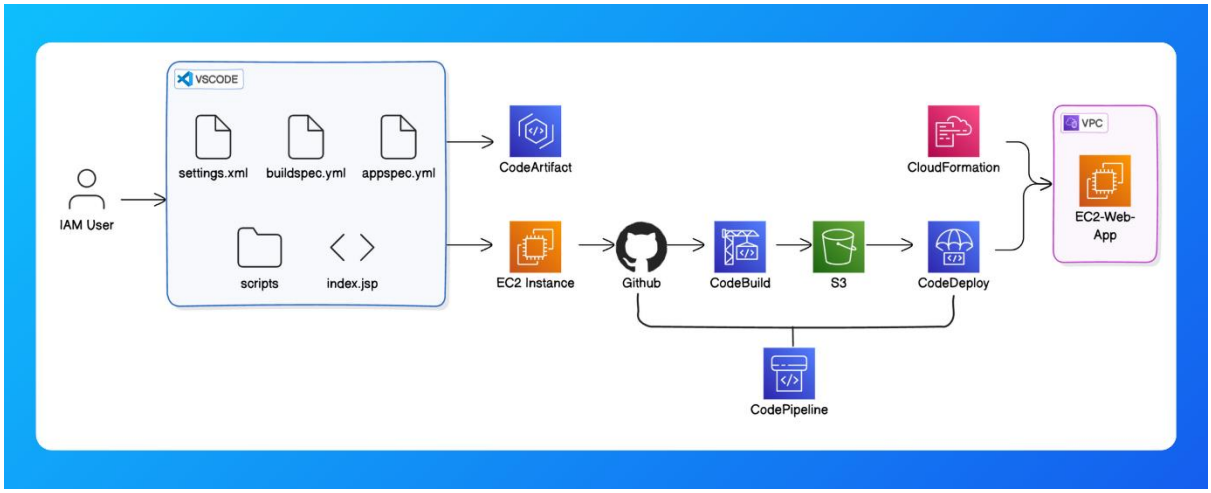
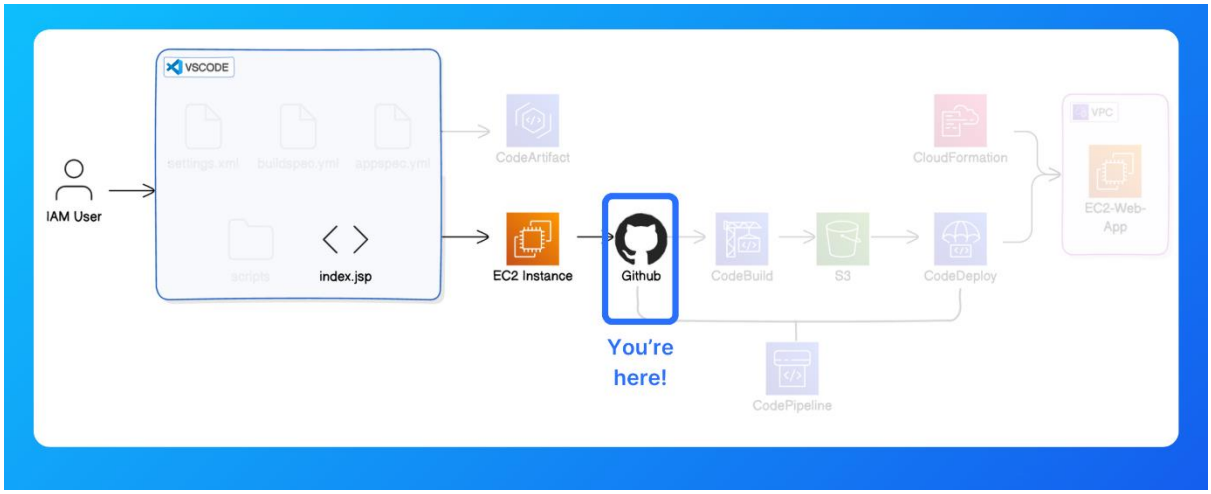
Connect a GitHub Repo with AWS

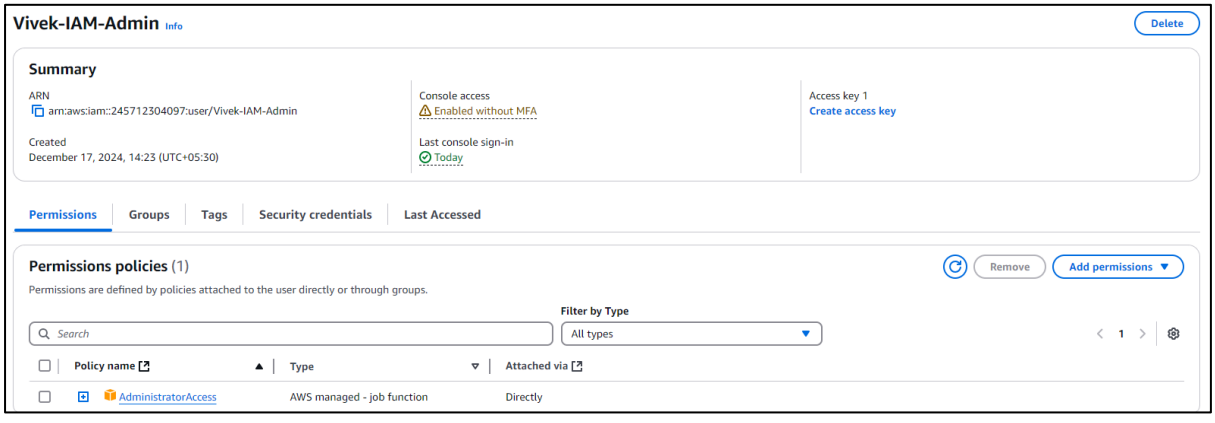




In this Project we will be Setting Up a Repository for our source code.

