

## CHAPTER 1

### INTRODUCTION

#### 1.1 General Introduction

##### 1.1.1 What Is Terraform?

Terraform is an open source “Infrastructure as Code” tool, created by HashiCorp.

A declarative coding tool, terraform enables developers to use a high-level configuration language called HCL (HashiCorp Configuration Language) to describe the desired “end-state” cloud or on-premises infrastructure for running an application. It then generates a plan for reaching that end-state and executes the plan to provision the infrastructure. Because Terraform uses a simple syntax, can provision infrastructure across multiple cloud and on-premises data centres, and can safely and efficiently re-provision infrastructure in response to configuration changes, it is currently one of the most popular infrastructure automation tools available. If your organization plans to deploy a hybrid cloud or multicloud environment, you’ll likely want or need to get to know Terraform. Terraform is an Infrastructure. As Code (IaC) client tool developed by HashiCorp. It allows the user to define both cloud and on-premise compute resources in human-readable configuration files. These files are created using the HashiCorp Configuration Language (HCL). The syntax is declarative with each block of code defining a resource to be provisioned. Declarative definitions (versus imperative) allow the user to define the desired state, rather than an exhaustive list of all the interim steps required to achieve that state.

##### 1.1.2 Why Infrastructure as Code (IaC)?

To better understand the advantages of Terraform, it helps to first understand the benefits of Infrastructure as Code (IaC). IaC allows developers to codify infrastructure in a way that makes provisioning automated, faster, and repeatable. It’s a key component of Agile and DevOps practices such as version control, continuous integration, and continuous deployment.

Infrastructure as code can help with the following:

- **Improve speed:** Automation is faster than manually navigating an interface when you need to deploy and/or connect resources.
- **Improve reliability:** If your infrastructure is large, it becomes easy to misconfigure a resource or provision services in the wrong order. With IaC, the resources are always provisioned and configured exactly as declared.
- **Prevent configuration drift:** Configuration drift occurs when the configuration that provisioned your environment no longer matches the actual environment. (See 'Immutable infrastructure' below.)
- **Support experimentation, testing, and optimization:** Because Infrastructure as Code makes provisioning new infrastructure so much faster and easier, you can make and test experimental changes without investing lots of time and resources; and if you like the results, you can quickly scale up the new infrastructure for production.

### 1.1.2 Why terraform?

There are a few key reasons developers choose to use terraform over other infrastructure as code tools:

- **Open source:** Terraform is backed by large communities of contributors who build plugins to the platform. Regardless of which cloud provider you use, it's easy to find plugins, extensions, and professional support. This also means terraform evolves quickly, with new benefits and improvements added consistently.
- **Platform agnostic:** Meaning you can use it with any cloud services provider. Most other iac tools are designed to work with single cloud provider.

- **Immutable infrastructure:** Most infrastructure as code tools create mutable infrastructure, meaning the infrastructure can change to accommodate changes such as a middleware upgrade or new storage server. The danger with mutable infrastructure is configuration drift—as the changes pile up, the actual provisioning of different servers or other infrastructure elements ‘drifts’ further from the original configuration, making bugs or performance issues difficult to diagnose and correct. Terraform provisions immutable infrastructure, which means that with each change to the environment, the current configuration is replaced with a new one that accounts for the change, and the infrastructure is reprovisioned. Even better, previous configurations can be retained as versions to enable rollbacks if necessary or desired.

## 1.2 Motivation

Not sure when to use IaC? The simplest answer is whenever you have to manage any type of infrastructure.

However, it becomes more complex with the exact requirements and tools. Some may require strict infrastructure management, while others may require both infrastructure and configuration management. Then comes platform-specific questions like if the tool has the necessary feature set, security implication, integrations, etc. On top of that, the learning curve comes into play as users prefer a simpler and more straightforward tool than a complex one.

## 1.3 Problem Statement:

To develop infrastructure for migrating data form local system to cloud platform using terraform and aws (provider), for 3 tier architecture application.

## 1.4 Report Outline

The project report outline contains following chapters:

- Chapter 1. Introduction.
- Chapter 2. Literature Review
- Chapter 3. System description.
- Chapter 4. Software description
- Chapter 5. Results are discussed.
- Chapter 6 . Business plan and commercial aspects.
- Chapter 7. Conclusion, Applications, Advantages, Disadvantages and Future scope are discussed.

## CHAPTER 2

### LITERATURE REVIEW

Of structural information standards (OASIS) was A founded in 1993 has a non-profit. The infrastructure resources are defined here for the providers and service required. The code is contained in configuration files with a .tf extension. There is additional option you store the configuration Java script. Object notation (JSON) format which then requires the. tf. json extension. Virtual machine in Amazon web service (AWS) three virtual machine, A virtual private cloud (VPC) network, security group and a load balancer service are required for this cluster. Terraform is high level language that can interface to terraform framework via the cloud development kit (CDK). The plan generation additional has a dependency on the existing infrastructure that is represented in the state. The states detail all the infrastructure resources that are currently present. The configuration contains an elastic compute (EC2) virtual machine. Terraform scans configuration files and a generating a corresponding plan the configuration files are written in the HashiCorp configuration language (HCL) the language is Declarative. A Provider defines provider plugin is used to translate the resource block into API calls to infrastructure provider Terraform enterprise is a self-hosted version of Terraform cloud. It offers the same cloud-based feature but set the designed to be employed within the enterprise private cloud. The Organization for the Advancement consortium that works on the development convergence and adoption of open sandstorm cyber security, cloud computing and related areas the two standards that are reverent on this paper are topology and orchestration specification for cloud Application (TOSCA) and cloud Application Management for platform (CAMP).

## CHAPTER 3

### SYSTEM DESCRIPTION

#### 3.1 GENERAL INFORMATION

- **Write**

The write stage focuses on developing the code required to drive the plan. The infrastructure resources are defined here for the providers and services required. Multiple providers may be included. The code is contained in configuration files with a .tf extension. There is an additional option to store the configuration in JavaScript Object Notation (JSON) format which then requires the .tf.json extension. The correct extension must be used in order for the Terraform tool to detect the configuration file while generating the plan. For example, a user wishes to create a mini-cluster of virtual machines in Amazon Web Service (AWS). Three virtual machines, a Virtual Private Cloud (VPC) network, security group and a load balancer service are required for this cluster. The Terraform configuration code declares each

of these resources separately with corresponding parameters. Section III goes into more detail of the configuration language.

- **Plan**

Once the configuration code has been written, the next stage is to run the Terraform tool to generate a plan. The tool is run either through the local Command Line Interface (CLI) or via

Other high-level language that can interface to the Terraform framework via the Cloud Development Kit (CDK). Section VI digs into further detail of the CDK. Running the Terraform plan will scan local directories for configuration files ending in .tf or .tf.json and process these into a list of actions to be sent to the provider(s). This list is called the execution plan and encompasses all create, update and destroy actions needed to make the target infrastructure match what is declared in the configuration code. The plan generation additionally has a dependency on the existing infrastructure that is represented in the state. The state

details all infrastructure resources that are currently present. The state file exists either locally in the file system or remotely. For example, the configuration contains an Elastic Compute (EC2) virtual machine named “VirtualMachine1”. Upon running Terraform to generate the plan, the current state file is checked. If the VirtualMachine1 EC2 is already existing, the plan does not create it. It will either be an update action or no operation.

- **Apply**

The final stage in the Terraform workflow is to apply. Running the tool on a plan executes each action against the corresponding provider. Figure 3.2 shows Terraform interacting with its provider plugin which subsequently calls into the Application Programming Interface (API) of the corresponding cloud provider (e.g. AWS). Provider modules act as the abstraction between the configuration code and unique API defined by each infrastructure provider. The providers are further discussed in Section V. As part of apply, the state is updated to represent the changes in the target infrastructure.

