**Deploying a node.js application using Aws ElasticBeanstalk and Aws CodePipeline**

In this project we will learn to deploy a Node.js Application using Elastic Beanstalk and we will do CI/CD using Aws CodePipeline.

Why we are using Elastic Beanstalk?

Elastic Beanstalk allows you to automate the process of deploying your applications onto the AWS cloud platform. Elastic Beanstalk itself provides an easy way to deploy and manage applications in the cloud without worrying about the infrastructure setup.

Prerequisites for this project:

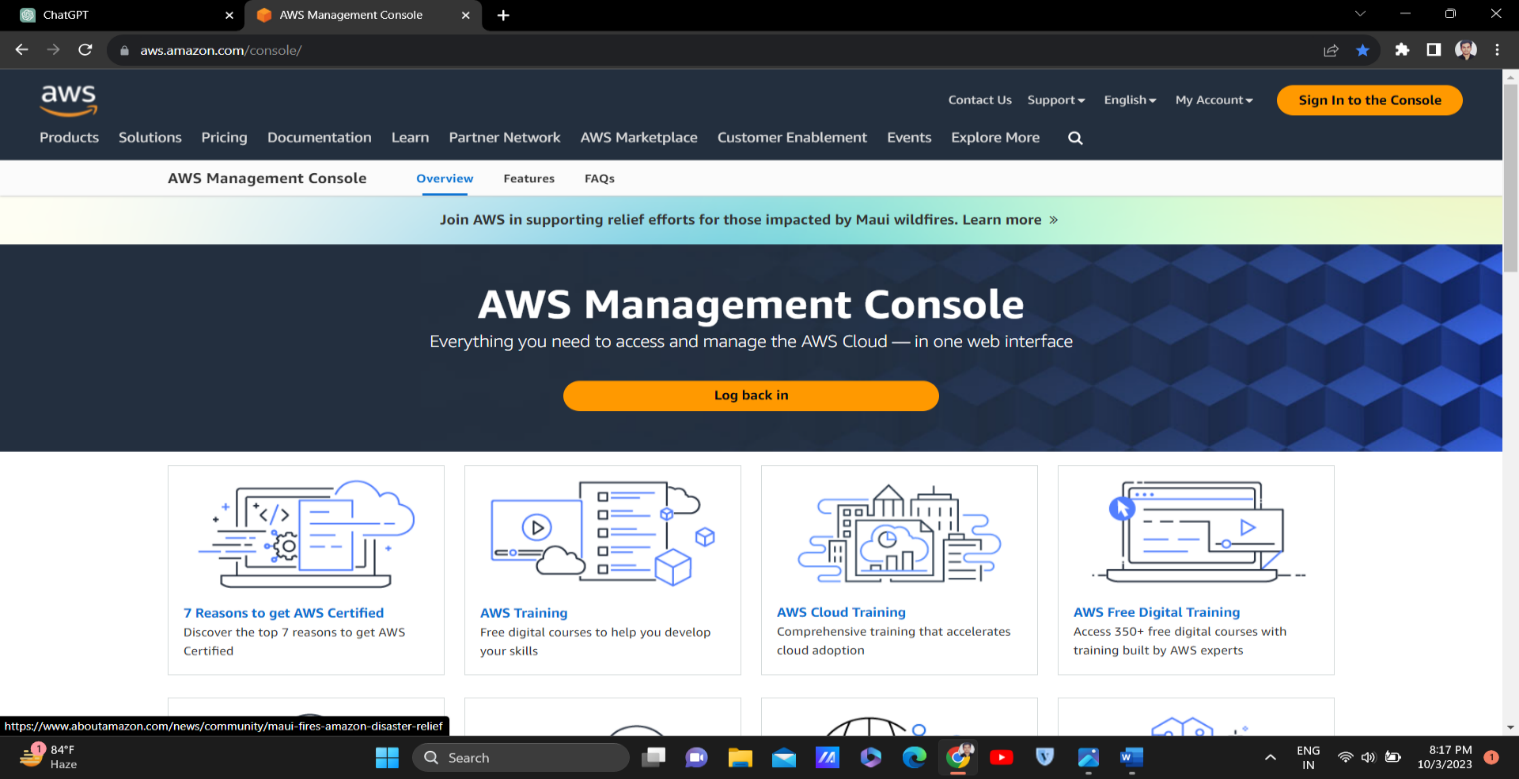
1. Download and Install Git
2. Download a sample node.js code from this website: <https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/tutorials.html>

Now let us start with our project.

**Step 1. Login to AWS Console**

<https://aws.amazon.com/console/> using this website login to AWS Console, if you do not

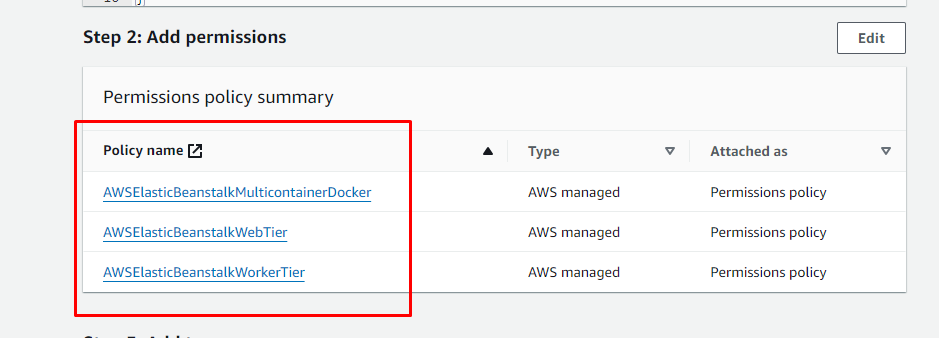
have an AWS account please sign up first and then login.



**Step 2. Create an IAM role**

Creating an IAM role for EC2 instances and attaching specific permissions is necessary when you are deploying applications to AWS Elastic Beanstalk environments because Elastic Beanstalk uses EC2 instances to run and manage your applications. The permissions specified in these IAM roles allow the EC2 instances to interact with other AWS services on your behalf, enabling Elastic Beanstalk to function properly.

* After logging into AWS Console, Search for IAM.
* Go to IAM and create a role.
* Trusted entity type – AWS Service > Use Case – EC2 > Role Details : Name – ElasticBeanstalk-Project > Next
* Now Add these permissions: AWS Elastic beanstalk multicontainer, Aws elastic beanstalk webtier, Aws Elastic beanstalk worker tier



* Now Review everything and click on Create Role. Role has been created.

