CS 423
Introduction To DevOps
Assignment # 2

Name: Syed Muhammad Ashhar Shah
Reg No: 2020478
Faculty: B(CS)

PROJECT INTRODUCTION

In this project we are going to automate the deployment process in which we have developer's who are working on developing the application and we have a QA-Engineer who is making sure that there are no bugs or issues in the underlying application.

For the deployment process we will be creating two separate servers, one will be for testing the application (for the QA and the Developers) and the other will be for staging the application (for the Client to view changes).

To automate this the Developers will first modify the code-base to add or remove features as per the requirements of the Client. Once done they will trigger a workflow which will deploy the changes to the testing server which can be viewed by the QA and the developer to see if there are any issues with the code-base and all the features work properly.

If there are no issues, the QA-Engineer will then initiate another workflow which will deploy all the changes to the staging server after which he can prompt the client to view said changes and provide his feedback.

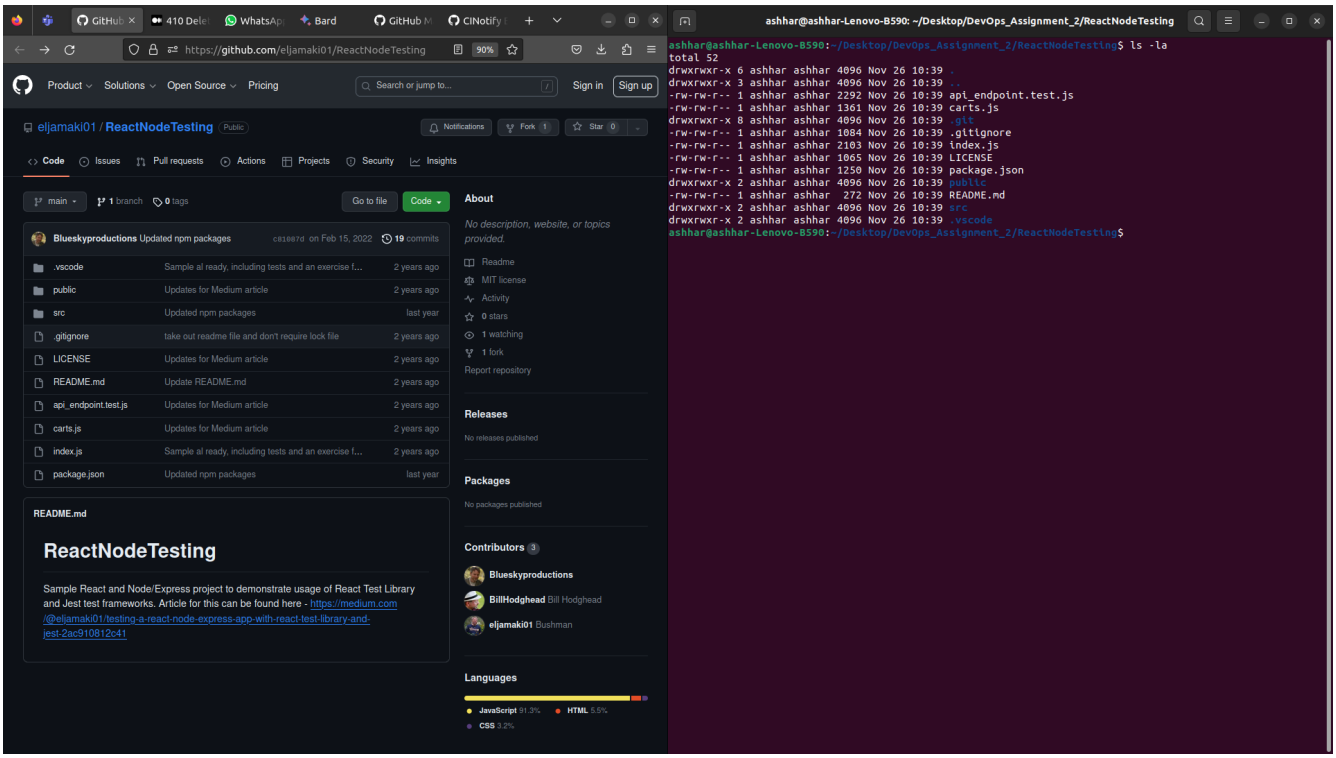
This process will continue when ever there is a new feature being developed on the code-base being modified.

TASK COMPLETION

Task 1: Selection and deployment on localhost of any simple open-source application.

1. Clone the following simple open-source application from GitHub built using ReactJs and NodeJs.
   - ReactNodeTesting app
   - You may also clone any simple open-source application built using any of the following technologies but must discuss with me before proceeding:
     - Front-end: React or Angular
     - Back-end: NodeJs, Python, Java, PHP

To clone the repository we will use the command:
- *git clone https://github.com/eljamaki01/ReactNodeTesting.git*



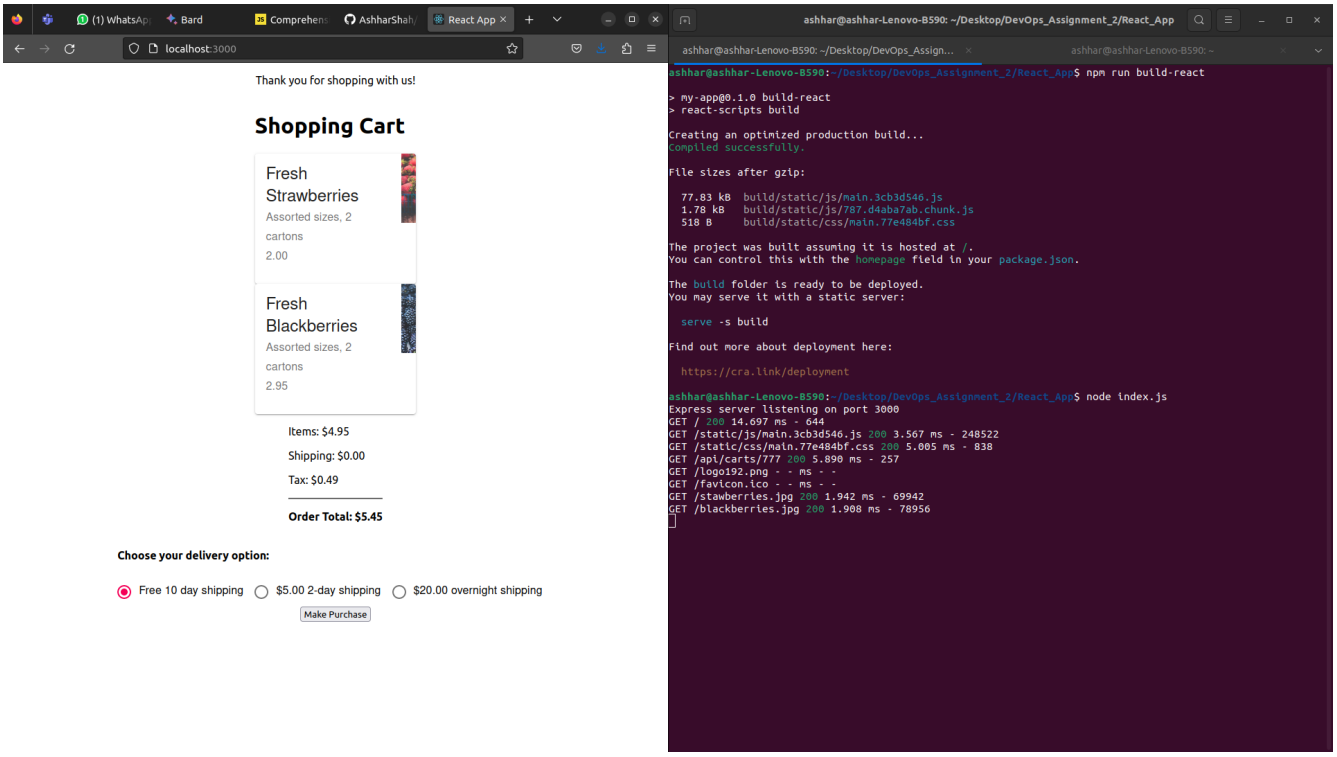
This will clone the git repository onto our local system.

2. Clone the following simple open-source application from GitHub built using ReactJs and NodeJs.

   To deploy the application on our system we will first install the node dependencies using the command:
   - *npm install*

   Once we have installed all the required dependencies we will host our application using the command:
   - *node index.js*

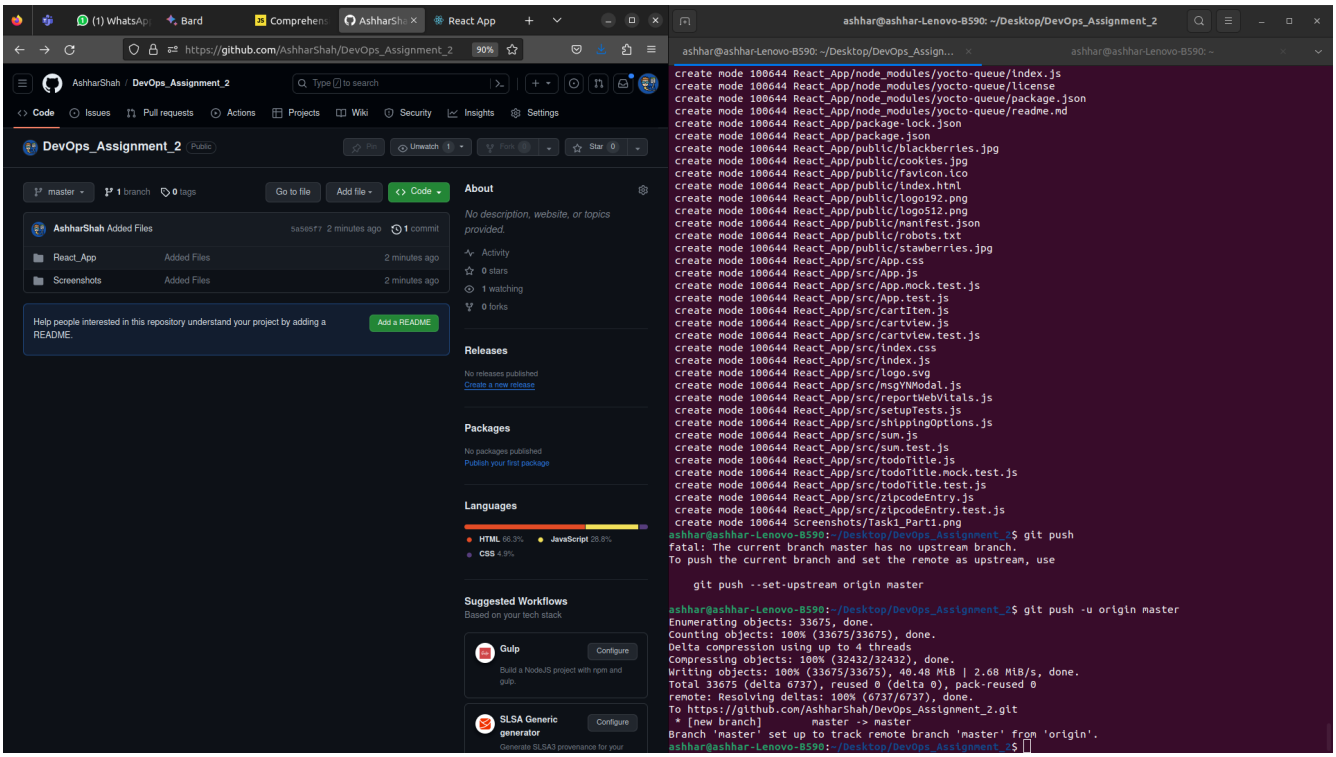
   We can also use nodemon via the command:
   - *npx nodemon index.js*



3. As you and your team will be making further changes to this application so you must initialize this cloned repository code to your new repository and push it to your GitHub repository. Add a second team member as a collaborator to your GitHub project.

