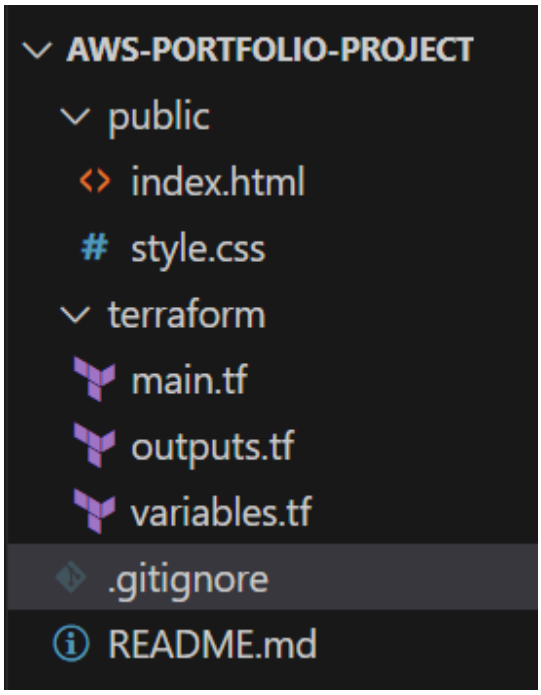
Static Portfolio Hosting on AWS using Terraform & GitHub Actions

Author: Harsh Pashine

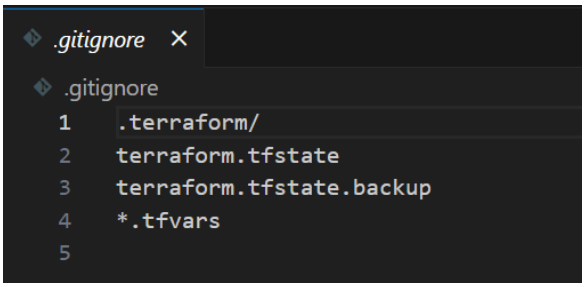
Project Folder: aws-portfolio-project

1. Project Structure Setup

- Project layout:



- Initialize Git [commands]:
  1. cd aws-portfolio-project
  2. git init
- Recommended .gitignore:

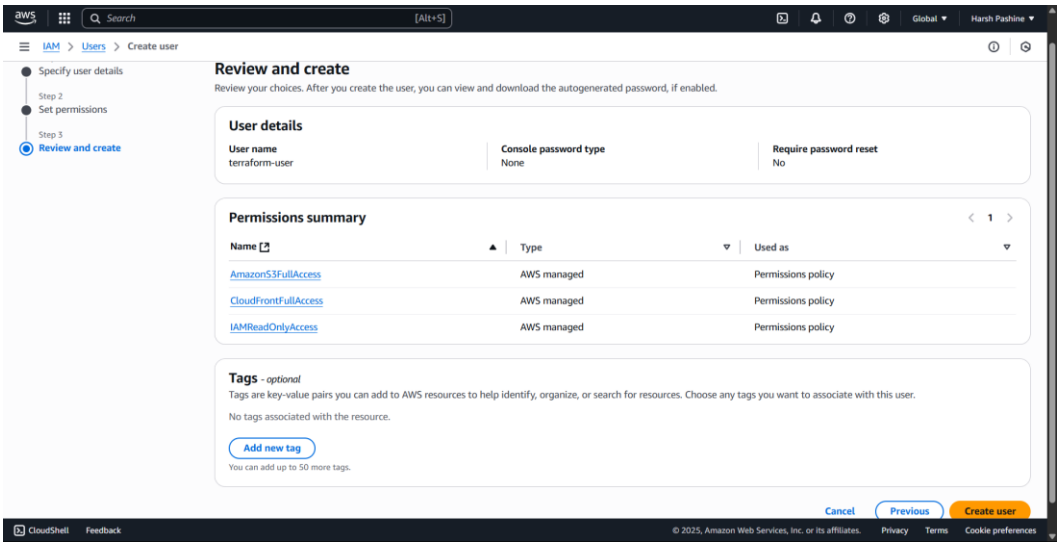


2. IAM User Creation

To securely manage AWS resources through Terraform and GitHub Actions, you need a dedicated IAM user with the right set of permissions.

