Lab 21 – End-to-End Production Architecture Using Terraform Modules (VPC + ALB + ASG + EC2)

Creator: Sandeep Kumar Sharma

🔍Scenario – Real Production End-to-End Setup

Your company wants to standardize how they deploy **complete application stacks** in AWS using Terraform.

They don’t want random Terraform files everywhere. They want: - **Reusable modules** for network and compute. - A clean separation between **VPC**, **application layer (ALB + ASG + EC2)**, and later **database**. - A single root Terraform project that can: - Create the VPC (multi-AZ, public + private subnets). - Create ALB in public subnets. - Create Auto Scaling EC2 instances in private subnets. - Wire everything using modules.

Your task: **Build an end-to-end production-style architecture using Terraform modules**.

This lab ties together everything you have learned so far—**VPC, modules, ALB, ASG, security groups, user data**—into a single, clean solution.

🕑Learning Objectives

- Design a modular Terraform folder structure like real projects.
- Create a **network module** for VPC and subnets.
- Create an **app module** for ALB + ASG + EC2.
- Use the root module to wire everything together.
- Understand how modules communicate using inputs and outputs.

🔗Learning Outcomes

By the end of this lab, the learner will be able to: - Build a complete 2-tier application architecture using Terraform. - Use modules to separate concerns and increase reusability. - Pass VPC and subnet information from one module to another. - Deploy a production-style, internet-facing web application.

Concept Explanation (Natural Style)

Till now, we have written labs that focus on **one topic at a time**: - VPCs - Security groups - ALB - Auto Scaling - S3

In real life, you don't deploy them separately. You deploy a **complete architecture**.

Also, no one likes a huge `main.tf` with 500 lines.

Instead, production teams: - Break Terraform into **modules**. - Keep VPC logic in one module. - Keep app logic in another. - Use the **root module** as the "orchestrator" that calls other modules.

This lab simulates exactly that.

We'll create: - `modules/network` → VPC and subnets (wraps the official VPC module) - `modules/app` → ALB, security groups, launch template, ASG - Root → calls these modules and passes data between them

---

💣Architecture Overview

Network Layer (Module: `network`)

- VPC: `10.2.0.0/16`
- Public subnets in 3 AZs
- Private app subnets in 3 AZs
- NAT + IGW

App Layer (Module: `app`)

- Security group for ALB
- Security group for EC2
- ALB in public subnets
- Launch template with web server
- Auto Scaling Group in private subnets

🛠️Hands-On: Step-by-Step

Step 1 – Create Folder Structure

In your terminal:

```
mkdir -p terraform-lab21-end-to-end/modules/network
mkdir -p terraform-lab21-end-to-end/modules/app
cd terraform-lab21-end-to-end

# Root files
touch main.tf variables.tf outputs.tf

# Module files
cd modules/network
touch main.tf variables.tf outputs.tf
cd ../app
touch main.tf variables.tf outputs.tf
cd ../../  # back to root
```

Your structure:

```
terraform-lab21-end-to-end/
├── main.tf
├── variables.tf
├── outputs.tf
└── modules/
    ├── network/
    │   ├── main.tf
    │   ├── variables.tf
    │   └── outputs.tf
    └── app/
        ├── main.tf
        ├── variables.tf
        └── outputs.tf
```

🕰️Part 1 – Network Module (VPC + Subnets)

`modules/network/variables.tf`

```hcl
variable "vpc_cidr" {
  type        = string
  description = "CIDR block for VPC"
}

variable "azs" {
  type        = list(string)
```

```
  description = "List of availability zones"
}

variable "public_subnets" {
  type        = list(string)
  description = "CIDRs for public subnets"
}

variable "private_subnets" {
  type        = list(string)
  description = "CIDRs for app private subnets"
}
```

modules/network/main.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
  }
}

provider "aws" {
  region = "ap-south-1"
}

module "vpc" {
  source  = "terraform-aws-modules/vpc/aws"
  version = "5.1.2"

  name = "lab21-vpc"
  cidr = var.vpc_cidr

  azs             = var.azs
  public_subnets  = var.public_subnets
  private_subnets = var.private_subnets

  enable_nat_gateway = true
  single_nat_gateway = true

  tags = {
    Project = "Terraform-Lab21"
    Layer   = "network"
```

```
  }
}
```

`modules/network/outputs.tf`

```
output "vpc_id" {
  value = module.vpc.vpc_id
}

output "public_subnets" {
  value = module.vpc.public_subnets
}

output "private_subnets" {
  value = module.vpc.private_subnets
}
```

🕰️Part 2 – App Module (ALB + ASG + EC2)

`modules/app/variables.tf`

```
variable "vpc_id" {
  type = string
}

variable "public_subnets" {
  type = list(string)
}

variable "private_subnets" {
  type = list(string)
}

variable "instance_type" {
  type    = string
  default = "t2.micro"
}
```

`modules/app/main.tf`

