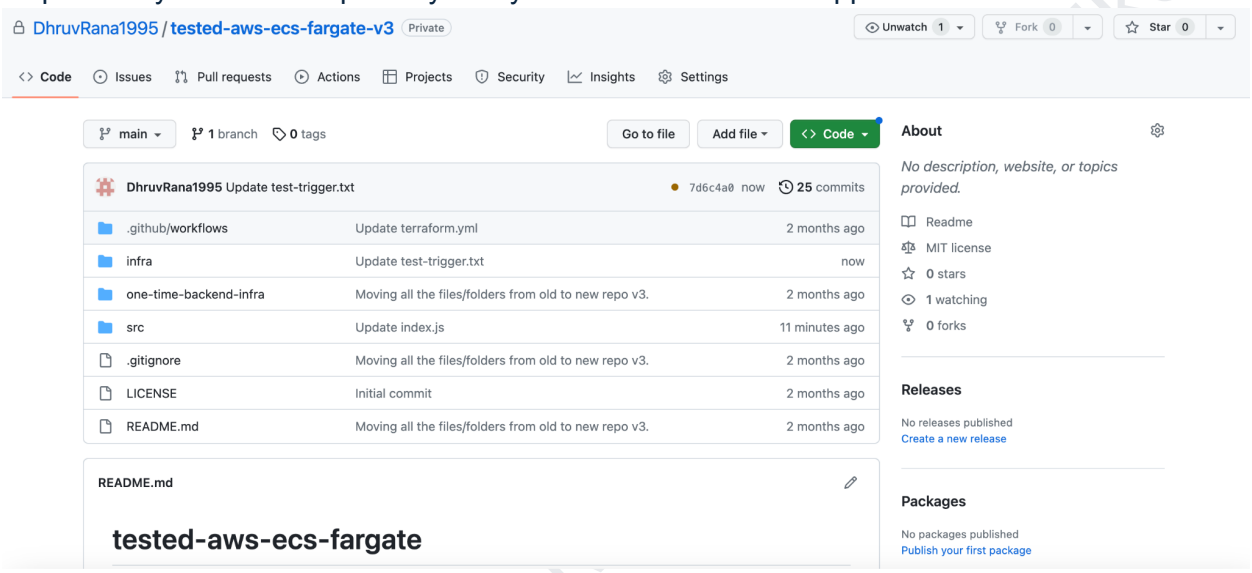
CI/CD Pipeline for App build and deployment on AWS ECS Fargate Cluster

To implement the CI/CD process for an Application to get it built in the pipeline using Docker tool and get that new Docker image deployed on the AWS Cloud platform on the AWS ECS Cluster we need to follow below listed steps:

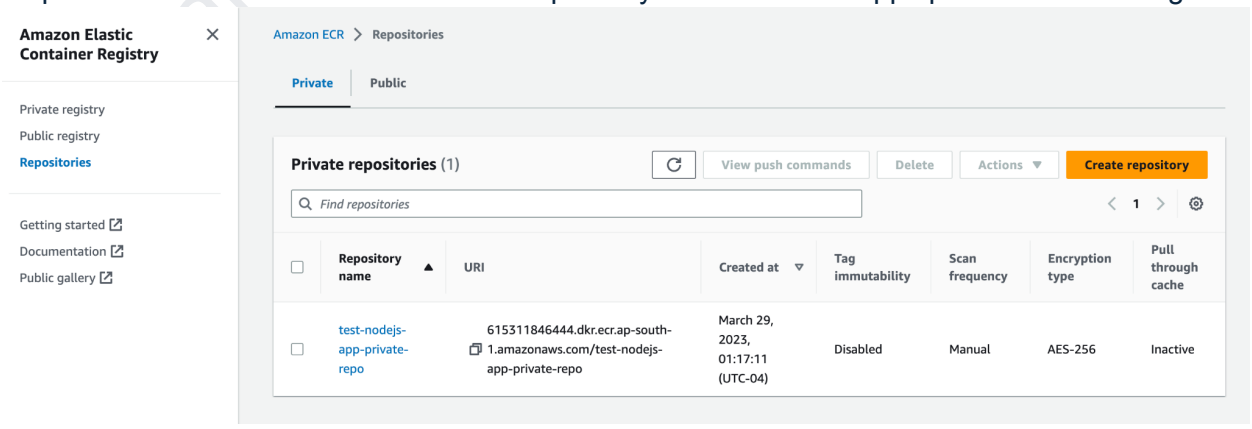
Step 1. Get your Github repository ready with the Infra and the App source code.