- Go to AWS Console → IAM → Users → Create user
- Attach these policies:
  1. AmazonS3FullAccess
  2. CloudFrontFullAccess
  3. IAMReadOnlyAccess

IAM user created for Terraform and CI/CD access.



3. Setting Up AWS Access Keys

To allow Terraform and GitHub Actions to authenticate with your AWS account, you'll need to generate a pair of programmatic access credentials using IAM.

- Go to: IAM → Users → Select User → Security Credentials tab → Access Keys → Create Access Key
- Use case: Command Line Interface (CLI)
- Save:
  1. Access Key ID
  2. Secret Access Key
- Store in Windows CMD:
  1. setx AWS_ACCESS_KEY_ID "your-access-key-id"
  2. setx AWS_SECRET_ACCESS_KEY "your-secret-access-key"
- Check via:
  1. echo %AWS_ACCESS_KEY_ID%

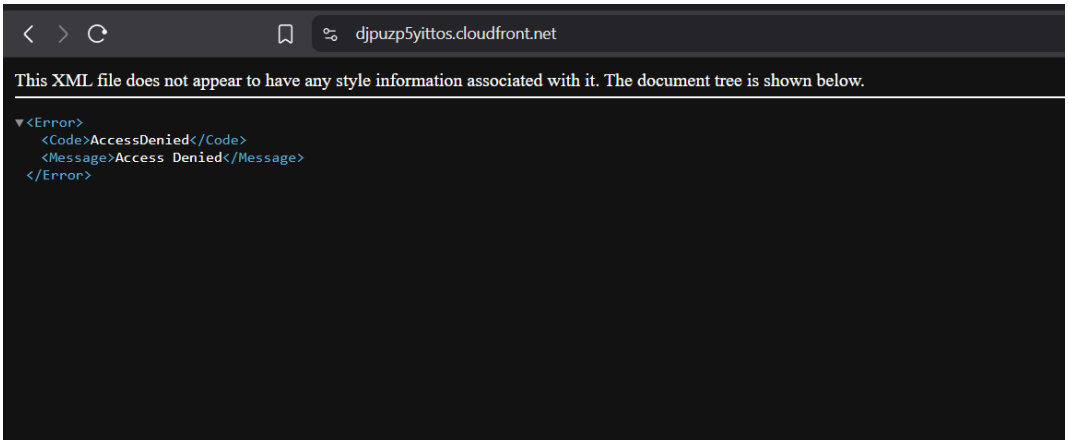
4. Terraform Initialization

Once the Terraform configuration files (main.tf, variables.tf, and outputs.tf) are in place, it's time to provision the AWS infrastructure.

- Navigate to the Terraform directory:
  1. cd terraform/
- Run the following Terraform commands in sequence:
  1. terraform init [Initializes the working directory and downloads the necessary provider plugins.]
  2. terraform validate [Checks the configuration files for syntax errors and ensures everything is logically correct.]
  3. terraform plan [Displays a preview of the actions Terraform will take resources it will create, change, or destroy.]
  4. terraform apply [Applies the planned changes and provisions the infrastructure on AWS. You'll be prompted to confirm by typing yes.]
- Terraform provisions the following AWS resources:
  1. S3 Bucket – For storing and serving the static portfolio files
  2. CloudFront Distribution – For global CDN and HTTPS access
  3. Origin Access Identity (OAI) – To restrict S3 access to only CloudFront
  4. S3 Bucket Policy – Grants CloudFront permission to read objects securely
- Terminal output showing successful terraform apply:

```
aws_cloudfront_distribution.cdn: Creation complete after 5m28s [id=E3A8JGQQQBTPNO]

Apply complete! Resources: 7 added, 0 changed, 0 destroyed.

Outputs:

cloudfront_url = "djpuzp5yittos.cloudfront.net"
PS C:\Users\LENOVO\Documents\aws-portfolio-project\terraform>
```