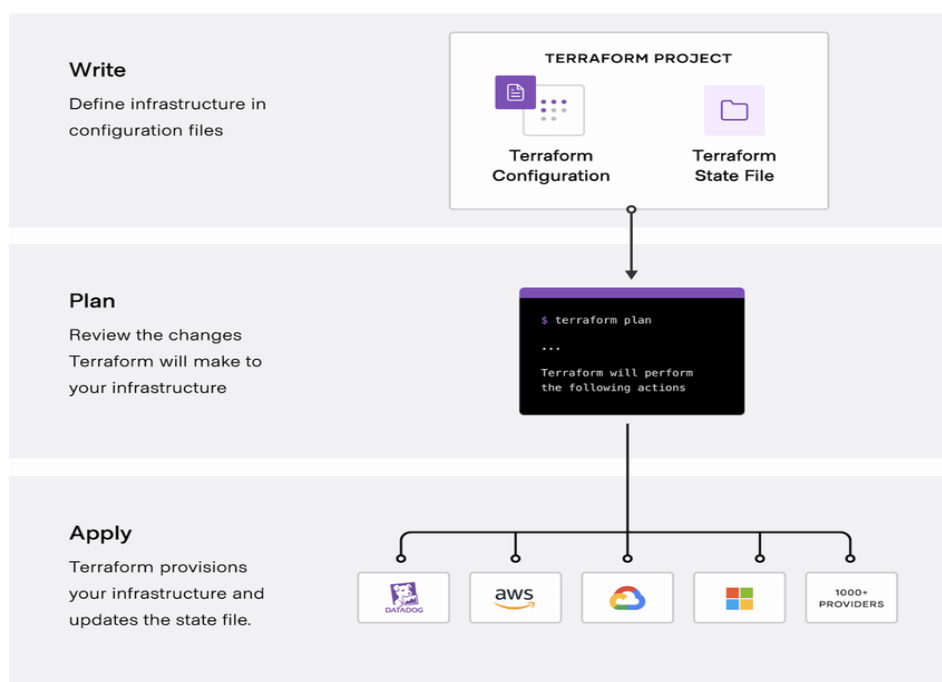


Fig 3.1. High-level workflow for Terraform covering write plan and apply stages

### 3.2 Providers

Terraform relies on plugins called providers to interact and abstract the various infrastructure providers. Each provider must be declared in the configuration using the “provider” block type. Once a provider has been declared, the corresponding plugin is included while generating the plan. Declared resources utilize the provider’s underlying API to perform create, update and delete actions needed to ensure the resource ends up in the desired state. Providers come from a publicly available registry of known plugins<sup>2</sup>. The list is extensive and covers all known cloud, Software as a Service (SaaS) and other APIs. These provider plugins allow the resource and data source blocks to be declared without needing details on the specific provider’s API. Each provider maintains documentation on the Terraform blocks it supports along with the corresponding parameters.

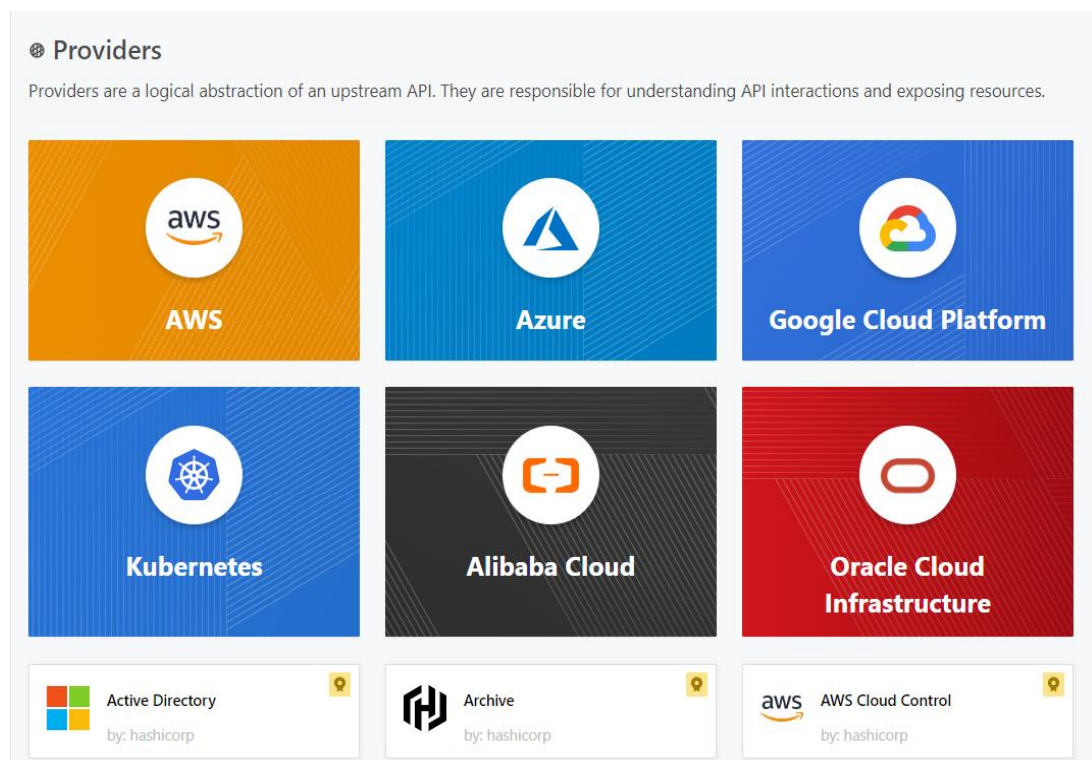


Fig3.3: Providers



## CHAPTER 4

### SOFTWARE DESCRIPTION

#### 4.1 General Introduction

##### 4.1.1 Configuration Language

Terraform scans configuration files and generates a corresponding plan. The configuration files are written in the HashiCorp Configuration Language (HCL). This language is declarative. Declarative offers an advantage over imperative in that the desired state of the infrastructure can be directly coded. An imperative language requires defining all the interim steps to arrive at the desired state. The main purpose of HCL is to define resources. The code is written in blocks with each block representing an infrastructure object.

A Terraform configuration is a complete document in HCL telling Terraform how to manage a given collection of infrastructure resources. Figure 3 shows example code that declares the required provider plugin as well as a VPC and corresponding subnet. The purpose of the block is defined by the block type. A variable block type is used as a parameter in other blocks. A provider defines what provider plugin is used to translate the resource block into API calls to the infrastructure provider. The resource block is used to define a concrete resource in the provider's infrastructure. Resource blocks are translated into create, update or delete API calls to the provider's target infrastructure service.