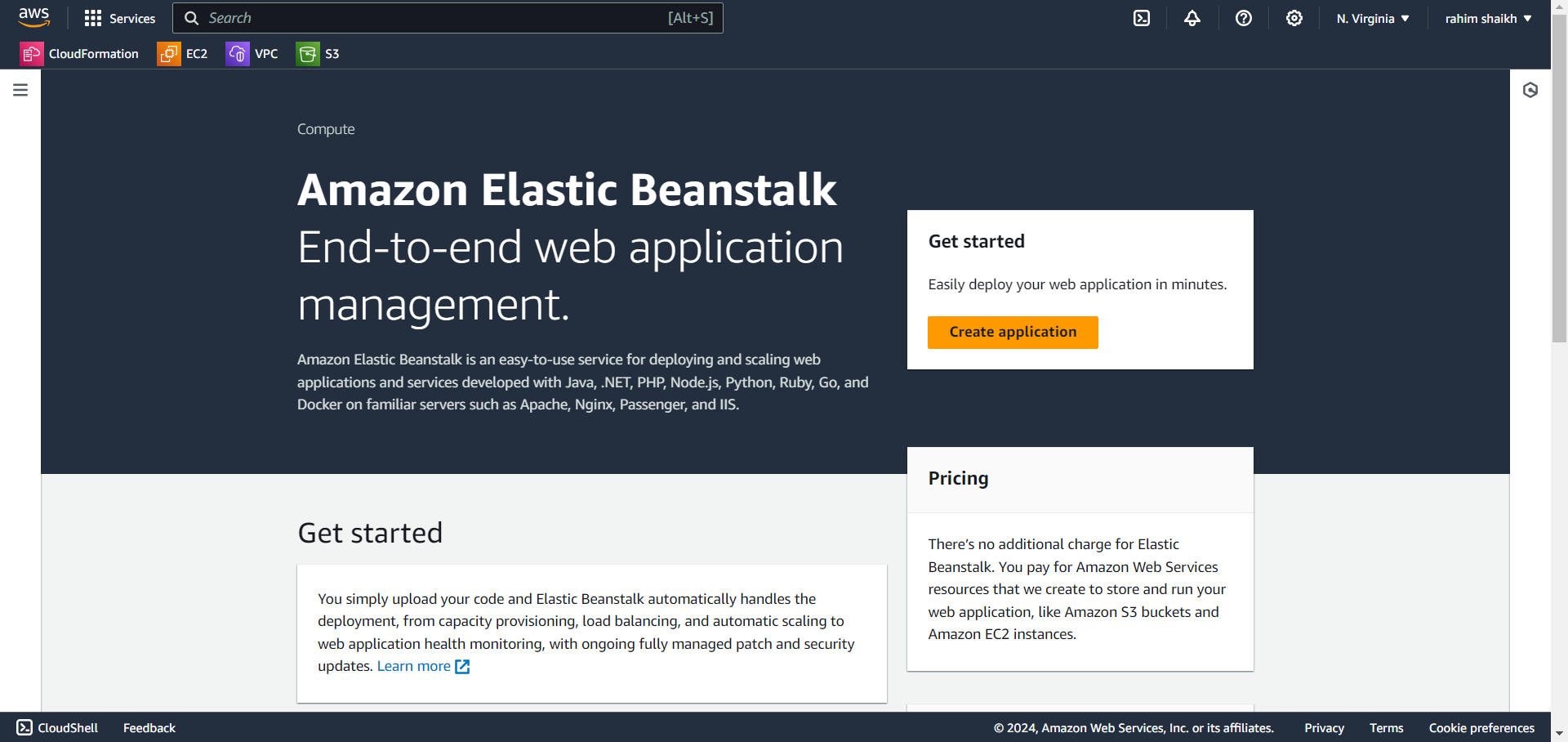
**Step 3. Create a Deployment Environment**

In continuous deployment, you need a deployment environment which can be either an EC2 server or a Docker container or Elastic Beanstalk. In our case I am going to use Elastic Beanstalk. Elastic Beanstalk serves as the deployment environment for hosting your Node.js application. Elastic Beanstalk abstracts away the complexities of managing infrastructure, allowing you to focus on deploying and managing your application code.

With Elastic Beanstalk, you do not need to worry about provisioning servers, configuring load balancers, or scaling your application. Instead, Elastic Beanstalk handles these tasks automatically based on your application's needs and the configuration you provide.

By choosing Elastic Beanstalk as your deployment environment, you benefit from its simplicity, scalability, and ease of use, making it an excellent choice for deploying and managing web applications, including those built with Node.js.

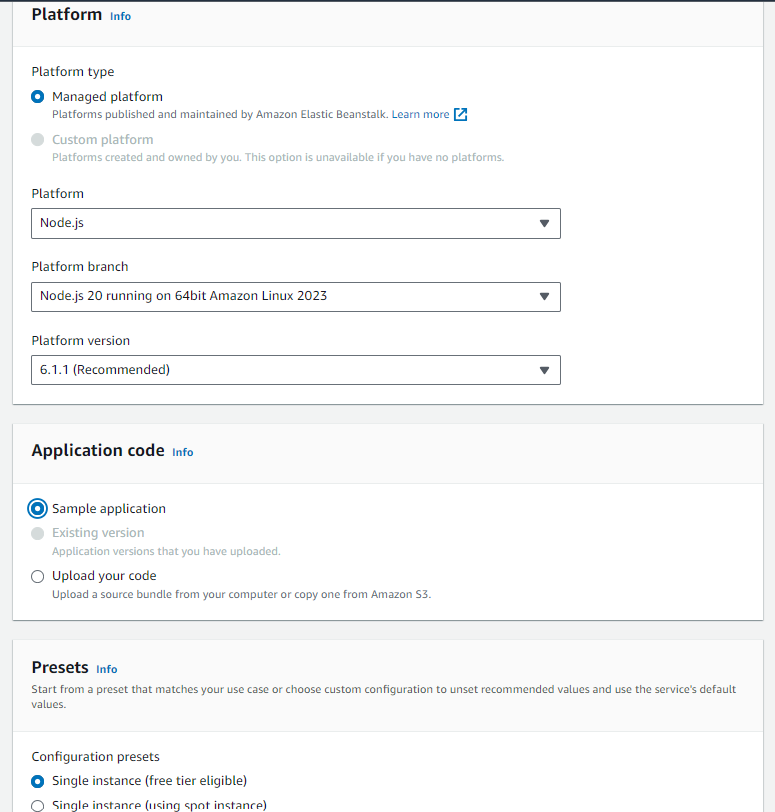
* Search for Elastic Beanstalk in the search bar on the homepage of AWS Console.
* An interface shown below will appear.



* Now click on Create Application.

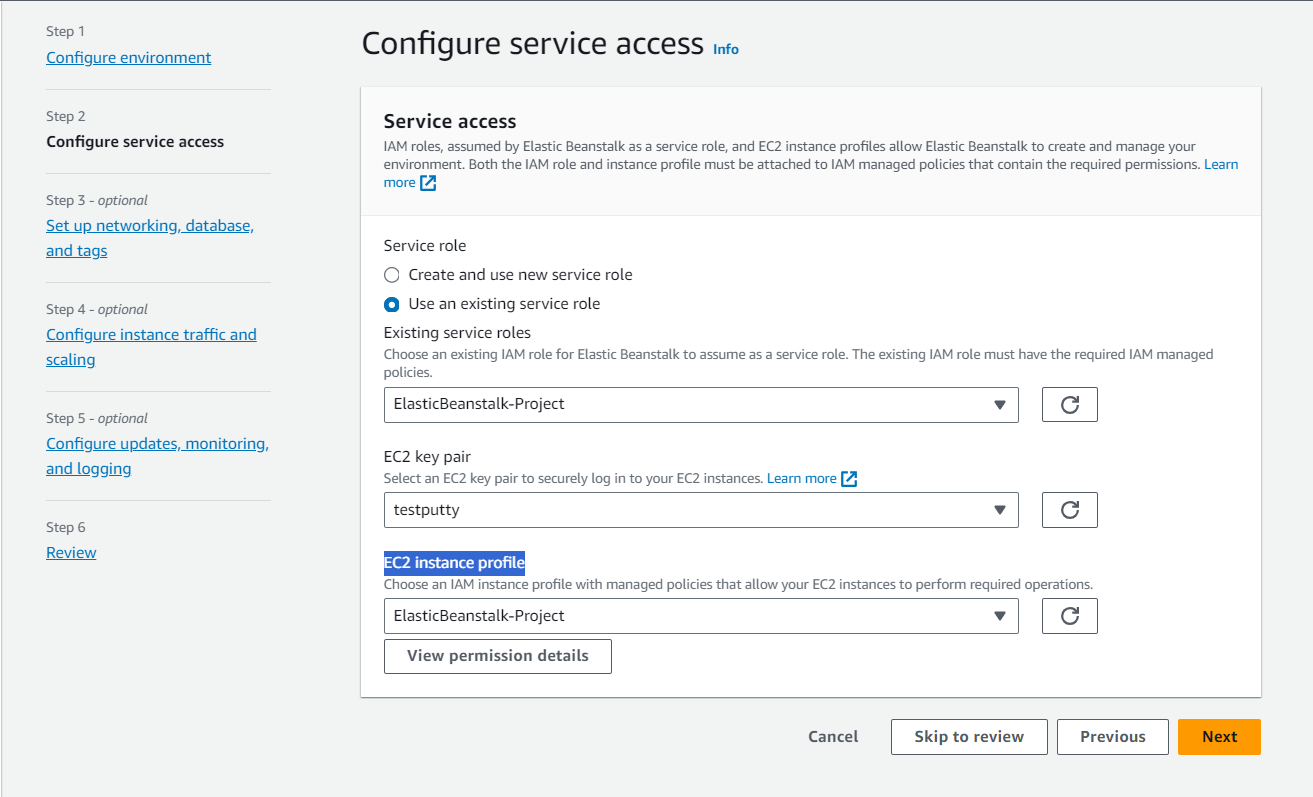
Step 1. Configure Environment

* Environment Tier > Web Server Environment
* Application Information: Application name > Give any name for eg. Beanstalk-project
* Environment Information: Environment name > It will appear automatically based on application name in our case it should be Beanstalk-project-env.
* Platform: Platform type > Managed platform, Platform > Node.js