This Project is the Follow-up for Project - Set Up a Web App in the Cloud

- Setup IAM User



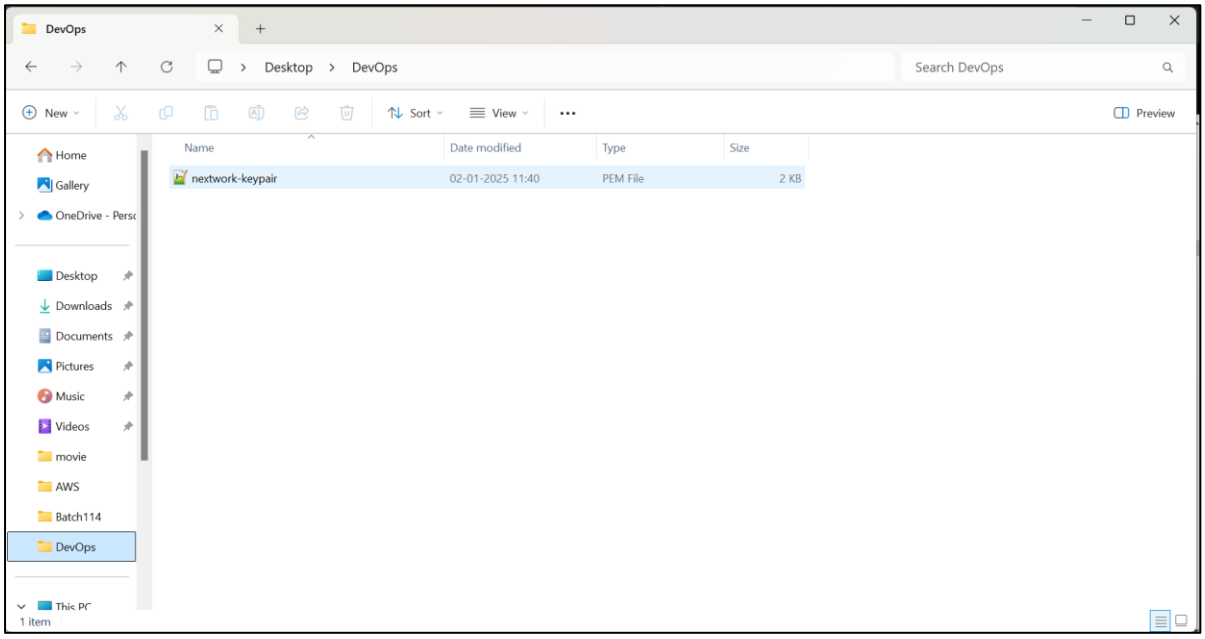
- Log In with your IAM Admin User.

Set Up Your Web App in the Cloud (Project 1 of this DevOps Series)

In this step, you're going to:

1. Launch an EC2 instance.
2. Set up VSCode on your local computer.
3. Use Remote - SSH to connect VSCode with your EC2 instance.

- Launch an EC2 Instance
    - **Enter the Name of the Instance**, eg: nextwork-devops-vivek
    - Choose **Amazon Linux 2023 AMI** under **Amazon Machine Image(AMI)**
    - Choose **t2.micro** under **Instance type**.
    - Under **Key pair (login)**, choose **Create a new key pair.**
    - Use nextwork-keypair as your key pair's name.
    - Keep the **Key pair type** as **RSA**, and the **Private key file format** as **.pem**
    - A new file will automatically download to your local computer.This is your private key. Before we lose track of our **.pem** file, let's organise it in our computer.
    - Head to your local computer's desktop.
    - Create a new folder in your desktop called **DevOps**.
    - Move your .pem file from your **Downloads** folder into your **DevOps** folder



    - Back to our EC2 instance setup, head to the **Network settings** section
    - For **Allow SSH traffic from**, select the dropdown and choose **My IP**. This makes sure only you can access your EC2 instance.
    - If your IP address is different from what's under **My IP**, select **Custom** from the dropdown instead. Enter your IP and make sure to add a /32 to the end e.g. 49.204.239.202/32
    - Leave the default values for the remaining sections

EC2 > Instances > Launch an instance

Launch an instance Info

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

nextwork-devops-vivek

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 | D

Browse more AMIs

Including AMIs from AWS, Marketplace and the Community

Amazon Machine Image (AMI)

Amazon Linux 2023 AMI — Free tier eligible

ami-0fd05997b4dff7aac (64-bit (x86), uefi-preferred) / ami-013b2876e77b2db31 (64-bit (Arm), uefi)

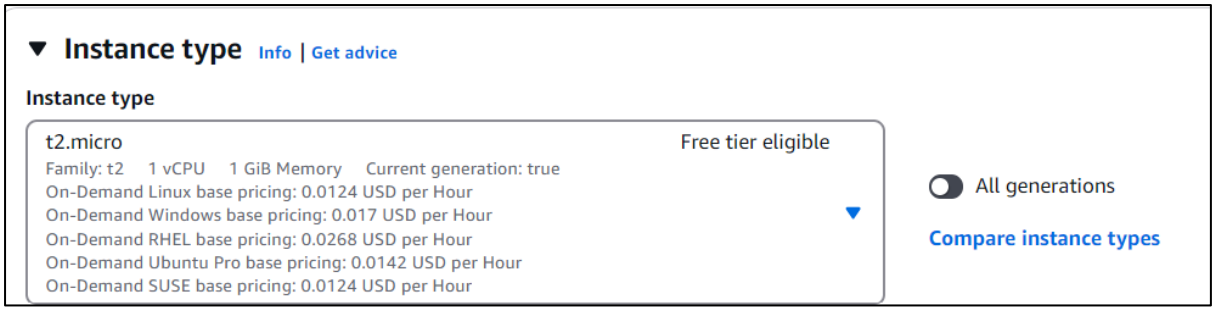
Virtualization: hvm ENA enabled: true Root device type: ebs

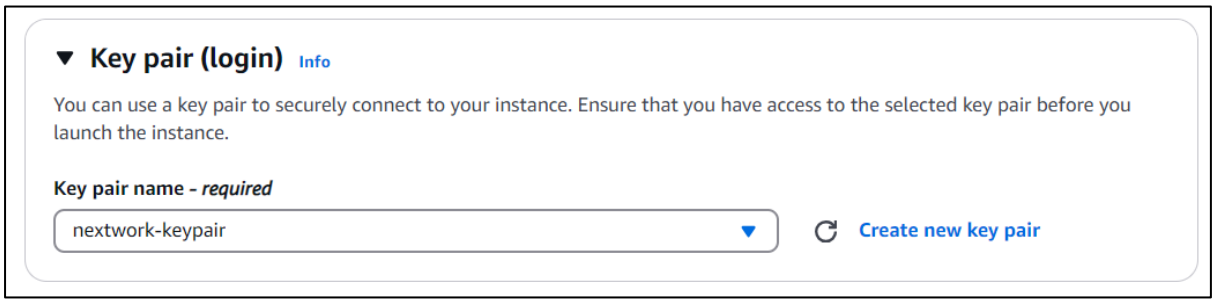
Description

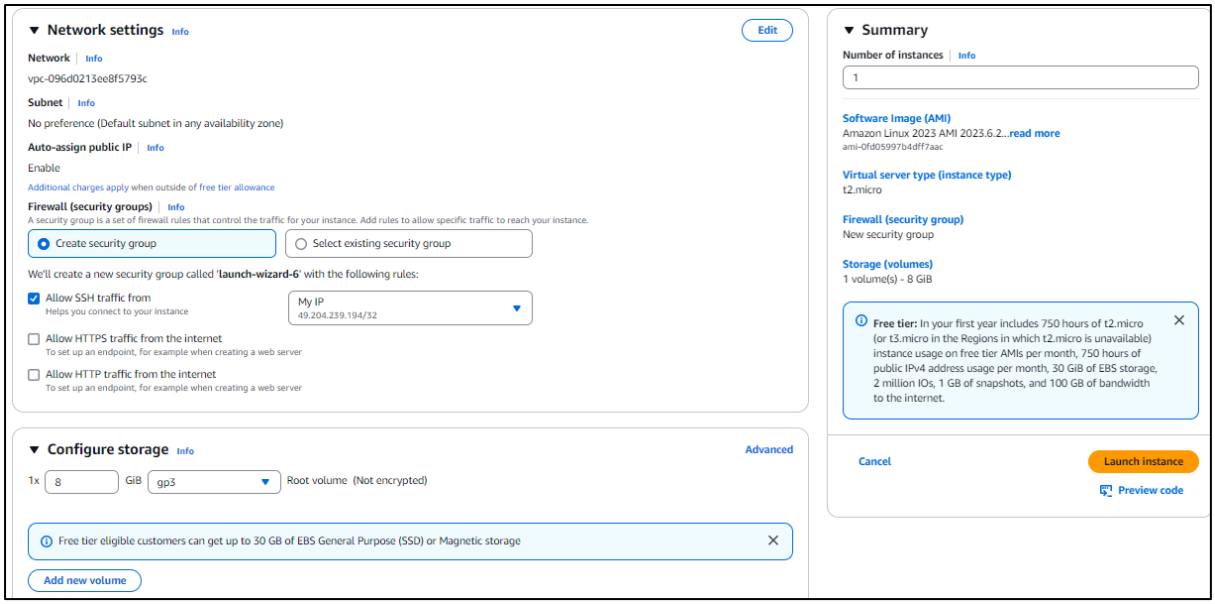
Amazon Linux 2023 is a modern, general purpose Linux-based OS that comes with 5 years of long term support. It is optimized for AWS and designed to provide a secure, stable and high-performance execution environment to develop and run your cloud applications.

