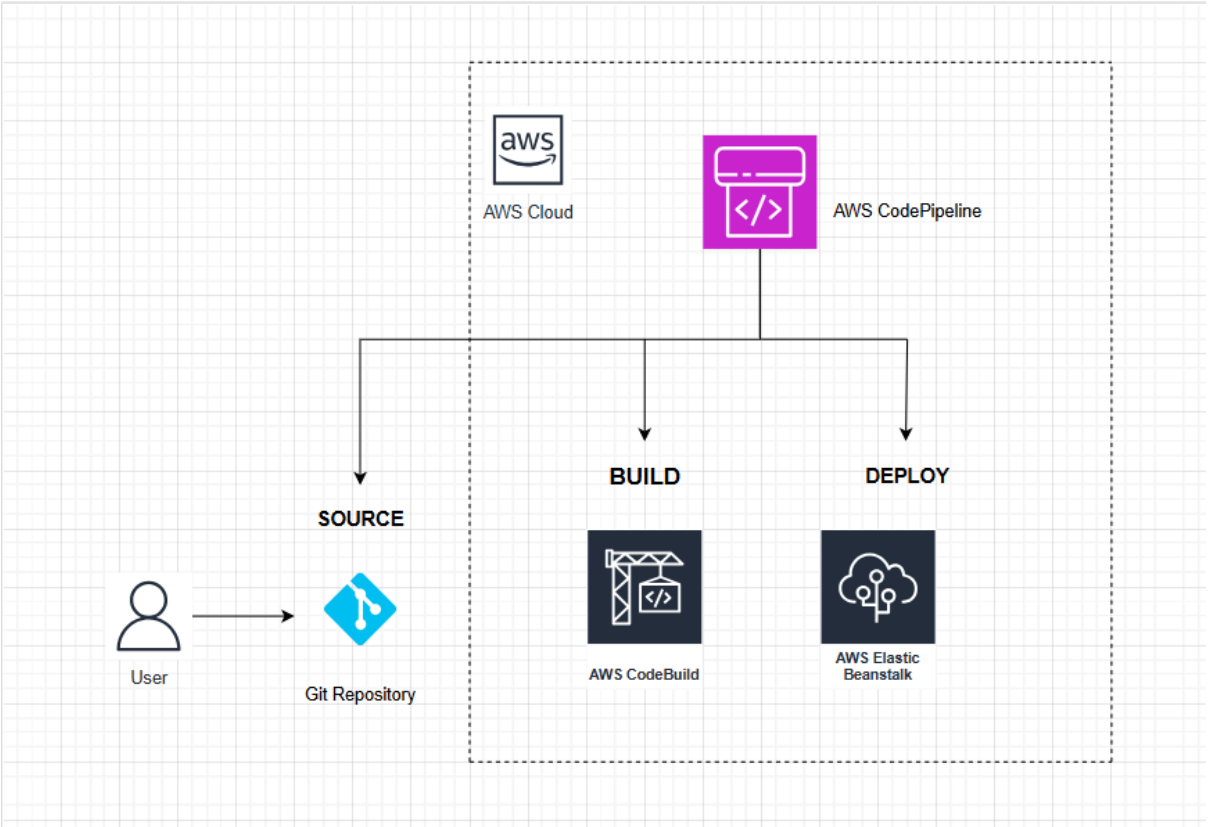
COMP4964 - ASSIGNMENT 4

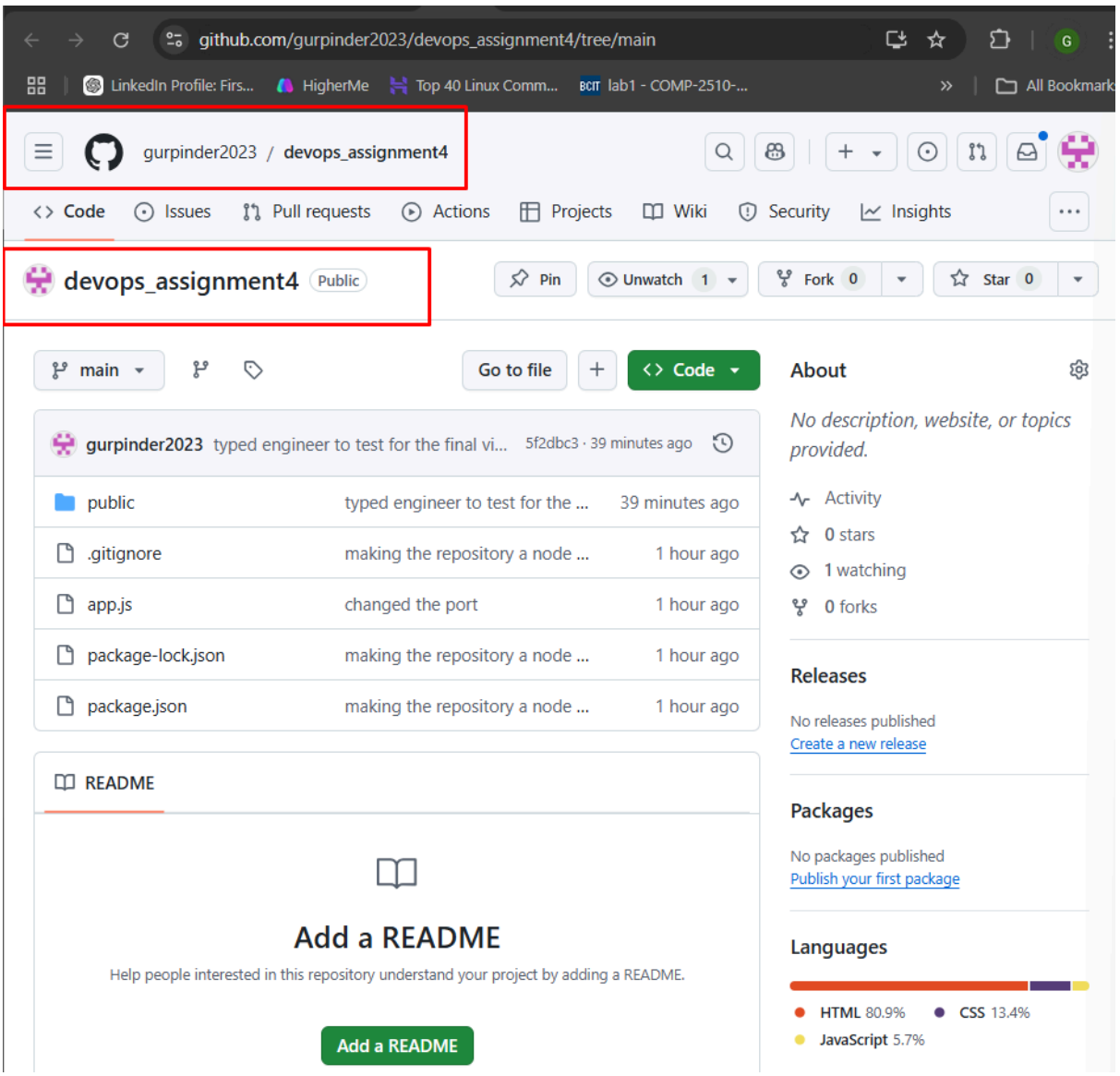
Architectural Diagram:



Steps to create this pipeline:

Step 1 : Creating the github repository