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

provider "registry.terraform.io/hashicorp/aws" {
  version = "4.46.0"
  hashes = [
    "h1:mRM6S6Yqze/+bhLrRbvWJYNcnQzI7Q/78pLq15KJWI=",
    "zh:1678e6a4bdb3d81a6713adc62ca0fdb8250c584e10c10d1daca72316e9db8df2",
    "zh:329903acf86ef6072502736dff4c43c2b50f762a958f76aa924e2d74c7fca1e3",
    "zh:33db8131fe0ec7e1d9f30bc9f65c2440e9c1f708d681b6062757a351f1df7ce6",
    "zh:3a3b010bc393784c16f4b6cdce7f76db93d5efa323fce4920bfea9e9ba6abe44",
    "zh:979e2713a5759a7483a065e149e3cb69db9225326fc0457fa3fc3a48aed0c63f",
    "zh:9b12af85486a96aedd8d7984b0ff811a4b42e3d88dad1a3fb4c0b580d04fa425",
    "zh:9efcf0067e16ad53da7504178a05eb2118770b4ae00c193c10ecad4cbfce308e",
    "zh:a10655bf1b6376ab7f3e55efadf54dc70f7bd07ca11369557c312095076f9d62",
    "zh:b0394dd42cbd2a718a7dd7ae0283f04769aaf8b3d52664e141da59c0171a11ab",
    "zh:b958e614c2cf6d9c05a6ad5e94dc5c04b97ebfb84415da068be5a081b5ebbe24",
    "zh:ba5069e624210c63ad9e633a8eb0108b21f2322bc4967ba2b82d09168c466888",
    "zh:d7dfa597a17186e7f4d741dd7111849f1c0dd6f7ebc983043d8262d2fb37b408",
    "zh:e8a641ca2c99f96d64fa2725875e797273984981d3e54772a2823541c44e3cd3",
    "zh:f89898b7067c4246293a8007f59f5cfcac7b8dd251d39886c7a53ba596251466",
    "zh:fb1e1df1d5cc208e08a850f8e84423bce080f01f5e901791c79df369d3ed52f2",
  ]
}
```

Fig.4.1: HCL code

### 4.1.2 Framework Environment

Environments that can run Terraform are the CLI, Terraform Cloud, Terraform Enterprise and CDK. The CLI is the most common. Pre-built binaries can be downloaded or the Golang source code<sup>1</sup> can be cloned and built. The Terraform tool runs on a local set of configuration files. These files can be organized into subdirectories which Terraform will automatically traverse. The state file is typically generated in the same directory that the tool runs from. However, there is a remote option which generates state files in a remote, central location such that multiple Terraform clients may apply their plans and still synchronize their view of the existing infrastructure. Terraform Cloud is a Terraform environment hosted by HashiCorp. As a hosted service, users log in to generate and apply plans.

The Terraform tool-chain itself is maintained by HashiCorp while the state files are centrally stored such that all users are running against the same current infrastructure state. Terraform Enterprise is a self-hosted version of Terraform Cloud. It offers the same cloud-based feature set but is designed to be deployed within an enterprise's private cloud. Another version of the CLI or local environment is the Cloud Development Kit (CDK). Rather than running the CLI tool directly, CDK permits five supported high-level languages to generate and apply Terraform plans. Code in these supported languages is able to call in to the Terraform framework, replacing the Terraform CLI. Section VI provides further details on CDK.

### 4.1.3 Cloud Development Kit

The Cloud Development Kit (CDK) for Terraform allows the use of other programming language to define and provision infrastructure. CDK gives access to the entire Terraform ecosystem without requiring development in HashiCorp Configuration Language (HCL) and running it via the CLI tool. Additionally, a user can more easily integrate with an existing tool-chain for testing and dependency management.

The following languages are currently supported: • Typescript • Python • Java • C#. CDK may be invoked from its five supported languages while configuration

code in HCL or JSON require the Terraform CLI. Kubernetes' Custom Resource Definitions (CRDS) are another possibility but will not be covered here.

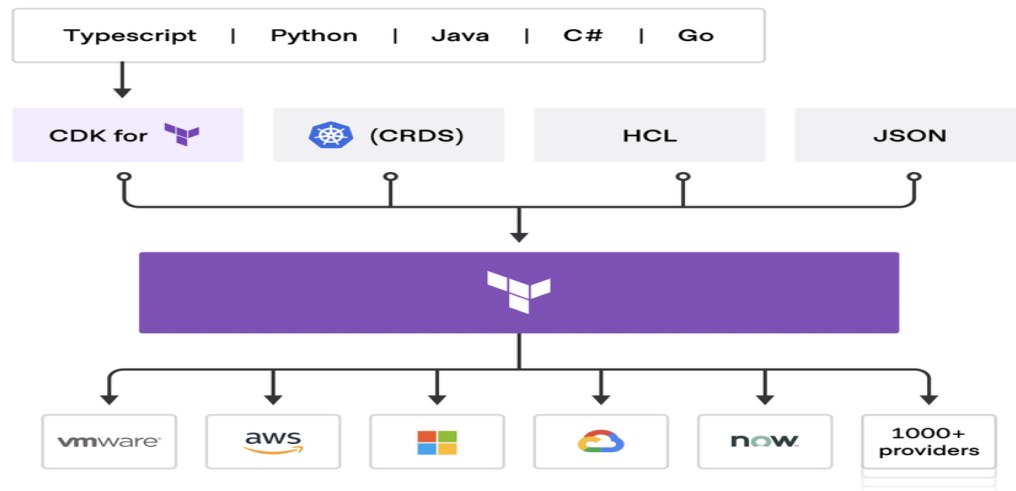


Fig.4.1.3: CDK and other pathways to define configuration, input to Terraform and provision infrastructure through multiple providers. Configuration input may be through CDK, CRDS, HCL or JSON.

## 4.2 Software Requirements

- Operating System: - Windows 7 or above
- Language: HCL
- Framework: Cloud
- Editor: Atom
- Backend Database: AWS

## 4.3 Methodology

### 4.3.1 What is ec2 instance

Amazon Elastic Compute Cloud (Amazon EC2) provides scalable computing capacity in the Amazon Web Services (AWS) Cloud. Using Amazon EC2 eliminates your need to invest in hardware up front, so you can develop and deploy applications faster. You can use Amazon EC2 to launch as many or as few virtual servers as you need, configure security and networking, and manage storage. Amazon EC2 enables you to scale up or down to handle changes in requirements or spikes in popularity, reducing your need to forecast traffic.

An Amazon EC2 instance is a virtual server in Amazon's Elastic Compute Cloud (EC2) for running applications on the Amazon Web Services (AWS) infrastructure. Set of machines to run our program.

### 4.3.2 What is ami in ec2

An Amazon Machine Image (AMI) is a master image for the creation of virtual servers known as EC2 instances in the Amazon Web Services (AWS) environment. The machine images are like templates that are configured with an operating system and other software that determines the user's operating environment then check for aws-instance

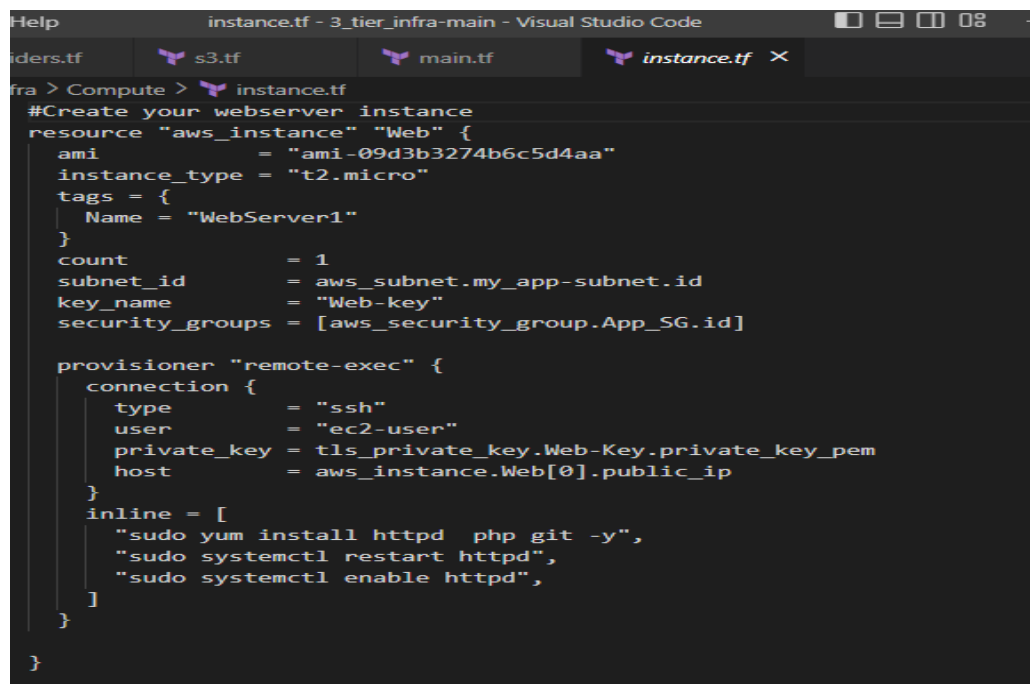
-> go to terraform aws docs

-> ec2 from left side of docs

-> look for aws instance

-> copy resource code

after that search for ami and instance\_type, ami depends on the region. instance\_type depends on you infra configure "t2.micro"



```

Help    instance.tf - 3_tier_infra-main - Visual Studio Code
iders.tf  s3.tf  main.tf  instance.tf X
fra > Compute > instance.tf
#Create your webserver instance
resource "aws_instance" "Web" {
  ami           = "ami-09d3b3274b6c5d4aa"
  instance_type = "t2.micro"
  tags = {
    Name = "WebServer1"
  }
  count              = 1
  subnet_id         = aws_subnet.my_app-subnet.id
  key_name          = "Web-key"
  security_groups   = [aws_security_group.App_SG.id]

  provisioner "remote-exec" {
    connection {
      type      = "ssh"
      user      = "ec2-user"
      private_key = tls_private_key.Web-Key.private_key_pem
      host      = aws_instance.Web[0].public_ip
    }
    inline = [
      "sudo yum install httpd php git -y",
      "sudo systemctl restart httpd",
      "sudo systemctl enable httpd",
    ]
  }
}

```

Fig.4.3.2: EC2 instance

### 4.3.3 Security group

Active Directory has two forms of common security principals: user accounts and computer accounts. These accounts represent a physical entity that is either a person or a computer. A user account also can be used as a dedicated service account for some applications.

Security groups are a way to collect user accounts, computer accounts, and other groups into manageable units.

In the Windows Server operating system, several built-in accounts and security groups are preconfigured with the appropriate rights and permissions to perform specific tasks. In Active Directory, administrative responsibilities are separated into two types of administrators:

- **Service administrators:** Responsible for maintaining and delivering Active Directory Domain Services (AD DS), including managing domain controllers and configuring AD DS.
- **Data administrators:** Responsible for maintaining the data that's stored in AD DS and on domain member servers and workstations.