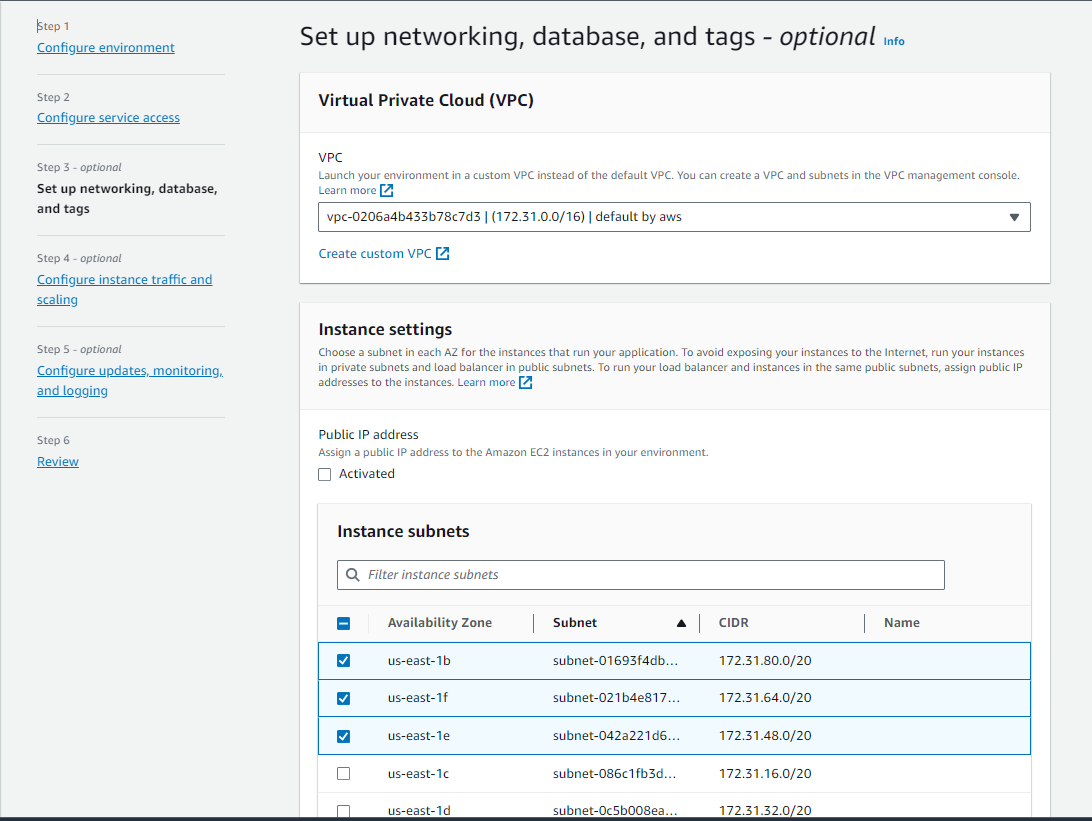
Step 2. Configure service access

* Service Access: Service Role > Use an existing service role, Existing Service Role > ElasticBeanstalk-Project (IAM role that we have created in Step 2), EC2 KeyPair > testvmpem.pem (I already have an existing key pair so I have selected that, if you do not have it, you can create one in EC2 section), EC2 instance profile > ElasticBeanstalk-Project.



Step 3. Set up networking, database, and tags

* VPC > Select the default vpc by aws
* Instance Setting: Public IP Address> Activated, Instance Subnets > Select any 2-3 AZ (this is for the high availability of the application)
* Leave other setting by default and skip the step 4 and step 5 and jump on step 6 review and click on submit



Once you have the environment, you can see the health check status and check the configurations of the environment. It will show the node.js configuration items you have selected during setup.

**Step 4. Create a IAM User**

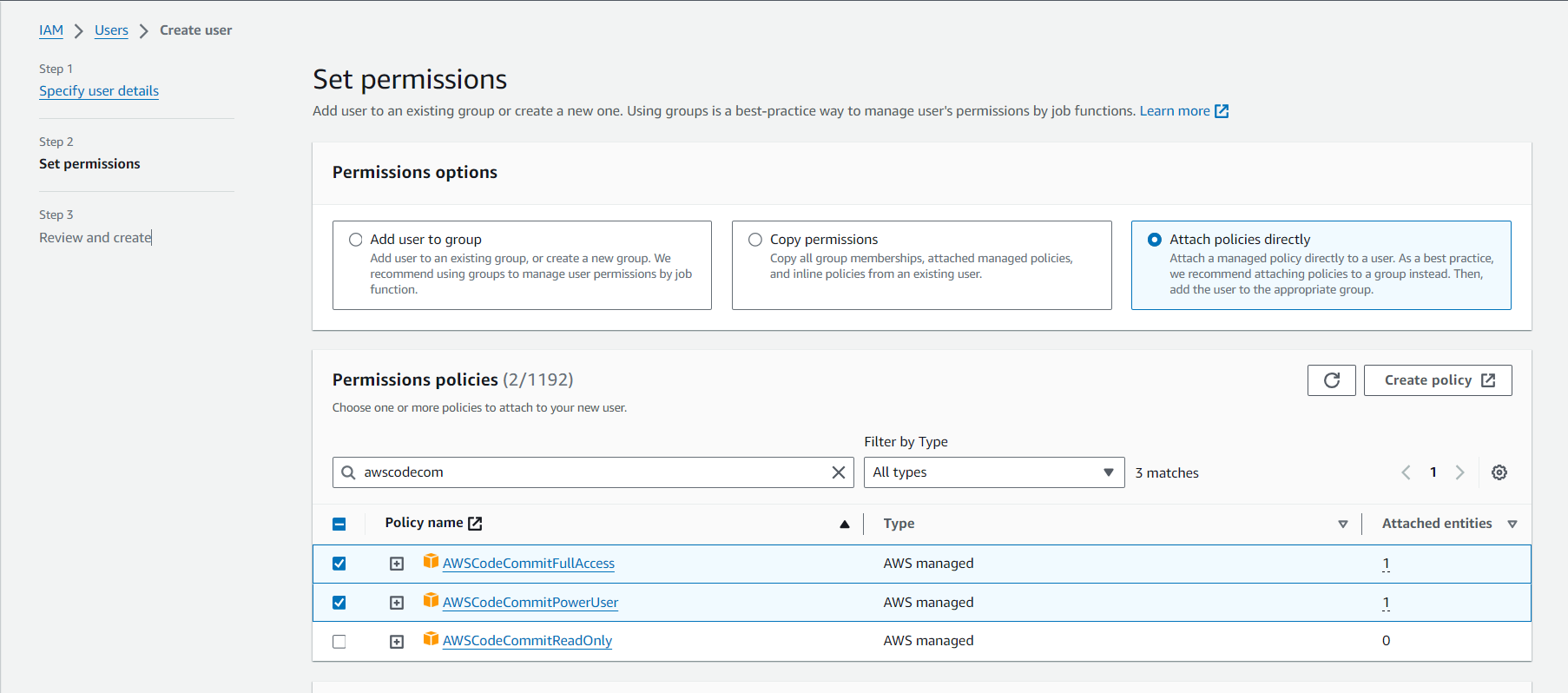
In the search bar, type IAM go to user section and create a user.

Step 1. Specify User Details

* User Details: User Name > Give any name for eg. Beanstalk-Project-User, click on next.

