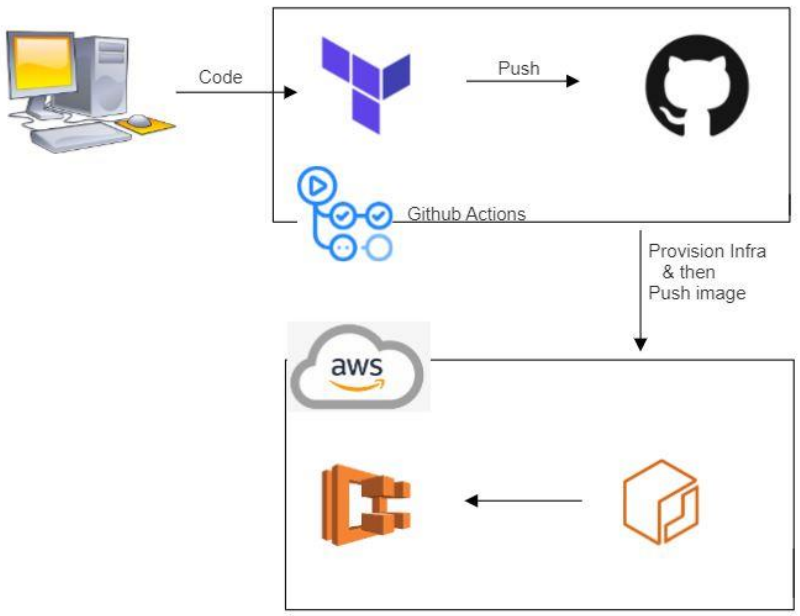
# Description of the services and CI/CD



**Amazon Elastic Container Registry** (Amazon ECR) is an AWS managed container image registry service that is secure, scalable, and reliable. Amazon ECR supports private repositories with resource-based permissions using AWS IAM. This is so that specified users or Amazon EC2 instances can access your container repositories and images. You can use your preferred CLI to push, pull, and manage Docker images, Open Container Initiative (OCI) images, and OCI compatible artifacts.

HashiCorp Terraform is **an infrastructure as code tool that lets you define both cloud and on-prem resources in human-readable configuration files that you can version, reuse, and share**. You can then use a consistent workflow to provision and manage all of your infrastructure throughout its lifecycle.

**Amazon Elastic Container Service** (Amazon ECS) is a fully managed container orchestration service that helps you easily deploy, manage, and scale containerized applications. As a fully managed service, Amazon ECS comes with AWS configuration and operational best practices built-in. This also means that you don't need to manage control plane, nodes, or add-ons. It's integrated with both AWS and third-party tools, such as Amazon Elastic Container Registry and Docker. This integration makes it easier for teams to focus on building the applications, not the environment. You can run and scale your container workloads across AWS Regions in the cloud, and on-premises, without the complexity of managing a control plane or nodes.

A serverless option with AWS Fargate. With AWS Fargate, you don't need to manage servers, handle capacity planning, or isolate container workloads for security. Fargate handles the infrastructure management aspects of your workload for you. You can schedule the placement of your containers across your cluster based on your resource needs, isolation policies, and availability requirements.

**GitHub Actions** is a continuous integration and continuous delivery (CI/CD) platform that allows you to automate your build, test, and deployment pipeline. You can create workflows that build and test every pull request to your repository, or deploy merged pull requests to production.GitHub Actions goes beyond just DevOps and lets you run workflows when other events happen in your repository. For example, you can run a workflow to automatically add the appropriate labels whenever someone creates a new issue in your repository.

**The components of GitHub Actions:**

You can configure a GitHub Actions *workflow* to be triggered when an *event* occurs in your repository, such as a pull request being opened or an issue being created. Your workflow contains one or more *jobs* which can run in sequential order or in parallel. Each job will run inside its own virtual machine *runner*, or inside a container, and has one or more *steps* that either run a script that you define or run an *action*, which is a reusable extension that can simplify your workflow.

**Workflows:** A workflow is a configurable automated process that will run one or more jobs. Workflows are defined by a YAML file checked in to your repository and will run when triggered by an event in your repository, or they can be triggered manually, or at a defined schedule.

**Events:** An event is a specific activity in a repository that triggers a workflow run. For example, activity can originate from GitHub when someone creates a pull request, opens an issue, or pushes a commit to a repository. You can also trigger a workflow to run on a schedule, by posting to a REST API, or manually.

**Jobs:** A job is a set of *steps* in a workflow that is executed on the same runner. Each step is either a shell script that will be executed, or an *action* that will be run. Steps are executed in order and are dependent on each other. Since each step is executed on the same runner, you can share data from one step to another. For example, you can have a step that builds your application followed by a step that tests the application that was built.

**Actions:** An *action* is a custom application for the GitHub Actions platform that performs a complex but frequently repeated task. Use an action to help reduce the amount of repetitive code that you write in your workflow files. An action can pull your git repository from GitHub, set up the correct toolchain for your build environment, or set up the authentication to your cloud provider.

**Runners:** A runner is a server that runs your workflows when they're triggered. Each runner can run a single job at a time. GitHub provides Ubuntu Linux, Microsoft Windows, and macOS runners to run your workflows; each workflow run executes in a fresh, newly-provisioned virtual machine. GitHub also offers larger runners, which are available in larger configurations.