   Since I am working on this as a solo project, I will not be adding another collaborator to the repository. I will only add project to my own GitHub repository by creating a repository with the command:
   - *gh repo create*

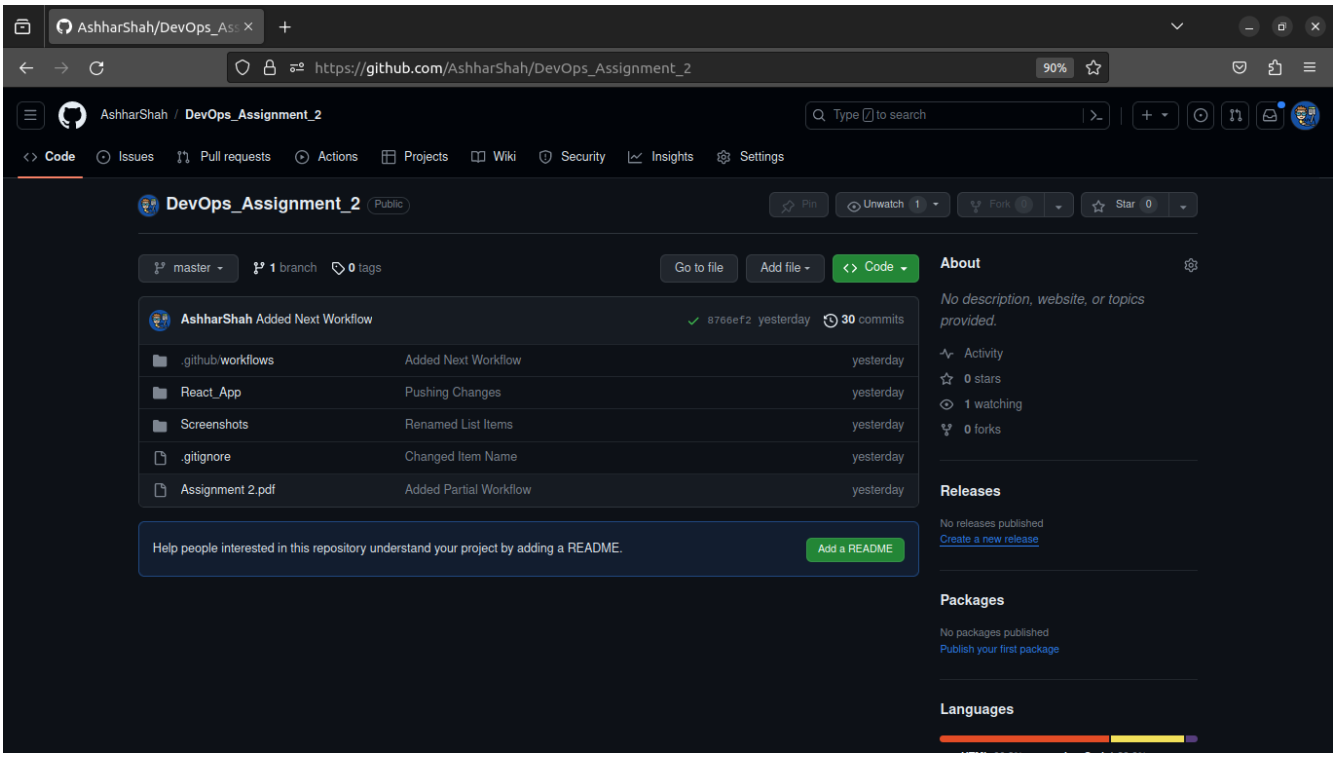
   Then I will push the local repository to the GitHub servers using the commands:
   - *git add .*
   - *git commit -m "Adding Files"*
   - *git push -u origin master*



4. You must make your repository public and add me as a collaborator to that repository as well.

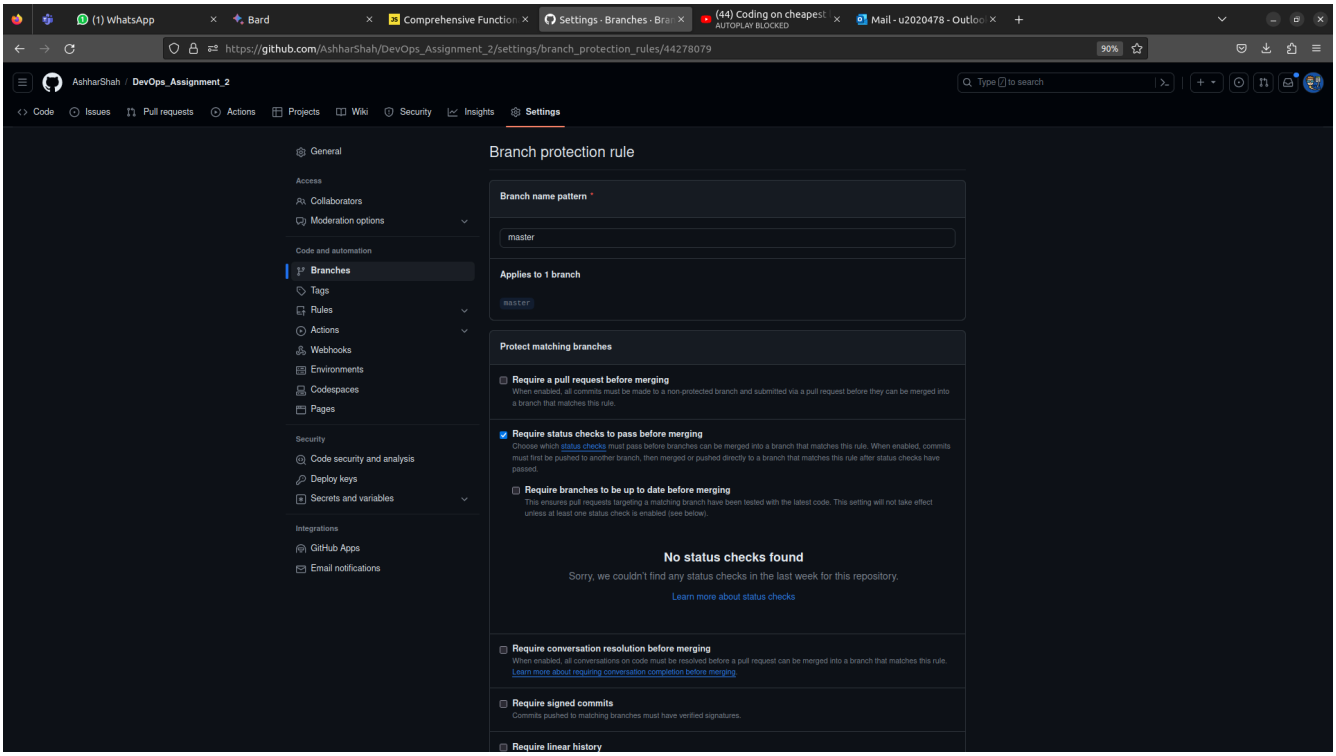
   When we initialize the repository in part 3, we already specify that the repository is going to be public.

   Link for verification:
   https://github.com/AshharShah/DevOps_Assignment_2



5. Update your repository settings such that no collaborator should be able to commit directly to the main branch.

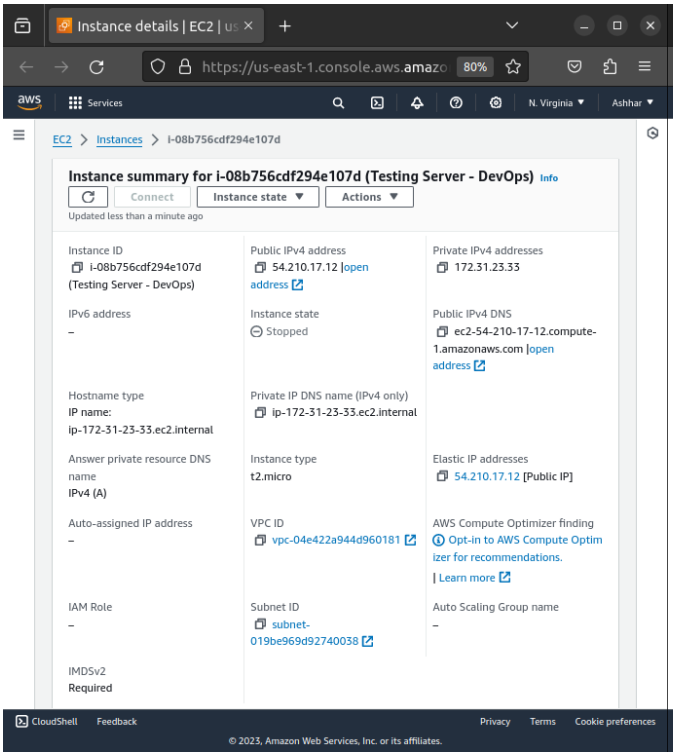
   To restrict any developer from pushing directly to the main branch. We will define a branch rule in the repository setting which disables the option to push to the master branch directly without a pull request.

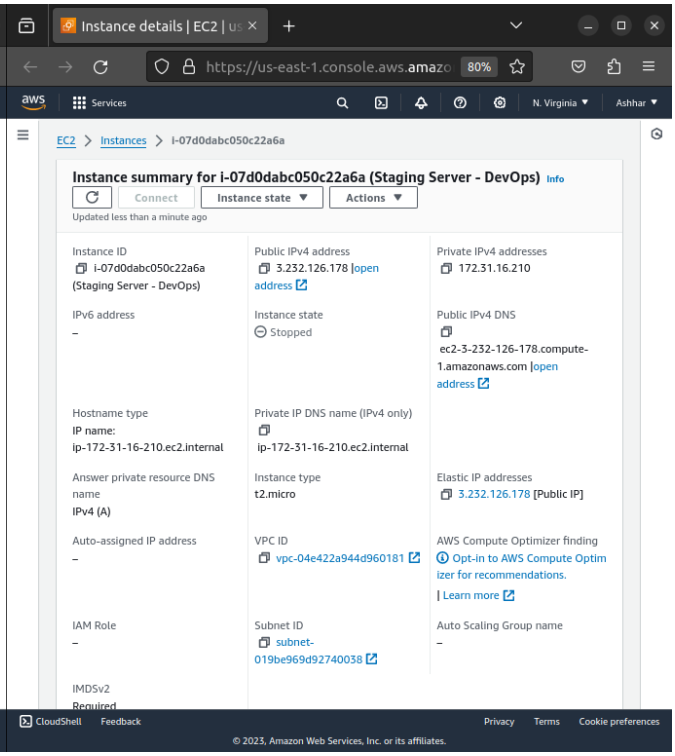


Task 2: Create two EC2 instances on AWS for Testing and Staging environment.