Step 2. Set Permissions

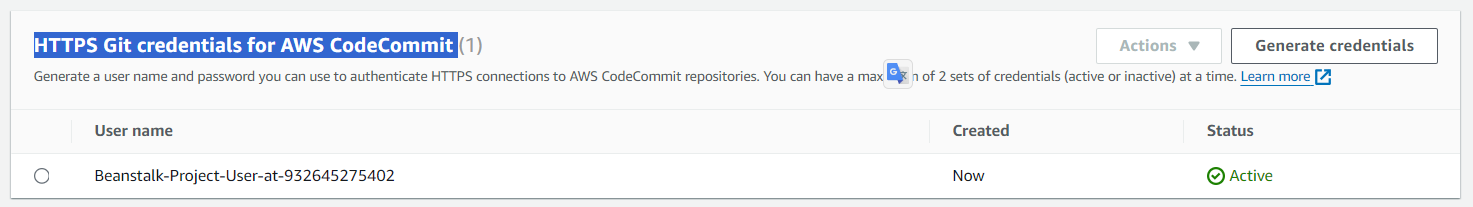
* Permission Options > Attach policies directly and attach the below ticked policies.



Step 3. Review and Create

* Now review everything and click on create user and IAM user is created.

Now go to the user that we have created and click on the security credentials, scroll down, and find HTTPS Git credentials for AWS CodeCommit and click on Generate Credentials and as soon as you click on generate credentials a pop up will appear, it will show that the credentials are generated and don’t forget to download it.

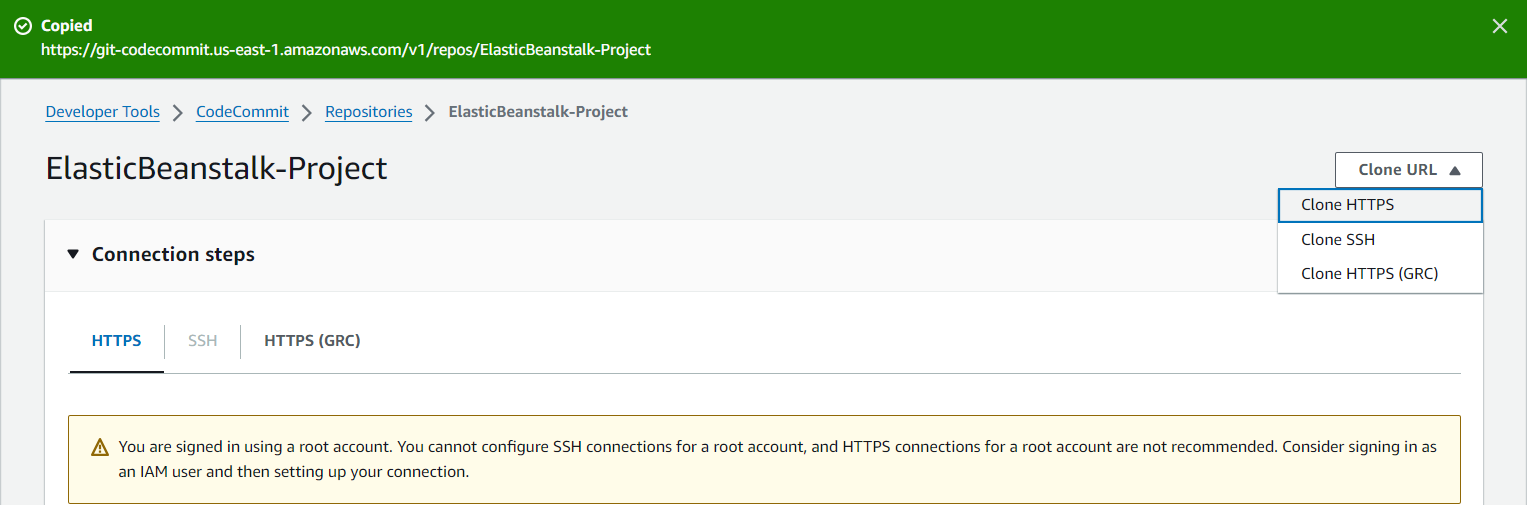


**Step 4 : Create code commit Repository**

Creating a CodeCommit repository is a crucial step in setting up a CI/CD pipeline in our project because we will keep our code in CodeCommit. CodeCommit is a fully-managed source control service provided by AWS, like Git repositories but hosted on AWS infrastructure.

I have a sample Node.js application that I am planning to deploy on Elastic Beanstalk. Additionally, I will configure CodePipeline to automate the deployment process. This pipeline will fetch the source code from my CodeCommit repository and perform actions accordingly. Whenever there is a change in the source code, it will trigger automatic deployment to the servers. This setup ensures a smooth deployment process, eliminating manual interventions and speeding up the delivery of changes to my application.

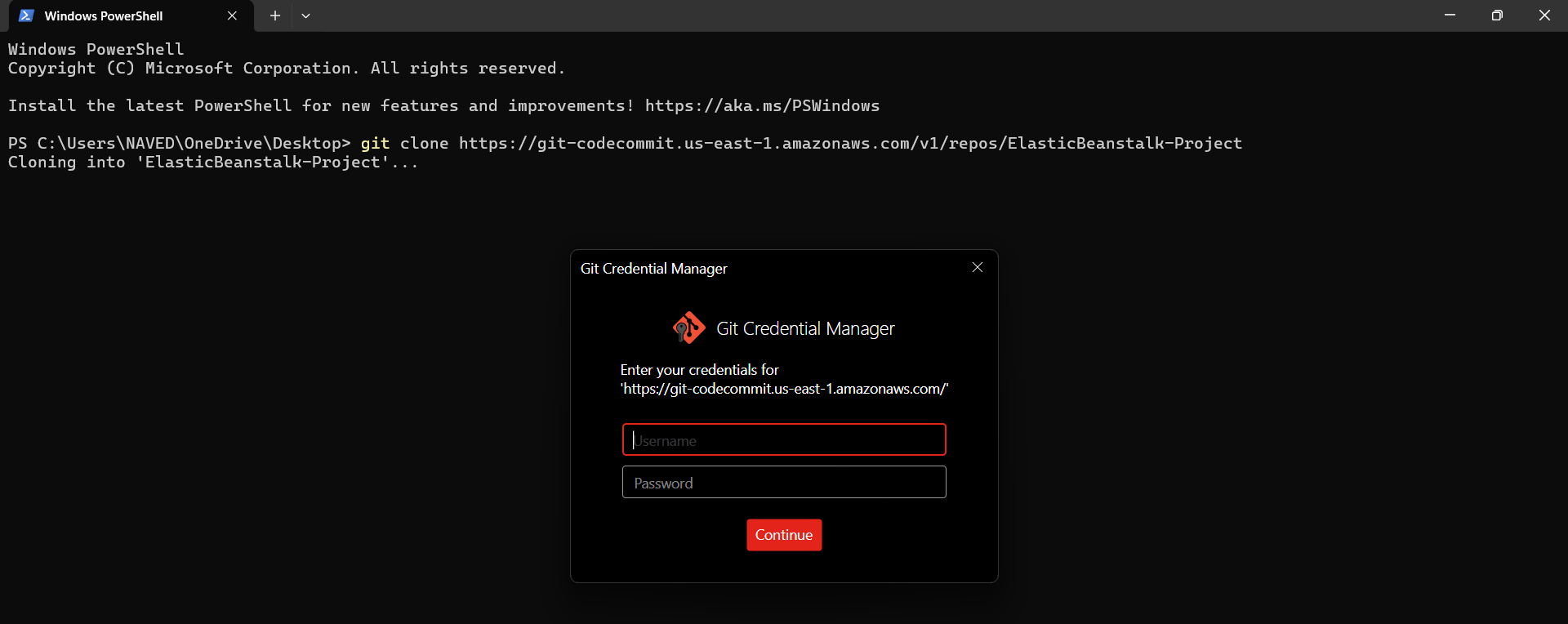
* Search for CodeCommit, Click on Create repository, Repository name > ElasticBeanstalk-Project and click on create.
* Now click on Clone URL > Clone HTTPS