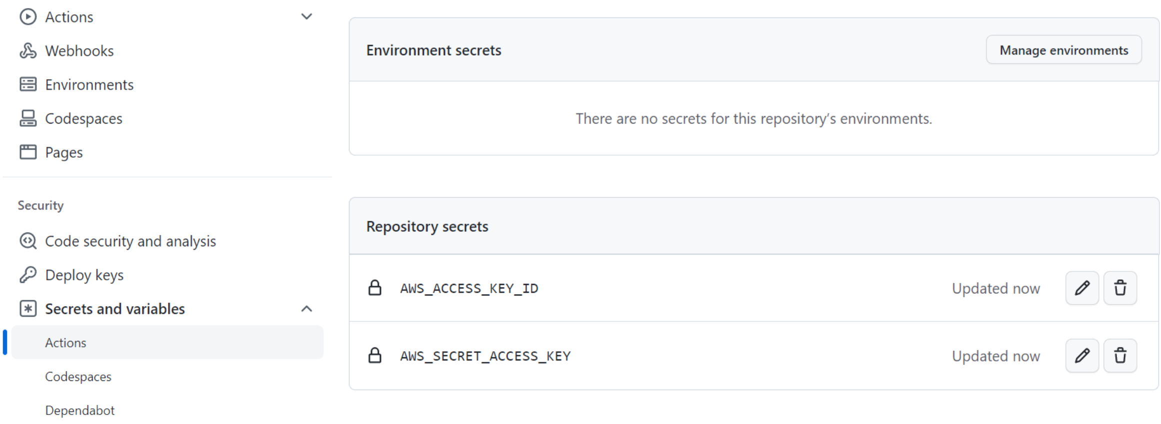
In recent times, most of the organizations have been migrating their monolithic applications to microservices which enables organizations to optimize resources, enhance collaboration, and streamline business processes. As part of the microservices architecture, each service owns its docker repository. Also, as the number of services increases, there will be a requirement to create a new repository for each service. In this example, we use Terraform to provision the infrastructure (ECS +ECR ) . We also create the docker container and push to ecr  and then manage the container on eks to deploy to the end user.

## Procedure for the task:

## Step 1: Create a github repository (say ecr-ecs-gha-ter)

After creating the github repository , Set the $SECRET used in github action workflow file .

Create the secret name as `AWS_ACCESS_KEY_ID` , `AWS_SECRET_ACCESS_KEY` to configure with aws cloud.



Environment secrets

There are no secrets for this repository's environments.