Amazon Linux 2023 AMI 2023.6.20241212.0 x86_64 HVM kernel-6.1

Architecture	Boot mode	AMI ID	Username	
64-bit (x86)	uefi-preferred	ami-0fd05997b4dff7aac	ec2-user	Verified provider

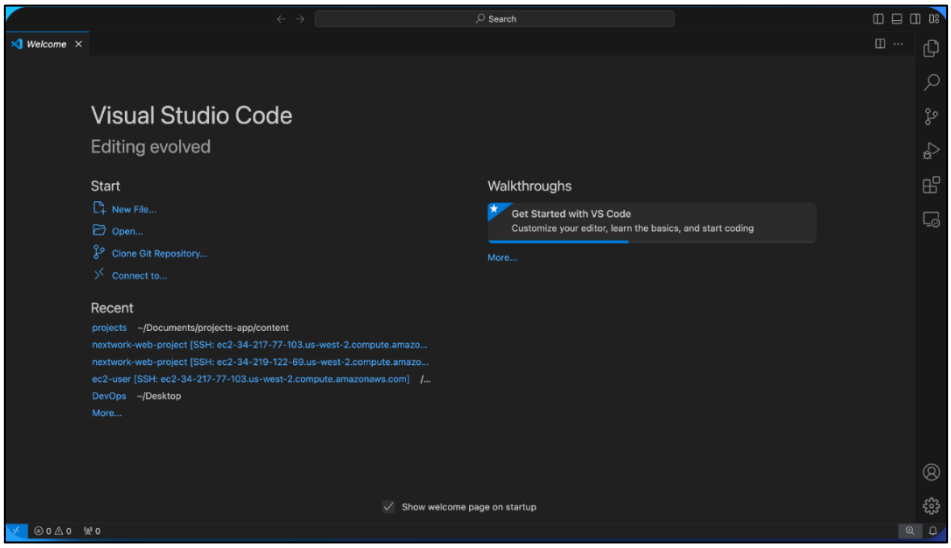




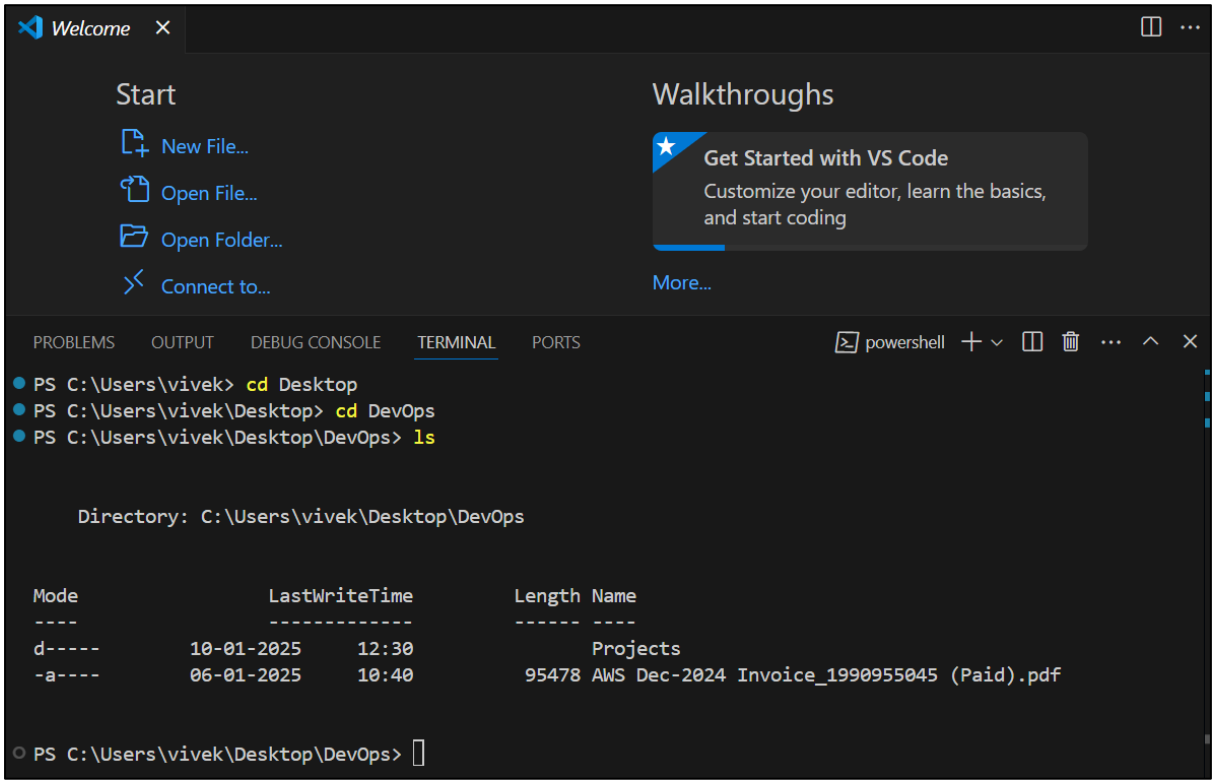


- choose **Launch instance**

- Open VS Code



  - Select **Terminal** from the top menu bar.
  - Select **New Terminal** from the dropdown
  - Navigate your terminal to the DevOps folder. You'll do this by entering this command in the terminal: cd Desktop and cd DevOps
  - Once you're in the DevOps folder, you might want to check if your .pem file is there. Use this command: ls



  - **Change the permissions of your .pem file**
  - In the terminal, run the following command to allow access to your .pem file.

ssh -i [PATH TO YOUR .PEM FILE] ec2-user@[YOUR PUBLIC IPV4 ADDRESS]

eg: ssh -i **C:\Users\vivek\Desktop\DevOps\nextwork-keypair.pem** ec2-user@ **15.206.185.184**