You can create your own private repository with all the code setup i.e., Terraform code base under infra directory and your App code (In this case we are going to consider NodeJs sample App code) under src directory.

Once, you have the code base ready, check in to the Git repository using SSH/HTTPs/Token based Auth process.

Step 2. Create a new AWS ECR Private repository to store all the App specific Docker images.



To store the built Docker images of the App code in the pipeline environment, we would need an ECR repository on the AWS Cloud platform.

You can create one in your AWS Account by navigating into the ECR → Repository → Private section → Click on Create Repository and provide the basic details like repository name and keep rest of the settings as it is, just hit the Create Repository button. You now have the ECR Private repo ready.

Step 3. Configure the Workflow specification file in a YML format and validate it online.

```yaml
name: "NodeJs app build and deploy"

on:
  push:
    branches:
      - main
    paths:
      - src/**
  pull_request:

jobs:
  terraform:
    name: "NodeJs app build and deploy"
    runs-on: ubuntu-latest
    continue-on-error: true
    environment: test
    env:
      AWS_ACCESS_KEY_ID: ${{ secrets.AK_VALUE }}
      AWS_SECRET_ACCESS_KEY: ${{ secrets.SK_VALUE }}
      AWS_PROFILE: ${{ secrets.AWS_PROFILE_NAME }}
      AWS_DEFAULT_REGION: ${{ secrets.AWS_REGION }}
      APP_PORT_NUMBER : ${{ secrets.APP_PORT_NUMBER }}
      AWS_PRIVATE_ECR_REPO_NAME: ${{ secrets.AWS_PRIVATE_ECR_REPO_NAME }}

    # Use the Bash shell regardless whether the GitHub Actions runner is ubuntu-latest, macos-latest, or windows-latest
    defaults:
      run:
        shell: bash

    steps:
      # Checkout the repository to the GitHub Actions runner
      - name: Checkout
        uses: actions/checkout@v3

      # Checking tools version
      - name: Checking AWS version
        run: aws --version
```

GitHub Actions works as per the configurations setup in the YML file present at the .github/workflows directory from the GitHub repo root location. Above is the config YML file which has the steps for a typical NodeJs App CI/CD steps using Docker, AWS CLI and Terraform tools.

This pipeline has the trigger set on the main branch (line #51) with any changes detected in the src/ directory (line #53) via a new commit/push event. Furthermore, it uses the Ubuntu latest base image for all the pipeline operations (line #59) with any error skipped if found during pipeline execution (line #60).

```
84      # Configuring AWS CLI for CI Steps
85      - name: Configuring AWS CLI
86        env:
87          AWS_AK_VALUE: ${{ secrets.AK_VALUE }}
88          AWS_SK_VALUE: ${{ secrets.SK_VALUE }}
89          AWS_REG_VALUE: ${{ secrets.AWS_REGION }}
90        run: |
91          aws configure set aws_access_key_id $AWS_AK_VALUE
92          aws configure set aws_secret_access_key $AWS_SK_VALUE
93          aws configure set region $AWS_REG_VALUE
94          aws s3 ls
95
96      # Verifying installed Docker version
97      - name: Verifying installed Docker version
98        run: |
99          docker --version
100         docker images
101
102     # AWS ECR Login, Build and Push the app Docker img
103     - name: CI Steps
104       if: always()
105       continue-on-error: true
106       working-directory: ./src
107       env:
108         AWS_PRIVATE_ECR_REPO_NAME: ${{ secrets.AWS_PRIVATE_ECR_REPO_NAME }}
109         AWS_ACCOUNT_ID: ${{ secrets.AWS_ACCOUNT_ID }}
110         AWS_REG_VALUE: ${{ secrets.AWS_REGION }}
111         APP_PORT_NUMBER : ${{ secrets.APP_PORT_NUMBER }}
112       run: |
113         aws ecr get-login-password --region $AWS_REG_VALUE | docker login --username AWS --password-stdin $AWS_ACCOUNT_ID.dkr.ecr.$AWS_REG_VALUE.amazonaws.com
114         echo commit id: $GITHUB_SHA
115         docker build --build-arg APP_PORT=$APP_PORT_NUMBER -t $AWS_PRIVATE_ECR_REPO_NAME:$GITHUB_SHA .
116         docker tag $AWS_PRIVATE_ECR_REPO_NAME:$GITHUB_SHA $AWS_ACCOUNT_ID.dkr.ecr.$AWS_REG_VALUE.amazonaws.com/$AWS_PRIVATE_ECR_REPO_NAME:$GITHUB_SHA
117         docker push $AWS_ACCOUNT_ID.dkr.ecr.$AWS_REG_VALUE.amazonaws.com/$AWS_PRIVATE_ECR_REPO_NAME:$GITHUB_SHA
118
```

From line #85-94 we are configuring the AWS CLI took by supplying the AWS CLI credentials i.e., Access Key, Secret Key and Region value. Once AWS CLI is configured we have kept a checkpoint at line #94 to verify if the AWS CLI is configured properly or not using S3 list option.