Repository secrets

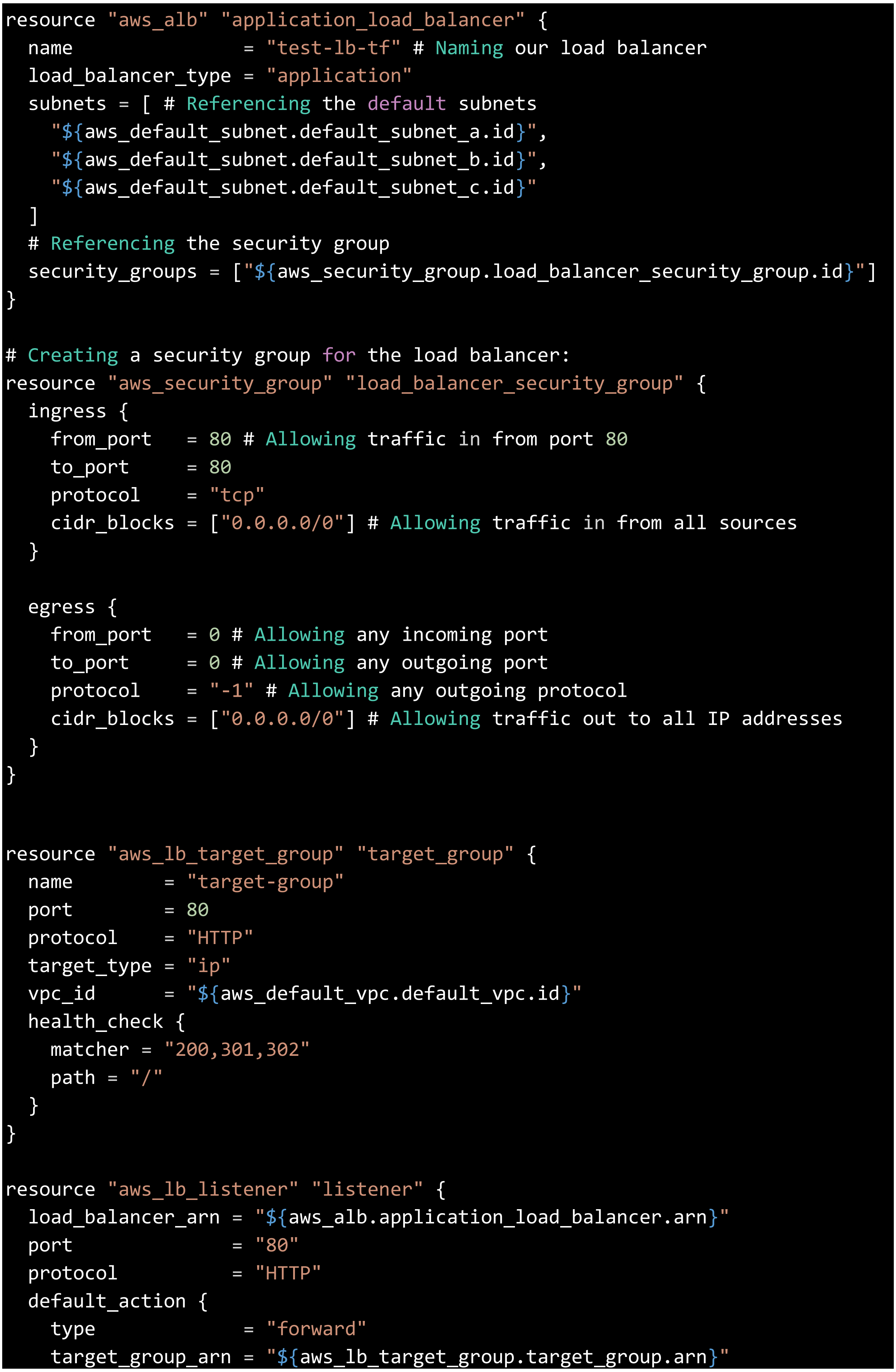
AWS_ACCESS_KEY_ID Updated now

AWS_SECRET_ACCESS_KEY Updated now

## Step 2: Create the terraform code to provision the ECR & ECS :

```
provider "aws" {
  version = "~> 4.53.0"
  region  = "us-east-1"
}

resource "aws_ecr_repository" "my_first_ecr_repo" {
 name = "my-first-ecr-repo"
}
```

```
resource "aws_ecr_repository_policy" "demo-repo-policy" {
  repository = aws_ecr_repository.my_first_ecr_repo.name
  policy     = <<EOF
  {
    "Version": "2008-10-17",
    "Statement": [
      {
        "Sid": "Set the permission for ECR",
        "Effect": "Allow",
        "Principal": "*",
        "Action": [
          "ecr:BatchCheckLayerAvailability",
          "ecr:BatchGetImage",
          "ecr:CompleteLayerUpload",
          "ecr:GetDownloadUrlForLayer",
          "ecr:GetLifecyclePolicy",
          "ecr:InitiateLayerUpload",
          "ecr:PutImage",
          "ecr:UploadLayerPart"
        ]
      }
    ]
  }
  EOF
}

resource "aws_ecs_cluster" "my_cluster" {
  name = "my-cluster"
}

resource "aws_ecs_task_definition" "my_first_task" {
  family                   = "my-first-task" # Naming our first task
  container_definitions    = <<DEFINITION
  [
    {
      "name": "my-first-task",
      "image": "${aws_ecr_repository.my_first_ecr_repo.repository_url}",
      "essential": true,
      "portMappings": [
        {
          "containerPort": 3100,
          "hostPort": 3100
        }
      ],
      "memory": 512,
      "cpu": 256
    }
  ]
  DEFINITION
  requires_compatibilities = ["FARGATE"]
  network_mode             = "awsvpc"
  memory                   = 512
  cpu                      = 256
  execution_role_arn       = "${aws_iam_role.ecsTaskExecutionRole.arn}"
}

resource "aws_iam_role" "ecsTaskExecutionRole" {
  name               = "ecsTaskExecutionRole"
  assume_role_policy =
"${data.aws_iam_policy_document.assume_role_policy.json}"
}

data "aws_iam_policy_document" "assume_role_policy" {
  statement {
    actions = ["sts:AssumeRole"]

    principals {
      type        = "Service"
      identifiers = ["ecs-tasks.amazonaws.com"]
    }
  }
}

resource "aws_iam_role_policy_attachment" "ecsTaskExecutionRole_policy" {
  role       = "${aws_iam_role.ecsTaskExecutionRole.name}"
  policy_arn = "arn:aws:iam::aws:policy/service-
role/AmazonECSTaskExecutionRolePolicy"
}


# Providing a reference to our default VPC
resource "aws_default_vpc" "default_vpc" {
}

# Providing a reference to our default subnets
resource "aws_default_subnet" "default_subnet_a" {
  availability_zone = "us-east-1a"
}

resource "aws_default_subnet" "default_subnet_b" {
  availability_zone = "us-east-1b"
}

resource "aws_default_subnet" "default_subnet_c" {
  availability_zone = "us-east-1c"
}
```



```
resource "aws_alb" "application_load_balancer" {
  name               = "test-lb-tf" # Naming our load balancer
  load_balancer_type = "application"
  subnets = [ # Referencing the default subnets
    "${aws_default_subnet.default_subnet_a.id}",
    "${aws_default_subnet.default_subnet_b.id}",
    "${aws_default_subnet.default_subnet_c.id}"
  ]
  # Referencing the security group
  security_groups = ["${aws_security_group.load_balancer_security_group.id}"]
}

# Creating a security group for the load balancer:
resource "aws_security_group" "load_balancer_security_group" {
  ingress {
    from_port   = 80 # Allowing traffic in from port 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"] # Allowing traffic in from all sources
  }

  egress {
    from_port   = 0 # Allowing any incoming port
    to_port     = 0 # Allowing any outgoing port
    protocol    = "-1" # Allowing any outgoing protocol
    cidr_blocks = ["0.0.0.0/0"] # Allowing traffic out to all IP addresses
  }
}


resource "aws_lb_target_group" "target_group" {
  name        = "target-group"
  port        = 80
  protocol    = "HTTP"
  target_type = "ip"
  vpc_id      = "${aws_default_vpc.default_vpc.id}"
  health_check {
    matcher = "200,301,302"
    path = "/"
  }
}

resource "aws_lb_listener" "listener" {
  load_balancer_arn = "${aws_alb.application_load_balancer.arn}"
  port              = "80"
  protocol          = "HTTP"
  default_action {
    type             = "forward"
    target_group_arn = "${aws_lb_target_group.target_group.arn}"
  }
}


resource "aws_ecs_service" "my_first_service" {
  name            = "my-first-service"
  cluster         = "${aws_ecs_cluster.my_cluster.id}"
  task_definition = "${aws_ecs_task_definition.my_first_task.arn}"
  launch_type     = "FARGATE"
  desired_count   = 3 # Setting the number of containers to 3

  load_balancer {
    target_group_arn = "${aws_lb_target_group.target_group.arn}"
    container_name   = "${aws_ecs_task_definition.my_first_task.family}"
    container_port   = 3100 # Specifying the container port
  }

  network_configuration {
    subnets          = ["${aws_default_subnet.default_subnet_a.id}",
"${aws_default_subnet.default_subnet_b.id}",
"${aws_default_subnet.default_subnet_c.id}"]
    assign_public_ip = true                                          #
Providing our containers with public IPs
    security_groups  = ["${aws_security_group.service_security_group.id}"] #
Setting the security group
  }
}


resource "aws_security_group" "service_security_group" {
  ingress {
    from_port = 0
    to_port   = 0
    protocol  = "-1"
    # Only allowing traffic in from the load balancer security group
    security_groups =
["${aws_security_group.load_balancer_security_group.id}"]
  }

  egress {
    from_port   = 0 # Allowing any incoming port
    to_port     = 0 # Allowing any outgoing port
    protocol    = "-1" # Allowing any outgoing protocol
    cidr_blocks = ["0.0.0.0/0"] # Allowing traffic out to all IP addresses
  }
}
```

