NodeJS Application on ECS EC2 using GitHub Workflows Pipeline using Terraform (Task 13)

Cloudelligent

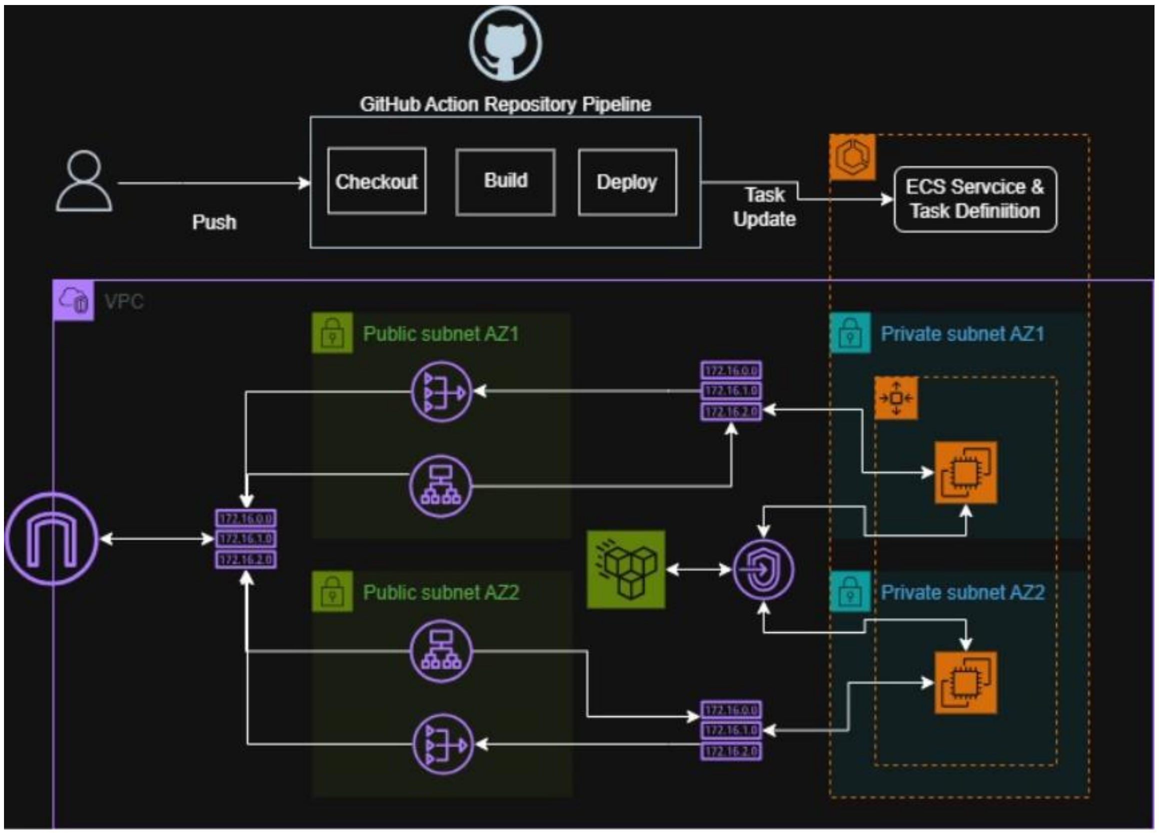
Zaeem Attique Ashar

Cloud Intern

Task Description:

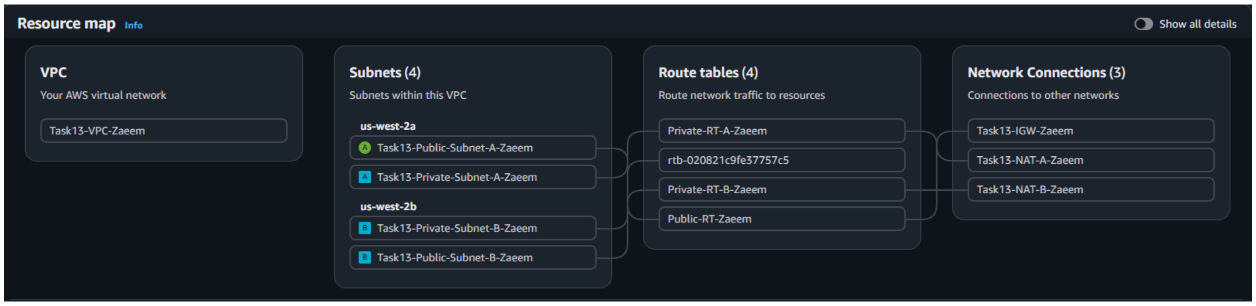
This project involves deploying a simple Node.js application using AWS Elastic Container Service. The application will be turned into a Docker Image using a Docker File stored in the repository. This image will be pushed to the ECR. An ECS EC2 Cluster and Task will be setup to run the application. After pulling the image from the ECR, it will be deployed on the ECS EC2 Cluster. The infrastructure will be written in terraform, and the github workflow will use that terraform code to deploy it on ECS.

Architecture Diagram:



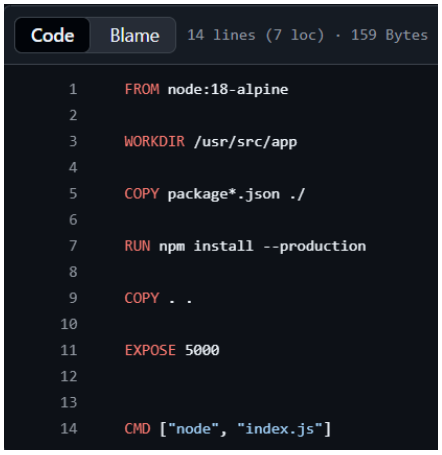
Task13.1: Create basic networking infrastructure