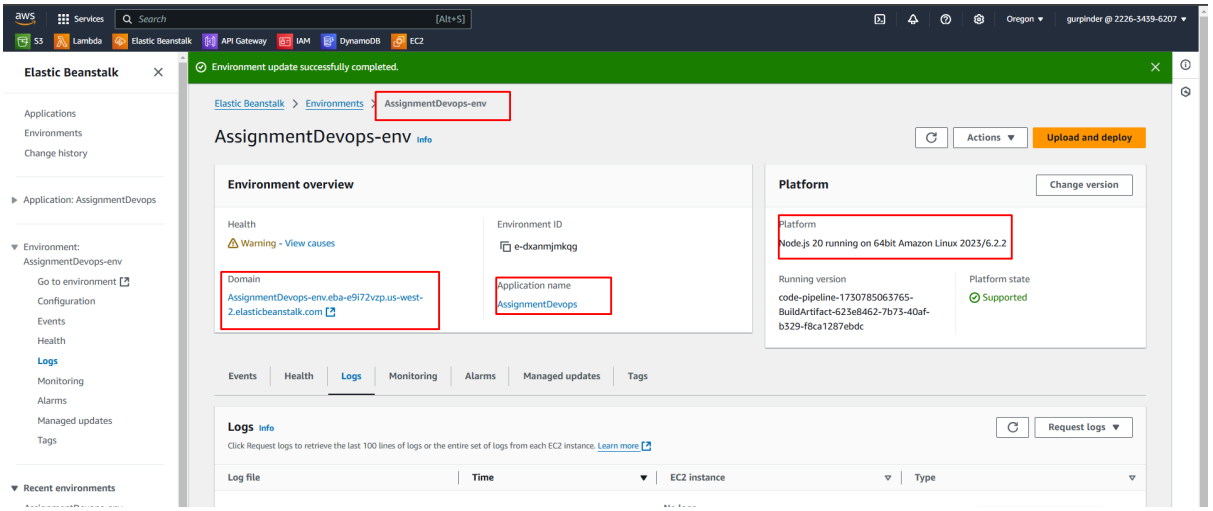
- Go to github and create a new repository and name it uniquely
- Clone the repository and add you code into it using any code editor like VS code .
- Commit and push the changes to the repository



