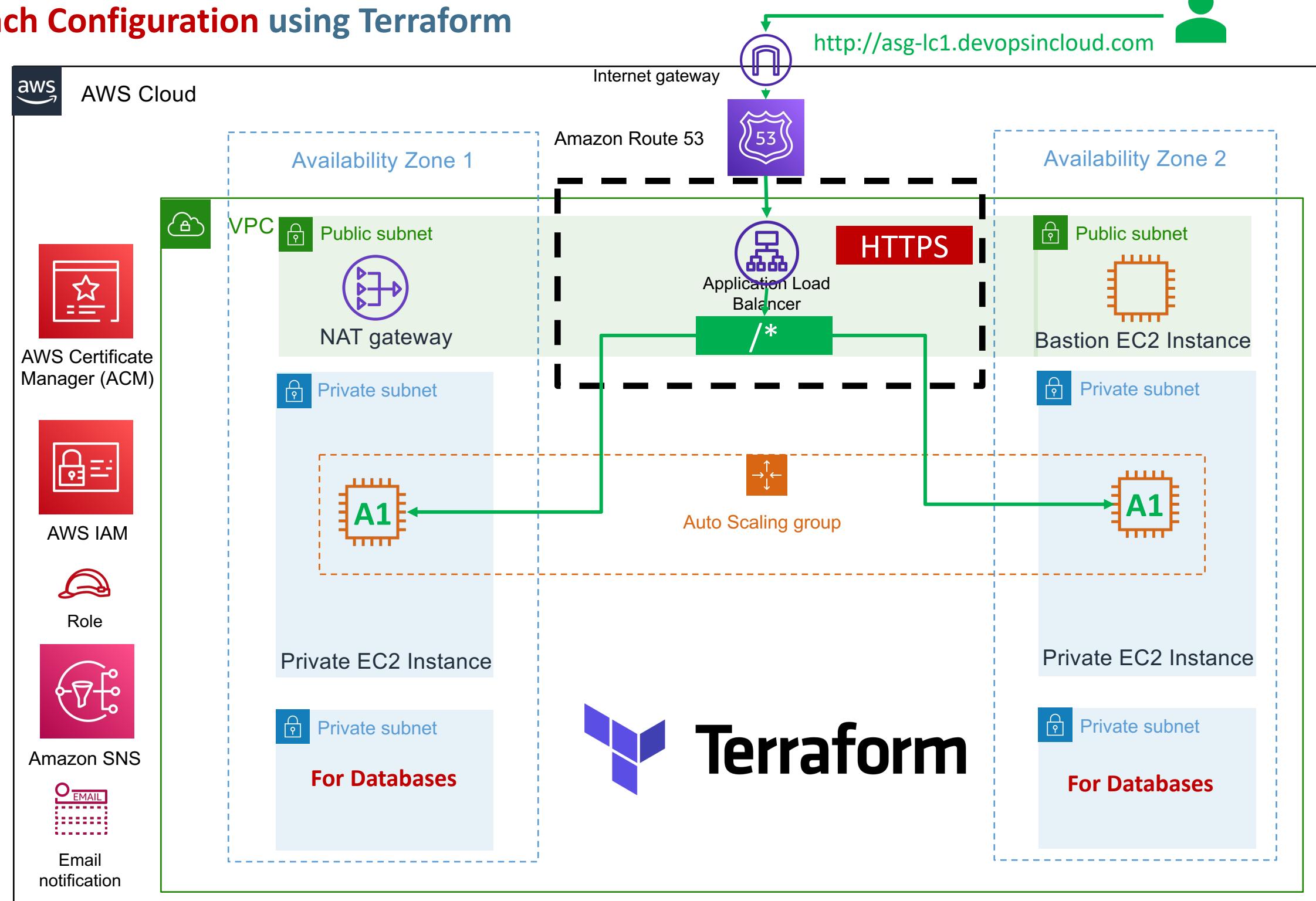


Amazon SNS

AWS Autoscaling with Launch Configuration using Terraform



<http://asg-lc1.devopsincloud.com>



What are we going to learn ?

```
└─ terraform-manifests
    └─ local-exec-output-files
    └─ private-key
    └─ app1-install.sh
    └─ c1-versions.tf
    └─ c2-generic-variables.tf
    └─ c3-local-values.tf
    └─ c4-01-vpc-variables.tf
    └─ c4-02-vpc-module.tf
    └─ c4-03-vpc-outputs.tf
    └─ c5-01-securitygroup-variables.tf
    └─ c5-02-securitygroup-outputs.tf
    └─ c5-03-securitygroup-bastionsg.tf
    └─ c5-04-securitygroup-privatesg.tf
    └─ c5-05-securitygroup-loadbalancersg.tf
    └─ c6-01-datasource-ami.tf
    └─ c6-02-datasource-route53-zone.tf
    └─ c7-01-ec2instance-variables.tf
    └─ c7-02-ec2instance-outputs.tf
    └─ c7-03-ec2instance-bastion.tf
    └─ c8-elasticip.tf
    └─ c9-nullresource-provisioners.tf
```

Added Autoscaling related tags

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

What are we going to learn ?

```
└── c10-01-ALB-application-loadbalancer-variables.tf  
└── c10-02-ALB-application-loadbalancer.tf  
└── c10-03-ALB-application-loadbalancer-outputs.tf  
└── c11-acm-cryptomanager.tf  
└── c12-route53-dnsregistration.tf  
└── c13-01-autoscaling-with-launchconfiguration-variables.tf  
└── c13-02-autoscaling-additional-resources.tf  
└── c13-03-autoscaling-with-launchconfiguration.tf  
└── c13-04-autoscaling-with-launchconfiguration-outputs.tf  
└── c13-05-autoscaling-notifications.tf  
└── c13-06-autoscaling-ttsp.tf  
└── c13-07-autoscaling-scheduled-actions.tf  
└── ec2instance.auto.tfvars  
└── terraform.tfvars  
└── vpc.auto.tfvars
```

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

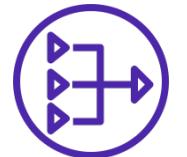
Autoscaling Group TTSP and Scheduled Actions



Amazon Virtual Private Cloud
(Amazon VPC)



Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet

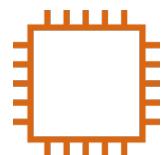
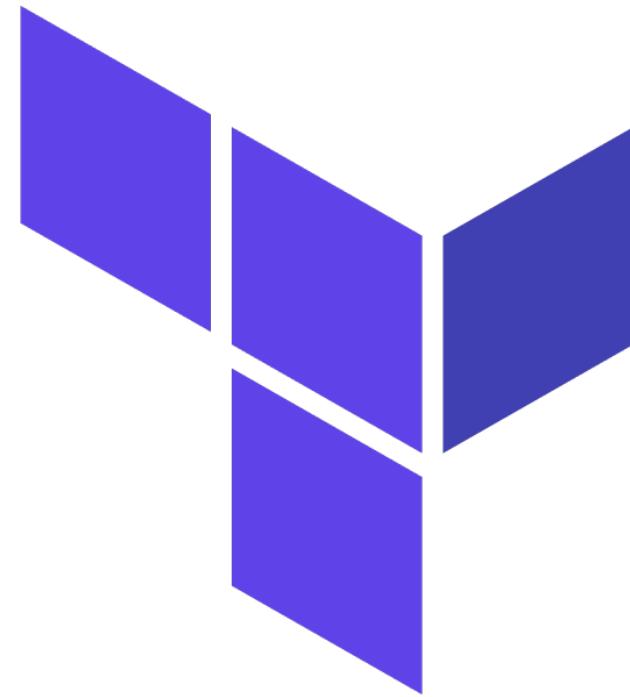
AWS

VPC with Web, App and DB Tiers

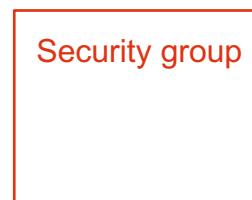
EC2 Instances, Bastion Host and Security Groups

Application Load Balancer

Autoscaling with Launch Templates



EC2 Instance



Security Groups



Classic Load Balancer



Application Load Balancer



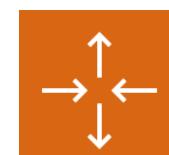
AWS Certificate Manager (ACM)



Amazon Route 53



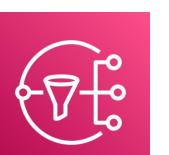
Amazon RDS DB



Autoscaling



AWS IAM



Amazon SNS

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.