- Create and configure a VPC
  - CIDR Block: 10.0.0.0/16
- Create and configure Subnets
  - Public Subnet A (us-west-2a), CIDR: 10.0.1.0/24
  - Private Subnet A (us-west-2a), CIDR: 10.0.2.0/24
  - Public Subnet B (us-west-2b), CIDR: 10.0.3.0/24
  - Private Subnet A (us-west-2a), CIDR: 10.0.4.0/24
- Create and configure NAT Gateways
  - NAT Gateway A in Public Subnet A
  - NAT Gateway B in Public Subnet B
- Create and configure Internet Gateway
  - Create and attach to the project's VPC
- Create and configure Route Tables
  - Public Route Table, Outbound rule: 0.0.0.0/0 -> IGW, attach to Public SN A&B
  - Private Route Table A, Outbound Rule: 0.0.0.0/0 -> NGW attach to Private SN A
  - Private Route Table B, Outbound Rule: 0.0.0.0/0 -> NGW attach to Private SN B



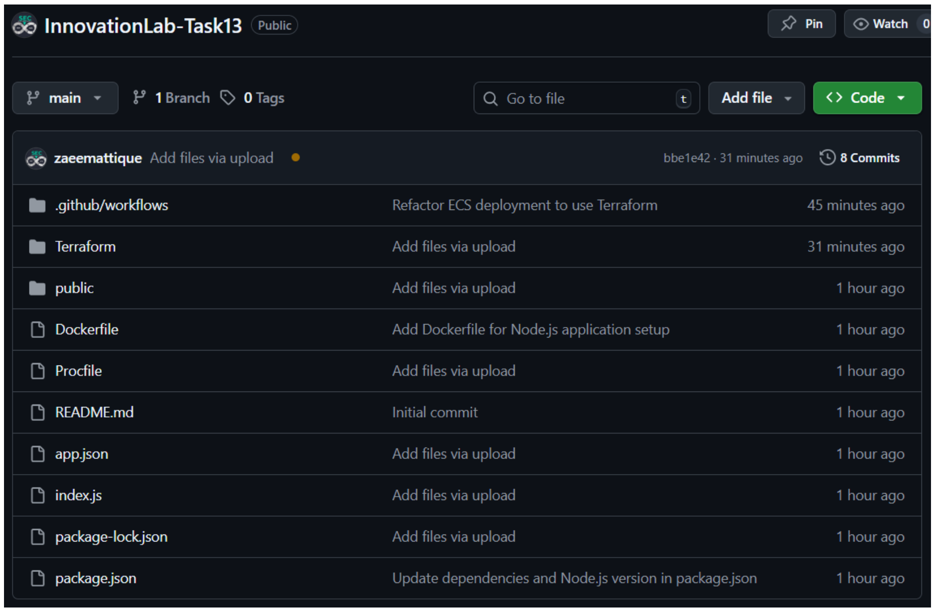
Task13.2: Prepare NodeJS Application and Dockerfile

- Remove node packages as they will be installed later in the build stage of the pipeline. Node packages are in the node_modules directory of the application.
- Write the Dockerfile which includes the steps to build the application. Here are the steps:
  - Uses node:18-alpine as the base image for the application image.
  - Set usr/src/app as the working directory.
  - Copy all the package file onto the node.
  - Use npm install to install all dependencies for the application.
  - Copy all the files from the app directory to the image.
  - Expose the container port 5000 for HTTP communication.
  - Use the "node" or "index.js" command to run the application.



```
FROM node:18-alpine

WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install --production

COPY . .

EXPOSE 5000


CMD ["node", "index.js"]
```

- Upload the rest of the application code to the GitHub repository for the dockerfile to use.

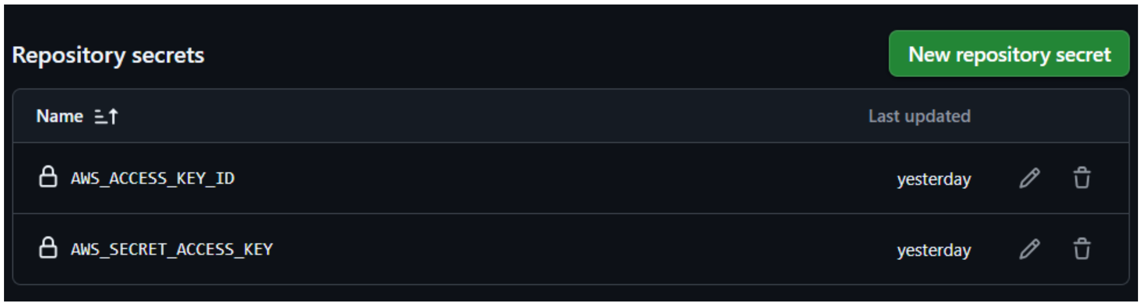


.github/workflows	Refactor ECS deployment to use Terraform	45 minutes ago
Terraform	Add files via upload	31 minutes ago
public	Add files via upload	1 hour ago
Dockerfile	Add Dockerfile for Node.js application setup	1 hour ago
Procfile	Add files via upload	1 hour ago
README.md	Initial commit	1 hour ago
app.json	Add files via upload	1 hour ago
index.js	Add files via upload	1 hour ago
package-lock.json	Add files via upload	1 hour ago
package.json	Update dependencies and Node.js version in package.json	1 hour ago

Task13.3: Configure AWS Access Keys as Secrets.

To safely store and access the AWS Access Keys to the use account and keep them safe from showing up in the logs, we must set them as secret variables in the GitHub repository. These keys can be used in the long term.