- **Are you sure you want to continue connecting (yes/no/[fingerprint]) will flash on the screen**
- **Type yes**

```
PS C:\Users\vivek\Desktop\DevOps> ssh -i C:\Users\vivek\Desktop\DevOps\Projects\nextwork-keypair.pem ec2
-user@15.206.185.184
The authenticity of host '15.206.185.184 (15.206.185.184)' can't be established.
ED25519 key fingerprint is SHA256:MR2TXF2jXa3RM0XIJJt238NbnN9ovYGjQNaBM5TTIQM.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

- You will connect to your EC-2 Instance

```
Warning: Permanently added '15.206.185.184' (ED25519) to the list of known hosts.
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-172-31-3-196 ~]$
```

- **Install Apache Maven and Amazon Corretto 8**

Now that the Connection DONE, This means your terminal has now entered into your EC2 instance and can use it like a computer that's right in front of you!

Our goal today is to set up a web app inside this instance, so let's install two tools that are going to help us build Java web apps.

Apache Maven and Amazon Corretto 8

- **In this step, you're going to:**

1. Install Apache Maven on your EC2 instance.
2. Install Amazon Corretto 8, a version of Java.
3. Verify the installations.

- Install Apache Maven using the commands below. You can copy and paste **all** of these lines into the terminal together, no need to run them line by line.

wget https://archive.apache.org/dist/maven/maven-3/3.5.2/binaries/apache-maven-3.5.2-bin.tar.gz

sudo tar -xzf apache-maven-3.5.2-bin.tar.gz -C /opt

echo "export PATH=/opt/apache-maven-3.5.2/bin:$PATH" >> ~/.bashrc

source ~/.bashrc

```
[ec2-user@ip-172-31-3-196 ~]$ wget https://archive.apache.org/dist/maven/maven-3/3.5.2/binaries/apache-m
aven-3.5.2-bin.tar.gz

sudo tar -xzf apache-maven-3.5.2-bin.tar.gz -C /opt

echo "export PATH=/opt/apache-maven-3.5.2/bin:$PATH" >> ~/.bashrc

source ~/.bashrc
--2025-01-10 07:49:01--  https://archive.apache.org/dist/maven/maven-3/3.5.2/binaries/apache-maven-3.5.2
-bin.tar.gz
Resolving archive.apache.org (archive.apache.org)... 65.108.204.189, 2a01:4f9:1a:a084::2
Connecting to archive.apache.org (archive.apache.org)|65.108.204.189|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 8738691 (8.3M) [application/x-gzip]
Saving to: ‘apache-maven-3.5.2-bin.tar.gz’

apache-maven-3.5.2-bin.ta 100%[===================================>]   8.33M  4.12MB/s    in 2.0s

2025-01-10 07:49:03 (4.12 MB/s) - ‘apache-maven-3.5.2-bin.tar.gz’ saved [8738691/8738691]

[ec2-user@ip-172-31-3-196 ~]$
```

Apache Maven Installed

- Now we're going to install Java 8, or more specifically, Amazon Correto 8.
- Run these commands:

sudo dnf install -y java-1.8.0-amazon-corretto-devel

export JAVA_HOME=/usr/lib/jvm/java-1.8.0-amazon-corretto.x86_64

export PATH=/usr/lib/jvm/java-1.8.0-amazon-corretto.x86_64/jre/bin/:$PATH

- To verify that Maven is installed correctly, run the following command next:

mvn -v

```
[ec2-user@ip-172-31-3-196 ~]$ mvn -v
Apache Maven 3.5.2 (138edd61fd100ec658bfa2d307c43b76940a5d7d; 2017-10-18T07:58:13Z)
Maven home: /opt/apache-maven-3.5.2
Java version: 1.8.0_432, vendor: Amazon.com Inc.
Java home: /usr/lib/jvm/java-1.8.0-amazon-corretto.x86_64/jre
Default locale: en, platform encoding: UTF-8
OS name: "linux", version: "6.1.119-129.201.amzn2023.x86_64", arch: "amd64", family: "unix"
[ec2-user@ip-172-31-3-196 ~]$
```

- To verify that you've installed Java 8 correctly, run this next:

java -version

```
[ec2-user@ip-172-31-3-196 ~]$ java -version
openjdk version "1.8.0_432"
OpenJDK Runtime Environment Corretto-8.432.06.1 (build 1.8.0_432-b06)
OpenJDK 64-Bit Server VM Corretto-8.432.06.1 (build 25.432-b06, mixed mode)
[ec2-user@ip-172-31-3-196 ~]$
```

Create the Application

We've assembled both Maven and Java into our EC2 instance. Now let's cut straight to generating the web app!

In this step, you're going to:

- Run Maven commands in your terminal to generate a Java web app.
- Use **mvn** to generate a Java web app. To do this, use these commands:

```
mvn archetype:generate \
  -DgroupId=com.nextwork.app \
  -DartifactId=nextwork-web-project \
  -DarchetypeArtifactId=maven-archetype-webapp \
  -DinteractiveMode=false
```

- Watch out for a **BUILD SUCCESS** message in your terminal once your application is all set up.