Step 2 : Deploy Web app- AWS ElasticBeanstalk

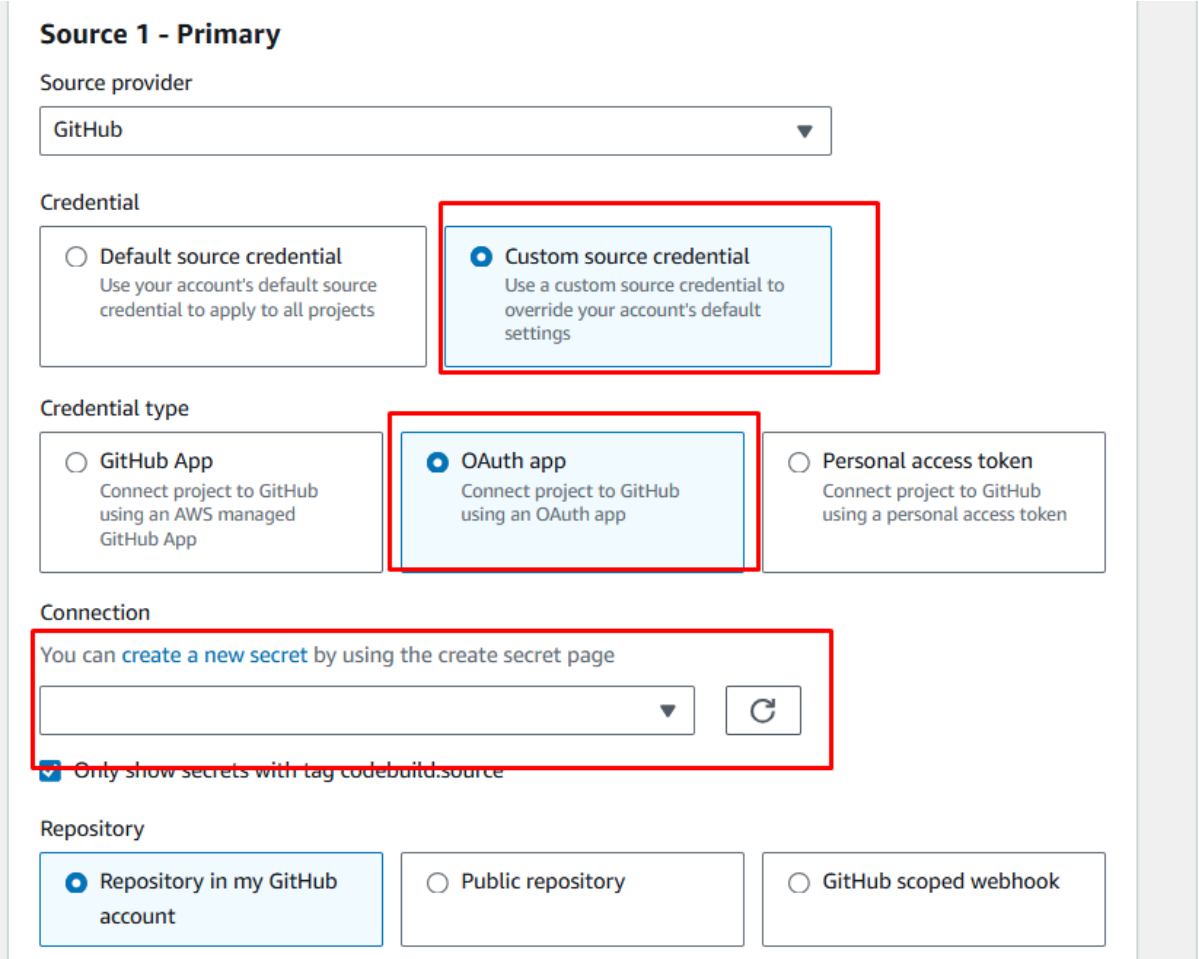
- Create Application: Open the AWS Elastic Beanstalk console, click on Create Application, select Web server environment, enter AssignmentDevops as the application name, and choose Node.js from the platform dropdown. Confirm the selections for Sample application and Single instance (free tier eligible), then click Next.
- Configure Service Access: Choose Use an existing service role for the Service Role and select the appropriate EC2 instance profile from the dropdown (e.g., aws-elasticbeanstalk-ec2-role). If no profiles are available, create a new IAM Role and return to this step to select it.
- Review and Submit: Click Skip to Review to accept default values, then review your choices on the summary page. Finally, click Submit to create the new environment.

- Monitor Deployment: Wait for the deployment to complete, indicated by a green banner with a checkmark at the top of the environment screen.

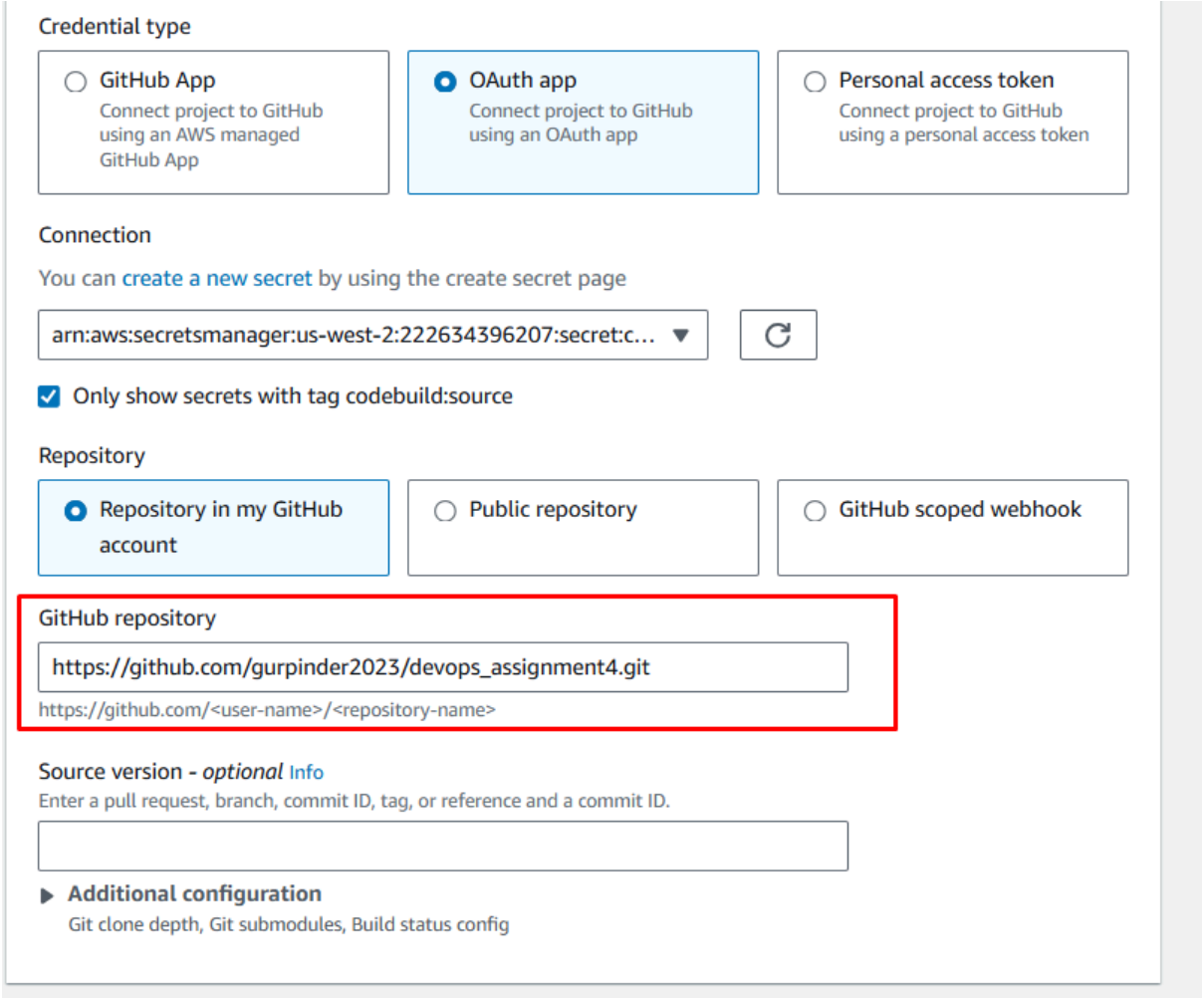


STEP 3 - Create Build Project - AWS CodeBuild

- In a new browser tab, open the AWS CodeBuild console.
- Choose the orange Create project button.
- In the Project name field, enter *Build-AssignmentDevops.*
- Select GitHub from the Source provider dropdown menu.
- Select custom source credentials and then select OAuthe app then you can create a new secret and configure the github or can choose exiting.



- A new browser tab will open asking you to give AWS CodeBuild access to your GitHub repo.
- Choose the green Authorize aws-codesuite button.
- Enter your GitHub password.
- Choose the orange Confirm button.
- Select Repository in my GitHub account.
- Enter *the url of your repository*
- After selecting your repo, your screen should look like this:



- For the environment make sure everything is like this :

Environment

Provisioning model Info

- (●) **On-demand**
  Automatically provision build infrastructure in response to new builds.
- ( ) **Reserved capacity**
  Use a dedicated fleet of instances for builds. A fleet's compute and environment type will be used for the project.