```
provider "aws" {
  region = "ap-south-1"
}

# Security group for ALB
resource "aws_security_group" "alb_sg" {
  name   = "lab21-alb-sg"
  vpc_id = var.vpc_id

  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

# Security group for EC2 (only ALB allowed)
resource "aws_security_group" "ec2_sg" {
  name   = "lab21-ec2-sg"
  vpc_id = var.vpc_id

  ingress {
    from_port       = 80
    to_port         = 80
    protocol        = "tcp"
    security_groups = [aws_security_group.alb_sg.id]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

```
# Launch template
resource "aws_launch_template" "lt" {
  name_prefix   = "lab21-lt-"
  image_id      = "ami-052c08d70def0ac62"
  instance_type = var.instance_type

  vpc_security_group_ids = [aws_security_group.ec2_sg.id]

  user_data = base64encode(<<EOF
#!/bin/bash
yum install -y httpd
systemctl enable httpd
systemctl start httpd
echo "<h1>Terraform Lab21 – App Module Running</h1>" > /var/www/html/index.html
EOF
  )
}

# Target group
resource "aws_lb_target_group" "tg" {
  name     = "lab21-tg"
  port     = 80
  protocol = "HTTP"
  vpc_id   = var.vpc_id

  health_check {
    path = "/"
    port = "traffic-port"
  }
}

# ALB
resource "aws_lb" "alb" {
  name               = "lab21-alb"
  load_balancer_type = "application"
  security_groups    = [aws_security_group.alb_sg.id]
  subnets            = var.public_subnets
}

resource "aws_lb_listener" "listener" {
  load_balancer_arn = aws_lb.alb.arn
  port              = 80
  protocol          = "HTTP"

  default_action {
    type             = "forward"
    target_group_arn = aws_lb_target_group.tg.arn
  }
```

```
}

# Auto Scaling Group
resource "aws_autoscaling_group" "asg" {
  name                = "lab21-asg"
  max_size            = 4
  min_size            = 1
  desired_capacity    = 2

  vpc_zone_identifier = var.private_subnets

  launch_template {
    id      = aws_launch_template.lt.id
    version = "$Latest"
  }

  target_group_arns = [aws_lb_target_group.tg.arn]

  health_check_type         = "EC2"
  health_check_grace_period = 60
}
```

`modules/app/outputs.tf`

```
output "alb_dns" {
  value = aws_lb.alb.dns_name
}
```

🕰️Part 3 – Root Module (Orchestrator)

`variables.tf` **(root)**

```
variable "vpc_cidr" {
  type    = string
  default = "10.2.0.0/16"
}

variable "azs" {
  type = list(string)
  default = [
    "ap-south-1a",
    "ap-south-1b",
```

```
    "ap-south-1c",
  ]
}

variable "public_subnets" {
  type = list(string)
  default = [
    "10.2.1.0/24",
    "10.2.2.0/24",
    "10.2.3.0/24",
  ]
}

variable "private_subnets" {
  type = list(string)
  default = [
    "10.2.11.0/24",
    "10.2.12.0/24",
    "10.2.13.0/24",
  ]
}
```

`main.tf` (root)

```
provider "aws" {
  region = "ap-south-1"
}

module "network" {
  source = "./modules/network"

  vpc_cidr       = var.vpc_cidr
  azs            = var.azs
  public_subnets = var.public_subnets
  private_subnets = var.private_subnets
}

module "app" {
  source = "./modules/app"

  vpc_id          = module.network.vpc_id
  public_subnets  = module.network.public_subnets
  private_subnets = module.network.private_subnets
}
```

`outputs.tf` (root)

```
output "alb_dns_name" {
  value = module.app.alb_dns
}
```

📩Step 4 – Initialize Terraform

From the root (`terraform-lab21-end-to-end`):

```
terraform init
```

This downloads: - AWS provider - VPC module (inside network module)

📩Step 5 – Plan

```
terraform plan
```

You should see: - VPC and subnets - ALB - Security groups - Launch template - Auto Scaling group

---

📩Step 6 – Apply

```
terraform apply
```

Type **yes**.

At the end, Terraform prints:

```
alb_dns_name = "lab21-alb-xxxxxxxx.ap-south-1.elb.amazonaws.com"
```

🔍Step 7 – Validate in AWS Console

1. **VPC → Your VPCs**
2. Confirm `lab21-vpc` with correct CIDR.
3. **Subnets**
4. 3 public, 3 private.
5. **EC2 → Auto Scaling Groups**
6. `lab21-asg` with 2 instances.
7. **EC2 → Load Balancers**
8. `lab21-alb` → status `active`.
9. **Test in Browser**
10. Open `http://<alb_dns_name>`
11. You should see: `Terraform Lab21 – App Module Running`

🗑️Step 8 – Destroy (Optional)

```
terraform destroy
```

Type **yes**.

This will remove: - ALB - ASG + EC2 - Security groups - VPC and subnets

> ⚠️In a real environment, be careful with destroy. Here it's safe because it's a lab.

📝Summary

In Lab 21, you: - Designed a **modular Terraform project** like a real production setup. - Created a **network module** wrapping the official VPC module. - Created an **app module** for ALB + ASG + EC2. - Wired the modules together in the root configuration. - Deployed an end-to-end **internet-facing, auto-scaled web application**.

This is a very close representation of how real-world Terraform projects are structured and deployed in enterprises.

End of Lab 21 – End-to-End Production Architecture with Terraform Modules