### Step 3: Create a github action workflow file to create the ECR and ECS on aws cloud and build the image using Dockerfile and push to ECR & deploying it to ECS,when pushing the code to main branch.

```
name: Node.js CI

on: [push]
jobs:
  build:

    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3
    - name: Use Node.js
      uses: actions/setup-node@v3
      with:
        node-version: '12.x'
    - name: Install dependencies
      run: npm install
    - run: npm init --y
    - run: npm install express

    - name: Checkout Repo
      uses: actions/checkout@v2

    - name: Terraform Setup
      uses: hashicorp/setup-terraform@v1

    - name: Terraform Init
      run: terraform init
      env:
        GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
        TF_ACTION_WORKING_DIR: '.'
        AWS_ACCESS_KEY_ID:  ${{ secrets.AWS_ACCESS_KEY_ID }}
        AWS_SECRET_ACCESS_KEY:  ${{ secrets.AWS_SECRET_ACCESS_KEY }}

    - name: Terraform validate
      run: terraform validate

    - name: Terraform Apply
      run: terraform apply -auto-approve
      env:
        GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
        TF_ACTION_WORKING_DIR: '.'
        AWS_ACCESS_KEY_ID:  ${{ secrets.AWS_ACCESS_KEY_ID }}
        AWS_SECRET_ACCESS_KEY:  ${{ secrets.AWS_SECRET_ACCESS_KEY }}

    - name: Check out code
```

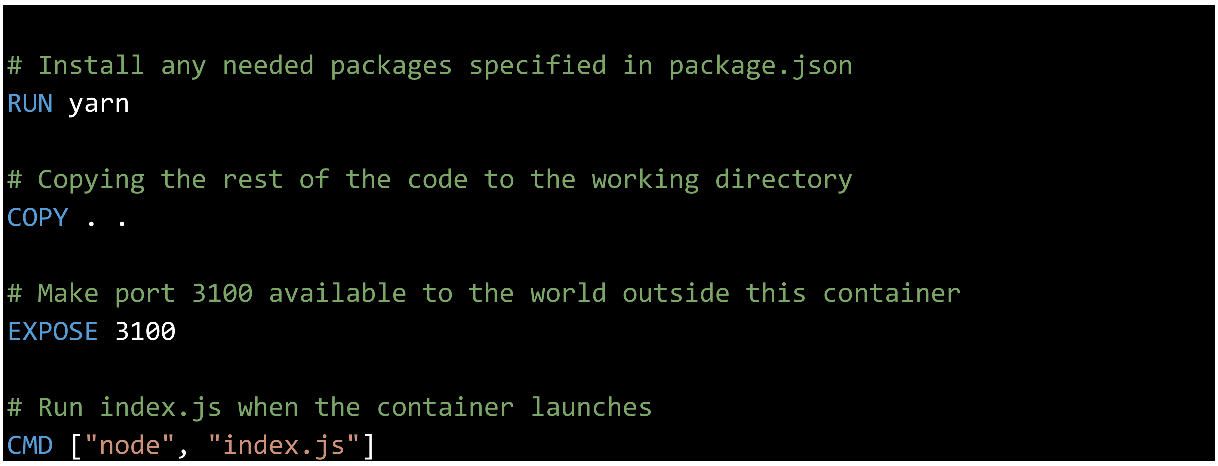
```
      uses: actions/checkout@v2

    - 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: us-east-1

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

    - name: Build, tag, and push image to Amazon ECR
      env:
        ECR_REGISTRY: ${{ steps.login-ecr.outputs.registry }}
        ECR_REPOSITORY: my-first-ecr-repo
        IMAGE_TAG: docker_image
      run: |
        docker build -t $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG .
        docker push $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG
        echo "::set-output name=image::$ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG"
```

### Step 4: Creating the nodejs app

```
const express = require('express')
const app = express()
const port = 3100

app.get('/', (req, res) => res.send('Simple APP with Terraform!'))

app.listen(port, () => console.log(`Example app listening on port ${port}!`))
```

### Step 5: Create the DOCKERFILE to create the docker image of the nodejs app.

```
# Use an official Node runtime as a parent image
FROM node:12.7.0-alpine

# Set the working directory to /app
WORKDIR '/app'

# Copy package.json to the working directory
COPY package.json .
```



