CS 423
Introduction To DevOps
Assignment # 3

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.

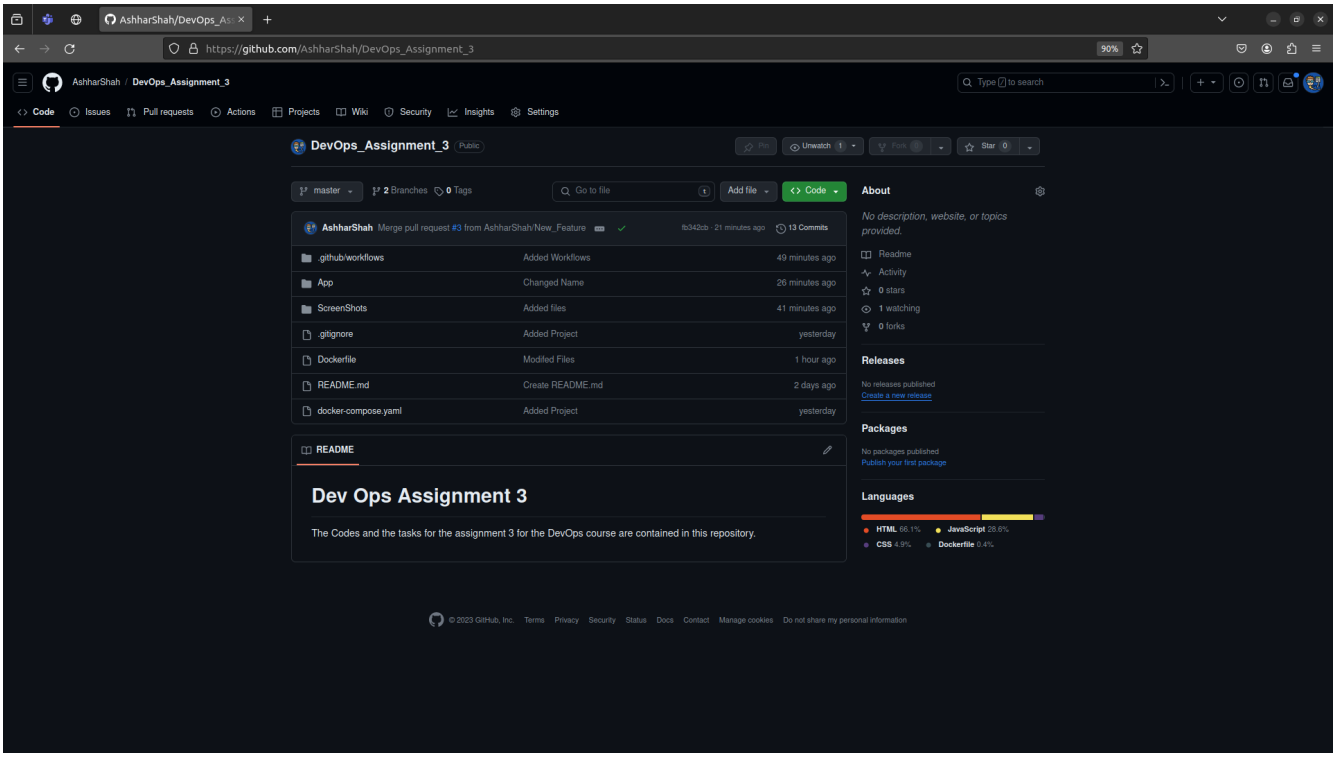
The application for both the servers will be running on containers using docker, the user will be able to access the application with the help of port mapping in which we will map port 3000 of the docker application to the port 3000 of the servers. We will also using Nginx for the server to be able to route the requests to the correct port on the server machine that is the port 3000 for our application that is running the React/Node application.

TASK COMPLETION

Task 1: Remember, in the previous assignment you cloned a simple open-source application from GitHub that was developed using ReactJs and NodeJS.

1. 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.

   We will first clone the repository and then initialize an empty repository on our GitHub account on which we will host the current files for the assignment.

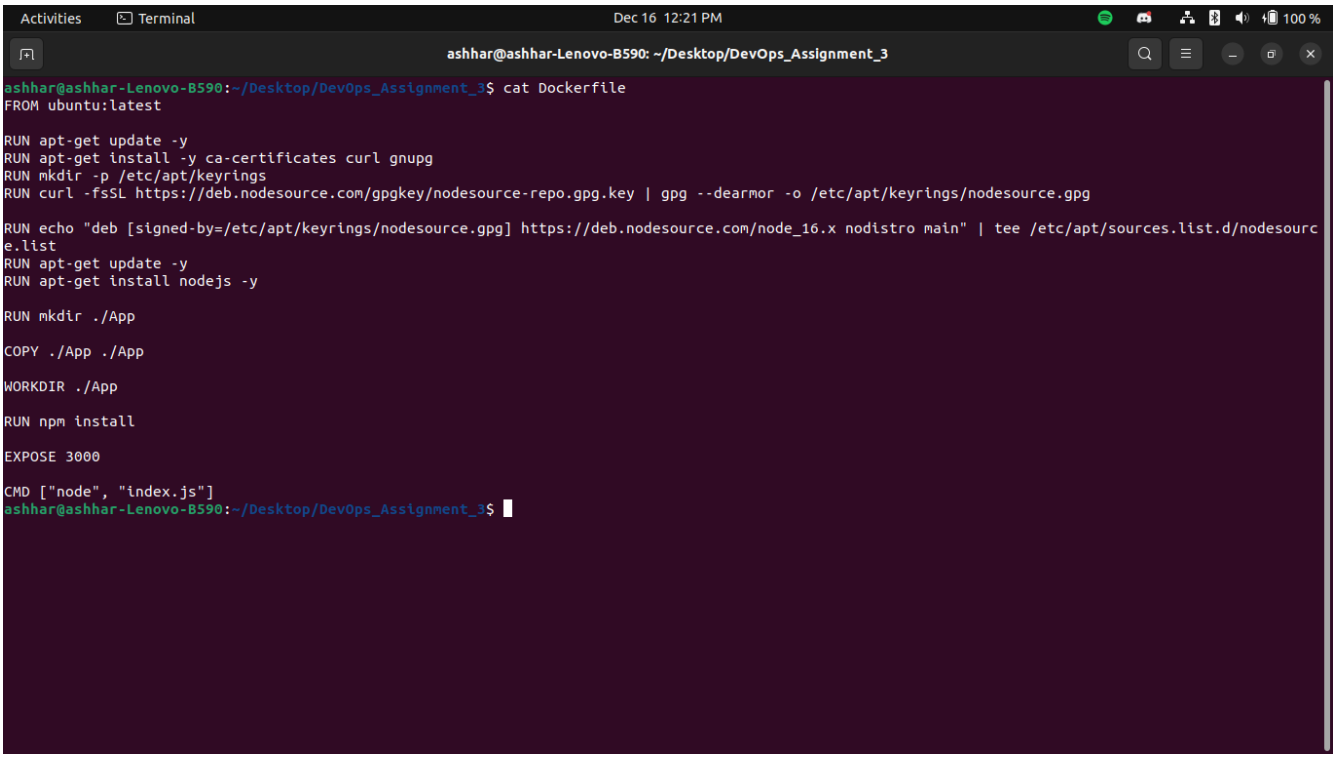


2. Deploy both frontend and backend services on local machine as separate containers named frontend & backend. Create a separate container for the database if it is being used by your application.

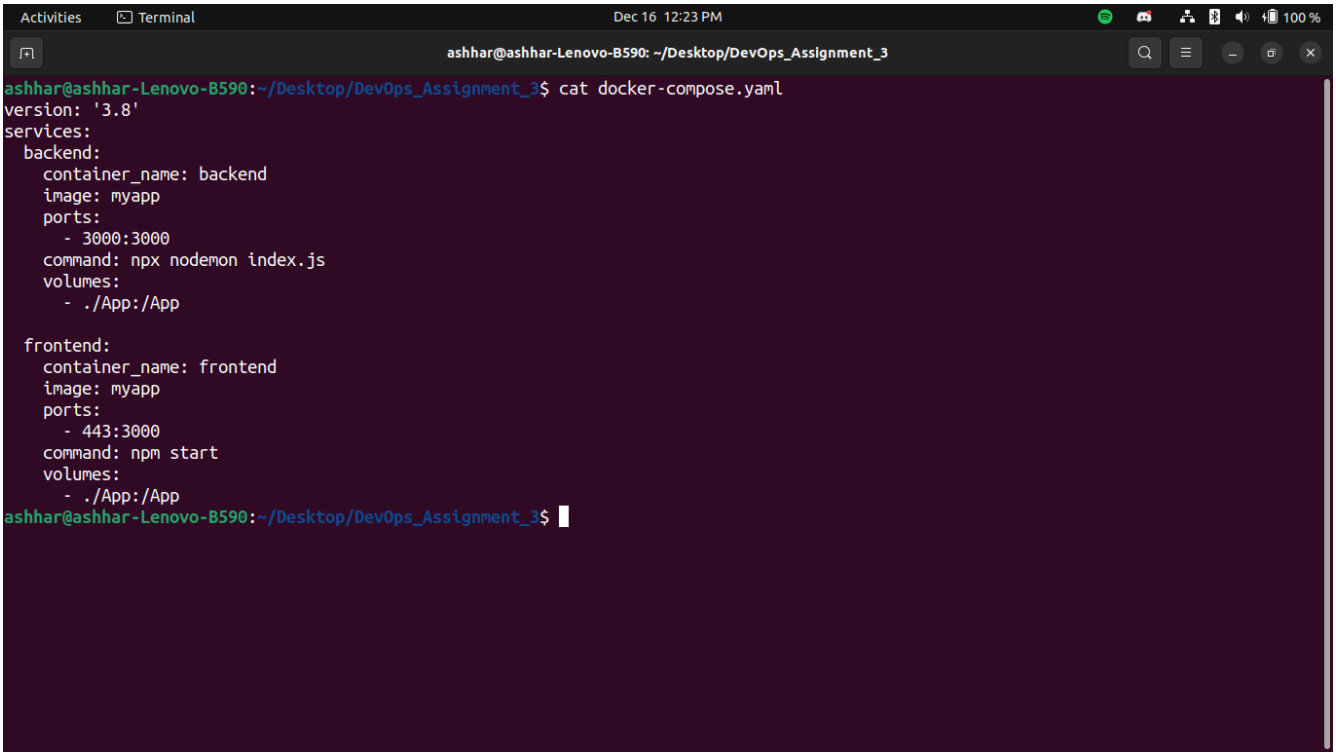
   Since our application has no database, we do not need to create one. We will only create containers for the backend and the frontend.

   First we will need to create a docker image that will be used to host the application. It will consist of the base image “ubuntu” and in the additional layers we will install “NodeJs” and copy the application folder “App” to the current containers.

   Below we can see the contents of our Dockerfile which will be used to build the image.



We will also create a compose file named “docker-compose.yaml” which will be used to host and manage the docker containers. Below we can see the contents of the compose file.



In the compose file we define two services, the frontend and the backend.

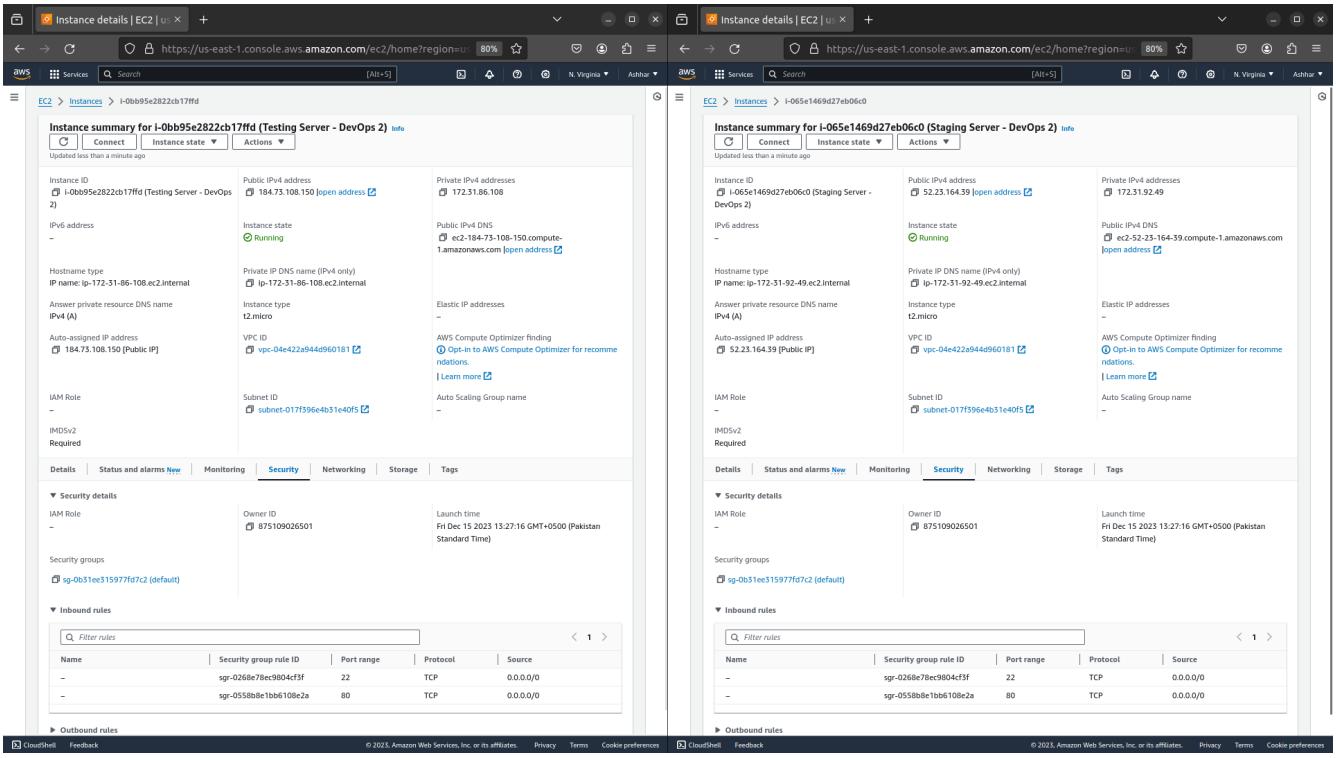
The frontend will run the React application using the command “npm start” while the backend will run the Node application using the command “npx nodemon index.js”. In both of the files we will add volumes which will track any changes on the local machine and sync those changes with the container.

If we edit the files inside the App folder on the local machine, the same changes will be reflected inside the App folder that is running on the containers. Hence we can modify the application contents inside the containerized environments.

Task 2: Create or use existing three EC2 instances on AWS for Dev, Testing and Staging environment.

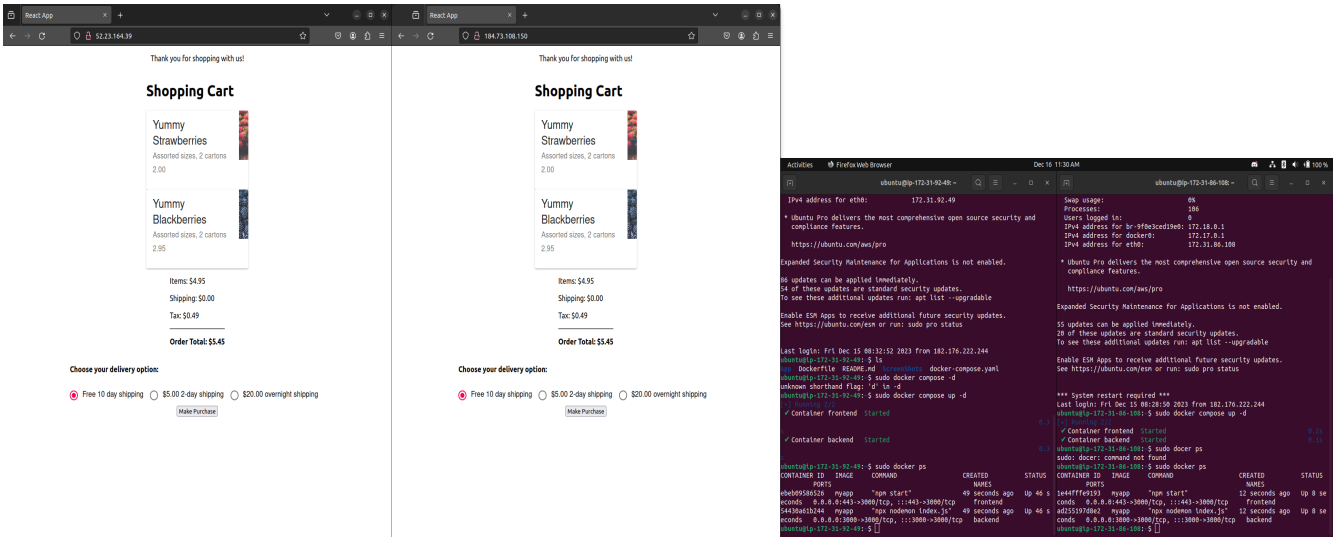
1. Please follow instructions from Task 2 of your previous assignment in case you are creating new EC2 instances.

   For this task we will create new EC2 instances instead of the ones that we have used for assignment 2. In the screenshot below we can see both the instances.



   Next step will be to setup dependencies on both the system so that we can run our React/Node application on docker containers on the EC2 instances that we have just created.

   In the screenshot below we can see that we can run the application using docker containers on both our EC2 instances which means that we have configured them correctly and have setup all the dependencies.



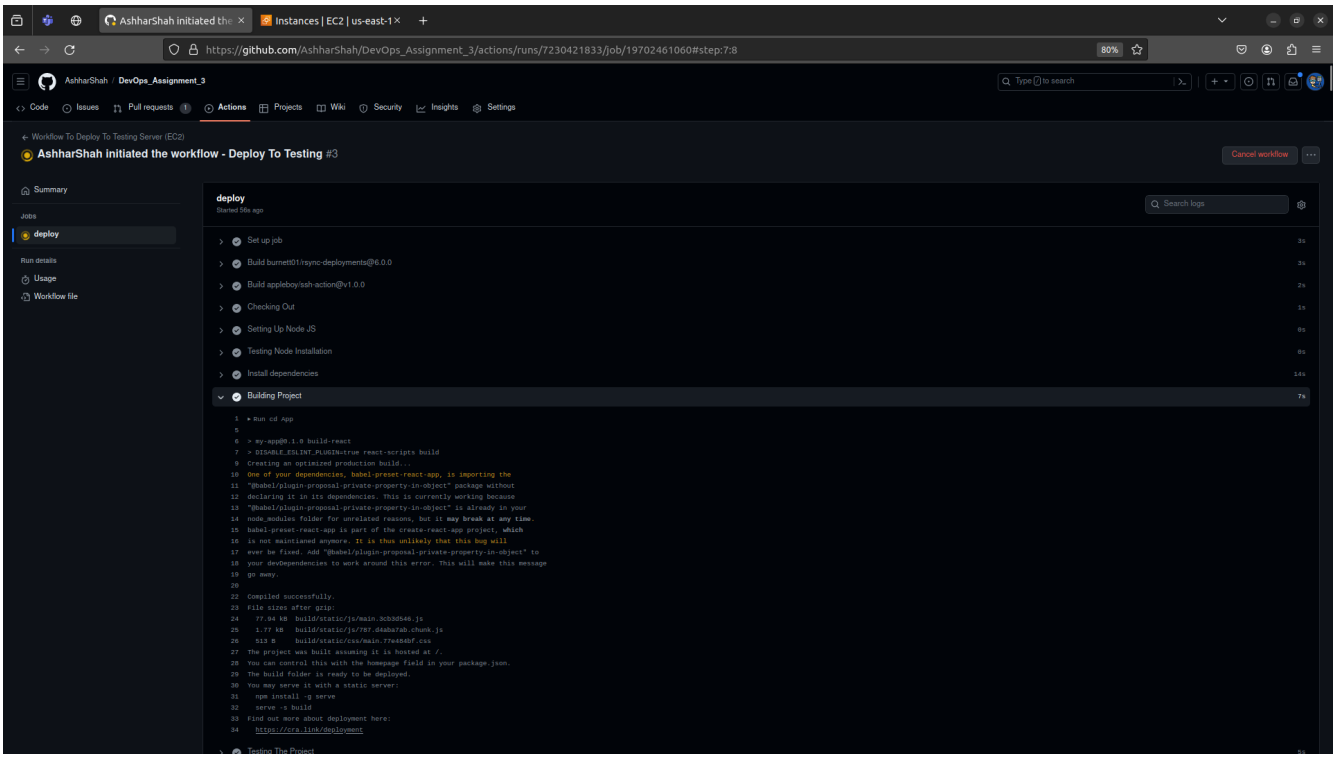
On the left we can see our Testing server and on the Right we can see our Staging server, both running the React application using the compose file that we setup in Part 1 above.

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:

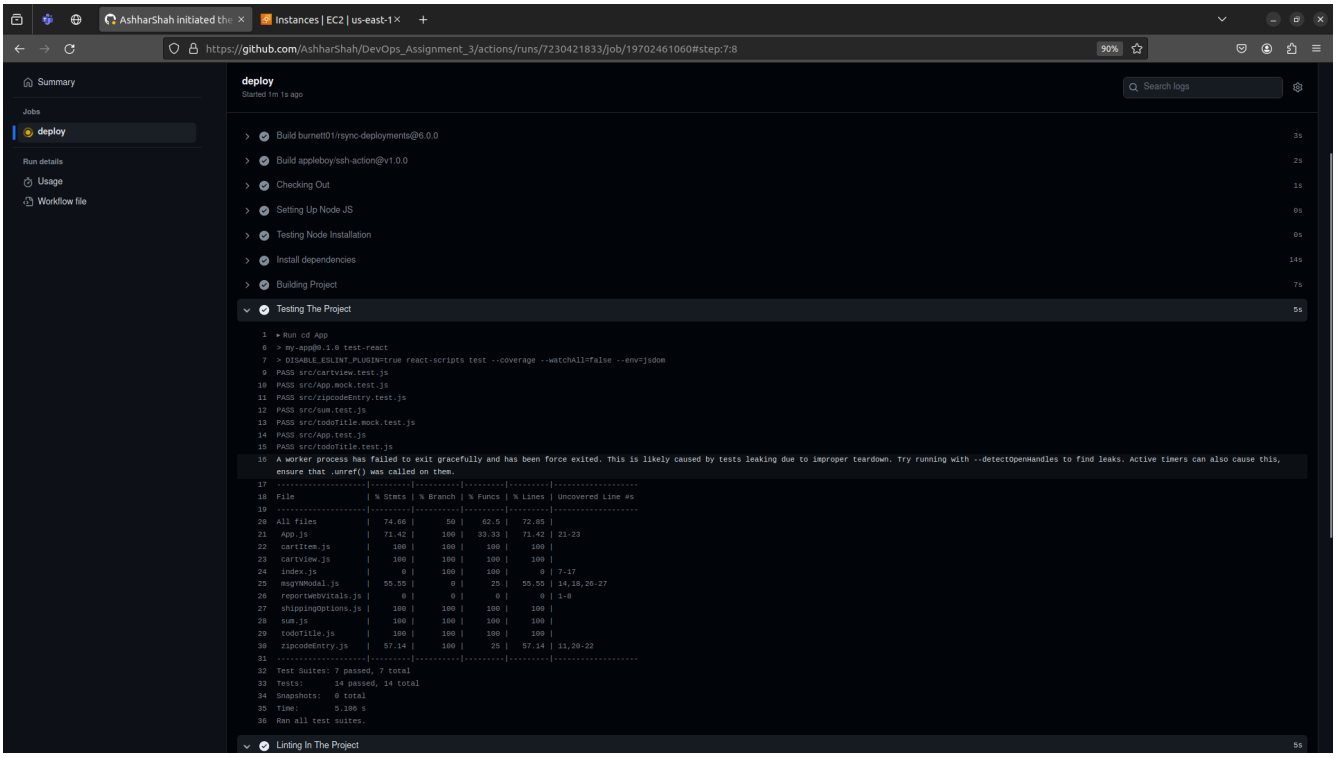
   (1) Build your project: Compile the source code and build the application or software. This step ensures that the code can be successfully compiled.

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



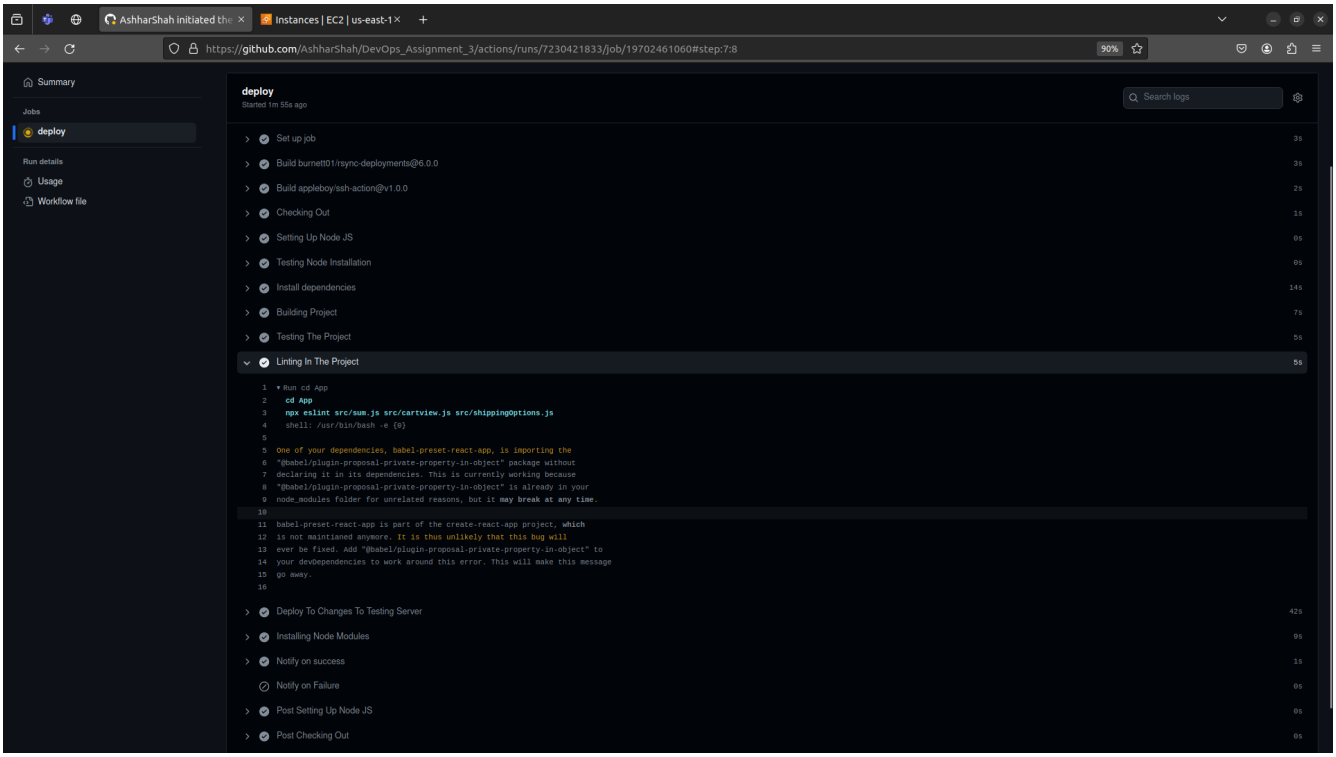
(2) Unit Testing: Run unit tests to verify that individual components or functions of the code work as expected. Unit tests are typically fast and focused on specific functionalities. (you must choose a project which has a few unit test cases, or you must create your own).

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



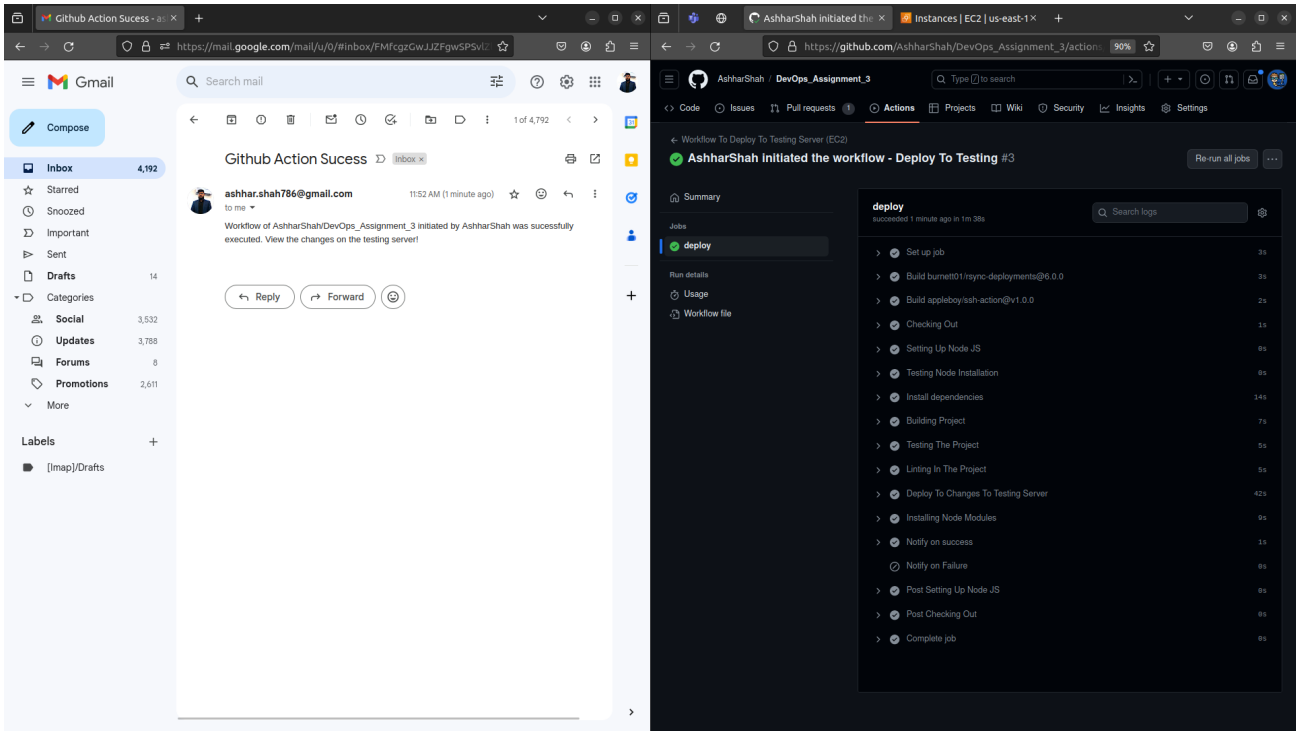
(3) Code Analysis/Linting: Perform static code analysis or linting to check for code quality, adherence to coding standards, and potential issues. This step helps maintain a consistent and high-quality code-base.

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.

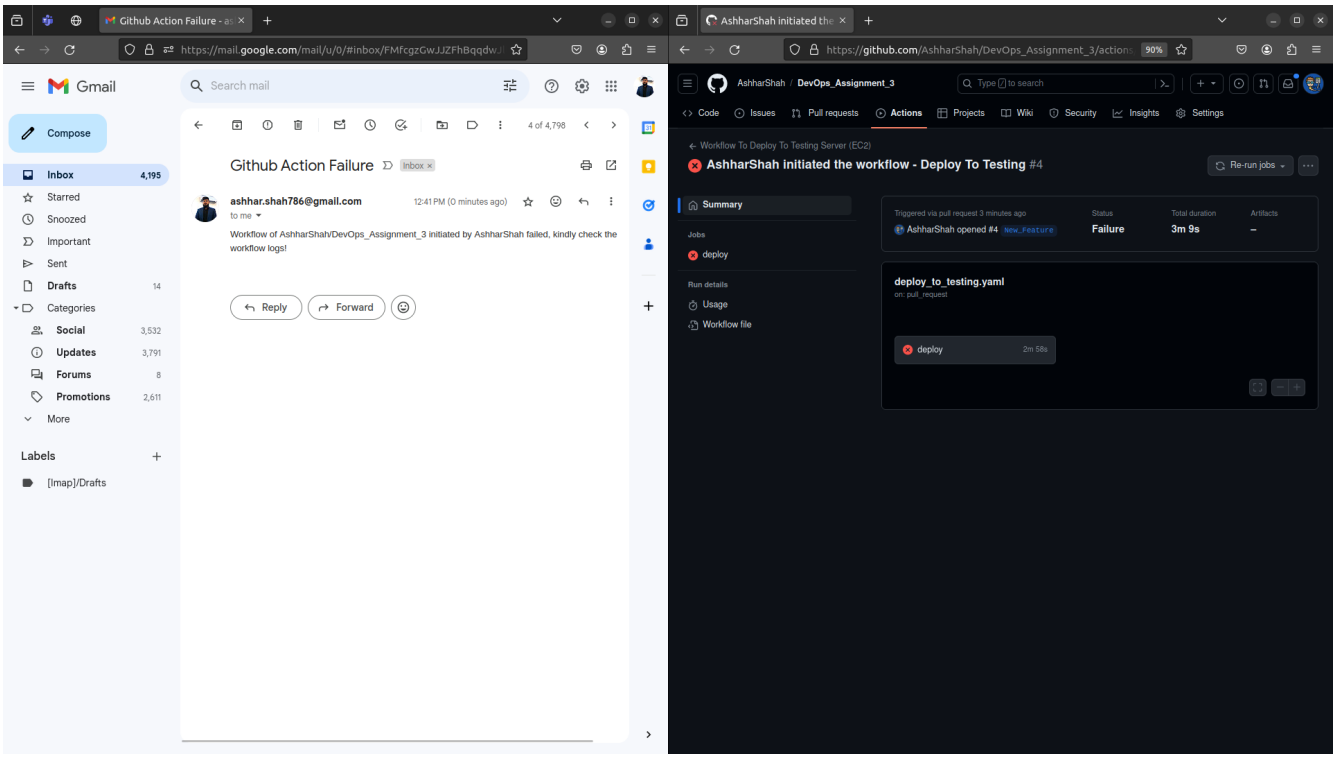


(4) Notification: Your workflow must send an email to me and the developer who created a pull request if the workflow fails. Moreover, it must send an email to the QA (to me) if it successfully deploys your change on the instance with all details how the QA can access the deployed app feature for testing.

Success:



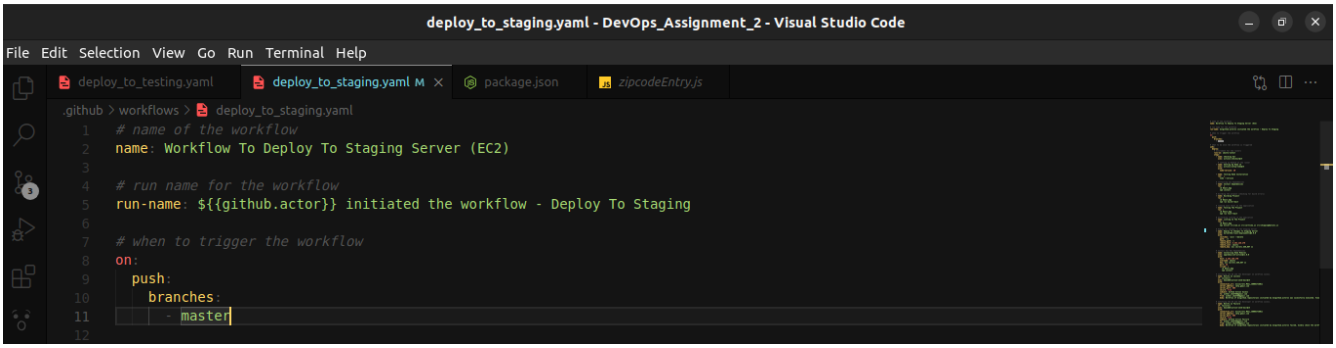
Failure:


Github Action Failure - a
https://mail.google.com/mail/u/0/#inbox/FMfcgzGwJJZFhBqqdwJ
Gmail
Search mail
Compose
Inbox 4,195
Starred
Snoozed
Important
Sent
Drafts 14
Categories
Social 3,532
Updates 3,791
Forums 8
Promotions 2,611
More
Labels
[Imap]/Drafts
4 of 4,798
Github Action Failure
Inbox ×
ashhar.shah786@gmail.com
12:41 PM (0 minutes ago)
to me
Workflow of AshharShah/DevOps_Assignment_3 initiated by AshharShah failed, kindly check the workflow logs!
Reply
Forward
AshharShah initiated the
https://github.com/AshharShah/DevOps_Assignment_3/actions
90%
AshharShah / DevOps_Assignment_3
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
Workflow To Deploy To Testing Server (EC2)
AshharShah initiated the workflow - Deploy To Testing #4
Re-run jobs
Summary
Jobs
deploy
Run details
Usage
Workflow file
Triggered via pull request 3 minutes ago
AshharShah opened #4 New_Feature
Status
Failure
Total duration
3m 9s
Artifacts
–
deploy_to_testing.yaml
on: pull_request
deploy
2m 58s

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.

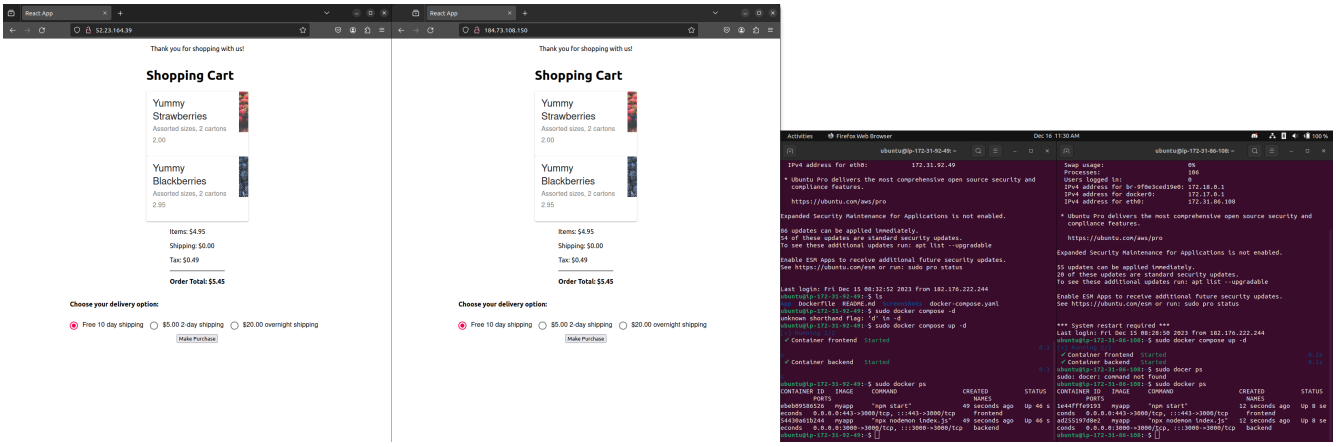


WORKFLOW DEMONSTRATION

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.

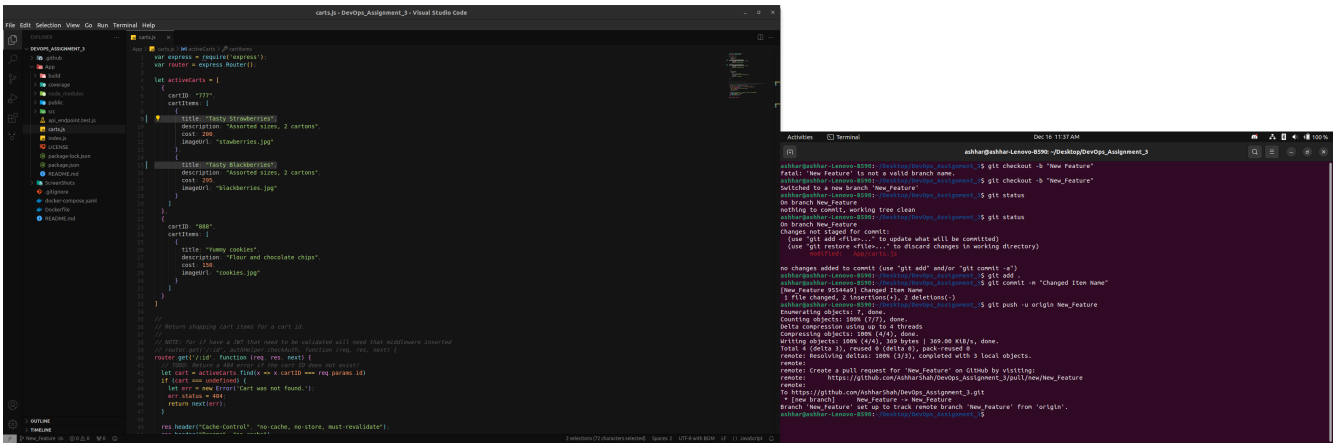
NOTE: The browser window on the Left represents the Testing Server while the browser window on the Right represents the Staging Server (SAME FOR THE CLI WINDOWS)

1. **Starting The Server & Application**
   - Our first step will be to start the EC2 instance's so that we can connect to them using our public IP's.
   - Next we will use the compose file to start our containers for the backend and the frontend that contains our react application. We will use port and volume mapping so that we can connect to the application and modify the code inside the containers.
   - To start the containers we will ssh into the servers and use the following command: "*sudo docker compose up -d*"



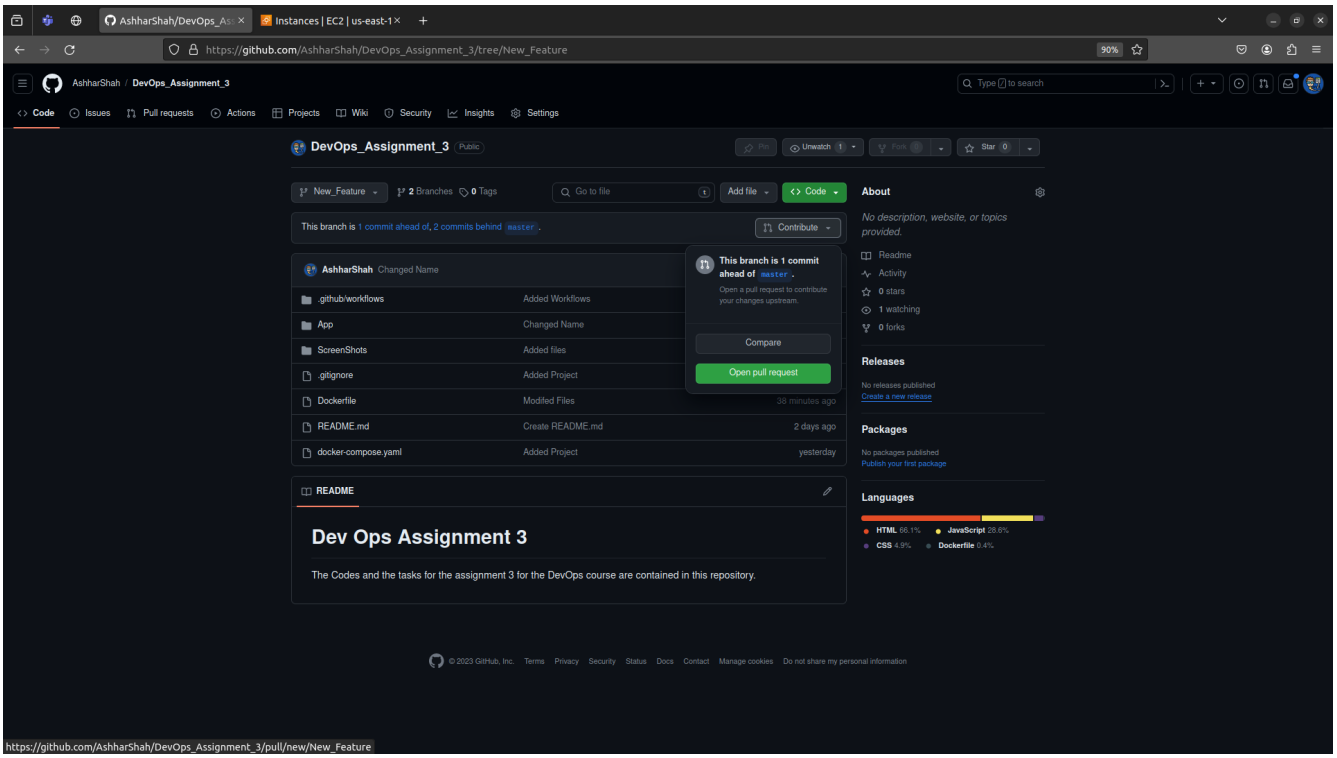
2. Modify The Code & Push To Origin

- Next we will create a new branch, in the example I have created the branch “New_Feature” 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 “New_Feature” branch on the “origin” server.

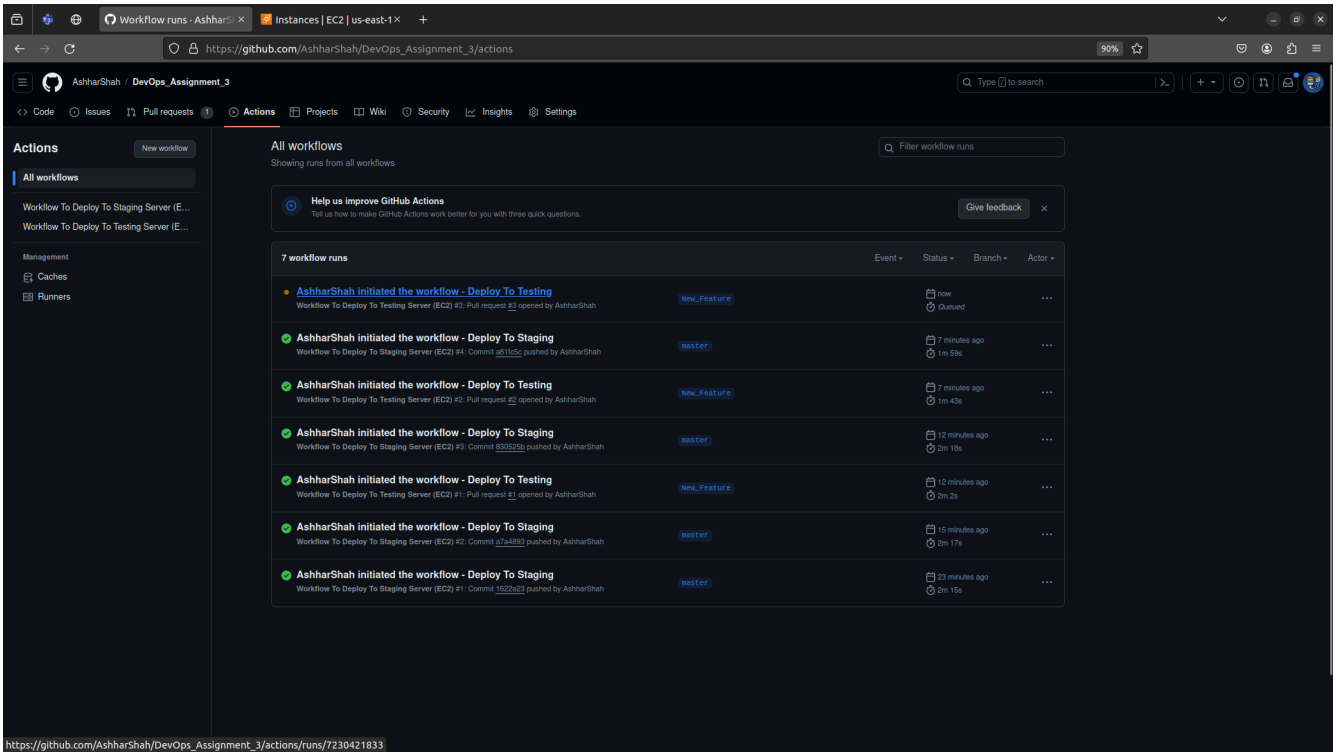


3. Create A Pull-Request

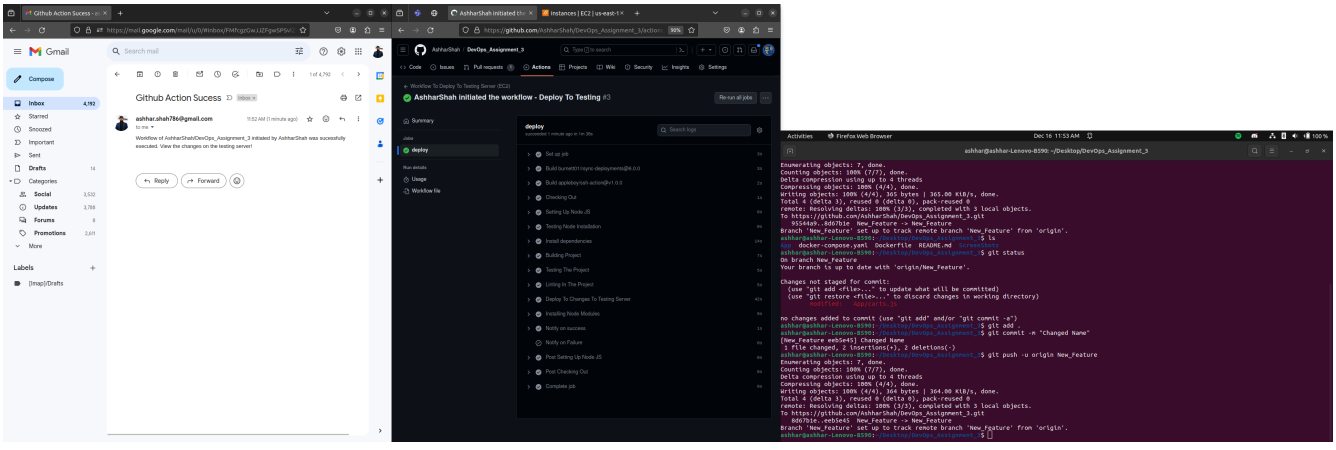
- Now that I have pushed the changes to the “origin” server. I will get an option to generate a pull-request as the “New_Feature” branch is ahead of the “master” branch.



- The pull-request when generated will trigger the workflow “Deploy To Testing” which we can view in the Actions tab of our repository.

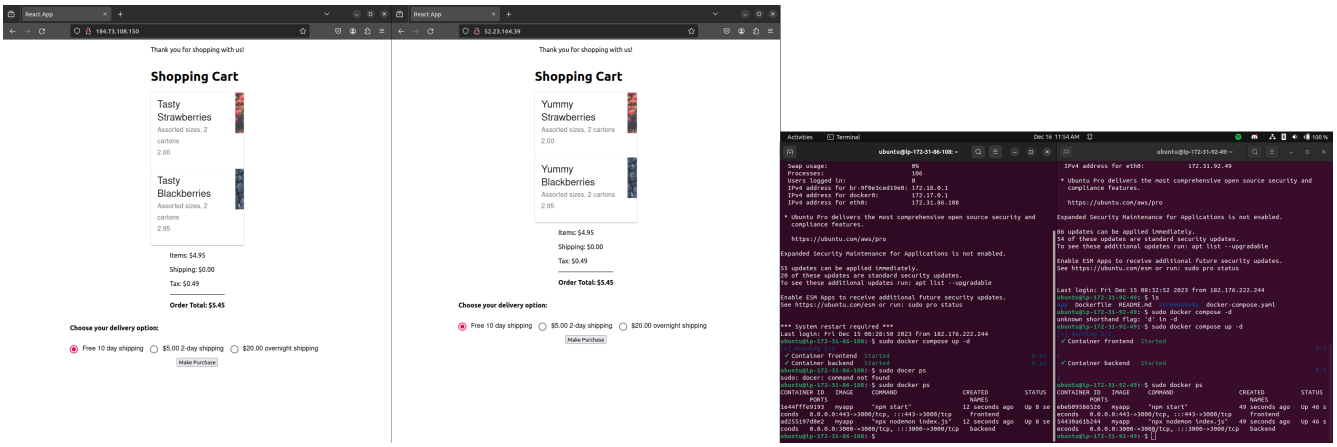


- This workflow when successfully executed, will send an email to the developer and the QA (me) which will prompt us that there were no errors. (If there was any failure it would also send an email saying that there was a error during the workflow’s execution).



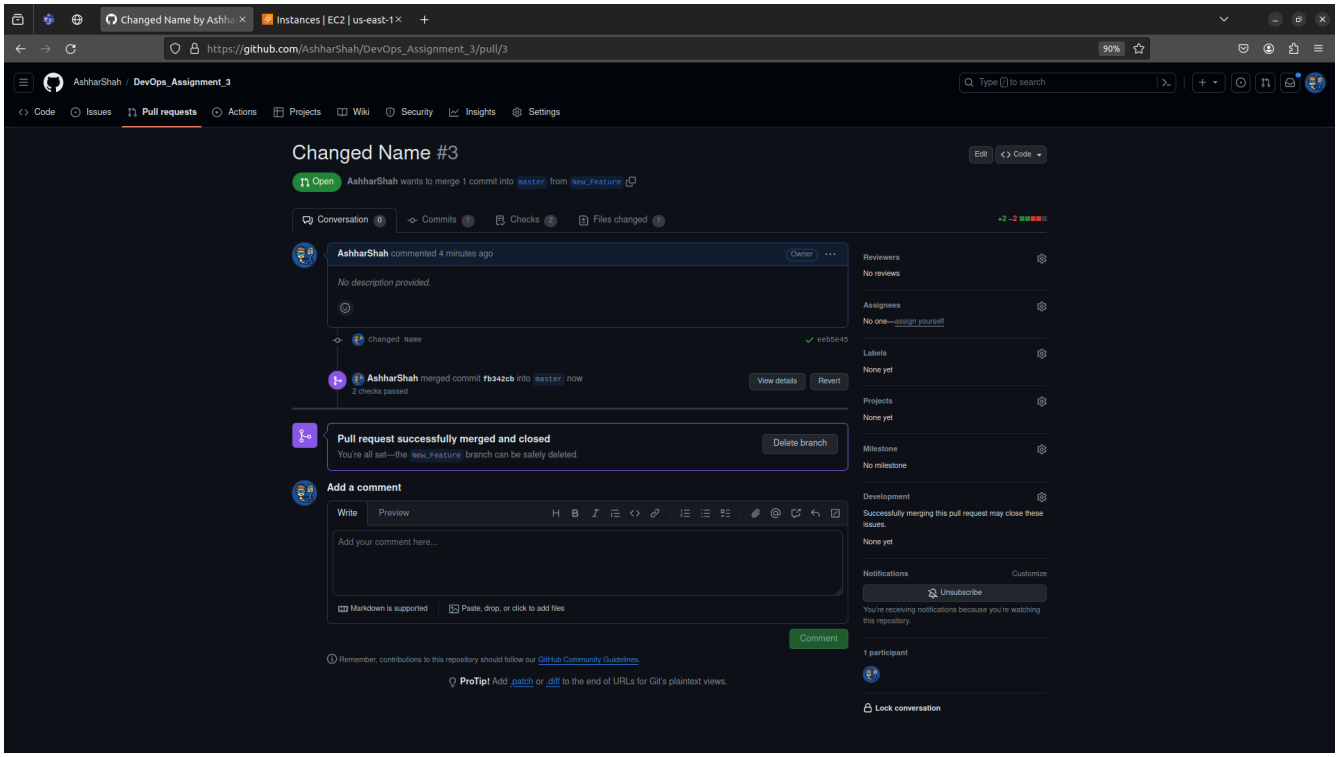
4. Verify Testing Changes

- 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.