AWS Autoscaling with Launch Templates using Terraform



<http://asg-lt1.devopsincloud.com>

Terraform ASG Resource

ASG with Launch Templates

ASG Instance Refresh

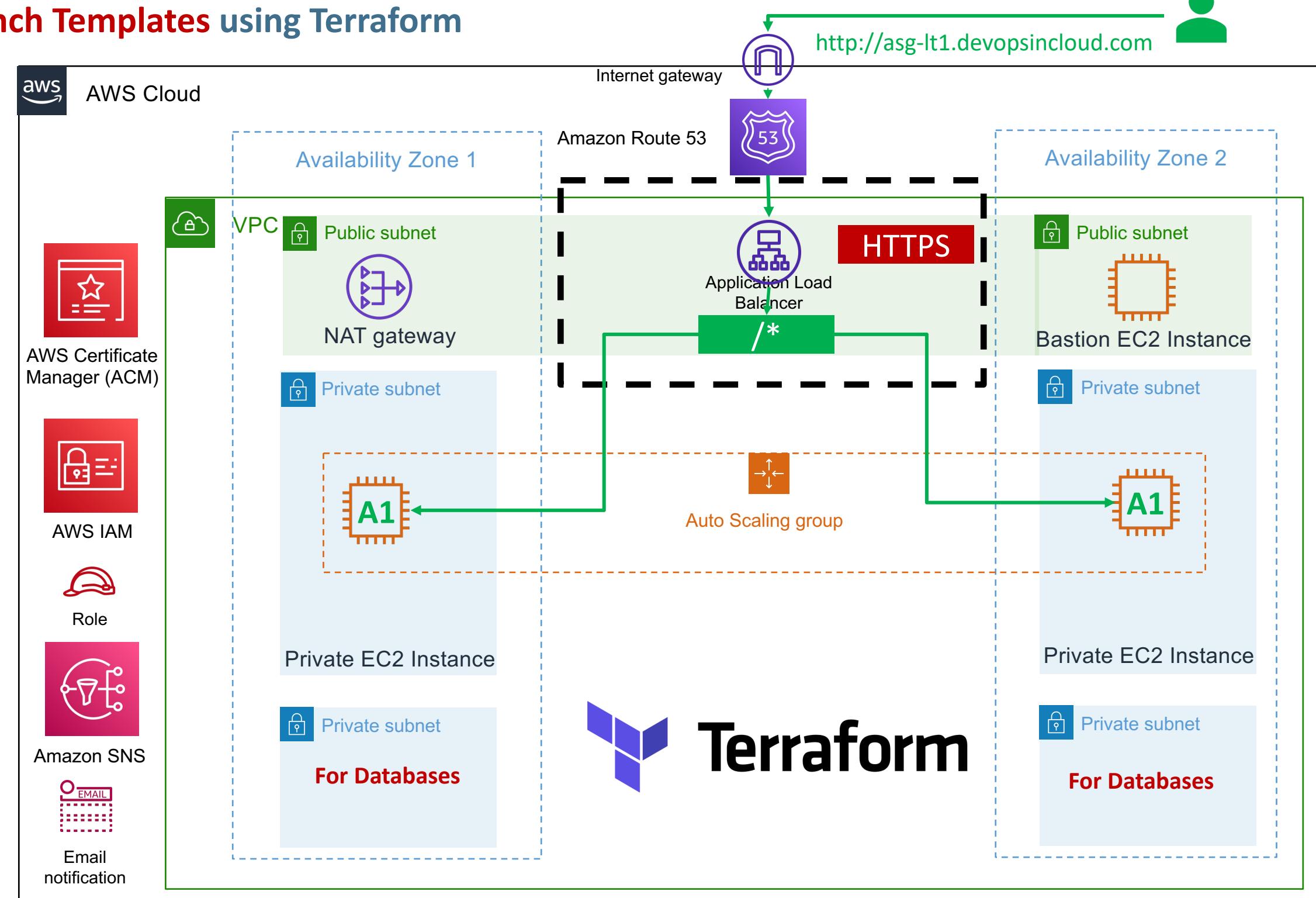
ASG Lifecycle Hooks

ASG TTSP

ASG Scheduled Actions

ASG Notifications

ASG Autoscaling Tests



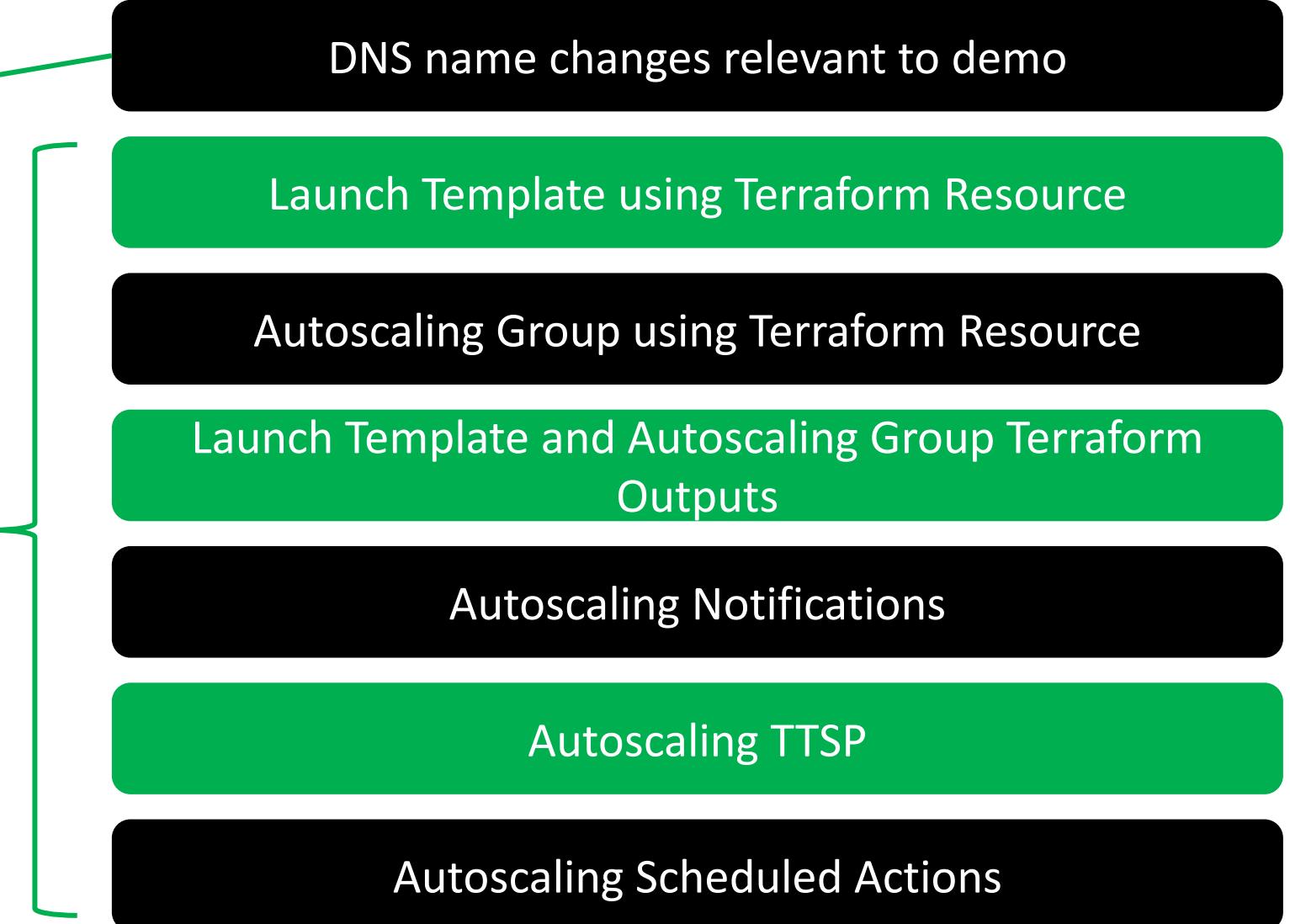
What are we going to learn ?

```
✓ terraform-manifests
  > local-exec-output-files
  > private-key
  └ app1-install.sh
    └ c1-versions.tf
    └ c2-generic-variables.tf
    └ c3-local-values.tf
    └ c4-01-vpc-variables.tf
    └ c4-02-vpc-module.tf
    └ c4-03-vpc-outputs.tf
    └ c5-01-securitygroup-variables.tf
    └ c5-02-securitygroup-outputs.tf
    └ c5-03-securitygroup-bastionsg.tf
    └ c5-04-securitygroup-privatesg.tf
    └ c5-05-securitygroup-loadbalancersg.tf
    └ c6-01-datasource-ami.tf
    └ c6-02-datasource-route53-zone.tf
    └ c7-01-ec2instance-variables.tf
    └ c7-02-ec2instance-outputs.tf
    └ c7-03-ec2instance-bastion.tf
    └ c8-elasticip.tf
    └ c9-nullresource-provisioners.tf
    └ c10-01-ALB-application-loadbalancer-variables.tf
    └ c10-02-ALB-application-loadbalancer.tf
    └ c10-03-ALB-application-loadbalancer-outputs.tf
```

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

What are we going to learn ?

```
└── c11-acm-certificatemanager.tf  
└── c12-route53-dnsregistration.tf  
└── c13-01-autoscaling-with-launchtemplate-variables.tf  
└── c13-02-autoscaling-launchtemplate-resource.tf  
└── c13-03-autoscaling-resource.tf  
└── c13-04-autoscaling-with-launchtemplate-outputs.tf  
└── c13-05-autoscaling-notifications.tf  
└── c13-06-autoscaling-ttsp.tf  
└── c13-07-autoscaling-scheduled-actions.tf  
└── ec2instance.auto.tfvars  
└── terraform.tfvars  
└── vpc.auto.tfvars
```





Amazon Virtual Private Cloud
(Amazon VPC)



Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet

AWS

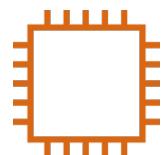
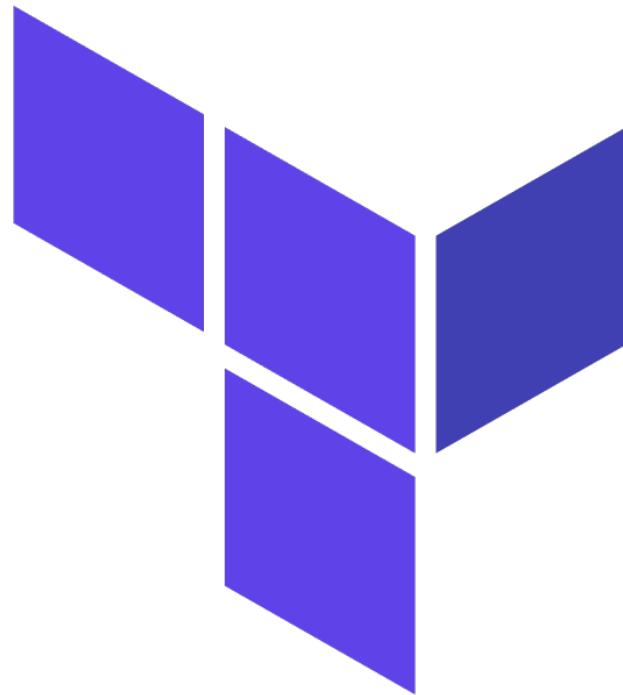
VPC with Web, App and DB Tiers

