

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 whenever there is a new feature being developed on the code-base being modified.

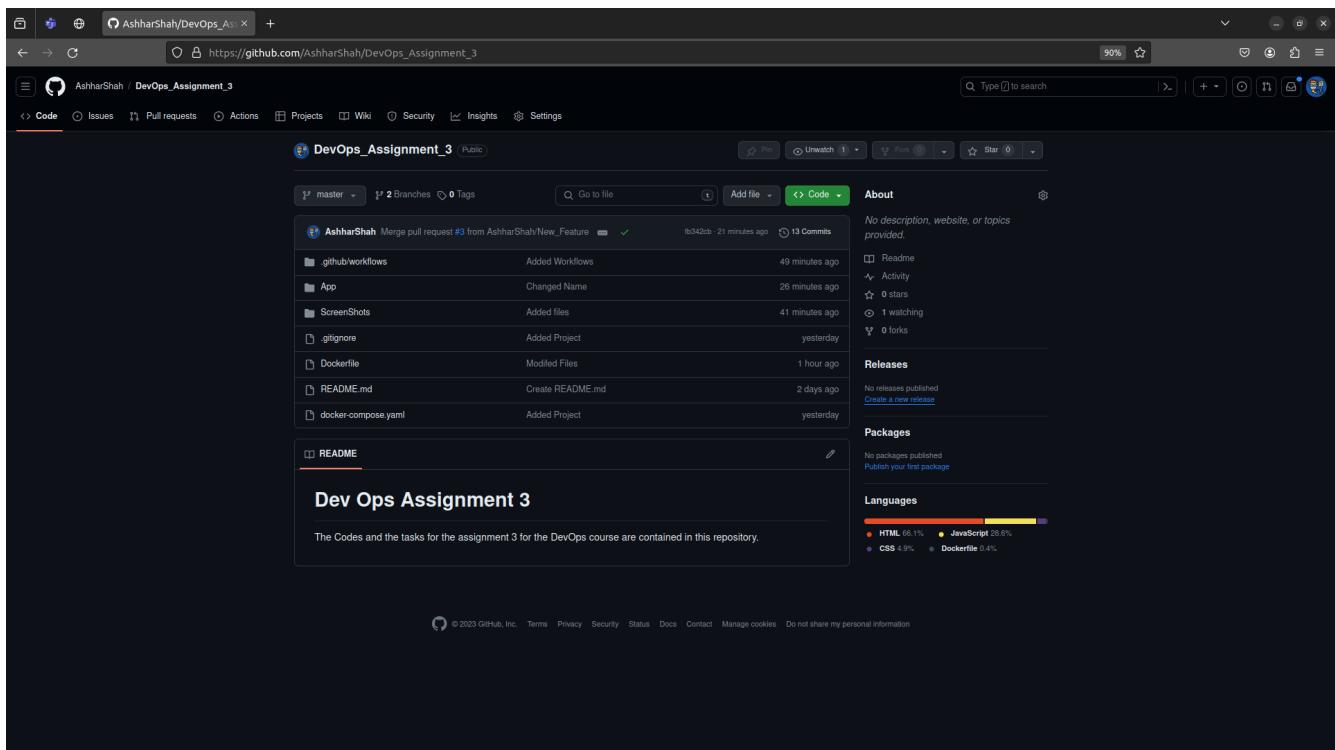
The application for both the servers will be running on containers using docker, the user will be able to access the application with the help of port mapping in which we will map port 3000 of the docker application to the port 3000 of the servers. We will also use Nginx for the server to be able to 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.

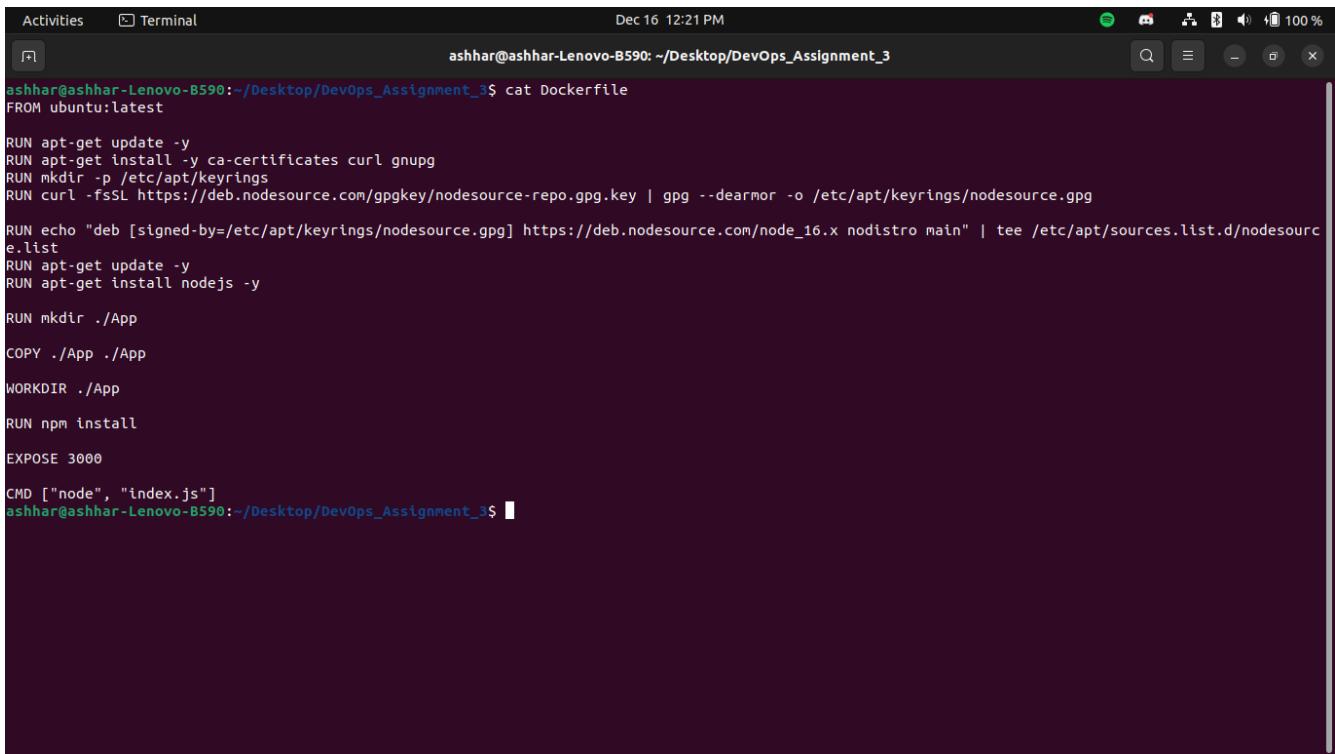


2. Deploy both frontend and backend services on local machine as separate containers named frontend & backend. Create a separate container for the database if it is being used by your application.

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.