Environment image

- (●) **Managed image**
  Use an image managed by AWS CodeBuild
- ( ) **Custom image**
  Specify a Docker image

Compute

- (●) **EC2**
  Optimized for flexibility during action runs
- ( ) **Lambda**
  Optimized for speed and minimizes the start up time of workflow actions

Operating system

Amazon Linux

Runtime(s)

Standard

Image

aws/codebuild/amazonlinux2-x86_64-standard:5.0

Image version

Always use the latest image for this runtime version

[ ] Use GPU-enhanced compute

Image version

Always use the latest image for this runtime version

[ ] Use GPU-enhanced compute

Service role

- (●) **New service role**
  Create a service role in your account
- ( ) **Existing service role**
  Choose an existing service role from your account

Role name

codebuild-new testing-service-role

Type your service role name

▶ **Additional configuration**
Timeout, privileged, certificate, VPC, compute type, environment variables, file systems, auto-retry

- Create buildspec file : for this make sure you have package-json file in the github and also the script having the start variable

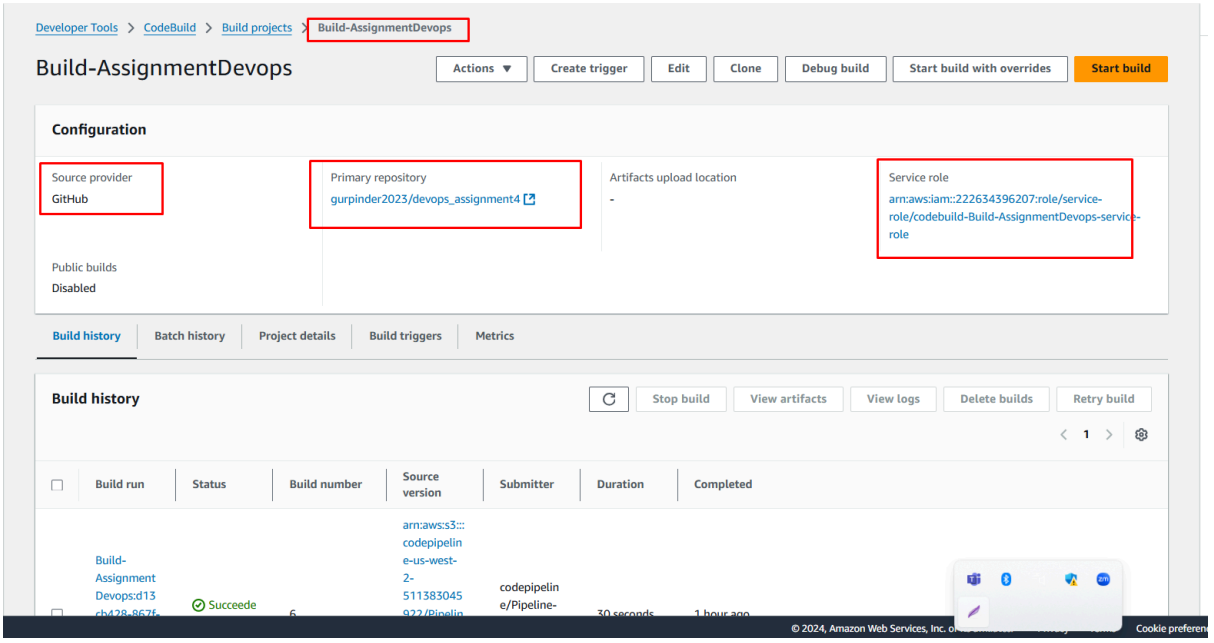
  Select Insert build commands.