EC2 Instances, Bastion Host and Security Groups

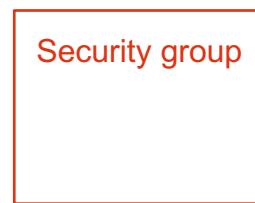
Autoscaling with Launch Templates

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



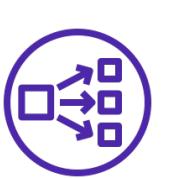
Autoscaling



AWS IAM



Amazon SNS



Network Load Balancer

AWS Network Load Balancer with TCP & TLS Listeners using Terraform

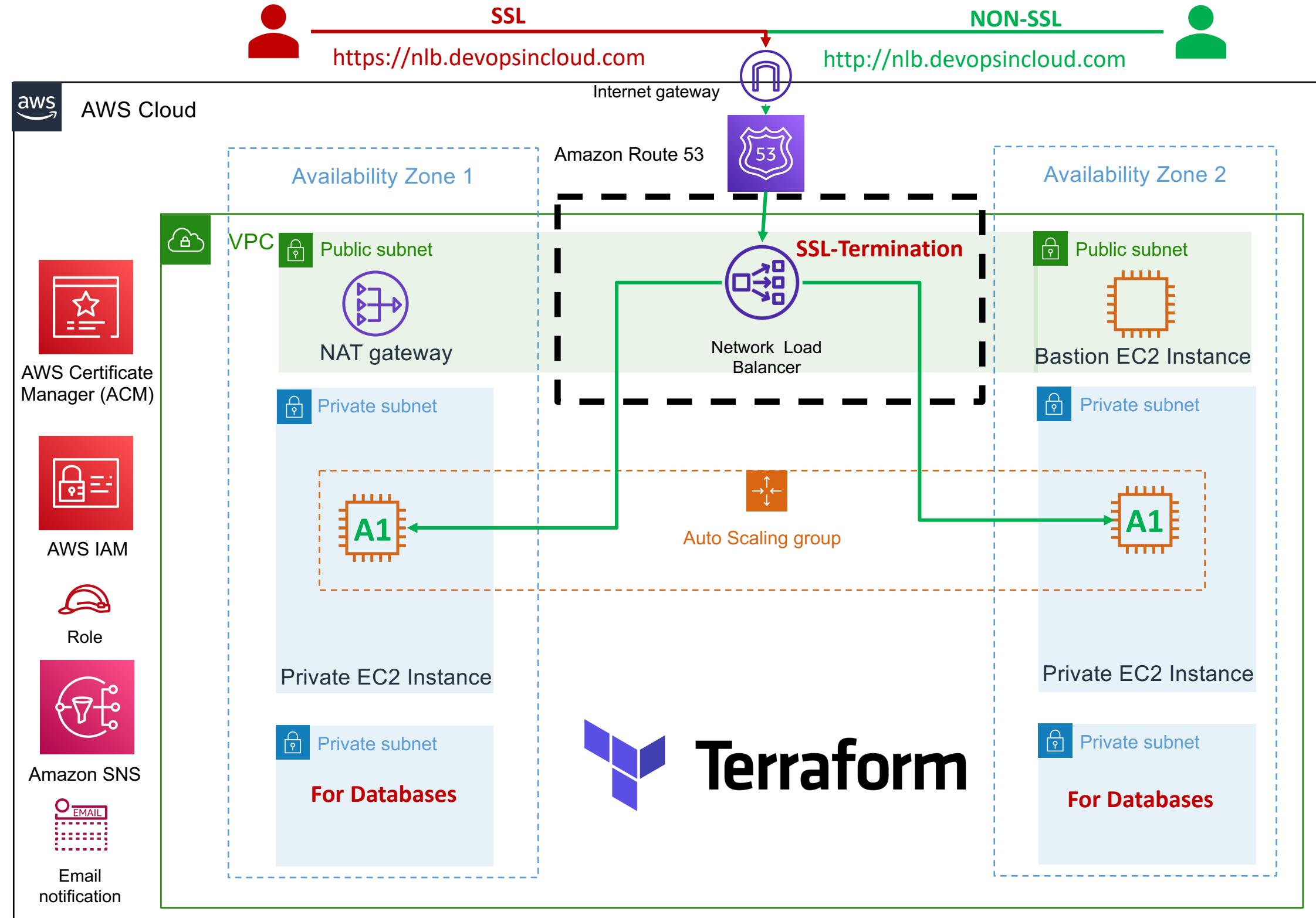
Terraform NLB Module

Terraform NLB TCP Listeners

Terraform NLB TLS Listeners

Terraform NLB Target Groups

Terraform NLB with EC2 Autoscaling Groups



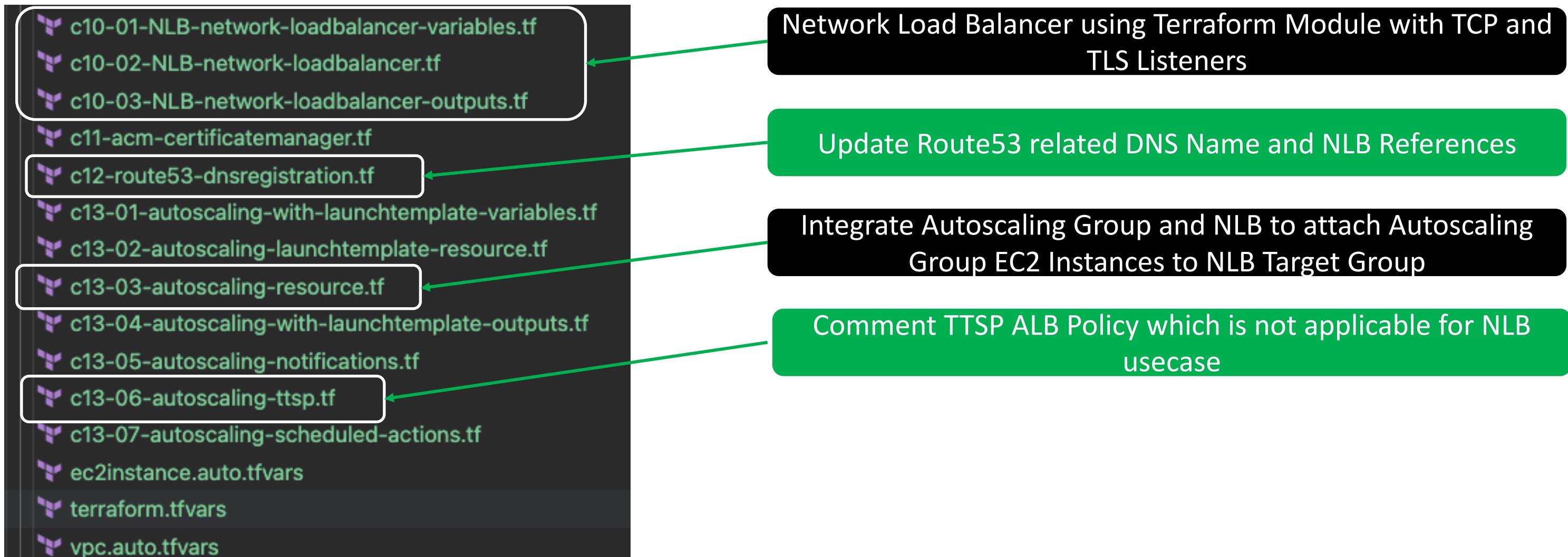
What are we going to learn ?

```
└─ terraform-manifests
    └─ .terraform
    └─ local-exec-output-files
    └─ private-key
    └─ .terraform.lock.hcl
    └─ app1-install.sh
    └─ c1-versions.tf
    └─ c2-generic-variables.tf
    └─ c3-local-values.tf
    └─ c4-01-vpc-variables.tf
    └─ c4-02-vpc-module.tf
    └─ c4-03-vpc-outputs.tf
    └─ c5-01-securitygroup-variables.tf
    └─ c5-02-securitygroup-outputs.tf
    └─ c5-03-securitygroup-bastionssg.tf
    └─ c5-04-securitygroup-privatesg.tf
    └─ c5-05-securitygroup-loadbalancerssg.tf
    └─ c6-01-datasource-ami.tf
    └─ c6-02-datasource-route53-zone.tf
    └─ c7-01-ec2instance-variables.tf
    └─ c7-02-ec2instance-outputs.tf
    └─ c7-03-ec2instance-bastion.tf
    └─ c8-elasticip.tf
    └─ c9-nullresource-provisioners.tf
```

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

No changes from c1 to c9
except c5-04

What are we going to learn ?

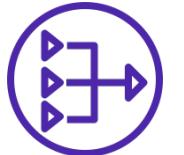




Amazon Virtual Private Cloud
(Amazon VPC)



Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet

AWS

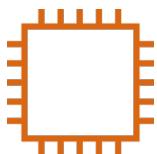
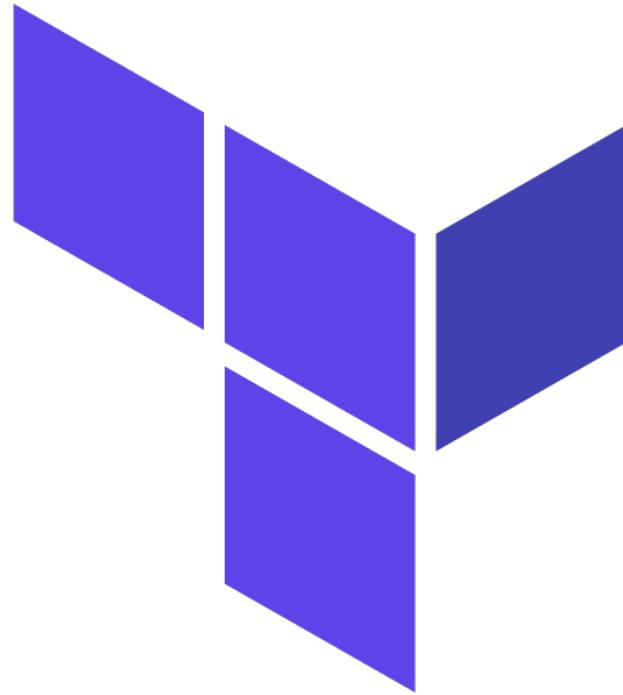
VPC with Web, App and DB Tiers

EC2 Instances, Bastion Host and Security Groups

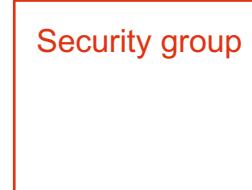
Application Load Balancer

Autoscaling with Launch Templates

CloudWatch Alarms



EC2 Instance



Security Groups



Classic Load Balancer



Application Load Balancer



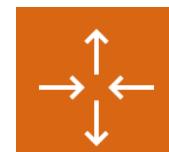
AWS Certificate Manager (ACM)