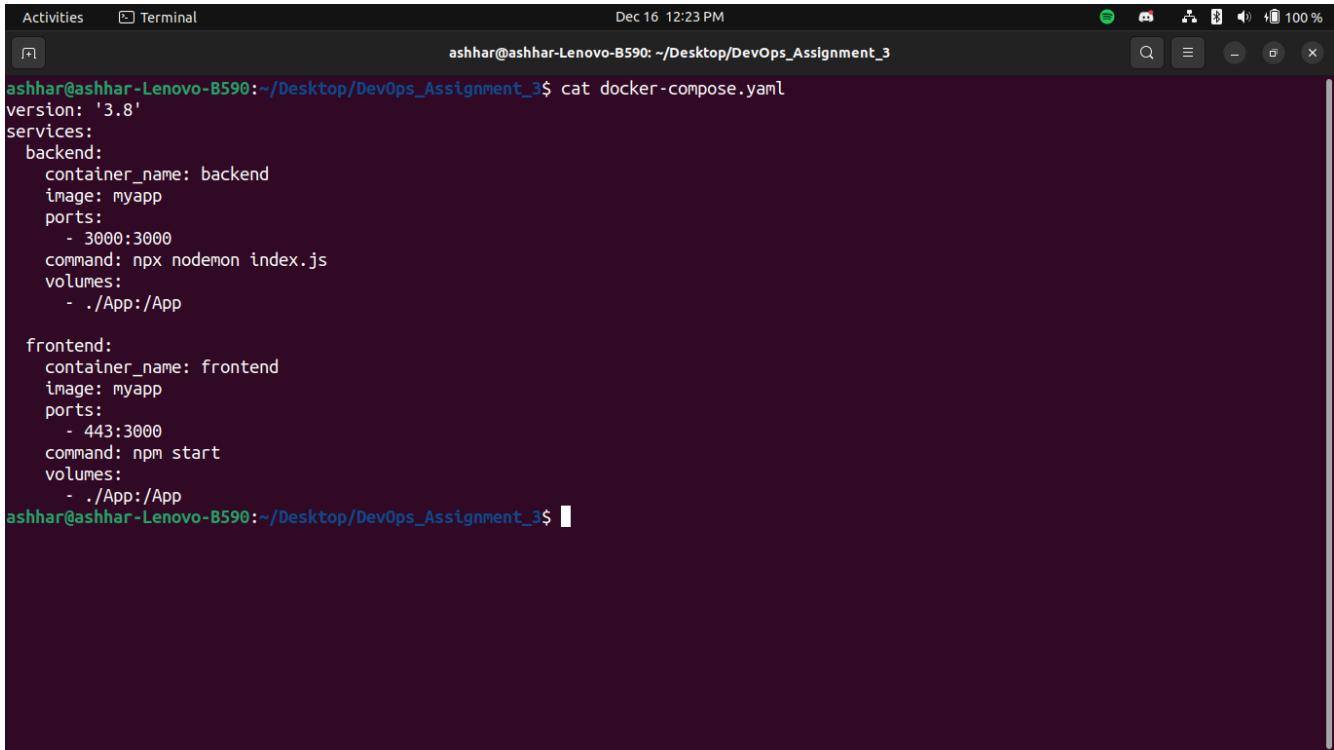
A screenshot of a Linux terminal window titled "ashhar@ashhar-Lenovo-B590: ~/Desktop/DevOps\_Assignment\_3". The terminal shows the following Dockerfile content:

```
FROM ubuntu:latest

RUN apt-get update -y
RUN apt-get install -y ca-certificates curl gnupg
RUN mkdir -p /etc/apt/keyrings
RUN curl -fsSL https://deb.nodesource.com/gpgkey/nodesource-repo.gpg.key | gpg --dearmor -o /etc/apt/keyrings/nodesource.gpg
RUN echo "deb [signed-by=/etc/apt/keyrings/nodesource.gpg] https://deb.nodesource.com/node_16.x nodistro main" | tee /etc/apt/sources.list.d/nodesource.list
RUN apt-get update -y
RUN apt-get install nodejs -y

RUN mkdir ./App
COPY ./App ./App
WORKDIR ./App
RUN npm install
EXPOSE 3000
CMD ["node", "index.js"]
```

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.



A screenshot of a Linux terminal window titled "ashhar@ashhar-Lenovo-B590: ~/Desktop/DevOps\_Assignment\_3\$". The window shows the command "cat docker-compose.yaml" being run, displaying its contents. The yaml file defines two services: "backend" and "frontend". The "backend" service uses the "myapp" image, runs on port 3000, and has a command of "npx nodemon index.js". It also maps the local directory "./App" to the container's "/App". The "frontend" service also uses the "myapp" image, runs on port 443, and has a command of "npm start". It maps the local directory "./App" to the container's "/App".

```
ashhar@ashhar-Lenovo-B590:~/Desktop/DevOps_Assignment_3$ cat docker-compose.yaml
version: '3.8'
services:
  backend:
    container_name: backend
    image: myapp
    ports:
      - 3000:3000
    command: npx nodemon index.js
    volumes:
      - ./App:/App

  frontend:
    container_name: frontend
    image: myapp
    ports:
      - 443:3000
    command: npm start
    volumes:
      - ./App:/App
ashhar@ashhar-Lenovo-B590:~/Desktop/DevOps_Assignment_3$
```

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.

The image shows two separate screenshots of the AWS EC2 Instance Details page, each displaying the configuration for a different EC2 instance.

**Instance 1 (Testing Server - DevOps 2):**

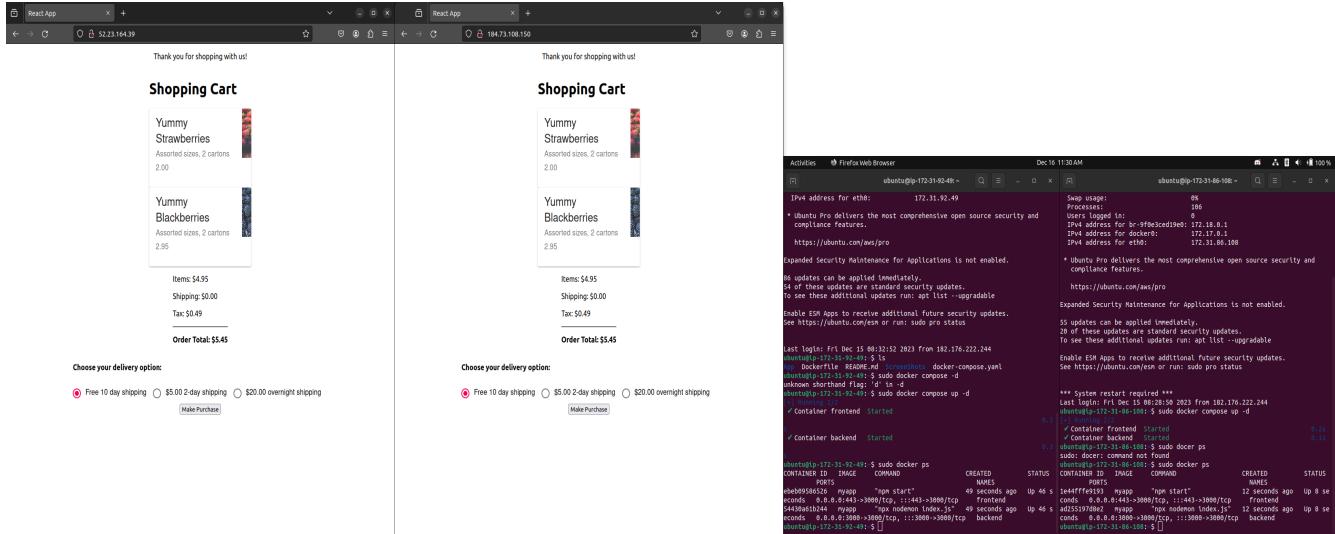
- Instance ID: i-0bb95e2822cb17ffd
- Public IPv4 address: 184.73.108.150
- Private IPv4 addresses: 172.3.186.108
- Public IPv4 DNS: ec2-184-73-108-150.compute-1.amazonaws.com
- Private IP DNS name (IPv4 only): ip-172-31-86-108.ec2.internal
- Instance state: Running
- Instance type: t2.micro
- VPC ID: vpc-04e422a944d960181
- Auto-assigned IP address: 184.73.108.150 (Public IP)
- IAM Role: Not specified
- Subnet ID: subnet-017f596e4031e40f5
- Auto Scaling Group name: Not specified

**Instance 2 (Staging Server - DevOps 2):**

- Instance ID: i-065e1469d27eb06c0
- Public IPv4 address: 52.23.164.39
- Private IPv4 addresses: 172.31.92.49
- Public IPv4 DNS: ec2-52-23-164-39.compute-1.amazonaws.com
- Private IP DNS name (IPv4 only): ip-172-31-92-49.ec2.internal
- Instance state: Running
- Instance type: t2.micro
- VPC ID: vpc-04e422a944d960181
- Auto-assigned IP address: 52.23.164.39 (Public IP)
- IAM Role: Not specified
- Subnet ID: subnet-017f596e4031e40f5
- Auto Scaling Group name: Not specified

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.