As we are going to use Docker tool to package the App build, we are checking the availability of the Docker tool by checking it's version and listing the available Docker images from line #97-100.

At last, once all the App & AWS prerequisites are ready we initiate the actual CI (Continuous Integration) steps from line #103-117 where we are first loggin in the ECR repository where we are going to push the Docker image. Then, executing docker build, tag and push commands to build and upload the latest App Docker image attached with a unique tag value i.e., GITHUB_SHA which has the commit ID value of the push event which triggered the pipeline.

```
119     # Configuring AWS CLI for CD Steps
120     - name: Configuring AWS CLI
121       env:
122         AWS_AK_VALUE: ${{ secrets.AK_VALUE }}
123         AWS_SK_VALUE: ${{ secrets.SK_VALUE }}
124         AWS_REG_VALUE: ${{ secrets.AWS_REGION }}
125         AWS_PROFILE_NAME_VALUE: ${{ secrets.AWS_PROFILE_NAME }}
126       run: |
127         aws configure set aws_access_key_id $AWS_AK_VALUE --profile $AWS_PROFILE_NAME_VALUE
128         aws configure set aws_secret_access_key $AWS_SK_VALUE --profile $AWS_PROFILE_NAME_VALUE
129         aws configure set region $AWS_REG_VALUE --profile $AWS_PROFILE_NAME_VALUE
130         aws s3 ls --profile $AWS_PROFILE_NAME_VALUE
131
132     # Deploying updated ECR repo image on the ECS Fargate Cluster using Terraform scripts
133     - name: CD Steps
134       working-directory: ./infra
135       env:
136         AWS_PRIVATE_ECR_REPO_NAME: ${{ secrets.AWS_PRIVATE_ECR_REPO_NAME }}
137         AWS_ACCOUNT_ID: ${{ secrets.AWS_ACCOUNT_ID }}
138         AWS_REG_VALUE: ${{ secrets.AWS_REGION }}
139         APP_PORT_NUMBER : ${{ secrets.APP_PORT_NUMBER }}
140       run: |
141         terraform --version
142         terraform init
143         terraform fmt -check
144         terraform validate
145         export UPDATED_DOCKER_IMG=$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REG_VALUE.amazonaws.com/$AWS_PRIVATE_ECR_REPO_NAME:$GITHUB_SHA
146         echo current docker image value: $UPDATED_DOCKER_IMG
147         terraform plan -var="app_image=$UPDATED_DOCKER_IMG" -var="app_port=$APP_PORT_NUMBER"
148         terraform apply -var="app_image=$UPDATED_DOCKER_IMG" -var="app_port=$APP_PORT_NUMBER" -auto-approve
149         terraform show
150         #terraform destroy -var="app_image=$UPDATED_DOCKER_IMG" -var="app_port=$APP_PORT_NUMBER" -auto-approve
151
152
```