**Step 5. Copy the Code to CodeCommit Repository**

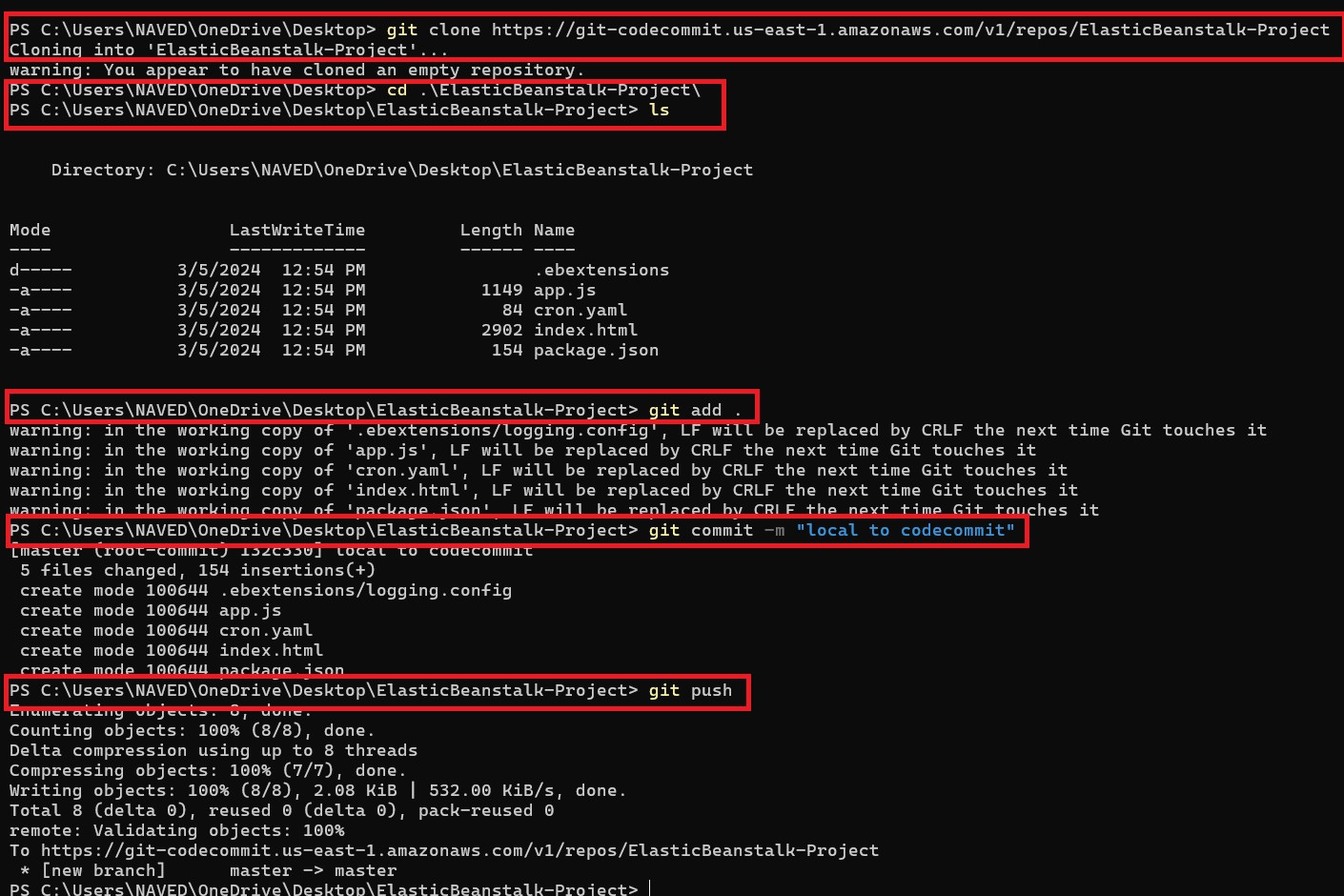
As we have already downloaded the code in our local machine (if not then you can download it from the link mentioned in prerequisite on page 1) we need to copy the code from the local machine to our CodeCommit repository that we have created just now.

* Open your terminal (CMD in windows)
* Check git is present on your computer, Command : git --version
* If yes then perform the next command
* Command: git clone url\_you *just*\_copied (from the above image Clone HTTPS)
* Now enter the Credentials you generated after creating a user for CodeCommit, which is mentioned in the excel file that we have downloaded after generating the credentials.
* Enter the username and the password.

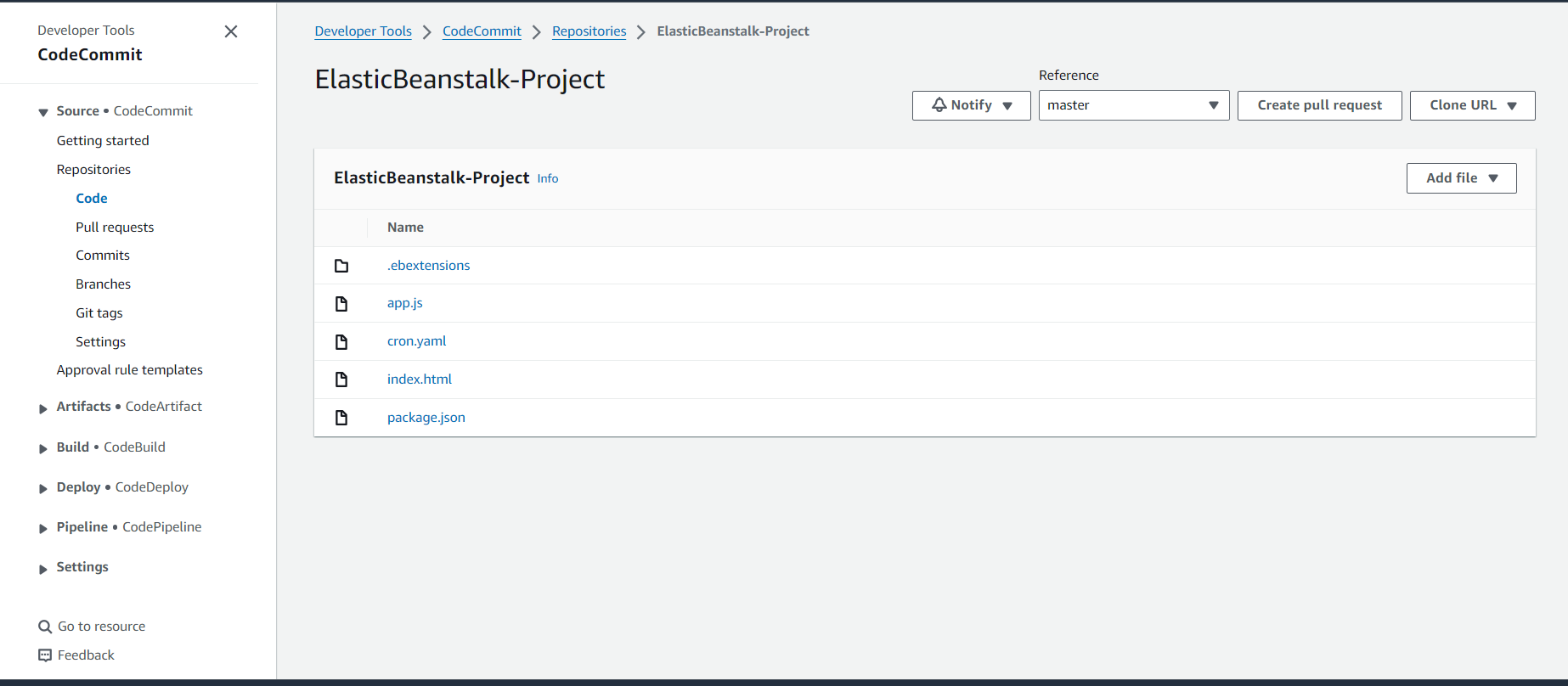


* Once you have cloned the CodeCommit Repo into your local, you will see a folder named “ElasticBeanstalk-Project” on your desktop.
* Now copy the all the content from the node,js folder and paste into ElasticBeanstalk-Project folder.
* After pasting, open command prompt and perform the command shown below to push your local code to CodeCommit repository
* Commands:

1. First, we will go in that folder: cd .\ElasticBeanstalk-Project\ (changes directory to the specified path)
2. Check if all the files are present in that folder: ls (lists all the files and folders)
3. Do “git add .” to add all the files.
4. Do “git commit -m “local to repo”” to commit the changes.
5. Finally, “git push” to push all the files to the CodeCommit Repo



