aws

HashiCorp

Terraform

- Terraform + AWS
- Learn while Doing it
- Programming Concepts
- Projects + Task
- Terraform Cloud
- Terraform Module
- Workspaces
- State Mangement

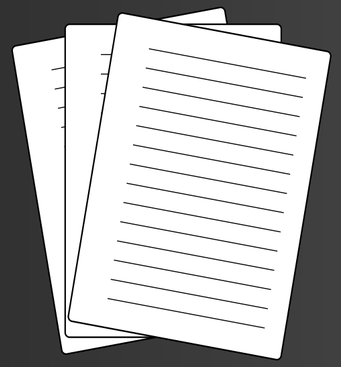
Learn AWS Concepts

- EC2
- VPC
- S3
- IAM

What is Terraform?

Terraform is an open-source infrastructure as code (IaC) tool.

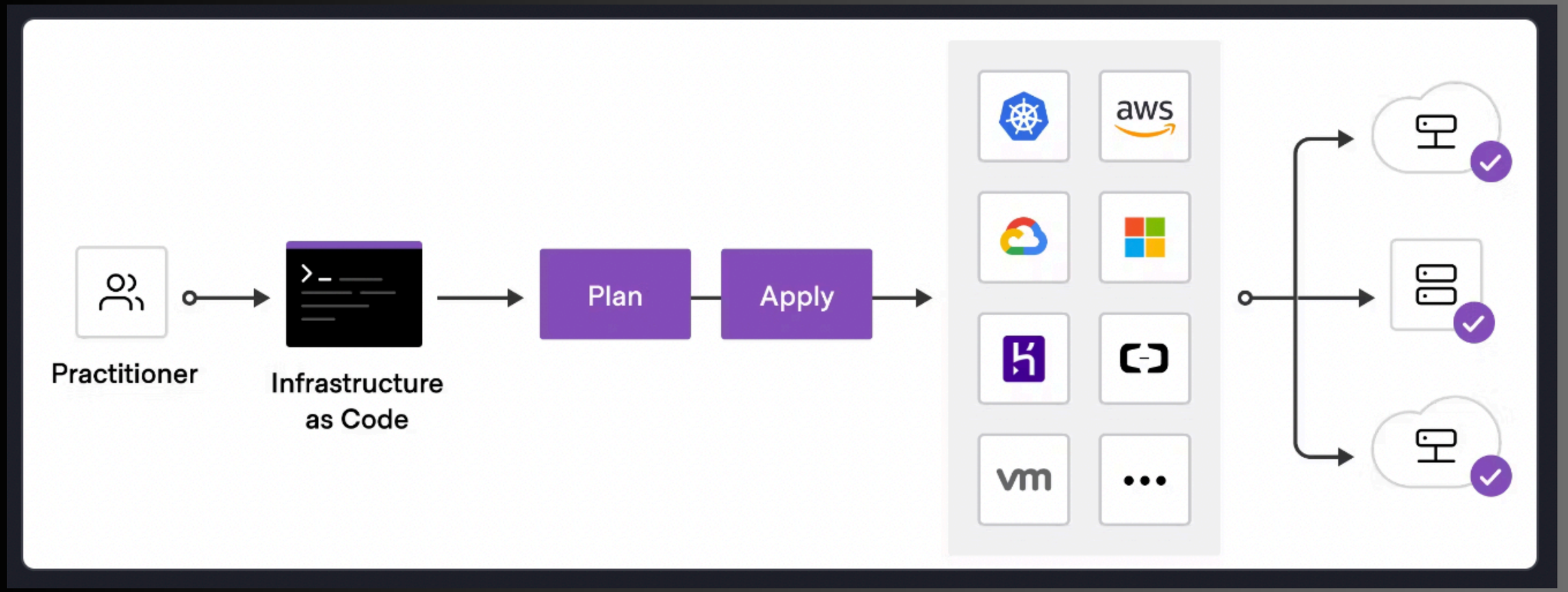
What is IaC?



Tools allow you to manage infrastructure with configuration files rather than through a graphical user interface.

What is IaC?

IaC allows you to build, change, and manage your infrastructure in a safe, consistent, and repeatable way by defining resource configurations that you can version, reuse, and share.

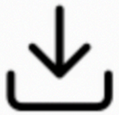

Practitioner
Infrastructure as Code
Plan
Apply
aws
vm

HashiCorp

Terraform

Scope: Identify the infrastructure for your project

Author: Write configuration to define your infrastructure



Initialize: Install the required Terraform providers

Plan: Preview the changes Terraform will make

Apply: Make the changes to your infrastructure

HashiCorp
Terraform

Multiple Cloud Platforms

Configuration for Humans

Track Resources with State

Collaborate with Terraform Cloud

Why Terraform?