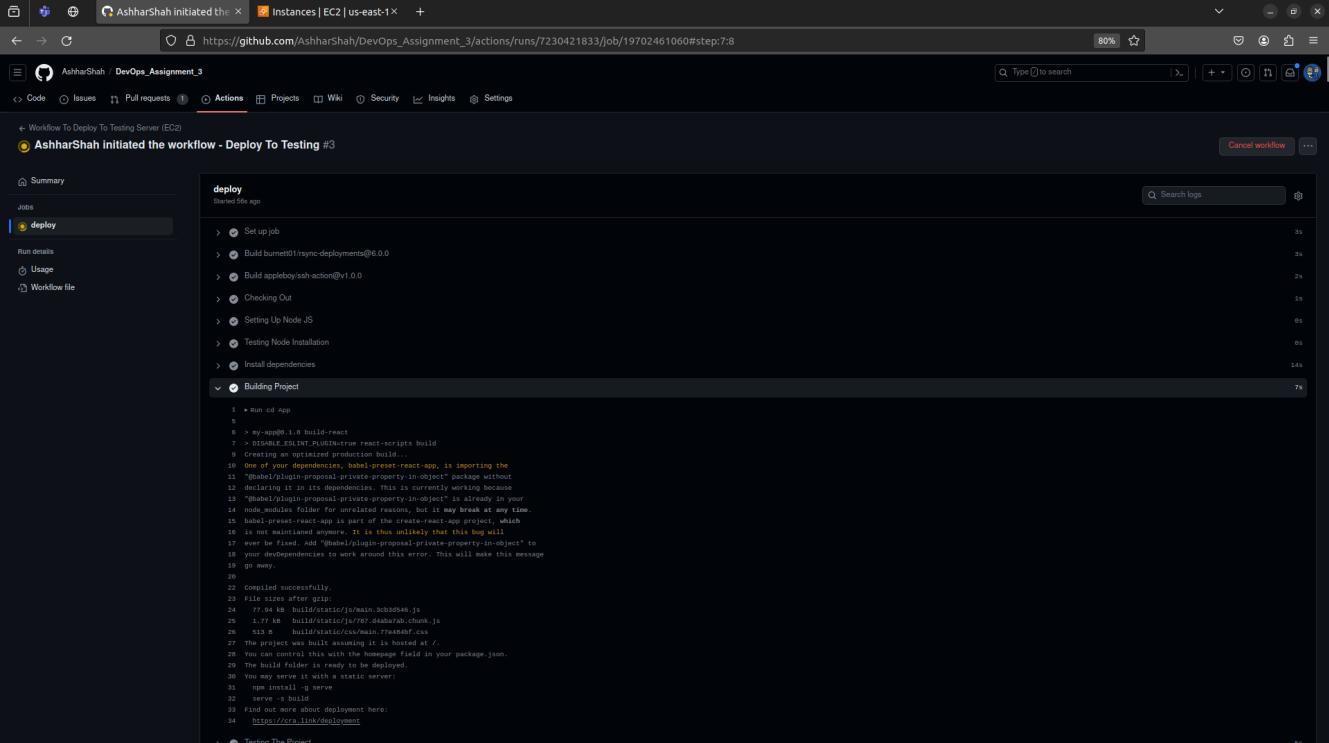
## Task 3: Automating application from development to deployment.

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>](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*



The screenshot shows a GitHub Actions workflow named "Workflow To Deploy To Testing Server (EC2)". The workflow has a single job named "deploy". The steps in the workflow are:

- Set up job
- Build burnetto1sync-deployments@6.0.0
- Build appleboy/ssh-action@v1.0.0
- Checking Out
- Setting Up Node JS
- Testing Node Installation
- Install dependencies
- Building Project

The "Building Project" step shows the command output:

```
1 + Run cd app
2
3 6 my-app@0.1.0 build-react
4 > 7 > 8 > 9 > 10 > 11 > 12 > 13 > 14 > 15 > 16 > 17 > 18 > 19 > 20 > 21 > 22 > 23 > 24 > 25 > 26 > 27 > 28 > 29 > 30 > 31 > 32 > 33 > 34
```

The log indicates a warning about a babel-preset-react-app dependency issue:

One of your dependencies, babel-preset-react-app, is importing the "Babel/plugin-proposal-private-property-in-object" package without declaring it in its dependencies. This is currently working because "Babel/plugin-proposal-private-property-in-object" is already in your node\_modules. However, this is a breaking change in Babel 7. It is recommended to declare this dependency. babel-preset-react-app is part of the create-react-app project, which is not maintained anymore. It is thus unlikely that this bug will ever be fixed. Add "Babel/plugin-proposal-private-property-in-object" to your dependencies to work around this error. This will make this message go away.

The workflow completed successfully with the following file sizes:

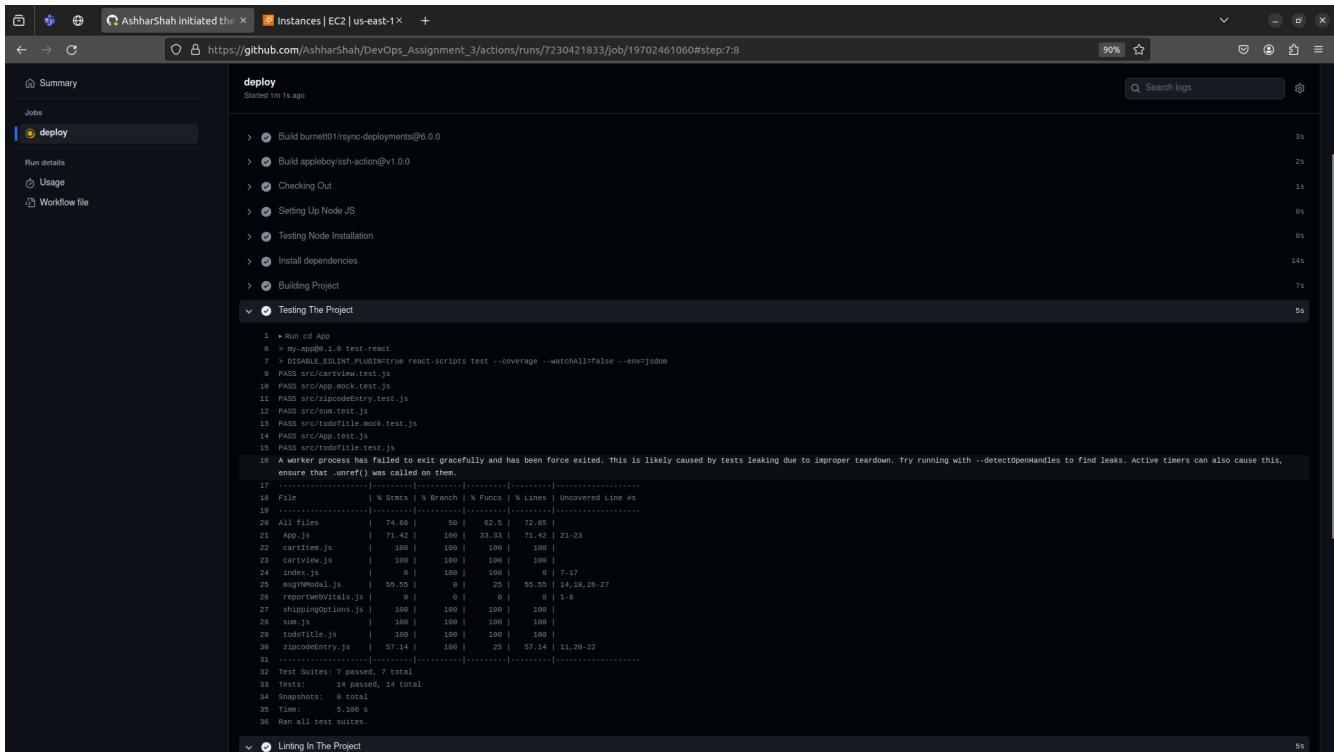
- 77.94 kB build/static/js/main.2c330446.js
- 1.73 kB build/static/js/reroute.abbabf.chunk.js
- 513 kB build/static/css/main.77876d9f.css

The final message is: "The project was built assuming it is hosted at /."