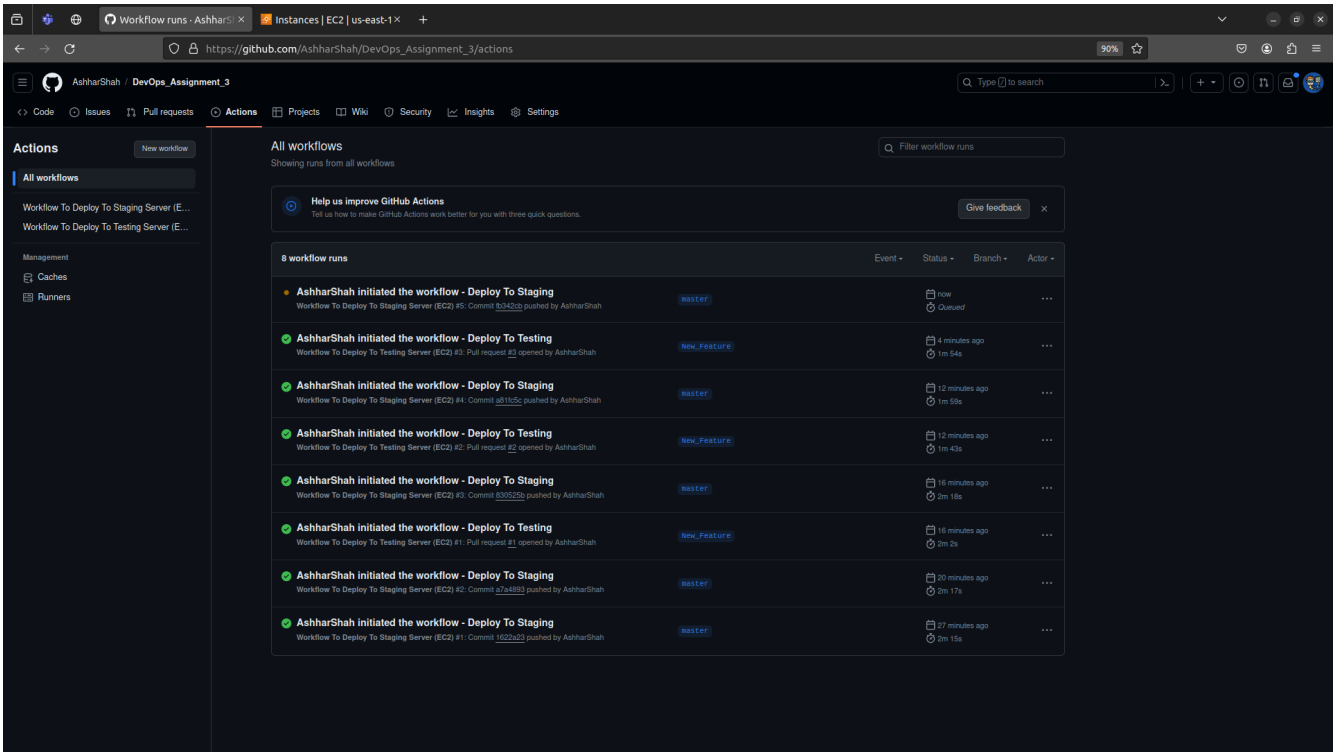
5. Merge Changes

- 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.

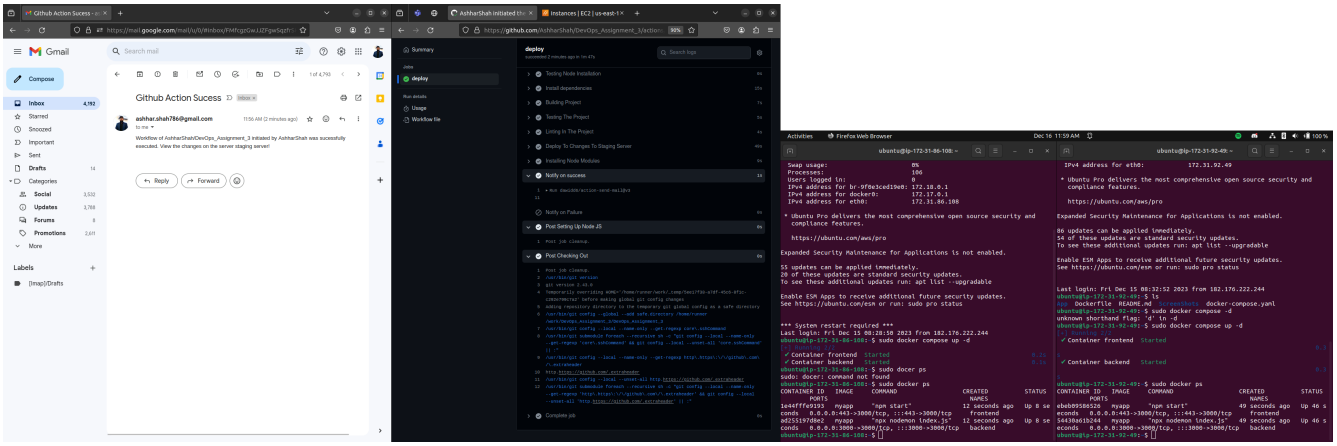


6. Merge Changes Trigger Workflow

- 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).

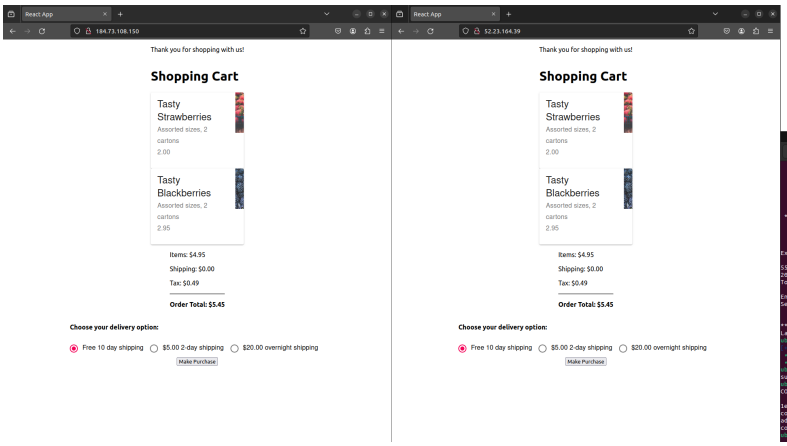


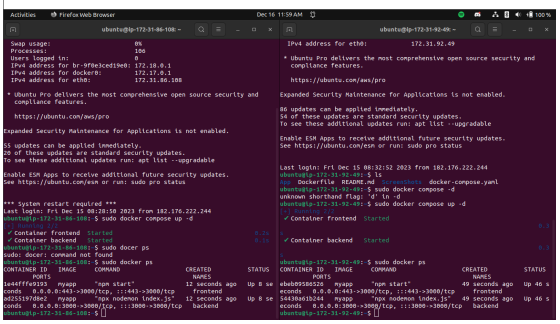
- The workflow once completed will send an email to the QA prompting him that there were no issues and all changes were successfully executed on the staging server which he can view.

