Assignment 08

Objective

Design and deploy a scalable web application infrastructure on AWS using Terraform or CloudFormation (Terraform is recommended). This assignment covers provisioning a VPC, configuring subnets, deploying an Application Load Balancer (ALB), setting up security groups, implementing Auto Scaling, and deploying an RDS instance for database requirements.

Prerequisites

Use the AWS academy Lab account. Do not use your personal AWS account.

Assignment Tasks

Step 0: Initial Setup

1. Attach the **AdministratorAccess** policy to the LabRole IAM role.
2. Remove any custom policies that are currently attached to LabRole (e.g., c147213a3805520l8971812t1w7111).
3. Launch an Ubuntu instance and assign the LabRole IAM role to the instance.
4. Install **Terraform** and the **AWS CLI** on the instance.

Step 1: Networking Configuration

5. **Create a VPC**
    a. **CIDR Block**: 10.0.0.0/16
    b. **Name**: webapp-vpc
    c. **Screenshot**: Capture the VPC creation summary page showing the VPC ID, name, and CIDR block (Screenshot #1).
6. **Create Subnets**
    a. **Public Subnets**:
        i. Subnet 1: CIDR 10.0.1.0/24, Availability Zone: us-east-1a
        ii. Subnet 2: CIDR 10.0.2.0/24, Availability Zone: us-east-1b
        iii. Subnet 3: CIDR 10.0.3.0/24, Availability Zone: us-east-1c
    b. **Private Subnets**:

        i. Subnet 4: CIDR 10.0.4.0/24, Availability Zone: us-east-1a
        ii. Subnet 4: CIDR 10.0.5.0/24, Availability Zone: us-east-1b
    c. **Screenshot**: Capture subnet configurations, including CIDR blocks and availability zones (Screenshot #2).

7. **Internet Gateway (IGW) and Route Tables**
    a. Attach an IGW to the VPC created above.
    b. Configure Route Tables:
        i. **Public Route Table**: Associate with public subnets and add a route for 0.0.0.0/0 to the IGW.
        ii. **Private Route Table**: Associate with private subnets.
    c. **Screenshot**: Show IGW attachment and route table configurations (Screenshot #3).

Step 2: Security Groups

1. **Security Group for ALB**
    a. Inbound: Allow HTTP traffic on port 80 from anywhere (0.0.0.0/0).
    b. Outbound: Allow all traffic (0.0.0.0/0).
    c. **Screenshot**: Capture ALB security group rules for both inbound and outbound traffic (Screenshot #4).
2. **Security Group for EC2 Instances (Web Application Servers)**
    a. Inbound: Allow HTTP traffic on port 8080 from the ALB's security group.
    b. Outbound: Allow all traffic (0.0.0.0/0).
    c. **Screenshot**: Show EC2 instances' security group settings (Screenshot #5).
3. **Security Group for RDS Instance**
    a. Inbound: Allow traffic from the Web Application EC2 instances' security group on the database port (3306 for MySQL or 5432 for PostgreSQL).
    b. **Screenshot**: Show RDS security group settings (Screenshot #6).

Step 3: Provision RDS Instance for the Web Application

- **RDS Instance Configuration**:
    - Engine: MySQL or PostgreSQL
    - Engine Version: Latest stable version (e.g., '16' for PostgreSQL)
    - Instance Class: db.t3.micro
    - Database Name: webapp_db

  - Username/Password: Set secure credentials
  - Subnets: Place in private subnets created in Step 1
  - Public Access: Disabled
  - Final DB snapshot creation should be disabled
  - Parameter Group: Use a predefined parameter group for the chosen engine and version. (eg., postgres16). Note, I had to set custom parameter “rds.force_ssl” to 0 for RDS to accept connections from web app.
  - RDS Security Group: Allow inbound traffic only from the Web Application EC2 instances' security group on the database port.
- **Screenshot**: RDS instance configuration summary, showing instance class, engine, and network settings (Screenshot #7).

