Evvo Assignment

Name: Antony Jebinraj Y
Role: DevOps Engineer
Email: jebin1848@gmail.com

Docker issue solved:

Dockerfile issue is python2 required for npm packages and I changed base image alpine to node:14 for warning packages.

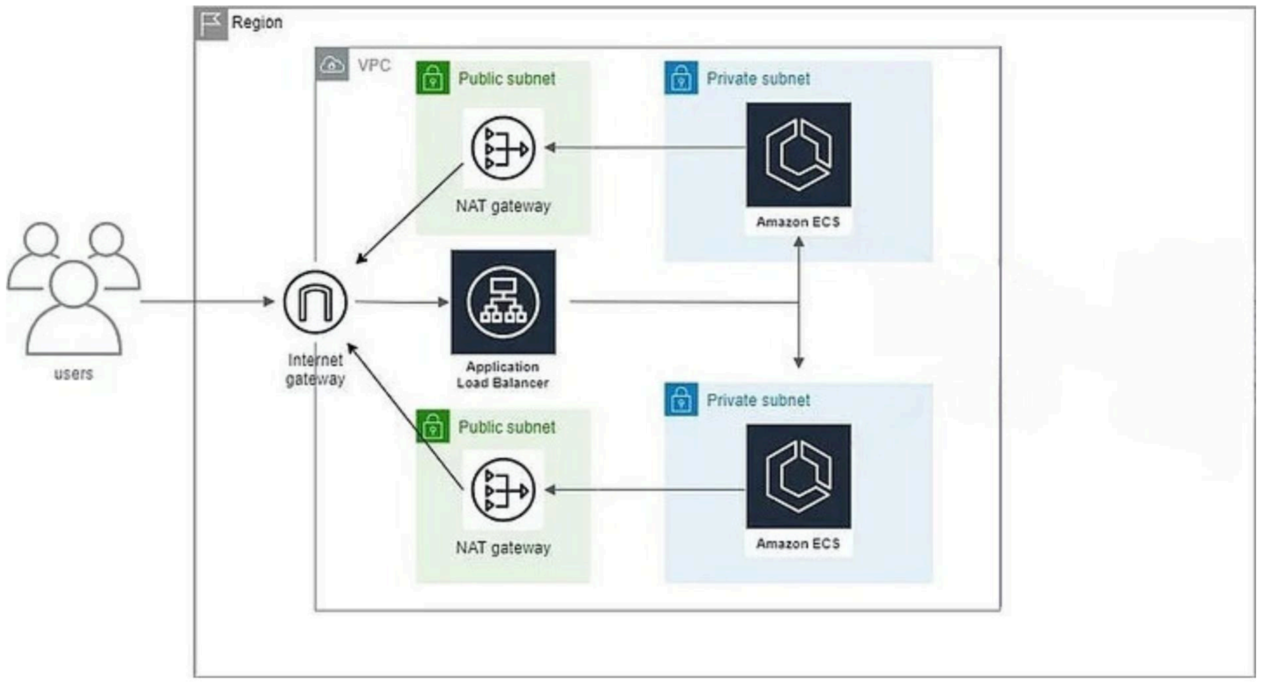
```
# Stage 1: Build dependencies using node:14 as base image
FROM node:14 as dependencies

# Install Python 2 (required for some npm packages)
RUN apt-get update && apt-get install -y python2
RUN ln -s /usr/bin/python2 /usr/local/bin/python

# Copy package.json and install dependencies
COPY package.json .
RUN npm install

# Stage 2: Build the final image using node:14
FROM node:14

# Set metadata labels for Docker
LABEL org.label-schema.schema-version="1.0"
LABEL org.label-schema.docker.cmd="docker run -d -p 3000:3000 --name
alpine_timeoff"

# Switch to root user to create a non-privileged user
USER root
RUN adduser --system app --home /app

# Switch back to non-privileged user
USER app

# Set the working directory inside the container
WORKDIR /app

# Copy application code from the build context to /app directory
```

```
COPY . /app

# Copy node_modules from the 'dependencies' stage
COPY --from=dependencies /node_modules ./node_modules

# Expose port 3000 to allow external access to the application
EXPOSE 3000

# Command to run the application when the container starts
CMD npm start
```

These are a few services and tools that will be used in this project:

1. Firstly AWS,
2. ECS — Containerization service in AWS
3. DynamoDB — NoSQL database service in AWS
4. ECR — A Container repository containing docker images
5. ALB — Application Load Balancer
6. Terraform — IaC(Infrastructure as Code) tool
7. Docker — Containerization platform
8. CICD — Github action.