  Choose Switch to editor.

  Replace the Buildspec in the editor with the code below:

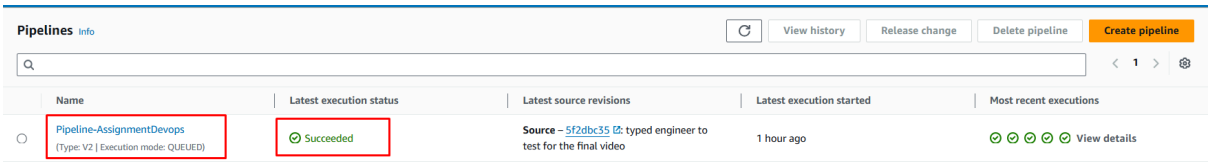
```
version: 0.2

phases:

    build:

        commands:

            - npm i --save

artifacts:

    files:

      - '**/*'
```

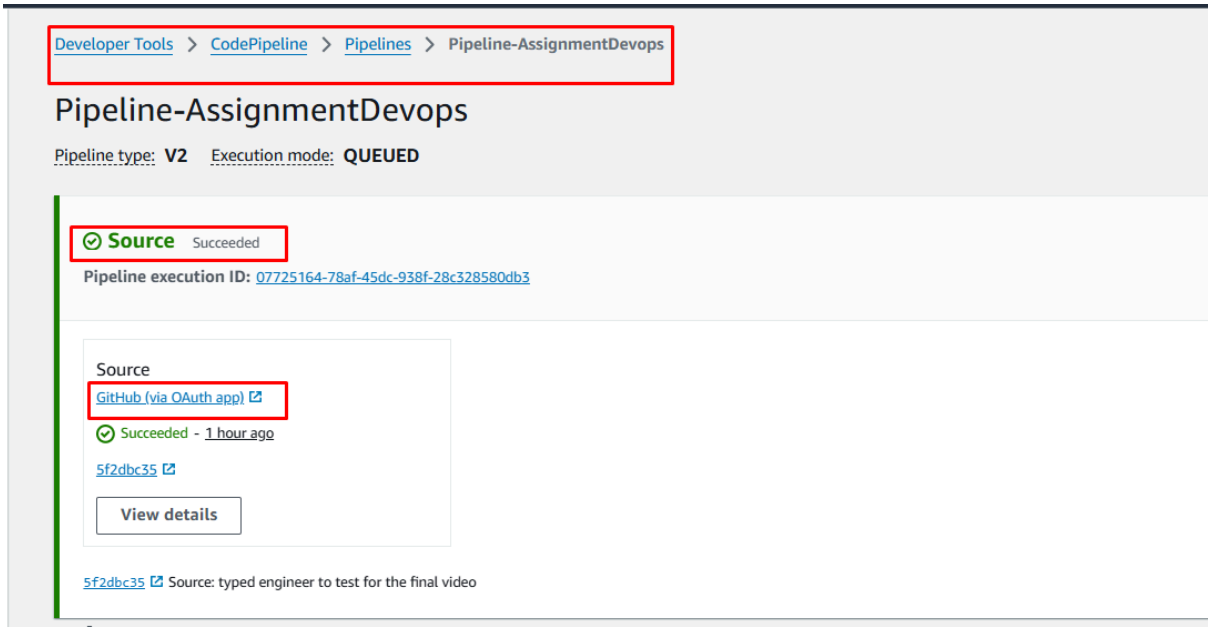
- Then create project and click start build