- CloudFront url output:



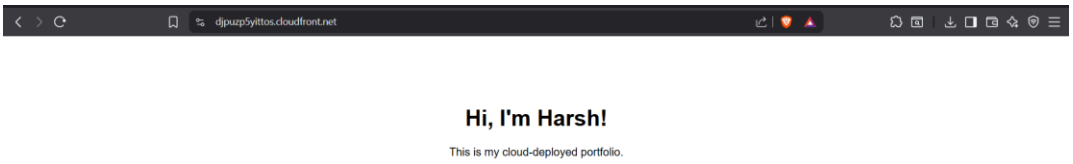
5. Manual Upload to S3 (First Time Only)

Used to verify infra and test CloudFront before CI/CD.

- Command:
  1. aws s3 sync ../public s3://harsh-portfolio-site-unique
- Output:

```
PS C:\Users\LENOVO\Documents\aws-portfolio-project> cd public
PS C:\Users\LENOVO\Documents\aws-portfolio-project\public> aws s3 sync ../public s3://harsh-portfolio-site-unique
upload: .\style.css to s3://harsh-portfolio-site-unique/style.css
upload: .\index.html to s3://harsh-portfolio-site-unique/index.html
```

- CloudFront url output:



6. GitHub Actions Setup for CI/CD

After the infrastructure is provisioned and tested, the next step is to automate deployment of the portfolio files (index.html, style.css, etc.) using GitHub Actions.

- Add Secrets to GitHub
  1. Go to your GitHub repository:
  2. Navigate to: Settings > Secrets and variables > Actions
  3. Click “New repository secret” and add:
     a. Name: AWS_ACCESS_KEY_ID [in value: Your IAM access key]
     b. Name: AWS_SECRET_ACCESS_KEY [in value: Your IAM secret access key]
- Create the Deployment Workflow
  1. In your repository, create a new file:
     a. .github/workflows/deploy.yml
  2. You can use this script:

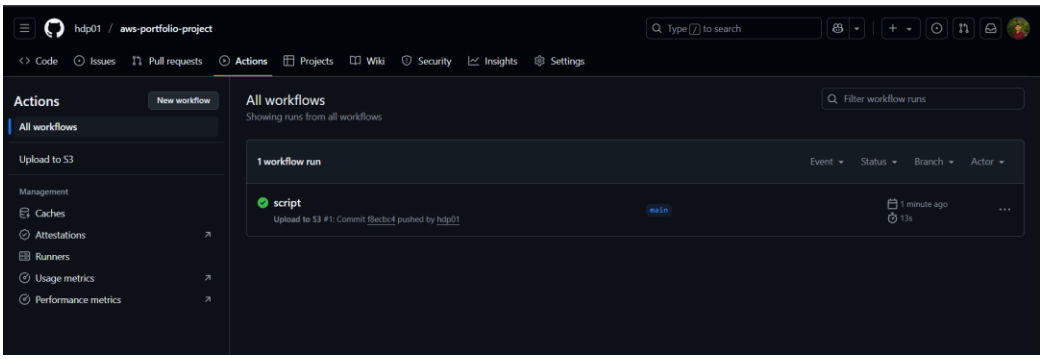
deploy.yml

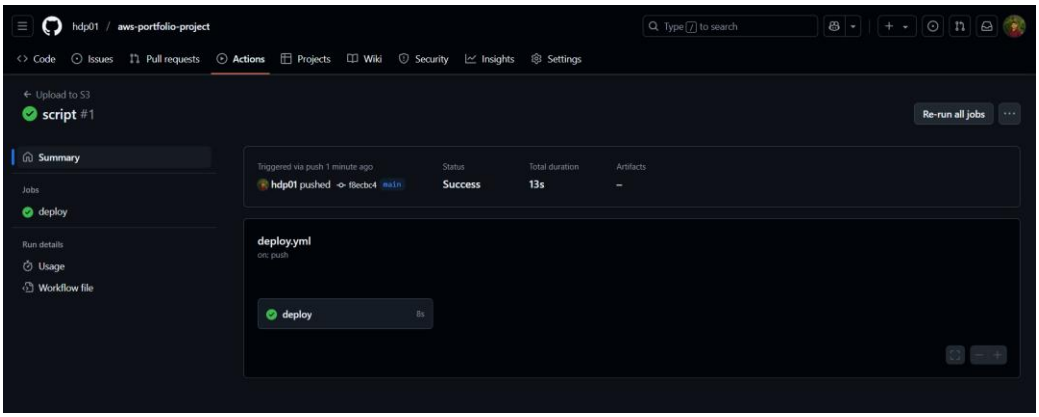
.github > workflows > deploy.yml > {} jobs > {} deploy > [ ] steps > {} 2

