Expert Cloud Consulting
Enhance Optimise & Scale
ASCP GPUonCLOUD Pvt Ltd

“Expert Cloud Consulting” -

Documentation for Terraform-Based Deployment of Multi-Tier Architecture on AWS with ALB, Bastion Host, and Private Web Server [Title,18, Arial]

07.May.2025 [Subtitle,14, Arial]

version 1.0

—

Contributed by Yogiraj Deshpande [Normal text,14, Arial]
Approved by Akshay Shinde(In Review)
Expert Cloud Consulting
Office #811, Gera Imperium Rise,
Hinjewadi Phase-II Rd, Pune, India – 411057

Week 5: Introduction to Infrastructure as Code (IaC)

Topics :

- Basics of Terraform and CloudFormation.
- Provisioning infrastructure on AWS.

Assignments:

- Use Terraform to create a multi-tier architecture:
- VPC, subnets, security groups.
- EC2 instances in private subnets.
- An internet-facing load balancer.

- Write a CloudFormation template to automate:
- Creation of an S3 bucket with versioning
- A Lambda function triggered by S3 events.

Resources:

- Getting Started with CloudFormation
- GitLab CI/CD: GitLab CI/CD Tutorial

“Expert Cloud Consulting”
Documentation for Terraform-Based Deployment of Multi-Tier Architecture on AWS with ALB, Bastion Host, and Private Web Server[Title,18, Arial]

1.0: Document Overview:

This document describes the implementation of a Terraform script that provisions a multi-tier architecture on AWS. It automates the deployment of a Virtual Private Cloud (VPC), public and private subnets, an internet gateway, a NAT gateway, security groups, EC2 instances (including a bastion host and a web server), and an Application Load Balancer (ALB). This architecture is suitable for hosting scalable and secure web applications.

2.0: Objective:

- Provision infrastructure as code using Terraform for better manageability and repeatability.
- Set up a secure and scalable network with VPC, public/private subnets, and routing components.
- Launch EC2 instances for bastion access and private application servers.
- Configure an ALB to route HTTP traffic to internal servers.
- Apply security best practices using security groups and subnet isolation.

3.0: Prerequisites:

1. AWS account with appropriate IAM permissions (EC2, VPC, ELB, etc.).
2. Terraform CLI installed.
3. AWS CLI configured with credentials.
4. Public/private key pair for EC2 access.
5. Basic knowledge of networking and AWS infrastructure components.

4.0: Step-by-Step Implementation:

4.1: Provider and Terraform Initialization:

- Create `provider.tf` and define the AWS provider and region.
- Initialize Terraform using `terraform init`.

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

4.2: VPC Configuration:

- Define a custom VPC block (e.g., `10.0.0.0/16`) using the `aws_vpc` resource.:

```
resource "aws_vpc" "main" {
  cidr_block = "10.0.0.0/16"
  tags = {
    Name = "multi-tier-vpc"
  }
}
```

4.3: Subnet Configuration:

- Create public and private subnets in different Availability Zones for redundancy.

```
resource "aws_subnet" "public_1" {
  vpc_id     = aws_vpc.main.id
  cidr_block = "10.0.1.0/24"
  availability_zone = "us-east-1a"
  map_public_ip_on_launch = true
}
```

4.4: Internet Gateway and NAT Gateway:

- Attach an internet gateway to the VPC for public access.
- Create a NAT Gateway in a public subnet to allow outbound internet access from private subnets.

```
resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.main.id
}
```

4.5: Route Tables and Associations:

- Create separate route tables for public and private subnets.
- Associate them accordingly and define internet/NAT routes.

```
resource "aws_route_table" "public_rt" {
  vpc_id = aws_vpc.main.id
}
```

4.6: Security Group Configuration:

- Create security groups to restrict and allow traffic as per instance roles (bastion, web server, ALB)

```
resource "aws_security_group" "web_sg" {
  vpc_id = aws_vpc.main.id
  ingress {
    from_port = 80
    to_port   = 80
    protocol  = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

4.7: EC2 Instances Deployment:

- Launch a bastion host in the public subnet.
- Launch a web server in the private subnet with Apache preinstalled via user data.

```
resource "aws_instance" "web" {
  ami = "ami-xxxxxxx"
  subnet_id = aws_subnet.private.id
  user_data = <<-EOF
    #!/bin/bash
    apt update
    apt install -y apache2
    systemctl start apache2
  EOF
}
```

4.8: Application Load Balancer (ALB):

- Create an ALB in the public subnet.
- Attach the private EC2 instance to the target group.
- Forward HTTP traffic from the ALB to the private web server.

```
resource "aws_lb" "app_lb" {
  name               = "app-lb"
  load_balancer_type = "application"
  subnets            = [aws_subnet.public_1.id]
}
```

5.0: Final Deployment and Access:

- Run `terraform plan` to validate the configuration.
- Apply the infrastructure using `terraform apply`.
- Access the application through the ALB DNS endpoint.

6.0: Conclusion:

This Terraform configuration successfully provisions a multi-tier AWS architecture with public and private subnets. It includes an internet-facing ALB, Bastion host, and necessary security groups. The setup enables secure access and efficient traffic routing. Automation through Terraform ensures consistent infrastructure deployment. This approach supports scalable and manageable cloud environments.