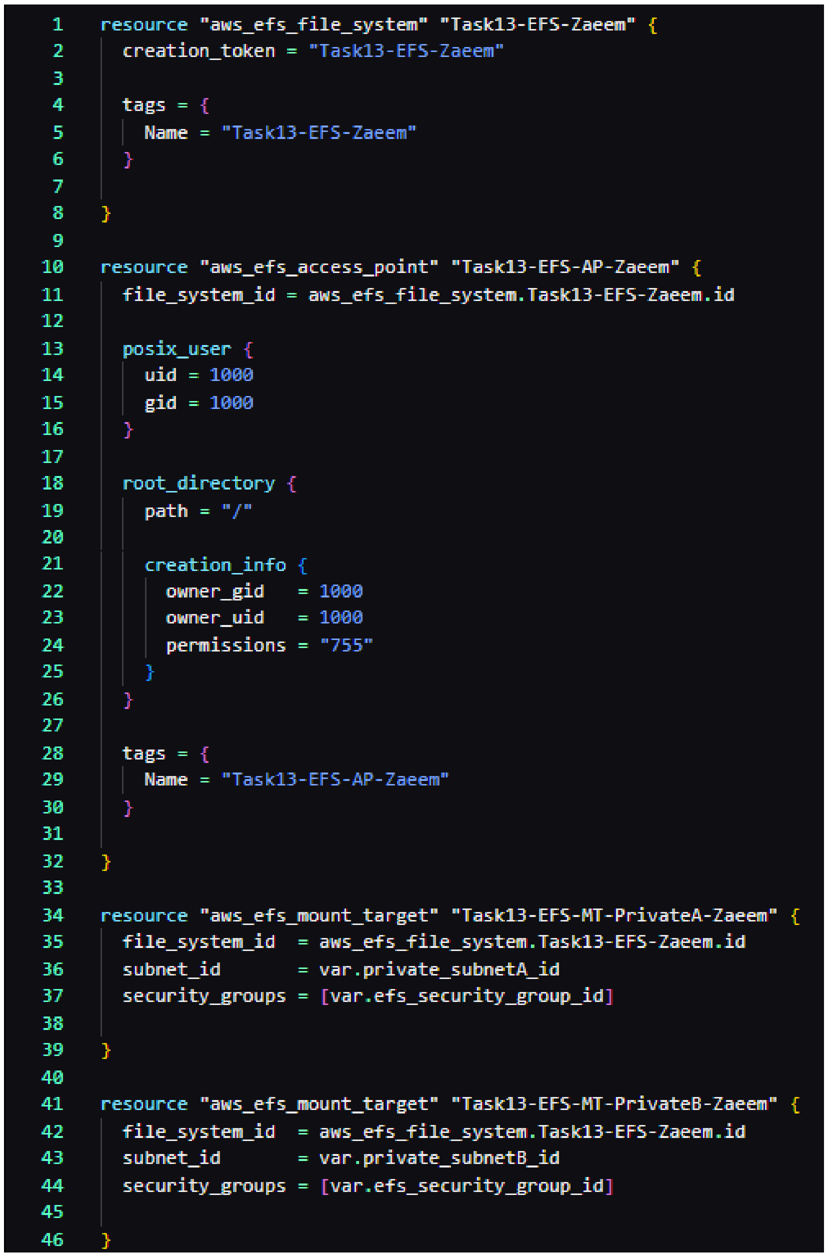
- Head over to the GitHub repository home page and navigate over to settings option on the top bar.
- Scroll down and find the Secrets and Variables option in the left panel and select Actions
- Scroll down and click on the 'New Repository Secret' button.
- Enter the name of the secret variable and then enter the secret value and save.
- Do the same for both the Access Key and the Secret Access Key.



Name	Last updated
AWS_ACCESS_KEY_ID	yesterday
AWS_SECRET_ACCESS_KEY	yesterday

Task13.4: Create EFS File System and Mount Points

- Create File System:
  - Name: Task13-EFS-Zaeem
  - Encryption: Yes
- Create Access Points:
  - File System: Task13-EFS-Zaeem
  - Name: Task13-EFS-AP-Zaeem
  - Root Directory: /
  - POSIX UID 1000, GID 1000
  - Owner UID 1000, Owner GID: 1000
  - AP Permissions: 0755
- Create Mount Points:
  - AZ us-west-2a, Subnet Private, SG Task13-EFS-SG-Zaeem
  - AZ us-west-2b, Subnet Private, SG Task13-EFS-SG-Zaeem



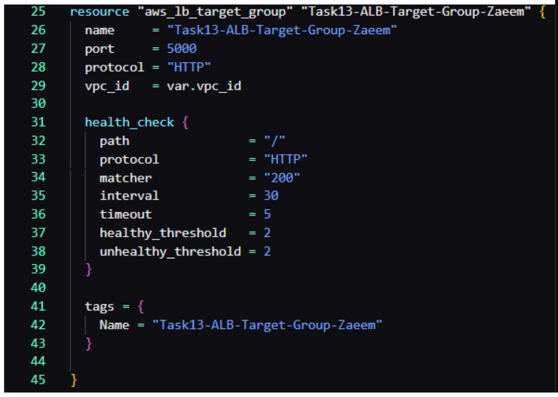
```
resource "aws_efs_file_system" "Task13-EFS-Zaeem" {
  creation_token = "Task13-EFS-Zaeem"

  tags = {
    Name = "Task13-EFS-Zaeem"
  }

}

resource "aws_efs_access_point" "Task13-EFS-AP-Zaeem" {
  file_system_id = aws_efs_file_system.Task13-EFS-Zaeem.id

  posix_user {
    uid = 1000
    gid = 1000
  }

  root_directory {
    path = "/"

    creation_info {
      owner_gid   = 1000
      owner_uid   = 1000
      permissions = "755"
    }
  }

  tags = {
    Name = "Task13-EFS-AP-Zaeem"
  }

}

resource "aws_efs_mount_target" "Task13-EFS-MT-PrivateA-Zaeem" {
  file_system_id  = aws_efs_file_system.Task13-EFS-Zaeem.id
  subnet_id       = var.private_subnetA_id
  security_groups = [var.efs_security_group_id]

}

resource "aws_efs_mount_target" "Task13-EFS-MT-PrivateB-Zaeem" {
  file_system_id  = aws_efs_file_system.Task13-EFS-Zaeem.id
  subnet_id       = var.private_subnetB_id
  security_groups = [var.efs_security_group_id]

}
```

Task13.5: Create Infrastructure for ECS Cluster

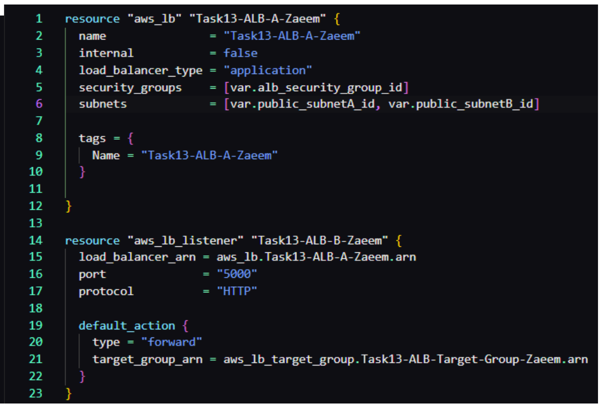
Now back to aws, we need to create an Application Load Balancer and a Target group that we will pass over to ECS when we create the service.