```
#create a security group

resource "aws_security_group" "app_sg" {
  name           = "App_SG"
  description    = "Allow Web Inbound Traffic"
  vpc_id         = aws_vpc.my_vpc.id
  ingress {
    from_port     = 80
    to_port       = 80
    protocol      = "tcp"
    cidr_blocks   = ["0.0.0.0/0"] #open to everyone
  }

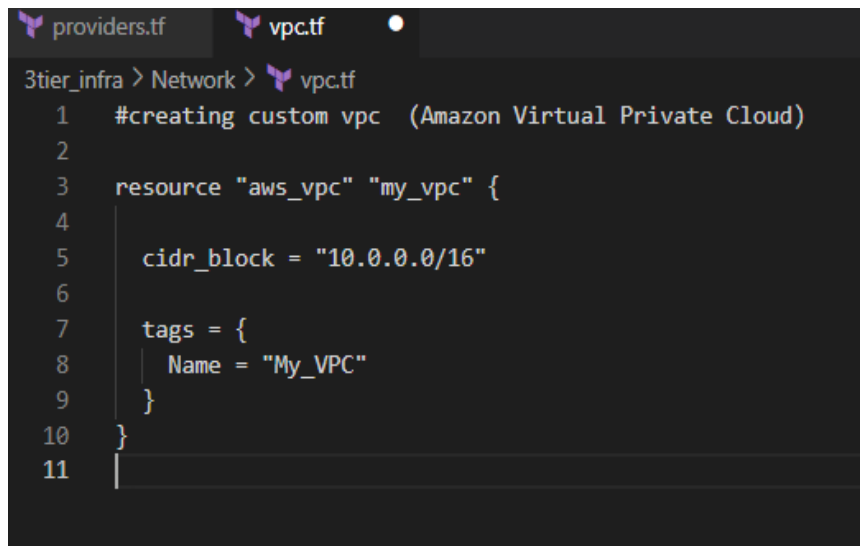
  ingress {
    from_port     = 22
    to_port       = 22
    protocol      = "tcp"
    cidr_blocks   = ["0.0.0.0/0"] #open to everyone
  }

  egress { #egress means to access return calls
    from_port     = 0
    to_port       = 0
    protocol      = "-1"
    cidr_blocks   = ["0.0.0.0/0"] #open to everyone
  }
}
```

Fig.4.3.3: Security Group

#### 4.3.4 Virtual Private Cloud

Amazon Virtual Private Cloud (VPC) is a commercial cloud computing service that provides users a virtual private cloud, by “provisioning a logically isolated section of Amazon Web Services (AWS) Cloud”. Enterprise customers are able to access the Amazon Elastic Compute Cloud (EC2) over an IPsec based virtual private network. Unlike traditional EC2 instances which are allocated internal and external IP numbers by Amazon, the customer can assign IP numbers of their choosing from one or more subnets. By giving the user the option of selecting which AWS resources are public facing and which are not, VPC provides much more granular control over security. For Amazon it is “an endorsement of the hybrid approach, but it's also meant to combat the growing interest in private clouds”.



```

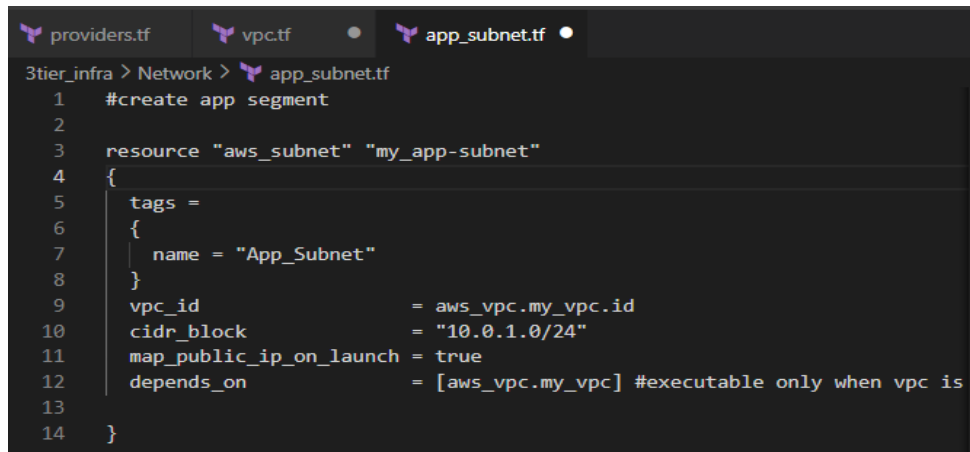
providers.tf  vpc.tf
3tier_infra > Network > vpc.tf
1  #creating custom vpc  (Amazon Virtual Private Cloud)
2
3  resource "aws_vpc" "my_vpc" {
4
5      cidr_block = "10.0.0.0/16"
6
7      tags = {
8          Name = "My_VPC"
9      }
10 }
11 |

```

Fig.4.3.4: Virtual Private Cloud

#### 4.3.5 APP\_SUBNET.tf

A subnet, or subnetwork, is a network inside a network. Subnets make networks more efficient. Through subnetting, network traffic can travel a shorter distance without passing through unnecessary routers to reach its destination.



```

providers.tf vpc.tf ● app_subnet.tf ●
3tier_infra > Network > app_subnet.tf
1  #create app segment
2
3  resource "aws_subnet" "my_app-subnet"
4  {
5      tags =
6      {
7          name = "App_Subnet"
8      }
9      vpc_id          = aws_vpc.my_vpc.id
10     cidr_block       = "10.0.1.0/24"
11     map_public_ip_on_launch = true
12     depends_on       = [aws_vpc.my_vpc] #executable only when vpc is
13
14 }

```

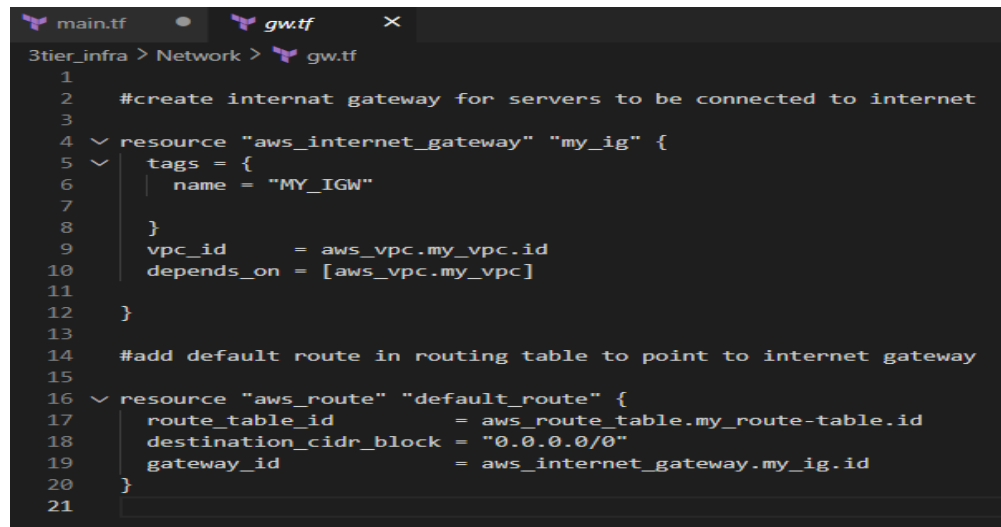
Fig.4.3.5: APP\_SUBNET.tf

### 4.3.6 GW.tf

An internet gateway is a horizontally scaled, redundant, and highly available VPC component that allows communication between your VPC and the internet. It supports IPv4 and IPv6 traffic. It does not cause availability risks or bandwidth constraints on your network traffic.

To enable access to or from the internet for instances in a subnet in a VPC using an internet gateway, you must do the following.

- Create an internet gateway and attach it to your VPC.
- Add a route to your subnet's route table that directs internet-bound traffic to the internet gateway.
- Ensure that instances in your subnet have a public IPv4 address or an IPv6 address.
- Ensure that your network access control lists and security group rules allow the desired internet traffic to flow to and from your instance.



```

1  #create internet gateway for servers to be connected to internet
2
3
4  resource "aws_internet_gateway" "my_ig" {
5    tags = {
6      name = "MY_IGW"
7    }
8
9    vpc_id      = aws_vpc.my_vpc.id
10   depends_on = [aws_vpc.my_vpc]
11 }
12
13
14 #add default route in routing table to point to internet gateway
15
16 resource "aws_route" "default_route" {
17   route_table_id      = aws_route_table.my_route_table.id
18   destination_cidr_block = "0.0.0.0/0"
19   gateway_id          = aws_internet_gateway.my_ig.id
20 }
21

```

Fig.4.3.6: GW.tf code

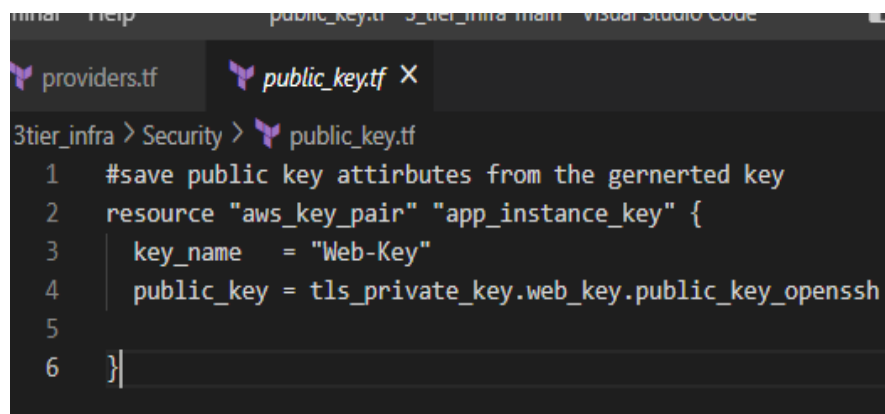
### 4.3.7 Public.key.tf

Provides an EC2 key pair resource. A key pair is used to control login access to EC2 instances.

Currently this resource requires an existing user-supplied key pair. This key pair's public key will be registered with AWS to allow logging-in to EC2 instances.

When importing an existing key pair the public key material may be in any format supported by AWS. Supported formats (per the AWS documentation) are:

- OpenSSH public key format
- Base64 encoded DER format
- SSH public key file format as specified in RFC4716.