```
[INFO] ----------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-webapp:1.0
[INFO] ----------------------------------------------------------------------------
[INFO] Parameter: basedir, Value: /home/ec2-user
[INFO] Parameter: package, Value: com.nextwork.app
[INFO] Parameter: groupId, Value: com.nextwork.app
[INFO] Parameter: artifactId, Value: nextwork-web-project
[INFO] Parameter: packageName, Value: com.nextwork.app
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: /home/ec2-user/nextwork-web-project
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 8.536 s
[INFO] Finished at: 2025-01-02T10:19:40Z
[INFO] Final Memory: 16M/92M
[INFO] ------------------------------------------------------------------------
```

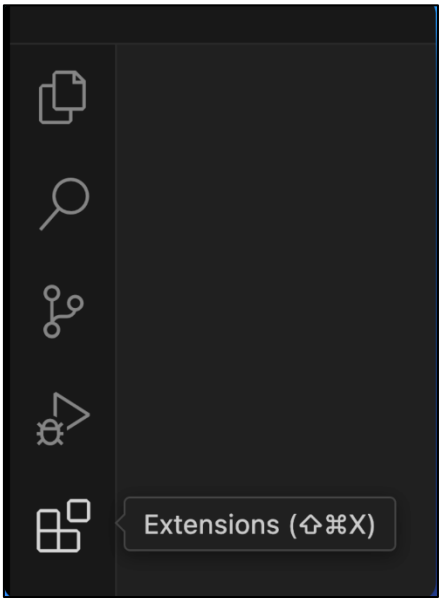
Connect VSCode with your EC2 Instance

In this step, you'll connect VSCode to your EC2 instance so you can see and edit the web app you've just created.

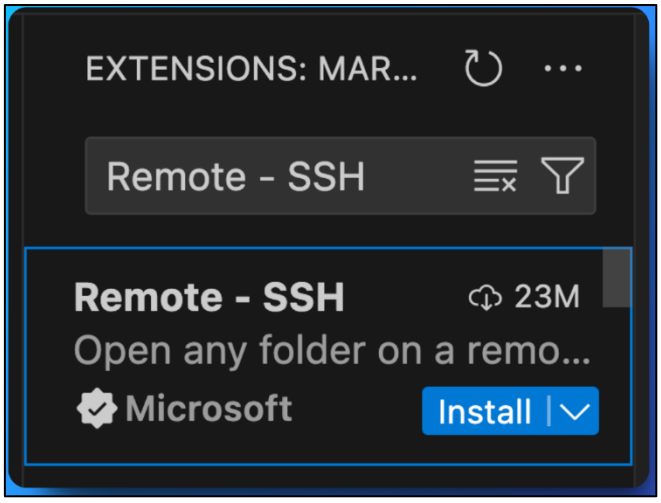
In this step, you're going to:

- Install an extension in VSCode.
- Use the extension to set up a connection between VSCode and your EC2 instance.
- Explore and edit your Java web app's files using VSCode.

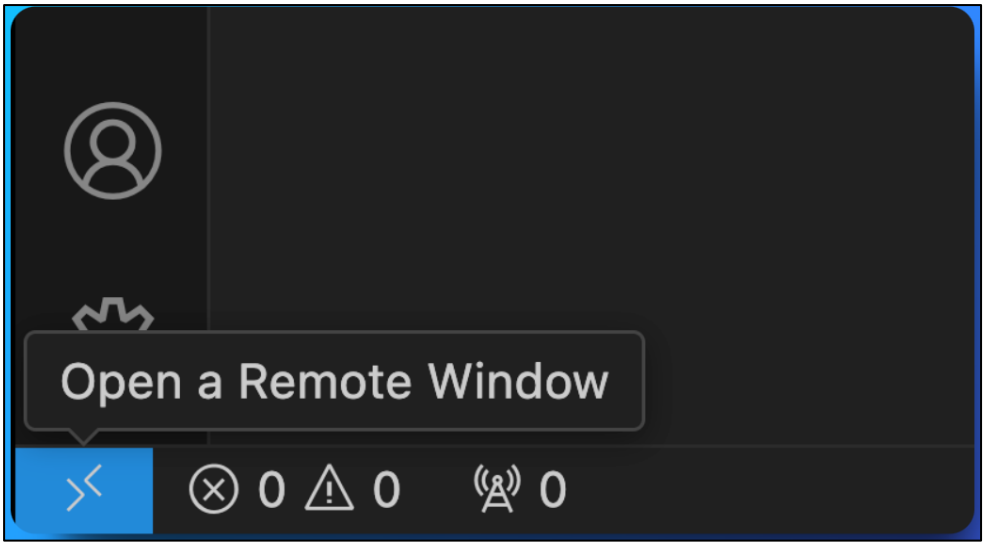
- Clicking on the **Extensions** icon at the side of your VSCode window.



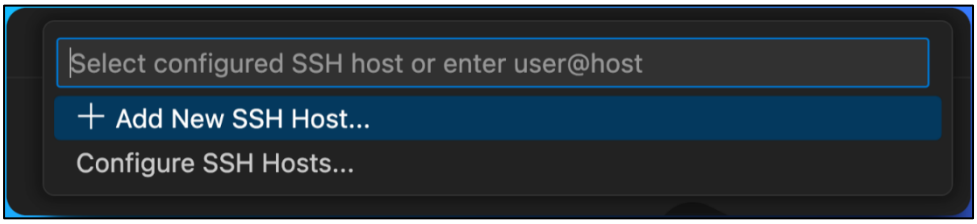
- In the search bar, type Remote - SSH and click **Install** for the extension.



- Click on the double arrow icon at the bottom left corner of your VSCode window. This button is a shortcut to use Remote - SSH.



- Select **Remote-SSH: Connect to Host...**
- Select **+ Add New SSH Host...**



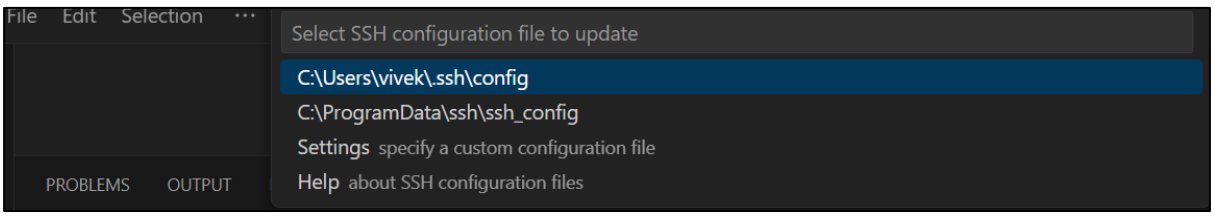
- Enter the SSH command you used to connect to your EC2 instance:

  ssh -i [PATH TO YOUR .PEM FILE] ec2-user@[YOUR PUBLIC IPV4 DNS]

- Replace **[PATH TO YOUR .PEM FILE]** with the actual path to your private key file (e.g., ~/Desktop/DevOps/nextwork-keypair.pem). Delete the square brackets!
- Replace **[YOUR PUBLIC IPV4 DNS]** with the Public DNS you just found. Delete the square brackets!

  Eg: ssh -i C:\Users\vivek\Desktop\DevOps\Projects\nextwork-keypair.pem ec2-user@ ec2-15-206-185-184.ap-south-1.compute.amazonaws.com

- Select the configuration file at the top of your window. It should look similar to /Users/username/.ssh/config



- A **Host added!** popup will confirm that you've set up your SSH Host - yay!
- Select the blue **Open Config** button on that popup.
- Confirm that all the details in your configuration file look correct:

- **Host** should match up with your EC2 instance's IPv4 DNS.
- **IdentityFile** should match up to nextwork-keypair.pem's location in your local computer.
- **User** should say ec2-user

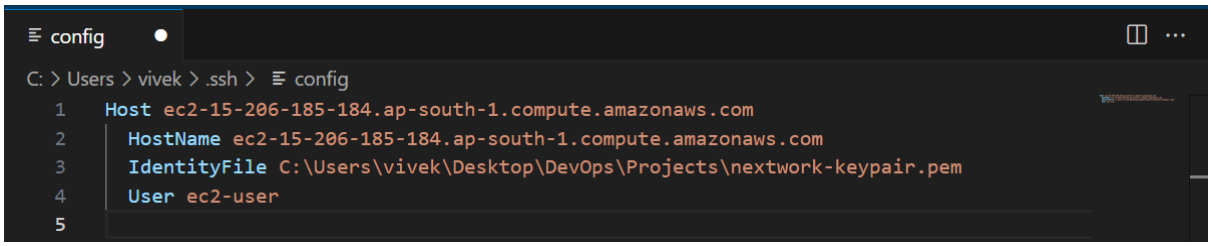
- **As you can see the Identity File Path is not correct, edit the config Identity File Path**

From

```
C:UsersvivekDesktopDevOpsProjectsnextwork-keypair.pem
```

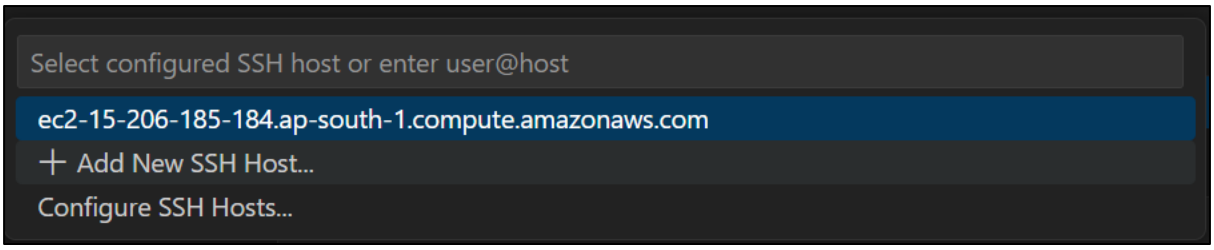
to

```
C:\Users\vivek\Desktop\DevOps\Projects\nextwork-keypair.pem
```

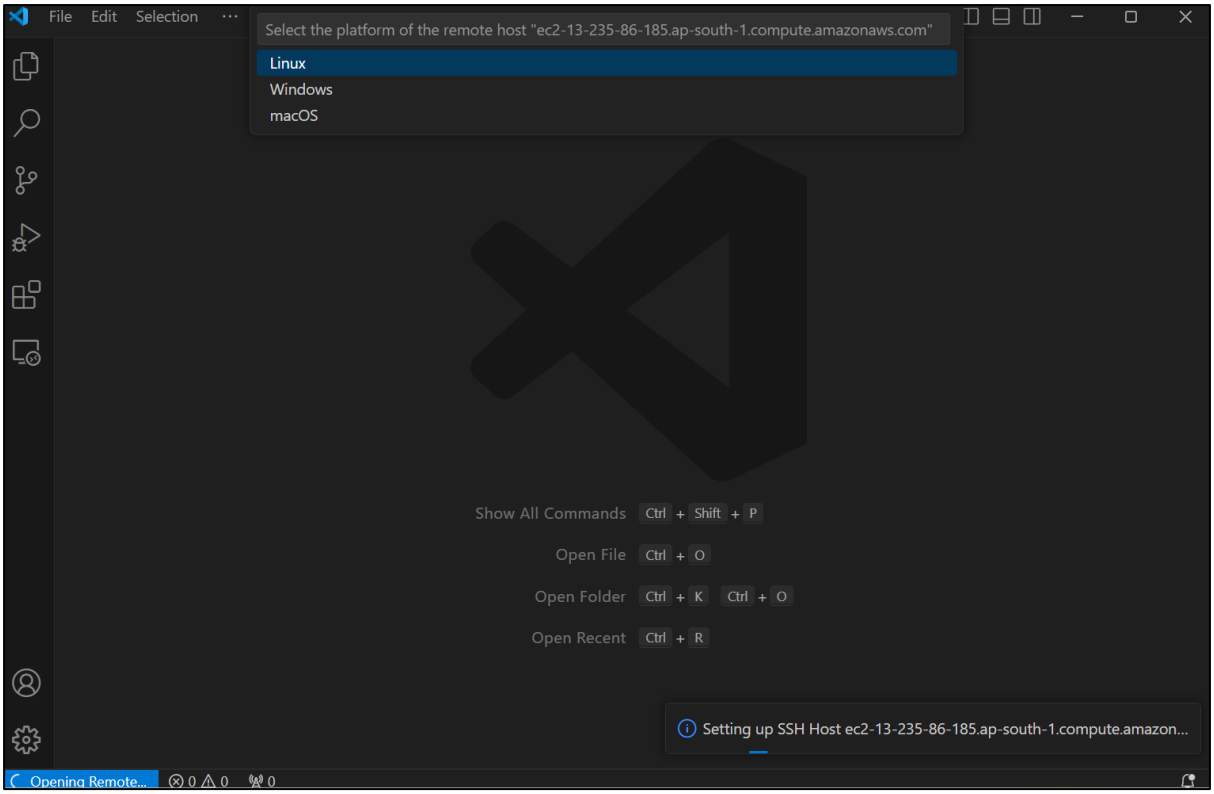


Then Press Ctrl+S to save the config.

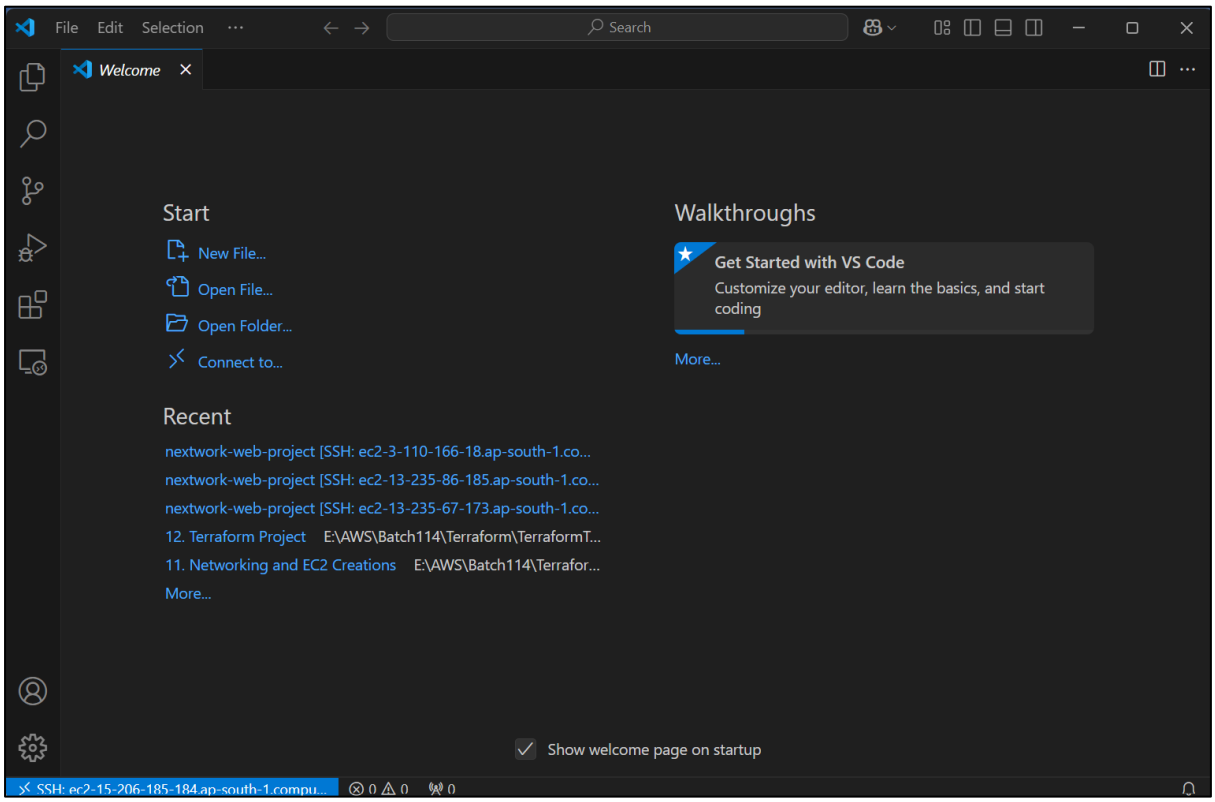
- Now you're ready to connect VSCode with your EC2 instance!
- Click on the double arrow button on the bottom left corner and select **Connect to Host** again.
- You should now see your EC2 instance listed at the top.



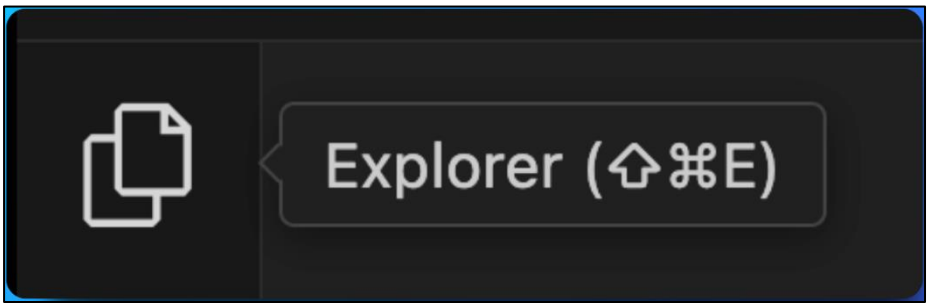
- Select the EC2 instance and off we go to a new VSCode window
- Check the bottom right hand corner of your new VSCode window - it should show your EC2 instance's IPV4 DNS.



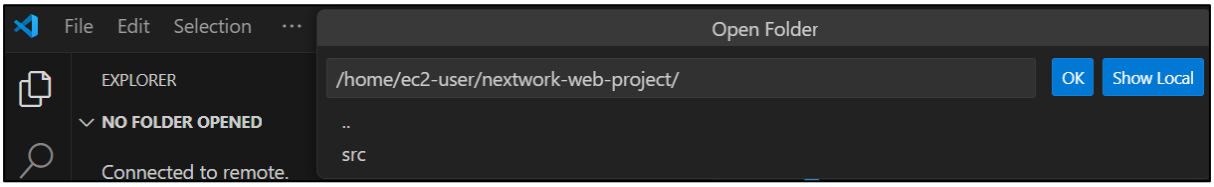
- Select **Linux** and then press **Continue**
- you've connected VSCode with your EC2 instance!



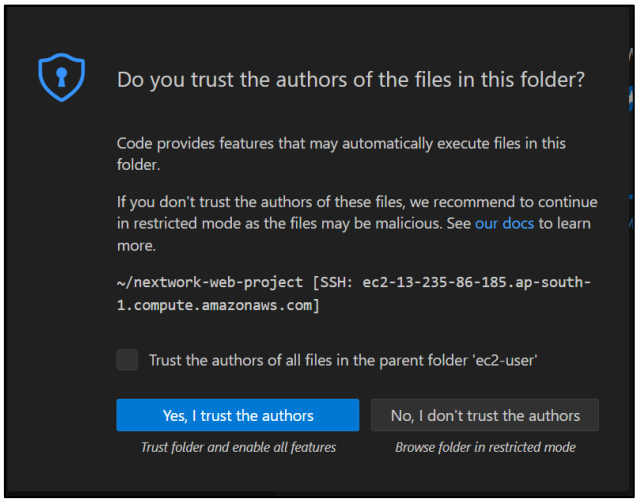
- Now let's open up your web app's files.
- From VSCode's left hand navigation bar, select the **Explorer** icon.



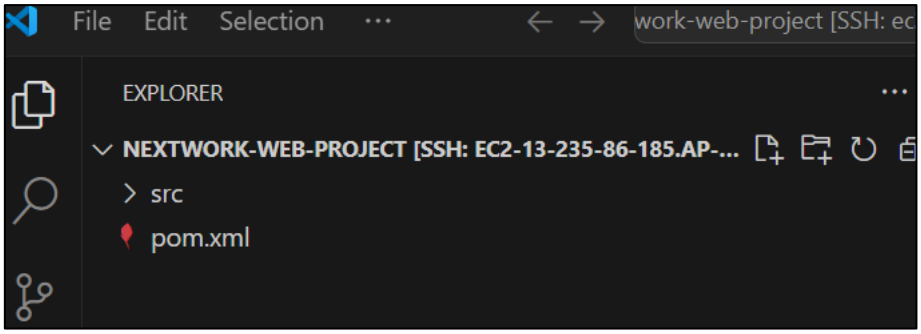
- Select **Open folder**.
- At the top of your VSCode window, you should see a drop down of different file and folder names. Ooooo, this is VSCode asking you which specific file/folder you'd like to open!
- Enter /home/ec2-user/nextwork-web-project.
- Press **OK.**



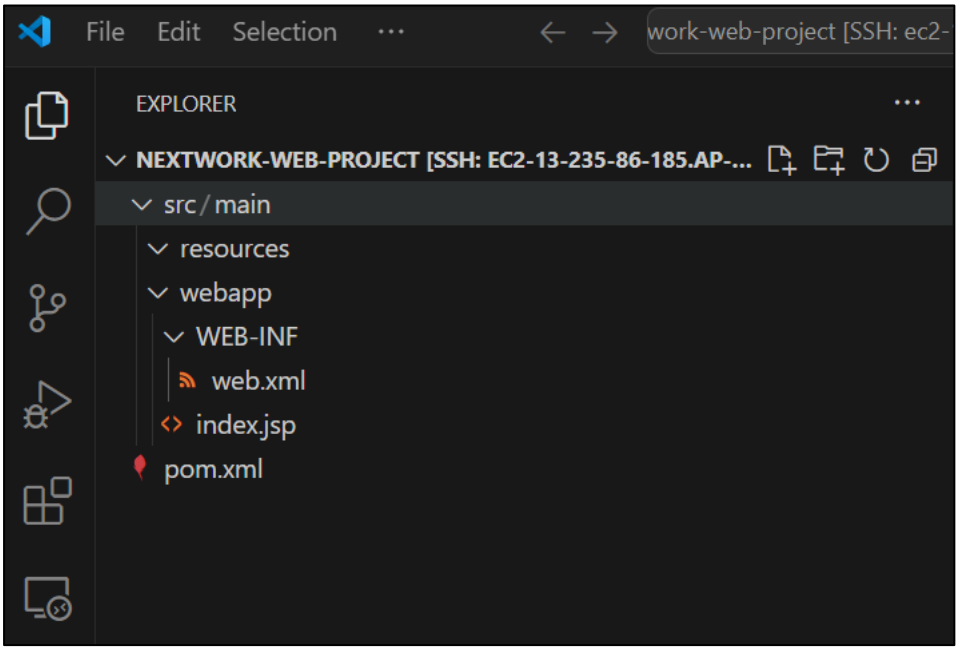