- Create and Configure Application Load Balancer and Target Group
    - Name: Task13-ALB-Zaeem
    - Scheme: Internet Facing
    - Load Balancer IP: IPv4
    - VPC: Task13-VPC-Zaeem

- AZ and Subnets: AZ1 Public SN & AZ2 Public SN
- Security Groups: Task13-ALB-SG-Zaeem
- Listener Protocol HTTP, Port 80
- Routing Action: Forward to TG
- Register Targets: None

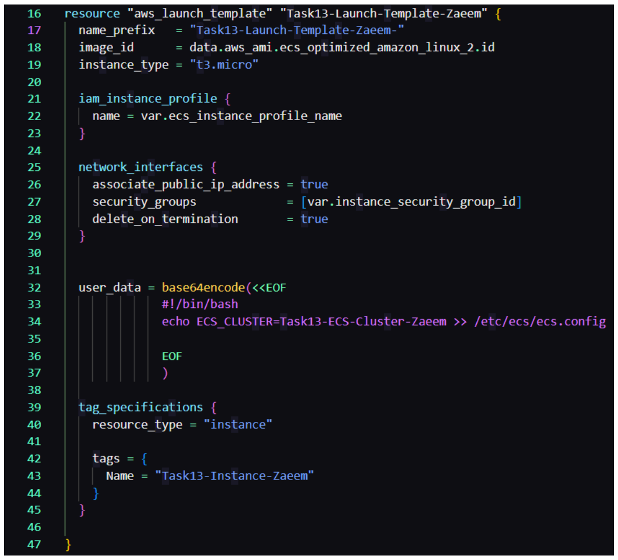


```
resource "aws_lb_target_group" "Task13-ALB-Target-Group-Zaeem" {
  name     = "Task13-ALB-Target-Group-Zaeem"
  port     = 5000
  protocol = "HTTP"
  vpc_id   = var.vpc_id

  health_check {
    path                = "/"
    protocol            = "HTTP"
    matcher             = "200"
    interval            = 30
    timeout             = 5
    healthy_threshold   = 2
    unhealthy_threshold = 2
  }

  tags = {
    Name = "Task13-ALB-Target-Group-Zaeem"
  }

}
```



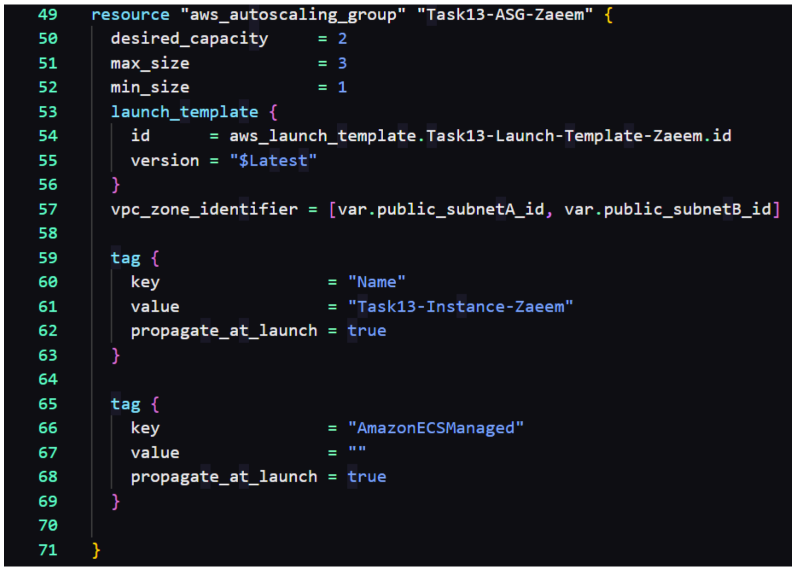
```
resource "aws_lb" "Task13-ALB-A-Zaeem" {
  name               = "Task13-ALB-A-Zaeem"
  internal           = false
  load_balancer_type = "application"
  security_groups    = [var.alb_security_group_id]
  subnets            = [var.public_subnetA_id, var.public_subnetB_id]

  tags = {
    Name = "Task13-ALB-A-Zaeem"
  }

}

resource "aws_lb_listener" "Task13-ALB-B-Zaeem" {
  load_balancer_arn = aws_lb.Task13-ALB-A-Zaeem.arn
  port              = "5000"
  protocol          = "HTTP"

  default_action {
    type = "forward"
    target_group_arn = aws_lb_target_group.Task13-ALB-Target-Group-Zaeem.arn
  }
}
```

- Create and Configure a Launch Template
    - Name: Task13-EC2-LT-Zaeem
    - Container instance AMI: Amazon Linux 2023
    - Instance Type: t3.micro
    - SSH Key pair: Task13-EC2
    - Subnet: Do not include
    - Availability Zone: Do not include
    - Security Group: Task13-EC2-SG-Zaeem
    - Storage Volume: 8GiB Default EBS volume



```
resource "aws_launch_template" "Task13-Launch-Template-Zaeem" {
  name_prefix   = "Task13-Launch-Template-Zaeem-"
  image_id      = data.aws_ami.ecs_optimized_amazon_linux_2.id
  instance_type = "t3.micro"

  iam_instance_profile {
    name = var.ecs_instance_profile_name
  }

  network_interfaces {
    associate_public_ip_address = true
    security_groups             = [var.instance_security_group_id]
    delete_on_termination       = true
  }


  user_data = base64encode(<<EOF
            #!/bin/bash
            echo ECS_CLUSTER=Task13-ECS-Cluster-Zaeem >> /etc/ecs/ecs.config

            EOF
            )

  tag_specifications {
    resource_type = "instance"

    tags = {
      Name = "Task13-Instance-Zaeem"
    }
  }

}
```