The architecture



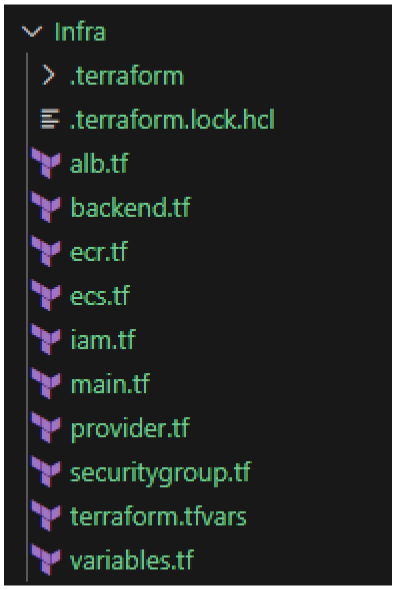
Terraform

Templates for ECS(fargate) and ALB pattern

To used this project,

1. AWS account, with access keys for a deployment role.
2. Docker installed on local machine.
3. Terraform installed on local machine
4. AWS CLI Installed

Folder structure of the directory



```
Infra
  .terraform
  .terraform.lock.hcl
  alb.tf
  backend.tf
  ecr.tf
  ecs.tf
  iam.tf
  main.tf
  provider.tf
  securitygroup.tf
  terraform.tfvars
  variables.tf
```

ECS

In the ECS file I define the cluster, the ECS service and the task definition.

src code for ecs.tf: (https://github.com/antonyjebin/evvo-alb-ecr-ecs/blob/main/Infra/ecs.tf)

- Creates an ECS cluster named `test-ecs-cluster`.

- Defines a task named `test-family with a container named `test-container`.

- Uses an image from an ECR repository.

- Sets port mappings and health check for the container.

- Specifies Fargate launch type with required configurations.

- Creates an ECS service named `test-ecs-service` in the previously defined cluster.

- Uses the task definition created earlier.

- Launch type is Fargate with 2 desired tasks.

- Specifies network configuration, assigning subnets and security groups.

- Registers tasks to a target group of the load balancer.

- Ensures the service depends on the load balancer listener..

This setup provisions an ECS cluster, defines a task using Fargate, and creates a service that runs two instances of the task, all configured to work within specified subnets and security groups, and balanced by a load balancer.

ECR

ECR for stored our docker images.

src code for ecr.tf: (https://github.com/UzairY/AWS-ALB-ECS/blob/main/ecr.tf)

- Create ECR repo.
- Apply lifecycle policy to manage images.
- Define and store commands for Docker operations.
- Login to ECR.
- Build Docker image and Tag Docker image.
- Push Docker image to the ECR repository.

VPC

VPC for isolated env of our application.

src code for main.tf: (https://github.com/UzairY/AWS-ALB-ECS/blob/main/main.tf)

- Creates a VPC with a specified CIDR block, DNS hostnames, and DNS support enabled.

- Attaches an Internet Gateway to the VPC for public internet access.

- Creates a NAT Gateway in a public subnet using the Elastic IP, dependent on the Internet Gateway.

- Creates two private subnets in specified availability zones.

- Creates two public subnets with automatic public IP mapping on instance launch.

- Creates a public route table with a route to the Internet Gateway and associates it with both public subnets.

- Creates a private route table with a route to the NAT Gateway and associates it with both private subnets.

This Terraform configuration sets up a VPC with both public and private subnets, an Internet Gateway, a NAT Gateway, and appropriate routing to allow public and private network communications.

ALB

src code for alb.tf: (https://github.com/UzairY/AWS-ALB-ECS/blob/main/alb.tf)

name: Specifies the name of the ALB.

internal: Set to `false` indicating the ALB is public.

load_balancer_type: Specifies the type as `application` for ALB.

subnets: Lists the IDs of the public subnets where the ALB will be deployed.

security_groups: Specifies the security group for the ALB

name: Specifies the name of the target group.

port: Uses a variable for the container port.

protocol: Sets the protocol to `HTTP`.

target_type: Specifies the target type as `ip`.

vpc_id: Specifies the VPC ID where the target group will be created.