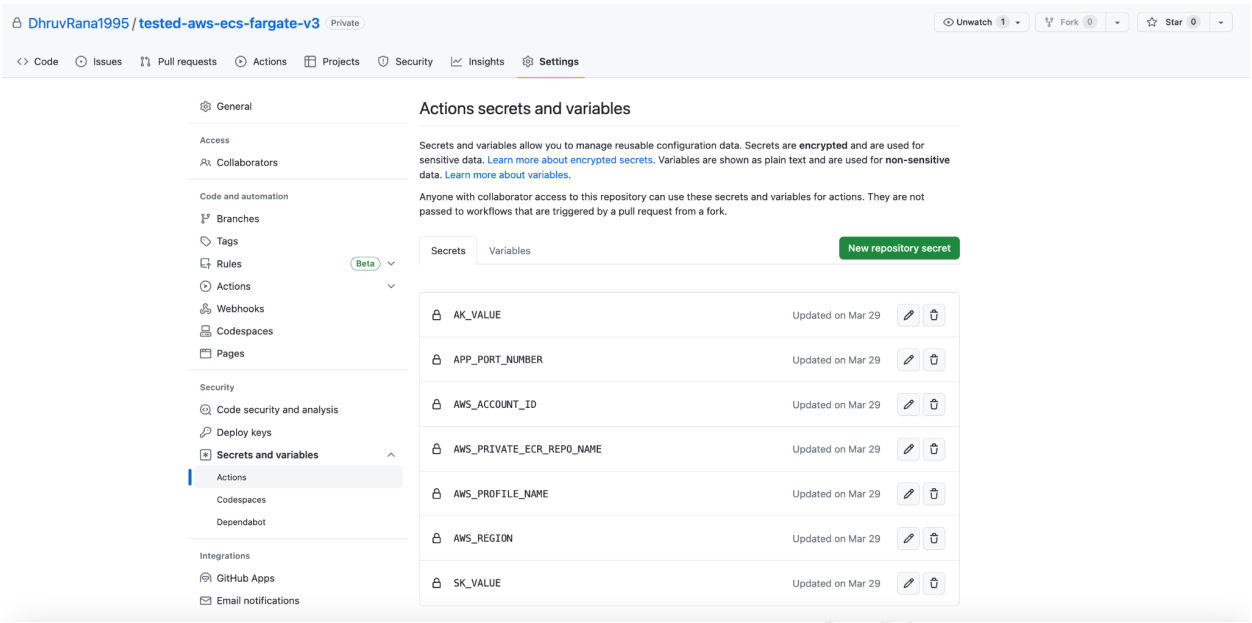
After completion of the CI steps, we re-configure the AWS CLI tool with Terraform specific Profile value that is extracted again from the GitHub Actions secrets. Once, AWS CLI is configured for the Terraform specific profile, we execute the standard Terraform operation with couple of App specific variable value i.e., App port and App Docker Image value.

Here, the App port number is obtained from the GitHub Actions secrets and the Docker image value is dynamically prepared at line #145 as an environment variable using various dynamic values such as AWS Account ID, Region code, ECR repo name and the Github commit id value.

All the other dynamic values except Github commit id value is coming from the GitHub Actions secrets because the commit id value is coming from GITHUB_SHA environment variable which is a default GitHub Actions environment variable available for all the Actions pipelines. A list of such default environment variable specifically for GitHub Actions workflows are listed in this document prepared by GitHub Official.

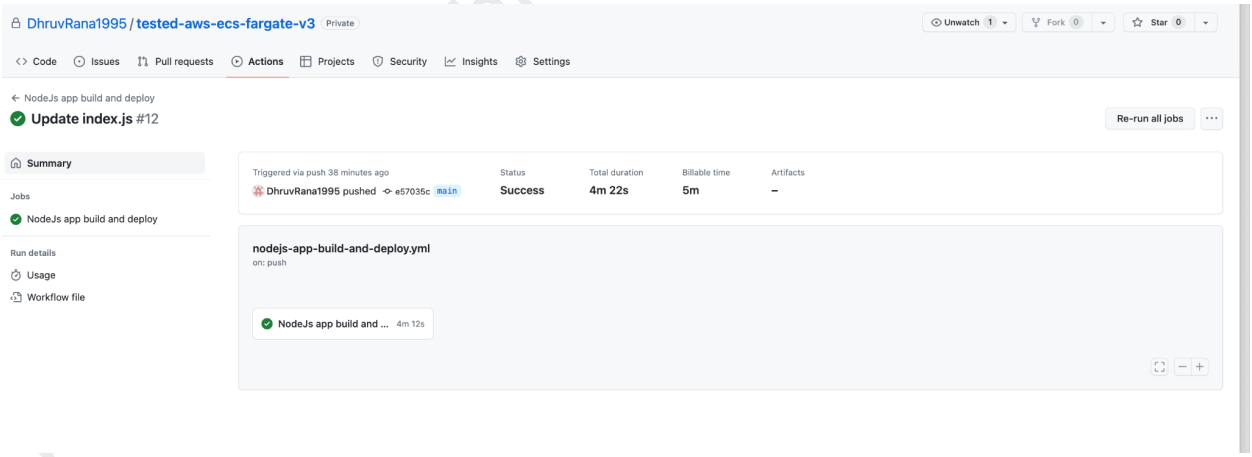
Step 4. Define the App/Infra secrets in the Github Actions Secrets to be used in the pipeline environment.

There a few environment variables defined on the beginning (line #62-68) whose values are extracted front he GitHub Actions secrets as shown below:



Step 5. Trigger the pipeline as per the trigger event set on the respective Git branch and check pipeline logs for more details.

Triggered GitHub Actions pipeline with execution metadata can be found as per the screen below:



Here are a few glimps of the Pipeline logs and how you can check the execution details to go through the error statements if any and troubleshoot the pipeline issues by understanding the problem statement by going through the logs as shown below: