Technical Challenge Completion Acknowledgment

I am pleased to inform you that I have successfully completed the technical challenge, and all deliverables have been met as per the task requirements. Below is the checklist of the required tasks and deliverables, with each item marked as completed:

Deliverables

- ✓ **Proper documentation as a README.md in the repo**
  The README file has been updated with a detailed explanation of the project, the infrastructure, and the CI/CD pipeline setup. It also includes architecture diagrams for both the infrastructure and CI/CD.
- ✓ **All code in a git public repo**
  The full project code is available in my public GitHub repository: Timeoff Management Application.
- ✓ **Fix the application if it has any issues**
  The application had some minor dependency issues since it has been open sourced 4-5 years ago, with some research I have fixed the issues with Dockerfile.
- ✓ **Demo the solution**
  The application is now live and can be accessed through a public URL, with HTTPS enabled for secure connections. A demo can be provided if needed.
- ✓ **Document what would you improve and what challenges you faced**
  With dedicated time frame I might be able to make it more secure security wise on infrastructure and application integrity,

Acceptance Criteria

- ✓ **Architecture diagram for the solution**
  The architecture diagrams for both the infrastructure and CI/CD pipeline are included in the documentation. These diagrams provide a clear overview of the solution architecture.
- ✓ **Required infra running in AWS and deployed using IaC**
  The infrastructure has been deployed on **AWS ECS** using **Terraform**. The necessary AWS resources, including VPC, subnets, security groups, load balancer, and ECS tasks, are configured and running.
- ✓ **App must be deployed in a fully automated way (CI/CD) triggered by a change in the git repo**
  The **GitHub Actions CI/CD** pipeline automatically builds and deploys the application every time there is a change in the main branch. The deployment process is fully automated.
- ✓ **App should be served using HTTPS and HTTP should be redirected to HTTPS**
  The application is deployed with **HTTPS** using a self-signed certificate. HTTP traffic is automatically redirected to HTTPS.
- ✓ **App needs to be HA and load balanced in at least 2 AZ**
  The application is deployed in a **highly available us-east-1 zone** setup with **2 availability zones (AZ)**, and the load balancing is handled by an **Application Load Balancer** (ALB).
- ✓ **App servers be in a private subnet & load balancer in a public subnet (exposed to public via internet gateway)**
  The application servers are running in **private subnets**, and the **ALB** is exposed to the public via a **public subnet** and an **internet gateway**.

Walkthrough for Setting Up the Project

1. **Provisioning Infrastructure (Terraform)**

   Initialize the Terraform workspace:

   ```
   terraform init
   ```

   Review the planned changes:

   ```
   terraform plan
   ```

   Apply the Terraform configuration to provision the infrastructure:

   ```
   terraform apply
   ```

   This will create all the necessary AWS resources as described in the main.tf file.

2. **Building and Deploying Docker Images**

   Login to Docker Hub (for pushing images):

   ```
   docker login
   ```

   Build the Docker image:

   ```
   docker build -t timeoff .
   ```

   Tag the Docker image:

   ```
   docker tag timeoff syedimran18/timeoff-management:latest
   ```

   Push the Docker image to Docker Hub:

   ```
   docker push syedimran18/timeoff-management:latest
   ```

3. **CI/CD Pipeline (GitHub Actions)**

   The **GitHub Actions CI/CD** pipeline automatically triggers when changes are pushed to the main branch. It builds and deploys the latest Docker image to AWS Fargate.

   Incremental changes are automatically picked up by the pipeline, ensuring continuous deployment and up-to-date application availability.

4. **GitHub Secrets**

   For security and smooth deployment, I have added the following credentials to GitHub Secrets:

   Docker Hub Credentials: **DOCKER_USERNAME, DOCKER_PASSWORD**

   AWS Credentials: **AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY**

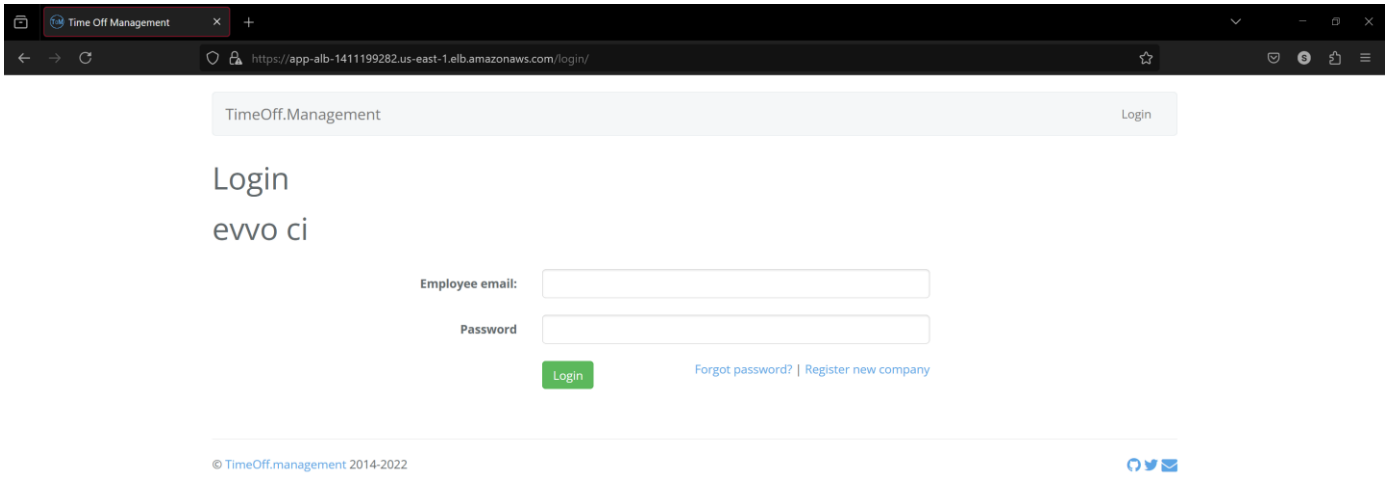
   These secrets are used in the GitHub Actions CI/CD pipeline to authenticate with Docker Hub and AWS, ensuring a secure and automated build and deployment process.

5. Initial Legacy deployment

   Once initial legacy deployment is completed the terraform main.tf code has a output section which will give a ECS Fargate service loadbalcer URL for example: https://app-alb-1411199282.us-east-1.elb.amazonaws.com/login/

Demo

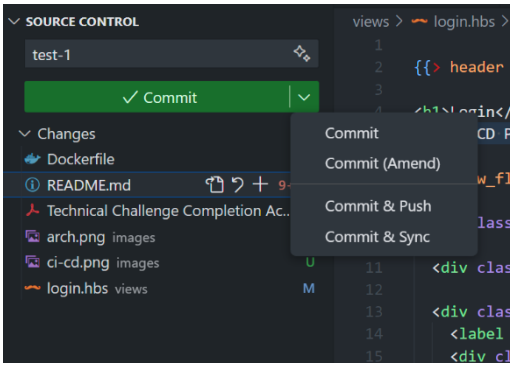
1. Once the initial legacy deployment when we visit the LB Url