```

1  #save public key attributes from the generated key
2  resource "aws_key_pair" "app_instance_key" {
3    key_name      = "Web-Key"
4    public_key    = tls_private_key.web_key.public_key_openssh
5  }
6

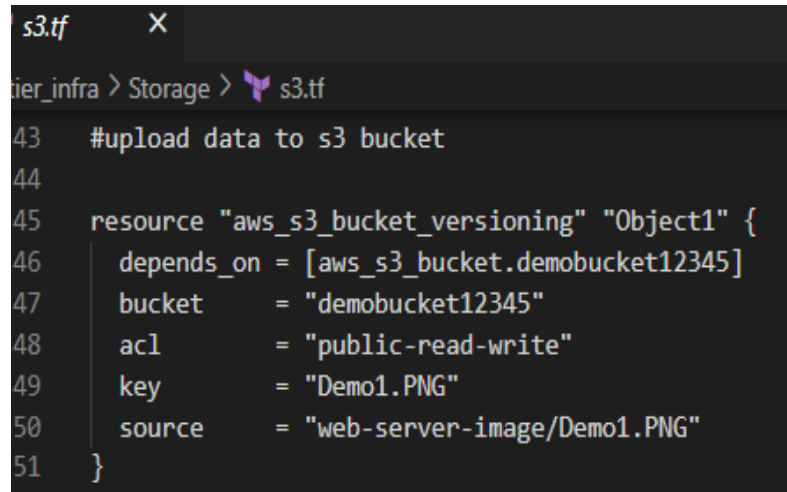
```

Fig.4.3.7: Public.key.tf



#### 4.3.8 S3.tf

Amazon Simple Storage Service (Amazon S3) is an object storage service that offers industry-leading scalability, data availability, security, and performance. You can use Amazon S3 to **store and retrieve any amount of data at any time, from anywhere.**

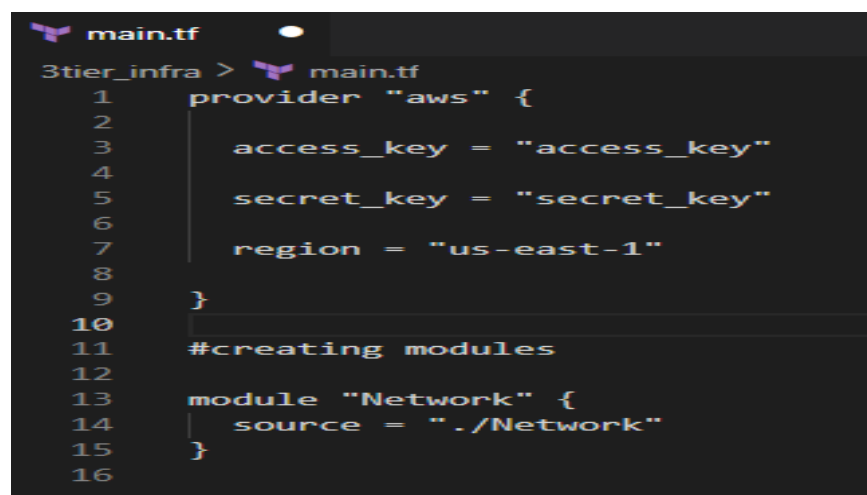


```
s3.tf
tier_infra > Storage > s3.tf
43 #upload data to s3 bucket
44
45 resource "aws_s3_bucket_versioning" "Object1" {
46     depends_on = [aws_s3_bucket.demobucket12345]
47     bucket      = "demobucket12345"
48     acl         = "public-read-write"
49     key         = "Demo1.PNG"
50     source      = "web-server-image/Demo1.PNG"
51 }
```

Fig.4.3.8: S3.tf code

#### 4.3.9 Main.tf

main.tf will **contain the main set of configurations for your module.** You can also create other configuration files and organize them however makes sense for your project. variables.tf will contain the variable definitions for your module.



```
main.tf
3tier_infra > main.tf
1  provider "aws" {
2
3      access_key = "access_key"
4
5      secret_key = "secret_key"
6
7      region = "us-east-1"
8
9  }
10
11 #creating modules
12
13 module "Network" {
14     source = "../Network"
15 }
16
```

Fig.4.3.9: Main.tf

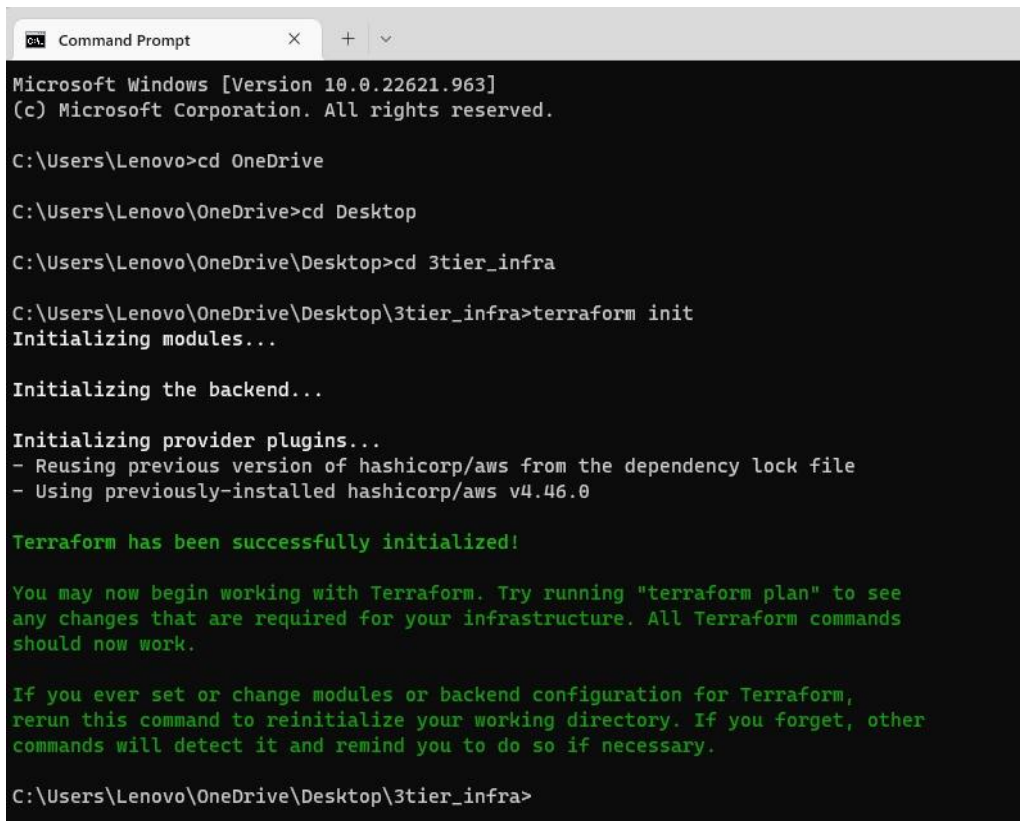
## CHAPTER 5

### RESULTS AND DISCUSSION

#### 5.1 Before Applying Terraform plan:

##### 5.1.1 Terraform init

We need to create an infrastructure building block using html scripts for 3 tier infrastructure. For this we need to create modules of all the resources and execute it on our local system i.e., cmd. Where all the files are stored by using **terraform init** command.



```
Microsoft Windows [Version 10.0.22621.963]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Lenovo>cd OneDrive
C:\Users\Lenovo\OneDrive>cd Desktop
C:\Users\Lenovo\OneDrive\Desktop>cd 3tier_infra
C:\Users\Lenovo\OneDrive\Desktop\3tier_infra>terraform init
Initializing modules...

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v4.46.0

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

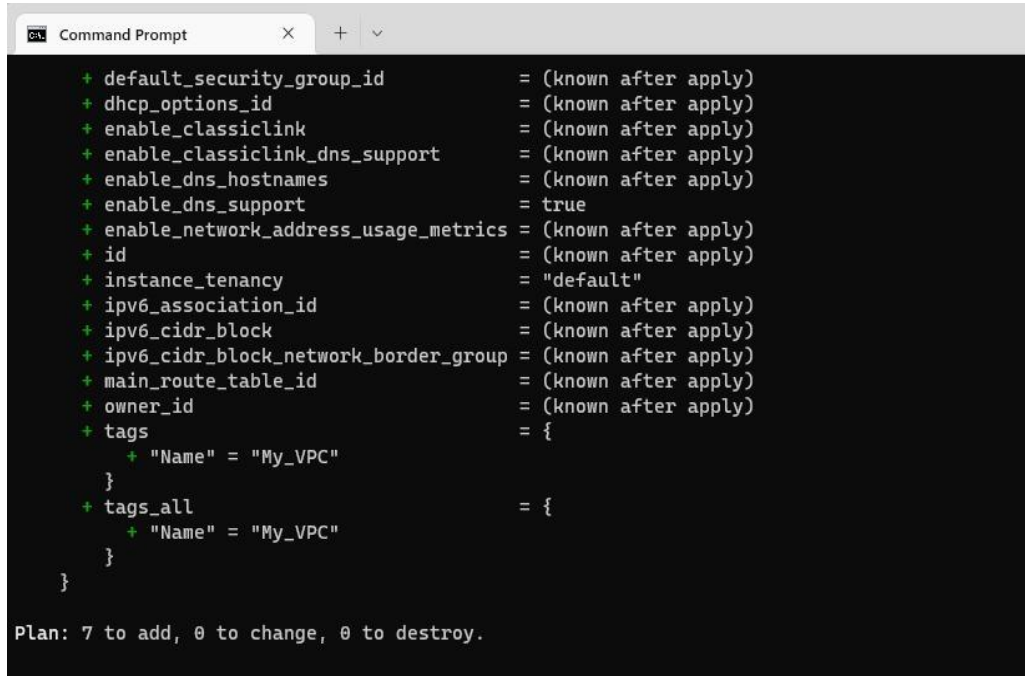
If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.

C:\Users\Lenovo\OneDrive\Desktop\3tier_infra>
```

Fig:5.1.1terraform init

### 5.1.2 Terraform plan

After initialising terraform to our folder, we then apply **terraform plan** to check for any errors.



```

+ default_security_group_id      = (known after apply)
+ dhcp_options_id                = (known after apply)
+ enable_classiclink             = (known after apply)
+ enable_classiclink_dns_support = (known after apply)
+ enable_dns_hostnames           = (known after apply)
+ enable_dns_support             = true
+ enable_network_address_usage_metrics = (known after apply)
+ id                             = (known after apply)
+ instance_tenancy               = "default"
+ ipv6_association_id            = (known after apply)
+ ipv6_cidr_block                = (known after apply)
+ ipv6_cidr_block_network_border_group = (known after apply)
+ main_route_table_id            = (known after apply)
+ owner_id                      = (known after apply)
+ tags                           = {
+   "Name" = "My_VPC"
+ }
+ tags_all                       = {
+   "Name" = "My_VPC"
+ }
}

Plan: 7 to add, 0 to change, 0 to destroy.

```

Fig:5.1.2 terraform plan

### 5.1.3 AWS dashboard before applying to terraform apply

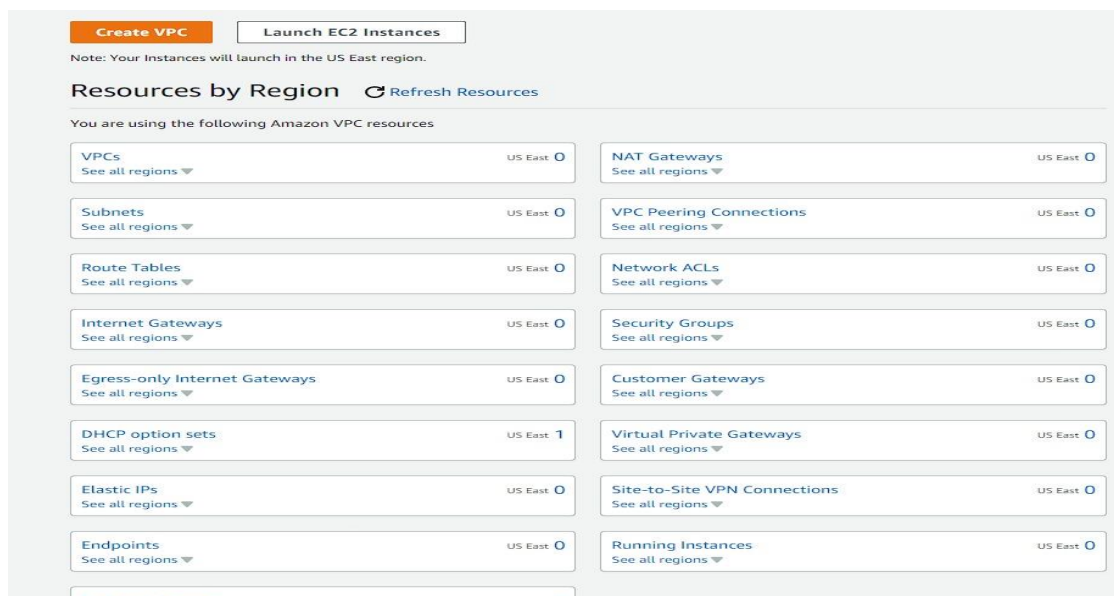
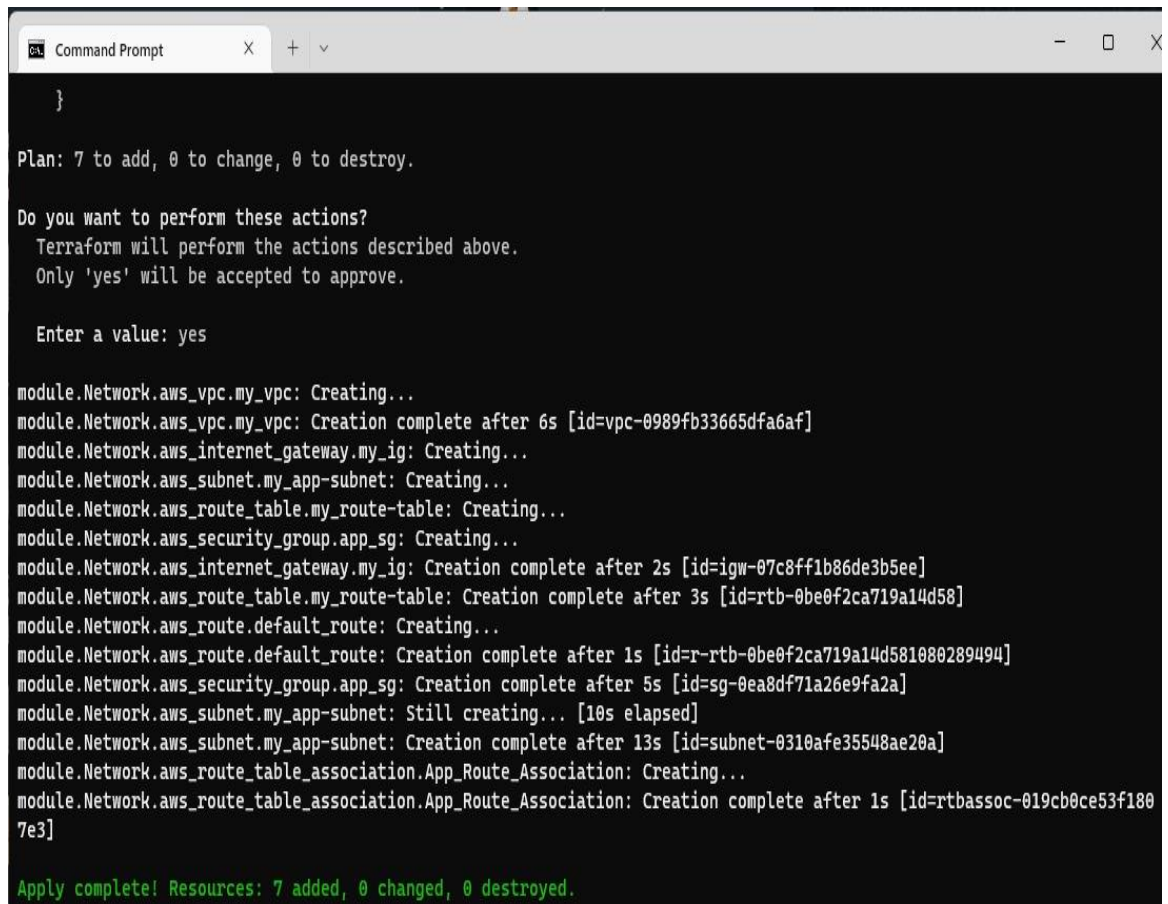


Fig. 5.1.3: Before Applying Terraform plan

### 5.1.4 Terraform Apply

Once the code is ready to execute, we can use **terraform apply** command



