Lab 18 – Production Deployment: ALB + Auto Scaling Group on Custom VPC

Creator: Sandeep Kumar Sharma

🔍Scenario – Real Production Requirement

Your company is deploying a large-scale web application. The architecture team has given you the following real-world requirements:

- Workloads **must run on EC2 instances** inside the private application subnets created in **Lab 17**.
- Traffic must be load-balanced using an **Application Load Balancer (ALB)** in public subnets.
- EC2 instances must scale automatically based on CPU usage.
- A **Launch Template + Auto Scaling Group (ASG)** must be created.
- The ALB must route traffic to ASG instances.
- Everything must follow a production-grade Terraform structure.

Your task: **Deploy ALB + ASG + Launch Template inside the production VPC created in Lab 17**.

🕑Learning Objectives

- Understand how load balancing and auto scaling work together in production.
- Configure Launch Template for EC2 metadata, user data, tags.
- Create an Auto Scaling Group across multiple AZs.
- Attach ASG instances to an ALB target group.
- Use security groups correctly between ALB and EC2.
- Deploy everything using Terraform in a reusable way.

🔗Learning Outcomes

By the end of this lab, you will be able to: - Build a real production ALB + ASG setup. - Create scalable architectures using Terraform. - Use VPC outputs (from Lab 17) to place resources in correct subnets. - Configure health checks, target groups, and scaling policies.

Concept Explanation (Natural Style)

In a production environment, you never run a website on a single EC2 instance.

Why? - What if that instance crashes? - What if traffic increases suddenly? - What if an AZ goes down?

That's why companies use:

🔧Auto Scaling Group

→ Automatically increases or decreases EC2 instances based on load.

🔧Application Load Balancer

→ Accepts traffic on port 80/443 and distributes across private EC2 instances.

This combination ensures: - High availability - Better performance - Zero downtime during scaling

Now let's implement this using Terraform.

💣Architecture You Will Build

Public Subnets:

- ALB is deployed here.
- It accepts external HTTP traffic.

Private App Subnets:

- ASG + EC2 instances run here.
- Only ALB can access them.

Security Groups:

- ALB SG → allows inbound HTTP from anywhere.
- EC2 SG → allows inbound only from ALB SG.

Everything runs inside the **VPC created in Lab 17**.

🛠️Hands-On Lab Steps

Step 1: Create Project Folder

```
mkdir terraform-lab18-alb-asg
cd terraform-lab18-alb-asg
```

```
touch main.tf variables.tf outputs.tf
```

Step 2: Write Terraform Code (main.tf)

> **Note:** Replace `<PATH-TO-LAB17>` with the actual folder path where Lab 17 VPC outputs exist if you're importing them.
> For simplicity, we redefine the module call (same as Lab 17) so this lab is self-contained.

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

# ------------------------------------------------------
# 1. Reuse the VPC Module from Lab 17 (Self-contained)
# ------------------------------------------------------
module "vpc" {
  source  = "terraform-aws-modules/vpc/aws"
  version = "5.1.2"

  name = "lab18-prod-vpc"
  cidr = "10.1.0.0/16"

  azs = ["ap-south-1a", "ap-south-1b", "ap-south-1c"]

  public_subnets       = ["10.1.1.0/24", "10.1.2.0/24", "10.1.3.0/24"]
  private_subnets      = ["10.1.11.0/24", "10.1.12.0/24", "10.1.13.0/24"]
  database_subnets     = ["10.1.21.0/24", "10.1.22.0/24", "10.1.23.0/24"]

  enable_nat_gateway   = true
  single_nat_gateway   = true
}

# ------------------------------------------------------
# 2. Security Groups
# ------------------------------------------------------
resource "aws_security_group" "alb_sg" {
  name   = "lab18-alb-sg"
  vpc_id = module.vpc.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"]
  }
}

resource "aws_security_group" "ec2_sg" {
  name   = "lab18-ec2-sg"
  vpc_id = module.vpc.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"]
  }
}

# ----------------------------------------------------
# 3. Launch Template (for ASG)
# ----------------------------------------------------
resource "aws_launch_template" "lab18_lt" {
  name_prefix   = "lab18-lt-"
  image_id      = "ami-052c08d70def0ac62"
  instance_type = "t2.micro"

  vpc_security_group_ids = [aws_security_group.ec2_sg.id]

  # Simple web server
  user_data = base64encode(<<EOF
#!/bin/bash
yum install -y httpd
```

```
systemctl enable httpd
systemctl start httpd
echo "<h1>Welcome to Lab18 - Served from AutoScaling</h1>" > /var/www/html/
index.html
EOF
  )
}

# ---------------------------------------------------
# 4. Target Group
# ---------------------------------------------------
resource "aws_lb_target_group" "lab18_tg" {
  name     = "lab18-tg"
  port     = 80
  protocol = "HTTP"
  vpc_id   = module.vpc.vpc_id

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

# ---------------------------------------------------
# 5. Application Load Balancer
# ---------------------------------------------------
resource "aws_lb" "lab18_alb" {
  name               = "lab18-alb"
  load_balancer_type = "application"
  security_groups    = [aws_security_group.alb_sg.id]
  subnets            = module.vpc.public_subnets
}

# Listener
resource "aws_lb_listener" "lab18_listener" {
  load_balancer_arn = aws_lb.lab18_alb.arn
  port              = 80
  protocol          = "HTTP"

  default_action {
    type             = "forward"
    target_group_arn = aws_lb_target_group.lab18_tg.arn
  }
}

# ---------------------------------------------------
# 6. Auto Scaling Group
# ---------------------------------------------------
```

```
resource "aws_autoscaling_group" "lab18_asg" {
  name                      = "lab18-asg"
  max_size                  = 4
  min_size                  = 1
  desired_capacity           = 2

  vpc_zone_identifier = module.vpc.private_subnets

  launch_template {
    id      = aws_launch_template.lab18_lt.id
    version = "$Latest"
  }

  target_group_arns = [aws_lb_target_group.lab18_tg.arn]

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

Step 3: Initialize Terraform

```
terraform init
```

Step 4: Review the Plan

```
terraform plan
```

You should see Terraform creating: - VPC - ALB - Target Group - Launch Template - Auto Scaling Group - Security Groups

---

Step 5: Apply

```
terraform apply
```

Type **yes**.

Step 6: Validate in AWS Console

🔧Go to EC2 → Auto Scaling Groups

- You should see **lab18-asg** with 2 running instances.

🔧Go to EC2 → Load Balancers

- ALB should be "active".

🔧Visit ALB DNS Name

Open browser:

```
http://<ALB-DNS-NAME>
```

You should see:

> **Welcome to Lab18 - Served from AutoScaling**