(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*

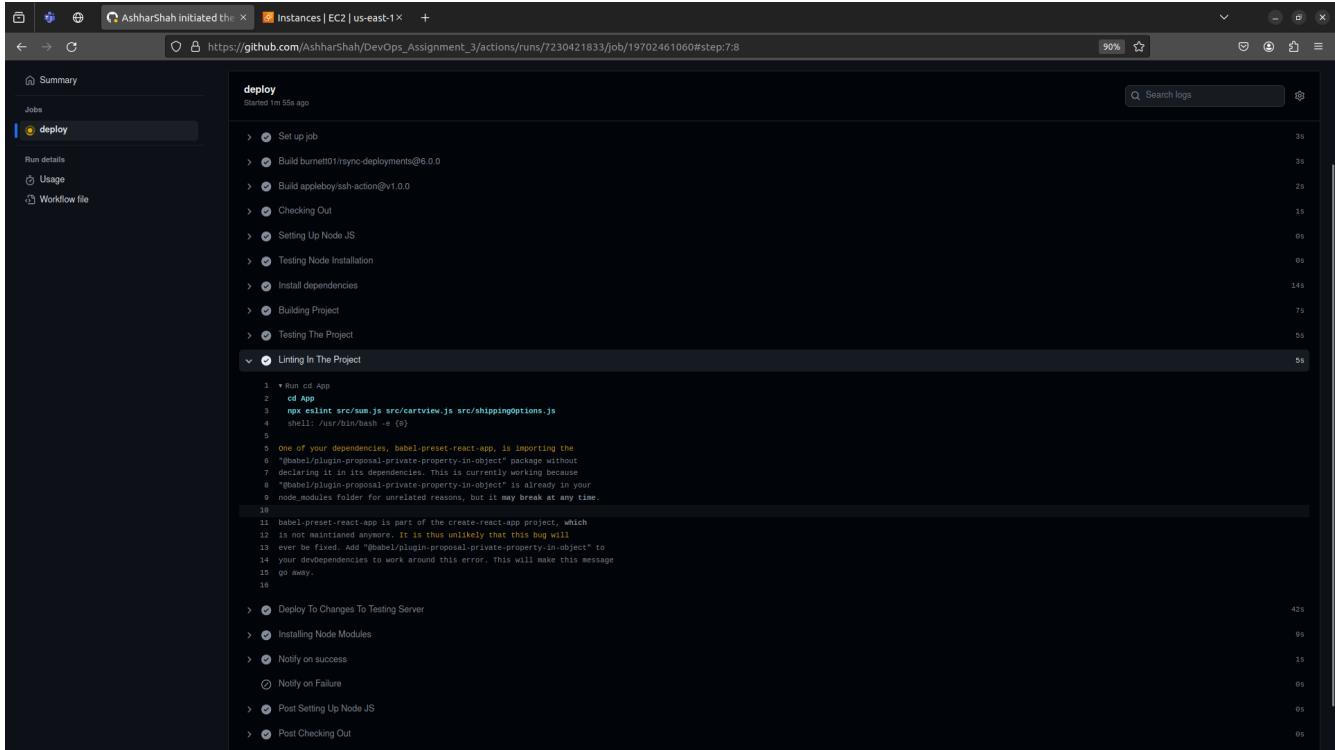


The screenshot shows a GitHub Actions workflow named "deploy" that has just started 1m ago. The workflow consists of several steps: "Build burnetto1/rsync-deployments@6.0.0", "Build appleboy/ssh-action@v1.0", "Checking Out", "Setting Up Node JS", "Testing Node Installation", "Install dependencies", "Building Project", and finally "Testing The Project". The "Testing The Project" step is currently active, showing the command output for running unit tests.

```
1  # Run cd APP
2  > my-app@0.1.0 test-react
3  > DISABLE_ESLINT_PLUGIN=true react-scripts test --coverage --watchAll=false --env=jsdom
4  PASS src/cartview.test.js
5  PASS src/cartitem.mock.test.js
6  PASS src/cartitem.test.js
7  PASS src/calculateEntry.test.js
8  PASS src/sum.test.js
9  PASS src/todoTitle.mock.test.js
10 PASS src/todoTitle.test.js
11 PASS src/App.test.js
12 PASS src/zipcodeEntry.test.js
13 PASS src/cartitem.test.js
14 PASS src/cartview.test.js
15 PASS src/sum.test.js
16 A worker process has failed to exit gracefully and has been force exited. This is likely caused by tests leaking due to improper teardown. Try running with --detectOpenHandles to find leaks. Active timers can also cause this, ensure that .unref() was called on them.
17 -----
18 File          | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s |
19 -----
20 All files    | 74.66 |      50 | 62.0 | 72.88 |
21 App.js       | 71.42 |     100 | 33.33 | 71.42 | 21-23
22 cartitem.js | 100 |     100 | 100 | 100 |
23 cartview.js | 100 |     100 | 100 | 100 |
24 index.js    | 0 |     100 | 100 | 0 | 7-17
25 zipcodeEntry.js | 55.55 |      0 | 25 | 55.55 | 14,16,20-27
26 sum.mock.js | 0 |      0 | 0 | 0 | 1-6
27 sumEntryMocks.js | 100 |     100 | 100 | 100 |
28 shipperOptions.js | 100 |     100 | 100 | 100 |
29 sum.js       | 100 |     100 | 100 | 100 |
30 todoTitle.js | 100 |     100 | 100 | 100 |
31 zipcodeEntry.js | 57.14 |     100 | 25 | 57.14 | 11,29-22
32 -----
33 Test Suites: 7 passed, 7 total
34 Tests: 14 passed, 14 total
35 Snapshots: 0 total
36 Time: 0.396 s
37 Ran all test suites.
```

(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](#).



The screenshot shows a GitHub Actions deployment log titled "deploy" initiated by "AshharShah". The log details the execution of various steps, with the "Linting In The Project" step expanded to show its command and output. The command is:

```
1 * Run cd App
2 cd App
3 npx eslint src/sum.js src/cartview.js src/shippingoptions.js
4 shell: /usr/bin/bash -e {0}
5
6 one of your dependencies, babel-preset-react-app, is importing the
7 @babel/plugin-proposal-private-property-in-object package without
8 declaring it in its dependencies. This is currently working because
9 @babel/plugin-proposal-private-property-in-object is already in your
10 node_modules folder for unrelated reasons, but it may break at any time.
11 babel-preset-react-app is part of the create-react-app project, which
12 is not maintained anymore. It is thus unlikely that this bug will
13 ever be fixed. Add "@babel/plugin-proposal-private-property-in-object" to
14 your devDependencies to work around this error. This will make this message
15 go away.
16
17 > ⚡ Set up job
18 > ⚡ Build Burnett01/rsync-deployments@6.0.0
19 > ⚡ Build sappleboy/ssh-action@v1.0.0
20 > ⚡ Checking Out
21 > ⚡ Setting Up Node JS
22 > ⚡ Testing Node Installation
23 > ⚡ Install dependencies
24 > ⚡ Building Project
25 > ⚡ Testing The Project
26 > ⚡ Linting In The Project
27 > ⚡ Deploy To Changes To Testing Server
28 > ⚡ Installing Node Modules
29 > ⚡ Notify on success
30 ⚡ Notify on Failure
31 > ⚡ Post Setting Up Node JS
32 > ⚡ Post Checking Out
```

(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:

The screenshot displays two browser windows side-by-side. The left window shows a Gmail inbox with a single email from 'ashhar.shah786@gmail.com' titled 'Github Action Success'. The email body contains the message: 'Workflow of AshharShah/DevOps\_Assignment\_3 initiated by AshharShah was successfully executed. View the changes on the testing server!'. Below the email are standard reply and forward buttons. The right window shows a GitHub Actions logs page for a workflow named 'Workflow To Deploy To Testing Server (EC2)'. The logs detail a successful deployment process, starting with 'Set up job' and ending with 'Complete job'. The logs also mention steps like 'Build', 'Testing Node Installation', and 'Notify on success'.

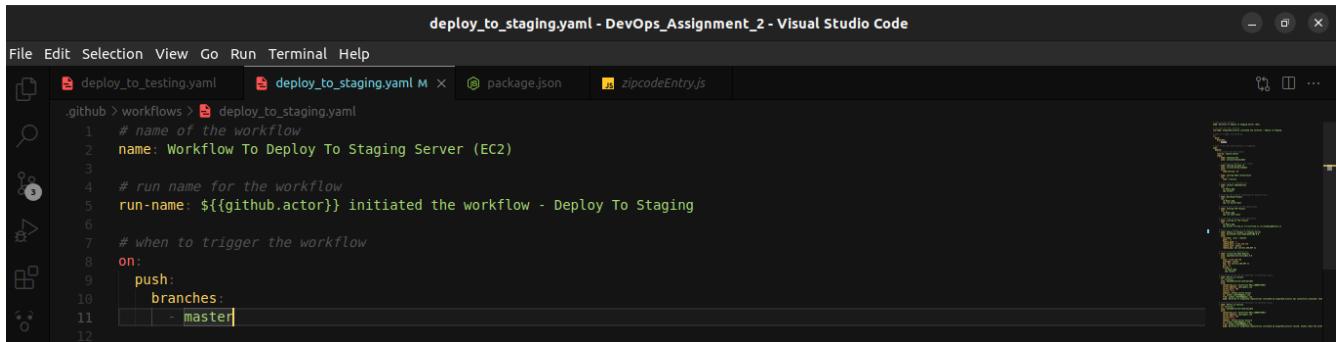
## Failure:

The image displays two adjacent browser windows. The left window is a Gmail inbox showing an email from 'ashhar.shah786@gmail.com' with the subject 'Github Action Failure'. The email body contains the message: 'Workflow of AshharShah/DevOps\_Assignment\_3 initiated by AshharShah failed, kindly check the workflow logs!'. The right window is a GitHub Actions workflow page for the repository 'AshharShah / DevOps\_Assignment\_3'. The workflow name is 'Workflow To Deploy To Testing Server (EC2)'. A red error icon indicates that the 'Deploy' job has failed. The status bar at the bottom of the GitHub window shows 'Failure'.

