

Lab-5



Angad Singh, Onosen Aziebge, Soumil Thete

Lab-5	1
Screenshot of the testing file	4
#filterByTerm is not defined :	5
#After defining filterByTerm:	5
# Test fails when an upper-case search term is passed as input	6
#After accounting for the upper-case term the test passes	6
A.1 Test for search term “uRL”	7
Result : Failed	7
A.2 . Empty Search Term	8
Result: Failed	8
#After making changes to the code : -	9
1. What is continuous integration?	10
2. What is meant by a pipeline in CI?	10
3. What is continuous delivery?	10
Create a new repository	11
Set up node.js	11
Change to self-hosted because it subs the get of action on the virtual private server and test the node-version (12.x)	12
Start commit and commit to new file	12
Clone repository	13
Cloned successfully	13
Open in visual studio code	14
Setup all the other configurations for example(gitignore, packagelock, express)	15
Commit and push changes	16
Add a runner	16
Follow instructions to set up runner then extract	16

Objective

The main goal of this lab is to get familiar with testing of JavaScripts and Continuous Integration using GitHub. The testing tool used for this lab is JEST. The initial part of the lab covers JavaScript Testing using JEST and the second part covers Continuous Integration in JavaScript

B.1 JEST

1. Screenshot of the testing file

```
describe("Filter function", () => {
  // test stuff
  test("it should filter by a search term (link)", () => {
    const input = [
      { id: 1, url: "https://www.url1.dev" },
      { id: 2, url: "https://www.url2.dev" },
      { id: 3, url: "https://www.link3.dev" }
    ];
    const output = [{ id: 3, url: "https://www.link3.dev" }];
    expect(filterByTerm(input, "link")).toEqual(output);
    expect(filterByTerm(input, "LINK")).toEqual(output);

    // actual test
  });
});

function filterByTerm(inputArr, searchTerm) {
  const regex = new RegExp(searchTerm, "i");
  return inputArr.filter(function (arrayElement) {
    return arrayElement.url.match(regex);
  });
}
```

2. Failed tests:

#filterByTerm is not defined :

```
FAIL __tests__/filterByTerm.spec.js
  Filter function
    ✕ it should filter by a search term (link) (1 ms)

    • Filter function > it should filter by a search term (link)

      ReferenceError: filterByTerm is not defined

       8 |         ];
       9 |         const output = [{ id: 3, url: "https://www.link3.dev" }];
    > 10 |         expect(filterByTerm(input, "link")).toEqual(output);
          |                   ^
      11 |
      12 |         // actual test
      13 |     });

    at Object.<anonymous> (__tests__/filterByTerm.spec.js:10:9)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 1 total
```

#After defining filterByTerm:

```
PASS __tests__/filterByTerm.spec.js
  Filter function
    ✓ it should filter by a search term (link) (2 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.395 s, estimated 1 s
```

Test fails when an upper-case search term is passed as input

```
FAIL __tests__/filterByTerm.spec.js
Filter function
  ✕ it should filter by a search term (link) (10 ms)

  ● Filter function > it should filter by a search term (link)

    expect(received).toEqual(expected) // deep equality

    - Expected   - 6
    + Received   + 1

    - Array [
    -   Object {
    -     "id": 3,
    -     "url": "https://www.link3.dev",
    -   },
    - ]
    + Array []

    15 |         const output = [{ id: 3, url: "https://www.link3.dev" }];
    16 |         expect(filterByTerm(input, "link")).toEqual(output);
    > 17 |         expect(filterByTerm(input, "LINK")).toEqual(output);
        |                                           ^
    18 |
    19 |         // actual test
    20 |     });

at Object.<anonymous> (__tests__/filterByTerm.spec.js:17:45)
```

#After accounting for the upper-case term the test passes

```
PASS __tests__/filterByTerm.spec.js
Filter function
  ✓ it should filter by a search term (link) (3 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.396 s, estimated 1 s
```

3. Exercise

A.1 Test for search term “uRL”

Result : Failed

```
FAIL __tests__/filterByTerm.spec.js
```

```
Filter function
```

```
  × it should filter by a search term (link) (6 ms)
```

```
• Filter function > it should filter by a search term (link)
```

```
  expect(received).toEqual(expected) // deep equality
```

```
    - Expected   - 2
```

```
    + Received   + 6
```

```
Test Suites: 1 failed, 1 total
```

```
Tests:       1 failed, 1 total
```

```
Snapshots:   0 total
```

```
Time:        0.426 s, estimated 1 s
```

A.2 . Empty Search Term

Result: Failed

FAIL __tests__/filterByTerm.spec.js

Filter function

× it should filter by a search term (link) (6 ms)

● Filter function > it should filter by a search term (link)

expect(received).toEqual(expected) // deep equality

```
- Expected   - 6
+ Received   + 1

- Array [
-   Object {
-     "id": 3,
-     "url": "https://www.link3.dev",
-   },
- ]
+ Array []

11 |           expect(filterByTerm(input, "LINK")).toEqual(output);
12 |
> 13 |           expect(filterByTerm(input, " ")).toEqual(output);
    |                                     ^
14 |
15 |           // actual test
16 |         });

at Object.<anonymous> (__tests__/filterByTerm.spec.js:13:42)
```

```
Test Suites: 1 failed, 1 total
Tests:       1 failed, 1 total
Snapshots:   0 total
Time:        0.433 s, estimated 1 s
```


#After making changes to the code :-

```
function filterByTerm(inputArr, searchTerm) {  
  if (!inputArr.length) {  
    throw Error("Input cannot be empty");  
  }  
  
  const regex = new RegExp(searchTerm, "i");  
  return inputArr.filter(function (arrayElement) {  
    return arrayElement.url.match(regex);  
  });  
}
```

Result After Making Changes: PASS

```
> lab_5@1.0.0 test  
> jest  
  
PASS __tests__/filterByTerm.spec.js  
  Filter function  
    ✓ it should filter by a search term (link) (2 ms)  
  
Test Suites: 1 passed, 1 total  
Tests:       1 passed, 1 total  
Snapshots:   0 total  
Time:        0.355 s, estimated 1 s  
Ran all test suites.
```

