



AWS Project : CI/CD on EC2 using github actions

Documentation

 By : Abouchiba Mohamed Yassir

To see the files used in this project go to this link:
https://github.com/M-Yassir/AWS-Projects/upload/main/CICD_on_EC2

Introduction

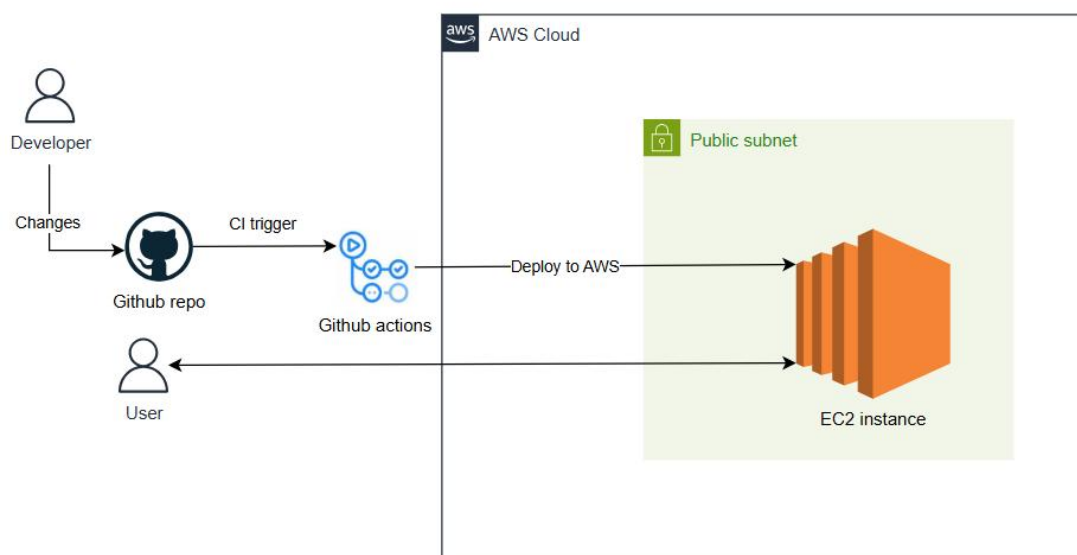
Project Description

This project demonstrates the implementation of a CI/CD pipeline for deploying applications on an Amazon EC2 instance using GitHub Actions. The pipeline automates building, testing, and deploying code changes, ensuring rapid and reliable software delivery. By integrating GitHub Actions with AWS EC2, it highlights how DevOps practices can streamline workflows, improve deployment efficiency, and maintain high-quality standards.

Key Features

- Automated Builds and Testing: GitHub Actions triggers builds and tests on code pushes, ensuring reliability.
- Seamless Deployment: Automatically deploys to EC2 after successful builds, reducing manual errors.
- Version Control Integration: Manages code changes and deployments directly from GitHub.
- Cost-Effective Solution: Utilizes free or low-cost tools, ideal for small to medium projects.

AWS architecture diagram



Step-by-Step implementation

NB: - All the necessary code is on the repository given above.
- The creation of the index.html file isn't covered here as it isn't our topic.

Step1: Creating a EC2 instance:

First, we go to the EC2 service for creating an instance and we click on 'launch instance'

The screenshot shows the AWS Management Console for the EC2 service in the US East (N. Virginia) region. The left sidebar contains navigation links for Dashboard, Instances, Images, Elastic Block Store, and Network & Security. The main content area is divided into several sections:

- Resources:** A table showing the number of EC2 resources in the region. The 'Launch instance' button is highlighted with a red box.
- Launch instance:** A section with a 'Launch instance' button (highlighted with a red box) and a 'Migrate a server' button.
- Service health:** A section showing the status of the EC2 service, which is 'operating normally'.
- EC2 Free Tier:** A section showing the free tier offers for EC2, including a forecast for the end of the month.
- Offer usage (monthly):** A section showing the usage of the free tier, with a progress bar indicating 0% usage.
- Account attributes:** A section showing account details, including the default VPC and settings.

Then, we fill in the necessary informations like it's shown below

Launch an instance

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.

Name and tags Info

Name: MyServer [Add additional tags](#)

Application and OS Images (Amazon Machine Image) Info

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Search our full catalog including 1000s of application and OS images

Recents **Quick Start**

Amazon Linux macOS Ubuntu Windows Red Hat SUSE Linux Debian

Amazon Machine Image (AMI)

Ubuntu Server 24.04 LTS (HVM), SSD Volume Type Free tier eligible

Description

Ubuntu Server 24.04 LTS (HVM), EBS General Purpose (SSD) Volume Type. Support available from Canonical (<http://www.ubuntu.com/cloud/services>).

Canonical, Ubuntu, 24.04, amd64 noble image

Architecture: 64-bit (x86) AMI ID: ami-04b4f1a9cf54c11d0 Username: ubuntu Verified provider

Summary

Number of instances: 1 Info

Software Image (AMI)

Canonical, Ubuntu, 24.04, amd64...[read more](#)

Virtual server type (instance type)

t2.micro

Firewall (security group)

New security group

Storage (volumes)

1 volume(s) - 8 GiB

Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 750 hours of public IPv4 address usage per month, 30 GiB of EBS storage, 2 million I/Os, 1 GB of snapshots, and 100 GB of bandwidth to the internet.

[Cancel](#) [Launch instance](#) [Preview code](#)

After scrolling down a little bit, an option for creating a new key pair will appear, we click on it:

▼ Instance type [Info](#) | [Get advice](#)

Instance type

t2.micro

Family: t2 1 vCPU 1 GiB Memory Current generation: true

On-Demand Windows base pricing: 0.0162 USD per Hour On-Demand Ubuntu Pro base pricing: 0.0134 USD per Hour

On-Demand SUSE base pricing: 0.0116 USD per Hour On-Demand RHEL base pricing: 0.026 USD per Hour

On-Demand Linux base pricing: 0.0116 USD per Hour

Free tier eligible

☐ All generations

[Compare instance types](#)

Additional costs apply for AMIs with pre-installed software

▼ Key pair (login) [Info](#)

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - required

Proceed without a key pair (Not recommended)

Default value ▼

[Create new key pair](#)

▼ Network settings [Info](#)

Network [Info](#)

vpc-0bb0d666ceb0a46d6

Subnet [Info](#)

No preference (Default subnet in any availability zone)

Auto-assign public IP [Info](#)

Enable

[Edit](#)

And we fill in the following informations:

Create key pair

Key pair name

Key pairs allow you to connect to your instance securely.

Server-key

The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.

Key pair type

☒ RSA

RSA encrypted private and public key pair

☐ ED25519

ED25519 encrypted private and public key pair

Private key file format

☒ .pem

For use with OpenSSH

☐ .ppk

For use with PuTTY

⚠ When prompted, store the private key in a secure and accessible location on your computer. You will need it later to connect to your instance. [Learn more](#)

Cancel

Create key pair

After clicking on 'create key pair', a PEM file should be downloaded, we'll need it later.

The final part should look like this:

The screenshot displays the AWS Management Console's 'Create new EC2 instance' wizard. It is divided into several sections:

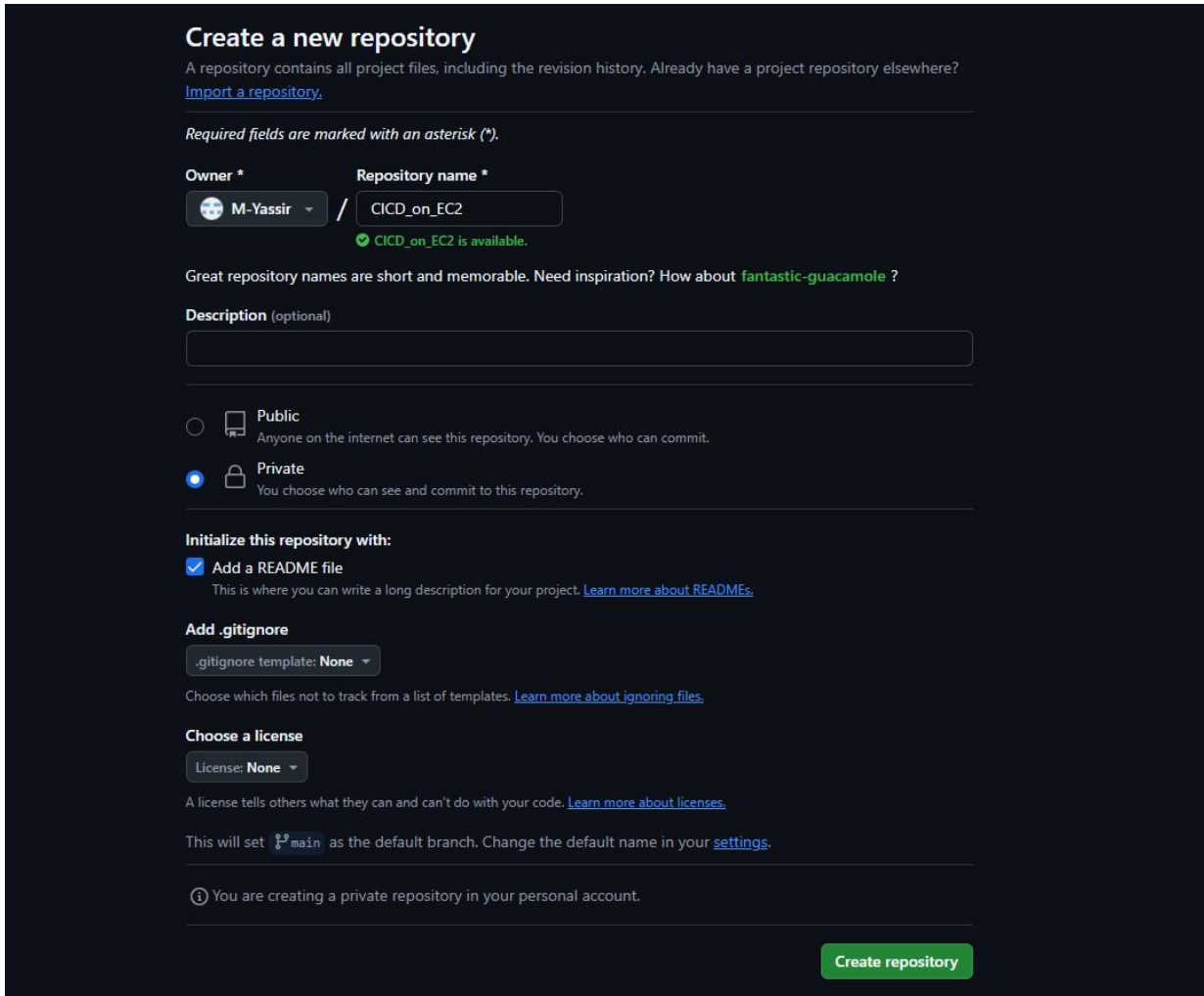
- Key pair (login):** A dropdown menu shows 'Server-key' as the selected key pair. A 'Create new key pair' link is available.
- Network settings:** The 'Network' is set to 'vpc-0bb0d666ceb0a46d6'. The 'Subnet' is set to 'No preference'. 'Auto-assign public IP' is enabled. Under 'Firewall (security groups)', the 'Create security group' option is selected. Three rules are configured: 'Allow SSH traffic from Anywhere', 'Allow HTTPS traffic from the internet', and 'Allow HTTP traffic from the internet'. A warning message states: 'Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.'
- Configure storage:** The 'Root volume' is set to '3000 IOPS (Not encrypted)' with a 'gp3' volume type. A message indicates: 'Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage'. There is an 'Add new volume' button and a 'Click refresh to view backup information' link.
- Summary:** Shows 'Number of instances' as 1. The 'Software Image (AMI)' is 'Canonical, Ubuntu, 24.04, amd64...'. The 'Virtual server type (instance type)' is 't2.micro'. The 'Firewall (security group)' is 'New security group'. The 'Storage (volumes)' is '1 volume(s) - 8 GiB'. A 'Free tier' notice is present. At the bottom, there are 'Cancel', 'Launch instance' (highlighted with a red border), and 'Preview code' buttons.

We click on 'Launch instance'.

A message should appear indicating that the operation was successful.

Step2: Create a github repository and integrate the CICD pipeline:

We create a new repository on github named 'CICD_on_EC2'



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).

Owner * M-Yassir / **Repository name *** CICD_on_EC2
✔ CICD_on_EC2 is available.

Great repository names are short and memorable. Need inspiration? How about [fantastic-guacamole](#) ?

Description (optional)

☐ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☒ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

☒ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore
.gitignore template: **None**

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license
License: **None**

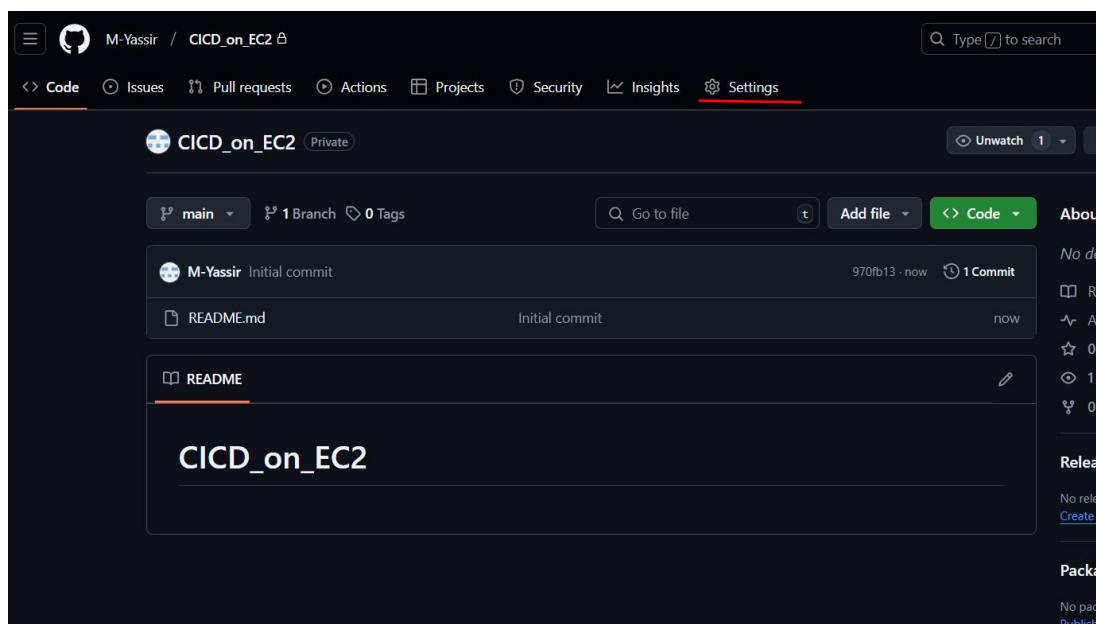
A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set `main` as the default branch. Change the default name in your [settings](#).

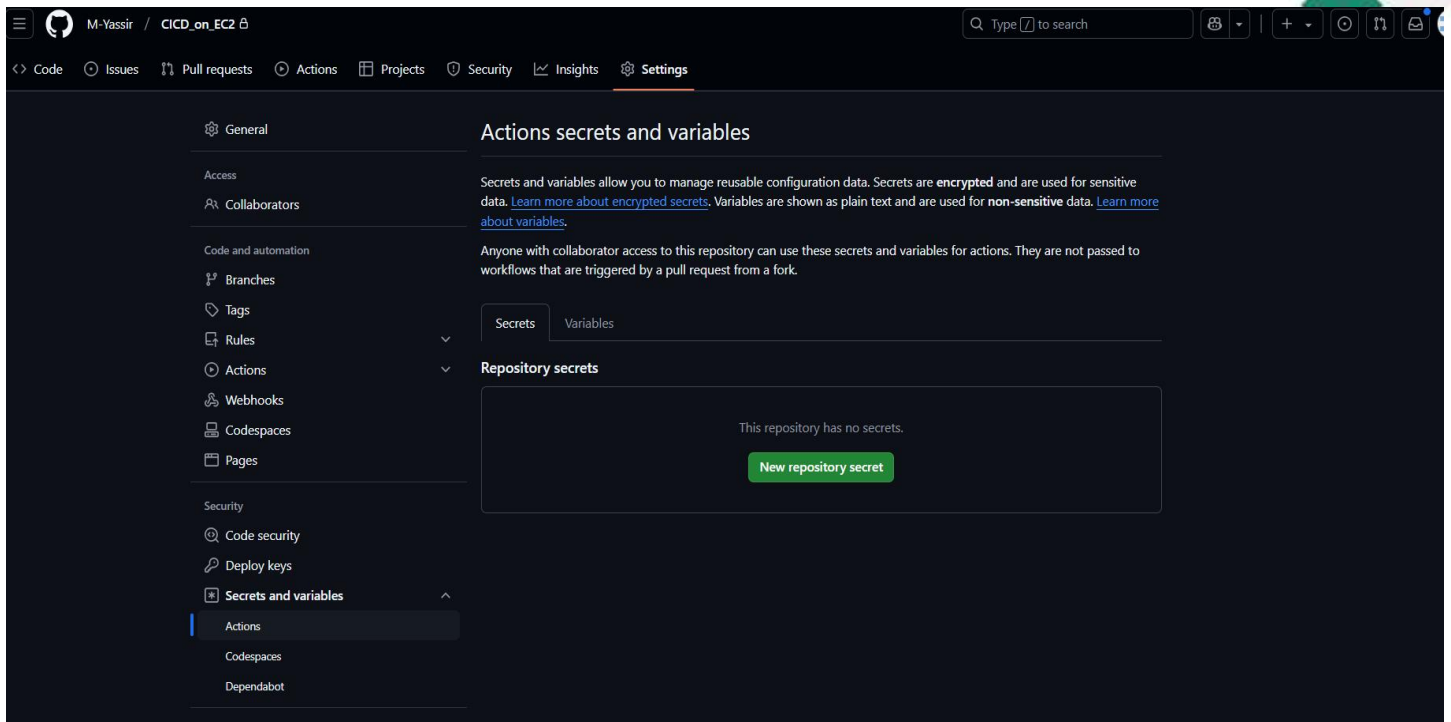
ⓘ You are creating a private repository in your personal account.

Create repository

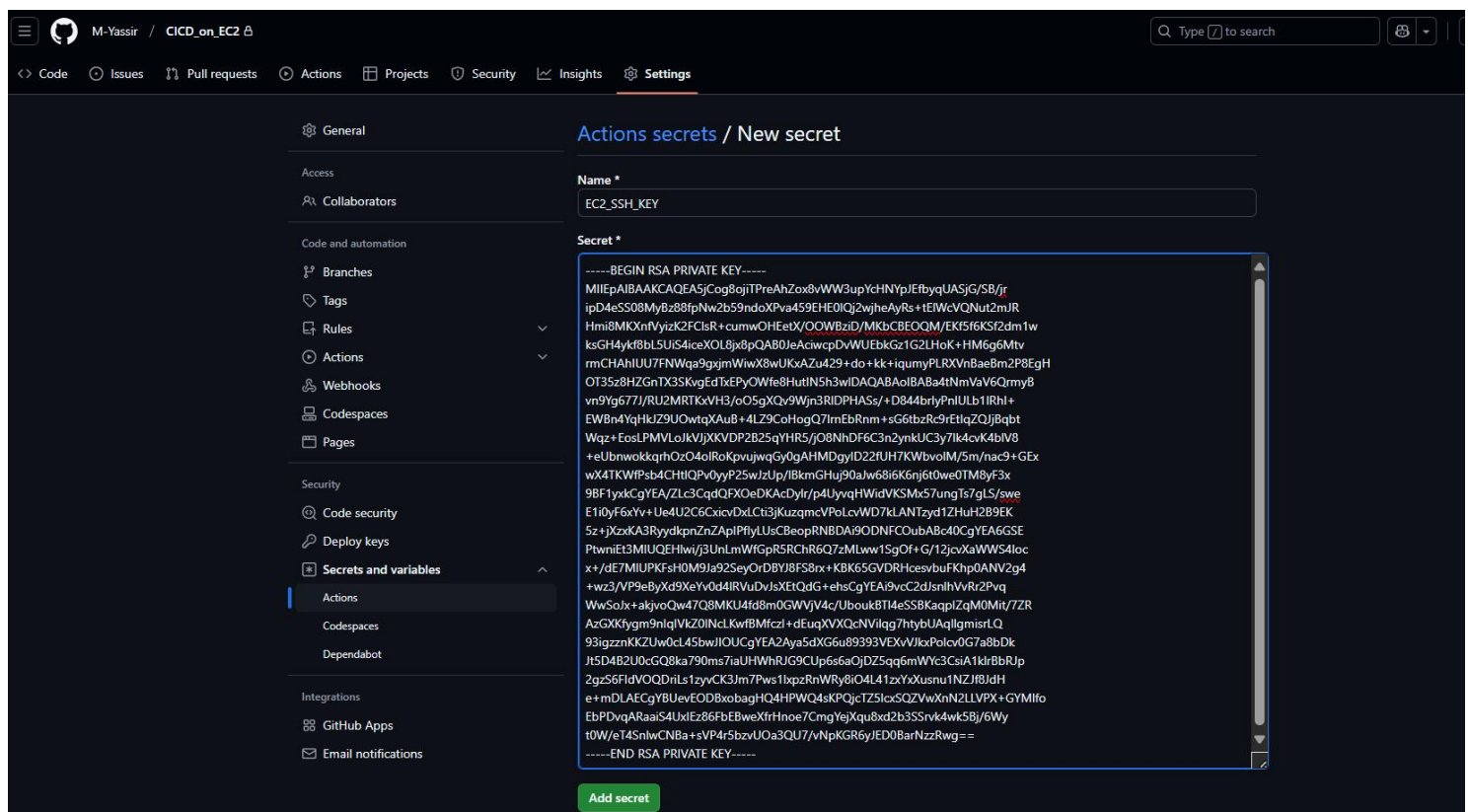
After clicking on 'create repository' we should get the following



We go to the settings tab on **Actions** in **Secrets and variables**, like it's shown below:

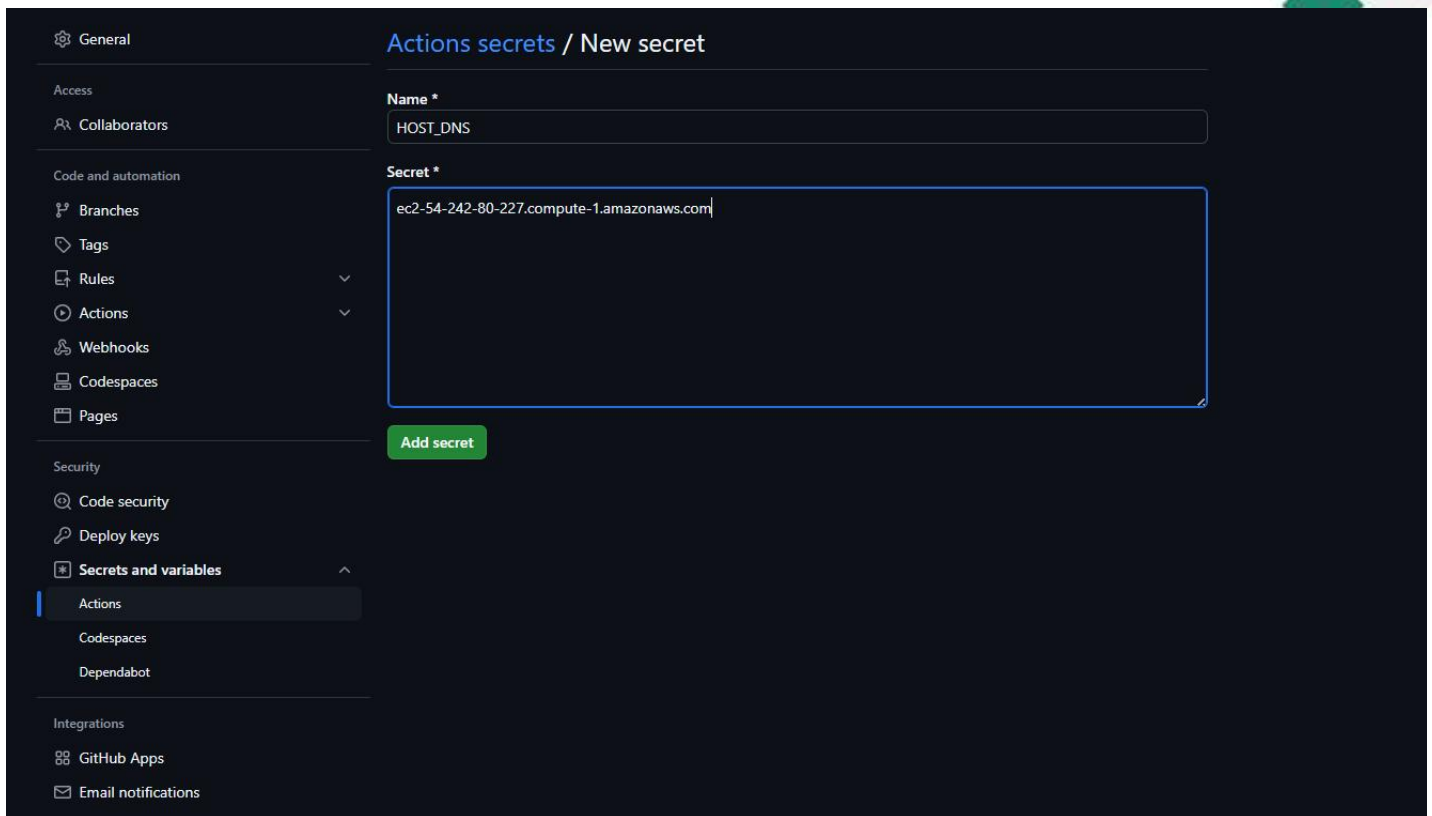


We click on 'New repository secret', we name it "EC2_SSH_KEY", we should open our previously downloaded PEM file, copy all of its content and paste it in 'secret*' as it's shown below:

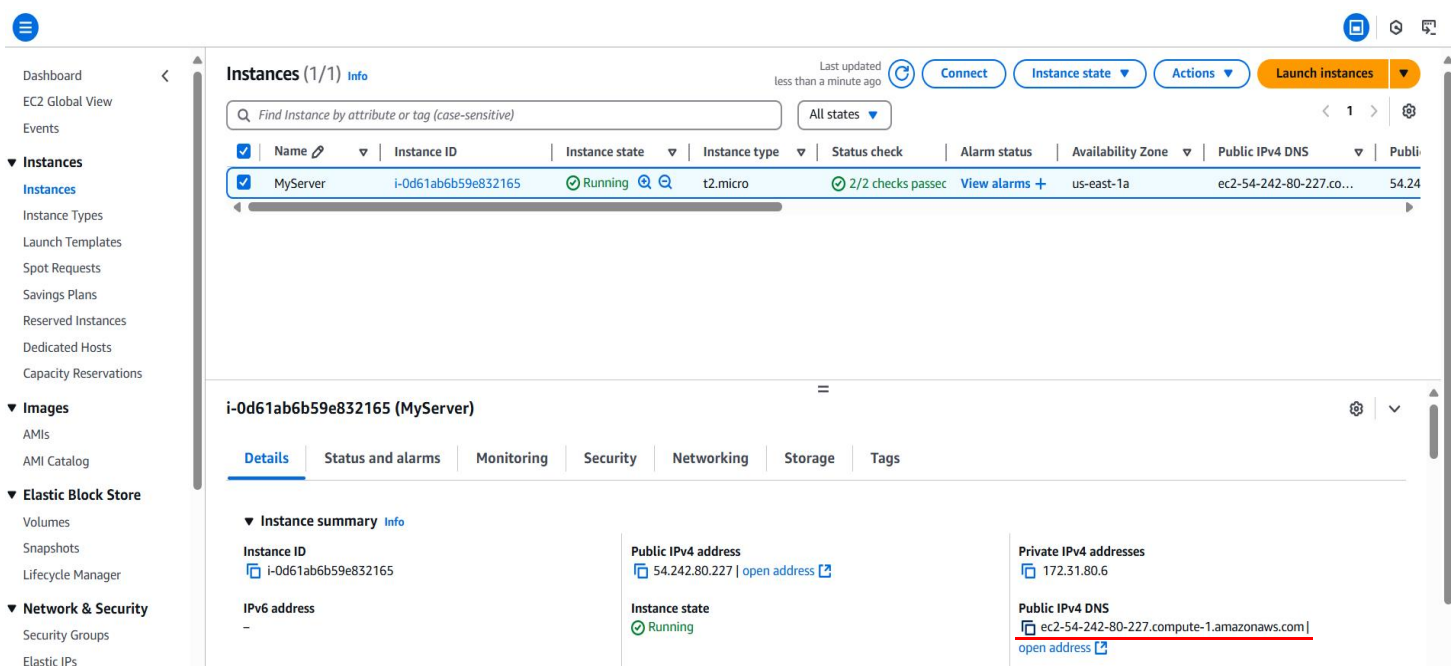


We click on 'Add secret'

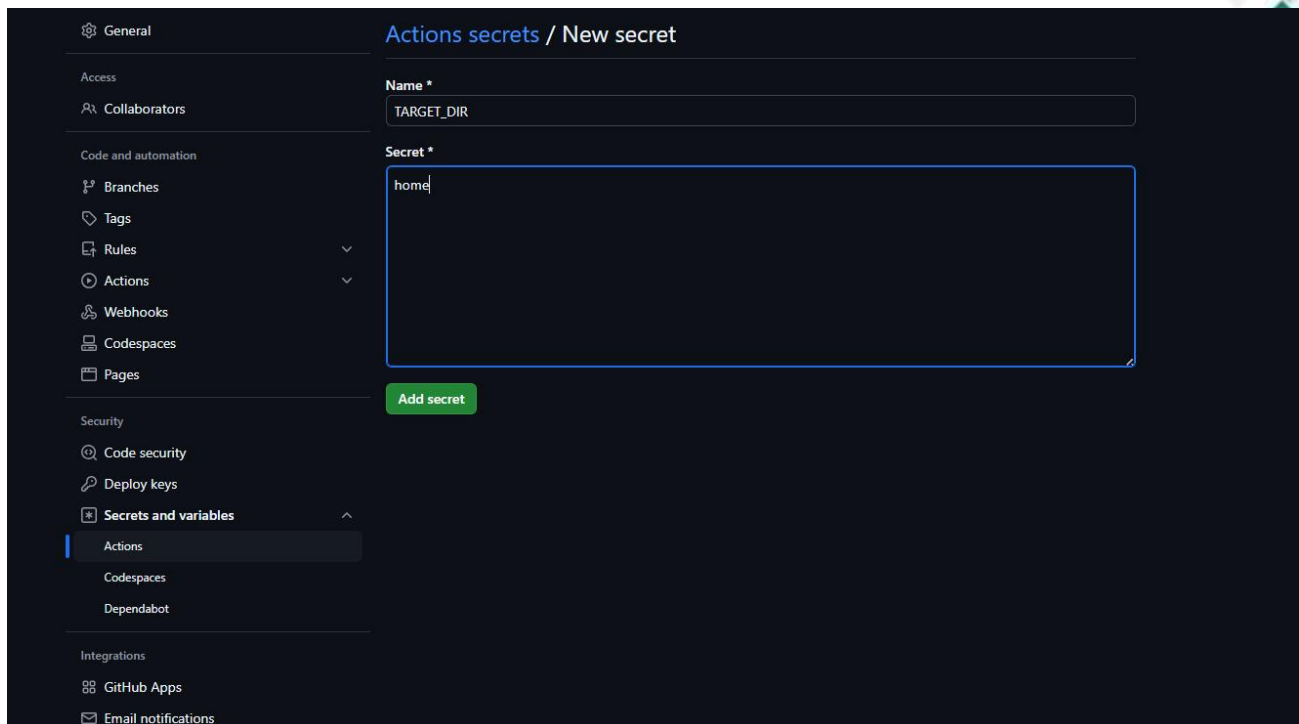
The same steps should be made when creating the other secrets, for the host DNS we should get this



The 'Secret*' (Public IPv4 DNS) value should be got on the **Details** when selecting the current EC2 instance in the **instances** tab like it is shown here

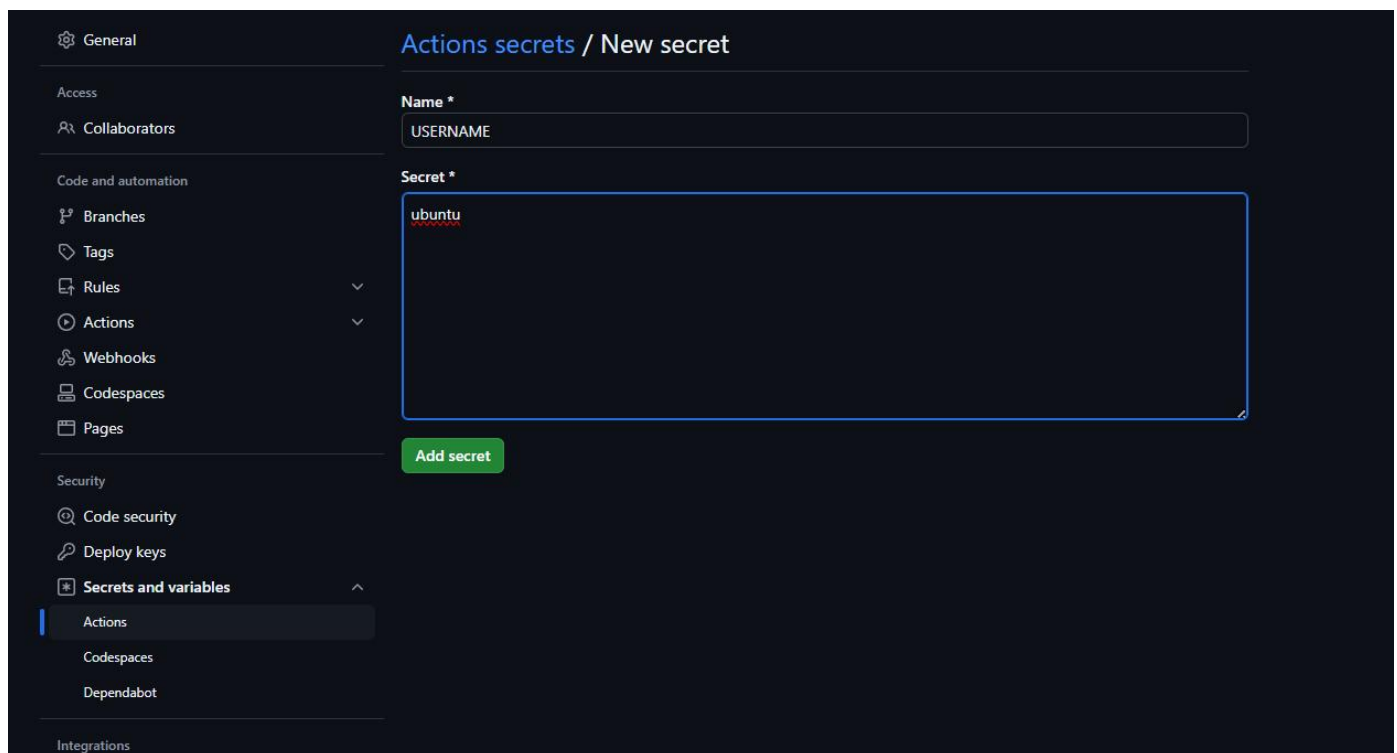


for target directory we should do this:



The screenshot shows the GitHub Actions secrets management interface. On the left, a sidebar contains navigation links: General, Access, Collaborators, Code and automation (with sub-links for Branches, Tags, Rules, Actions, Webhooks, Codespaces, and Pages), Security (with sub-links for Code security, Deploy keys, and Secrets and variables), and Integrations (with sub-links for GitHub Apps and Email notifications). The 'Secrets and variables' section is expanded, and 'Actions' is selected. The main area is titled 'Actions secrets / New secret'. It features a 'Name *' field containing 'TARGET_DIR' and a 'Secret *' text area containing 'home'. A green 'Add secret' button is located below the text area.

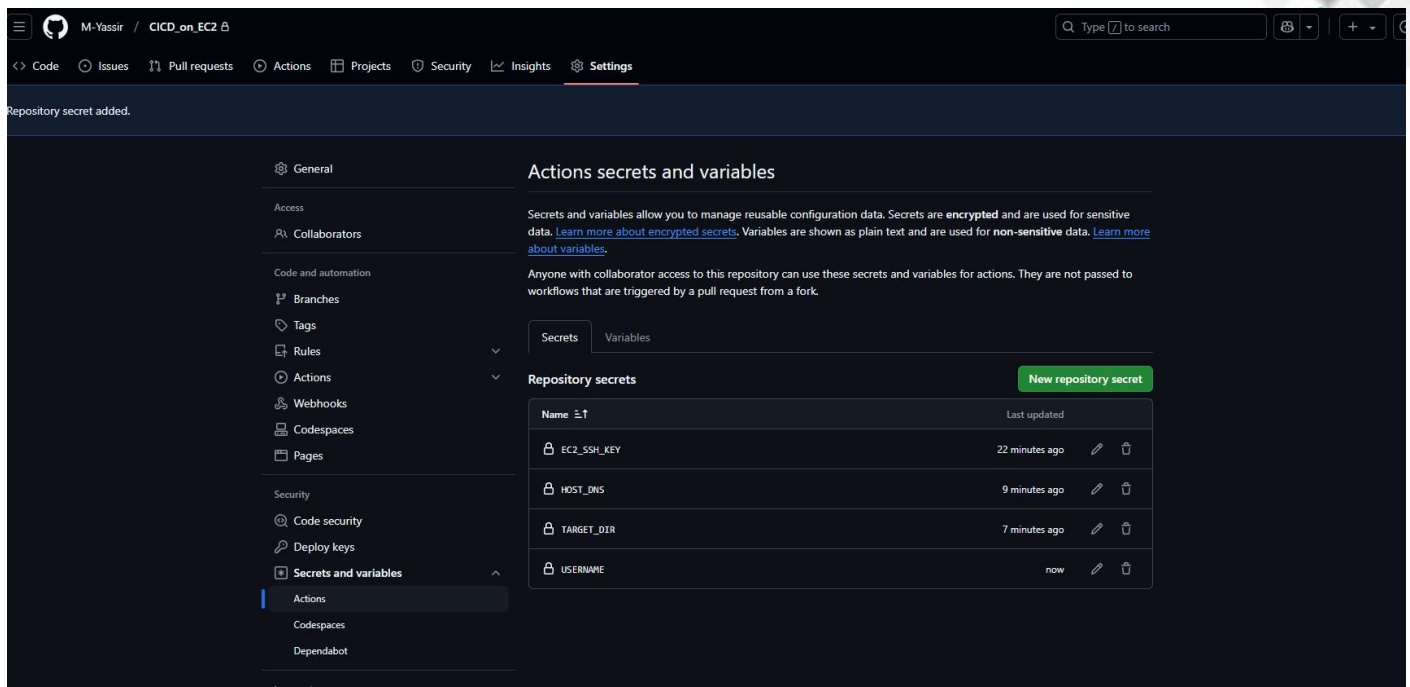
and for username as the default name of an ubuntu EC2 instance is 'ubuntu' we should do this:



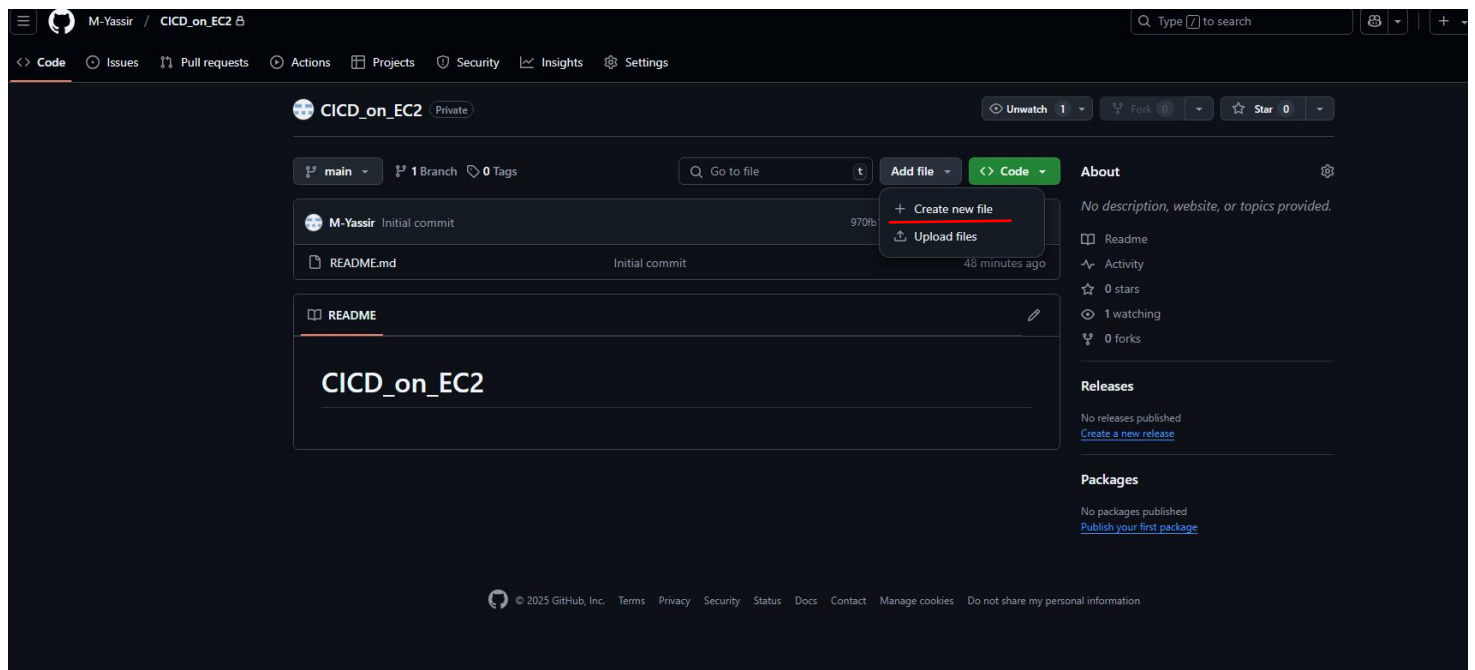
This screenshot is similar to the previous one, showing the 'Actions secrets / New secret' page. In this instance, the 'Name *' field contains 'USERNAME' and the 'Secret *' text area contains 'ubuntu'. The 'Add secret' button remains visible at the bottom of the form.

NB: In order to avoid unexpected errors, all the fields should be the same as above

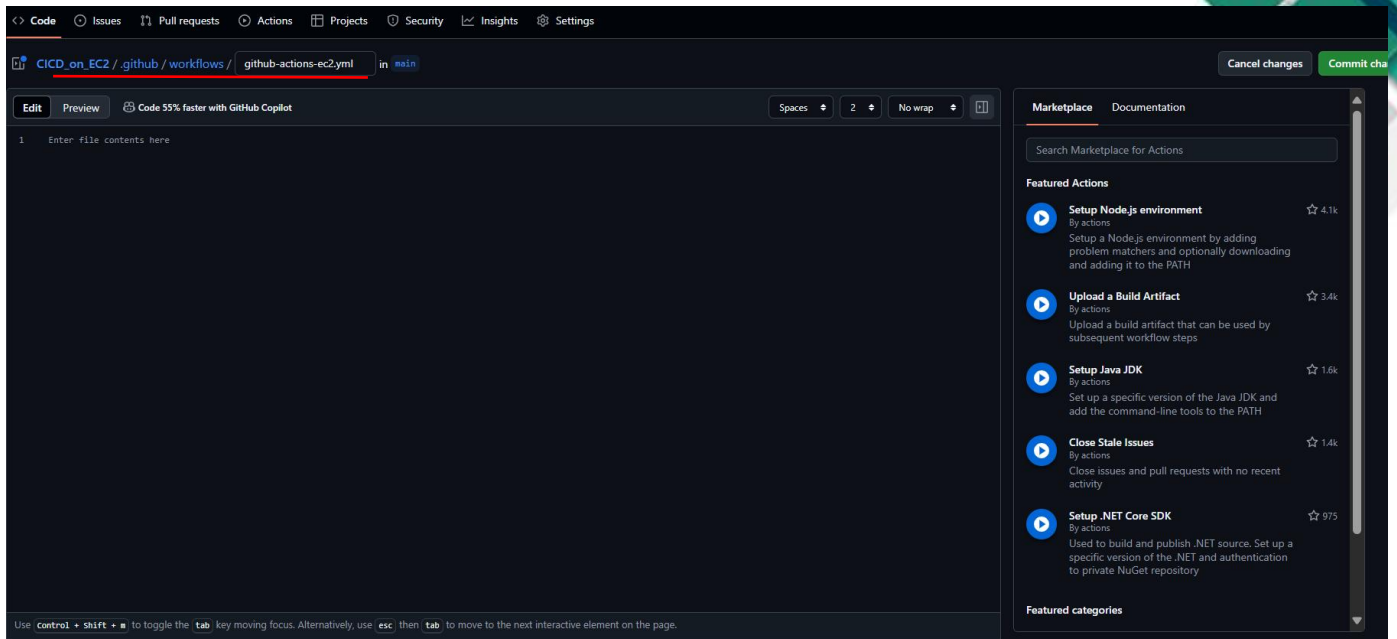
at the end, we should get something like this



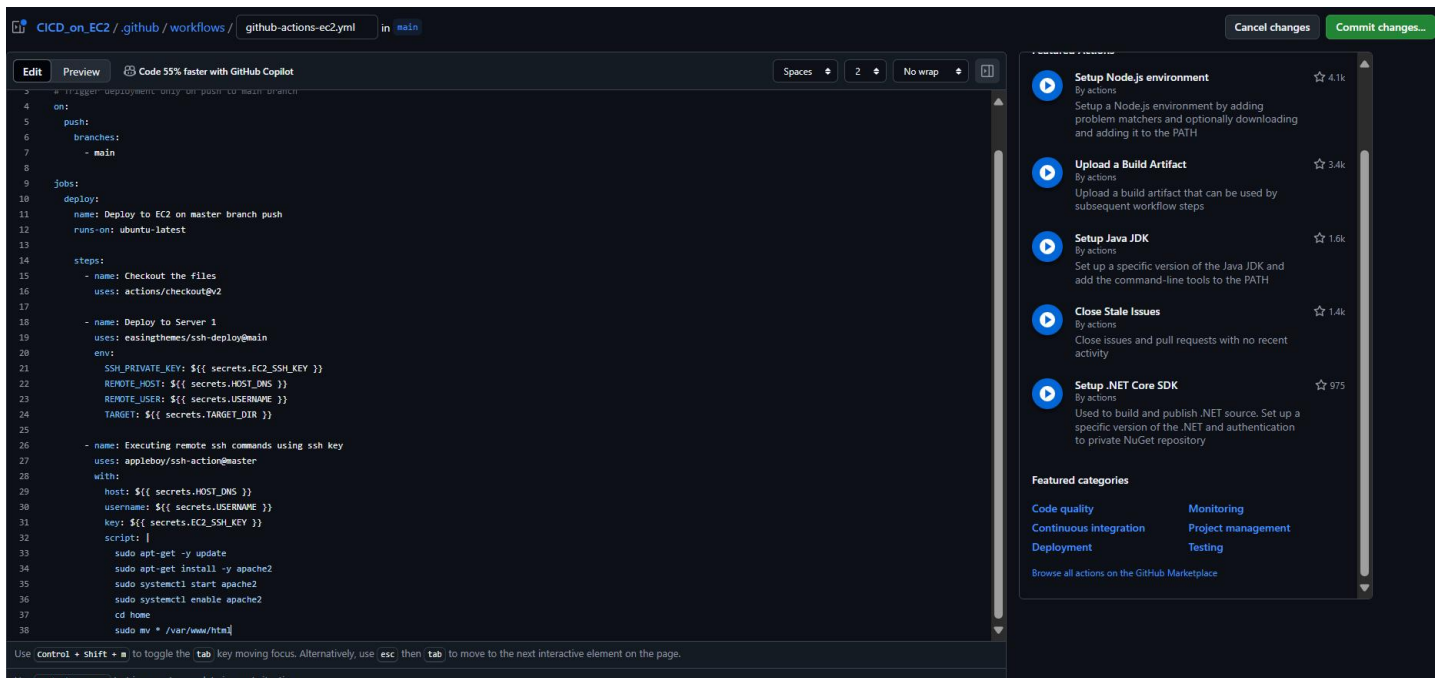
Then, we go to the Code tab, click on 'Add file' and 'Create new file'



Before adding file contents, we should add the following name
“.github/workflows/github-actions-ec2.yml” to the title “CICD_on_EC2”

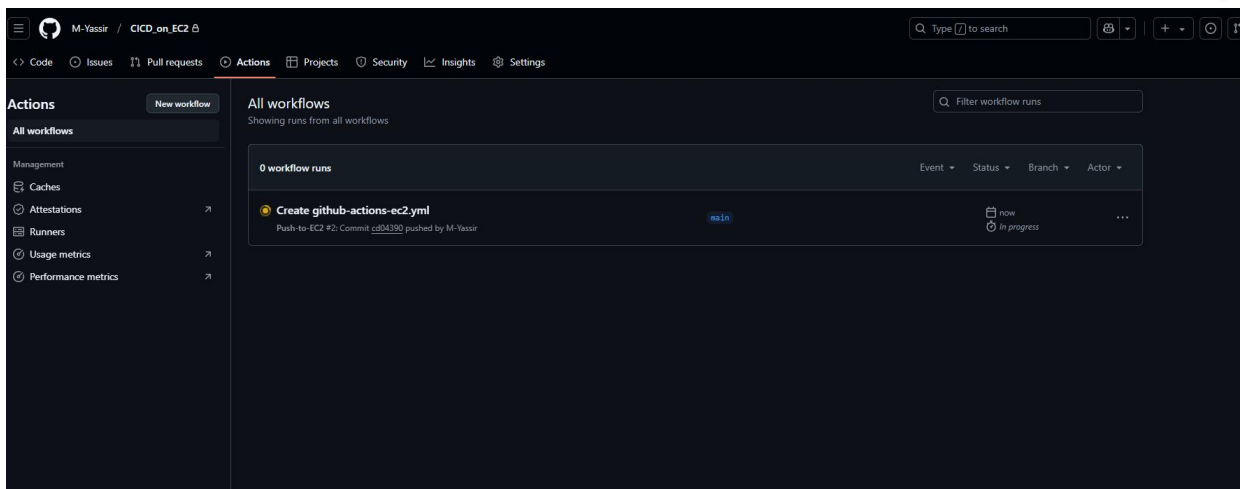


Then, we should type the following code in it (the code is on the repository)

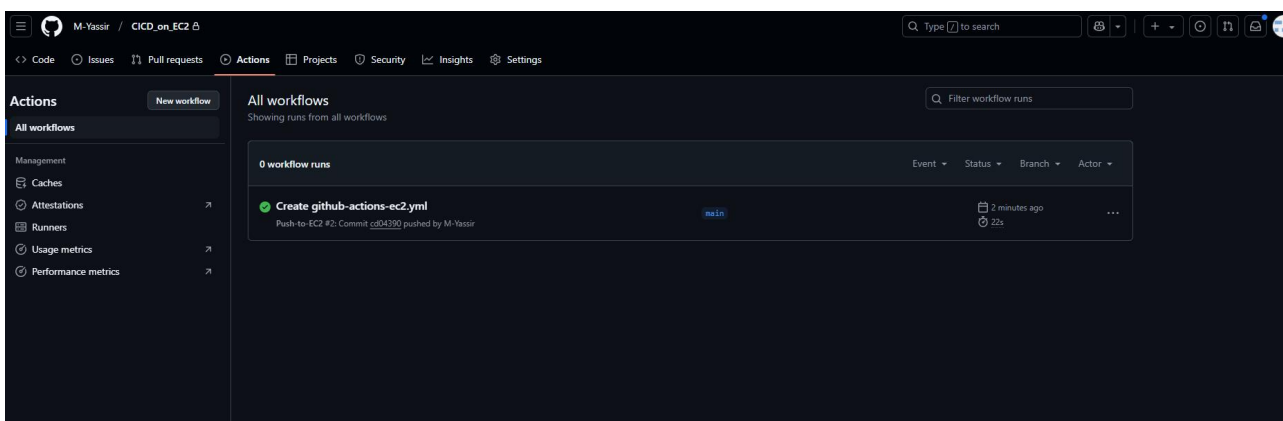


We click on “Commit changes”

We go to the 'Actions' tab, we should see this

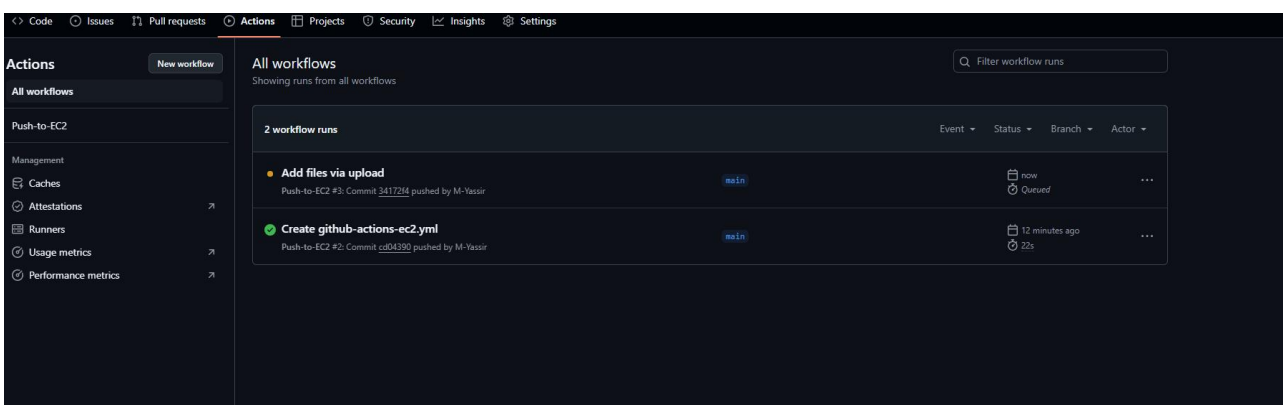


Later, it should be like this

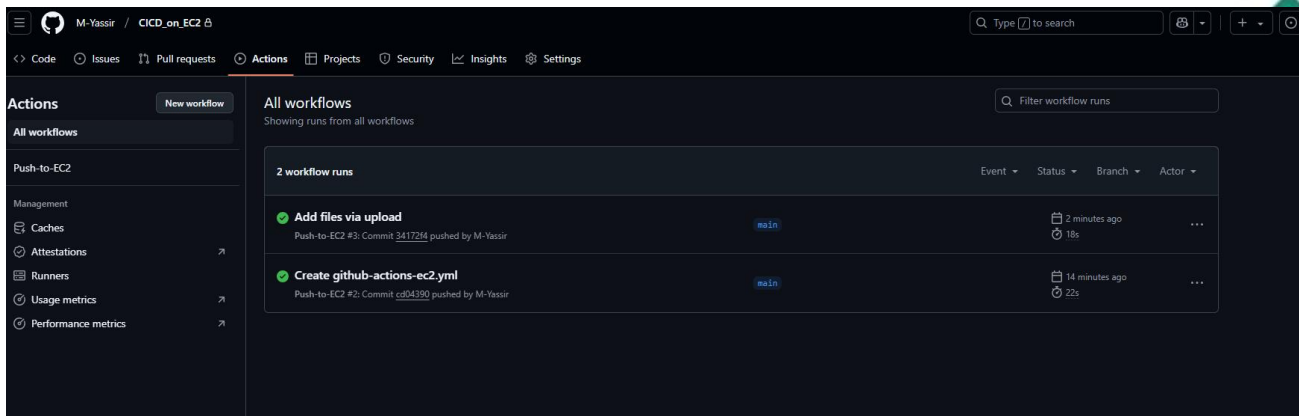


meaning that the operation was successful.

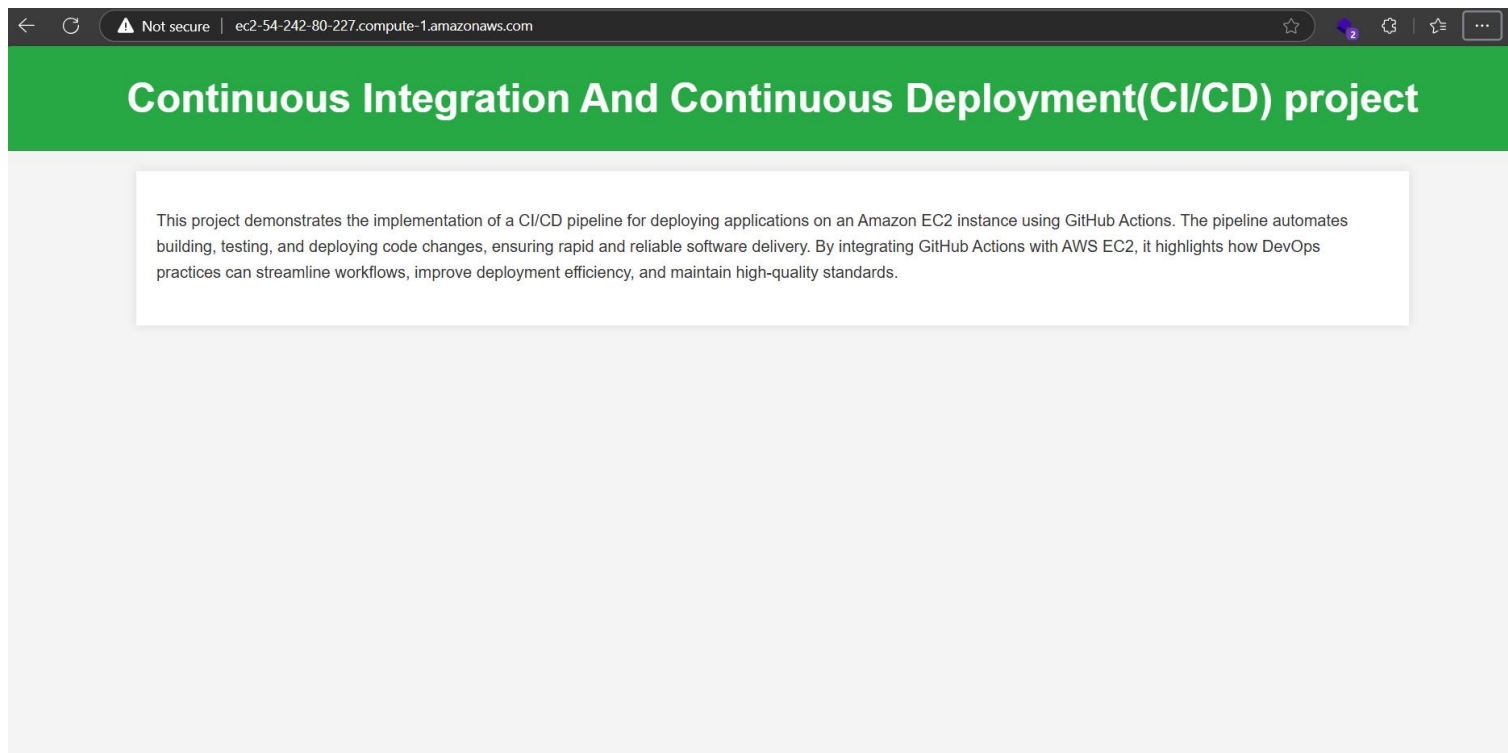
Let's try it out, we're going to add an index.html file to this repo (the code is on the repository), we should get this on the actions tab



Later we should get this

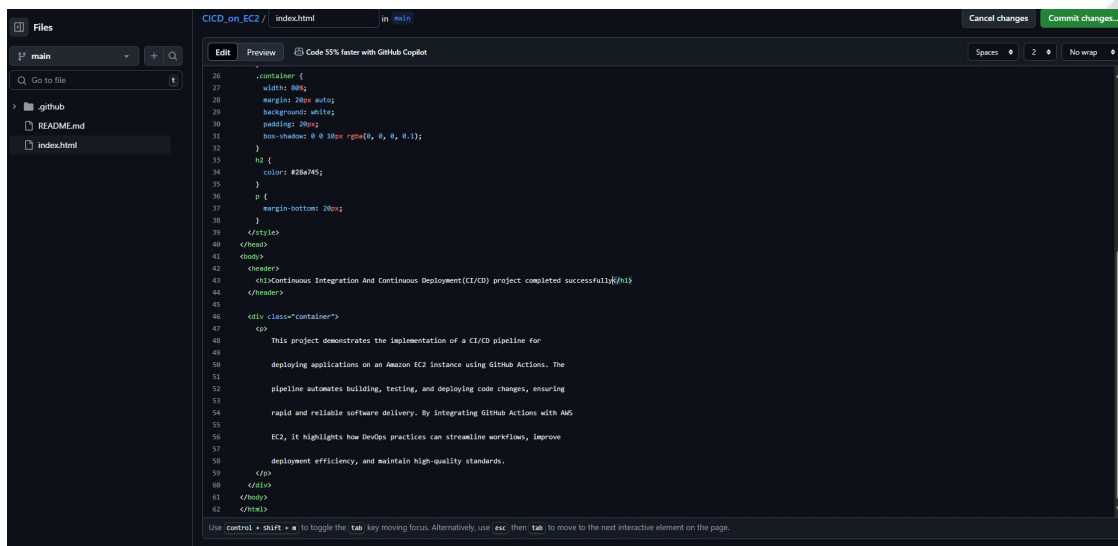


Let's go to the **Public IPv4 DNS** of the instance



and it works.

Let's go now to modify the header of the index.html file on github (add "completed successfully" to the header)



```
26 .container {
27   width: 80%;
28   margin: 20px auto;
29   background: white;
30   padding: 20px;
31   box-shadow: 0 0 10px rgb(0, 0, 0.1);
32 }
33 h2 {
34   color: #28a745;
35 }
36 p {
37   margin-bottom: 20px;
38 }
39 </style>
40 </head>
41 <body>
42 <header>
43   <h1>Continuous Integration And Continuous Deployment(CI/CD) project completed successfully</h1>
44 </header>
45
46 <div class="container">
47   <p>
48     This project demonstrates the implementation of a CI/CD pipeline for
49
50     deploying applications on an Amazon EC2 instance using GitHub Actions. The
51
52     pipeline automates building, testing, and deploying code changes, ensuring
53
54     rapid and reliable software delivery. By integrating GitHub Actions with AWS
55
56     EC2, it highlights how DevOps practices can streamline workflows, improve
57
58     deployment efficiency, and maintain high-quality standards.
59   </p>
60 </div>
61 </body>
62 </html>
```

By committing changes and waiting till it finishes the update and revisit the public ipv4 DNS of that instance, we should get this



and it also works.

Conclusion

This project has been an insightful journey into building a Continuous Integration and Continuous Deployment (CI/CD) pipeline for an EC2-based web application using GitHub Actions. By leveraging GitHub Actions for automation and deploying directly to an EC2 instance, I successfully implemented an efficient and reliable deployment process for managing application updates.

Challenges Faced & Lessons Learned:

1) GitHub Actions Workflow Configuration

One of the major lessons was configuring the GitHub Actions workflow to ensure seamless deployment.

I learned how to structure multi-step workflows, manage secrets securely, and ensure proper communication with the EC2 instance.

2) Monitoring Deployment Success

Initially, verifying whether deployments were successful on the EC2 instance was not straightforward.

I learned how to incorporate logging and verification steps to ensure that the latest code was properly deployed and accessible.

3) SSH Key Authentication & Security

Managing secure connections to the EC2 instance via SSH required proper handling of keys and permissions.

I learned how to store and use SSH keys safely in GitHub Secrets and automate secure deployments without exposing sensitive information.

Final Takeaways:

Through this project, I gained hands-on experience in CI/CD pipeline development and learned how GitHub Actions and EC2 work together to create a streamlined deployment process. Some key skills I developed include:

- ✓ Automating deployments to EC2 using GitHub Actions
- ✓ Configuring multi-step workflows with secure secret management
- ✓ Verifying deployment success through automated logging
- ✓ Managing SSH-based deployments for cloud environments

This project has deepened my understanding of CI/CD pipelines for cloud-based environments and reinforced the importance of automation and continuous delivery. Moving forward, I am excited to apply these skills to build even more robust and scalable deployment pipelines.