GitHub Workflow - YAML GitHub Workflow (github-workflow.json)

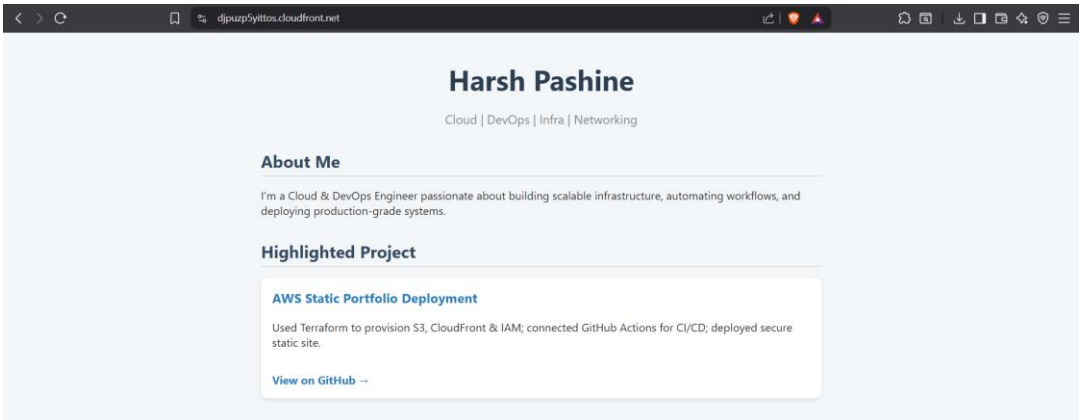
```yaml
name: Upload to S3

on:
  push:
    branches: [main]

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v4

      - name: Configure AWS credentials
        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: us-east-1

      - name: Sync to S3
        run: aws s3 sync ./public s3://harsh-portfolio-site-unique --delete
```

- Now, any changes to your public/ folder will be automatically deployed to S3 after each commit.
- Output [GitHub Actions page showing a successful workflow run]:





- CloudFront url output:



7. Teardown & Cleanup

To remove everything and avoid AWS billing:

- Command [Empty the S3 Bucket]:

1. aws s3 rm s3://harsh-portfolio-site-unique –recursive

- Output

```
PS C:\Users\LENOVO\Documents\aws-portfolio-project> aws s3 rm s3://harsh-portfolio-site-unique --recursive
delete: s3://harsh-portfolio-site-unique/index.html
delete: s3://harsh-portfolio-site-unique/style.css
PS C:\Users\LENOVO\Documents\aws-portfolio-project>
```

- Destroy Infrastructure with Terraform
  1. Navigate to the Terraform directory and run the destroy command:
     a. terraform destroy

```
aws_s3_bucket.portfolio: Destruction complete after 2s

Destroy complete! Resources: 7 destroyed.
PS C:\Users\LENOVO\Documents\aws-portfolio-project\terraform>
```

- Delete IAM user (Optional)
  1. Go to AWS Console → IAM → Users
  2. Select your IAM user (e.g., terraform-user)
  3. Click Delete
  4. Confirm to remove the user and associated credentials

8. Learnings & Summary

- Used Terraform for Infrastructure as Code (IaC)
- Secured S3 access using CloudFront OAI
- Automated deployment with GitHub Actions
- Verified infrastructure manually, then automated
- Cleaned up AWS resources to avoid cost