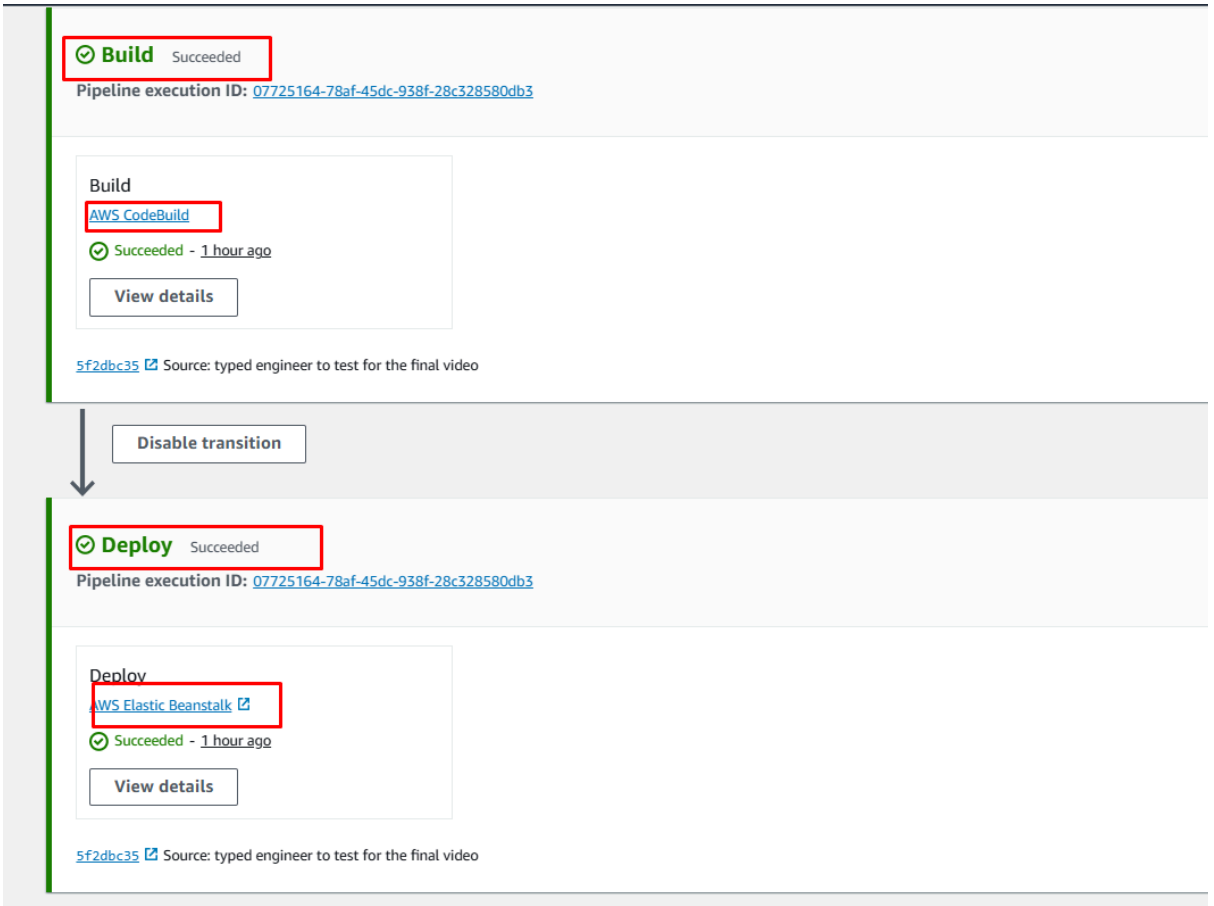


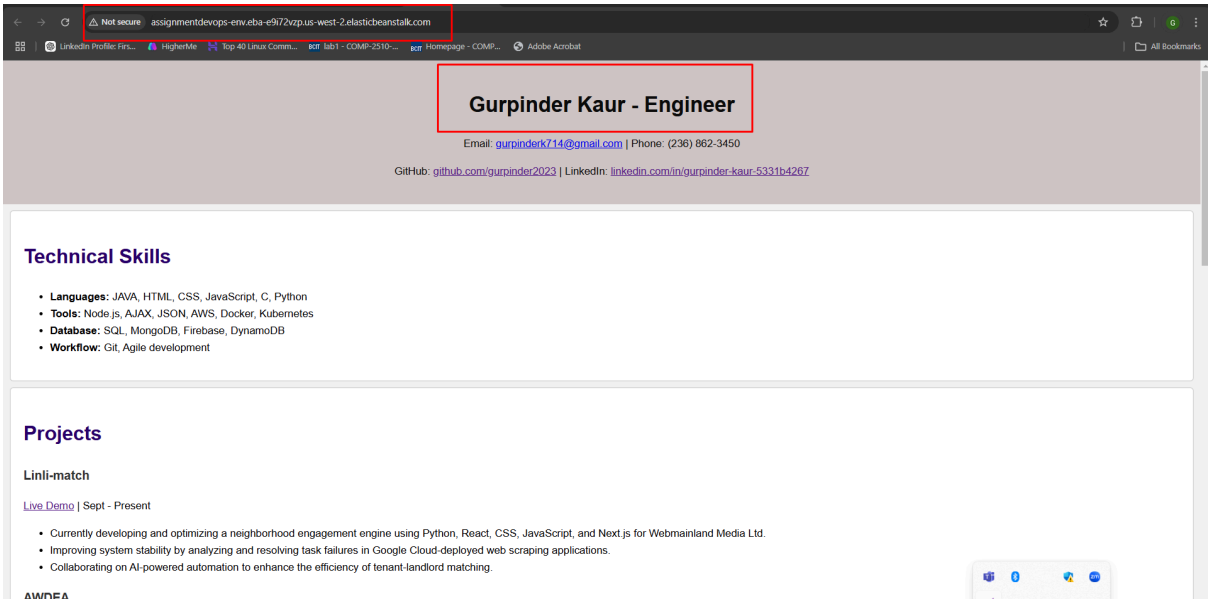
STEP 4 : Creating code pipeline

- Create a new pipeline and for the source select your github repository from the main branch that is created in step1
- For the build stage select AWS codeBuild and select the project that is build in step3
- For the deploy select AWS beanstalk and select the application created in step 2
- Once the pipeline is create you can test it throught the elastic beanstalk link






Build Succeeded
Pipeline execution ID: 07725164-78af-45dc-938f-28c328580db3
Build
AWS CodeBuild
Succeeded - 1 hour ago
View details
5f2dbc35 Source: typed engineer to test for the final video
Disable transition
Deploy Succeeded
Pipeline execution ID: 07725164-78af-45dc-938f-28c328580db3
Deploy
AWS Elastic Beanstalk
Succeeded - 1 hour ago
View details
5f2dbc35 Source: typed engineer to test for the final video


Not secure assignmentdevops-env.eba-e9i72vzp.us-west-2.elasticbeanstalk.com
Gurpinder Kaur - Engineer
Email: gurpinderk714@gmail.com | Phone: (236) 862-3450
GitHub: github.com/gurpinder2023 | LinkedIn: linkedin.com/in/gurpinder-kaur-5331b4267
Technical Skills
Languages: JAVA, HTML, CSS, JavaScript, C, Python
Tools: Node.js, AJAX, JSON, AWS, Docker, Kubernetes
Database: SQL, MongoDB, Firebase, DynamoDB
Workflow: Git, Agile development
Projects
Linli-match
Live Demo | Sept - Present
Currently developing and optimizing a neighborhood engagement engine using Python, React, CSS, JavaScript, and Next.js for Webmainland Media Ltd.
Improving system stability by analyzing and resolving task failures in Google Cloud-deployed web scraping applications.