Step 4: EC2 Instances Running the Web Application

1. **Create a Launch Template (or Launch Configuration)**
   a. AMI: “ami-03a6c16a66bbe0a7a”
   b. Instance Type: t3.micro
   c. **User Data Script Requirements**:
      i. Webapp environment file:
      Generate a .env file (/usr/bin/csye6225/.env) with the values:
      DB_HOST, DB_PORT, DB_USERNAME, DB_PASSWORD, DB_NAME.
      Note: Ensure that actual values are retrieved dynamically from the RDS aws_db_instance terraform or cloudformation resource.
      ii. Set secure file permissions to restrict access:
      sudo chmod 600 /usr/bin/csye6225/.env
      sudo chown csye6225:csye6225 /usr/bin/csye6225/.env
      iii. Start webapp service.
      sudo systemctl start webapp.service

   d. **Screenshot**: Launch Template configuration page, including the instance type and user data script (Screenshot #8).

Note: The AMI provided for this assignment is a Go web API application deployed as a systemd service which will run on port 8080. This service setup enables automatic startup and management of the application process on each instance.

Step 5: Application Load Balancer (ALB) Configuration

1. **Create an ALB**
    a. Name: webapp-alb
    b. Scheme: internet-facing
    c. Type: application
    d. Subnets: Associate with public subnets created in Step 1
    e. Security Group: Attach the ALB security group created in Step 2
2. **Load Balancer Listener**
    a. Listener on Port 80: Forward HTTP traffic to the target group.
3. **Create Target Group**
    a. Name: webapp-target-group
    b. Protocol: HTTP, Port: 8080
    c. VPC: Select the VPC created in Step 1
    d. Health Checks: Protocol: HTTP, Path: /healthz
    e. **Screenshot**: Target group settings, including health check configurations (Screenshot #9).

Step 6: Implement Auto Scaling

1. **Create Auto Scaling Group (ASG)**
    a. Min Size: 1, Max Size: 3, Desired Capacity: 1
    b. Subnets: Public subnets from Step 1
    c. Health Check Type: ELB
    d. Associate ASG with Launch Template and Target Group created in Step 4 and Step 5.
    e. **Screenshot**: ASG settings, including instance counts (Screenshot #10).
2. **Define CloudWatch Alarms and Auto Scaling Policies**
    a. **Scale Up Policy**: Add 1 instance when CPU utilization > 70%.
    b. **Scale Down Policy**: Remove 1 instance when CPU utilization < 30%.
    c. **Screenshot**: CloudWatch alarm configurations for scaling policies (Screenshot #11).

Step 7: Testing and Validation

1. **Auto Scaling Behavior**

   a. Perform load testing on the Get User endpoint (GET http://<alb-dns>/v1/user/self).
   b. Monitor ASG for instance scaling behavior as traffic increases.
   c. **Screenshot**: Scaling up and down of instances in the ASG during load testing (Screenshot #12).
2. **Verify Application Functionality**
   a. Test Create User and Get User APIs.
   b. **Screenshot**: Request and response for each API call (Screenshot #13).

API Endpoints for Go Web Application

1. **Health Check**
   a. **Method**: GET, **URL**: http://<alb-dns>/healthz
2. **Create User**
   a. **Method**: POST, **URL**: http://<alb-dns>/v1/user
   b. Sample Request Body:
```
{
  "email": "sample.email@example.com",
  "password": "samplePassword123",
  "first_name": "Sample",
  "last_name": "User"
}
```
3. **Get User Details**
   a. **Method**: GET, **URL**: http://<alb-dns>/v1/user/self
   b. Authentication: Basic Auth with the created user's username and password.

Submission

- Submit a PDF including all screenshots with titles and descriptions for each, via Canvas.
- File Name: Use your last name.

Grading

- No late assignments are accepted.

- **PDF Report**: 3 points per required screenshot (13 screenshots - max 40 points).
- **Code**: 40 points for successfully provisioning resources.
- **Maximum**: 80 points (40 for report + 40 for code).