1. Give an appropriate name such that it represents environment.

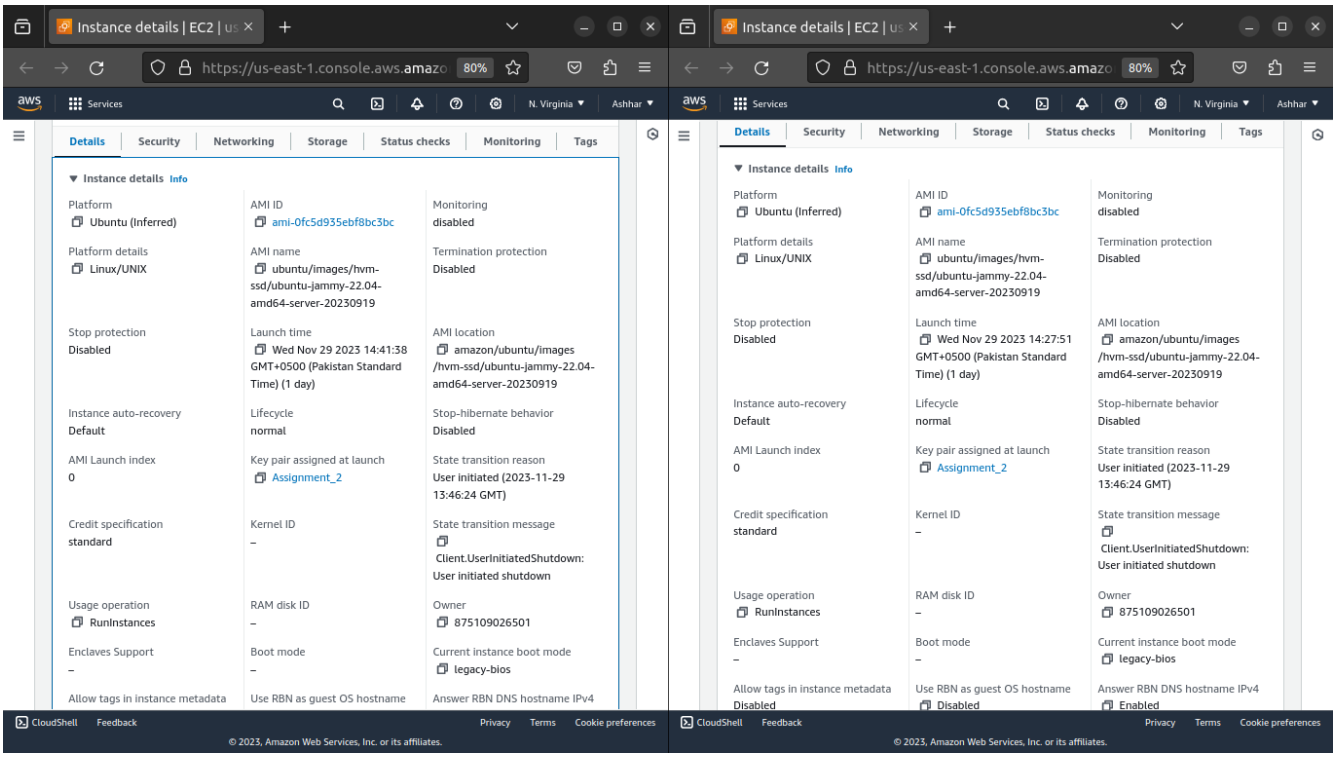
   The instances will have the names:
   - *Testing Server – DevOps*
   - *Staging Server – DevOps*





2. Each environment Amazon Machine Image (AMI) should be Ubuntu Server 22.04 LTS (HVM)

   Both the servers are of t2.micro instances both running the Ubuntu operating system of version 22.04 LTS which can be seen in the screenshot below.

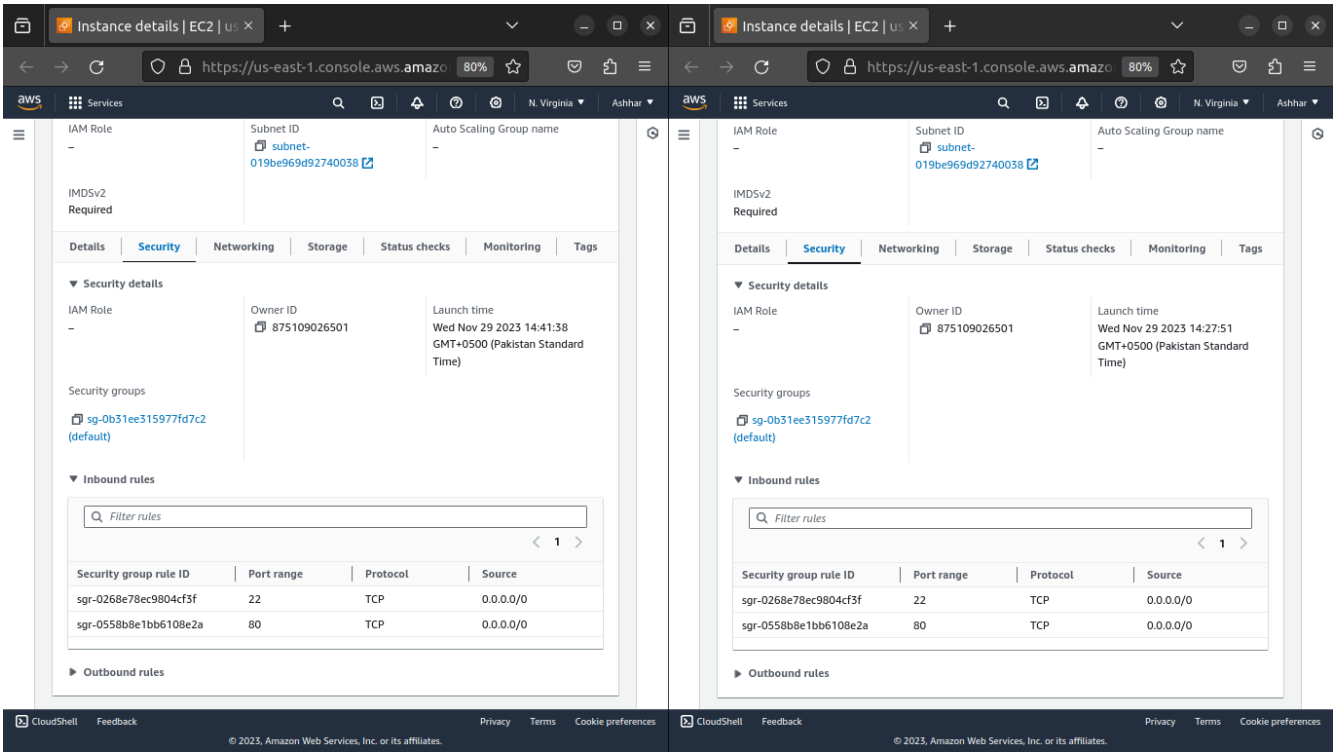


3. All instances must share one common security group such that if you add/remove any rule, it should affect all instances.

   Both the instances have the same security group:
   - sg-0b31ee315977fd7c2

   The security group has the following rules:

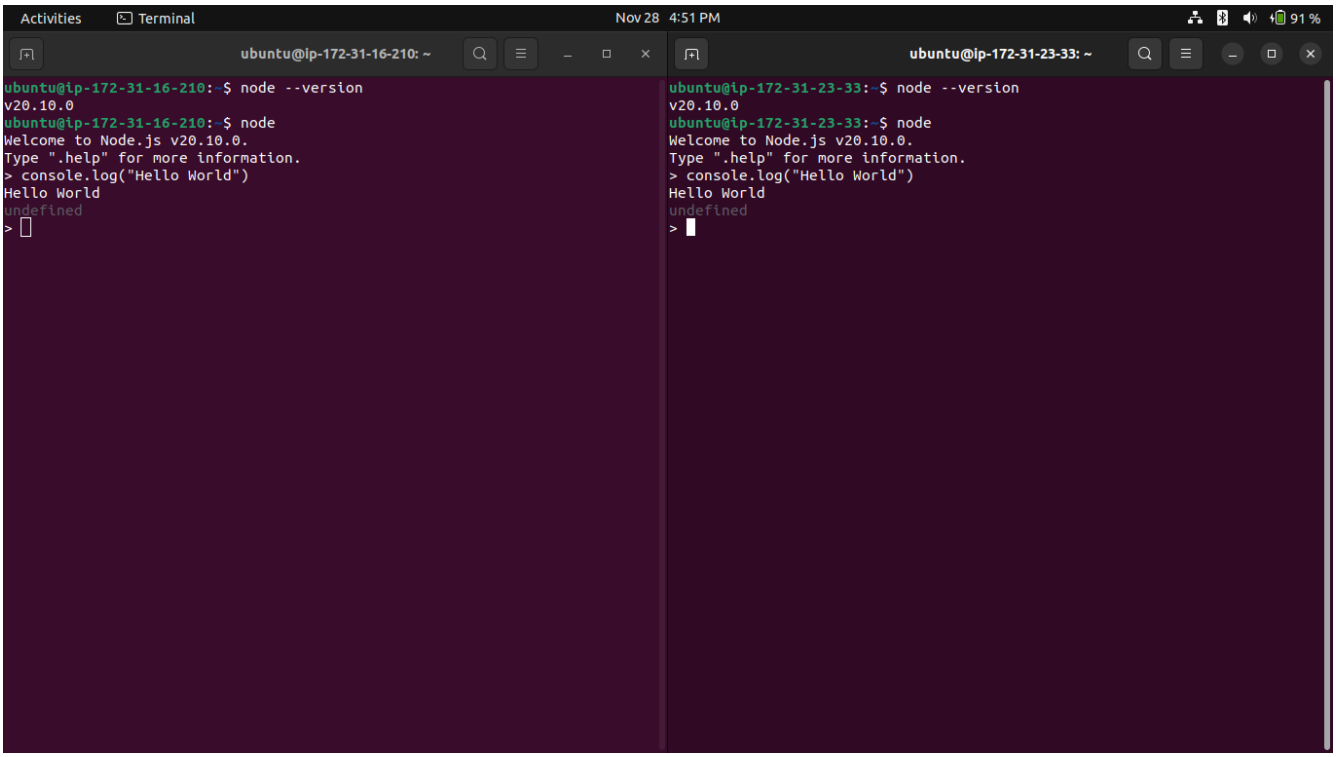
	Rule ID	Port	Protocol	Source	
a)	sgr-0268e78ec9804cf3f	22	TCP	0.0.0.0/0	(SSH)
b)	sgr-0558b8e1bb6108e2a	80	TCP	0.0.0.0/0	(HTTP)