Collaborating on AI-powered automation to enhance the efficiency of tenant-landlord matching.

FINAL TESTING:

So you can test your pipeline by changing anything in the code and then committing and pushing it on the github and you will see the changes will be reflected in the hosted application on the github

CODEBASE:

I created a simple Node.js application with HTML and CSS files in a public directory, using the Express framework to serve these pages. I set up an `app.js` file that serves static files and renders the main HTML file.

```
JS app.js   <> index.html   .gitignore   {} package.json
JS app.js > port
1   const express = require('express');
2   const app = express();
3   const path = require('path');
4   const port = 8080;
5
6   // Serve static files from the public directory
7   app.use(express.static(path.join(__dirname, 'public')));
8
9   app.listen(port, () => {
10      console.log(`Server running on port ${port}`);
11  });
12
13
```

```
DEVOPS_ASSIGNMENT4
  > node_modules
  v public
    <> index.html
    # style.css
  .gitignore
  JS app.js
  {} package-lock.json
  {} package.json
```

Troubleshooting Guide:

Issue: **HTTP 500 (Bad Gateway) Error** when accessing the Elastic Beanstalk application URL.

Description: Upon initial deployment, accessing the application URL resulted in a 500 Bad Gateway error.

Cause: The application was configured to listen on port 3000, but Elastic Beanstalk defaults to port 8080. This mismatch caused the server to be unreachable.

Solution:

1. Open the `app.js` file (or whichever file starts your server).
2. Change the port to `8080` to align with Elastic Beanstalk's default