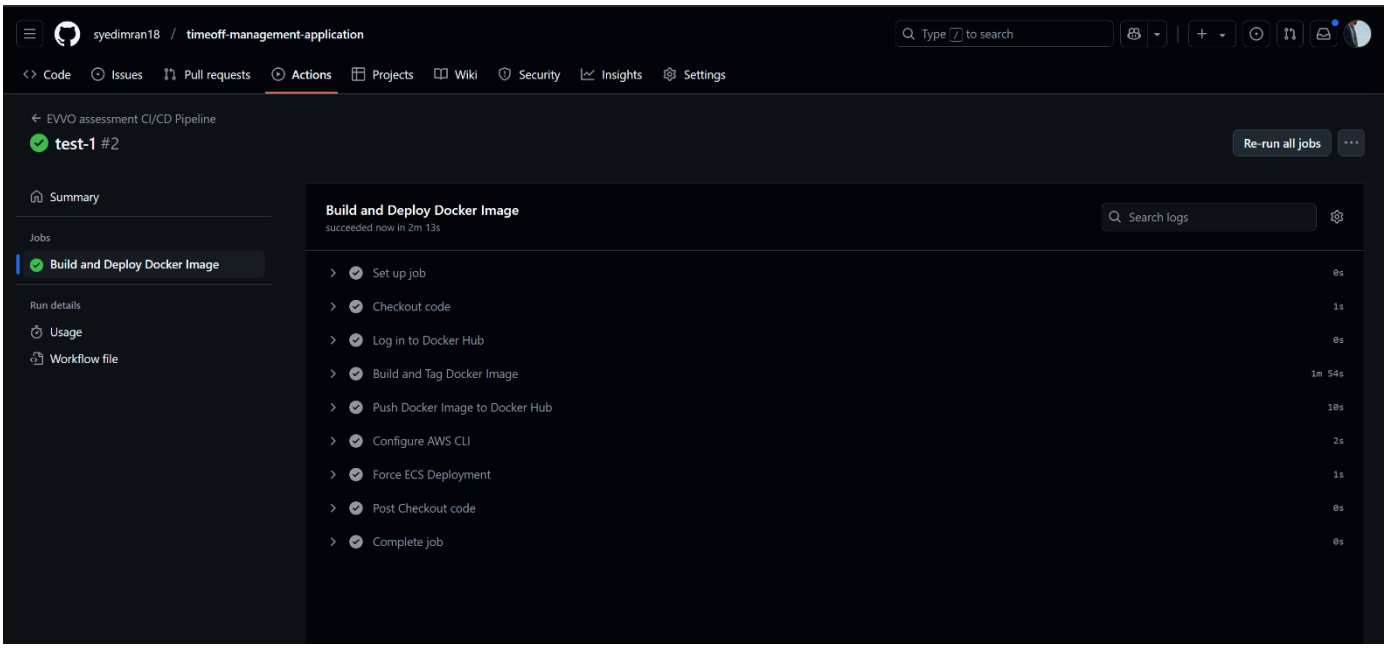
2. Testing the ci/cd workflows

Making changes in login page

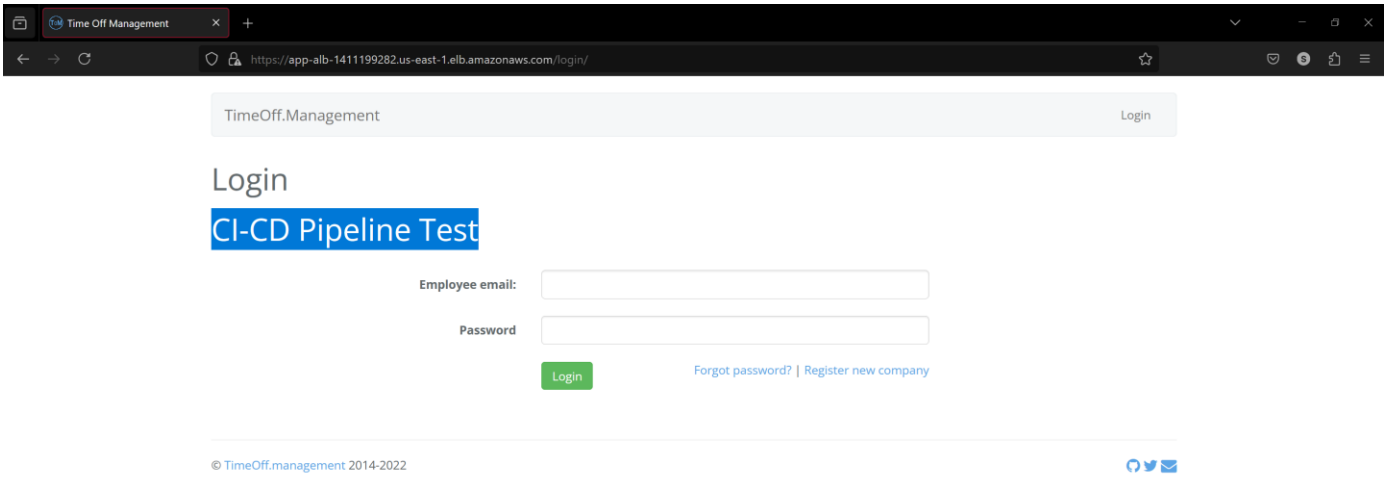
commit and push new changes to github



github action ci/cd pipelline



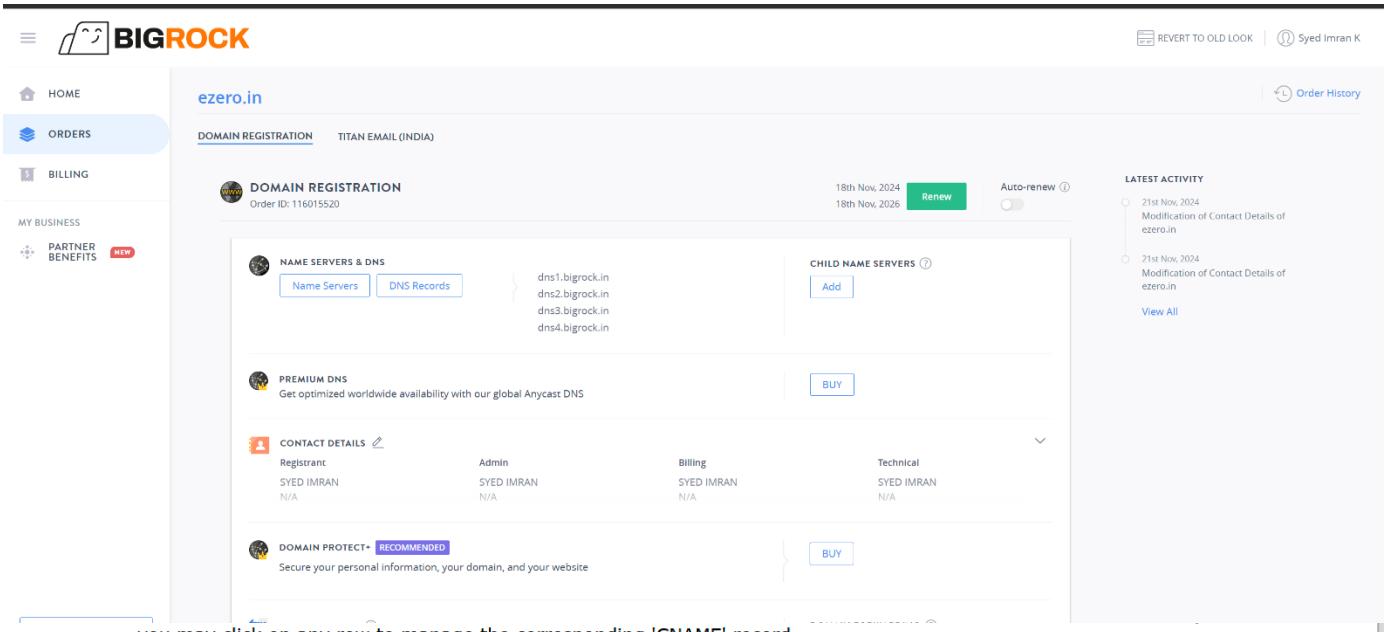
Post ci/cd pipeline when finished changes, as we made In login page for the demo.



Additional tasks from my end

I'm mapping the load balancer url to my personal domain name for additional layer of my tech perks 😊

Add new DNS record, CName