PART C

1. What is continuous integration?

Continuous integration is a form of development methodology involving daily developer integrations, it is usually done to detect bugs quickly and fix them. The goal is to minimize lead time; the time elapsed between development writing one new line of code and this new code being used by live users in production. It involves automating build and testing every single time builder commits code to the version control

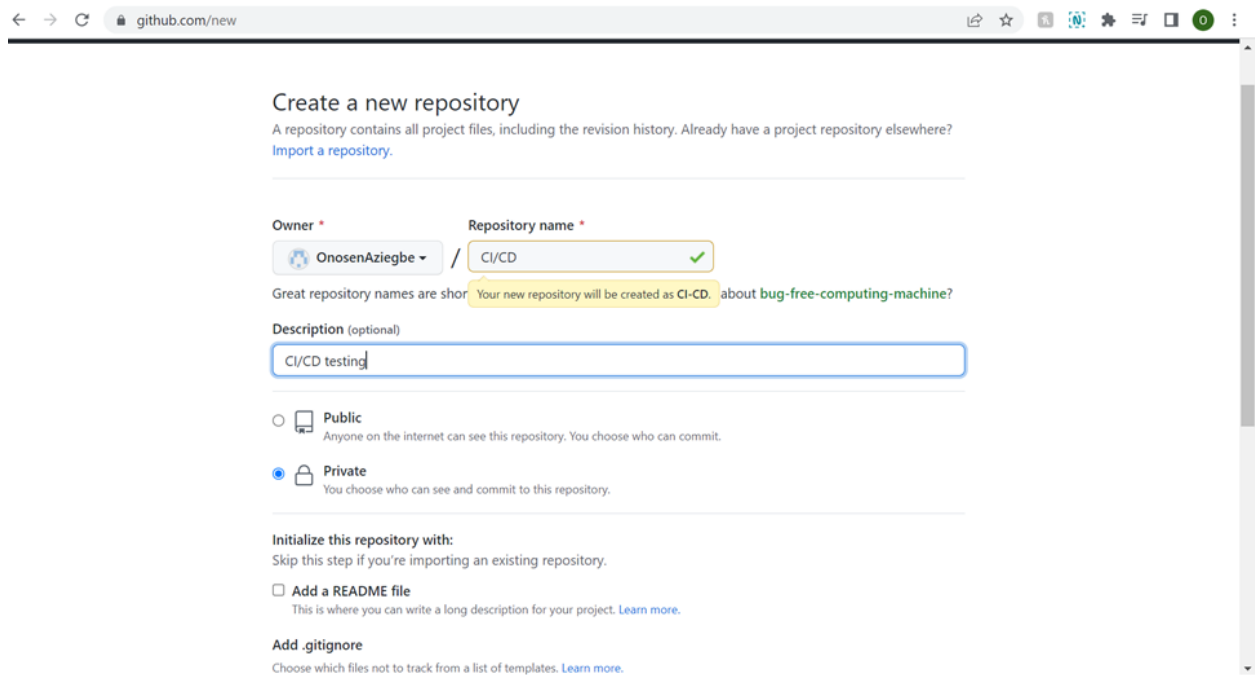
2. What is meant by a pipeline in CI?

It is an implementation of the continuous paradigm, where automated builds, tests, and deployments are orchestrated as one release workflow. The different steps are committing, acceptance, user acceptance testing, capacity, production.

3. What is continuous delivery?

Continuous delivery is done after continuous integration; it involves automating the delivery of applications to selected infrastructure environments. It is fast, automated feedback on the production readiness of your application every time there are code changes, infrastructure or configuration. The benefits of implementing CI/CD include, faster releases, smaller backlog, customer satisfaction and automated processes

Create a new repository



The screenshot shows the GitHub 'Create a new repository' page. The browser address bar shows 'github.com/new'. The page title is 'Create a new repository'. Below the title, it says 'A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)'

The 'Owner' dropdown is set to 'OnosenAziegbe'. The 'Repository name' dropdown is set to 'CI/CD' with a green checkmark. Below this, a message says 'Great repository names are short' followed by a yellow box containing 'Your new repository will be created as CI-CD.' and then 'about bug-free-computing-machine?'.

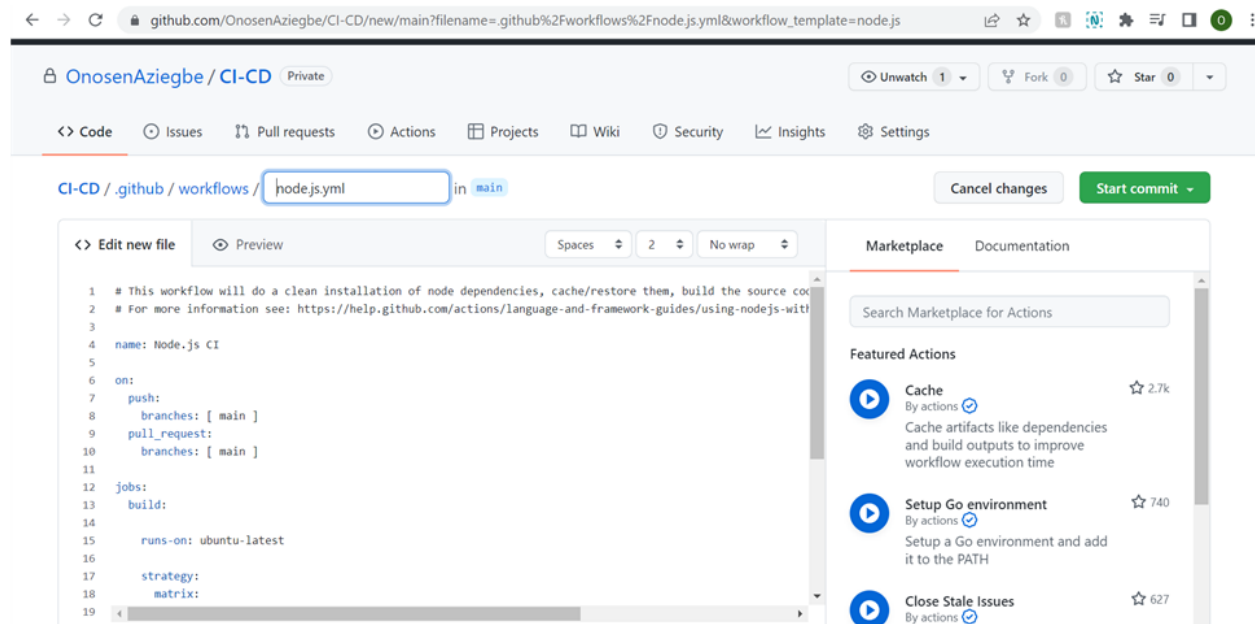
The 'Description (optional)' text area contains 'CI/CD testing'.

There are two radio buttons for visibility: 'Public' (unselected) and 'Private' (selected). The 'Private' option has a subtext: 'You choose who can see and commit to this repository.'

Under 'Initialize this repository with:', it says 'Skip this step if you're importing an existing repository.' There is an unchecked checkbox for 'Add a README file' with subtext 'This is where you can write a long description for your project. [Learn more.](#)'

At the bottom, there is an 'Add .gitignore' section with the text 'Choose which files not to track from a list of templates. [Learn more.](#)'

Set up node.js



The screenshot shows the GitHub repository 'OnosenAziegbe / CI-CD' (Private). The breadcrumb navigation shows 'CI-CD / .github / workflows / node.js.yml' in 'main'. There are buttons for 'Cancel changes' and 'Start commit'.

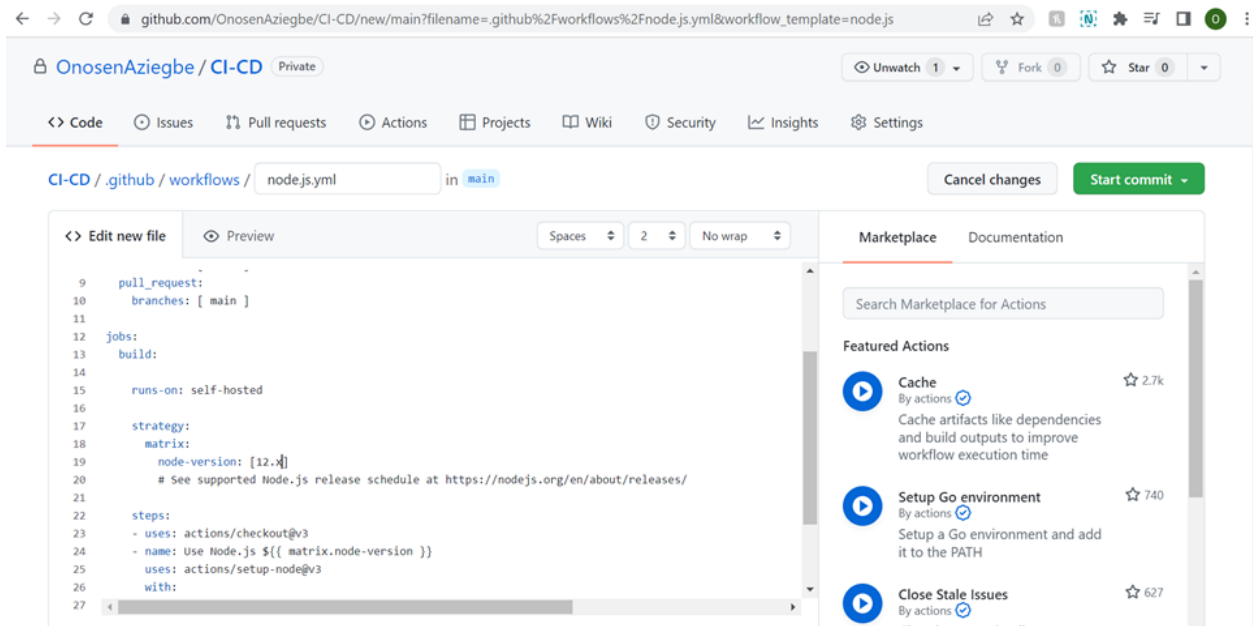
The 'Edit new file' tab is active, showing the content of 'node.js.yml'. The file content is as follows:

```
1 # This workflow will do a clean installation of node dependencies, cache/restore them, build the source code and run tests across multiple versions of node
2 # For more information see: https://help.github.com/actions/language-and-framework-guides/using-nodejs-with-github-actions
3
4 name: Node.js CI
5
6 on:
7   push:
8     branches: [ main ]
9   pull_request:
10    branches: [ main ]
11
12 jobs:
13   build:
14
15     runs-on: ubuntu-latest
16
17     strategy:
18       matrix:
19
```

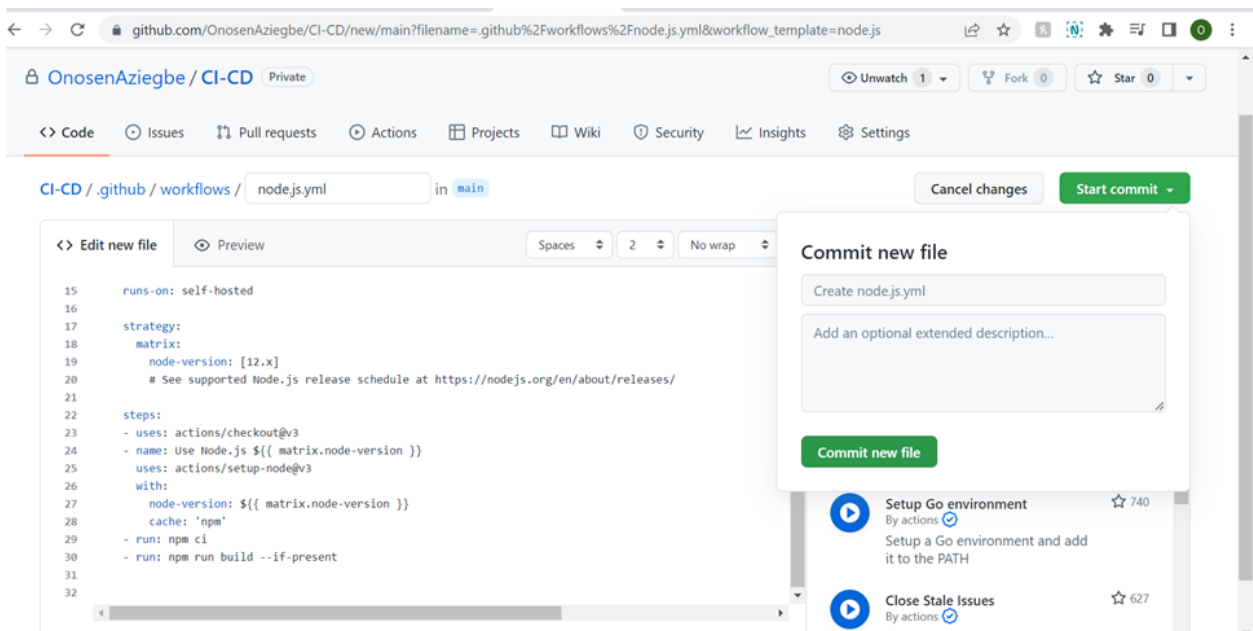
On the right side, the 'Marketplace' tab is active, showing 'Search Marketplace for Actions'. Under 'Featured Actions', there are three actions listed:

- Cache** (By actions, 2.7k stars): Cache artifacts like dependencies and build outputs to improve workflow execution time.
- Setup Go environment** (By actions, 740 stars): Setup a Go environment and add it to the PATH.
- Close Stale Issues** (By actions, 627 stars):

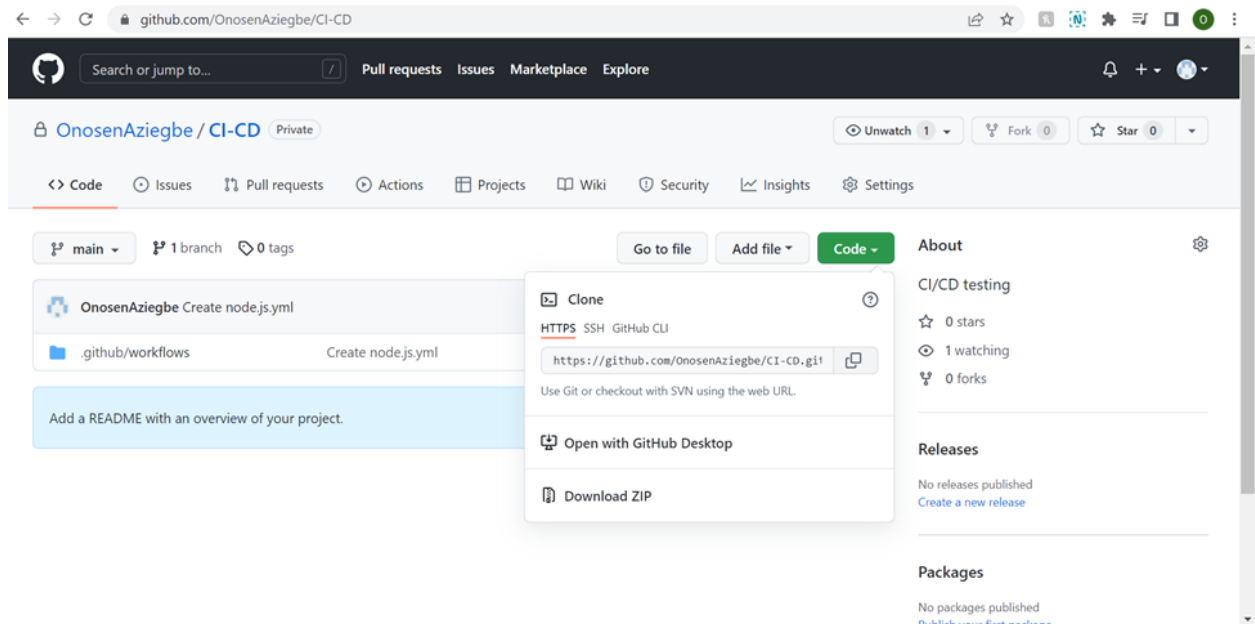
Change to self-hosted because it subs the get of action on the virtual private server and test the node-version (12.x)