- **Automate Setup**: Quickly create and configure resources like servers, databases, and networks.
- **Consistency**: Ensure every environment is set up the same way, reducing errors.
- **Scalability:** Easily replicate and scale infrastructure for different environments.
- **Version Control**: Track and manage infrastructure changes just like software code.
- **Flexibility**: Works with multiple cloud providers and on-premises setups.

EC2

Manual

Terraform

Terraform Config

- It uses .tf extension
- Format is HCL (Hashicorp Config Language)
- Declarative Language
- State Management

Terraform supports JSON format also

- HCL:

```hcl
provider "aws" {
  region = "us-east-1"
}
```

- JSON:

```json
{
  "provider": {
    "aws": {
      "region": "us-east-1"
    }
  }
}
```

State Management

The state file (terraform.tfstate) maintains a detailed record of the current state of managed resources

This state file can be stored locally or remotely, with remote storage options enabling collaboration by sharing the state across teams and environments.

Variables

```
variable "region" {
  description = "The AWS region to create resources in"
  default     = "us-east-1"
}
```

Outputs

```
output "instance_public_ip" {
  value = aws_instance.example.public_ip
}
```

S3

AWS S3 (Amazon Simple Storage Service) is a scalable, high-speed, web-based cloud storage service designed for online backup and archiving of data and applications.

Exercise:

- Create a S3 bucket using TF config
- Upload a File
- Output the bucket name

Terraform Remote State Management

- Create S3 bucket
- Backend block for remote state management

```
terraform {
  backend "s3" {
    bucket = "my-terraform-state"
    key    = "path/to/my/terraform.tfstate"
    region = "us-west-2"
  }
}
```

Project:

Deploy Static website on AWS using S3

- **Provider Configuration**: Specifies AWS and random providers.
- **Bucket Creation:** Creates an S3 bucket with a unique name.
- **Public Access:** Configures public access to the bucket.
- **Website Configuration:** Sets up the bucket for static website hosting.
- **File Uploads:** Uploads the index.html and error.html files to the bucket.
- **Website Endpoint:** Outputs the URL of the static website.

For the purposes of this module you'll use the Amazon S3 website endpoint URL that we supply. It takes the form

`http://{your-bucket-name}.s3-website.{region}.amazonaws.com`.

Summary

- resource "aws_s3_bucket" "mywebapp-bucket"
- resource "aws_s3_bucket_public_access_block" "example"
- resource "aws_s3_bucket_policy" "mywebapp"
- resource "aws_s3_bucket_website_configuration" "mywebapp"
- resource "aws_s3_object" "index_html"
- resource "aws_s3_object" "styles_css"
- output "name"

Virtual Private Cloud (VPC)

Virtual Private Cloud (VPC)

A private, isolated network within the AWS cloud where you can launch and manage your resources securely.

Website is ready
Where to deploy?

Website is ready
Where to deploy?

Website is ready
Where to deploy?

US

Asia

Europe

Website is ready
Where to deploy?

Asia

North

East

South

REGION

Website is ready
Where to deploy?

Asia

- North — Singapore
- East — Mumbai, Hyderabad
- South — Tokyo

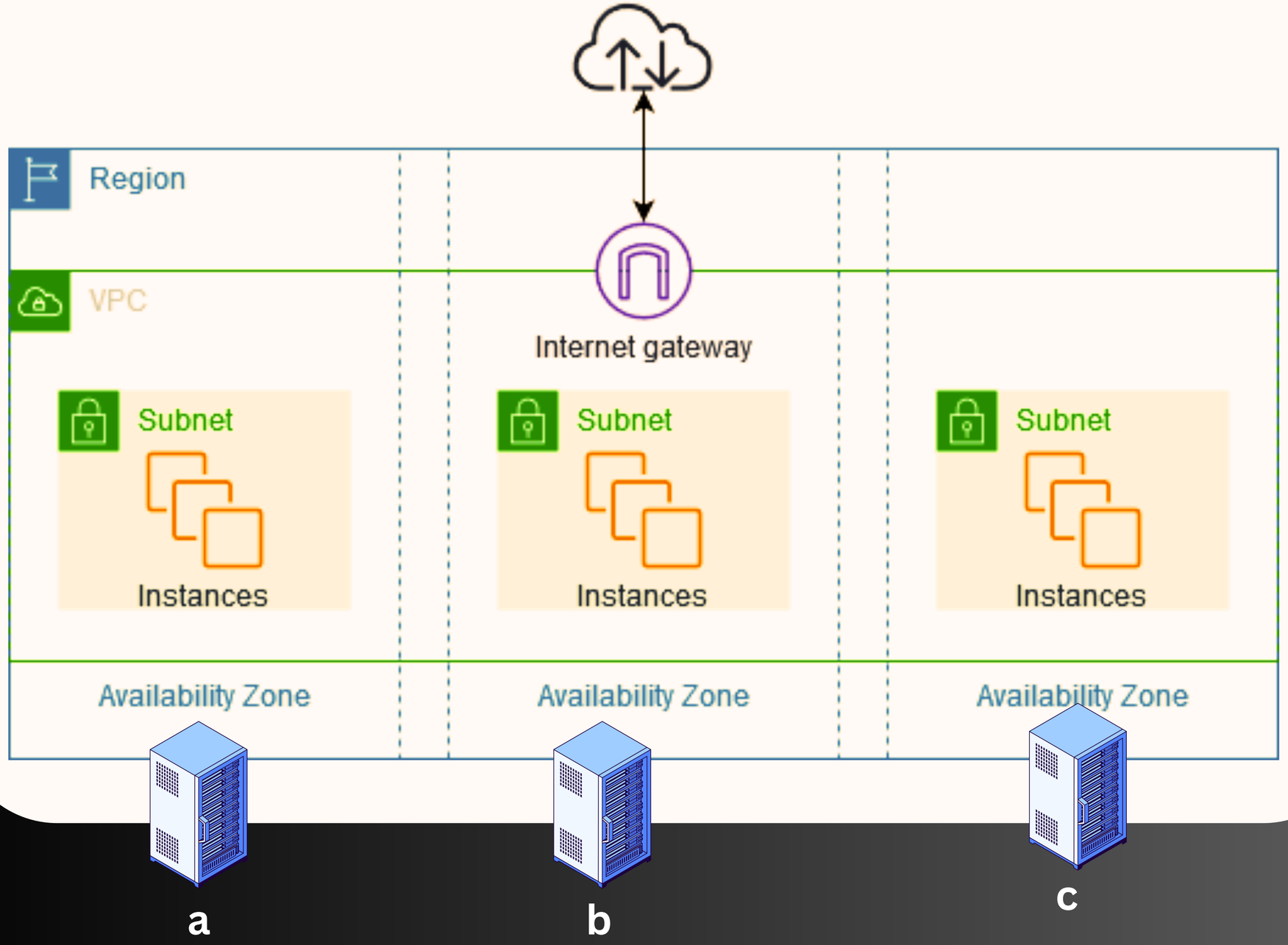
Availability Zones

a b c

Mumbai

What is Subnets?

A subnet is a smaller, segmented part of a larger network that isolates and organizes devices within a specific IP address range.


Region
VPC
Internet gateway
Subnet
Instances
Subnet
Instances
Subnet
Instances
Availability Zone
Availability Zone
Availability Zone
a
b
c

VPC CIDR Block

When you create a VPC, you specify a CIDR block that defines the IP address range for the entire VPC. For example:

sh Copy code

```
10.0.0.0/16
```

This block allows for 65,536 IP addresses (but in reality, 65,531 usable addresses).

CIDR (Classless Inter-Domain Routing) is a method for allocating IP addresses and routing Internet Protocol (IP) packets.

What happens when creating subnet?

CIDR Block Allocation:
You specify a range of IP addresses (CIDR block) within the VPC's IP address range for the subnet.

This determines the pool of IP addresses available for instances in the subnet.

Subnet CIDR Blocks

Within the VPC, you can create subnets by allocating smaller CIDR blocks from the VPC's range. For example:

- Public Subnet: `**`10.0.1.0/24`**`
- Private Subnet: `**`10.0.2.0/24`**`

Each of these subnets has 256 IP addresses (251 usable).

Explanation of 10.0.1.0/24

- An IPv4 address is 32 bits long.
- Example in binary: `10.0.1.0` -> `00001010.00000000.00000001.00000000`

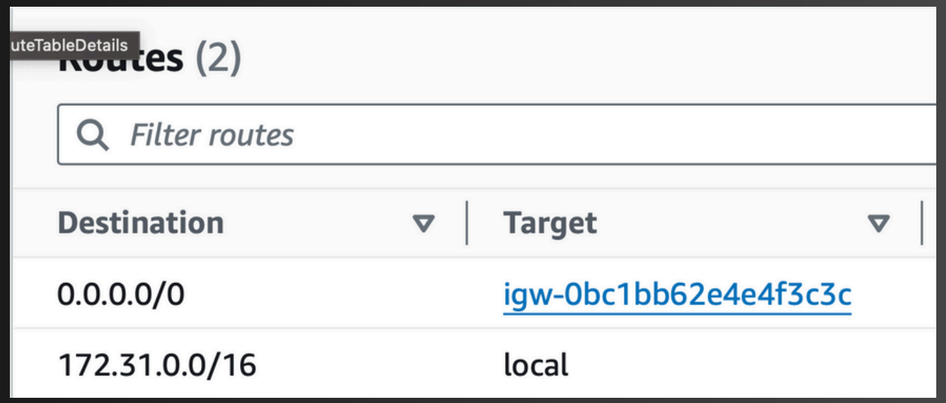
- **The /24 indicates that the first 24 bits are the network portion of the address.**
- **The remaining 8 bits are available for host addresses within the network.**

10.0.1.0 to 10.0.1.255 is the full range.