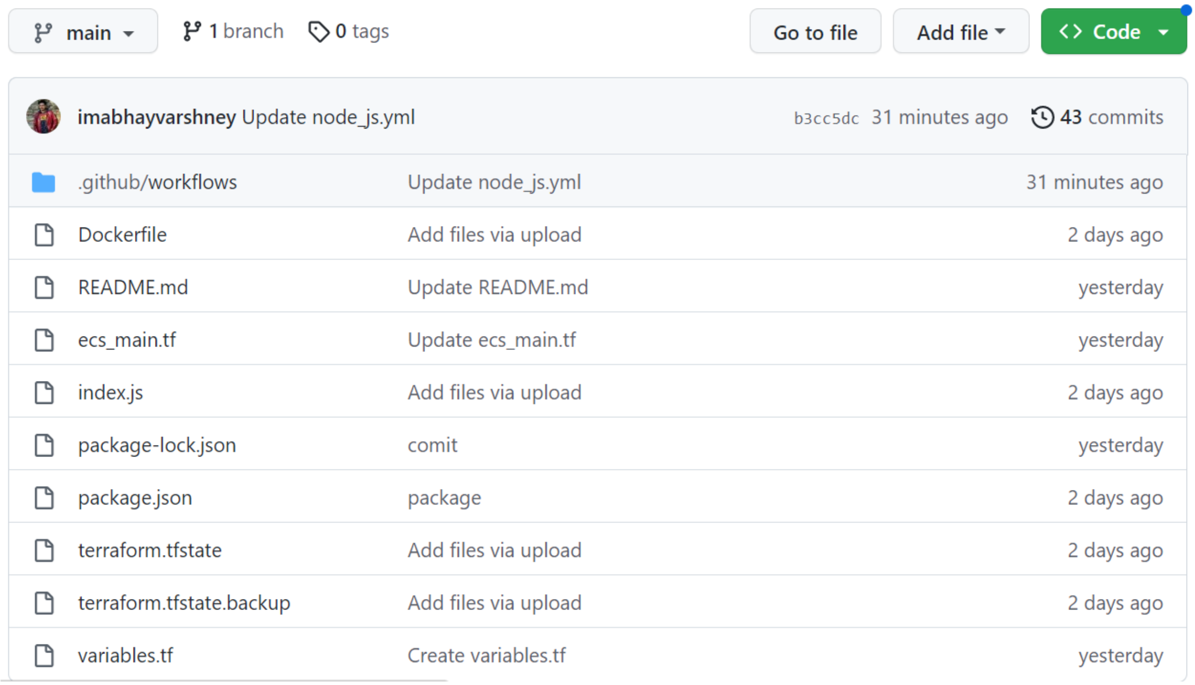
```
# Install any needed packages specified in package.json
RUN yarn

# Copying the rest of the code to the working directory
COPY . .

# Make port 3100 available to the world outside this container
EXPOSE 3100

# Run index.js when the container launches
CMD ["node", "index.js"]
```

## Step 6: Push the code to github repository from  local repository.

$ git add .

$ git commit -m “commit”

$ git push -u origin main



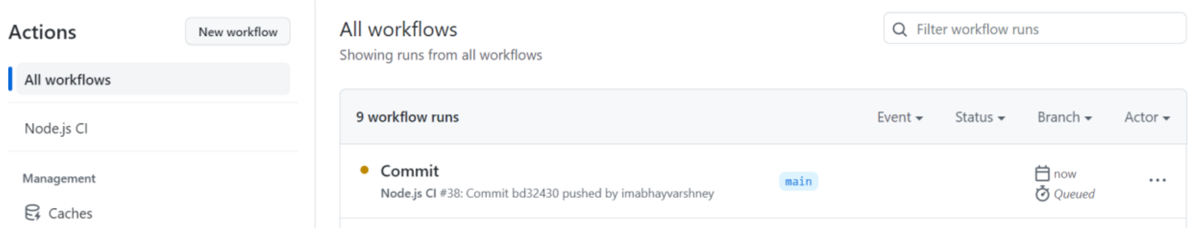
imabhayvarshney Update node_js.yml b3cc5dc 31 minutes ago 43 commits

| | | |
|---|---|---|
| .github/workflows | Update node_js.yml | 31 minutes ago |
| Dockerfile | Add files via upload | 2 days ago |
| README.md | Update README.md | yesterday |
| ecs_main.tf | Update ecs_main.tf | yesterday |
| index.js | Add files via upload | 2 days ago |
| package-lock.json | comit | yesterday |
| package.json | package | 2 days ago |
| terraform.tfstate | Add files via upload | 2 days ago |
| terraform.tfstate.backup | Add files via upload | 2 days ago |
| variables.tf | Create variables.tf | yesterday |

## Step 7 : Browse the alb url to see the deployed nodejs application.

## Step 8: SNAPSHOTS :

Workflow started and created the infrastructure  & build & push the image to ECR and deployed to ECS.