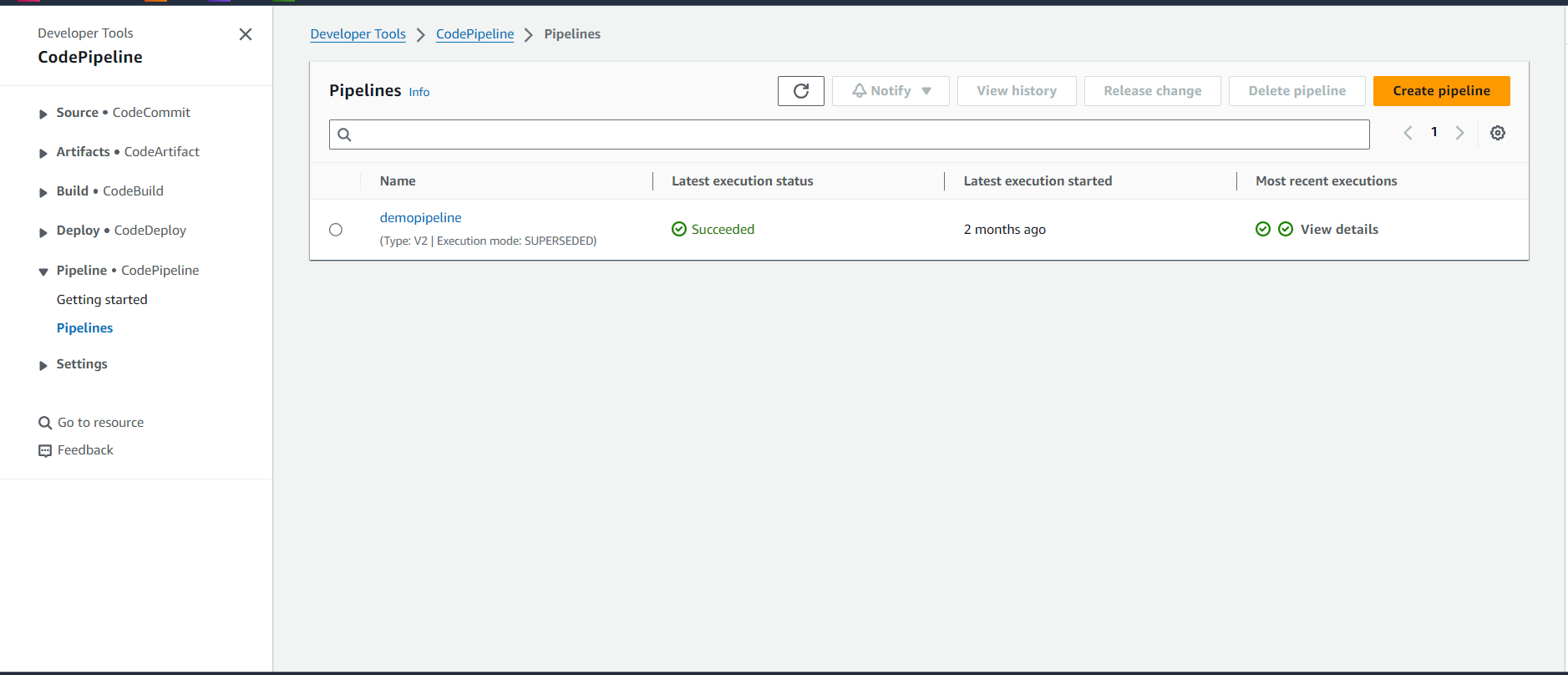
Now we can see that all the files and folder from our local machines is pushed into our CodeCommit Repository.



**Step 6. Create a CodePipeline**

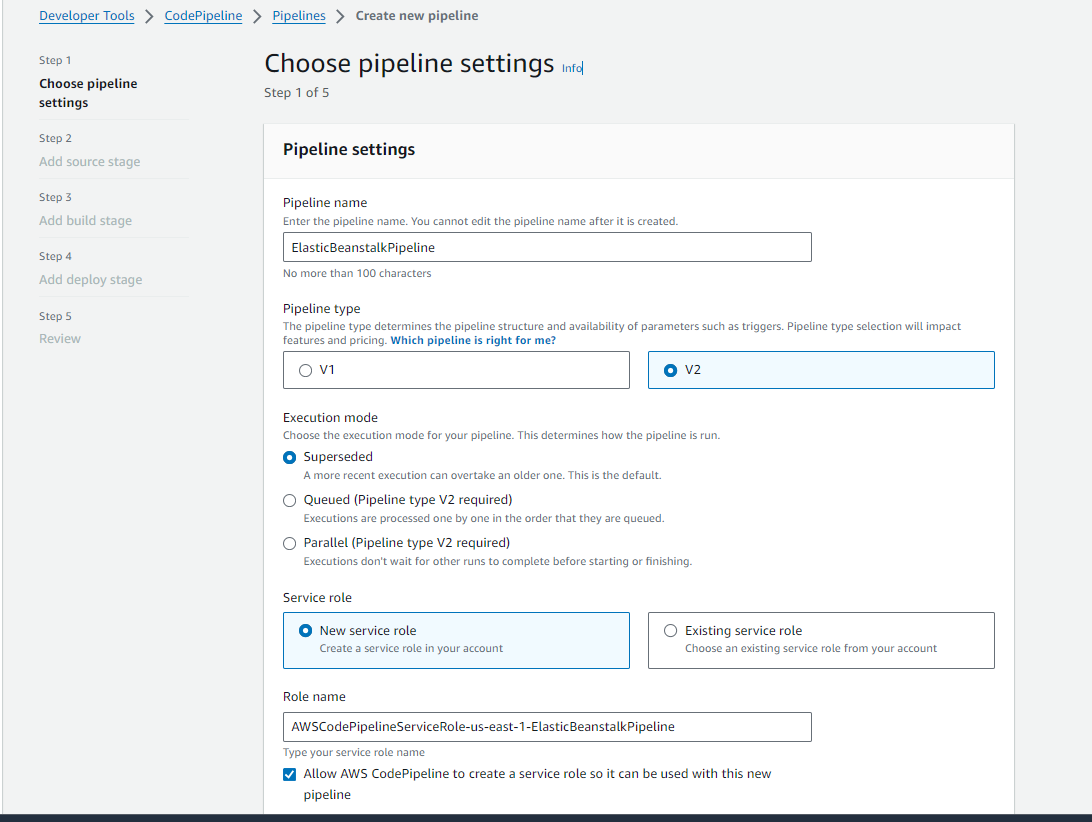
In this project, AWS CodePipeline is used to automate the deployment process of a Node.js application to Elastic Beanstalk. This automation saves time and effort by automatically detecting changes in the CodeCommit repository and triggering deployments. CodePipeline ensures consistency across deployments, improves efficiency by eliminating manual interventions, provides visibility into the deployment process, and scales with the project's needs. Overall, CodePipeline streamlines the deployment workflow and enhances the reliability of deployments.