- Create and Configure Auto Scaling Group
    - Name: Task13-ASG-Zaeem
    - Launch Template: Task13-EC2-LT-Zaeem
    - VPC: Task13-VPC-Zaeem
    - AZ: Private SN A & Private SN B
    - Load Balancer: Attach to existing
    - Attach to an Existing Target Group
    - Group Size: Desired capacity 2, min 1, max 3
    - No scling policies



```
resource "aws_autoscaling_group" "Task13-ASG-Zaeem" {
  desired_capacity     = 2
  max_size             = 3
  min_size             = 1
  launch_template {
    id      = aws_launch_template.Task13-Launch-Template-Zaeem.id
    version = "$Latest"
  }
  vpc_zone_identifier = [var.public_subnetA_id, var.public_subnetB_id]

  tag {
    key                 = "Name"
    value               = "Task13-Instance-Zaeem"
    propagate_at_launch = true
  }

  tag {
    key                 = "AmazonECSManaged"
    value               = ""
    propagate_at_launch = true
  }

}
```

Task13.6: Create ECR Repository for the Docker Image

- Head over to the ECR Dashboard and click on create repository.
- Name: task13/zaeem
- Image Tag Mutability: Mutable
- Encryption Config: AES-256
- Click on the create button.

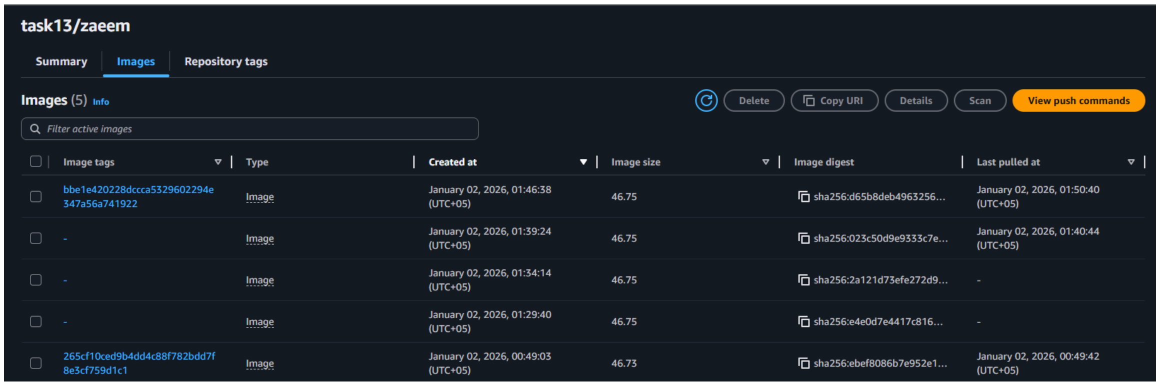
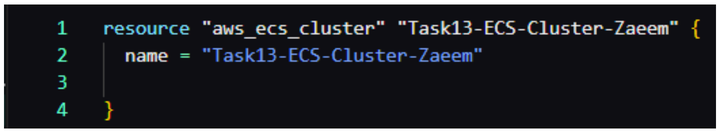


Image tags	Type	Created at	Image size	Image digest	Last pulled at
bbe1e420228dccca5329602294e347a56a741922	Image	January 02, 2026, 01:46:38 (UTC+05)	46.75	sha256:d65b8deb4963256...	January 02, 2026, 01:50:40 (UTC+05)
-	Image	January 02, 2026, 01:39:24 (UTC+05)	46.75	sha256:023c50d9e9333c7e...	January 02, 2026, 01:40:44 (UTC+05)
-	Image	January 02, 2026, 01:34:14 (UTC+05)	46.75	sha256:2a121d73efe272d9...	-
-	Image	January 02, 2026, 01:29:40 (UTC+05)	46.75	sha256:e4e0d7e4417c816...	-
265cf10ced9b4dd4c88f782bdd7f8e3cf759d1c1	Image	January 02, 2026, 00:49:03 (UTC+05)	46.73	sha256:ebef8086b7e952e1...	January 02, 2026, 00:49:42 (UTC+05)

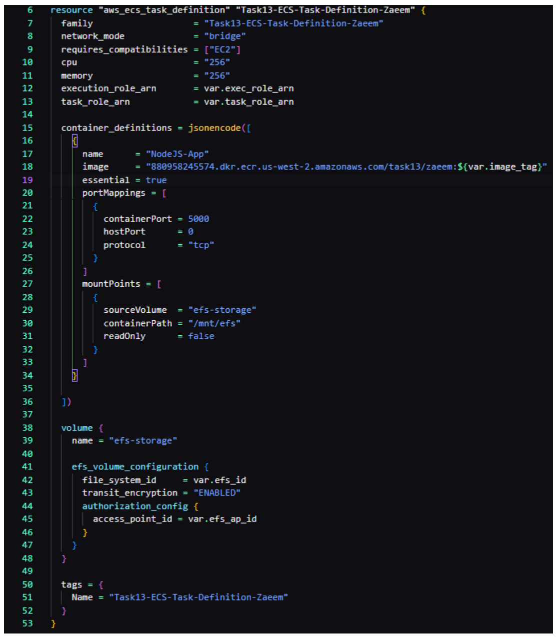
Task13.7: Create ECS Cluster, Task Definition and Service to deploy the application

- For the ECS Cluster configure the following:
  - Name: Task13-ECS-Cluster-Zaeem
  - Infrastructure: EC2 using capacity provider
  - Under monitoring, set the level of observability as container insights
  - Click on create



```
resource "aws_ecs_cluster" "Task13-ECS-Cluster-Zaeem" {
  name = "Task13-ECS-Cluster-Zaeem"

}
```