Route Table

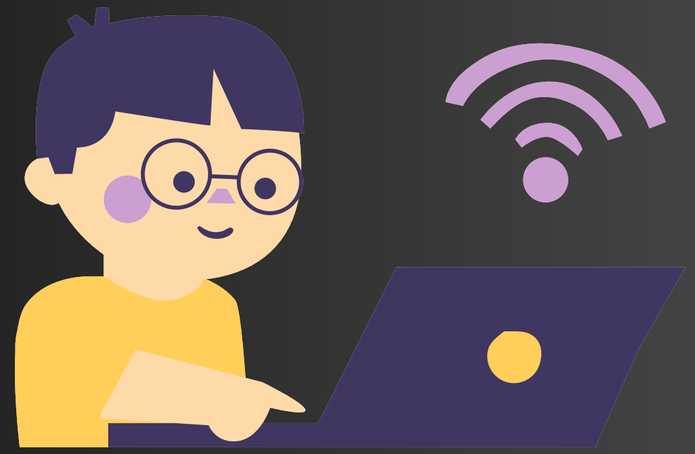
What is a Route Table?

A route table is a set of rules, called routes, that are used to determine where network traffic from your subnets or gateway is directed. Each subnet in your VPC must be associated with a route table, which controls the routing for that subnet.



Internet Gateway

An Internet Gateway is a component that allows communication between instances in your VPC and the internet.



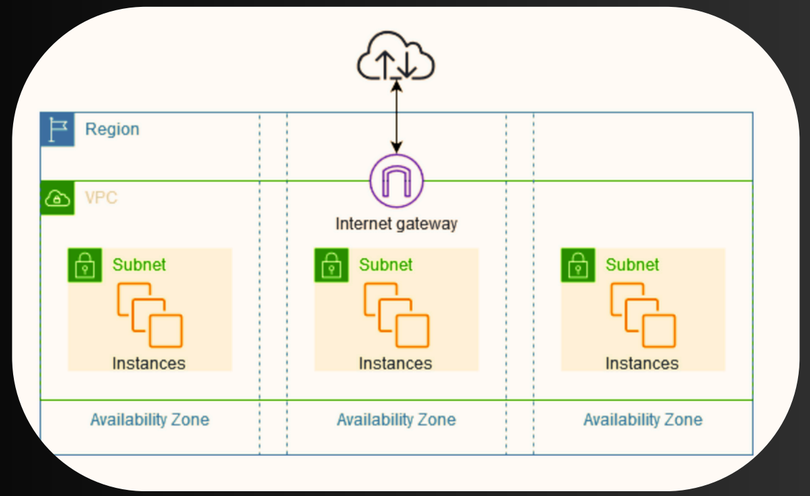
Security Groups: Network firewall rules that control inbound and outbound traffic for instances.

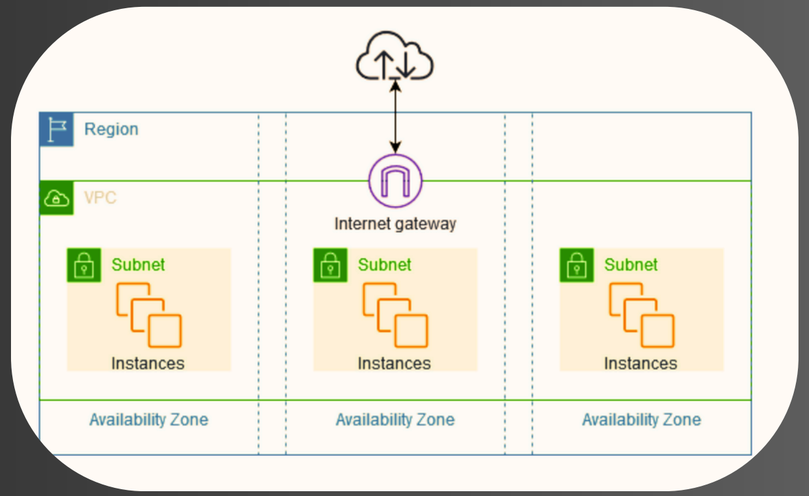
Network ACLs (Access Control Lists): Optional layer of security for your VPC that acts as a firewall for controlling traffic in and out of one or more subnets.

NAT (Network Address Translation) Gateway:

Enables instances in a private subnet to connect to the internet or other AWS services, but prevents the internet from initiating connections to those instances.

VPC Peering: A networking connection between two VPCs that enables you to route traffic between them privately.





VPC Endpoints: Allows you to privately connect your VPC to supported AWS services and VPC endpoint services powered by AWS PrivateLink.

Bastion Host: A special-purpose instance that provides secure access to your instances in private subnets.

Elastic IP Addresses: Static IP addresses designed for dynamic cloud computing.

VPC Flow Logs: Capture information about the IP traffic going to and from network interfaces in your VPC.

Direct Connect: Establishes a dedicated network connection from your premises to AWS.

Transit Gateway: A network transit hub that you can use to interconnect your VPCs and on-premises networks.

- Create VPC.
- Create Public Subnet.
- Create Private Subnet.
- Create Internet Gateway.
- Attach Internet Gateway to VPC.
- Create Route Table for Public Subnet.
- Add Route to Internet Gateway in Public Route Table.
- Associate Public Subnet with Public Route Table.
- Create Route Table for Private Subnet (if using NAT, otherwise optional).
- Associate Private Subnet with Private Route Table.

Exercise on VPC:

- A VPC with a CIDR block of 10.0.0.0/16.
- One public subnet with a CIDR block of 10.0.1.0/24.
- One private subnet with a CIDR block of 10.0.2.0/24.
- One Internet Gateway.
- One public route table with a route to the Internet Gateway, and the correct association between the public subnet and the public route table.

Project: VPC + EC2 + NGINX + HTTP Access:

- A VPC with pubic and private subnet
- A EC2 instance using public subnet we created.
- Setup nginx webserver
- Create security group rule to enable HTTP access
- Output the webserver URL on terminal

Data Source in Terraform?

It allows you to fetch and use information from

- external sources or
- existing resources within your cloud infrastructure.

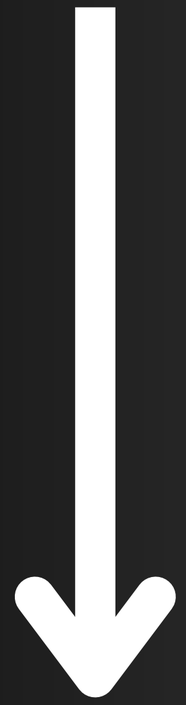
Useful for obtaining dynamic data that you need for your configurations.

Data Source TASK?

- Create an EC2 instance using existing
  - VPC
  - private-subnet
  - security-group

Terraform Variables

Terraform Variables



environment var

terraform.tfvars

*.auto.tfvars

-var & -var-file

export TF_VAR_key=value

Terraform Functions

Terraform Functions

- Built-in functions that you can call from within expressions to transform and combine values.

```
max(5, 12, 9)
```

Terraform Functions

- #value = lower(local.name)
- #value = startswith(local.name, "Hello")
- #value = join("-", var.list)
- #value = split("-", var.string)
- #value = trimspace(var.string)
- #value = length(var.list)
- #value = merge(var.map1, var.map2)
- #value = contains(var.list, "d")
- #value = max(1, 2, 3) and min(1, 2, 3)
- #value = abs(var.number)
- #value = toset(var.list) #to convert list into set (will remove the duplicates)
- #value = tolist(var.set)

Multiple Resources using

Count

for_each

Create 2 subnets
Using count

subnet-1
10.0.0.0/24

subnet-2
10.0.1.0/24

Create 2 subnets
Create 4 ec2 instance, 2 in each subnet

subnet-1

ec2-1

ec2-2

subnet-2

ec2-3

ec2-4

Create 2 subnets
Create 2 ec2 instance, 1 in each subnet

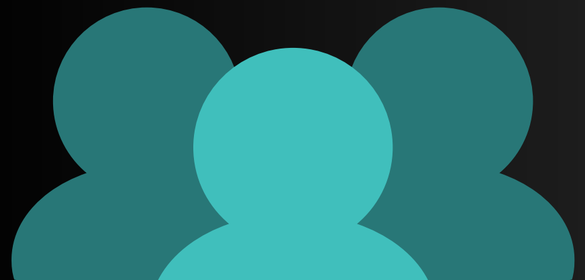
subnet-1 ec2-1 (ubuntu)

subnet-2 ec2-2 (amazon-linux)

Project: IAM

Task: AWS IAM Management

- Provide user and roles info via YAML file
- Read the YAML file and process data
- Create IAM users
- Generate Passwords for the users
- Attach policy/roles to each users



```
[
 {
  roles    = ["AmazonEC2FullAccess"]
  username = "raju"
 },
 {
  roles    = ["AmazonS3ReadOnlyAccess"]
  username = "sham"
 },
 {
  roles    = ["AmazonS3ReadOnlyAccess", "AmazonEC2FullAccess"]
  username = "baburao"
 },
]
```

index 0

index 1

index 2

```
+ output = [
    + [
        + {
            + role     = "AmazonEC2FullAccess"
            + username = "raju"
          },
      ],
    + [
        + {
            + role     = "AmazonS3ReadOnlyAccess"
            + username = "sham"
          },
      ],
    + [
        + {
            + role     = "AmazonS3ReadOnlyAccess"
            + username = "baburao"
          },
        + {
            + role     = "AmazonEC2FullAccess"
            + username = "baburao"
          },
      ],
  ]
```

The flatten function in Terraform is used to transform a list of lists into a single, flat list.

```
[
  { username = "raju", role = "AmazonEC2FullAccess" },
  { username = "sham", role = "AmazonS3ReadOnlyAccess" },
  { username = "baburao", role = "AmazonS3ReadOnlyAccess" },
  { username = "baburao", role = "AmazonEC2FullAccess" }
]
```

```
user_role_pairs = flatten(
  [for user in local.users : [for role in user.roles : {
    username = user.username
    role     = role
    }
    ]
])
```

```
{
  roles    = ["AmazonS3ReadOnlyAccess", "AmazonEC2FullAccess"]
  username = "baburao"
 }
```

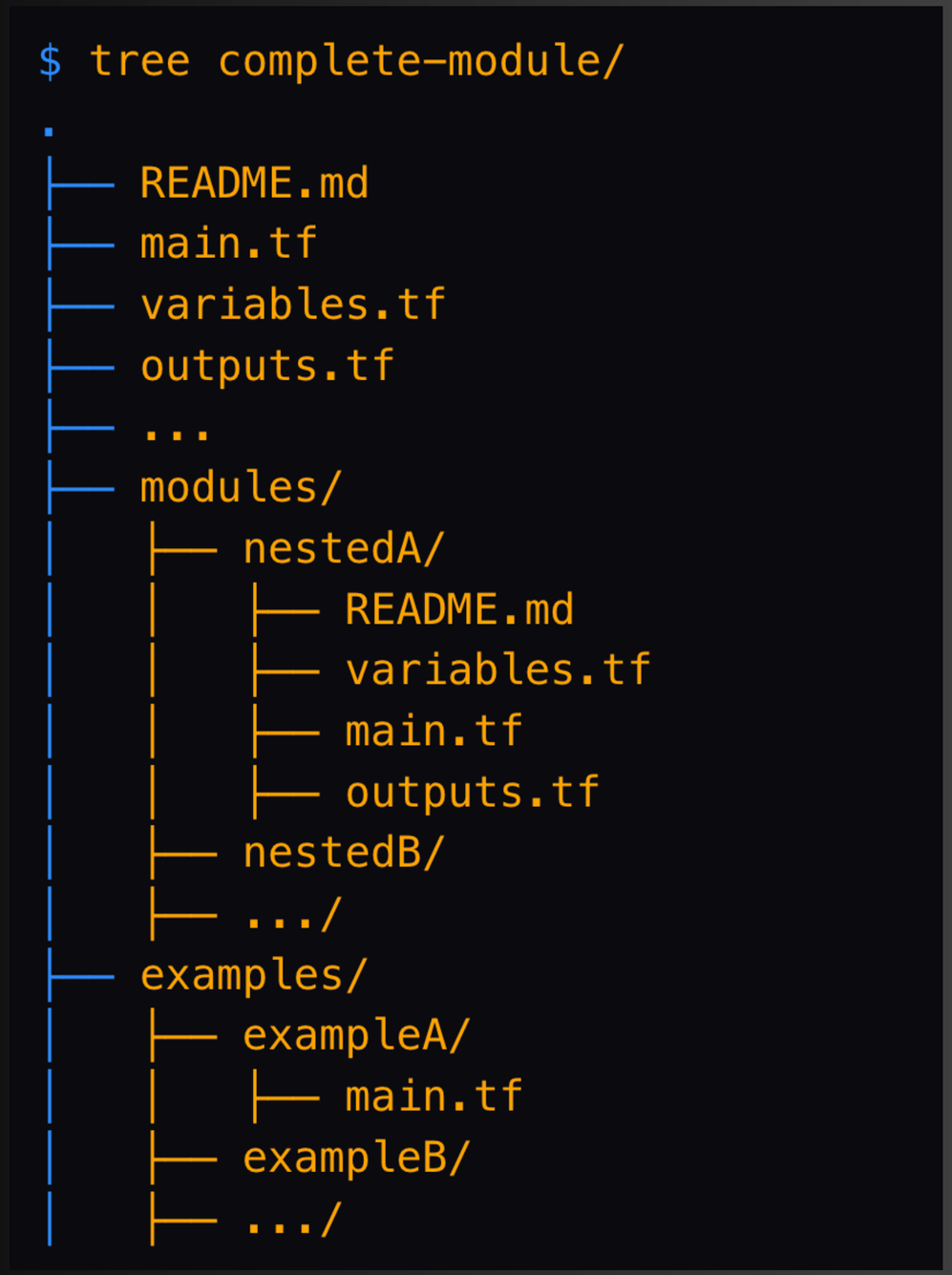
Terraform Modules

Terraform Modules:

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

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

```
$ tree minimal-module/
.
├── README.md
├── main.tf
├── variables.tf
├── outputs.tf
```


$ tree complete-module/
.
├── README.md
├── main.tf
├── variables.tf
├── outputs.tf
├── ...
├── modules/
│   ├── nestedA/
│   │   ├── README.md
│   │   ├── variables.tf
│   │   ├── main.tf
│   │   ├── outputs.tf
│   ├── nestedB/
│   ├── .../
├── examples/
│   ├── exampleA/
│   │   ├── main.tf
│   ├── exampleB/
│   ├── .../

Building our own Module

Requirements

```
$ tree minimal-module/
.
├── README.md
├── main.tf
├── variables.tf
└── outputs.tf
```

- Accept cidr_block from user to create VPC
- User can create multiple subnets
  - Get CIDR block for subnet from user
  - Get AZS (availability zone)
  - User can mark a subnet as public (default is private)
    - if public, create IGW
    - Associate public subnet with Routing table

Prepare Module For Publish

- README.md file
- LICENSE
- Examples
- Push code in GitHub
- Terraform Registry

Terraform Dependency

Terraform Dependencies

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

resource "aws_instance" "example" {
  ami           = "ami-123456"
  instance_type = "t2.micro"

  depends_on = [
    aws_security_group.example
  ]
}
```

Resource Lifecycle

Terraform Lifecycle Block

```
resource "aws_instance" "example" {
  ami           = "ami-123456"
  instance_type = "t2.micro"

  lifecycle {
    create_before_destroy = true
  }
}
```

prevent_destroy

ignore_changes

replace_triggered_by

```
resource "aws_iam_user_login_profile" "profile" {
  for_each        = aws_iam_user.users
  user            = each.value.name
  password_length = 12

  lifecycle {
    ignore_changes = [
      password_length,
      password_reset_required,
      pgp_key,
    ]
  }
}
```

ignore_changes

```
resource "aws_instance" "example" {
  ami           = "ami-123456"
  instance_type = "t2.micro"

  lifecycle {
    replace_triggered_by = [
      "aws_security_group.example.id",
      "aws_key_pair.example.key_name"
    ]
  }
}
```

replace_triggered_by

Validations

Terraform Validations

preconditions

postconditions

Allow you to define checks that must be true before a resource is created (precondition) and after a resource is created (postcondition).

preconditions

```
lifecycle {
  precondition {
    condition     = var.bucket_name != ""
    error_message = "The bucket name must not be empty."
  }
}
```

postconditions

```
lifecycle {
  postcondition {
    condition     = aws_s3_bucket.example.arn != ""
    error_message = "The bucket ARN must not be empty after creation."
  }
}
```

Conditions Task

- Create EC2 instance

- Implement preconditions:
  - Inside the resource block, add a lifecycle block.
  - Add precondition blocks to ensure that the security_group id is created

- Implement postcondition:
  - Add another lifecycle block within the resource.
  - Add a postcondition block to ensure that the instance has a public IP address after creation.

```
resource "aws_instance" "example" {
  ami           = var.ami_id
  instance_type = var.instance_type
  security_groups = [aws_security_group.example.name]

  lifecycle {
    precondition {
      condition     = aws_security_group.example.id != ""
      error_message = "The security group must be created before the instance."
    }

    postcondition {
      condition     = self.public_ip != ""
      error_message = "The instance must have a public IP address after creation."
    }
  }
}
```

assert

```
check "ec2_instance_validation" {
 description = "Ensure EC2 instance is using an approved AMI and instance type."

 assert {
  condition     = var.ami_id != ""
  error_message = "AMI ID must not be empty."
 }

 assert {
  condition     = contains(var.production_instance_type, var.instance_type)
  error_message = "Instance type must be one of the approved types for production:
${join(", ", var.production_instance_type)}."
 }
}
```

State Manipulation

- List all resources in the state:
  - terraform state list
- Show details of a specific resource:
  - terraform state show <resource_address>
- Move a resource to a different address:
  - terraform state mv <source_address> <destination_address>
- Remove a resource from the state:
  - terraform state rm <resource_address>
- Pull the current state:
  - terraform state pull
- Push a local state file to the remote backend:
  - terraform state push <state_file>
- List all state commands:
  - terraform state

Terraform Import

terraform import is a command in Terraform that allows you to import existing infrastructure resources into your Terraform state.

Use-Case

- Assuming you have already created an EC2 instance.
- Create a resource block in tf config (initially you can keep it empty)
- Use terraform import command
  - terraform import aws_instance.main ec2_id
- Terraform show to inspect the imported resource.
- Update the resource block accordingly.
  -

Workspaces

Allows you to manage multiple sets of infrastructure configurations within a single configuration directory.

Each workspace has its own state file

tfstate

tfstate

tfstate

workspace-dev

workspace-test

workspace-prod

tf config

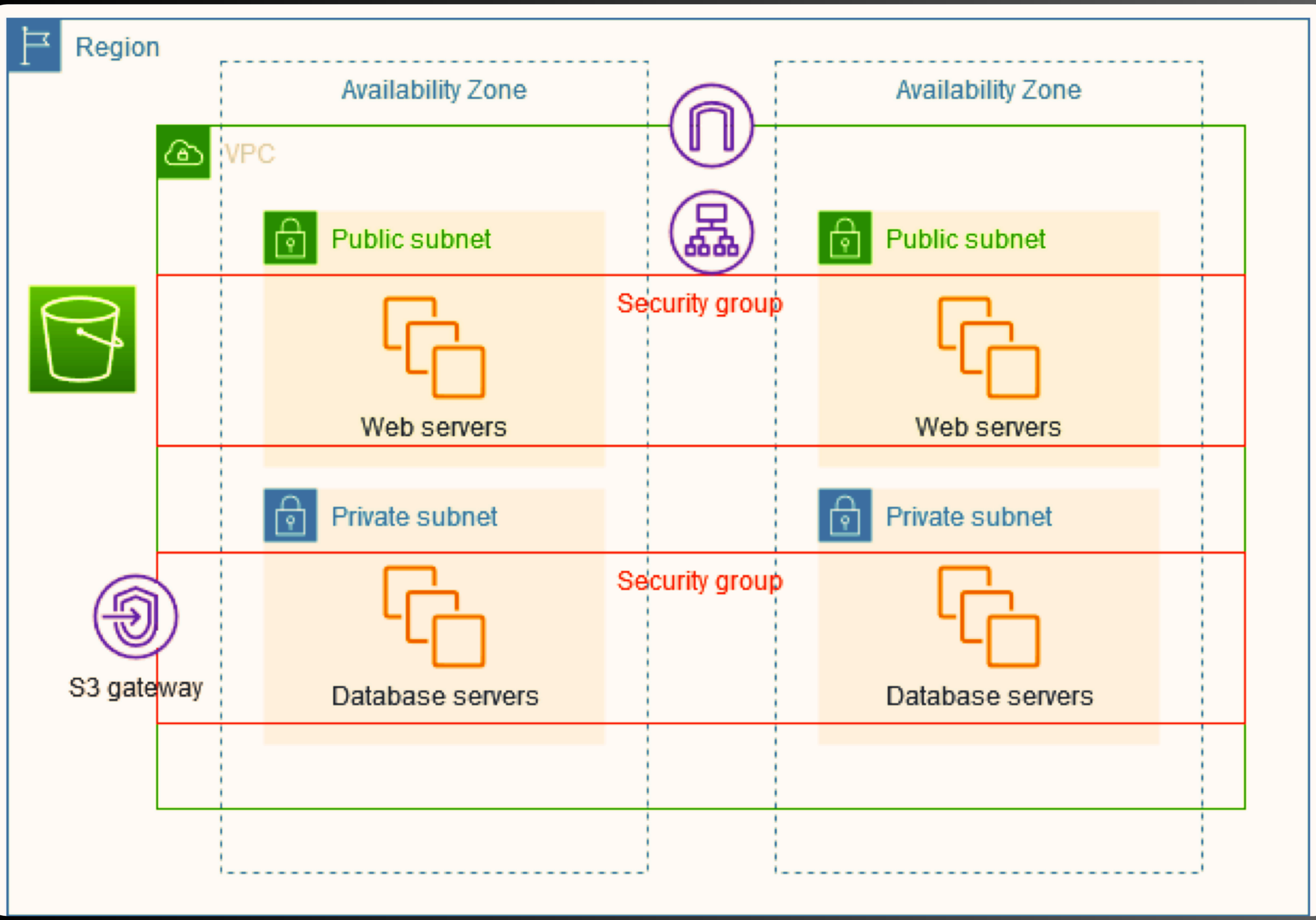
- Listing Workspaces
  - terraform workspace list
- Creating a Workspace
  - terraform workspace new <workspace_name>
- Selecting a Workspace
  - terraform workspace select <workspace_name>
- Showing the Current Workspace
  - terraform workspace show
- Deleting a Workspace
  - terraform workspace select default
  - terraform workspace delete <workspace_name>

Terraform Cloud

Terraform Cloud is a managed service provided by HashiCorp that facilitates collaboration on Terraform configurations.

Providing features like

- remote state management,
- version control system (VCS) integration,
- automated runs, and
- secure variable management.


Region
Availability Zone
Availability Zone
VPC
Public subnet
Public subnet
Security group
Web servers
Web servers
Private subnet
Private subnet
Security group
S3 gateway
Database servers
Database servers