4. Configure your environments such that it includes required dependencies for your React and Nodejs project.

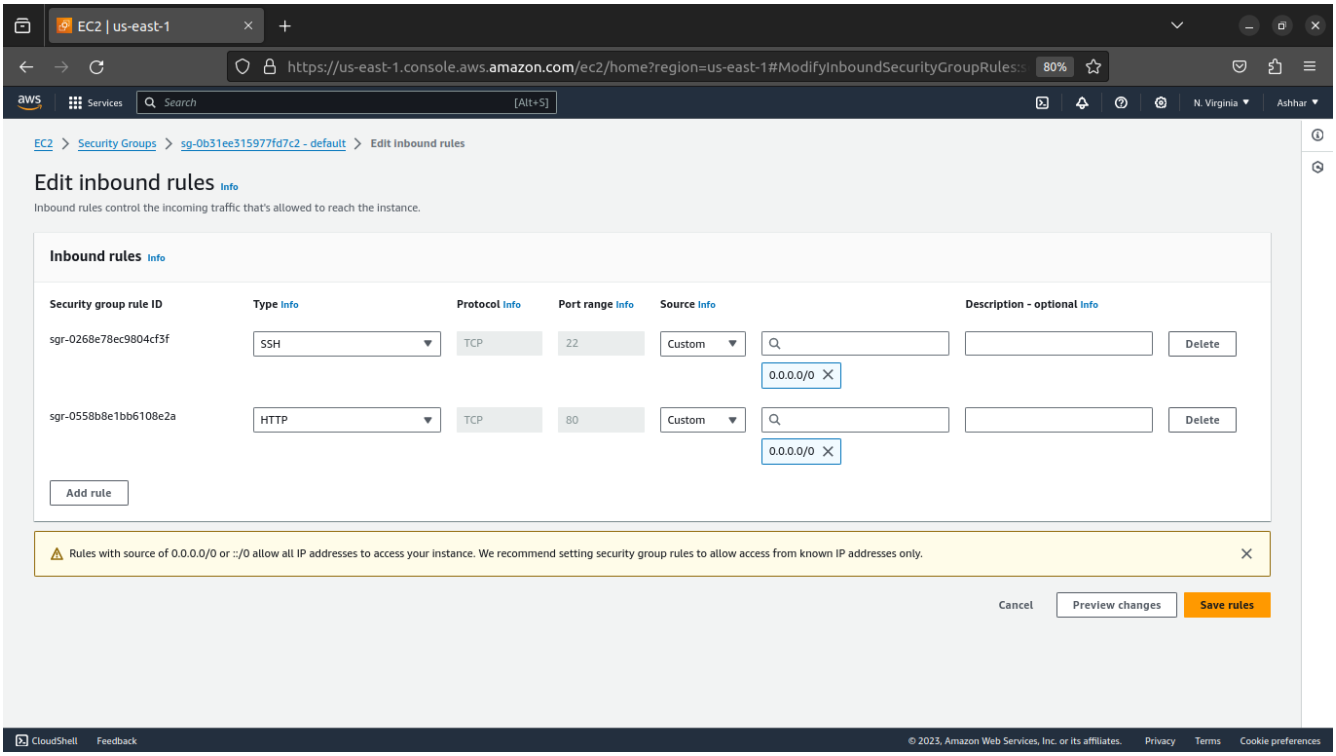
   To configure our environment we will need to install NodeJs along with Node Package Manager (npm) on our AWS instances.

   To do this we will simply SSH into the instances and run the following set of commands:
   - *sudo apt-get update*
   - *sudo apt-get install -y ca-certificates curl gnupg*
   - *sudo mkdir -p /etc/apt/keyrings*
   - *curl -fsSL https://deb.nodesource.com/gpgkey/nodesource-repo.gpg.key | sudo gpg --dearmor -o /etc/apt/keyrings/nodesource.gpg*
   - *NODE_MAJOR=20*
   - *echo "deb [signed-by=/etc/apt/keyrings/nodesource.gpg] https://deb.nodesource.com/node_$NODE_MAJOR.x nodistro main" | sudo tee /etc/apt/sources.list.d/nodesource.list*
   - *sudo apt-get update*
   - *sudo apt-get install nodejs -y*



5. Add or remove rules in your security group where required.

   Since we only want to SSH into the web server and host an application. We will only enable the SHH and HTTP ports in the security group settings.



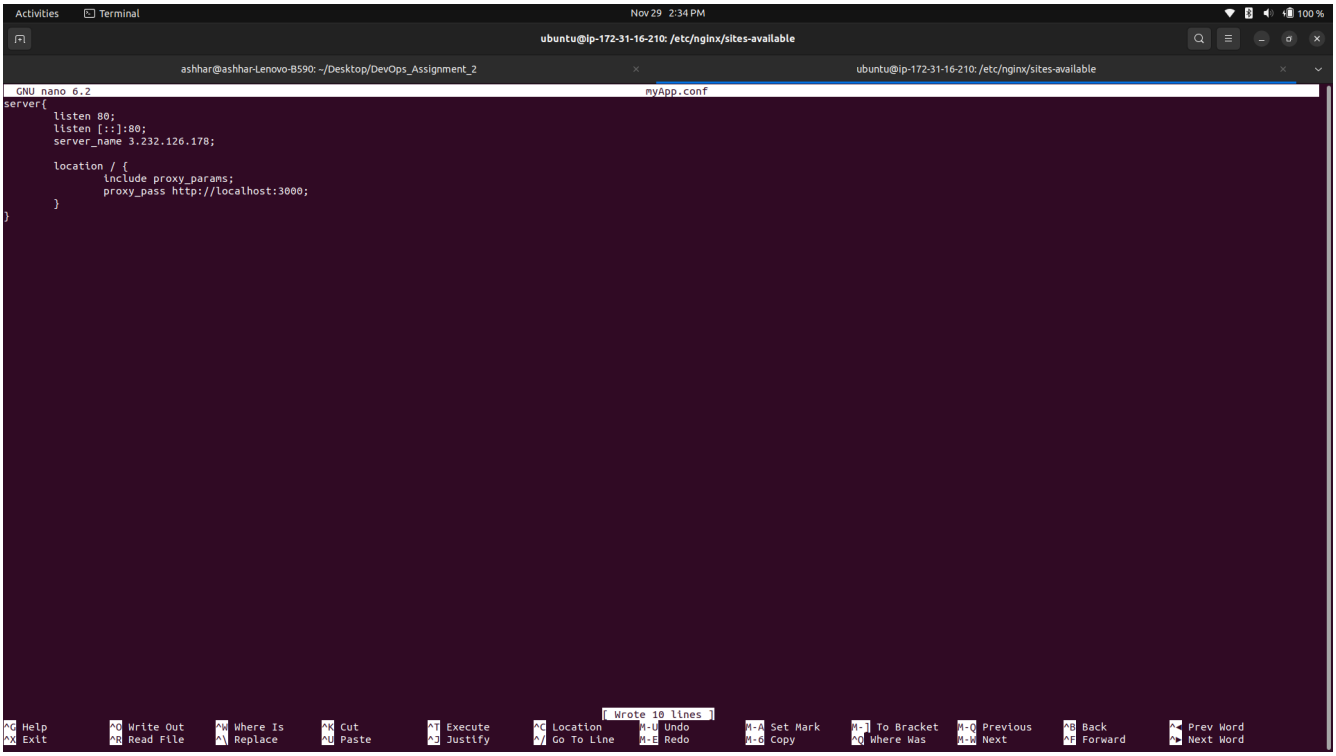
6. As you will be working in a group you may create first AWS instance in one account and second in other account.

   Since this is a solo project, I will be creating both EC2 instances on the same account which is my personal AWS account.

   Also to host the Node/React application I will be using a Nginx HTTP server.

   To install nginx on the instances I will use the command:
   - *sudo apt install nginx*

   Then I will configure the nginx server to point toward by NodeJs application that runs on the localhost:3000. To do this we will first create a config file in the sites-available folder and link it to the folder sites-enabled. To do this we will use the following commands:
   - *sudo touch /etc/nginx/sites-available/myApp.conf*
   - *sudo ln /etc/nginx/sites-available/myApp.conf etc/nginx/sites-enabled*
   - *sudo nginx -t*
   - *sudo systemctl restart nginx.service*

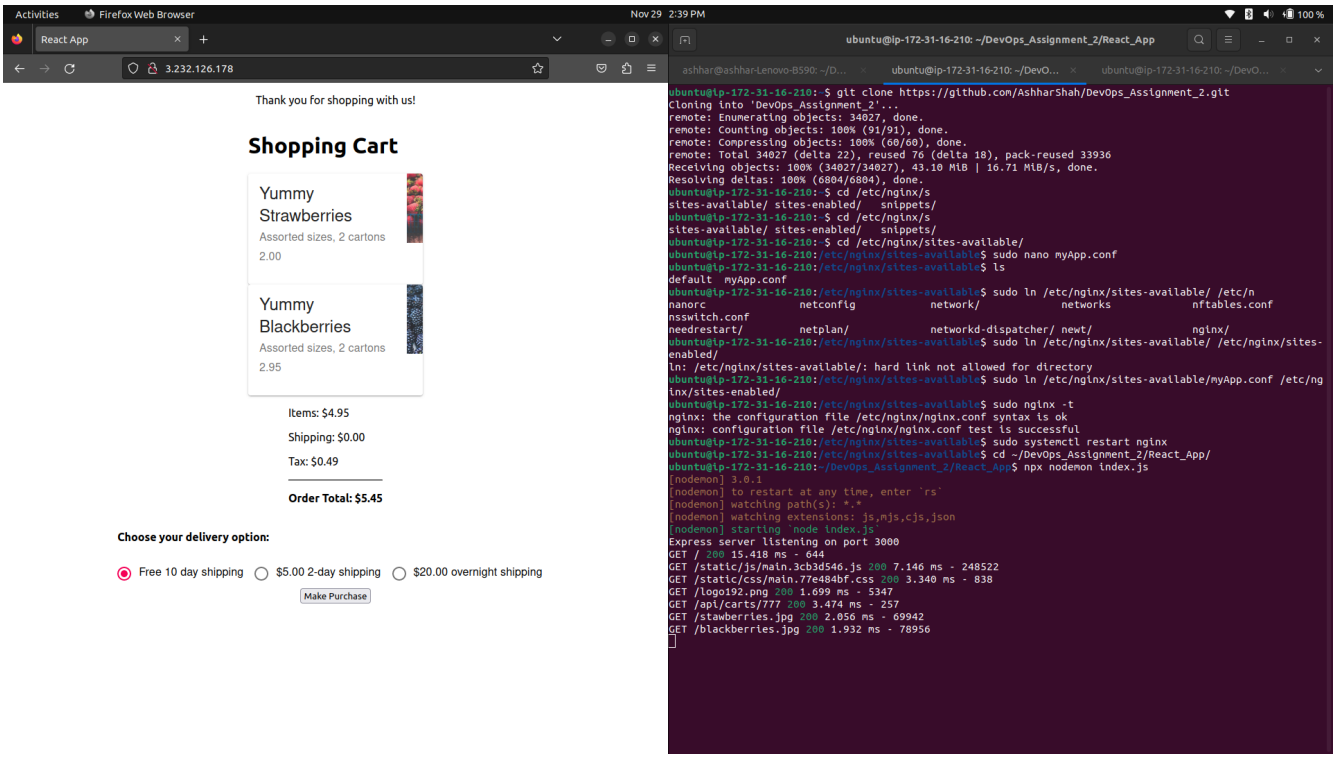


Then to start the application we will simply navigate into the react application's folder then use the command:

- *node index.js*

OR

- *npx nodemon index.js*

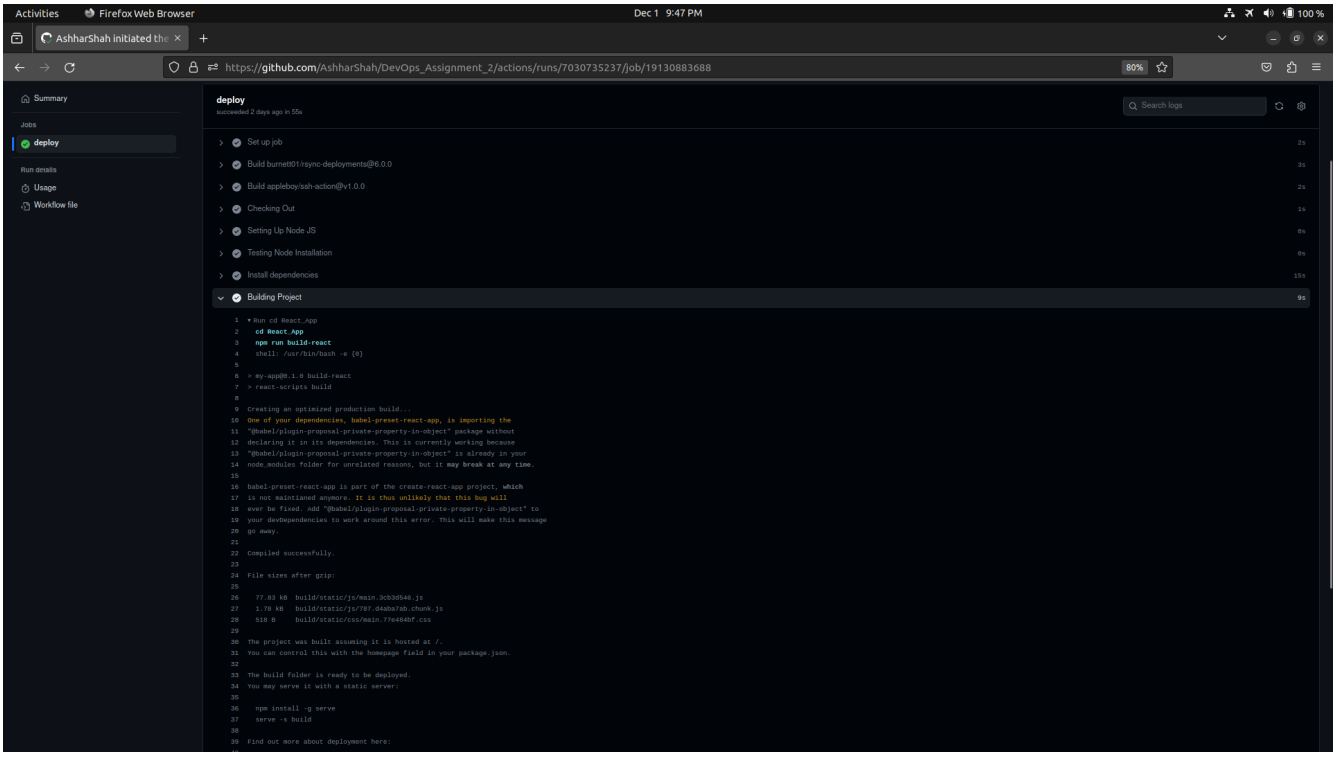


Task 3: Automating application from development to deployment.

1. You or your team members will always create features or fix branches from the main branch to add something new or update anything existing in future. Create a pull request, once done with the changes on the feature or fix branch. This pull request must trigger a workflow in your GitHub actions which will deploy changes for QA to test on AWS testing server. There should also be a button on your workflow to trigger this workflow without an pull requests if needed. QA should be able to access the deployed project using the link: http://<Instance_IP>. Your workflow must do the following:

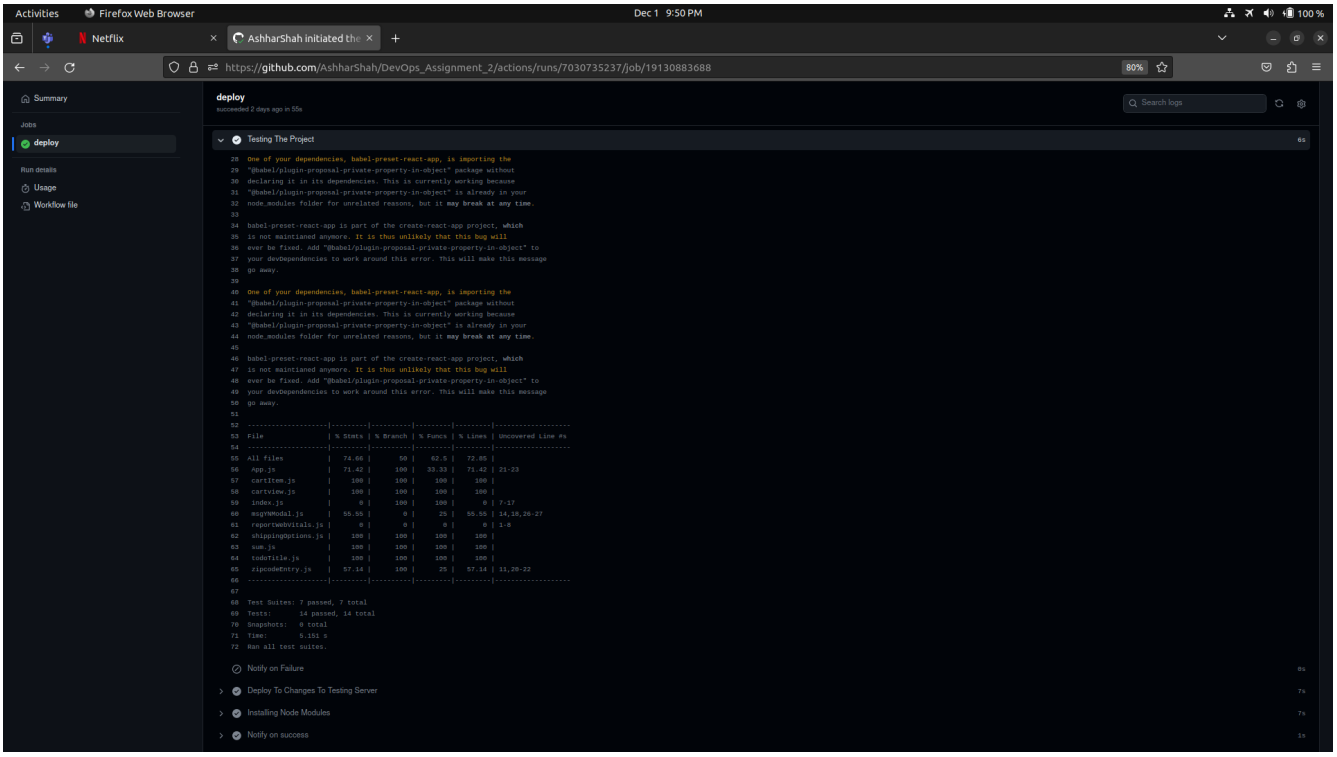
   (1) Build your project: Compile the source code and build the application or software. This step ensures that the code can be successfully compiled.

   To build the project we will simply navigate into the directory and run the command:
   - *npm run build-react*



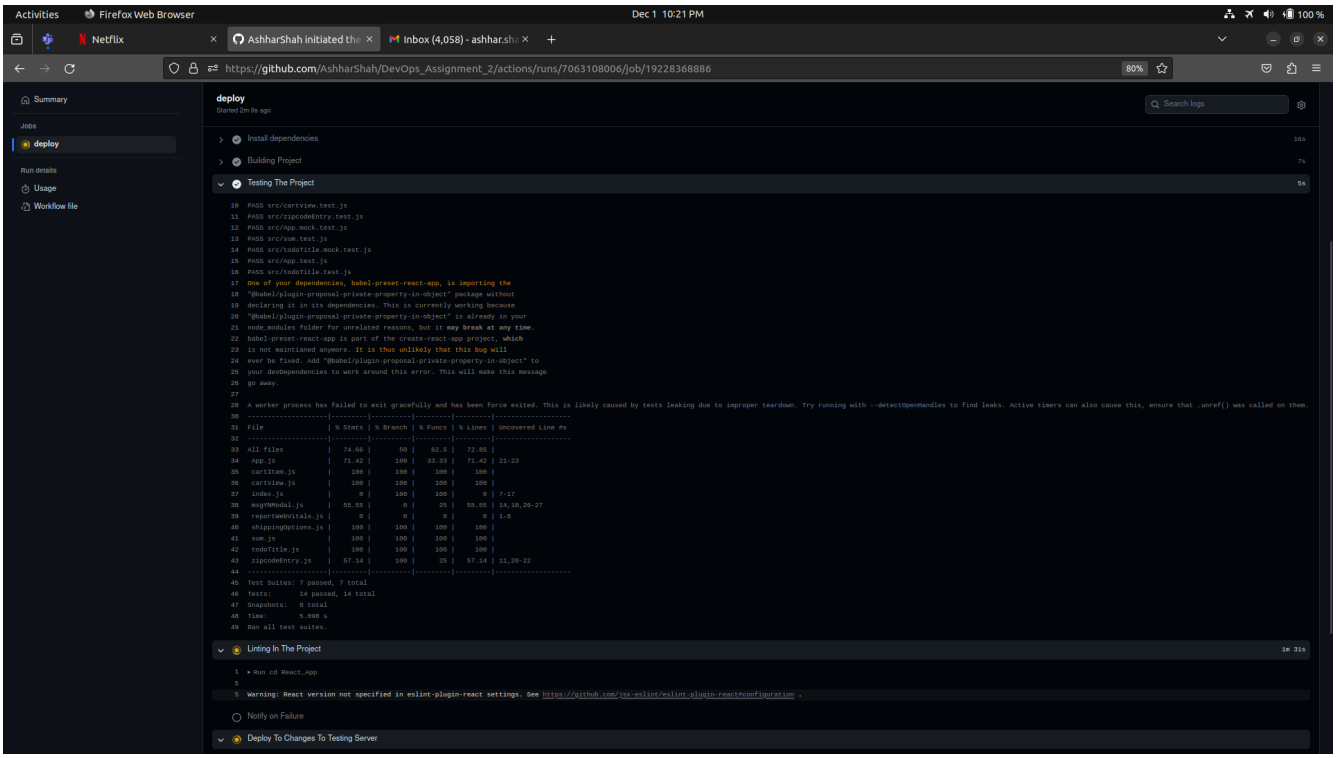
(2) Unit Testing: Run unit tests to verify that individual components or functions of the code work as expected. Unit tests are typically fast and focused on specific functionalities. (you must choose a project which has a few unit test cases, or you must create your own).