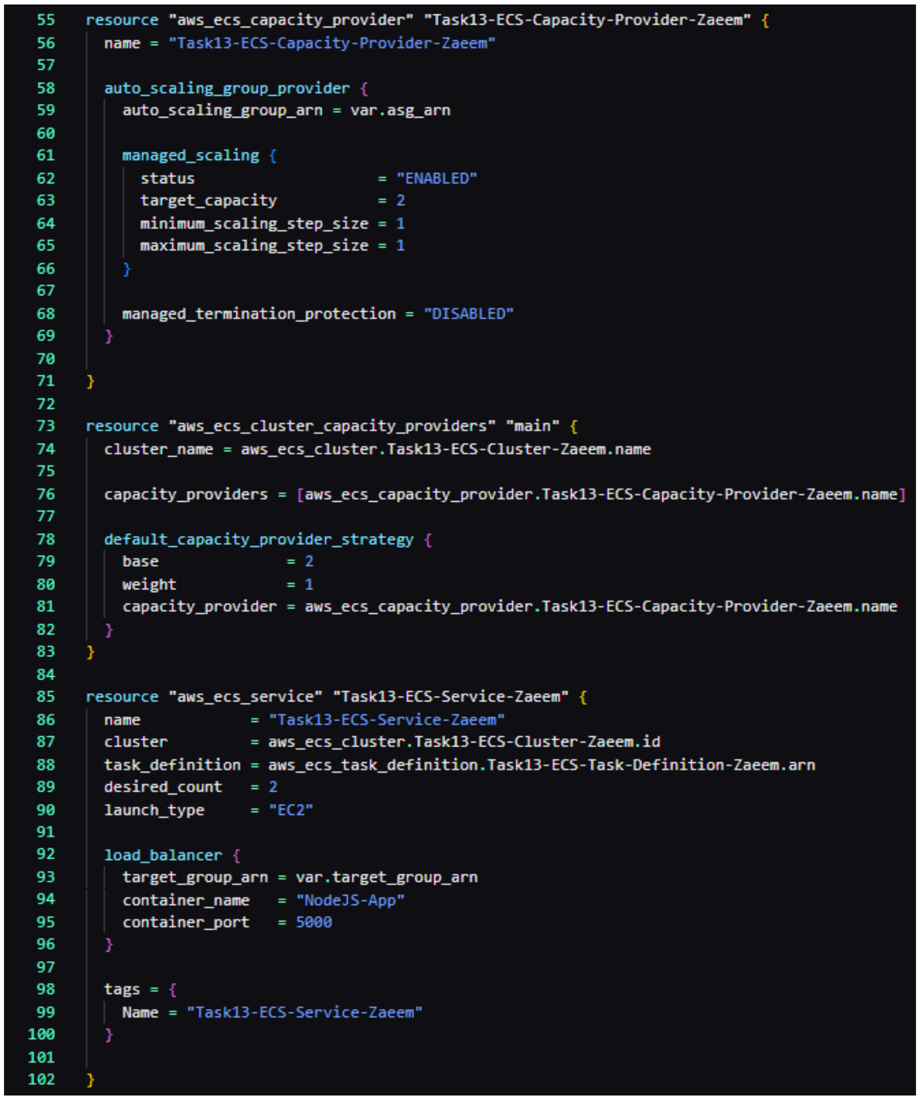
- For the Task definition configure the following:
    - Task definition family name: Task13-ECS-TD-Zaeem
    - Launch type: AWS ECS
    - OS Architecture: Default
    - Task Size: 2vCPU and 4GB Memory
    - Select task roles and execution roles.
    - Container name: NodeJS-App
    - Select Image URI from the created ECR repo
    - Port mapping: Container port 5000, Protocol TCP, Port name: App-port, HTTP
    - Click on Create



```
resource "aws_ecs_task_definition" "Task13-ECS-Task-Definition-Zaeem" {
  family                   = "Task13-ECS-Task-Definition-Zaeem"
  network_mode             = "bridge"
  requires_compatibilities = ["EC2"]
  cpu                      = "256"
  memory                   = "256"
  execution_role_arn       = var.exec_role_arn
  task_role_arn            = var.task_role_arn

  container_definitions = jsonencode([
    {
      name      = "NodeJS-App"
      image     = "880958245574.dkr.ecr.us-west-2.amazonaws.com/task13/zaeem:${var.image_tag}"
      essential = true
      portMappings = [
        {
          containerPort = 5000
          hostPort      = 0
          protocol      = "tcp"
        }
      ]
      mountPoints = [
        {
          sourceVolume  = "efs-storage"
          containerPath = "/mnt/efs"
          readOnly      = false
        }
      ]
    }

  ])

  volume {
    name = "efs-storage"

    efs_volume_configuration {
      file_system_id     = var.efs_id
      transit_encryption = "ENABLED"
      authorization_config {
        access_point_id = var.efs_ap_id
      }
    }
  }

  tags = {
    Name = "Task13-ECS-Task-Definition-Zaeem"
  }
}
```

- Now head over to the services panel in the cluster we created previously and click on the create service button.
- Configure the following:
    - Task definition family: Task13-ECS-TD-Zaeem
    - Compute options: Launch Type, EC2, Latest
    - Scheduling Strategy: Replica, Desired Tasks: 2
    - Deployment Option: Rolling Updates
    - Under load balancing: Load Balancer Type: Application
    - Select the container name from the drop-down menu.

- Use an existing Load Balancer and choose the previously created ALB.
- Use an existing Target Group and select the previously created TG.
- Click on create service.