Amazon Route 53



Amazon RDS DB



Autoscaling



AWS IAM



Amazon SNS

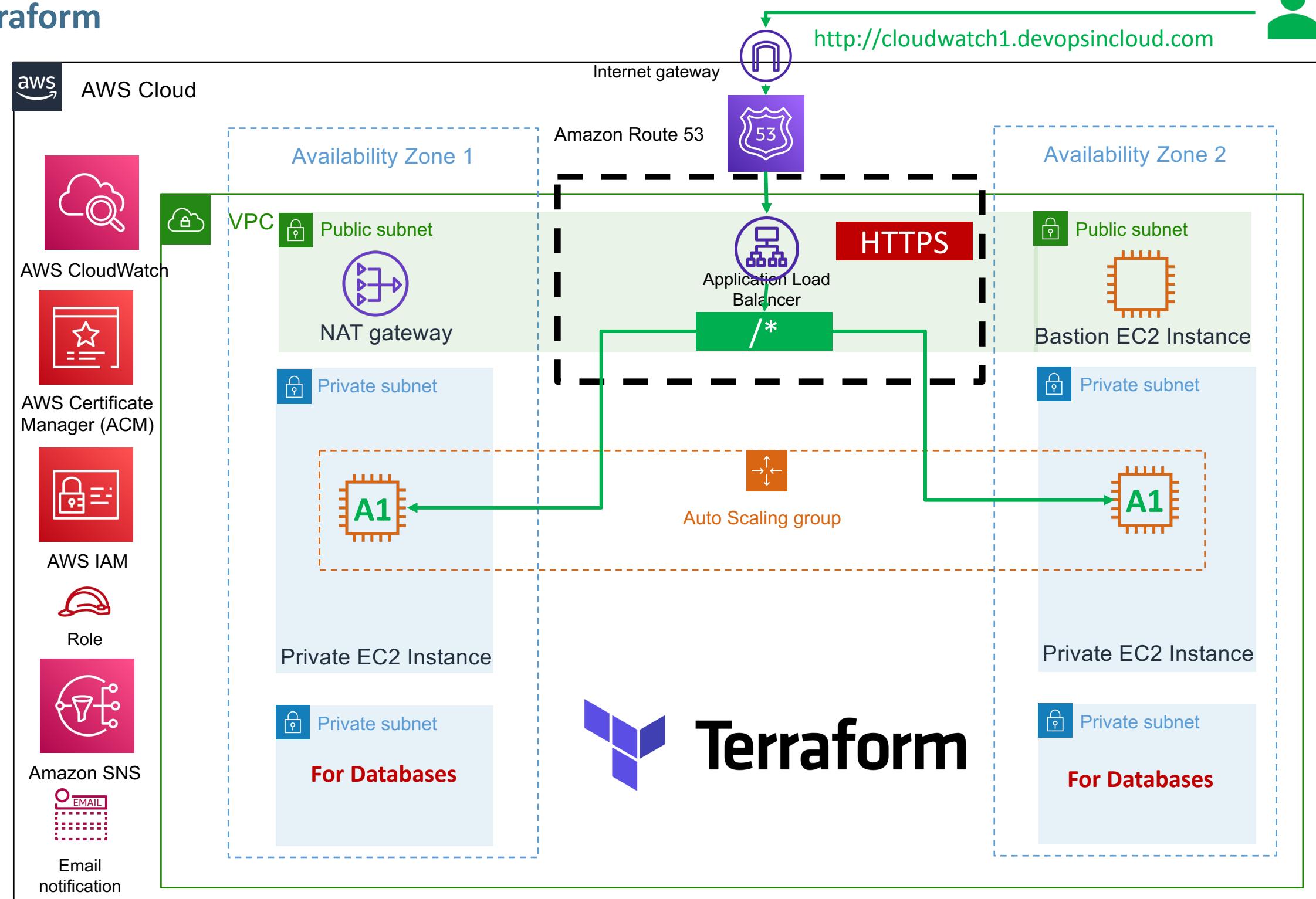


CloudWatch

AWS CloudWatch using Terraform



<http://cloudwatch1.devopsincloud.com>



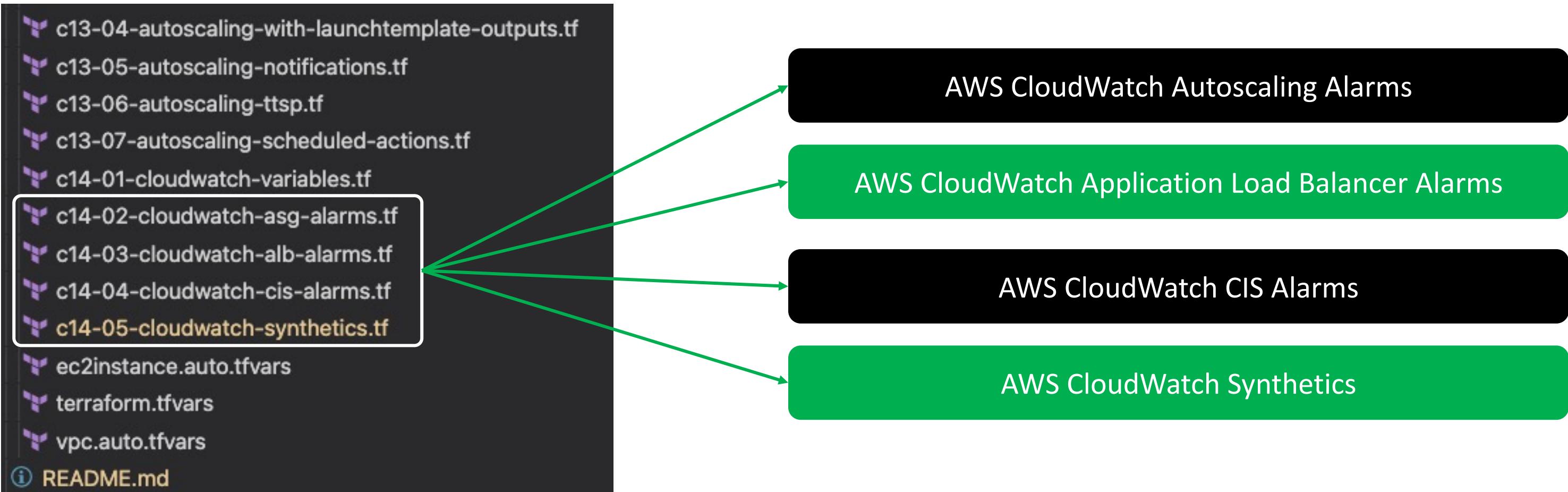
```
✓ terraform-manifests
  > local-exec-output-files
  > private-key
  > sswebsite2
    └─ app1-install.sh
    └─ c1-versions.tf
    └─ c2-generic-variables.tf
    └─ c3-local-values.tf
    └─ c4-01-vpc-variables.tf
    └─ c4-02-vpc-module.tf
    └─ c4-03-vpc-outputs.tf
    └─ c5-01-securitygroup-variables.tf
    └─ c5-02-securitygroup-outputs.tf
    └─ c5-03-securitygroup-bastionsg.tf
    └─ c5-04-securitygroup-privatesg.tf
    └─ c5-05-securitygroup-loadbalancersg.tf
    └─ c6-01-datasource-ami.tf
    └─ c6-02-datasource-route53-zone.tf
    └─ c7-01-ec2instance-variables.tf
    └─ c7-02-ec2instance-outputs.tf
    └─ c7-03-ec2instance-bastion.tf
    └─ c8-elasticip.tf
    └─ c9-nullresource-provisioners.tf
    └─ c10-01-ALB-application-loadbalancer-variables.tf
    └─ c10-02-ALB-application-loadbalancer.tf
    └─ c10-03-ALB-application-loadbalancer-outputs.tf
    └─ c11-acm-certificatemanager.tf
    └─ c12-route53-dnsregistration.tf
    └─ c13-01-autoscaling-with-launchtemplate-variables.tf
    └─ c13-02-autoscaling-launchtemplate-resource.tf
    └─ c13-03-autoscaling-resource.tf
```

What are we going to learn ?

AWS CloudWatch Synthetics related Node JS Files for Heart Beat Monitor

No changes from
c1 to c13

What are we going to learn ?





Amazon Virtual Private Cloud
(Amazon VPC)



Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



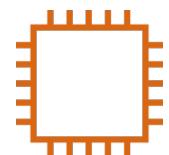
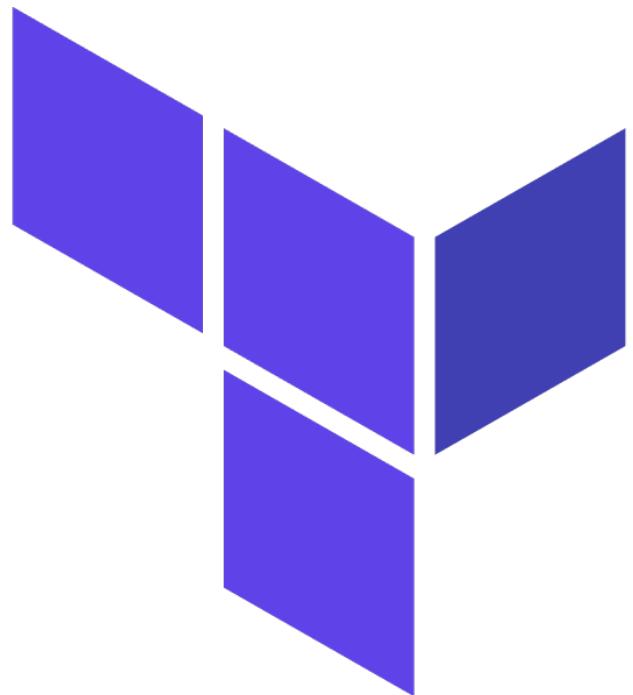
Amazon Simple Storage
Service (Amazon S3)



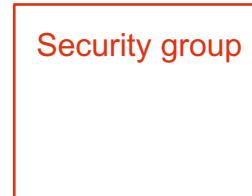
Amazon
DynamoDB

AWS

VPC with Web, App and DB Tiers Build Terraform Modules Locally



EC2 Instance



Security Groups



Classic Load
Balancer



Application Load
Balancer



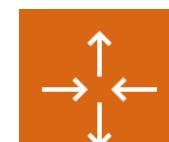
AWS Certificate
Manager (ACM)



Amazon Route 53



Amazon RDS DB



Autoscaling



AWS IAM



Amazon SNS



CloudWatch

Build Terraform Modules Locally

Build from Scratch

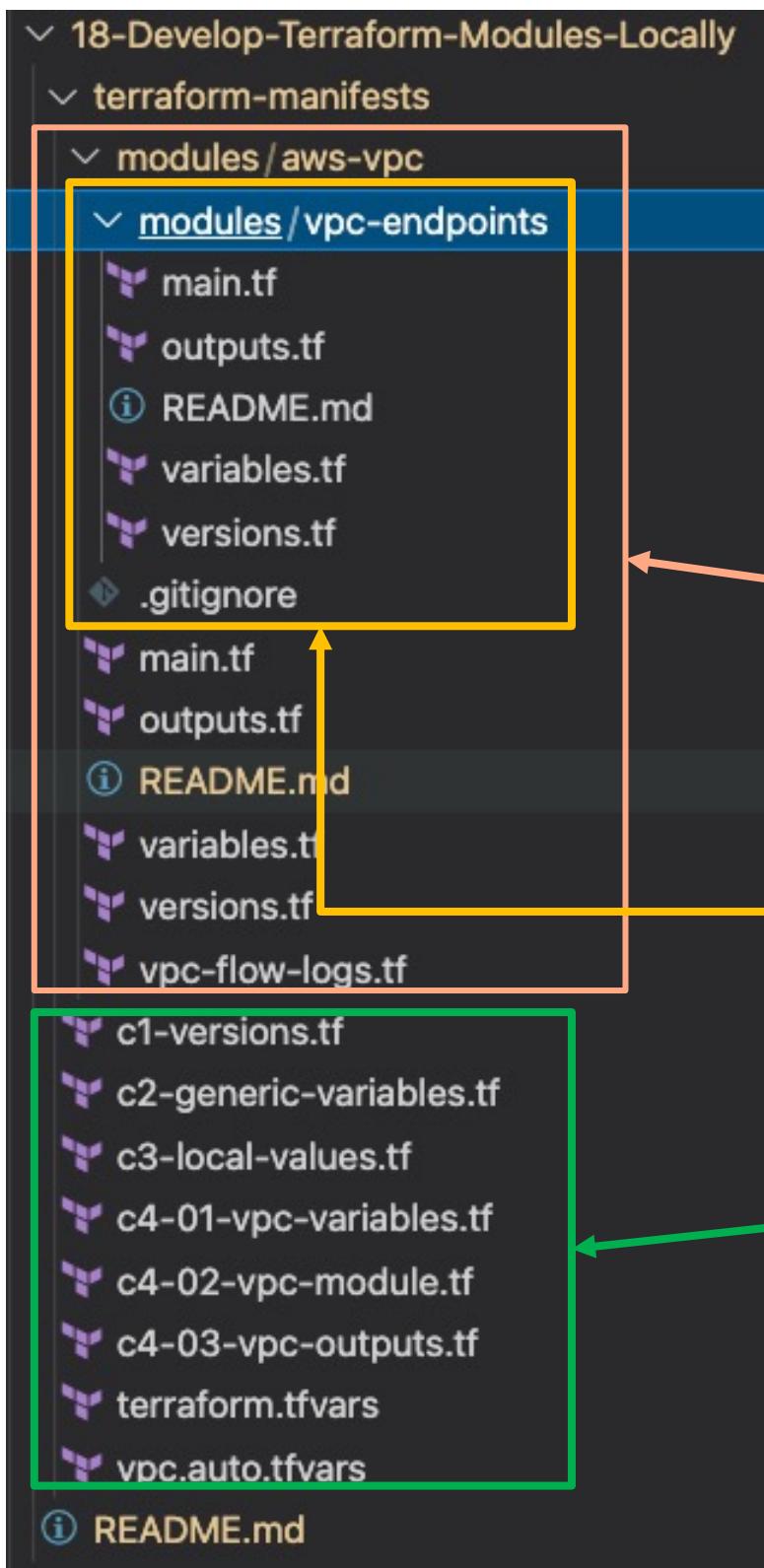
If already not available, then use this approach

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



What are we going to learn ?

AWS VPC Module

AWS VPC Modules - VPC Endpoints Sub Module

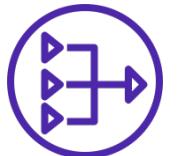
Root Module calling the VPC Local Module



Amazon Virtual Private Cloud
(Amazon VPC)



Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



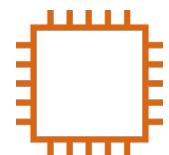
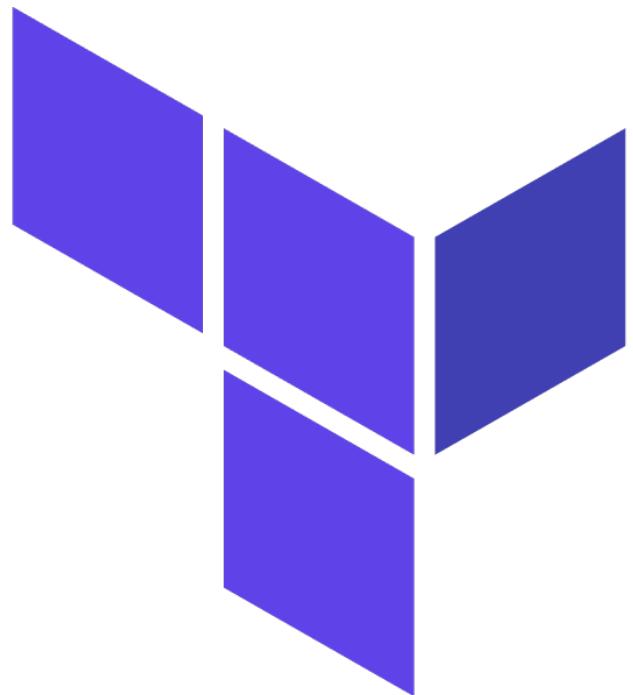
Amazon Simple Storage
Service (Amazon S3)



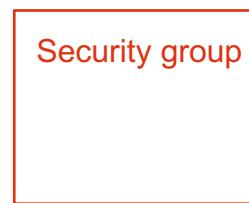
Amazon
DynamoDB

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



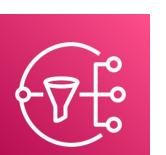
Amazon RDS DB



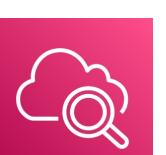
Autoscaling



AWS IAM



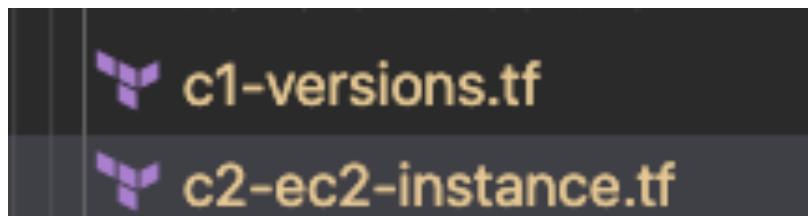
Amazon SNS



CloudWatch

Desired & Current Terraform States

Terraform Configuration Files



Real World Resource – EC2 Instance

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
web	i-0663449fef49e9cf5	Running	t2.micro	2/2 checks ...	No alarms +	us-east-1b
Instance: i-0663449fef49e9cf5 (web)						
Details	Security	Networking	Storage	Status checks	Monitoring	Tags
Instance ID i-0663449fef49e9cf5 (web)	Public IPv4 address 54.144.73.100 open address	Private IPv4 addresses 172.31.94.137	Public IPv4 DNS ec2-54-144-73-100.compute-1.amazonaws.com open address	Private IPv4 DNS ip-172-31-94-137.ec2.internal	VPC ID vpc-54972d2e (default-vpc)	Subnet ID subnet-d2e590fc
Instance state Running	Elastic IP addresses -	AWS Compute Optimizer finding Opt-in to AWS Compute Optimizer for recommendations.	IAM Role -			

Desired State



Current State

Terraform State

Terraform
Local
State
Storage

Terraform
Remote
State
Storage

What is Terraform Backend ?