To run the test cases on our react we will use the following command in the workflow:
- *npm run test-react*



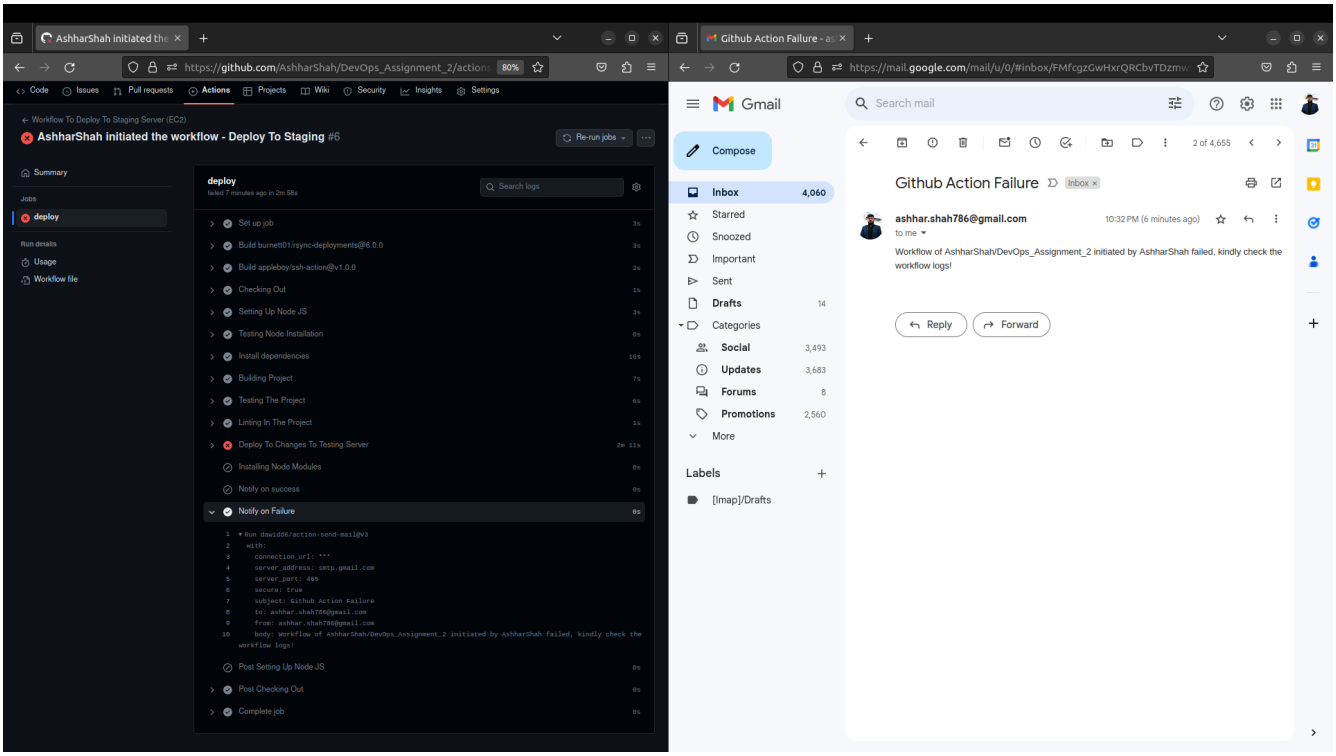
(3) Code Analysis/Linting: Perform static code analysis or linting to check for code quality, adherence to coding standards, and potential issues. This step helps maintain a consistent and high-quality code-base.

In the application, there is no linting setup so we will configure eslint in our react application. To do so I have used the following article.

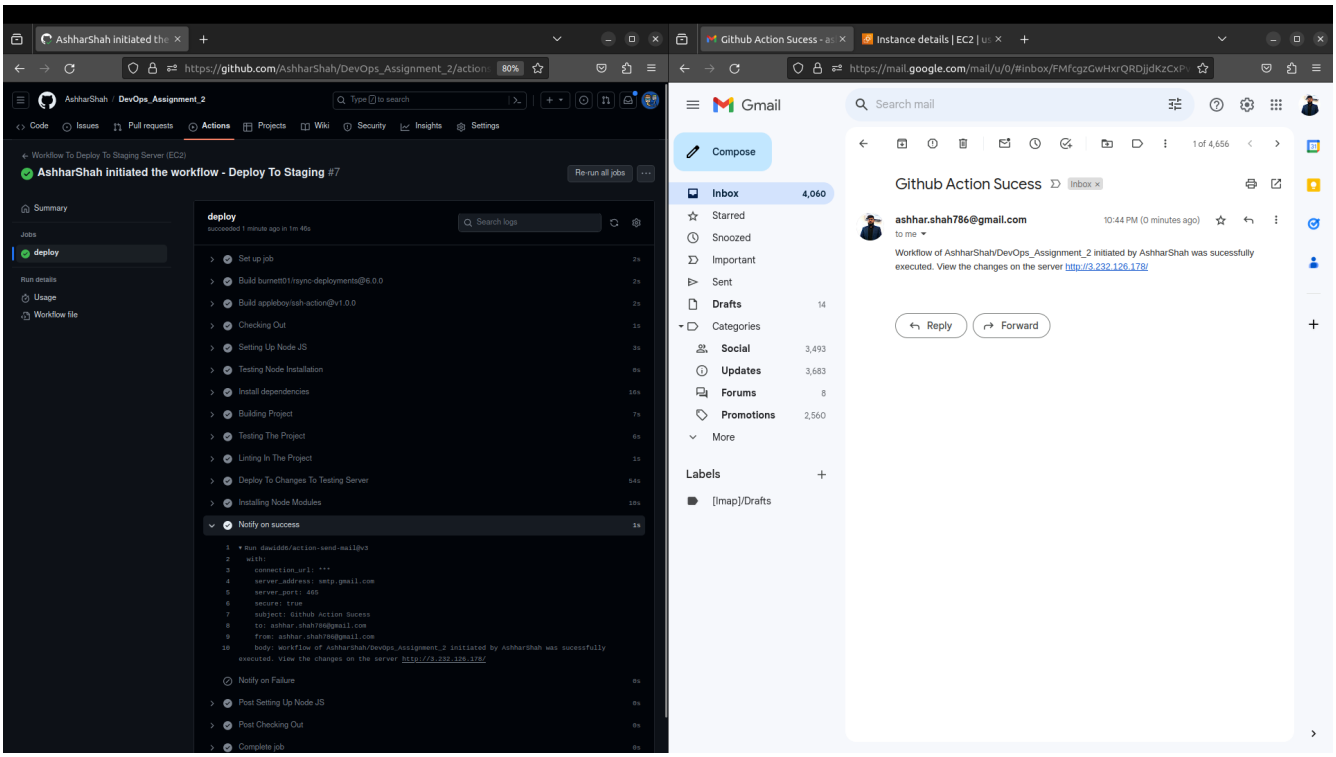


(4) Notification: Your workflow must send an email to me and the developer who created a pull request if the workflow fails. Moreover, it must send an email to the QA (to me) if it successfully deploys your change on the instance with all details how the QA can access the deployed app feature for testing.

Failure:



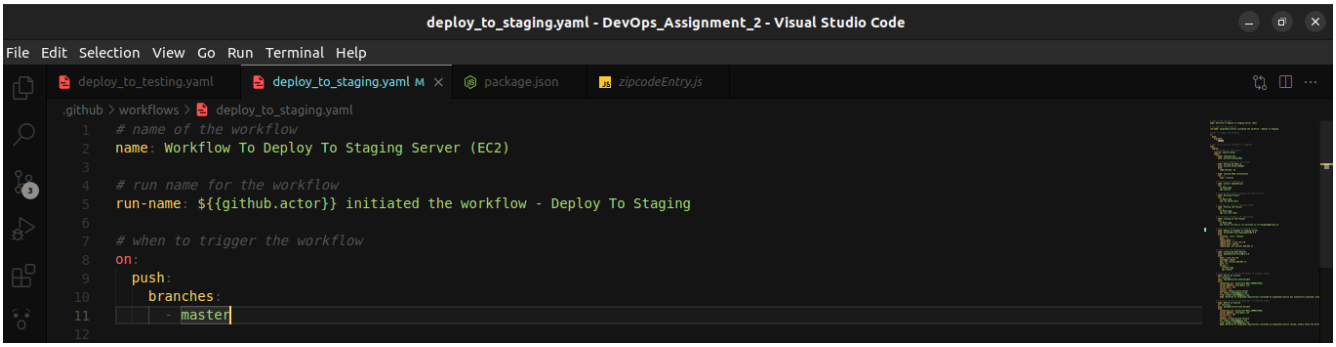
Success:


AshharShah initiated the
https://github.com/AshharShah/DevOps_Assignment_2/action
AshharShah / DevOps_Assignment_2
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
Workflow To Deploy To Staging Server (EC2)
AshharShah initiated the workflow - Deploy To Staging #7
Re-run all jobs
Summary
Jobs
deploy
Run details
Usage
Workflow file
deploy
succeeded 1 minute ago in 1m 46s
Search logs
Set up job
Build burnett01/rsync-deployments@6.0.0
Build appleboy/ssh-action@v1.0.0
Checking Out
Setting Up Node JS
Testing Node Installation
Install dependencies
Building Project
Testing The Project
Linting In The Project
Deploy To Changes To Testing Server
Installing Node Modules
Notify on success
Run dawidd6/action-send-mail@v3
with:
connection_url: ***
server_address: smtp.gmail.com
server_port: 465
secure: true
subject: Github Action Sucess
to: ashhar.shah786@gmail.com
from: ashhar.shah786@gmail.com
body: Workflow of AshharShah/DevOps_Assignment_2 initiated by AshharShah was sucessfully executed. View the changes on the server http://3.232.126.178/
Notify on Failure
Post Setting Up Node JS
Post Checking Out
Complete job
Github Action Sucess - a
Instance details | EC2 | u
https://mail.google.com/mail/u/0/#inbox/FMfcgzGwHxrQRDjjdKzCxP
Gmail
Search mail
Compose
Inbox 4,060
Starred
Snoozed
Important
Sent
Drafts 14
Categories
Social 3,493
Updates 3,683
Forums 8
Promotions 2,560
More
Labels
[Imap]/Drafts
1 of 4,656
Github Action Sucess
Inbox
ashhar.shah786@gmail.com
10:44 PM (0 minutes ago)
to me
Workflow of AshharShah/DevOps_Assignment_2 initiated by AshharShah was sucessfully executed. View the changes on the server http://3.232.126.178/
Reply
Forward

2. Assume that the QA validates all the changes made in a Testing environment and everything looks fine. Now let's say QA confirm the pull request and merges the changes into master/main branch. This push must trigger a workflow in your GitHub actions which will deploy changes for client and team members to on AWS staging server. There should also be a button on your workflow to trigger this workflow without any push event if needed. All users should be able to access the deployed project using the link: http://<Instance_IP>. Your workflow must do all the tasks you did in your previous workflow.