2. Assume that the QA validates all the changes made in a Testing environment and everything looks fine. Now let's say QA confirms 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 an 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>](http://<Instance_IP>). Your workflow must do all the tasks you did in your previous workflow.

For this we will create another workflow by 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.



```
deploy_to_staging.yaml - DevOps_Assignment_2 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
github > workflows > deploy_to_staging.yaml M package.json zipcodeEntry.js ...
1 # name of the workflow
2 name: Workflow To Deploy To Staging Server (EC2)
3
4 # run name for the workflow
5 run-name: ${github.actor} initiated the workflow - Deploy To Staging
6
7 # when to trigger the workflow
8 on:
9   push:
10     branches:
11       - master
```

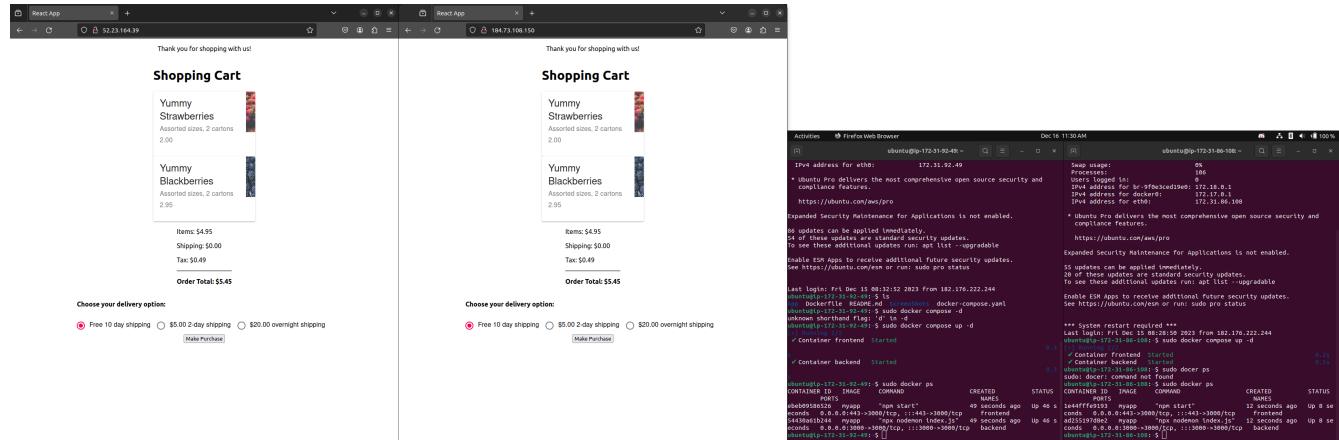
# WORKFLOW DEMONSTRATION

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

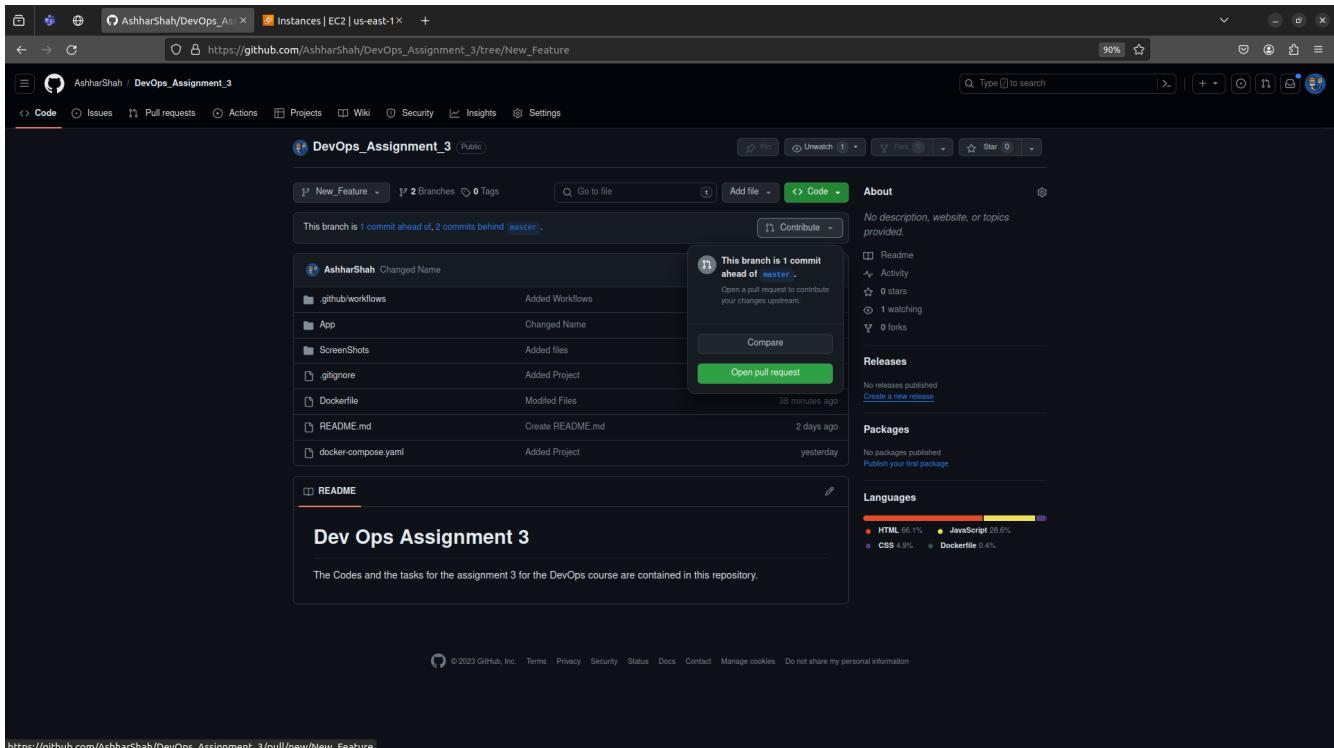
- Next we will create a new branch, in the example I have created the branch “New\_Feature” in which I will modify the underlying code by changing the name of the items from “Yummy” to “Tasty”.
- After this I will push all the changes that I have made to the “New\_Feature” branch on the “origin” server.

```
ashhar@Ashhar-Lenovo-8350:~/Desktop/DevOps_Assignment_3$ git checkout -b 'New_Feature'
fatal: 'New_Feature' is not a valid branch name.
ashhar@Ashhar-Lenovo-8350:~/Desktop/DevOps_Assignment_3$ git checkout -b 'New_Feature'
Switched to a new branch 'New_Feature'
ashhar@Ashhar-Lenovo-8350:~/Desktop/DevOps_Assignment_3$ git status
On branch New_Feature
nothing to commit, working tree clean
ashhar@Ashhar-Lenovo-8350:~/Desktop/DevOps_Assignment_3$ git status
Changes not staged for commit:
  (use "git add <file>..." to record your changes)
    (use "git restore <file>..." to discard changes in working directory)

no changes added to commit (use "git add" and/or "git commit -m")
ashhar@Ashhar-Lenovo-8350:~/Desktop/DevOps_Assignment_3$ git add .
ashhar@Ashhar-Lenovo-8350:~/Desktop/DevOps_Assignment_3$ git commit -m "Changed Item Name"
[New_Feature 95544e6] Changed item name
 1 file changed, 2 insertions(+), 2 deletions(-)
ashhar@Ashhar-Lenovo-8350:~/Desktop/DevOps_Assignment_3$ git push u origin New_Feature
remote: Create a pull request for 'New_Feature' on GitHub by visiting:
remote:   https://github.com/AshharShah/DevOps_Assignment_3/pull/new/New_Feature
remote: Branch 'New_Feature' set up to track remote branch 'New_Feature' from 'origin'.
ashhar@Ashhar-Lenovo-8350:~/Desktop/DevOps_Assignment_3$
```

## 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.

The screenshot shows the GitHub Actions tab for the repository "AshharShah / DevOps\_Assignment\_3". The "All workflows" section is selected. A table lists seven workflow runs:

Run	Event	Status	Branch	Actor
AshharShah initiated the workflow - Deploy To Testing	Workflow To Deploy To Staging Server (EC2) #3: Pull request #3 opened by AshharShah	now	Queued	
AshharShah initiated the workflow - Deploy To Staging	Workflow To Deploy To Staging Server (EC2) #4: Commit ab1f5cc pushed by AshharShah	7 minutes ago	1m 59s	
AshharShah initiated the workflow - Deploy To Testing	Workflow To Deploy To Testing Server (EC2) #2: Pull request #2 opened by AshharShah	7 minutes ago	1m 43s	
AshharShah initiated the workflow - Deploy To Staging	Workflow To Deploy To Staging Server (EC2) #3: Commit 5002eb pushed by AshharShah	12 minutes ago	2m 16s	
AshharShah initiated the workflow - Deploy To Testing	Workflow To Deploy To Testing Server (EC2) #1: Pull request #1 opened by AshharShah	12 minutes ago	2m 2s	
AshharShah initiated the workflow - Deploy To Staging	Workflow To Deploy To Staging Server (EC2) #2: Commit 17e4b95 pushed by AshharShah	15 minutes ago	2m 17s	
AshharShah initiated the workflow - Deploy To Staging	Workflow To Deploy To Staging Server (EC2) #1: Commit 02c202 pushed by AshharShah	23 minutes ago	2m 15s	

[https://github.com/AshharShah/DevOps\\_Assignment\\_3/actions/runs/7230421833](https://github.com/AshharShah/DevOps_Assignment_3/actions/runs/7230421833)

- 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).

The screenshot shows an email from "ashharshah7@gmail.com" titled "GitHub Action Success" and the GitHub Actions log for the "Deploy To Testing" workflow.

Email Content:

Subject: GitHub Action Success  
To: ashharshah7@gmail.com  
From: GitHub Action  
Timestamp: 10:47 AM (1 minute ago)

Workflow: Workflow To Deploy To Testing Server (EC2)  
Status: Success  
Details: Workflow of AshharShah/DevOps\_Assignment\_3 initiated by AshharShah was successfully executed. View the changes on the testing server!

Actions Log:

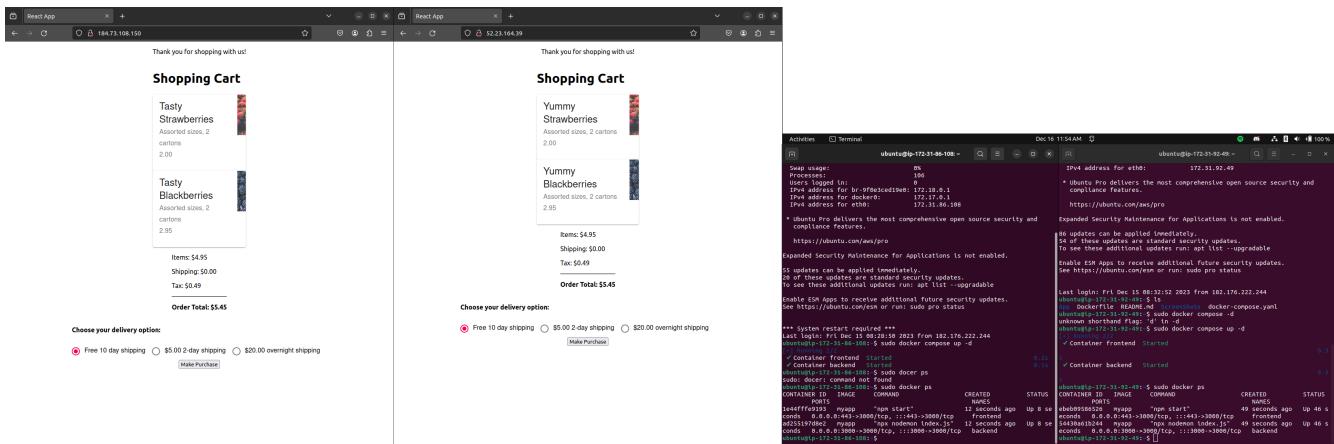
```

actions/checkout@v2
  1  Set up job
  2  Run .github/workflows/deploy.yml
  3  Build appveyor\deploy\Deployment.ps1
  4  Build appveyor\deploy\Deployment.ps1
  5  Checking Out
  6  Setting Up Node.js
  7  Setting Up Node Installation
  8  Install Dependencies
  9  Building Project
  10  Testing The Project
  11  Logging In The Project
  12  Deploy To Changes To Testing Server
  13  Installing Node Modules
  14  Notify on success
  15  Notify on Failure
  16  Print Setting Up Node.js
  17  Print Checking Out
  18  Complete Job
  19  Publishing objects: 7 items
  20  Counting objects: 100K (7/7), done.
  21  Delta compression using up to 8 threads
  22  Compressing objects: 100% (5/5), done.
  23  Writing objects: 100% (5/5), done.
  24  Total 100K (5/5), pack-reused 0
  25  remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
  26  0:00 [00:00] 0:00:00 3B/s
  27  0554e9... ed67fe New Feature -> New_Feature
  28  fatal: unable to find remote ref 'origin/New_Feature'
  29  ashharshah@Lenovo-8150:~/Desktop/DevOps_Assignment_3$ git push u origin New_Feature
  30  Branch 'New_Feature' set up to track 'refs/heads/New_Feature'.
  31  Your branch is up to date with 'origin/New_Feature'.
  32  Changes not staged for commit:
  33  (use "git add <file>" to update what will be committed)
  34  (use "git restore <file>..." to discard changes in working directory)
  35  nothing added
  36  0 changes added to commit (use "git add" and/or "git commit -a")
  37  ashharshah@Lenovo-8150:~/Desktop/DevOps_Assignment_3$ git add .
  38  ashharshah@Lenovo-8150:~/Desktop/DevOps_Assignment_3$ git commit -m "Changed New"
  39  [New_Feature ed67fe] Changed New
  40  1 file changed, 2 insertions(+), 2 deletions(-)
  41  ashharshah@Lenovo-8150:~/Desktop/DevOps_Assignment_3$ git push u origin New_Feature
  42  Branch 'New_Feature' set up to track 'refs/heads/New_Feature'.
  43  Your branch is up to date with 'origin/New_Feature'.
  44  Counting objects: 100K (7/7), done.
  45  Delta compression using up to 8 threads
  46  Compressing objects: 100% (5/5), done.
  47  Writing objects: 100% (5/5), done.
  48  Total 100K (5/5), pack-reused 0
  49  remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
  50  0:00 [00:00] 0:00:00 3B/s
  51  0554e9... ed67fe New Feature -> New_Feature
  52  fatal: unable to find remote ref 'origin/New_Feature'
  53

