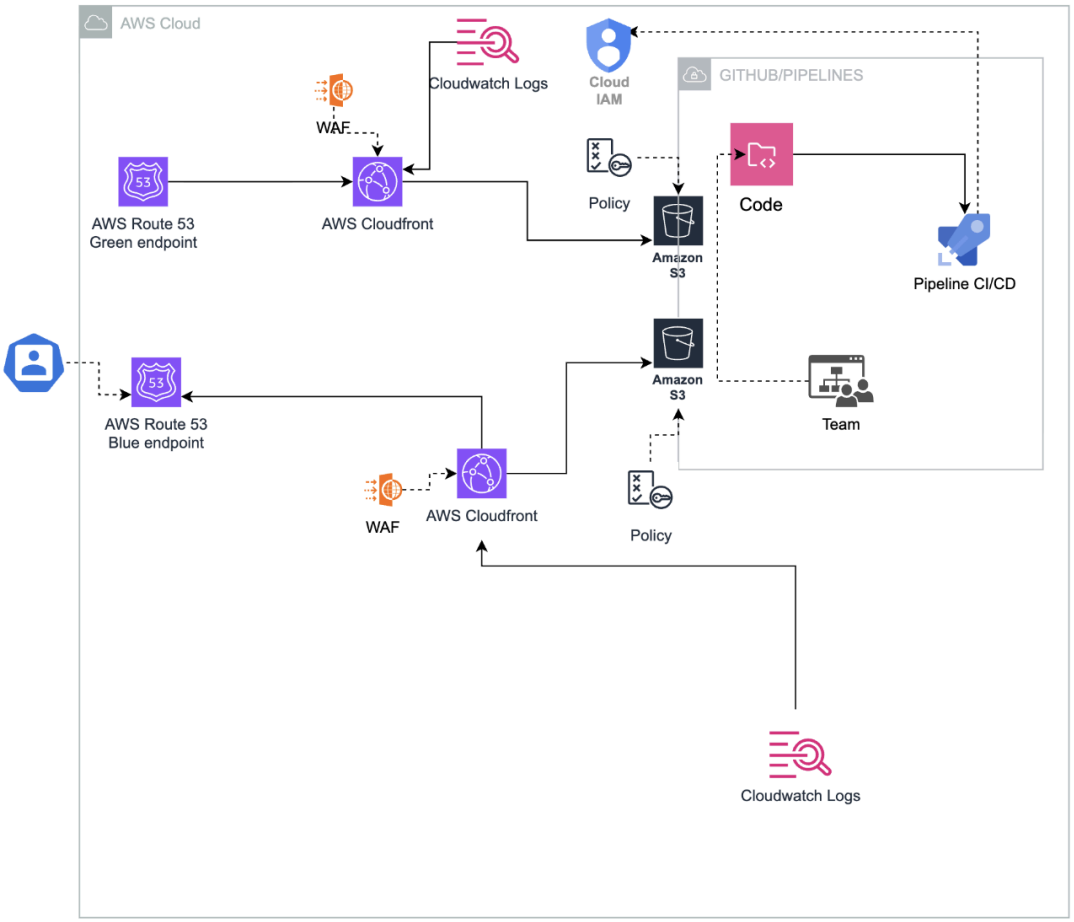
Repository explanation: https://github.com/crcasis/shakers-accenture-tech/tree/main

- Infra: Modules and environments.
- Diagram: The diagram for the technical assessment.
- Policy: Policies used for the s3 bucket and github actions pipelines
- Github actions: 2 github actions created. Deploy.yml to be able to deploy a new version of the application based of the requirements of the document. Terraform.yml to be able to plan and deploy infrastructure using terraform based in modules (i created 3 examples to explain the deployment workflow)
- Src folder: contains basic html website to explain how to upload the content to s3 buckets using github actions pipeline: deploy.yml
- Readme.md explanation of the repository

Based of the diagram:



Once the developer uploads a change to the src folder, the deploy.yml pipeline executes. It uses the authentication stored in secrets, employing OIDC to authenticate with AWS. It checks if the environment is blue or green, detects the active environment, and deploys to the inactive environment. It runs smoke tests to ensure the site is working. If everything is

OK, it switches to the active environment. If something fails, it performs an automatic rollback, invalidates CloudFront, saves evidence of the deployment, and sends a Slack notification regardless of whether it succeeds or fails.

```yaml
name: Deploy Blue/Green S3 Accenture

on:
  push:
    branches: [ "main" ]
    paths:
      - 'src/**'

permissions:
  id-token: write
  contents: read

env:
  APP_NAME: ${{ secrets.APP_NAME }}
  BUCKET_BLUE: ${{ secrets.BUCKET_BLUE }}
  BUCKET_GREEN: ${{ secrets.BUCKET_GREEN }}
  CONFIG_BUCKET: ${{ secrets.CONFIG_BUCKET }}
  ACTIVE_ENV_KEY: ${{ secrets.ACTIVE_ENV_KEY }}
  CLOUDFRONT_DIST_ID: ${{ secrets.CLOUDFRONT_DIST_ID }}
  SLACK_WEBHOOK: ${{ secrets.SLACK_WEBHOOK }}  Explain and Fix Problem (⇧⌘<)

jobs:
```

In the Terraform pipeline, once a developer pushes a change to the `infra` folder and the `dev/**`, `feature/**`, and `bugfix/**` branches, a `Terraform plan` is executed. When a change is made in the `infra` folder, `Terraform apply` is executed in `main`. This structure is commonly used in Terraform deployments to avoid executing changes that are still in the testing phase.

```
- name: Terraform Format Check
  run: |
    terraform fmt -check -recursive

- name: Terraform Validate
  run: |
    if [[ "${GITHUB_REF_NAME}" == "main" ]]; then
      cd envs/prod
    else
      cd envs/dev
    fi
    terraform validate

- name: Terraform Plan / Apply
  id: terraform
  run: |
    if [[ "${GITHUB_REF_NAME}" == "main" ]]; then
      echo "Branch main detected → running terraform apply"
      cd envs/prod
      terraform apply -auto-approve
    else
      echo "Non-main branch detected → running terraform plan"
      cd envs/dev
      terraform plan -out=tfplan
```

I've added WAF alongside CloudFront to provide advanced site protection, improve resilience, and reduce security risks.
WAF adds a configurable filter that blocks malicious traffic before it reaches CloudFront.
Another recommendation is to add AWS Shield Advanced. The standard version comes activated, but it's always advisable to use the advanced version. It prevents L3 and L4 attacks.

Observability
For observability, using Grafana, my idea would be to unify all the data from CloudWatch, S3 Logs, WAF, CloudFront, and Route 53 into a single dashboard.
It's quite simple for development teams to access even if they don't have access to AWS. And it allows sending alerts to Slack or Teams.

Failure Scenarios to Consider

There are errors that can occur, for example, permission issues with buckets. It would be necessary to modify the policy applied to the bucket.
Often, there are problems because the user or role being used lacks permissions to upload files, or the CloudFront invalidation process doesn't complete or execute. More permissions would need to be granted to the role/user being used. It's recommended to use Terraform for this rather than doing it manually.
In the case of certain errors in the frontend or at the pipeline level, they would need to be fixed, tested, a pull request (PR) created, and after validation, the entire process merged into the master branch.

In terms of architecture design, another option would be to have a single bucket with two CloudFront distributions. Each version would be placed in folder /blue or /green. The deployment's GitHub actions would need to be modified to make this work.

For the Accenture task I used Gemini, Windsurf and AWS documentation. I also recommend to use github wiki for devops documentation.