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

Terraform
State Storage



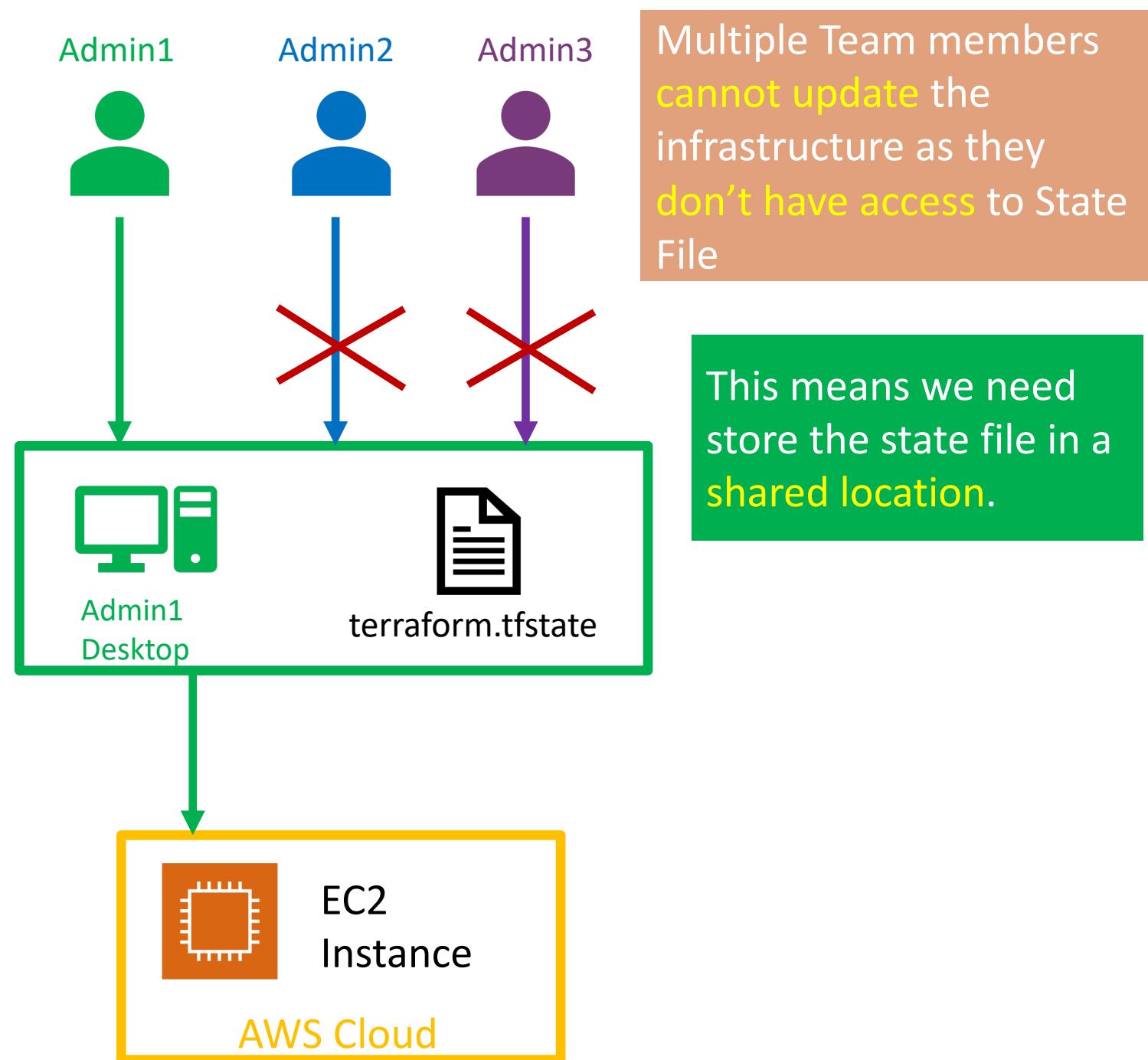
AWS S3 Bucket

Terraform
State Locking

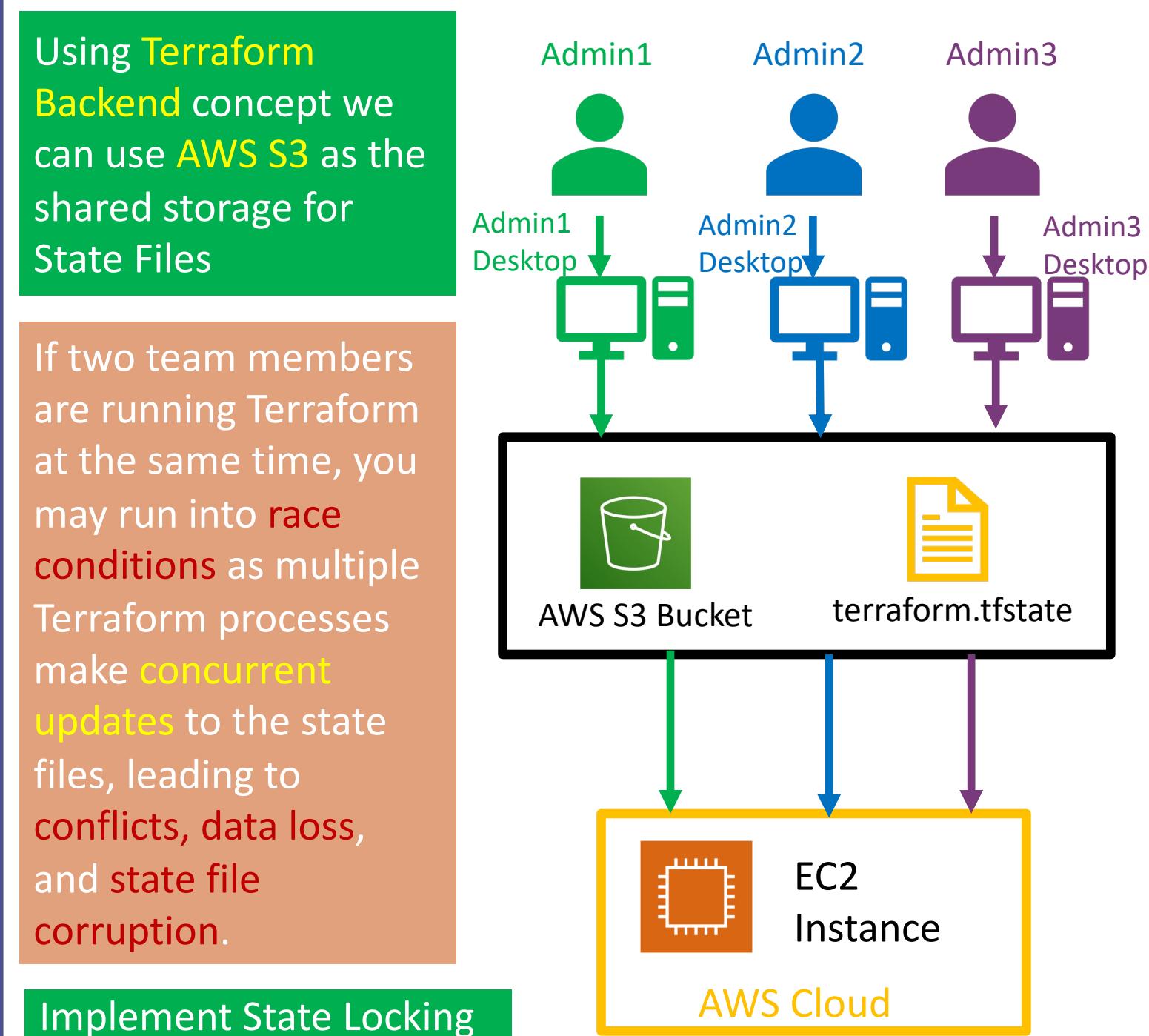


AWS DynamoDB

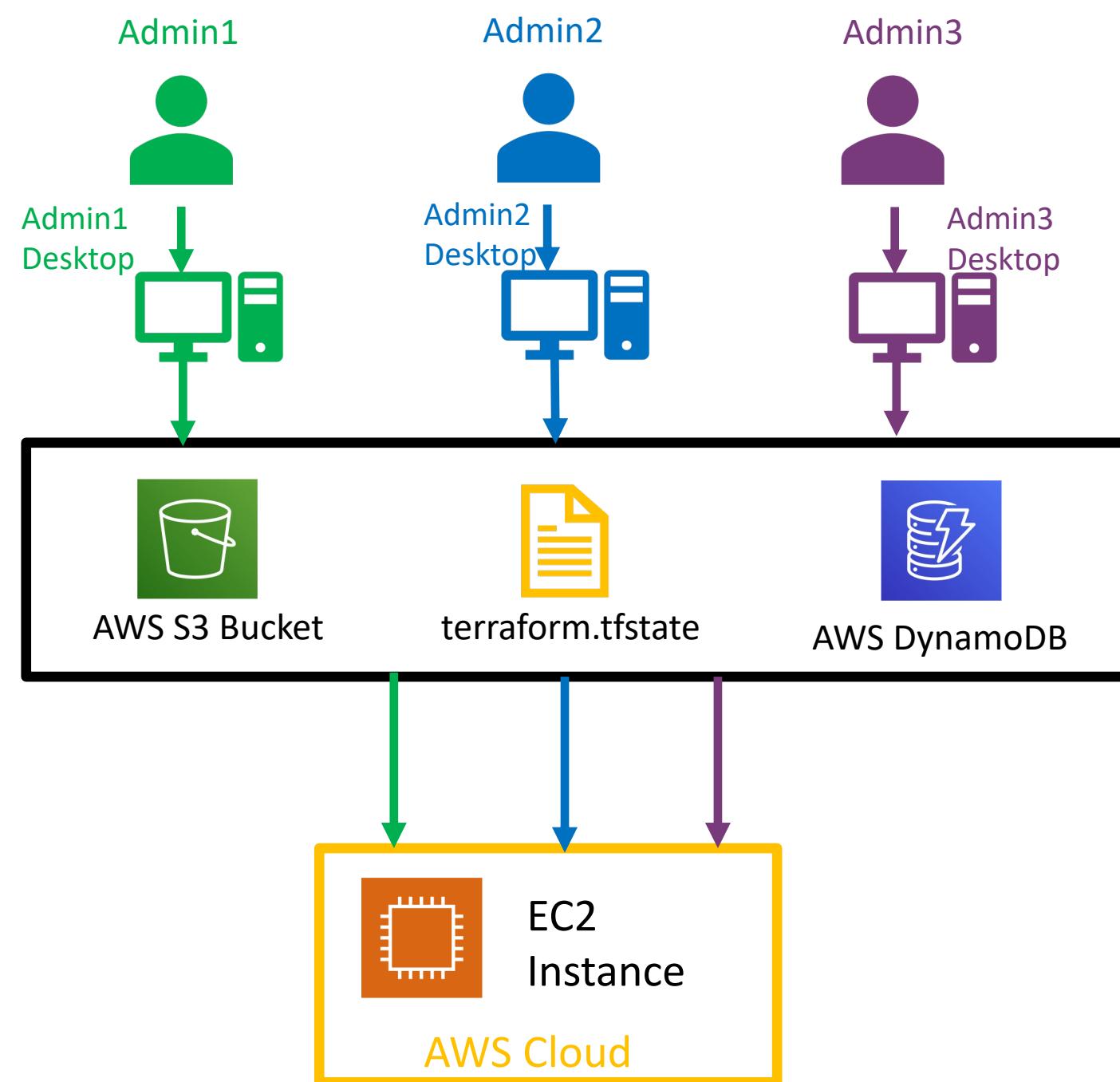
Local State File



Remote State File



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.

Terraform Remote State File with State Locking

Terraform State Storage to Remote Backend

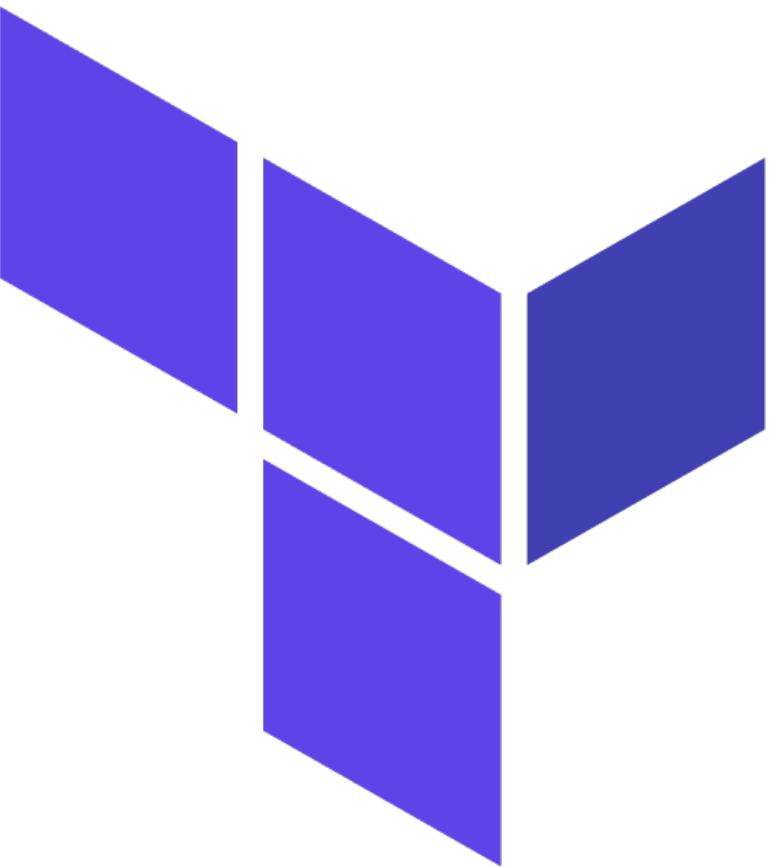
Terraform State Locking

```
# 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-stack simplify"
    key    = "dev/terraform.tfstate"
    region = "us-east-1"
}

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

Terraform Backends



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

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

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**.

What are Operations ?
`terraform apply`
`terraform destroy`

Terraform Backends

Backend Types

Enhanced Backends

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

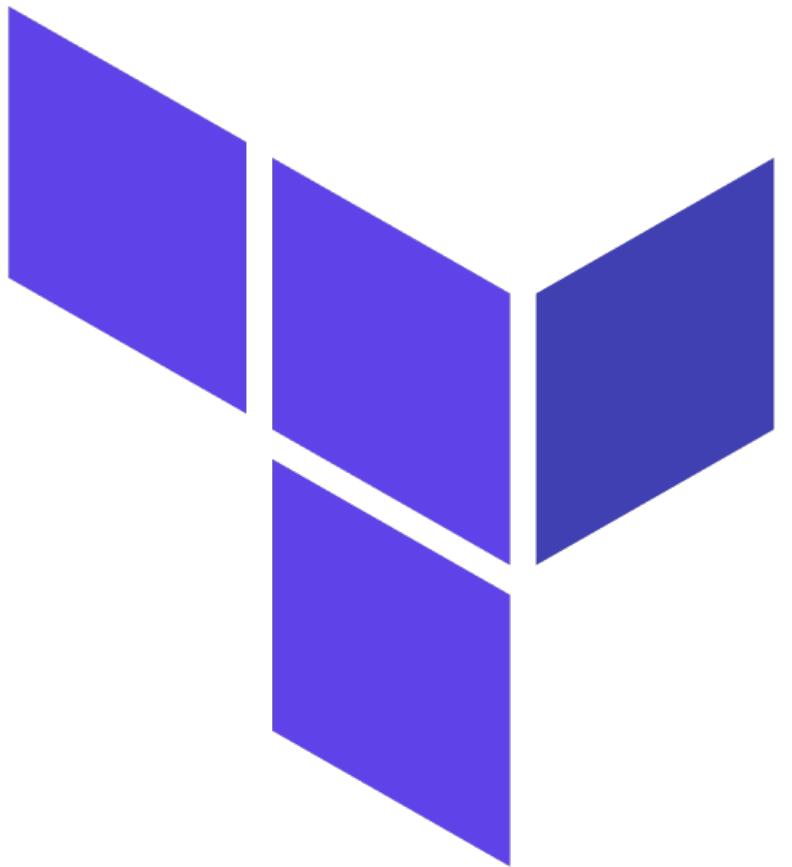
Example for Remote Backend
Performing Operations : Terraform Cloud, Terraform Enterprise

Standard Backends

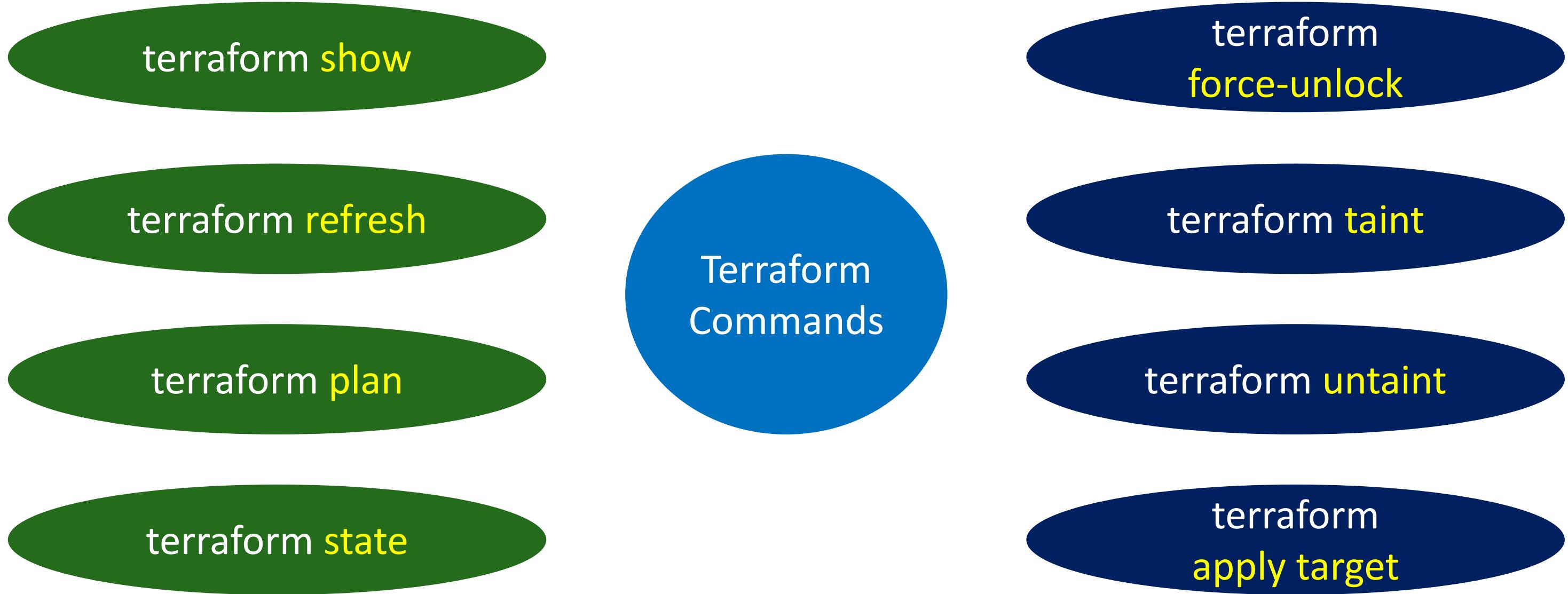
Standard backends **only store state**, and **rely** on the local backend for performing operations.

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

Terraform State Commands



Terraform Commands – State Perspective





Amazon Virtual Private Cloud
(Amazon VPC)



Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet

AWS

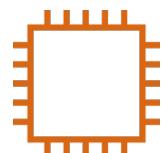
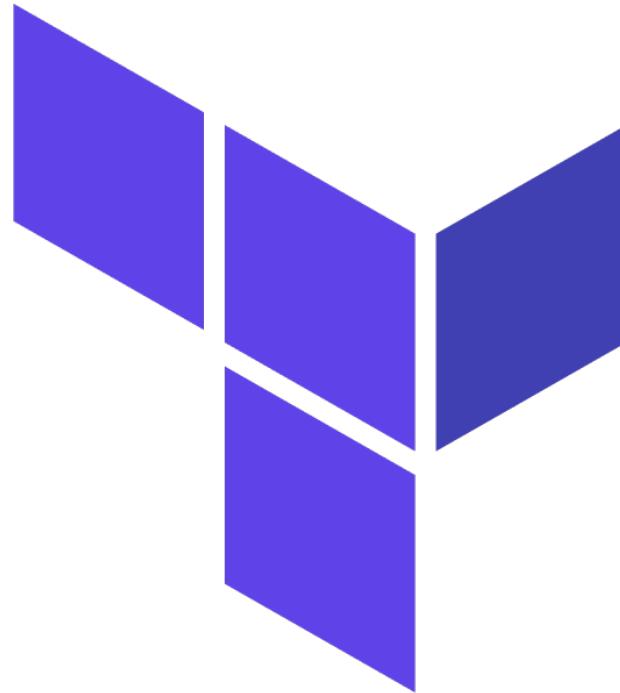
VPC with Web, App and DB Tiers

EC2 Instances, Bastion Host and Security Groups

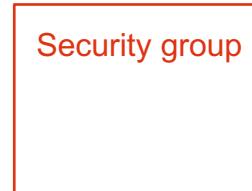
Application Load Balancer

Autoscaling with Launch Templates

Terraform Remote State Datasource



EC2 Instance



Security Groups



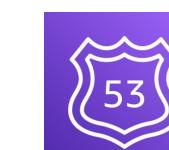
Classic Load Balancer



Application Load Balancer



AWS Certificate Manager (ACM)



Amazon Route 53



Amazon RDS DB



Autoscaling



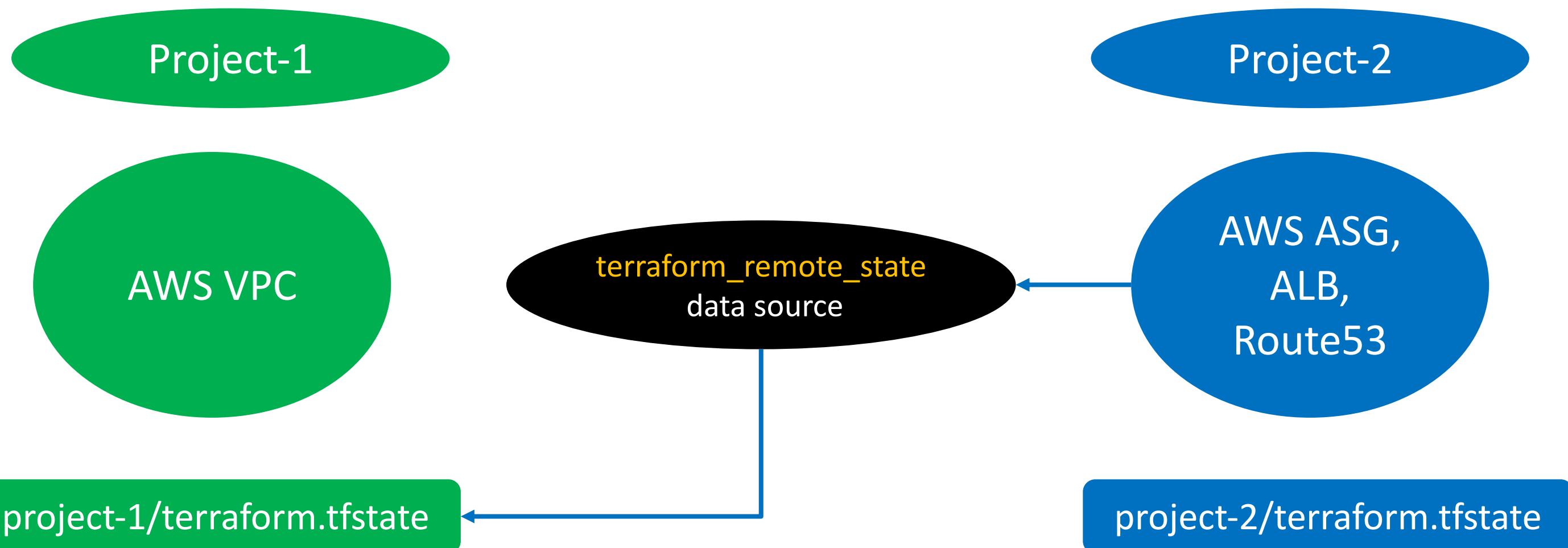
AWS IAM



Amazon SNS

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.



Terraform Remote State Datasource



<http://tfremoteds.devopsincloud.com>

Project-1

VPC Resources

Project-2

Autoscaling Groups

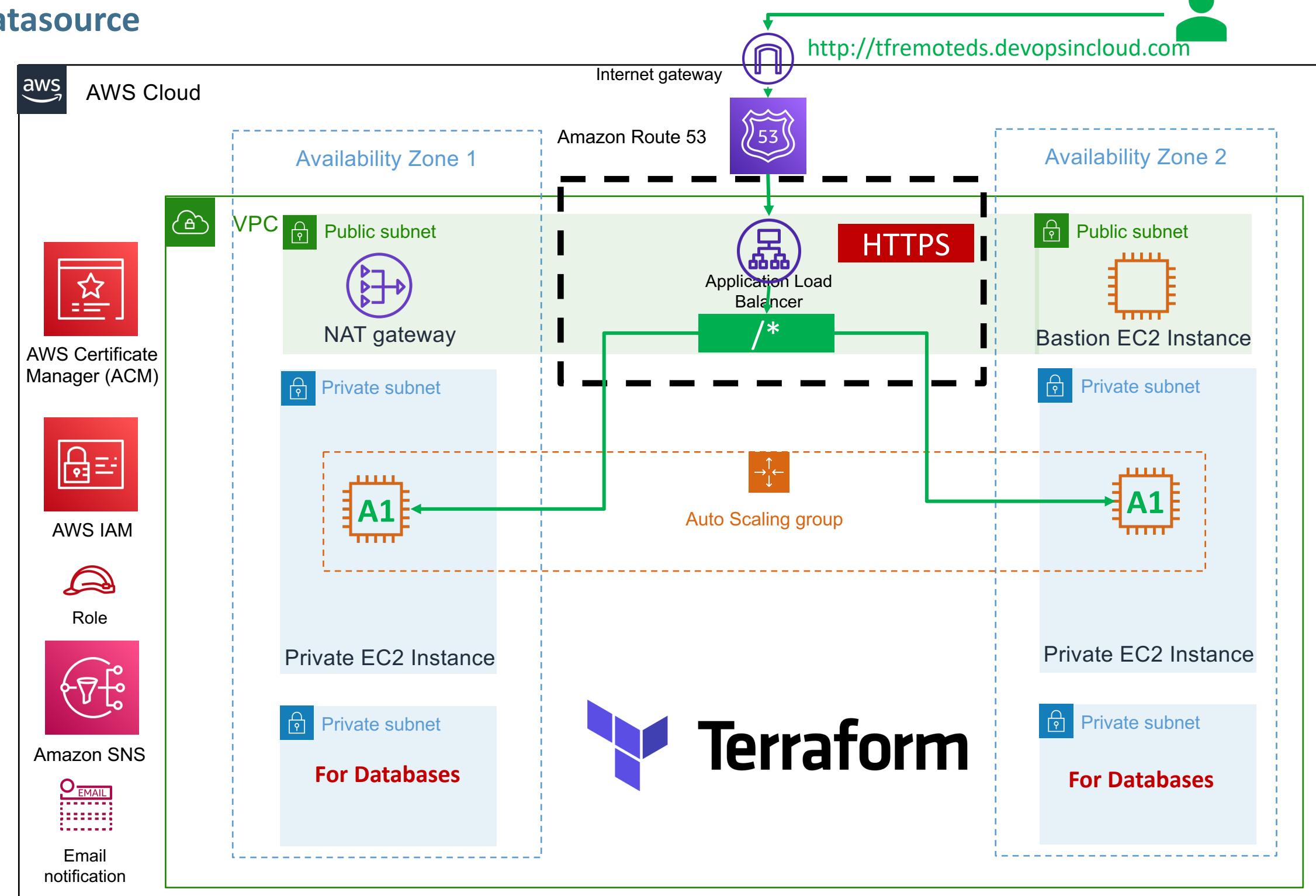
Application Load Balancers

AWS Route53

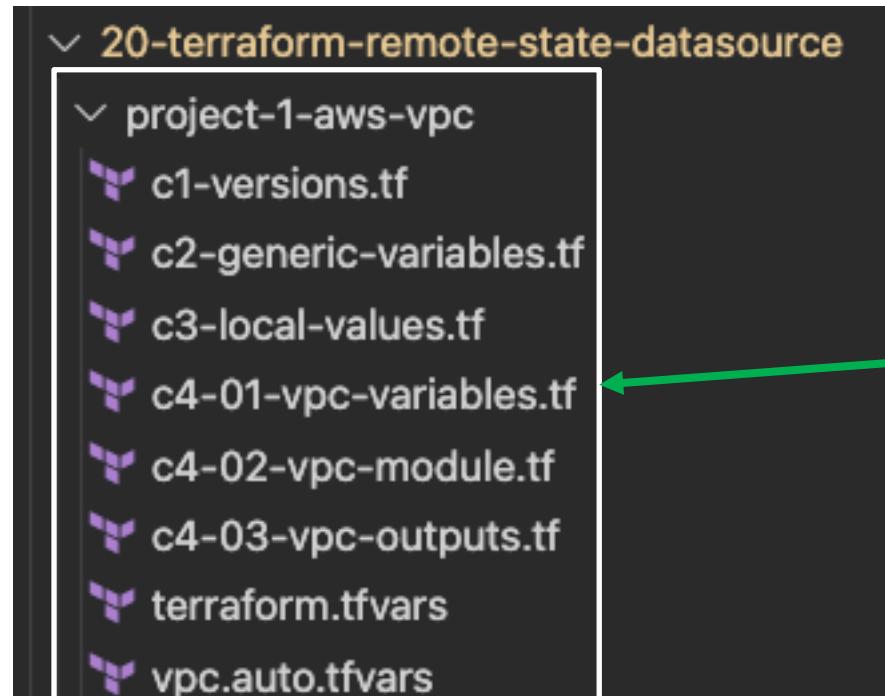
AWS Certificate Manager

AWS SNS

AWS IAM



What are we going to learn ?



Project-1
Create VPC Resources with
Terraform Backend as S3

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"
}

}
```

project-2-app1-with-asg-and-alb

- > local-exec-output-files
- > private-key
- app1-install.sh
- c0-terraform-remote-state-datasource.tf
- c1-versions.tf
- c2-generic-variables.tf
- c3-local-values.tf
- c5-01-securitygroup-variables.tf
- c5-02-securitygroup-outputs.tf
- c5-03-securitygroup-bastionssg.tf
- c5-04-securitygroup-privesg.tf
- c5-05-securitygroup-loadbalancerssg.tf
- c6-01-datasource-ami.tf
- c6-02-datasource-route53-zone.tf
- c7-01-ec2instance-variables.tf
- c7-02-ec2instance-outputs.tf
- c7-03-ec2instance-bastion.tf
- c8-elasticip.tf
- c9-nullresource-provisioners.tf
- c10-01-ALB-application-loadbalancer-variables.tf
- c10-02-ALB-application-loadbalancer.tf
- c10-03-ALB-application-loadbalancer-outputs.tf
- c11-acm-certificatemanager.tf
- c12-route53-dnsregistration.tf
- c13-01-autoscaling-with-launchtemplate-variables.tf
- c13-02-autoscaling-launchtemplate-resource.tf
- c13-03-autoscaling-resource.tf
- c13-04-autoscaling-with-launchtemplate-outputs.tf
- c13-05-autoscaling-notifications.tf
- c13-06-autoscaling-ttsp.tf
- c13-07-autoscaling-scheduled-actions.tf

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.

- ec2instance.auto.tfvars
- terraform.tfvars

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"
    }
}
```

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"
}

}
```



Amazon Virtual Private Cloud
(Amazon VPC)



Internet gateway



NAT gateway



Elastic IP
address



Route table



Public Subnet



Private Subnet

AWS

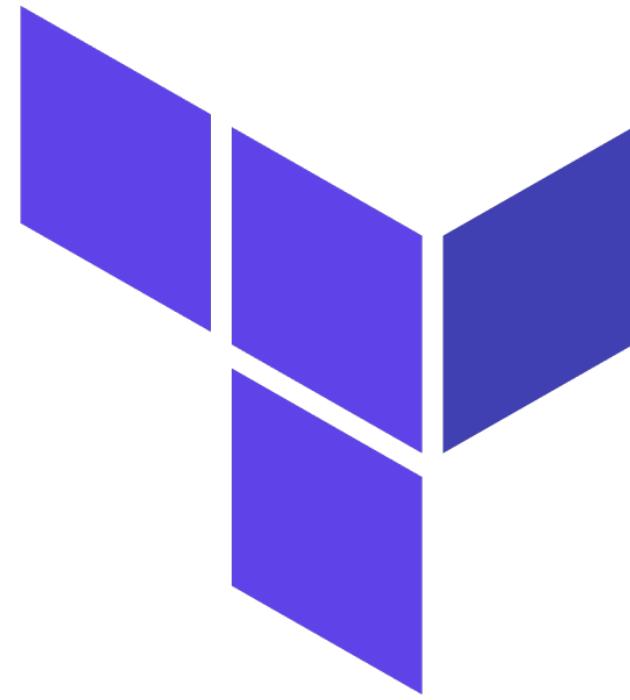
VPC with Web, App and DB Tiers

EC2 Instances, Bastion Host and Security Groups

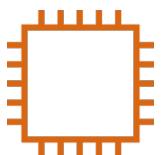
Application Load Balancer

Autoscaling with Launch Templates

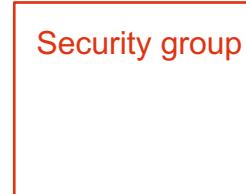
IaC DevOps with AWS CodePipeline



Amazon Simple Storage
Service (Amazon S3)



EC2 Instance



Security Groups



Classic Load
Balancer



Application Load
Balancer



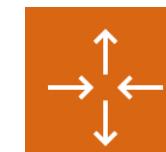
AWS Certificate
Manager (ACM)