Health_check: Configures health check settings:

path: Health check path `/health`.

protocol: Sets the protocol for health check to `HTTP`.

matcher: Expecting a `200` status code for healthy responses.

port: Uses `traffic-port`, which means the health check uses the port defined above.

healthy_threshold: Number of successful checks before considering the target healthy (2).

unhealthy_threshold: Number of failed checks before considering the target unhealthy (2).

timeout: Time to wait for a response (10 seconds).

interval: Time between health checks (30 seconds).

load_balancer_arn: Specifies the ARN of the ALB to attach the listener to.

port: Sets the listener to listen on port `80`.

protocol: Uses the `HTTP` protocol.

Default_action: Specifies the default action:

type: Action type is `forward`.

target_group_arn: ARN of the target group to forward requests to.

This configuration sets up an ALB, configures a target group with health checks, and defines an HTTP listener that forwards requests to the target group.

IAM

IAM (Identity and Access Management) roles are used to give resources and services permissions and restriction.

src code for iam.tf: (https://github.com/antonyjebin/evvo-alb-ecr-ecs/blob/main/Infra/iam.tf)

1. ECS Task Execution Role (`aws_iam_role.ecsTaskExecutionRole`)

Name: Defines the name of the IAM role as `test-app-ecsTaskExecutionRole`.

Assume Role Policy: Specifies the policy document that allows the ECS service (`ecs-tasks.amazonaws.com`) to assume this role (`sts:AssumeRole`).

2. Assume Role Policy Document (`data.aws_iam_policy_document.assume_role_policy`):

Actions: Specifies the action `sts:AssumeRole`, which allows entities (in this case, ECS tasks) to assume the role.

Principals: Defines that the entity assuming the role is a service (`Service` type) with the identifier `ecs-tasks.amazonaws.com`.

3. ECS Task Execution Role Policy Attachment (`aws_iam_role_policy_attachment.ecsTaskExecutionRole_policy`):

Role: Specifies the IAM role (`aws_iam_role.ecsTaskExecutionRole.name`) to attach the policy to.

Policy ARN: Attaches the AWS managed policy `AmazonECSTaskExecutionRolePolicy`, which grants necessary permissions for ECS task execution.

4. ECS Task Role (`aws_iam_role.ecsTaskRole`):

Name: Defines the name of the IAM role as `ecsTaskRole`.

Assume Role Policy: Specifies the same assume role policy document (`data.aws_iam_policy_document.assume_role_policy.json`) as the ECS task execution role.

5. ECS Task Role Policy Attachment (`aws_iam_role_policy_attachment.ecsTaskRole_policy`):

- Role: Specifies the IAM role (`aws_iam_role.ecsTaskRole.name`) to attach the policy to.

- Policy ARN: Attaches the AWS managed policy `AmazonDynamoDBFullAccess`, which grants full access to Amazon DynamoDB.

These resources together define two IAM roles: one (`ecsTaskExecutionRole`) for ECS task execution and the other (`ecsTaskRole`) for ECS task. The assume role policies allow ECS to assume these roles for their respective tasks and permissions.

src code for securitygroup.tf

(https://github.com/antonyjebin/evvo-alb-ecr-ecs/blob/main/Infra/securitygroup.tf)

1. ECS Security Group (`aws_security_group.ecs_sg`):

   - Defines a security group named "demo-sg-ecs" for the ECS application.

   - Specifies it belongs to the VPC identified by `aws_vpc.vpc.id`.

   - Automatically revokes all rules when the security group is deleted (`revoke_rules_on_delete = true`).

2. ECS Security Group Inbound Rule (`aws_security_group_rule.ecs_alb_ingress`):

   - Allows inbound traffic to the ECS security group from the ALB security group (`aws_security_group.alb_sg.id`).

   - Allows all traffic (`protocol = "-1"`, `from_port = 0`, `to_port = 0`).

   - Description explains it allows inbound traffic from ALB.

3. ECS Security Group Outbound Rule (`aws_security_group_rule.ecs_all_egress`):

   - Allows all outbound traffic (`protocol = "-1"`, `from_port = 0`, `to_port = 0`) from the ECS security group.

   - Allows traffic to any destination (`cidr_blocks = ["0.0.0.0/0"]`).

   - Description explains it allows outbound traffic from ECS.

4. ALB Security Group (`aws_security_group.alb_sg`):

   - Defines a security group named "demo-sg-alb" for the ALB.