you may click on any row to manage the corresponding 'CNAME' record

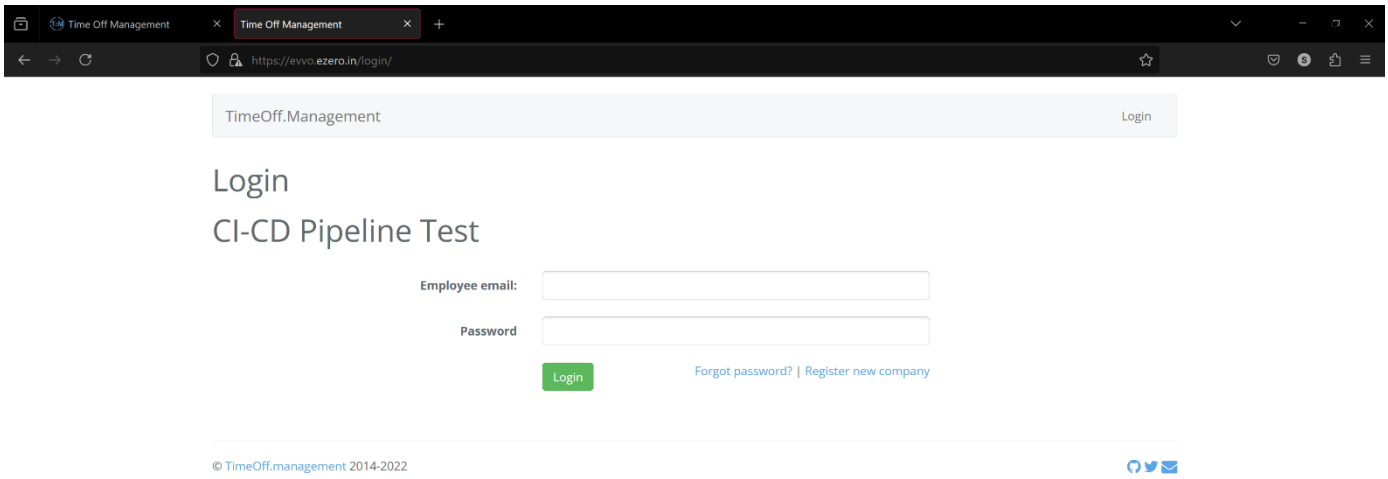
List of CNAME Records

Add CNAME Record | 1 | pg no | Jump To

Sr No	Record Id	Name	Value	Status
1	148067412	evvo.ezero.in	app-alb-1411199282.us-east-1.elb.amazonaws.com	Active

1 | pg no | Jump To

Now we can visit evvo.ezero.in to access our application



I hope this meets the expectations outlined in the technical challenge. If there are any further questions or if you need additional details, feel free to let me know.

Looking forward to your feedback!