```

}

Plan: 7 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

Enter a value: yes

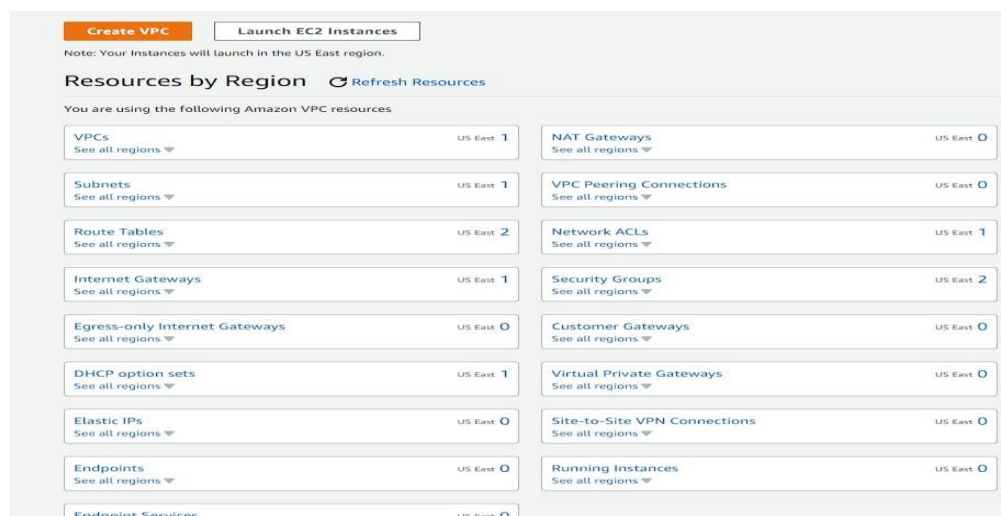
module.Network.aws_vpc.my_vpc: Creating...
module.Network.aws_vpc.my_vpc: Creation complete after 6s [id=vpc-0989fb33665dfa6af]
module.Network.aws_internet_gateway.my_ig: Creating...
module.Network.aws_subnet.my_app-subnet: Creating...
module.Network.aws_route_table.my_route-table: Creating...
module.Network.aws_security_group.app_sg: Creating...
module.Network.aws_internet_gateway.my_ig: Creation complete after 2s [id=igw-07c8ff1b86de3b5ee]
module.Network.aws_route_table.my_route-table: Creation complete after 3s [id=rtb-0be0f2ca719a14d58]
module.Network.aws_route.default_route: Creating...
module.Network.aws_route.default_route: Creation complete after 1s [id=r-rtb-0be0f2ca719a14d580289494]
module.Network.aws_security_group.app_sg: Creation complete after 5s [id=sg-0ea8df71a26e9fa2a]
module.Network.aws_subnet.my_app-subnet: Still creating... [10s elapsed]
module.Network.aws_subnet.my_app-subnet: Creation complete after 13s [id=subnet-0310afe35548ae20a]
module.Network.aws_route_table_association.App_Route_Association: Creating...
module.Network.aws_route_table_association.App_Route_Association: Creation complete after 1s [id=rtbassoc-019cb0ce53f1807e3]

Apply complete! Resources: 7 added, 0 changed, 0 destroyed.

```

Fig:5.1.4 terraform apply

### 5.2 After Applying Terraform plan



Resources by Region		Refresh Resources
You are using the following Amazon VPC resources		
VPCs See all regions	US East 1	NAT Gateways See all regions
Subnets See all regions	US East 1	VPC Peering Connections See all regions
Route Tables See all regions	US East 2	Network ACLs See all regions
Internet Gateways See all regions	US East 1	Security Groups See all regions
Egress-only Internet Gateways See all regions	US East 0	Customer Gateways See all regions
DHCP option sets See all regions	US East 1	Virtual Private Gateways See all regions
Elastic IPs See all regions	US East 0	Site-to-Site VPN Connections See all regions
Endpoints See all regions	US East 0	Running Instances See all regions
Endpoint Services See all regions	US East 0	

Fig. 5.2: After Applying Terraform plan

### 5.2.1 Terraform Destroy

Once we complete the task on the platform, we need to destroy the plan created so stop further billing for our project, so we use **terraform destroy** command to close all the running resources and instance created.

```

- "Name" = "My_VPC"
} -> null
}

Plan: 0 to add, 0 to change, 7 to destroy.

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

Enter a value: yes

module.Network.aws_route_table_association.App_Route_Association: Destroying... [id=rtbassoc-019cb0ce53f1807e3]
module.Network.aws_route.default_route: Destroying... [id=r-rtb-0be0f2ca719a14d581080289494]
module.Network.aws_security_group.app_sg: Destroying... [id=sg-0ea8df71a26e9fa2a]
module.Network.aws_route_table_association.App_Route_Association: Destruction complete after 2s
module.Network.aws_subnet.my_app-subnet: Destroying... [id=subnet-0310afe35548ae20a]
module.Network.aws_route.default_route: Destruction complete after 3s
module.Network.aws_internet_gateway.my_ig: Destroying... [id=igw-07c8ff1b86de3b5ee]
module.Network.aws_route_table.my_route-table: Destroying... [id=rtb-0be0f2ca719a14d58]
module.Network.aws_security_group.app_sg: Destruction complete after 4s
module.Network.aws_internet_gateway.my_ig: Destruction complete after 1s
module.Network.aws_subnet.my_app-subnet: Destruction complete after 1s
module.Network.aws_route_table.my_route-table: Destruction complete after 2s
module.Network.aws_vpc.my_vpc: Destroying... [id=vpc-0989fb33665dfa6af]
module.Network.aws_vpc.my_vpc: Destruction complete after 1s

Destroy complete! Resources: 7 destroyed.

C:\Users\Lenovo\OneDrive\Desktop\3tier_infra>

```

Fig:5.2.1 terraform destroy

## CHAPTER 6

### BUSINESS PLAN COMMERCIAL ASPECT

#### 6.1 Commercial Aspect

- **IaC boosts productivity through automation**

One of the first and most obvious benefits of an infrastructure model like IaC is that it improves the productivity of your operations teams across the IT sector, and allows you to automate all infrastructure processes and changes to save time, money, and minimize the risk of human error. Up until the creation of IaC, infrastructure changes had to be handled and managed through extensive and complex manual work, which would invariably drain resources and sometimes cause setbacks to occur.

- **Minimizing risk of human error**

That's why implementing Infrastructure as Code should be your priority, as it standardizes all processes and logs them to create detailed reports and documentation of how the infrastructure works, how it's managed and deployed, and how new employees can continue the work without setbacks. This is a much-needed failsafe for when the top IT experts leave your company, carrying institutional knowledge with them out the door.

- **Facilitating financial savings**

Given the fact that one of the core functionalities of the Infrastructure as Code model is to automate all processes in your IT infrastructure, it allows your engineers and other IT experts to shift their focus from manual grunt work to more mission-critical tasks and projects. This allows you, as their leader, to minimize payroll costs by making better hiring decisions and scale the salaries of your employees to fit their roles and workload. You can use the IaC function of spinning down environments when they're not in use to maximize financial savings even further over the long term.

- **Increased efficiency in software development**

IaC allows IT experts to maximize efficiency and productivity on numerous fronts, especially when it comes to software development and integration with DevOps, for example. Given the fact that you can deploy cloud architectures in numerous stages to improve the efficiency of the development process, your developers can develop new software while constantly making incremental progress in a controlled sandbox environment.

What's more, IaC allows your QA (Quality Assurance) department to continuously test different iterations of the software at different stages by always having an updated copy of the production, and you can deploy the infrastructure easily without any setbacks.

## **6.2 Business Plan**

### **1. Decreased risk**

Provisioning all your infrastructure by hand is risky. It requires manual work that is error-prone. It may require a single person to do. That person could leave the company, taking all that knowledge with them. Infrastructure as code minimizes both of these risks. By representing infrastructure as reproducible blocks of code we are far less error-prone. Infrastructure as code lives in a source code repository. Its history and changes are visible to everyone on the team.

### **2. Stable & consistent environments for faster iterations**

When environments have to be manually configured or modified it slows down product development. This is especially true if the product wants to change its architecture to better serve its users. With infrastructure as code environments are stable, consistent, and easily modifiable. They live in code alongside the product, so when we want to change one we can change the other at the same time. This harmony means that new features can be developed for the product faster. There is less overhead to managing a given environment.



### **3. Cost optimization**

When all resources are represented in code you can see what is running and what shouldn't be. Optimizing cost maintains product profit margins. Those optimizations become much easier with infrastructure as code.

### **4. Self-documenting**

There is a philosophy in software development that says good code is easy to read. It often doesn't need extensive comments because it's clear what it's doing. The idea is that a new developer should be able to come in, read the code, and understand the logic that is happening. With infrastructure as code, it is self-documenting like any other code. This can benefit the product by making it easier to add more people to the team. With self-documenting code, you can reduce the time it takes for a new developer to onboard into the team.



## CHAPTER 6

### CONCLUSION

#### 6.1 Applications

##### •IT sector.

The information technology (IT) sector includes companies that produce software, hardware or semiconductor equipment, and companies that provide internet or related services. To migrate data from local system to cloud platforms.

The information technology (IT) sector includes companies that produce software, hardware or semiconductor equipment, and companies that provide internet or related services, Business Process Management (BPM), software products and engineering services, and hardware.

##### •Any IT industry from small – large scale, who want to Automate their work.

Automating infrastructure provisioning with IaC means that developers don't need to manually provision and manage servers, operating systems, storage, and other infrastructure components each time they develop or deploy an application.

These days each and every company is following the process of automation the work to have a secured, scalable, easy to edit and high efficient work flow.

##### • Where scalability and Speed of deployments are on high priorities.

Scalability refers to the ability of the business to set up its systems to grow during times of high demand and scale back when demand decreases. The optimum environment for achieving the appropriate level of scalability is DevOps, thanks to IaC. People can interact, concentrate on their work, have more creative alternatives, and swiftly deploy software through these techniques.

IaC give us leverage to deploy any code or infrastructure at a higher speed as it is easy to edit and highly scalable, we just need to look for the infrastructure needs and edit the code accordingly and deploy to the production team.

## **6.2 Advantages**

### **1. Cost reduction**

Every task done contributes to the heavy financial budget of every building team. Most times, the time taken to complete the task results in high pricing. However, since significant tasks, like time-consuming infrastructure configuration, are automated by IaC, the engineers or IT experts can finish these tasks in no time and focus on other mission-critical tasks. This would help minimize costs from necessary tasks and improve salaries of other functions according to their demands. The team can even decide to save this money for important decisions in the future.

### **2. Speed**

There are many processes in building, monitoring, and managing infrastructure structures, but as IaC has made it possible to automate almost every process, the work can be two times faster. Automation stretches from significant procedures like virtualization to user account management, databases, network management, and even minor operations like adding environments and resources when needed or shutting down when it is not the case. This automation feature promises faster and simpler procedures. This speed is not only for the developers but for the rest of the team as simple code improves clarity and, in turn, efficiency and speed.

### **3. Low risk of human errors**

On this benefit, there are two descriptions of human errors. The first is the face meaning which implies mistakes made by engineers or IT personnel in the manual process of infrastructure building. The second side to human error is the adaptation of new employees to the infrastructure built by the former IT expert. IaC solves these two issues as automation reduces the risks of human-made mistakes by cutting down long processes.

Also, IaC standardized and logs all processes during the building stage. This creates detailed reports and documentation of how the infrastructure works, how

it is managed, deployed, and how new employees can continue managing without setbacks.

#### **4. Improved consistency**

Still on human errors, infrastructure as code aids consistency as it helps avoid human errors and incompatibility in activities, like deployment and configuration that are prone to the already stated. Paradoxically, one must execute these activities prone to inconsistencies as fast as possible. This makes it almost impossible to manually carry out activities like deployment and configuration without at least one inconsistency.

IaC improves consistency during these activities by preventing waste of valuable resources, unwanted downtime, and setbacks that can cause inconsistencies in the configuration. These inconsistencies can be challenging to remove when discovered.

#### **5. Eliminate configuration drift**

Another critical benefit of infrastructure as code is that it is idempotent. This implies that one can deploy the code many times, with the first deployment being the actual deployment and subsequent deployments having no essential effect.

Amongst the many advantages of this idempotency feature, the most striking is how IaC prevents configuration drift. If someone changes a resource, not in sync with your IaC pipeline, you need to correct this and get the resources back to the correct state as fast as possible. But with IaC, the specifications of your environment configuration are already in the code. This implies that the correction happens automatically if the change is implemented.

#### **6. Improved security strategies**

Amongst other features of IaC, a feature that can help in improving security strategies is the one-way deployment feature. With IaC, computing, storage, and networking services are all provisioned with code and deployed the same way, often using a private or public cloud. This can also be the case for security standards. They can be easily created and deployed. This would enhance other strategies as this is a way to avoid security gatekeeper review and approval for

almost every security change. This would prove most beneficial for infrastructures requiring tight security, resulting in multiple security passes.

## **7. Accountability**

This benefit is an accumulated result of various features of infrastructure as code, majorly self-documentation and source control. These features document actions made in the configuration and give you access to the source code, respectively. These features make the source code of the configuration easily accessible and transparent enough to explain why users made specific changes when they were made and the result of those changes. Thus, IaC improves accountability and visibility of automated configuration and deployment methods to the other team members, which in turn improves the team's productivity.

## **8. Stable and scalable environment**

Using IaC, teams can configure their environments according to their specifications rapidly and at scale. As stated earlier, they can also create these environments with sure consistency as they can deploy the configuration several times. These several deployments also make the environment stable, preventing incompatibility caused by configuration drift or missing dependencies. In conclusion, working with best practices and IaC tools to create environments is a sure bet to make a stable one due to the impressive features of IaC.

## **9. Self-documentation**

Remember that IaC creates detailed reports and documentation of every process. This is another impressive feature of IaC that holds multiple benefits. When employees leave a company or a person is setting up the code model for another person, IaC documents every action, process, and change. IaC also tracks and tests every configuration like a code. This documentation can be a yardstick for a new employee or configuration manager.

### 6.3 Disadvantages

While introducing IaC as an approach for automated infrastructure management can give you a crucial competitive edge, it also has certain limitations. Here are a few key disadvantages:

**Conventions and logic:** Your developers need to understand IaC scripts fully. Irrespective of whether they are written in HashiCorp Configuration Language (HCL) or plain Python or Ruby, they need to be confident in applying the logic and conventions. Even if a small part of the engineering team is unfamiliar with the declarative approach or other core IaC concepts, it could become a bottleneck. If everybody in the team needs to understand these scripts but doesn't, onboarding and rapid scaling can be challenging.

**Traceability and maintenance:** While IaC gives you a great way to track changes to infrastructure and monitoring of the infrastructure drift, maintaining your IaC setup becomes a challenge once it crosses a certain scale. When IaC is used throughout the organization by multiple teams, the traceability and versioning of these configurations become complex.

**Role-Based Access Control (RBAC):** Access Management can also become a challenge. Setting up roles and permissions across different parts of your organization that have recent access to scripts can be quite demanding.

**Feature lag:** IaC tooling that is vendor-agnostic, such as Terraform, often lags behind vendor feature releases. This could be due to tool vendors having to update providers to fully cover the latest cloud features. It will be tough to leverage a new cloud feature unless you extend functionalities, wait for your vendor to provide coverage, or introduce new dependencies.

What does the future hold for the constantly evolving infrastructure automation market?

## 6.4 Conclusion

Infrastructure as Code is slowly but surely becoming the norm for organizations that seek automation and faster delivery. Developing applications at a faster rate is only possible through a streamlined workflow and an improved development environment.

However, coming up with optimal IaC solutions for your unique IT architecture isn't something that should be approached lightly, with insufficient resources, or the lack of guidance. But once you've set up your IaC environment the right way, your development process will immediately start yielding results.

## 6.5 Future Scope

The IaC market has been in a state of constant evolution with new solutions to emerging challenges being experimented with.

While a couple of years ago, the debate was between using CloudFormation and Terraform for IaC, with CloudFormation being the first choice, the choice a year ago was between CloudFormation and Terraform, with Terraform being the first choice.

It has become clear that engineers will prefer a multi-vendor, open-source and open-approach IaC framework with a vast third-party ecosystem like Terraform.

This means that more organizations will move more towards code than GUI, using several layers of code through IaC. The future will be more about managing code in order to improve the efficiency of developers while at the same time creating strong policies for your organization.

When it comes to infrastructure management, one of the main benefits of IaC will be the ability to create and reuse parts of the defined infrastructure. For instance, terraform can use different modules to encapsulate logically connected components into a single entity and customize them using the same input variables that you have defined. When you use these modules to define your infrastructure at a high level, it becomes far easier to separate the development, staging, and production environments just by-passing different values to the same

modules. This minimizes code duplication, maximizes conciseness and lets you automate extensive workflows through reusable IaC.

At InspiriSYS, we have moved beyond the traditional DevOps framework to a far more holistic approach through IaC, allowing your business to make seamless and rapid changes to the existing infrastructure. We have nearly three decades of expertise in providing advanced solutions which can automate every stage of your IT infrastructure, end-to-end. Our popular IT infrastructure automation solutions will let you kickstart your business journey successfully. Get in touch with us for the best-in-class IaC solutions.

## REFERENCES

[1] HashiCorp, Terraform Documentation. Accessed: May 2, 2022. [Online].

Available: <https://www.terraform.io/docs>

[2] Wikipedia, OASIS (organization). Accessed: May 6, 2022. [Online].

Available: [https://en.wikipedia.org/wiki/OASIS \(organization\)](https://en.wikipedia.org/wiki/OASIS_(organization))

[3] Oasis Open, TOSCA Version 2.0. Accessed: May 7, 2022. [Online].

Available: <https://docs.oasis-open.org/tosca/TOSCA/v2.0/TOSCAv2.0.html>

[4] Oasis Open, CAMP Charter. Accessed: May 7, 2022. [Online]. Available:

<https://www.oasis-open.org/committees/camp>

[5] <https://www.terraform.io/>

[6] <https://www.ibm.com/topics/terraform>

[7] <https://registry.terraform.io/providers/hashicorp/aws/latest/docs>