   - Specifies it belongs to the same VPC (`aws_vpc.vpc.id`).

- Automatically revokes all rules when the security group is deleted.

5. ALB Security Group Inbound HTTP Rule (`aws_security_group_rule.alb_http_ingress`):

- Allows inbound HTTP traffic (`protocol = "TCP"`, `from_port = 80`, `to_port = 80`) from any source (`cidr_blocks = ["0.0.0.0/0"]`).

- Description explains it allows HTTP traffic from the internet to the ALB.

6. ALB Security Group Inbound HTTPS Rule (`aws_security_group_rule.alb_https_ingress`):

- Allows inbound HTTPS traffic (`protocol = "TCP"`, `from_port = 443`, `to_port = 443`) from any source (`cidr_blocks = ["0.0.0.0/0"]`).

- Description explains it allows HTTPS traffic from the internet to the ALB.

7. ALB Security Group Outbound Rule (`aws_security_group_rule.alb_egress`):

- Allows all outbound traffic (`protocol = "-1"`, `from_port = 0`, `to_port = 0`) from the ALB security group.

- Allows traffic to any destination (`cidr_blocks = ["0.0.0.0/0"]`).

- Description explains it allows outbound traffic from the ALB.

These Terraform resources set up security groups to control traffic to and from an ECS application and an ALB within an AWS VPC, ensuring that necessary ports are open for communication while maintaining security by limiting inbound traffic to specific ports and sources.

Variables

src code for variable.tf:

(https://github.com/antonyjebin/evvo-alb-ecr-ecs/blob/main/Infra/variables.tf)

1. region: Specifies the main AWS region where all resources will be deployed.

2. vpc_cidr: Defines the CIDR block (network range) for the main Virtual Private Cloud (VPC) in AWS.

3. public_subnet_1 and public_subnet_2: CIDR blocks for two public subnets within the VPC, typically used for resources that need direct internet access.

4. private_subnet_1 and private_subnet_2: CIDR blocks for two private subnets within the VPC, usually used for resources that should not have direct internet access.

5. availibility_zone_1 and availibility_zone_2: Names of two Availability Zones (AZs) where AWS resources can be deployed to achieve high availability and fault tolerance.

6. default_tags: Default tags applied to resources like EC2 instances or load balancers, indicating metadata such as application name ("Demo App") and environment ("Dev").

7. container_port: Specifies the port number that needs to be exposed for the application running in containers (e.g., Docker containers on ECS or EKS).

8. credential_profile: Name of the AWS credential profile in your local AWS credentials file, used for authentication and authorization when interacting with AWS services programmatically.

terraform.tfvars

(https://github.com/antonyjebin/evvo-alb-ecr-ecs/blob/main/Infra/terraform.tfvars)

1. Region: Set the AWS region where your infrastructure will be deployed. In this case, it's `eu-west-1` (Ireland).

2. VPC CIDR: Define the CIDR block for your Virtual Private Cloud (VPC). It's set to `10.0.0.0/16`, providing a range of IP addresses for your resources.

3. Public Subnets: Define two public subnets where your resources can have public internet access:

   - Subnet 1: `10.0.16.0/20` in `eu-west-1a`.

   - Subnet 2: `10.0.32.0/20` in `eu-west-1b`.

4. Private Subnets: Define two private subnets where resources won't have direct internet access:

- Subnet 1: `10.0.80.0/20` in `eu-west-1a`.

- Subnet 2: `10.0.112.0/20` in `eu-west-1b`.

5. Availability Zones: Specify the availability zones (`eu-west-1a` and `eu-west-1b`) where your subnets and resources will be distributed for high availability.