Search for CodePipeline in the search bar, Now click on Create pipeline



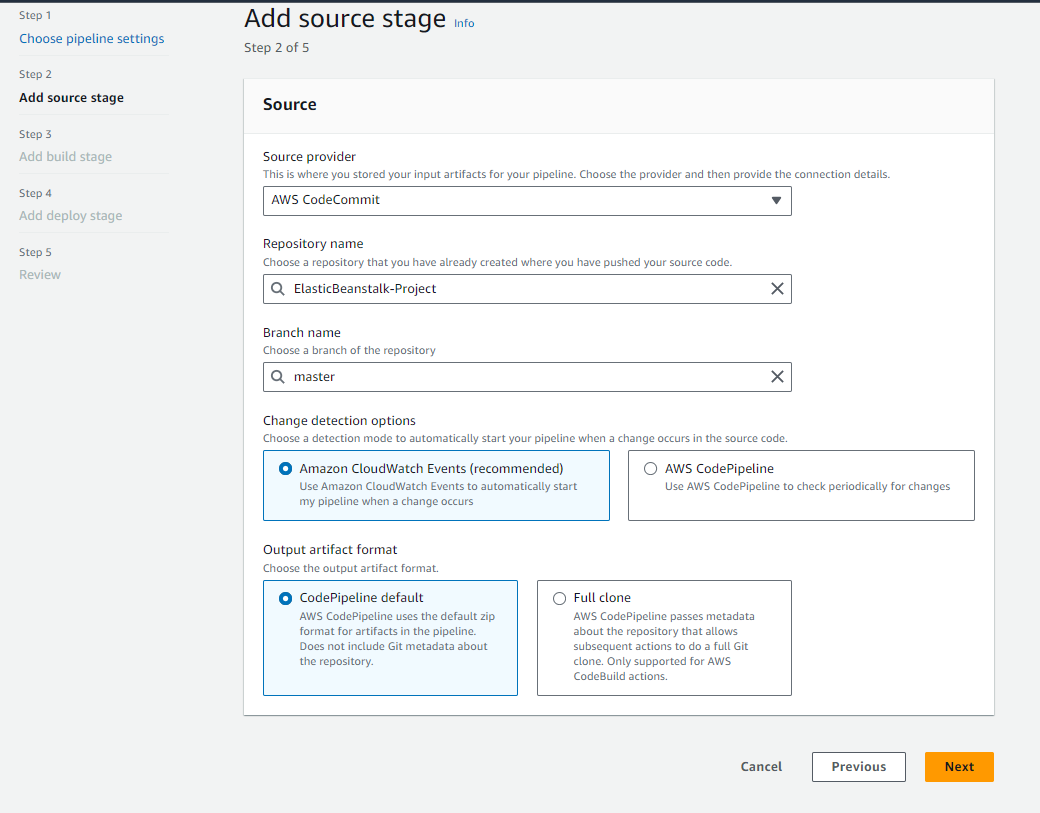
Step 1. Choose pipeline settings

* Pipeline Settings: Pipeline name: ElasticBeanstalkPipeline (or any name of your choice), Service role> New service role and click on next, keep other settings by default.



Step 2. Add source stage

* Source: Source provider > CodeCommit (as our source code is in codecommit), Repository name > ElasticBeanstalk-Project, Branch name > master and keep other options by default and click on next.

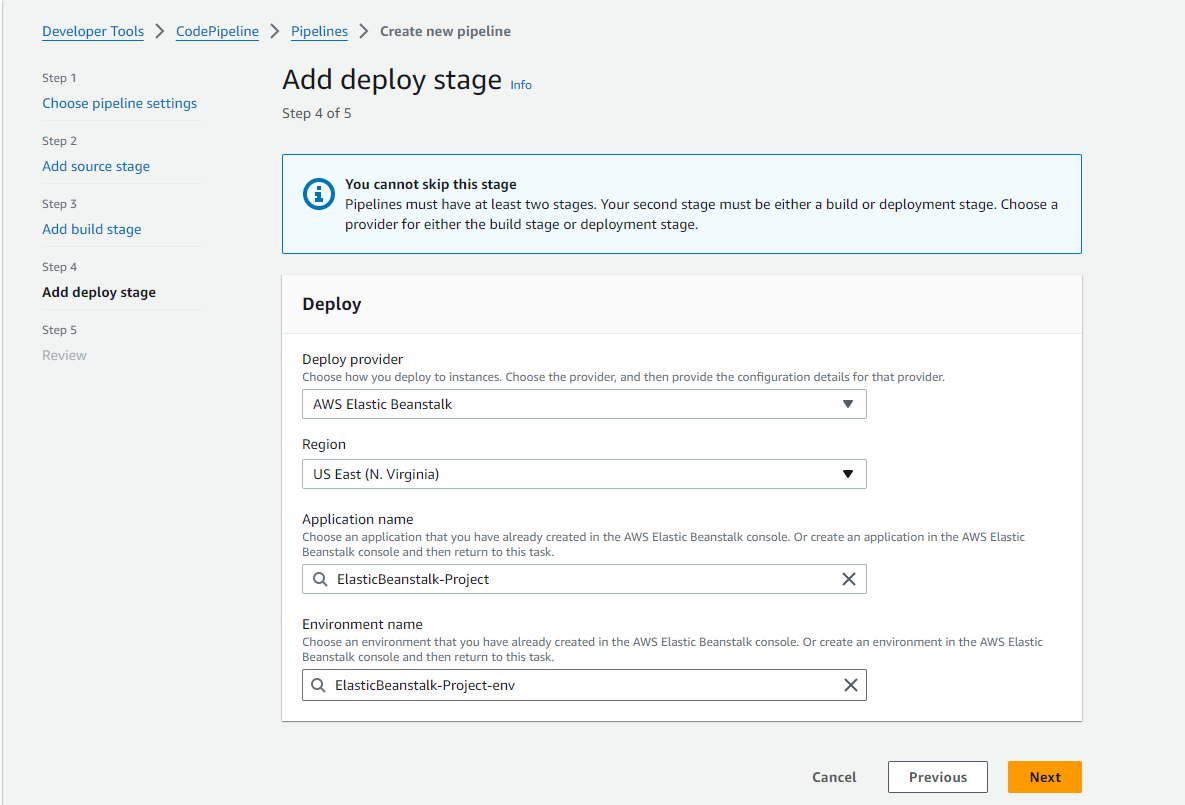


Step 3. Add build stage

* Directly click on skip build stage.

Step 4. Add deploy stage

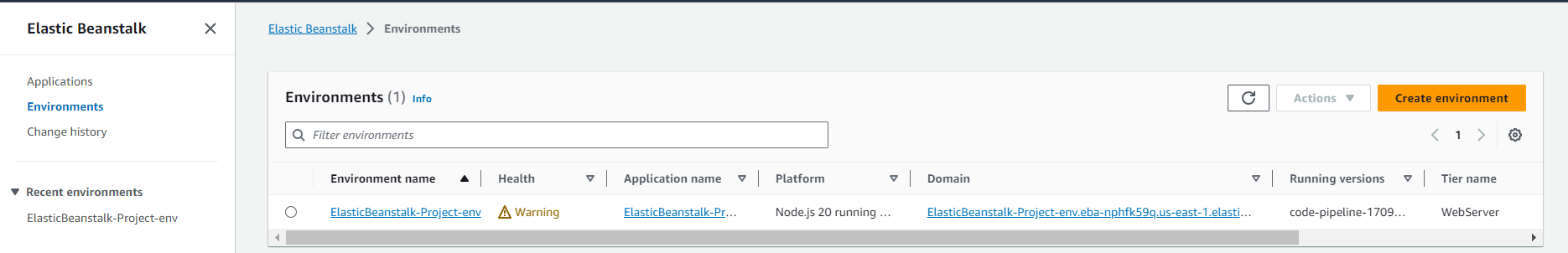
* Deploy: Deploy provider> AWS ElasticBeanstalk, Application name > ElasticBeanstalk-Project, Environment name > ElasticBeanstalk-Project-env and click on next.