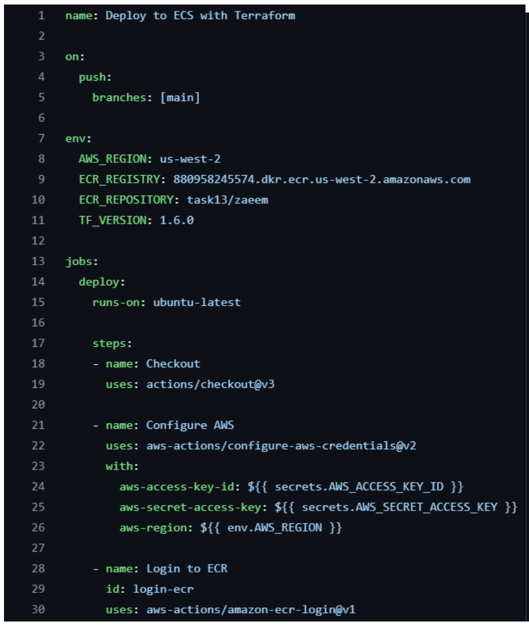


```
resource "aws_ecs_capacity_provider" "Task13-ECS-Capacity-Provider-Zaeem" {
  name = "Task13-ECS-Capacity-Provider-Zaeem"

  auto_scaling_group_provider {
    auto_scaling_group_arn = var.asg_arn

    managed_scaling {
      status                    = "ENABLED"
      target_capacity           = 2
      minimum_scaling_step_size = 1
      maximum_scaling_step_size = 1
    }

    managed_termination_protection = "DISABLED"
  }

}

resource "aws_ecs_cluster_capacity_providers" "main" {
  cluster_name = aws_ecs_cluster.Task13-ECS-Cluster-Zaeem.name

  capacity_providers = [aws_ecs_capacity_provider.Task13-ECS-Capacity-Provider-Zaeem.name]

  default_capacity_provider_strategy {
    base              = 2
    weight            = 1
    capacity_provider = aws_ecs_capacity_provider.Task13-ECS-Capacity-Provider-Zaeem.name
  }
}

resource "aws_ecs_service" "Task13-ECS-Service-Zaeem" {
  name            = "Task13-ECS-Service-Zaeem"
  cluster         = aws_ecs_cluster.Task13-ECS-Cluster-Zaeem.id
  task_definition = aws_ecs_task_definition.Task13-ECS-Task-Definition-Zaeem.arn
  desired_count   = 2
  launch_type     = "EC2"

  load_balancer {
    target_group_arn = var.target_group_arn
    container_name   = "NodeJS-App"
    container_port   = 5000
  }

  tags = {
    Name = "Task13-ECS-Service-Zaeem"
  }

}
```

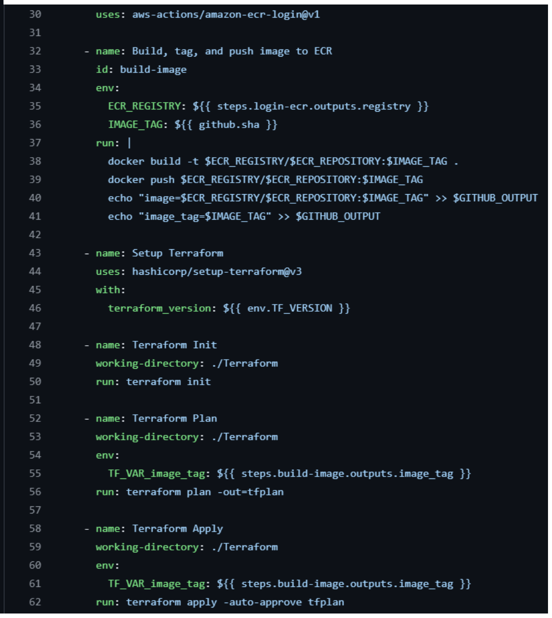
Task13.8: Creating GitHub Workflow

This will be a pipeline that will build and deploy the application code that we have uploaded into the repository by making use of the aws Credentials and the Dockerfile.

- Select the create new file button in the github repository and set the new file name as follows: '.github/workflows/deploy.yml '.
- Write the YAML code of the pipeline which includes the configurations:
  - Name: Deploy to ECS
  - Trigger on push to main branch of the repository
  - Set the following env variables:
    - AWS_REGION: us-west-2
    - ECR_REGISTRY: 880958245574.dkr.ecr.us-west-2.amazonaws.com
    - ECR_REPOSITORY: zaeem/task13
    - CONTAINER_NAME: NodeJS-App
  - Create only one job named 'Deploy' that runs on ubuntu-latest runner.
  - First step checks out code and copies it to the runner using checkout@v3
  - Second, use configure-aws-credentials@v2 to use the keys to config aws cli
  - Third, use amazon-ecr-login@v1 to login to the ECR to push the image later
  - The fourth stage uses the dockerfile to build the app image and push it to ECR and store the image tag in a local variable to use it in terraform.
  - Fifth stage uses setup-terraform@v3 action to configure Terraform on the runner
  - Sixth stage sets the working directory for terraform as ./Terraform and runs init
  - Sixth stage runs the terraform plan passing the image tag variable.
  - Seventh stage uses the terraform to apply command and deploy the update.



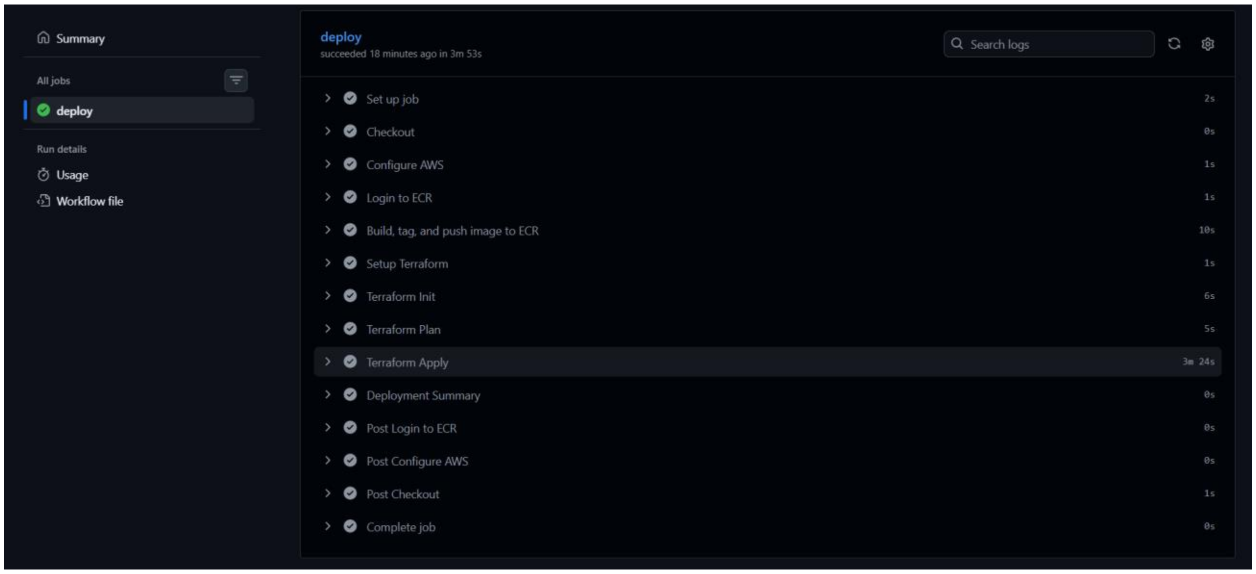
```yaml
name: Deploy to ECS with Terraform

on:
  push:
    branches: [main]

env:
  AWS_REGION: us-west-2
  ECR_REGISTRY: 880958245574.dkr.ecr.us-west-2.amazonaws.com
  ECR_REPOSITORY: task13/zaeem
  TF_VERSION: 1.6.0

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout
      uses: actions/checkout@v3

    - name: Configure AWS
      uses: aws-actions/configure-aws-credentials@v2
      with:
        aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
        aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
        aws-region: ${{ env.AWS_REGION }}

    - name: Login to ECR
      id: login-ecr
      uses: aws-actions/amazon-ecr-login@v1
```



```yaml
      uses: aws-actions/amazon-ecr-login@v1

    - name: Build, tag, and push image to ECR
      id: build-image
      env:
        ECR_REGISTRY: ${{ steps.login-ecr.outputs.registry }}
        IMAGE_TAG: ${{ github.sha }}
      run: |
        docker build -t $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG .
        docker push $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG
        echo "image=$ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG" >> $GITHUB_OUTPUT
        echo "image_tag=$IMAGE_TAG" >> $GITHUB_OUTPUT

    - name: Setup Terraform
      uses: hashicorp/setup-terraform@v3
      with:
        terraform_version: ${{ env.TF_VERSION }}

    - name: Terraform Init
      working-directory: ./Terraform
      run: terraform init

    - name: Terraform Plan
      working-directory: ./Terraform
      env:
        TF_VAR_image_tag: ${{ steps.build-image.outputs.image_tag }}
      run: terraform plan -out=tfplan

    - name: Terraform Apply
      working-directory: ./Terraform
      env:
        TF_VAR_image_tag: ${{ steps.build-image.outputs.image_tag }}
      run: terraform apply -auto-approve tfplan
```