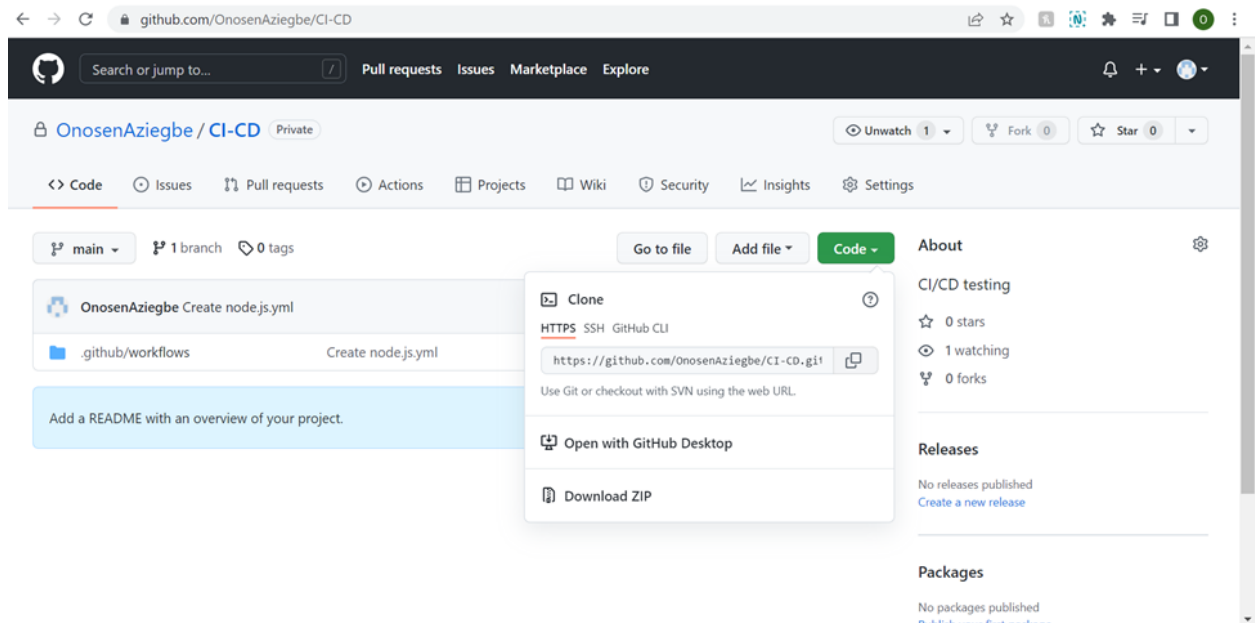
Start commit and commit to new file

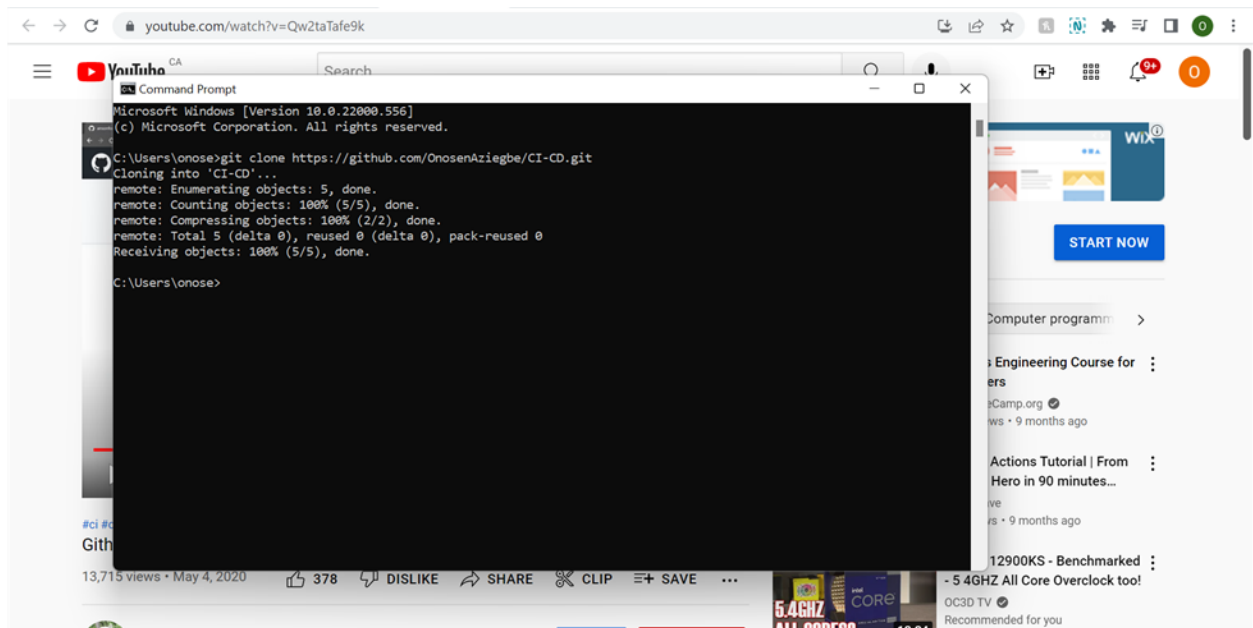


Clone repository

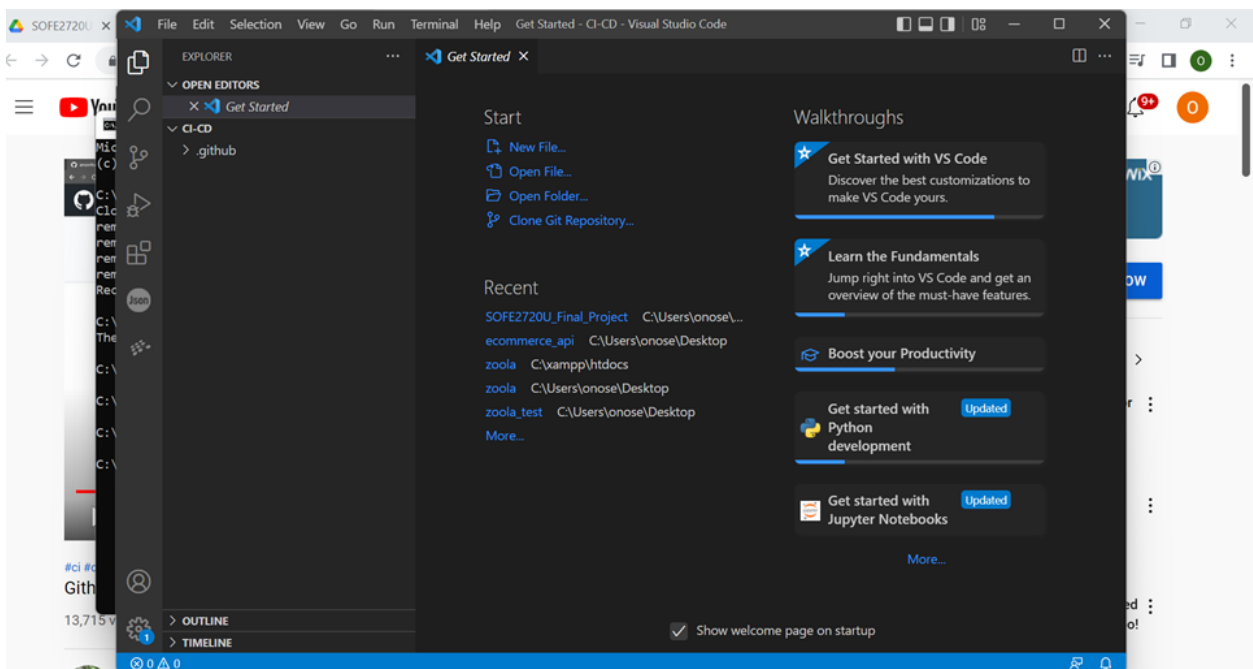


Cloned successfully