```

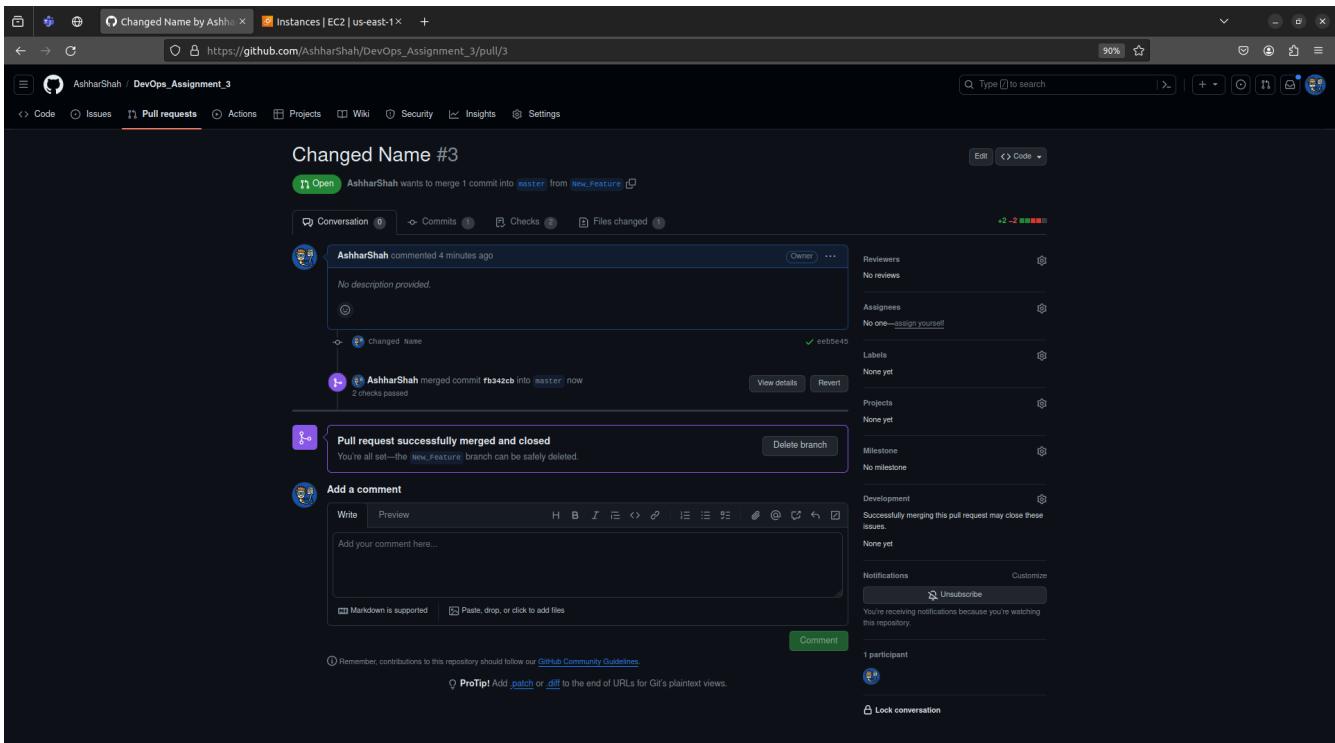
## 4. Verify Testing Changes

- 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.



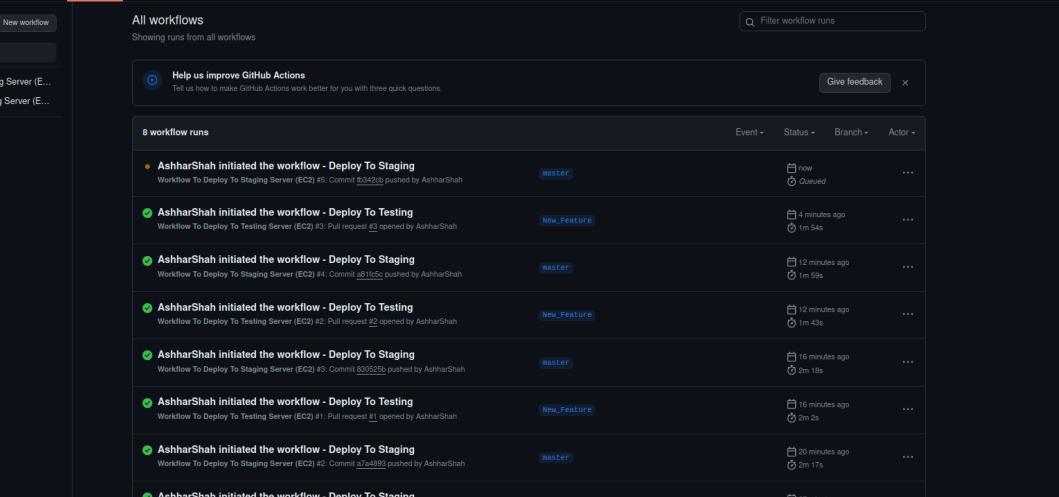
## 5. Merge Changes

- 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 screenshot shows the GitHub Actions interface for a repository named 'DevOps\_Assignment\_3'. The left sidebar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The main content area displays a list of workflow runs under the 'All workflows' tab. A search bar at the top right allows filtering of workflow runs.

**Actions**

All workflows

Showing runs from all workflows

Help us improve GitHub Actions

Tell us how to make GitHub Actions work better for you with three quick questions.

Give feedback

Management

Caches

Runners

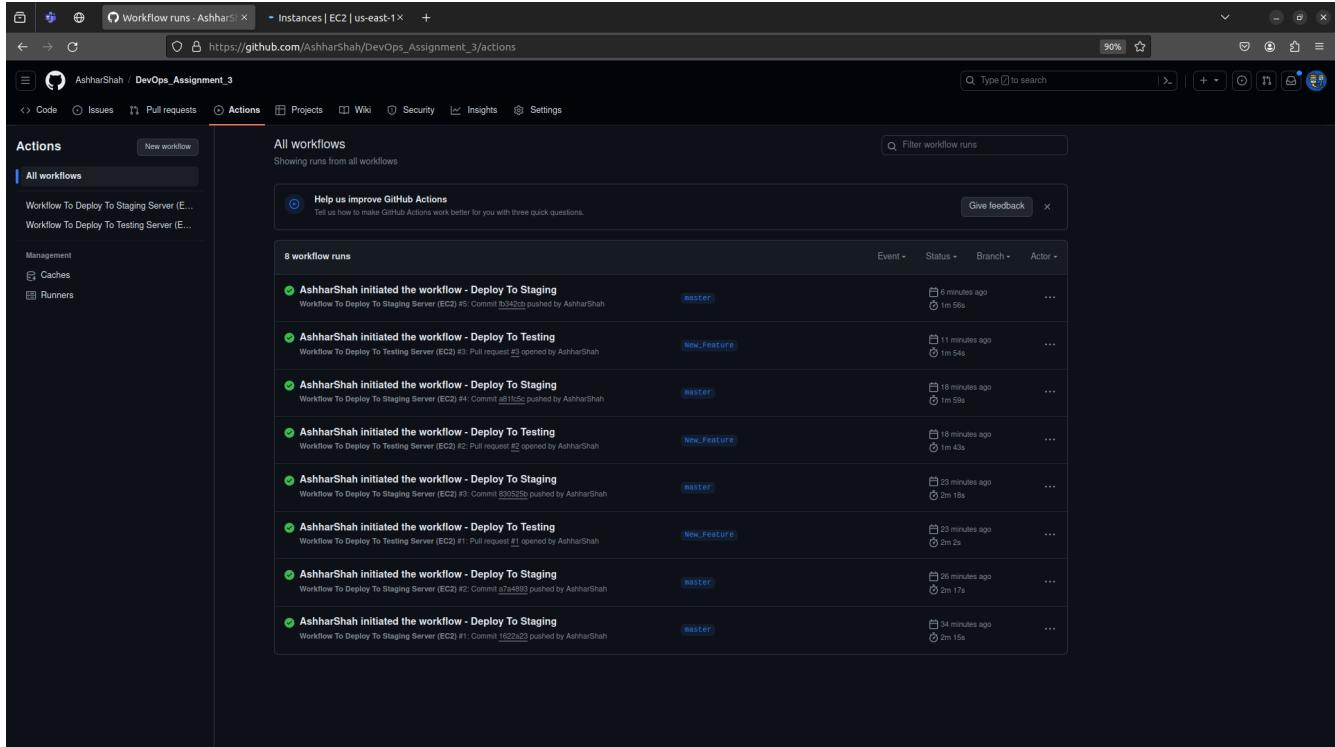
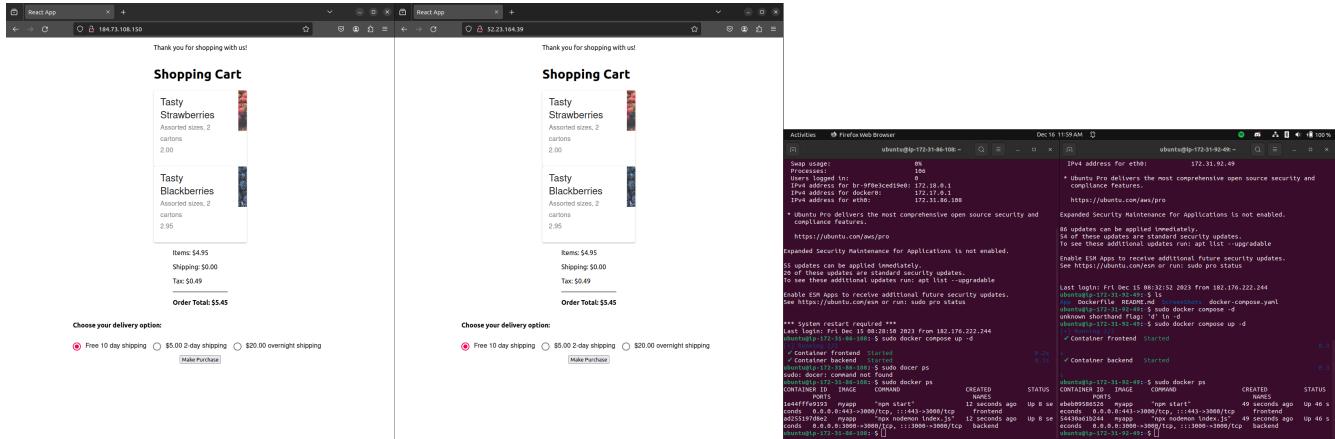
8 workflow runs