Task13.9: Triggering the GitHub Workflow

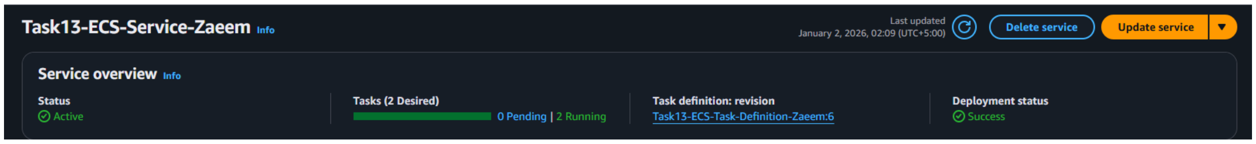
As we commit to the workflow file, a github workflow will run to implement the jobs and steps defined in the workflow file. It will output logs at each stage and output errors as well.



```
Summary                 deploy
                        succeeded 18 minutes ago in 3m 53s            Search logs
All jobs
✓ deploy                > ✓ Set up job                                 2s
                        > ✓ Checkout                                   0s
Run details             > ✓ Configure AWS                              1s
Usage                   > ✓ Login to ECR                               1s
Workflow file           > ✓ Build, tag, and push image to ECR         10s
                        > ✓ Setup Terraform                            1s
                        > ✓ Terraform Init                             6s
                        > ✓ Terraform Plan                             5s
                        > ✓ Terraform Apply                        3m 24s
                        > ✓ Deployment Summary                         0s
                        > ✓ Post Login to ECR                          0s
                        > ✓ Post Configure AWS                         0s
                        > ✓ Post Checkout                              1s
                        > ✓ Complete job                               0s
```

Task13.10: Testing the Deployment of the Application

- Head over to the cluster and open the service we created. There should be the desired number of tasks running that we mentioned in the Service Configuration. If not, the rolling updates have not been finished yet.



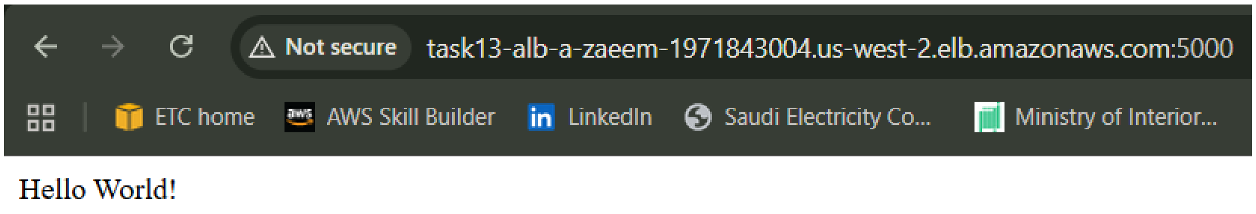
Task13-ECS-Service-Zaeem Info

Last updated January 2, 2026, 02:09 (UTC+5:00)

Service overview Info

Status	Tasks (2 Desired)	Task definition: revision	Deployment status
✓ Active	0 Pending | 2 Running	Task13-ECS-Task-Definition-Zaeem:6	✓ Success

- Head over to the load balancer tab and use the ALB DNS to communicate with the frontend of the application through the browser.



Not secure task13-alb-a-zaeem-1971843004.us-west-2.elb.amazonaws.com:5000

Hello World!

- There are two ways to rollback an update that is deployed:
    - Fastest: Head over to the cluster dash and click on the update service button. In the configuration, select the previous version of the task definition and also check the force deployment button. This will update the service instantly.
    - Cleanest: Roll back to an older commit/ fix the code and the workflow will be automatically triggered upon commit. This will create a new task definition version and deploy it onto the service.

Task13.11: Problems faced and Solutions Learned

- We can wither write the infrastructure how we wish as IaC and use it inside the workflow so that when a new deployment is created, it will be according to our defined infrastructure.
- Alternatively, we can avoid using IaC by creating all the infrastructure such as cluster, task definition and service, and pulling the config into a JSON inside the workflow, modifying it at each run and deploying using that as the IaC.
- The target group of the ALB for an ECS cluster does not have any targets registered initially because it will auto register targets as they are created by EC2.
- The package.json file must include the correct Node version and all the dependencies that are required for the application to run.