   For this we will create another workflow by the which implements the same changes but works only when we perform a push to the main branch which is initiated by a pull-request's validation.

   The rest of the code will be the same, we will simply change the server to and the "on" statement in the workflow.

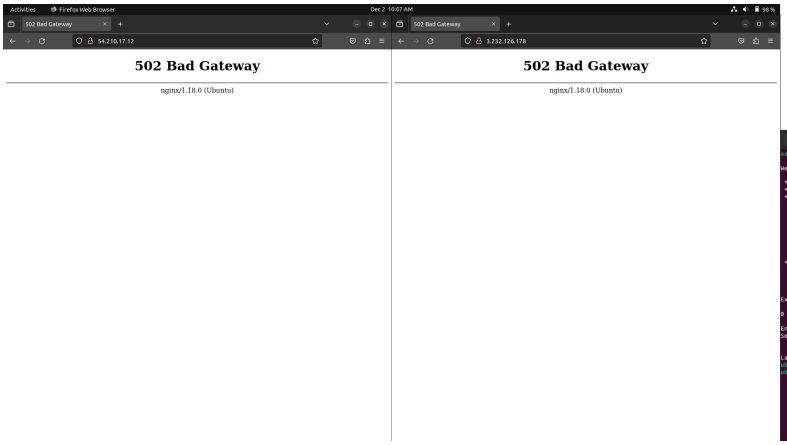


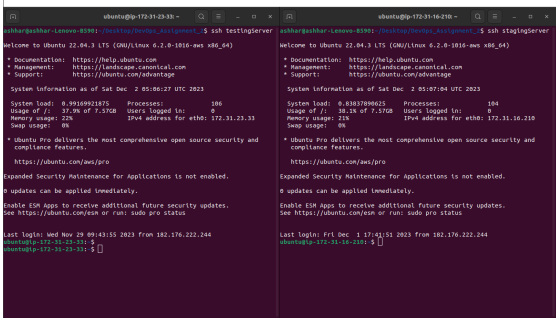
WORKFLOW DEMONSTRATION

Now that we have looked at how to do the tasks that were mentioned in the assignment, lets take a look at a step by step demonstration that shows how the workflow works to automate the deployment process.

NOTE: The browser window on the Left represents the Testing Server while the browser window on the Right represents the Staging Server (SAME FOR THE CLI WINDOWS)

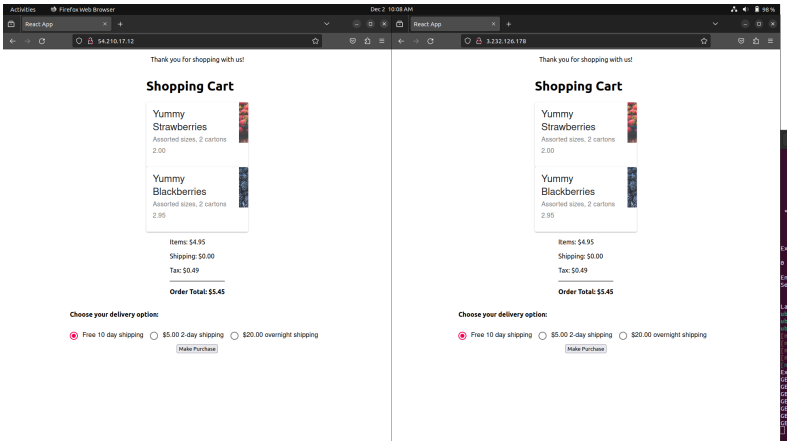
1. **Starting The Server**
   - Our first step will be to start the EC2 instance's so that we can connect to them using our public IP's.
   - Initially we would get a error 502 which represents Bad Gateway, this is because Nginx is configured to point to localhost:3000. However there is no application running on that specified port at the moment.

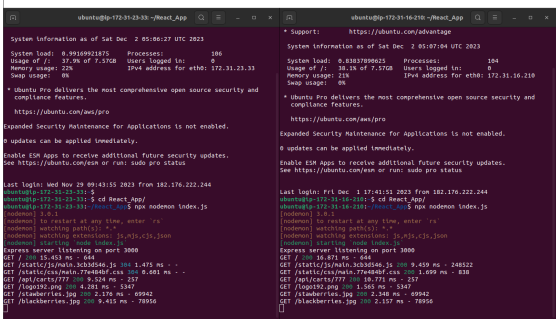




2. Starting NodeJs Application

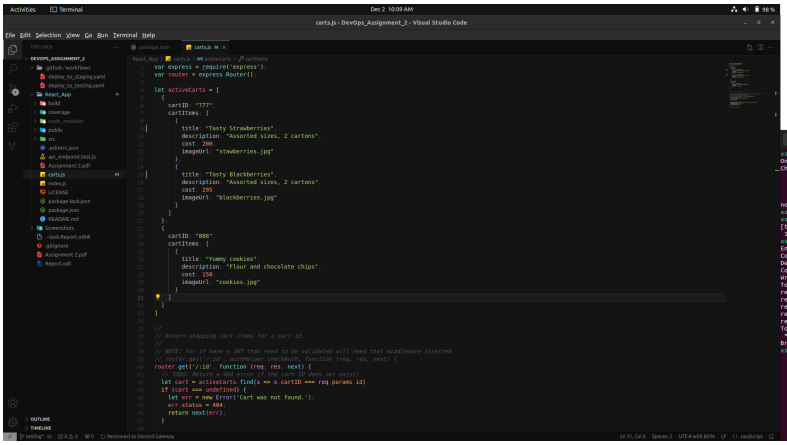
- The next step will be to start the Node/React app so we can see it when we connect to the server via HTTP.
- Once we start the application on both the server we will see that we can view the application when we refresh the web browser.

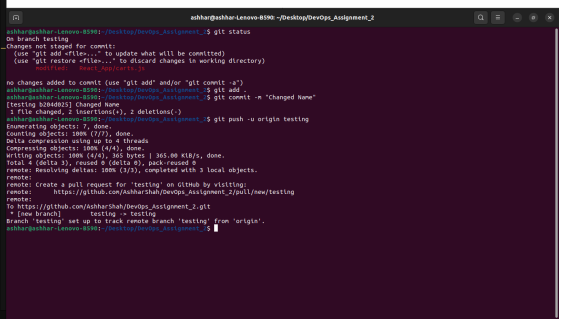




3. Modify The Code & Push To Origin

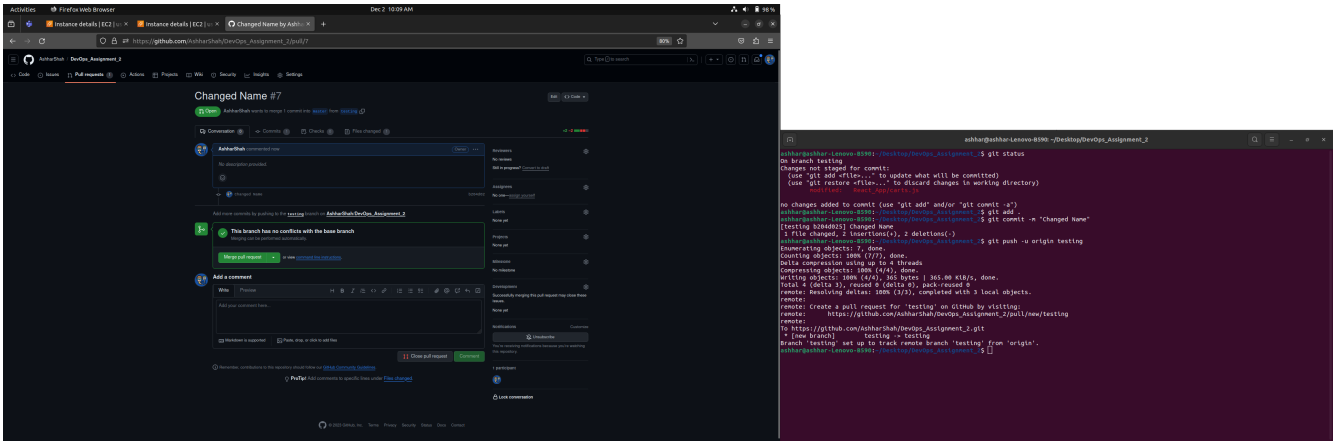
- Next we will create a new branch, in the example I have created the branch “testing” in which I will modify the underlying code by changing the name of the items from “Yummy” to “Tasty”.
- After this I will push all the changes that I have made to the “testing” branch on the “origin” server.