Event	Status	Branch	Actor
AshharShah initiated the workflow - Deploy To Staging	now	master	AshharShah
AshharShah initiated the workflow - Deploy To Testing	queued	New_Feature	AshharShah
AshharShah initiated the workflow - Deploy To Staging	4 minutes ago	master	AshharShah
AshharShah initiated the workflow - Deploy To Testing	1m 54s	New_Feature	AshharShah
AshharShah initiated the workflow - Deploy To Staging	12 minutes ago	master	AshharShah
AshharShah initiated the workflow - Deploy To Testing	1m 59s	New_Feature	AshharShah
AshharShah initiated the workflow - Deploy To Staging	12 minutes ago	master	AshharShah
AshharShah initiated the workflow - Deploy To Testing	1m 43s	New_Feature	AshharShah
AshharShah initiated the workflow - Deploy To Staging	16 minutes ago	master	AshharShah
AshharShah initiated the workflow - Deploy To Testing	2m 18s	New_Feature	AshharShah
AshharShah initiated the workflow - Deploy To Staging	16 minutes ago	master	AshharShah
AshharShah initiated the workflow - Deploy To Testing	2m 2s	New_Feature	AshharShah
AshharShah initiated the workflow - Deploy To Staging	20 minutes ago	master	AshharShah
AshharShah initiated the workflow - Deploy To Staging	17s	master	AshharShah
AshharShah initiated the workflow - Deploy To Staging	22 minutes ago	master	AshharShah
AshharShah initiated the workflow - Deploy To Staging	2m 15s	master	AshharShah

- 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.

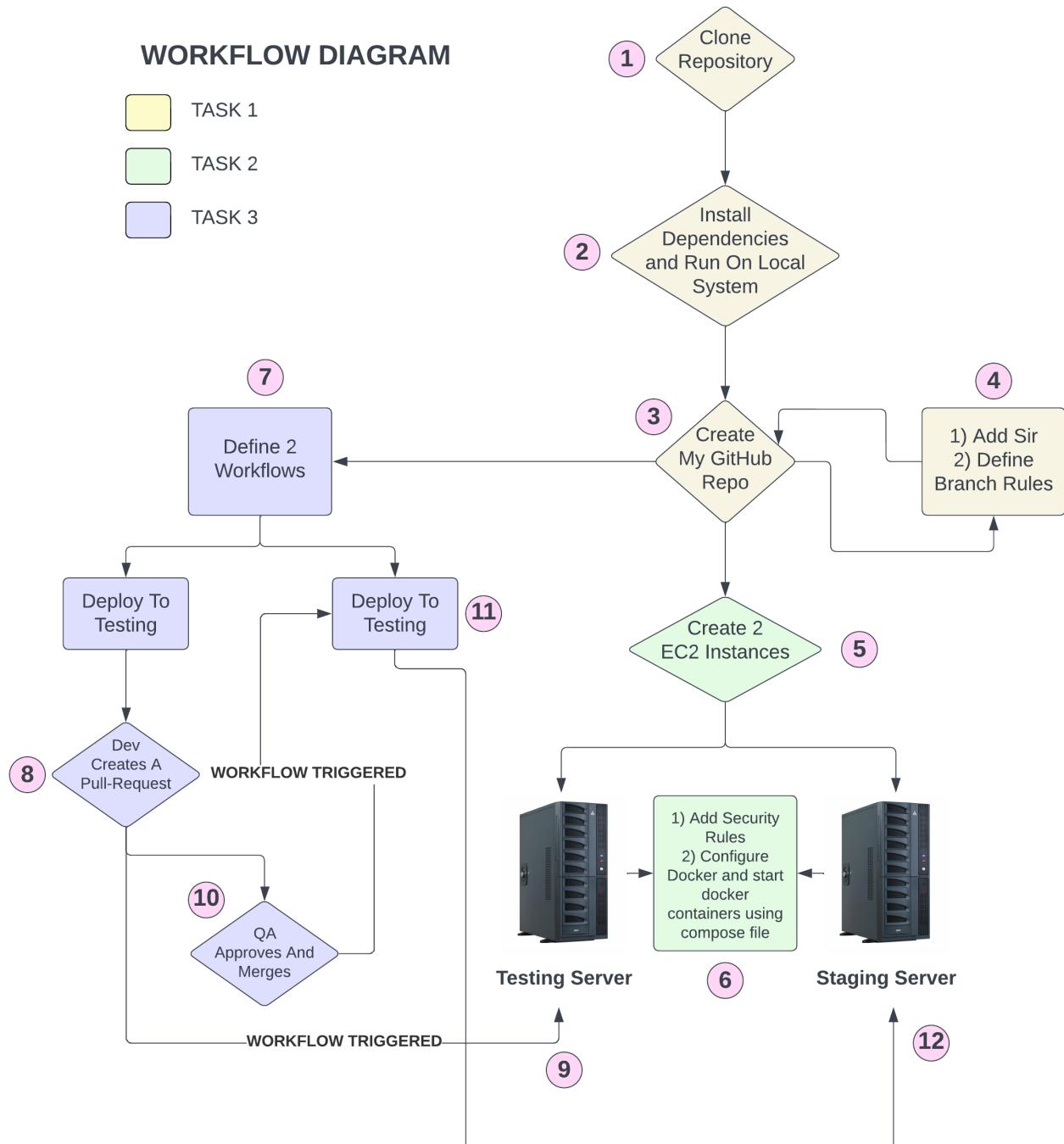


## CHALLENGES FACED

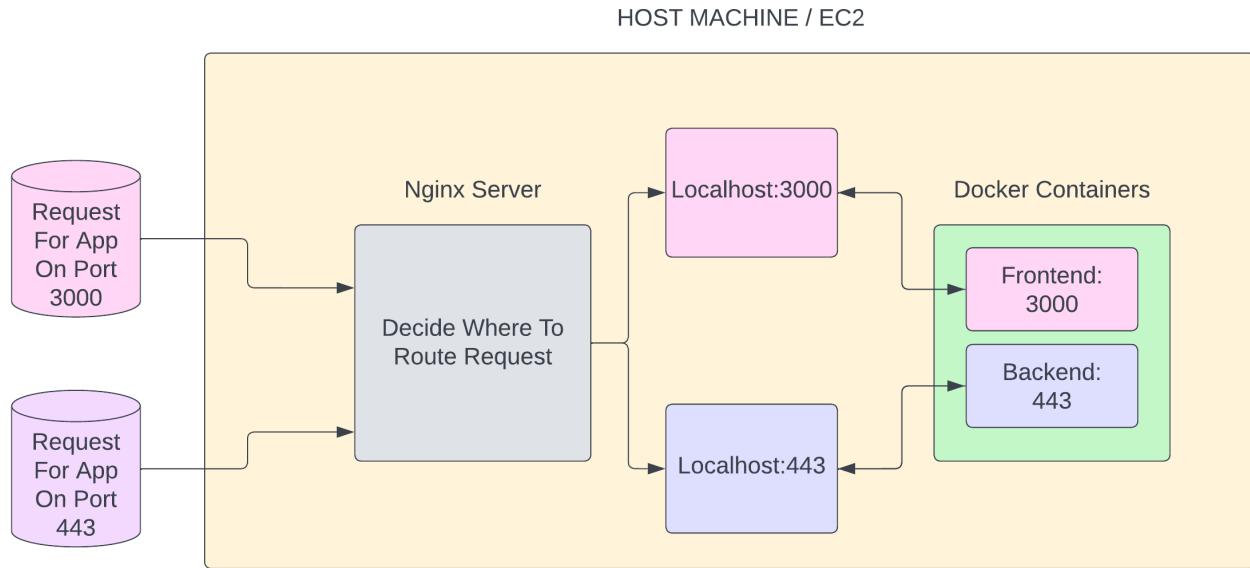
During the project I faced a few challenges which are listed in the points below.

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.