Actions

New workflow

All workflows

Node.js CI

Management

Caches

All workflows

Showing runs from all workflows

Filter workflow runs

9 workflow runs

Event ▾ Status ▾ Branch ▾ Actor ▾

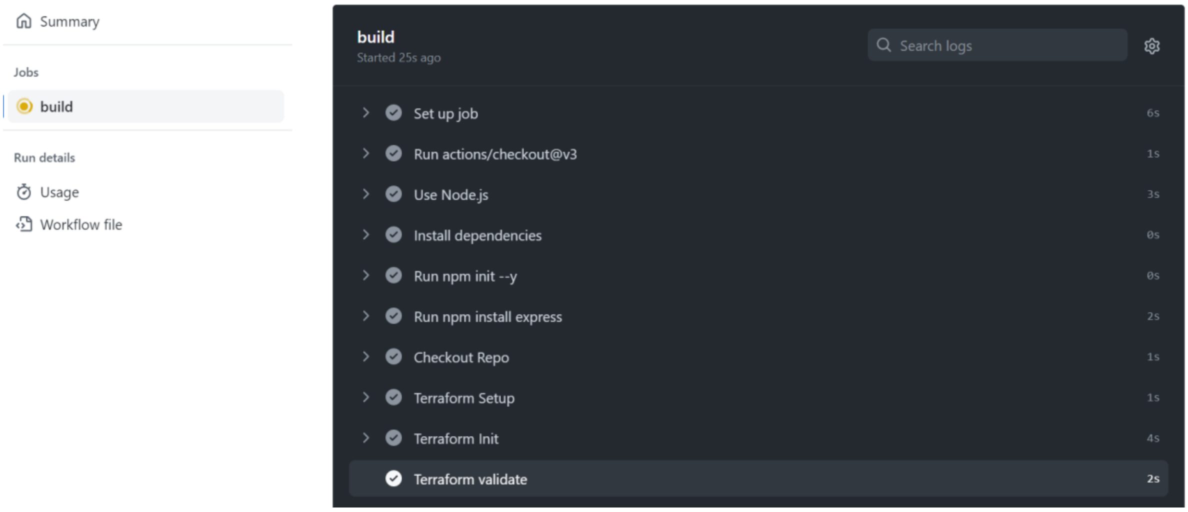
● Commit

Node.js CI #38: Commit bd32430 pushed by imabhayvarshney

main

now

Queued



Summary

Jobs

● build

Run details

Usage

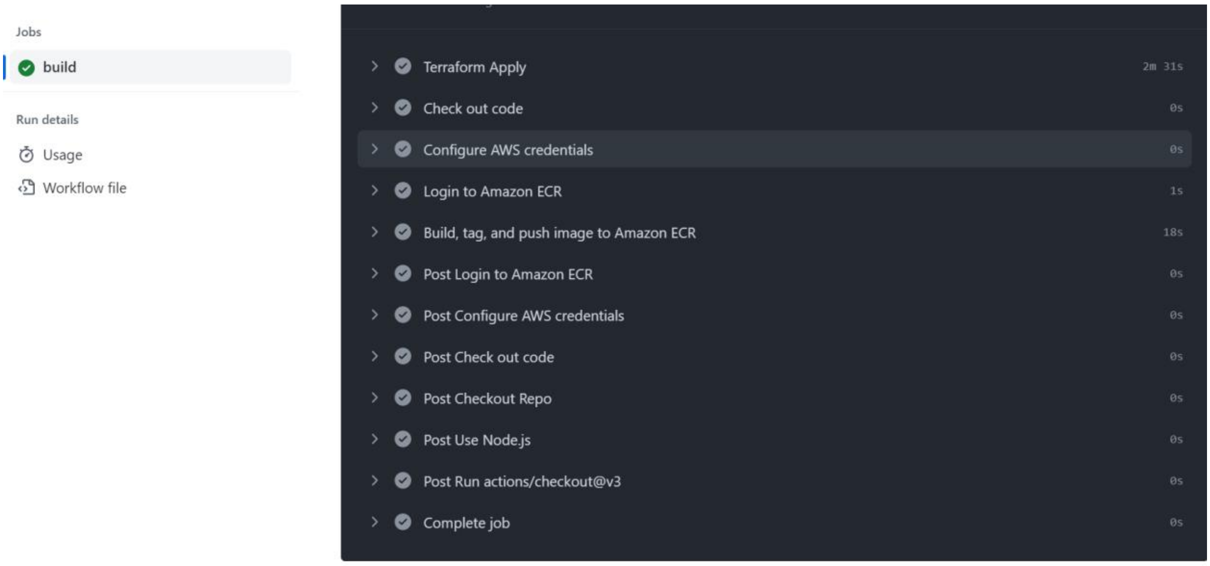
Workflow file

build

Started 25s ago

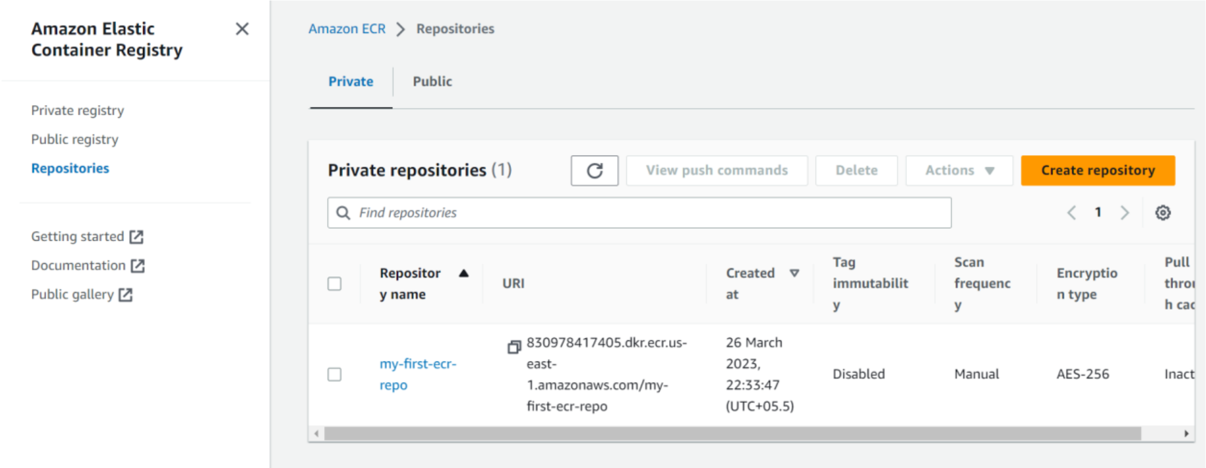
Search logs

| Step | Time |
|---|---|
| Set up job | 6s |
| Run actions/checkout@v3 | 1s |
| Use Node.js | 3s |
| Install dependencies | 0s |
| Run npm init --y | 0s |
| Run npm install express | 2s |
| Checkout Repo | 1s |
| Terraform Setup | 1s |
| Terraform Init | 4s |
| Terraform validate | 2s |