6. Container Port: Set the port (`3000`) on which your application inside containers will listen for incoming traffic.

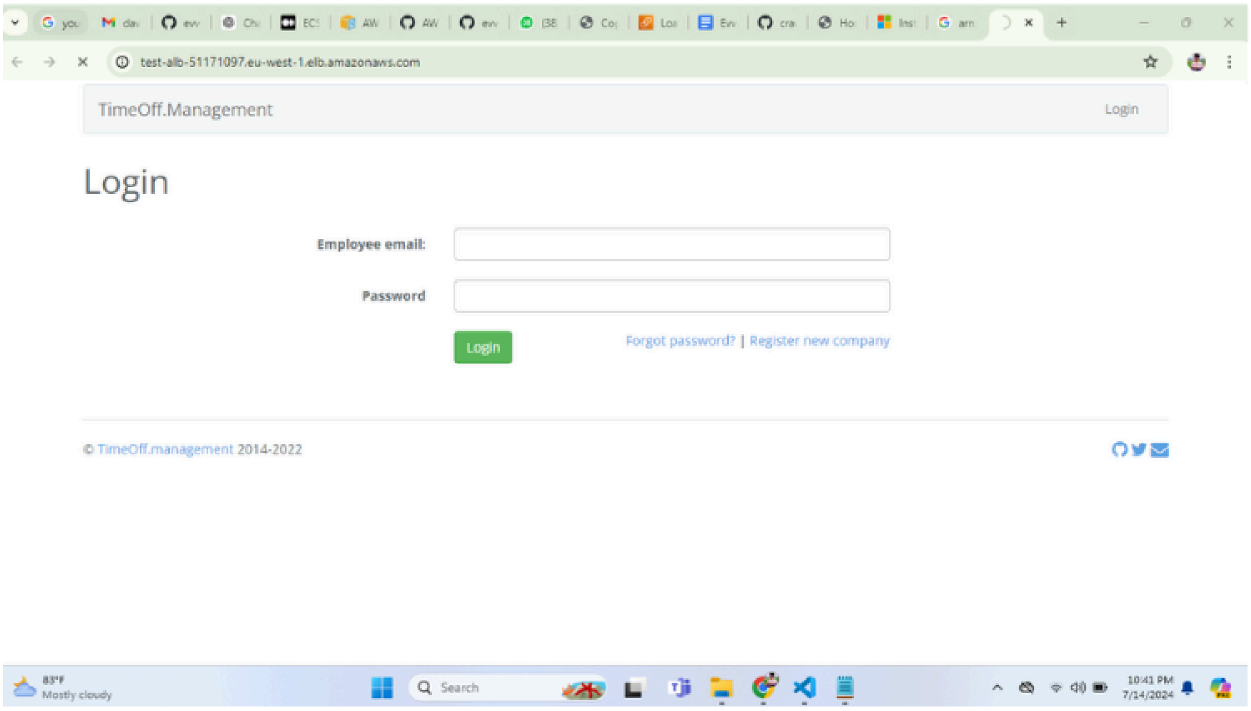
7. Credential Profile: Specify the AWS CLI profile (`evvo`) that contains the credentials to authenticate and authorize your Terraform scripts to interact with AWS.

Deployment

Steps to use:

1. Clone the repo to your desired location
2. Configured the terraform.tfvars file with my desired values
3. Run "terraform init", "terraform plan" and "terraform apply"
4. To access the application, navigate to the Load Balancer in the AWS console and paste the dns name of the Load Balancer in a new tab.

Accessing the app


test-alb-51171097.eu-west-1.elb.amazonaws.com

TimeOff.Management

Login

Employee email:

Password

© TimeOff.management 2014-2022

Notes: I cant buy domain. If I have domain and next level is:

- Add an SSL certificate for HTTPS requests
- Security groups can be more secure
- IAM roles can be more restrictive
- ECR can also use a VPC endpoint to pull images

CI/CD Pipelines for Auto Deployment:

main.yml

```
name: Deploy to ECS

on:
  push:
    branches:
      - main  # adjust branch name if needed

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
```

```
        uses: actions/checkout@v2

      - name: Configure AWS Credentials
        uses: aws-actions/configure-aws-credentials@v1
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: ${{ secrets.AWS_REGION }}  # replace with your AWS
region stored in secrets

      - name: Login to AWS ECR
        run: |
          aws ecr get-login-password --region eu-west-1 --profile personal
| docker login --username AWS --password-stdin
918822628996.dkr.ecr.eu-west-1.amazonaws.com

      - name: Build Docker image
        run: |
          docker build -t my-test-repo ../

      - name: Tag Docker image
        run: |
          docker tag my-test-repo:latest
918822628996.dkr.ecr.eu-west-1.amazonaws.com/my-test-repo:latest

      - name: Push Docker image to ECR
        run: |
          docker push
918822628996.dkr.ecr.eu-west-1.amazonaws.com/my-test-repo:latest

      - name: Update ECS Service
        run: |
          aws ecs update-service --cluster test-ecs-cluster --service
test-ecs-service --force-new-deployment
```