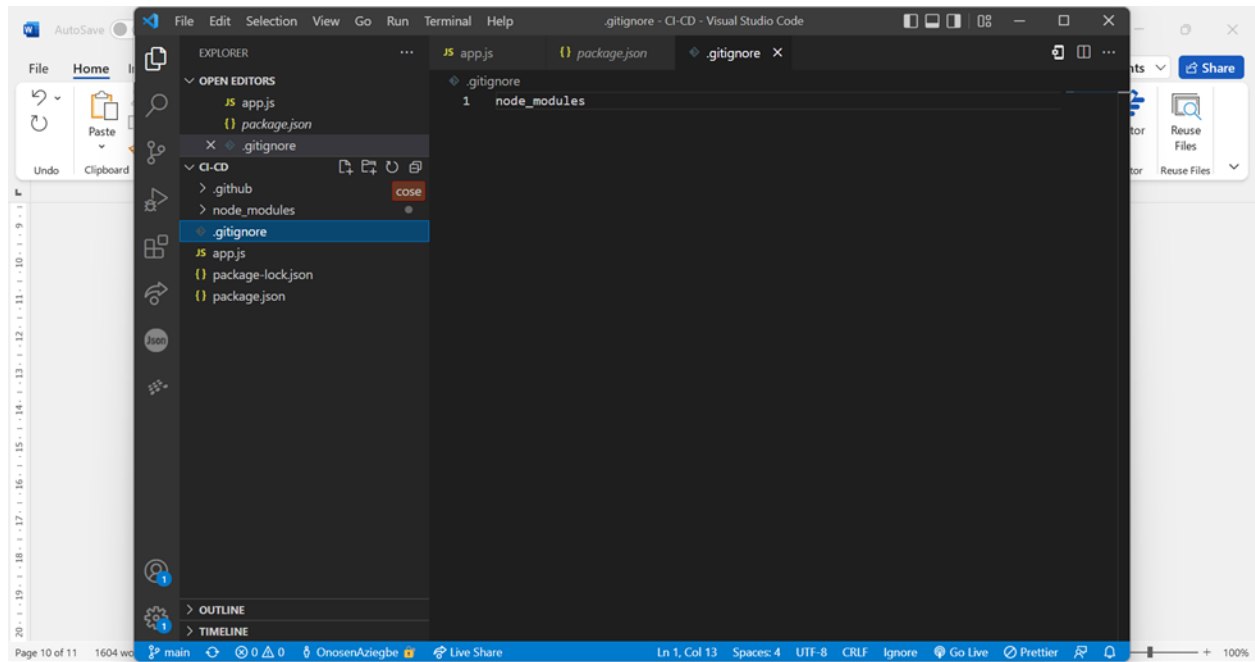




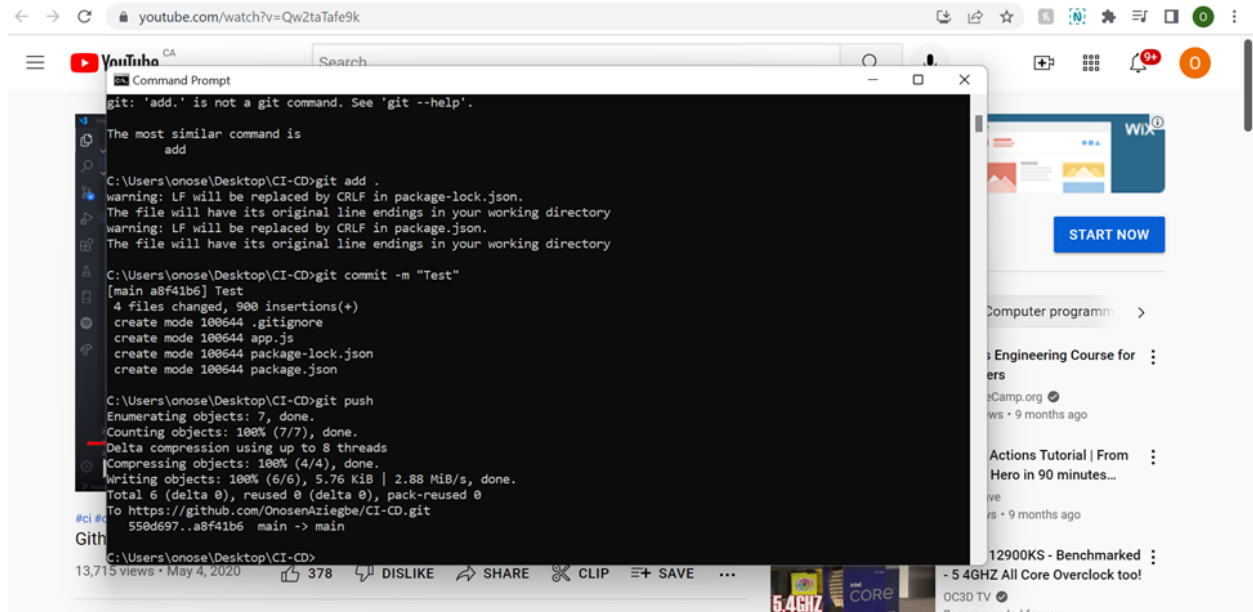
Open in visual studio code



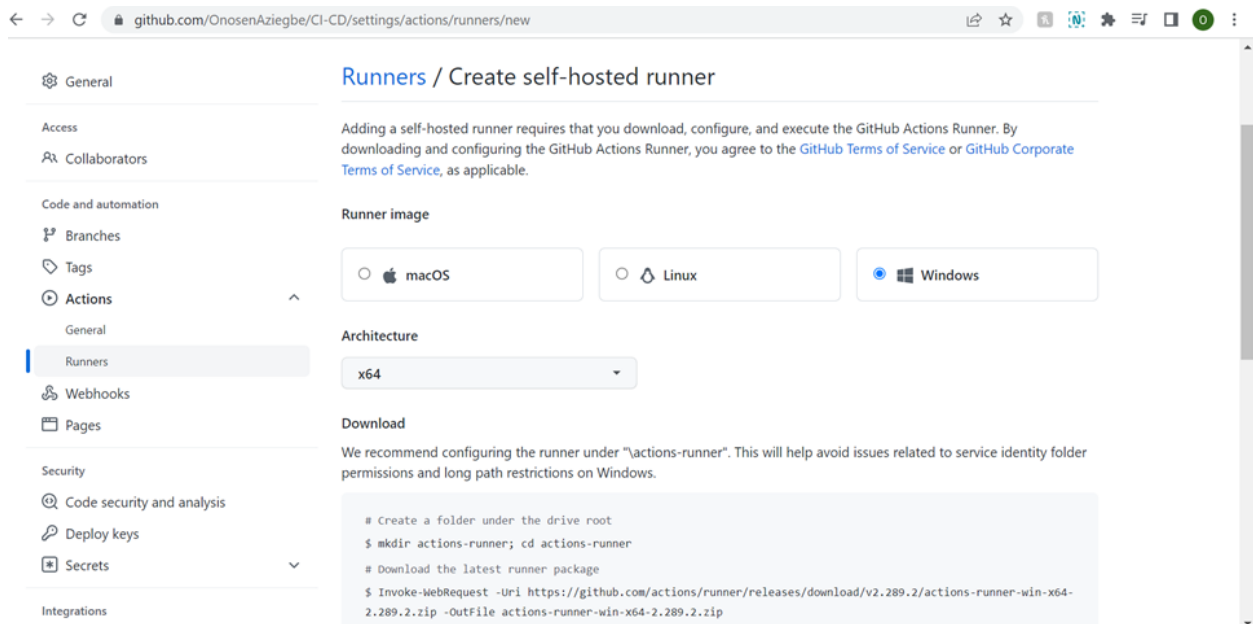
Setup all the other configurations for example(gitignore, packagelock, express)