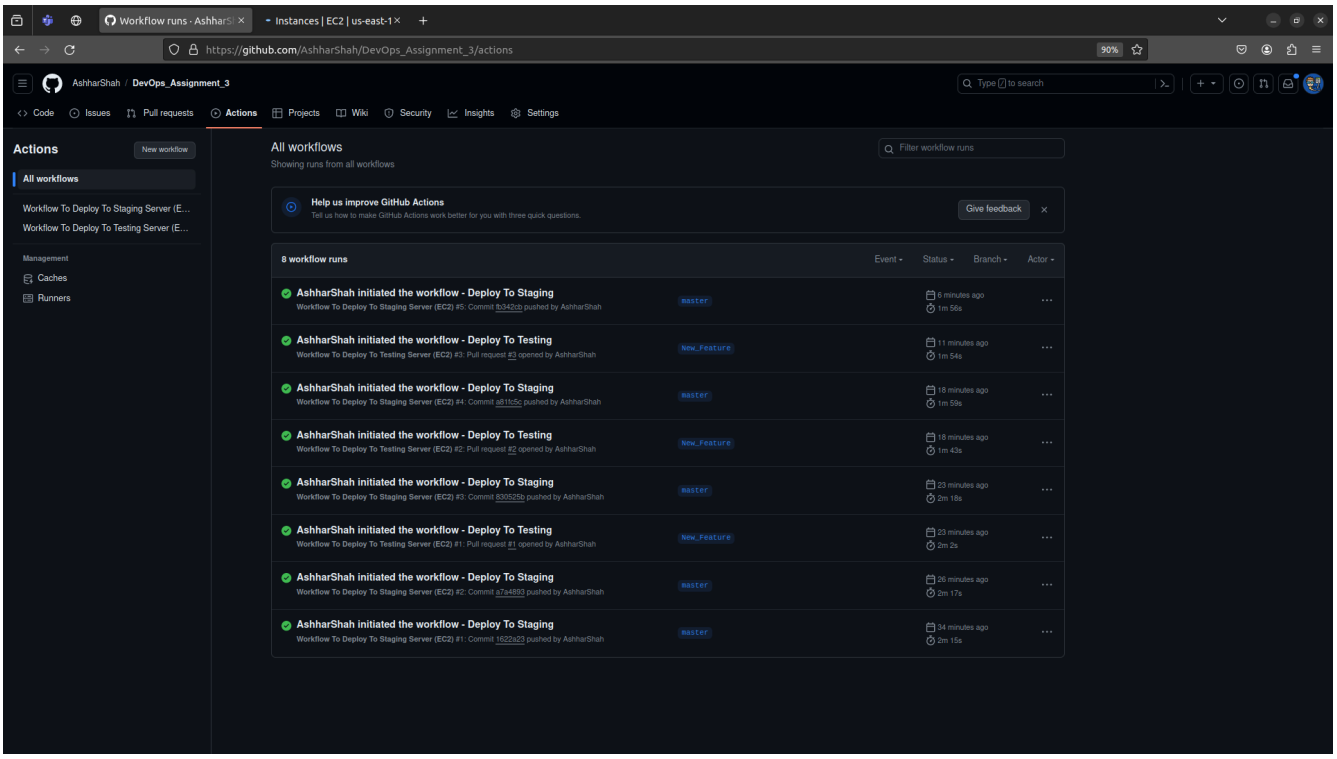


7. Client Verification

- 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

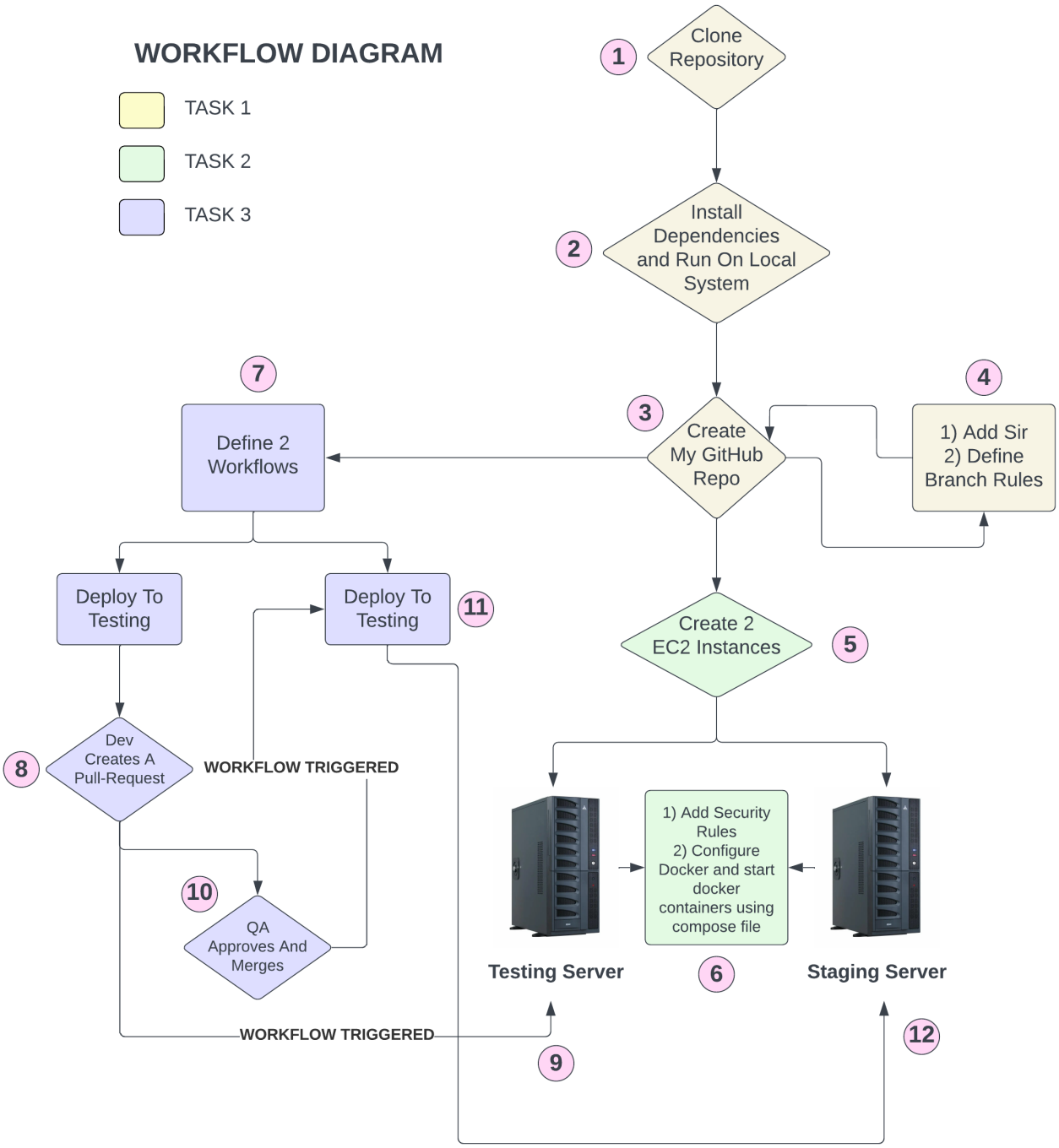
During the project I faced a few challenges which are listed in the points below.

1. **Application Startup Automation:** I had difficulty in setting up a connection between files on the host and the files on the container such that when I modify the application files on the host machine, the same files get modified in the container application. For this purpose I used volume mapping inside our docker containers.

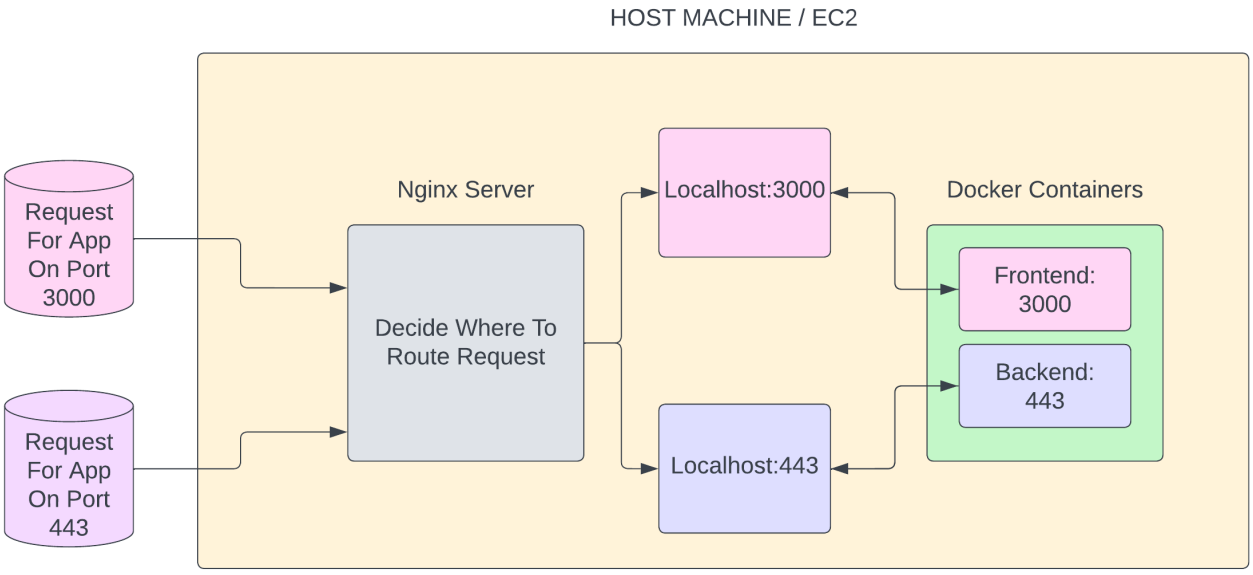
2. **Linting In Application:** In the application there was no linting steps configured by default so I had to do research on how to use eslint to perform linting in React/Node applications to check if the syntax is up-to the required standards.

3. **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



DOCKER DIAGRAM



In the above image, we can see that on the EC2 Host machine we have Nginx and Docker.

Docker is running two applications called the frontend and the backend in their separate containers. The two applications are also mapped onto the port 3000 and port 443 of the host machine. Using these two ports the user can view the frontend and the backend application.

When a user requests for a application webpage, the request will first be passed to the nginx server which is listening for incoming requests. It will then route it to the correct port on the Host machine.