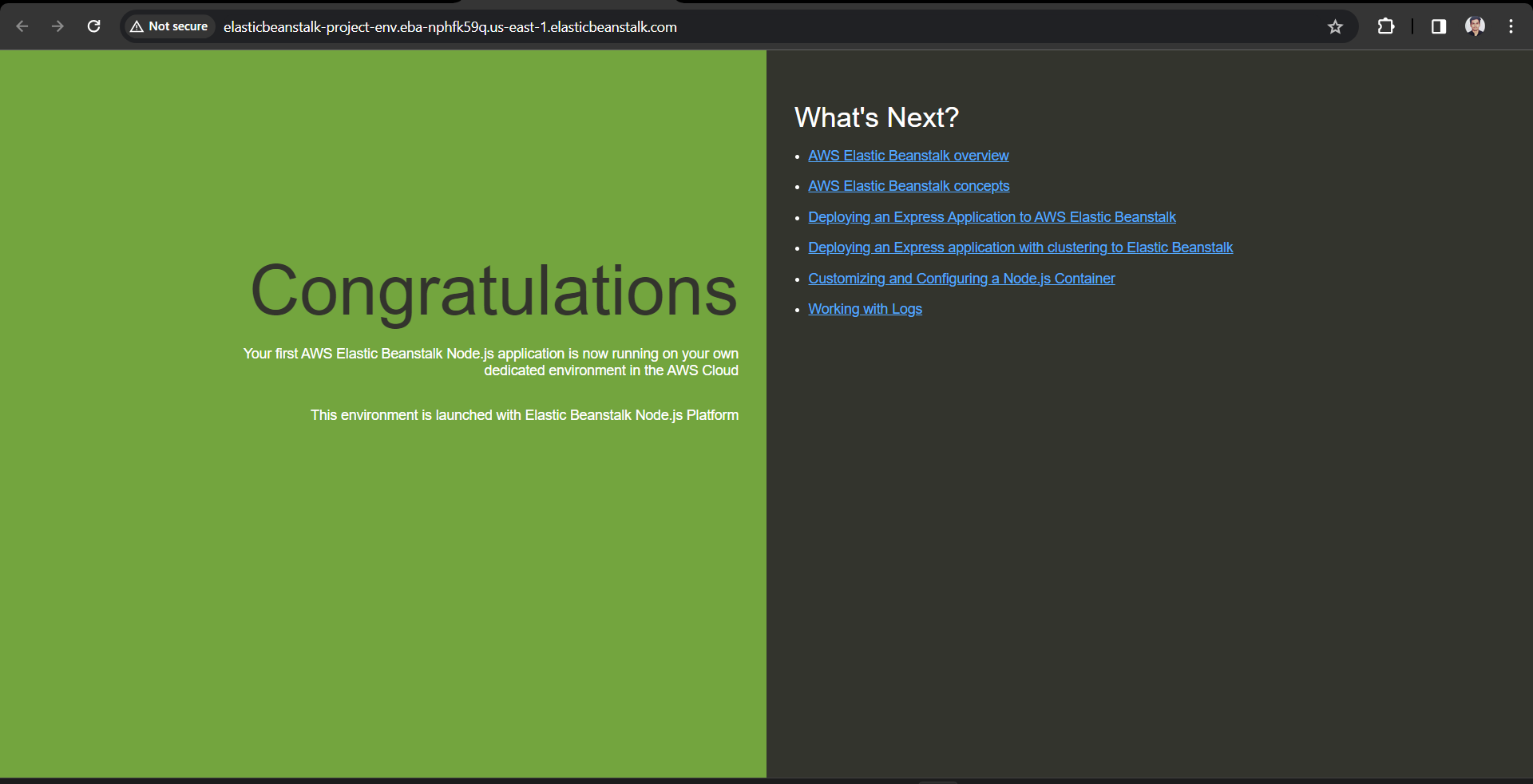
Step 5. Review

* After reviewing everything click on create pipeline.

Now wait until the pipeline is created, once completed go to elasticbeanstalk there you will find a domain, click on that domain.



Once you click on the domain you will see an interface as shown below.



Congratulations!!! We have successfully deployed our Node.js application using Aws ElasticBeanstalk, Aws CodeCommit, and Aws CodePipeline.

**Step 7. Commit changes and update**

We can update our code and commit the changes to the repository. CodePipeline will detect your updated sample code and then automatically initiate deploying it to your EC2 instance via Elastic Beanstalk. Let us update our index.html page, commit changes, and after some time, you will see the pipeline being updated. It will automatically pull the updated code and will start reflecting the changes.

Let us see how to do it:

Step 1. Open your CMD

Step 2. Go to the folder that we have cloned i.e. ElasticBeanstalk-Project in which our code is kept.

Step 3. Type the command: notepad index.html to open the index.html file.

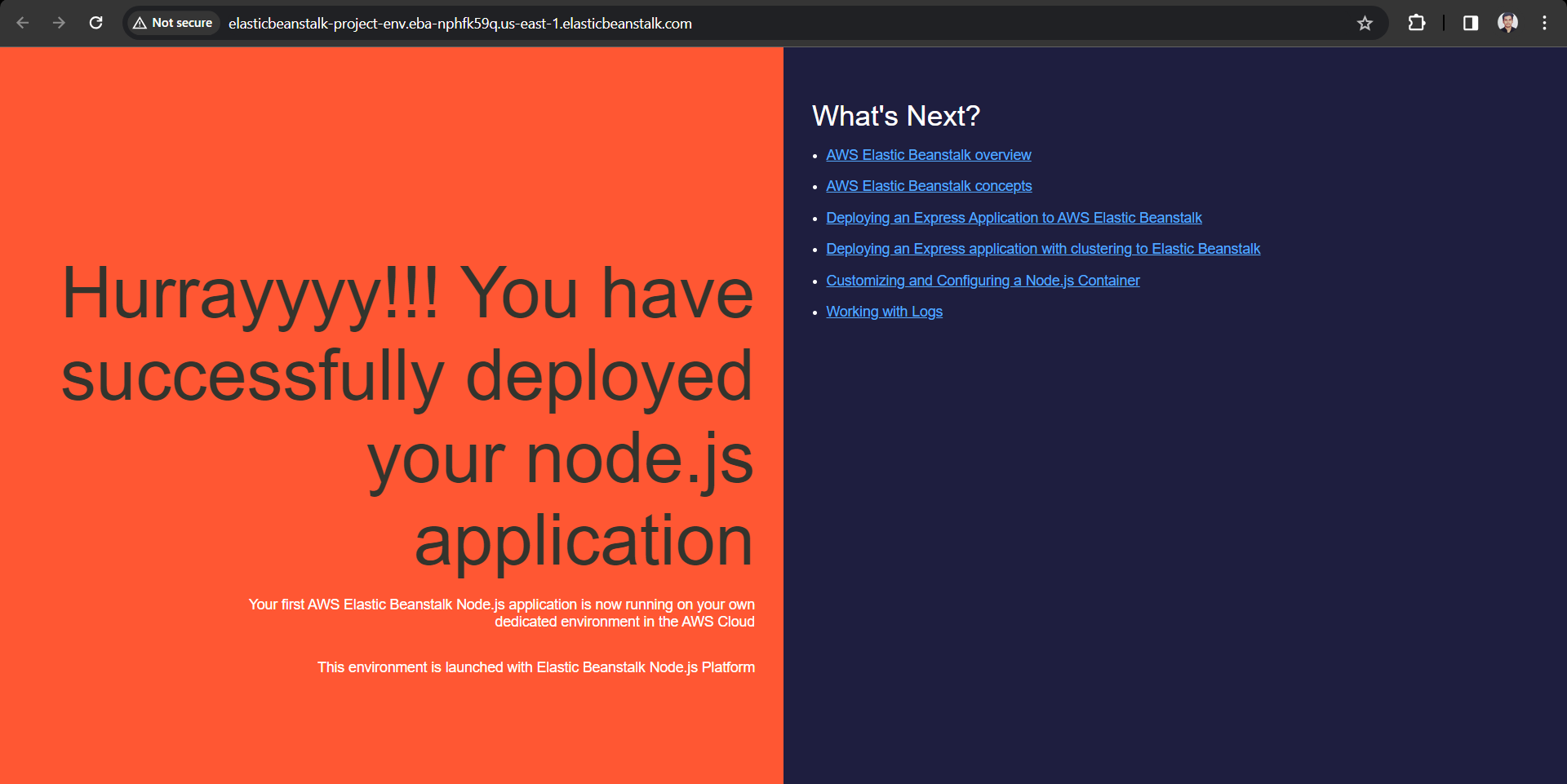
Step 4. Do the changes in the index.html file as per your choice and save it (CTRL+S).

Step 5. Whenever you make changes locally you need to push in to your repository. To push the updated code to the repository, enter the following commands:

* git add .
* git commit -m “my first changes”
* git push

Now the changes that you have done locally is pushed to your repository. Now the CodePipeline will check if there are any changes in the code, if yes then it will automatically initiate deploying it to your EC2 instance via ElasticBeanstalk.

Now when you click the domain on ElasticBeanstalk, you will see the changes visible on the website. In my case it looks like below:



This is how you can deploy your node.js application using AWS ElasticBeanstalk and do Continuous Integration and Continuous Deployment using AWS CodePipeline.