Amazon Route 53



Amazon RDS DB



Autoscaling



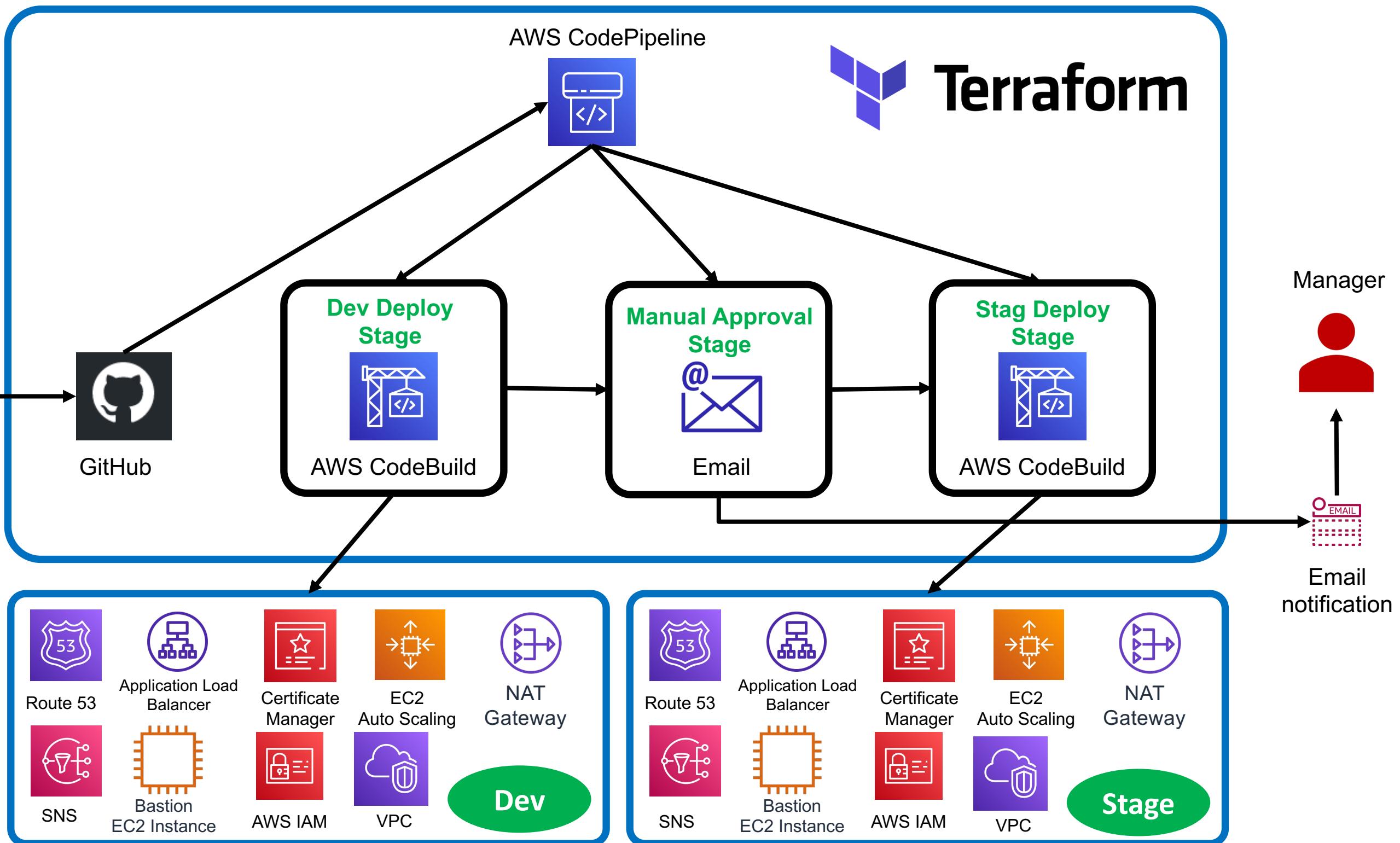
AWS IAM



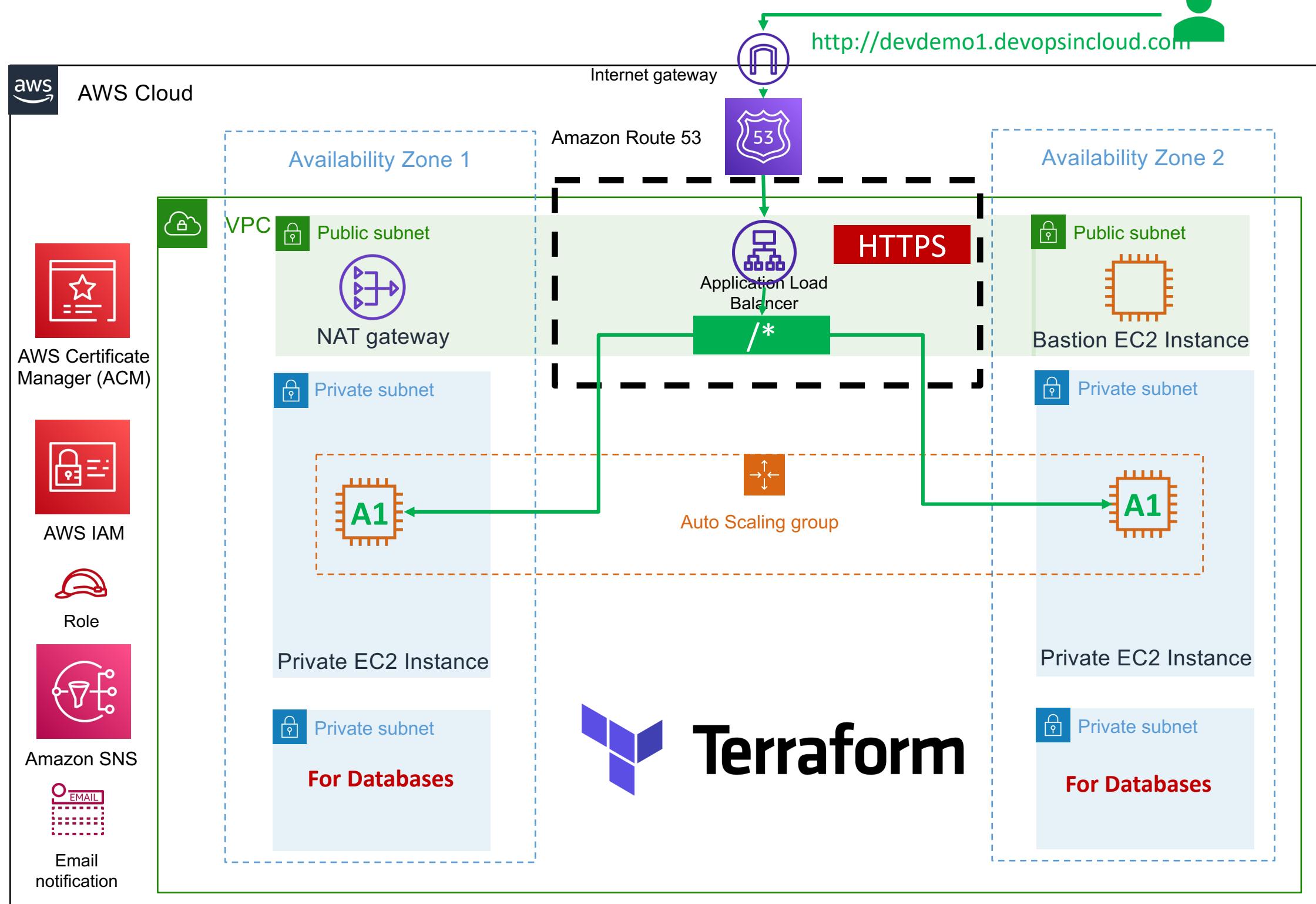
Amazon SNS



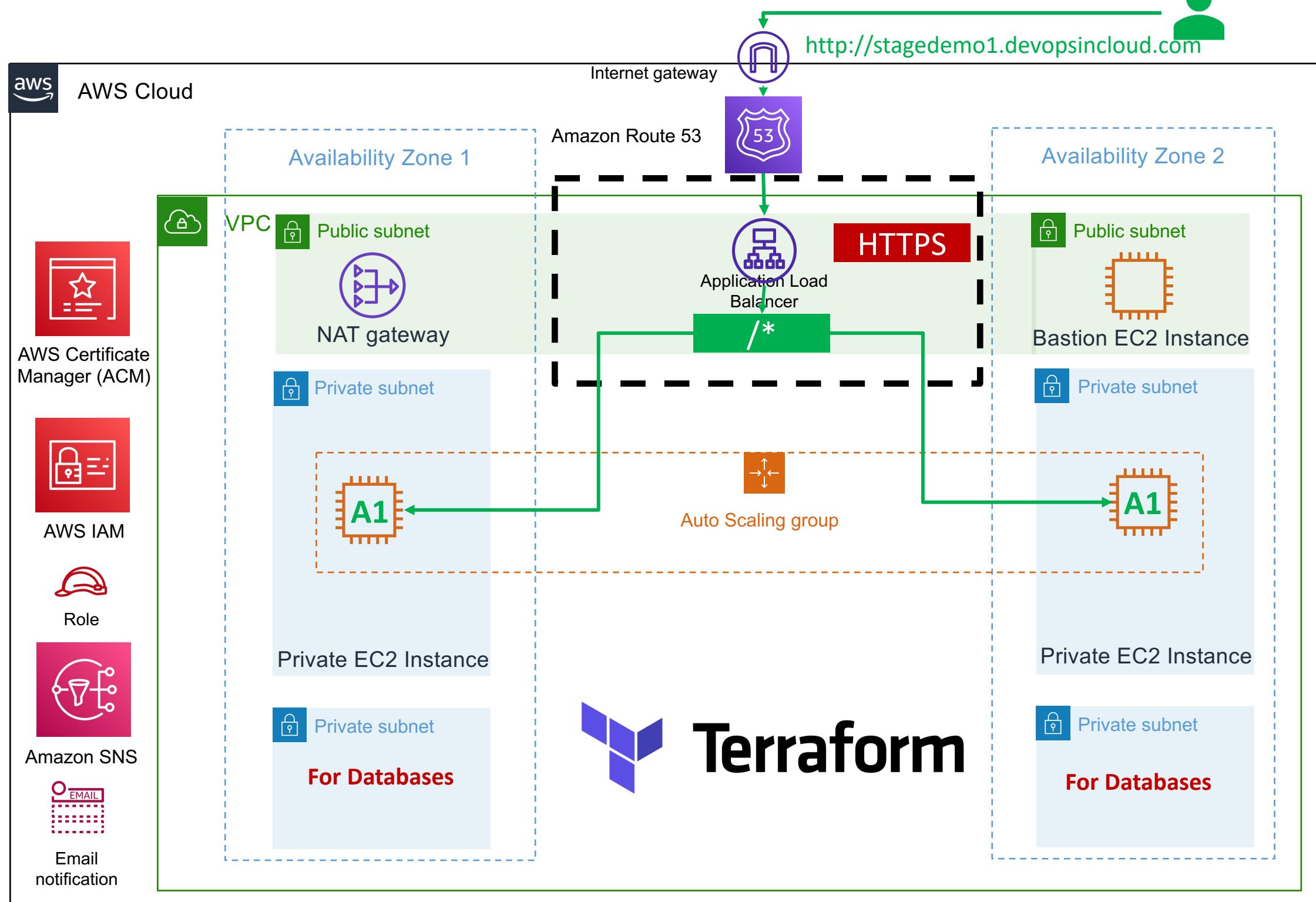
Amazon
DynamoDB



Dev Environment



Staging Environment



```
✓ Git-Repo-Files
  ✓ terraform-manifests
    > private-key
    └ app1-install.sh
    └ c1-versions.tf
    └ c2-generic-variables.tf
    └ c3-local-values.tf
    └ c4-01-vpc-variables.tf
    └ c4-02-vpc-module.tf
    └ c4-03-vpc-outputs.tf
    └ c5-01-securitygroup-variables.tf
    └ c5-02-securitygroup-outputs.tf
    └ c5-03-securitygroup-bastionsg.tf
    └ c5-04-securitygroup-privatesg.tf
    └ c5-05-securitygroup-loadbalancersg.tf
    └ c6-01-datasource-ami.tf
    └ c6-02-datasource-route53-zone.tf
    └ c7-01-ec2instance-variables.tf
    └ c7-02-ec2instance-outputs.tf
    └ c7-03-ec2instance-bastion.tf
    └ c8-elasticip.tf
    └ c9-nullresource-provisioners.tf
    └ c10-01-ALB-application-loadbalancer-variables.tf
    └ c10-02-ALB-application-loadbalancer.tf
    └ c10-03-ALB-application-loadbalancer-outputs.tf
    └ c11-acm-certificatemanager.tf
    └ c12-route53-dnsregistration.tf
    └ c13-01-autoscaling-with-launchtemplate-variables.tf
    └ c13-02-autoscaling-launchtemplate-resource.tf
    └ c13-03-autoscaling-resource.tf
    └ c13-04-autoscaling-with-launchtemplate-outputs.tf
    └ c13-05-autoscaling-notifications.tf
    └ c13-06-autoscaling-ttsp.tf
```

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

```
✓ 22-laC-DevOps-using-AWS-CodePipeline
  ✓ Git-Repo-Files
    > terraform-manifests
    └ .gitignore
    ! buildspec-dev.yml
    ! buildspec-stag.yml
    └ README.md
```

Thank You