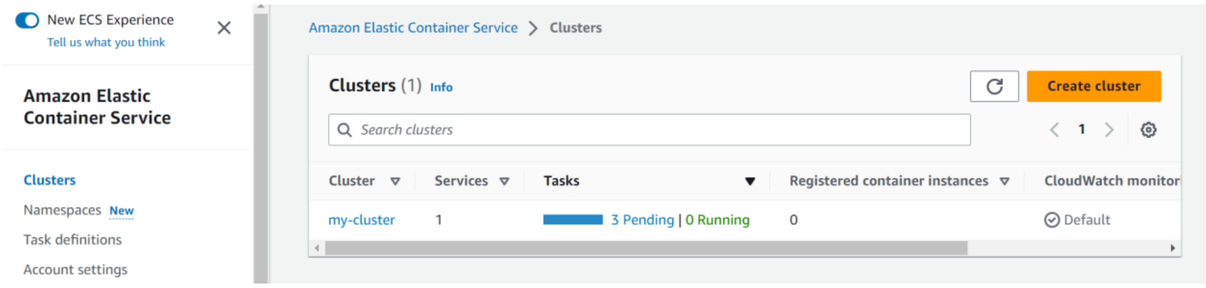
```
> Terraform Apply                                   2m 31s
> Check out code                                    0s
> Configure AWS credentials                         0s
> Login to Amazon ECR                               1s
> Build, tag, and push image to Amazon ECR          18s
> Post Login to Amazon ECR                          0s
> Post Configure AWS credentials                    0s
> Post Check out code                               0s
> Post Checkout Repo                                0s
> Post Use Node.js                                  0s
> Post Run actions/checkout@v3                      0s
> Complete job                                      0s
```

Created ECR :



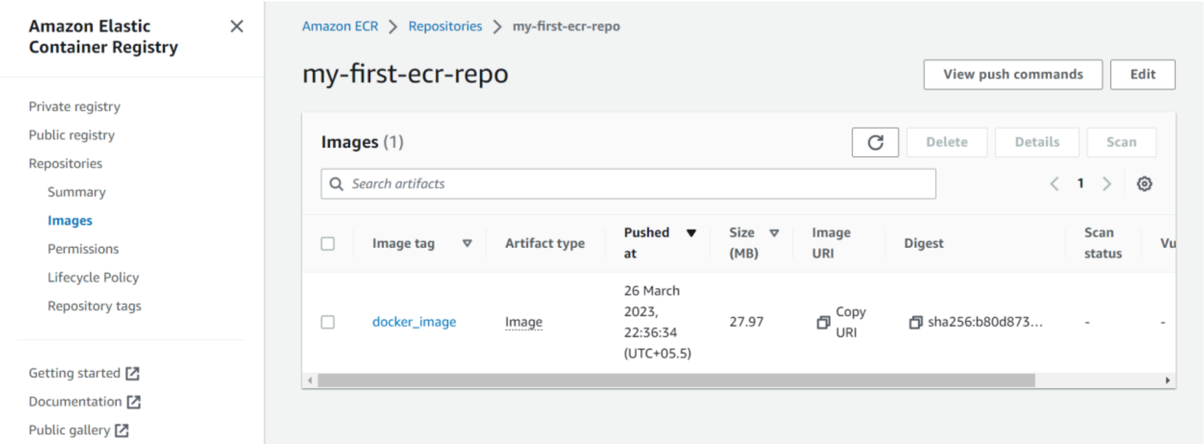
| □ | Repository name ▲ | URI | Created at ▽ | Tag immutability | Scan frequency | Encryption type | Pull through cac |
|---|---|---|---|---|---|---|---|
| □ | my-first-ecr-repo | 830978417405.dkr.ecr.us-east-1.amazonaws.com/my-first-ecr-repo | 26 March 2023, 22:33:47 (UTC+05.5) | Disabled | Manual | AES-256 | Inact |

Created ECS :



| Cluster ▽ | Services ▽ | Tasks | Registered container instances ▽ | CloudWatch monitor |
|---|---|---|---|---|
| my-cluster | 1 | 3 Pending \| 0 Running | 0 | Default |

Pushed image in ECR:

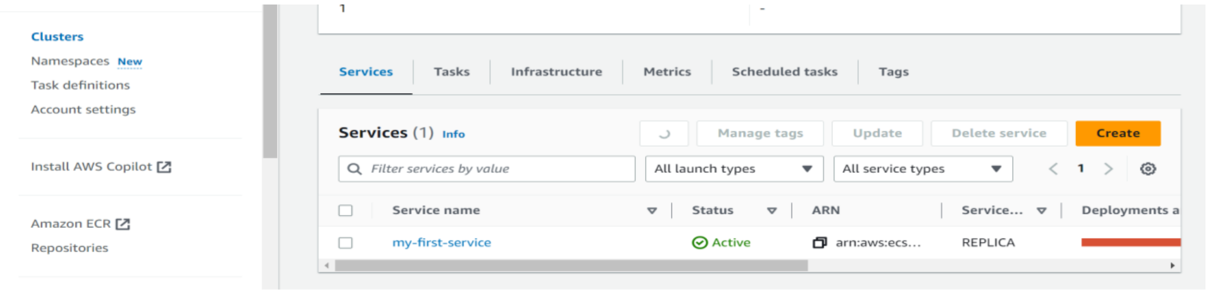


my-first-ecr-repo

Images (1)