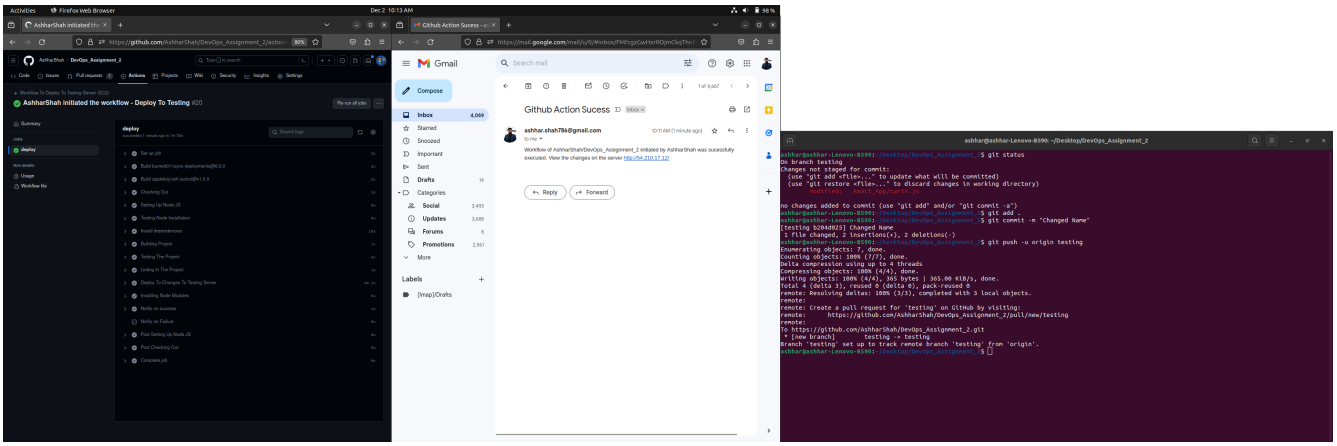


4. Create A Pull-Request

- Now that I have pushed the changes to the “origin” server. I will get an option to generate a pull-request as the “testing” branch is ahead of the “master” branch.

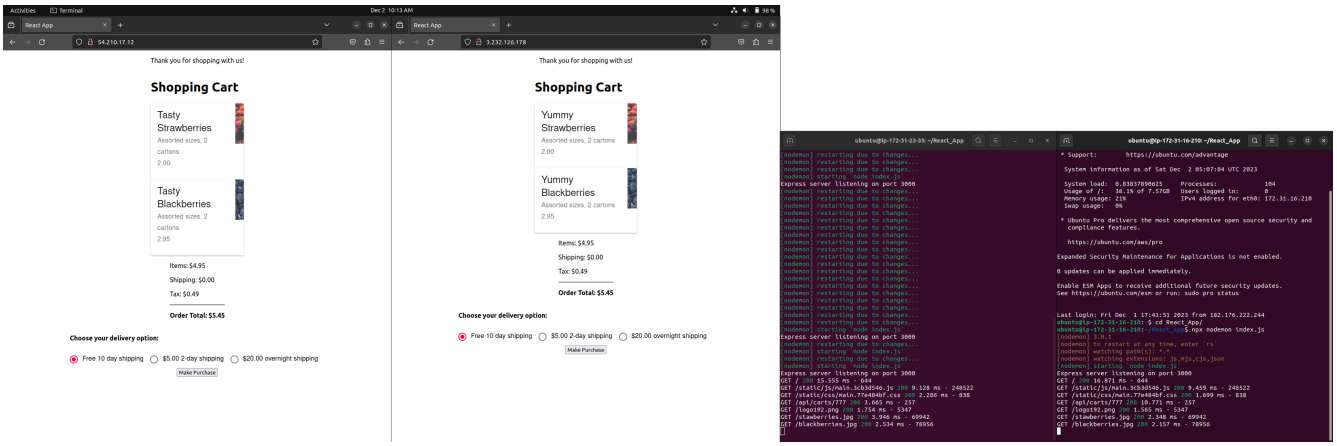


- The pull-request when generated will trigger the workflow “Deploy To Testing” which we can view in the Actions tab of our repository.
- This workflow when successfully executed, will send an email to the developer and the QA (me) which will prompt us that there were no errors. (If there was any failure it would also send an email saying that there was a error during the workflow’s execution).



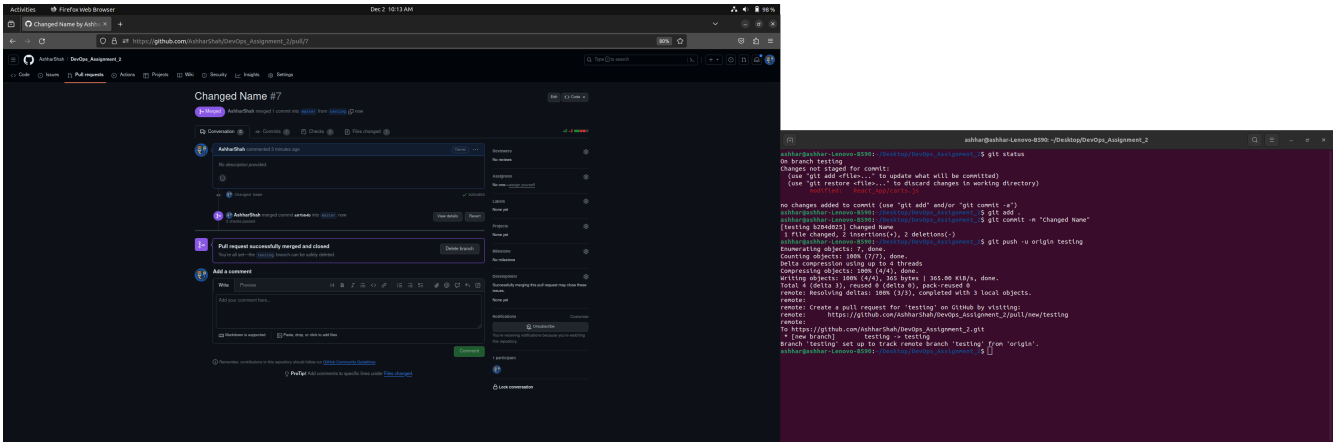
5. Verify Testing Changes

- Now the changes will be deployed to the testing server which can be viewed by the QA and the developer for any errors or bugs.
- We can see in the screenshot below that both the “testing” and the “staging” server’s are running different application versions when we view the name of the cart items.



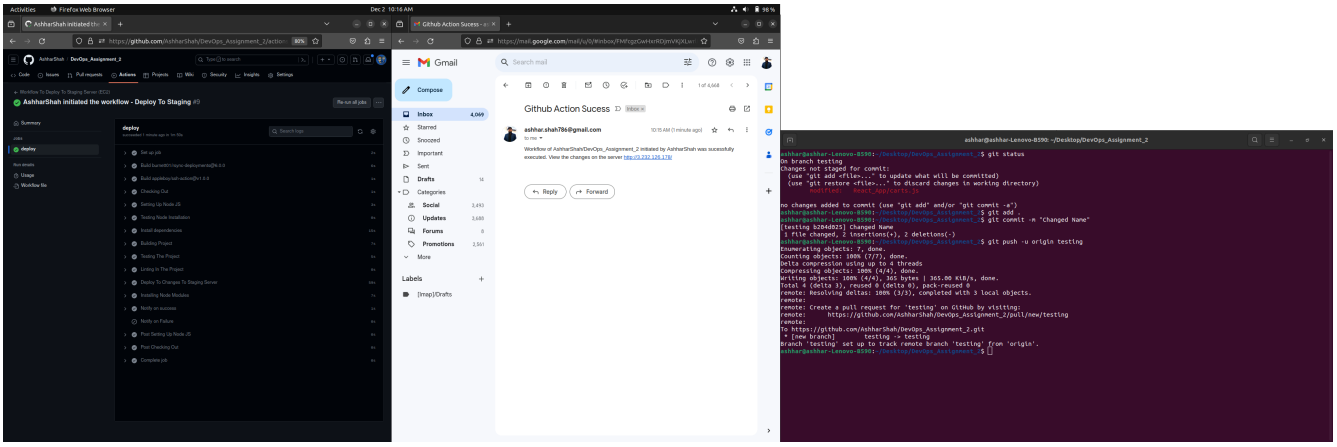
6. Merge Changes

- If the QA decides that there are no errors in the application and the application is ready to be deployed on the staging server so the client can view the changes.



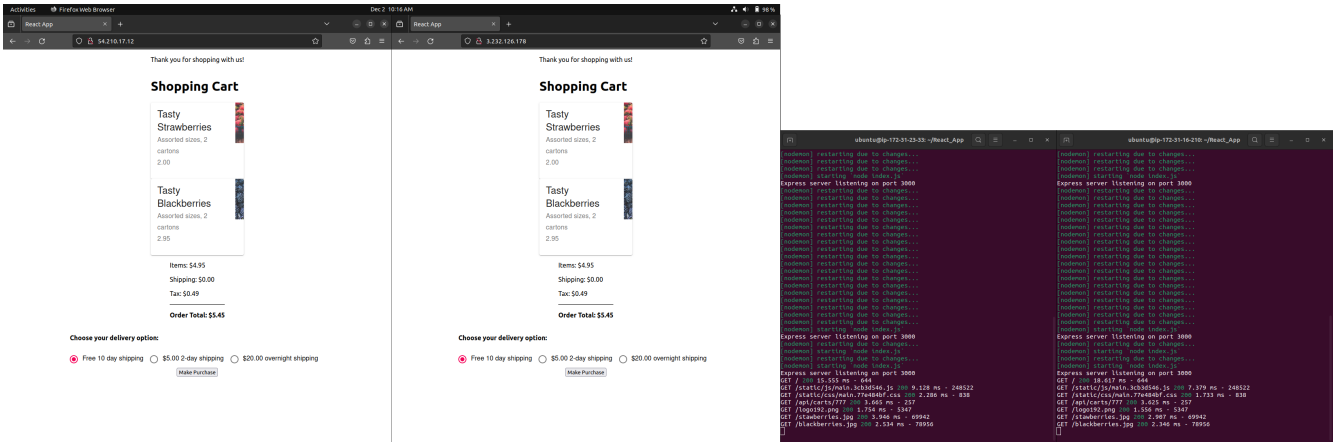
7. Deploy To Staging

- The QA will merge the “testing” branch with the “master” branch via the generated pull-request. This will then run another workflow “Deploy to Staging” which will deploy all the changes to the staging server and send an email to the QA (me).



8. Client Verification

- Now that all the application has been deployed onto the staging server, the client can view the application by simply opening his web browser and entering the Public IP of the staging server.
- At this point the application running on the “testing” and “staging” server will be identical.



CHALLENGES FACED

During the project I faced a few challenges which are listed in the points below.

1. **Setting up Nginx:** As this was my first time working with Nginx, I had difficulty in setting up the HTTP Nginx server to serve the Node/React application and make it connect to the localhost:3000.

2. **Linting In Application:** In the application there was no linting steps configured by default so I had to do research on how to use eslint to perform linting in React/Node applications to check if the syntax is up-to the required standards.

3. **Application Startup Automation:** I also had difficulty in modifying the server's setting such that when the server start's up. The React/Node application runs by default and for the changes we don't have to keep shutting down the application and restating so the changes are viewed. For this I had used nodemon which tracks the changes in the Node/React application and automatically restarts the server if there are any changes made to the application.

4. **AWS Resource Limit:** Another challenge was that I was using free-tier AWS service and in the project I had reserved 2 Elastic IP's for both of the server. I was only given 1 hour of Elastic IP service which I could use and had to utilize them efficiently. The issue with dynamic IP was that during each workflow run, I would have to change the IP's provided in the workflows.

FLOW DIAGRAM