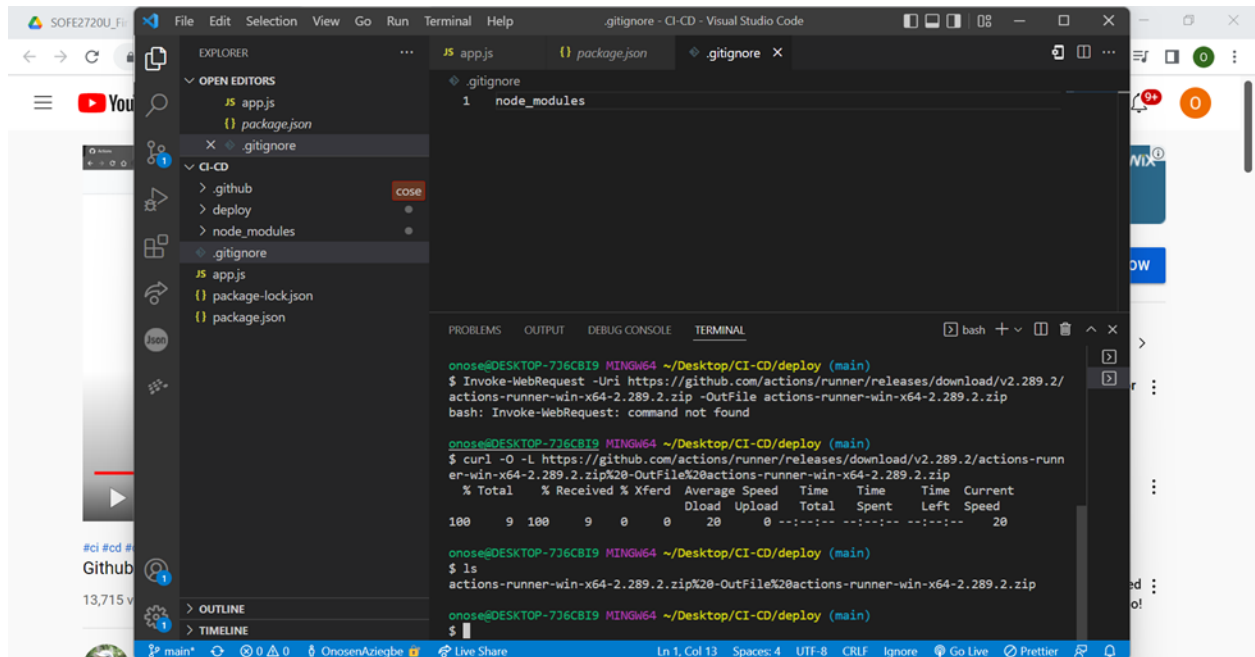
Commit and push changes



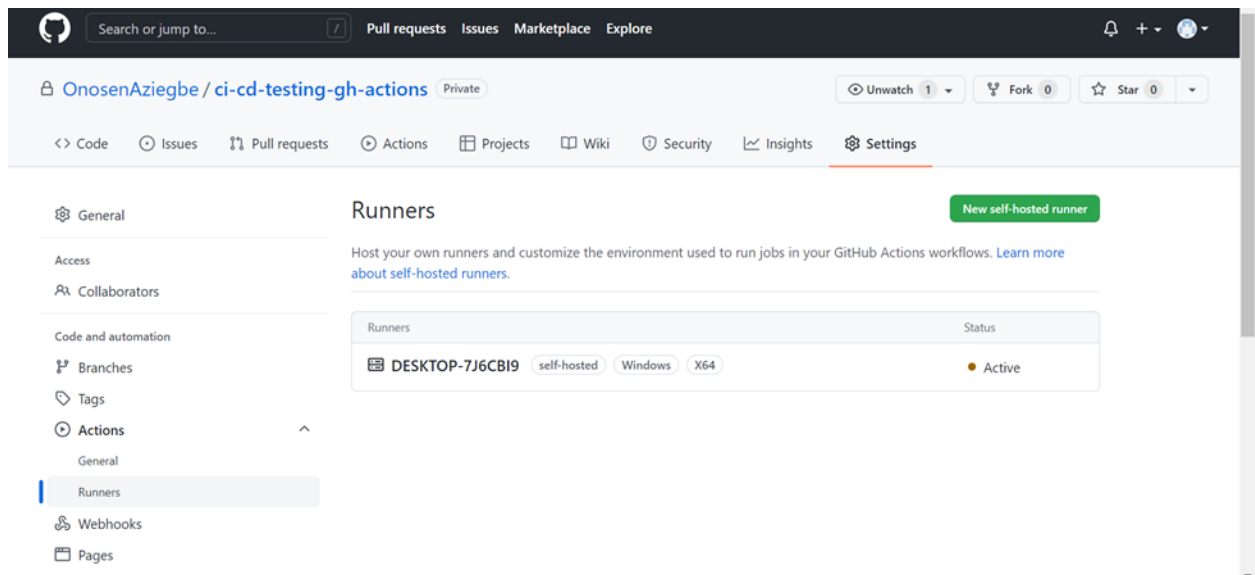
Add a runner



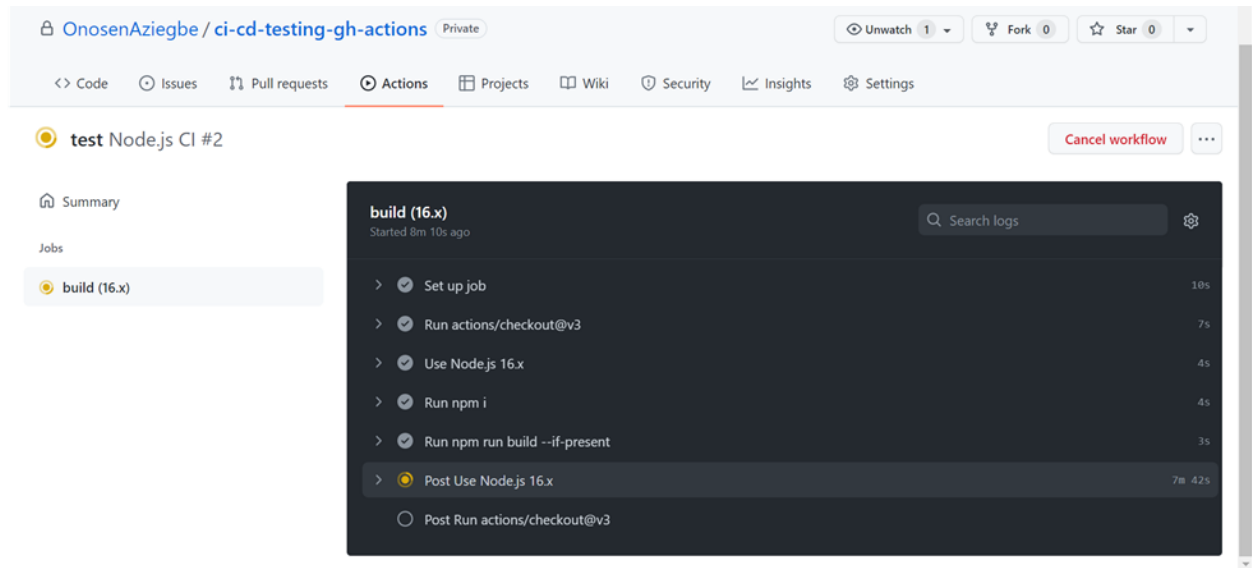
Follow instructions to set up runner then extract



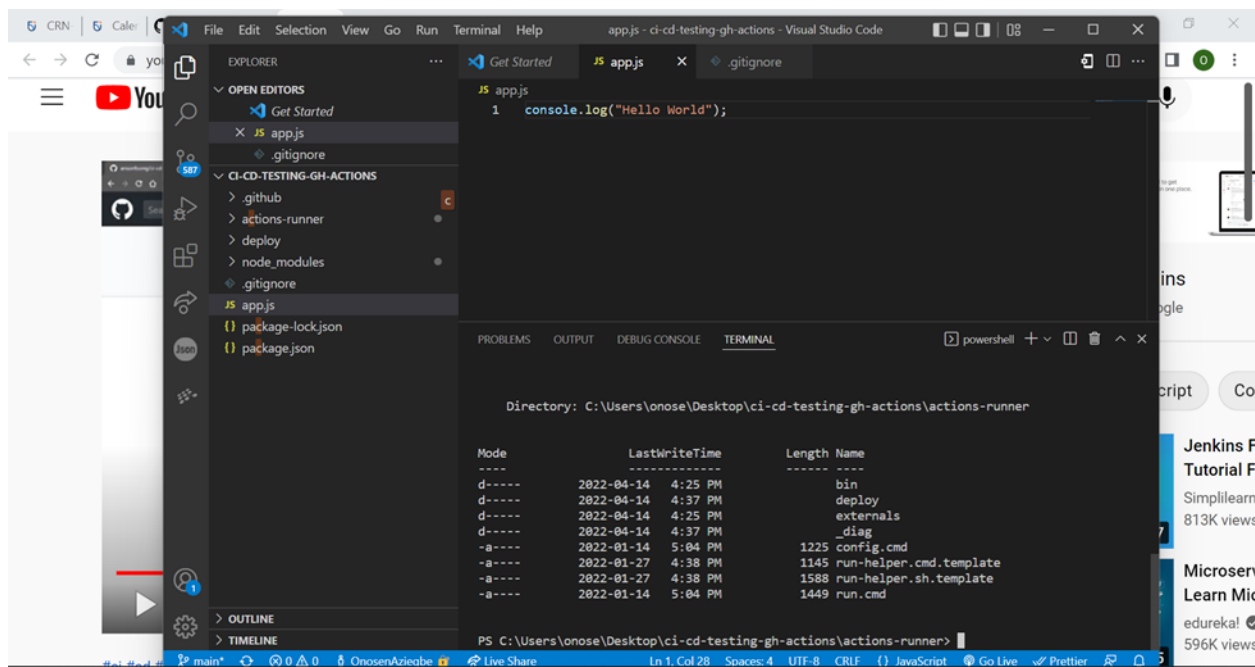
Initiate self runner following the self runner steps



Conduct successful test



Deploy folder now shown in ls



id	name	namespace	version	mode	pid	uptime	status	cpu	mem	user
0	DeployApp	default	1.0.0	fork	3807	1s	online	0%	38.3mb	anson

(PM2)[WARN] Current process list running is not in sync with saved list. DiscordBot Lavalink MusicBot ExpressApp differs. Type 'pm2 save' to synchronize.
anson@ubuntu-s-lvcpu-1gb-nyc1-01:~/deploy/deploy/ci-cd-testing-gh-actions/ci-cd-testing-gh-actions\$

Run App.js and commit changes

The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left displays the file structure of the project 'ci-cd-testing-gh-actions'. The file 'app.js' is selected and open in the editor. The code in 'app.js' is as follows:

```

1  const express = require('express');
2  const app = express();
3
4  app.get('/users/test', (req,res)=>{
5    res.send({msg: "Testing"});
6  });
7
8  app.listen(5000, () => console.log('Running on port 5000'));
9

```

The status bar at the bottom indicates the file is 'Ln 9, Col 1', uses 'UTF-8' encoding, and has 'CRLF' line endings. It also shows the file is a JavaScript file and has Prettier formatting enabled.

Should be able to see hello world port after running

Hello World

Commit all other changes