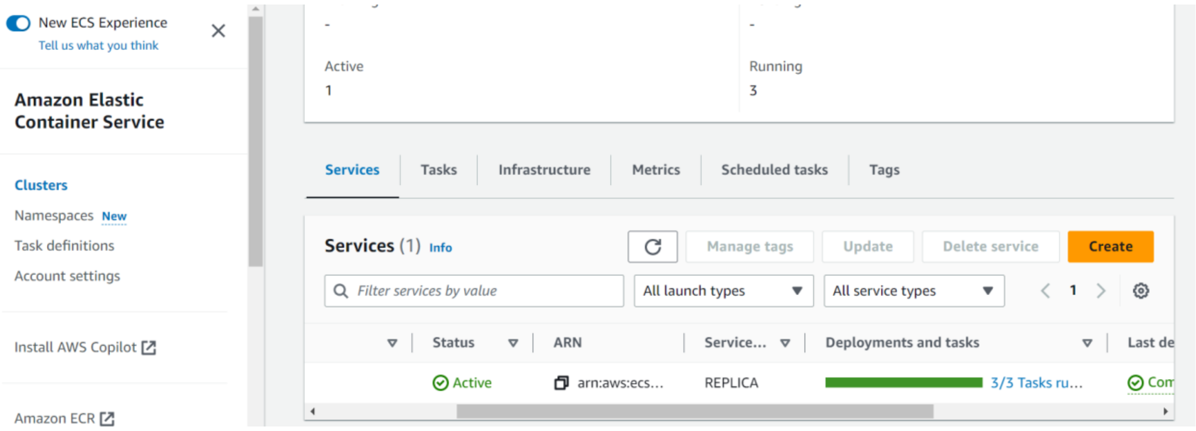
| | Image tag | Artifact type | Pushed at | Size (MB) | Image URI | Digest | Scan status | Vu |
|---|---|---|---|---|---|---|---|---|
| □ | docker_image | Image | 26 March 2023, 22:36:34 (UTC+05.5) | 27.97 | Copy URI | sha256:b80d873... | - | - |

Created service :



Services (1)

| | Service name | Status | ARN | Service... | Deployments a |
|---|---|---|---|---|---|
| □ | my-first-service | Active | arn:aws:ecs... | REPLICA | |

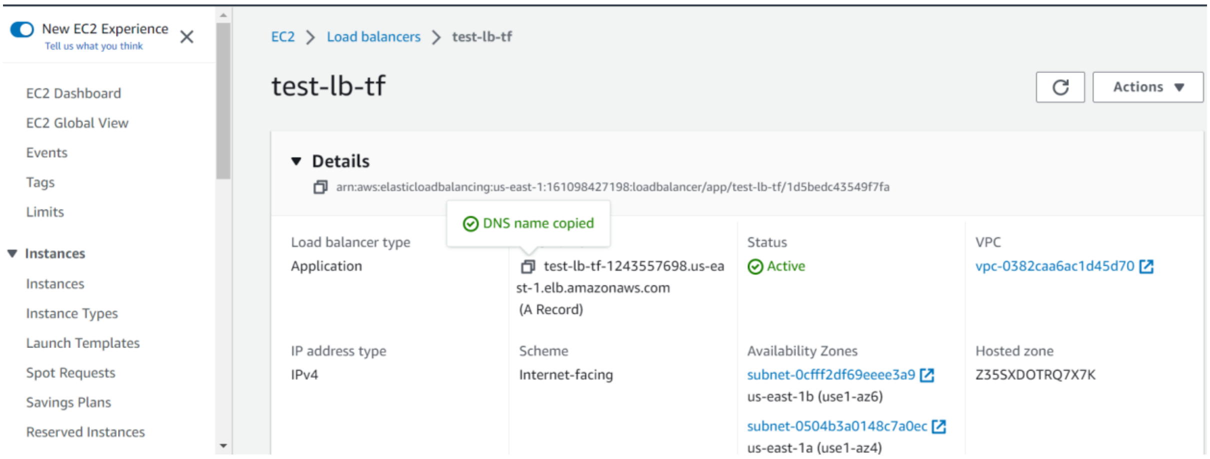


| - | - |
|---|---|
| Active<br>1 | Running<br>3 |

Services (1)

| Status | ARN | Service... | Deployments and tasks | Last de |
|---|---|---|---|---|
| Active | arn:aws:ecs... | REPLICA | 3/3 Tasks ru... | Com |

Created load balancer:



test-lb-tf

Details

arn:aws:elasticloadbalancing:us-east-1:161098427198:loadbalancer/app/test-lb-tf/1d5bedc43549f7fa

DNS name copied

Load balancer type
Application

test-lb-tf-1243557698.us-east-1.elb.amazonaws.com
(A Record)

Status
Active

VPC
vpc-0382caa6ac1d45d70

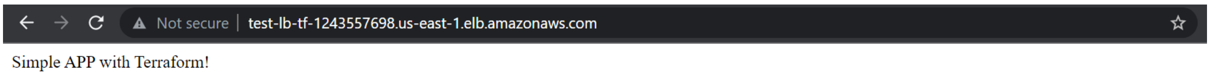
IP address type
IPv4

Scheme
Internet-facing

Availability Zones
subnet-0cfff2df69eeee3a9
us-east-1b (use1-az6)
subnet-0504b3a0148c7a0ec
us-east-1a (use1-az4)

Hosted zone
Z35SXDOTRQ7X7K

Deployed node js app:



test-lb-tf-1243557698.us-east-1.elb.amazonaws.com

Simple APP with Terraform!

You can use my github repository for the the code and workflow:

https://github.com/imabhayvarshney/ecr-ecs-gha-ter.git