- VSCode might show you a popup asking if you trust the authors of the files in this folder. If you see this popup, select **Yes, I trust the authors.**



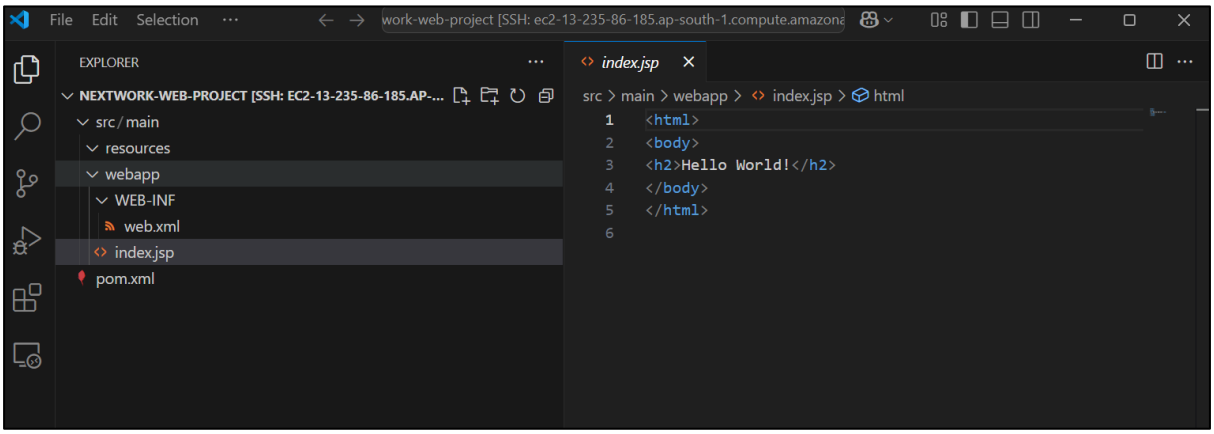
- Check your VSCode window's file explorer again - a folder called **nextwork-web-project** is here!



- Try expanding all the subfolders in the file explorer. All folders have a > icon next to their name.



- From your file explorer, click into **index.jsp**.



- **index.jsp** is a file used in Java web apps. It's similar to an HTML file because it contains markup to display web pages.

However, index.jsp can also include Java code, which lets it generate dynamic content.

This means content can change depending on things like user input or data from a database. Social media apps are great examples of web apps because the content you see is always changing, updating and personalised to you. HTML files are static and can't include Java code. That's why it's so important to install Java in your EC2 instance - so you can run the Java code in your web app!

- Welcome to editor view of index.jsp. Now we're really using VSCode's IDE abilities - editing code is much easier here than in the terminal.
- Let's try modifying **index.jsp** by changing the placeholder code to the code snippet below. Don't forget to replace **{YOUR NAME}** from the following code with your name:

```
<html>

<body>

<h2>Hello {YOUR NAME}!</h2>

<p>This is my NextWork web application working!</p>

</body>

</html>
```

- Save the changes you've made to **index.jsp** by selecting **Command/Ctrl + S** on your keyboard.

Install Git

Now that your development environment is ready, the next step is to set up Git on your EC2 instance.

In this step, you're going to:

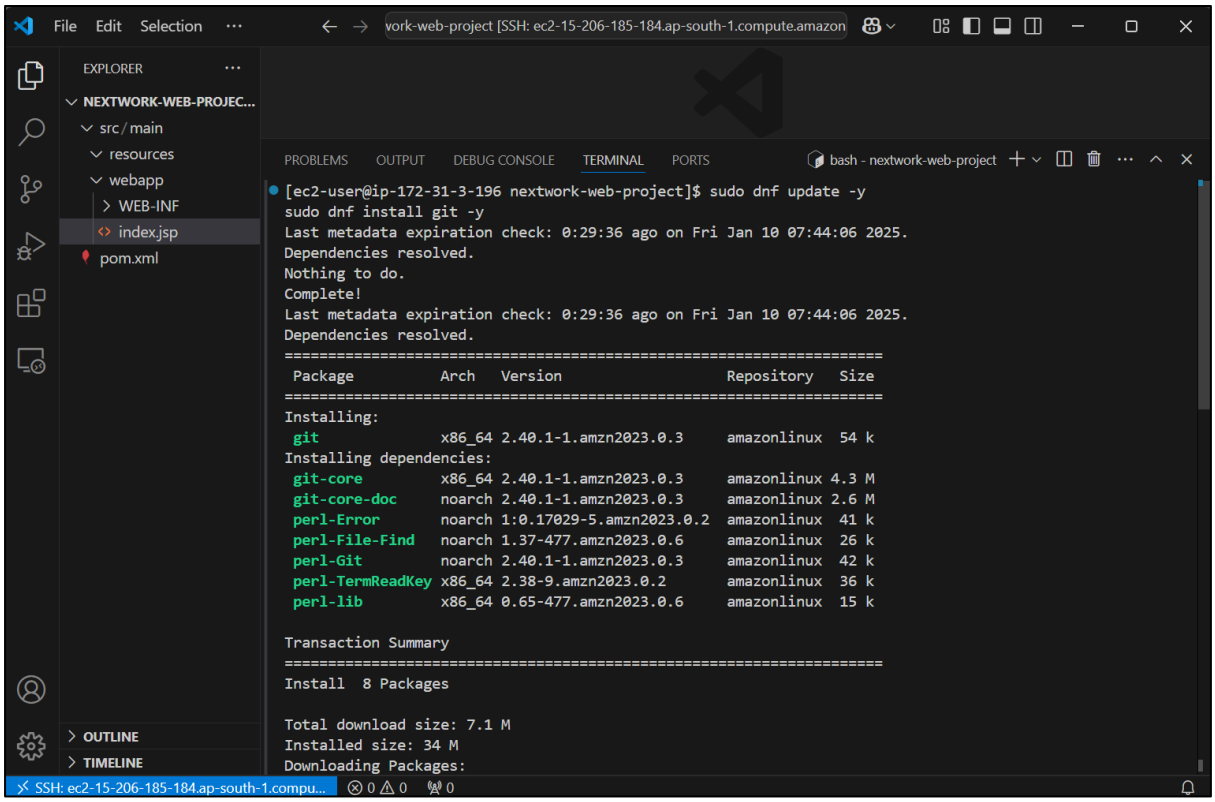
1. Install Git on your EC2 instance.

What is Git?

Git is like a time machine and filing system for your code. It tracks every change you make, which lets you go back to an earlier version of your work if something breaks.
You can also see **who** made specific changes and **when** they were made, which makes teamwork/collaboration a lot easier.
Extra for Experts: Git is often called a **version control system** since it tracks your changes by taking snapshots of what your files look like at specific moments, and each snapshot is considered a 'version'.

- Open your EC2 instance's terminal.
- In the terminal, run these commands to install Git:

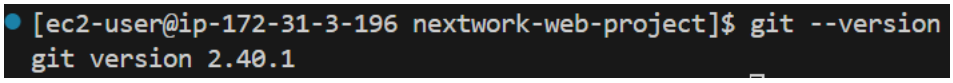
  sudo dnf update -y
  sudo dnf install git -y



What do these commands do?

- sudo dnf update -y tells your EC2 instance to find all the latest updates of software it has (e.g. Java, Maven) and install them straight away.
- sudo dnf install git -y installs Git on your EC2 instance. It's best practise to update your existing software before installing new ones, just in case there are compatibility issues between new and old software.
- **Extra for Experts:** -y is a shortcut for "yes," meaning you're giving your EC2 instance your approval in advance for any time the system might ask questions like "should I proceed with the installation?"

- Verify the installation using the following command: git –version



Set up GitHub

Git is installed

Next up, we'll set you up with GitHub.

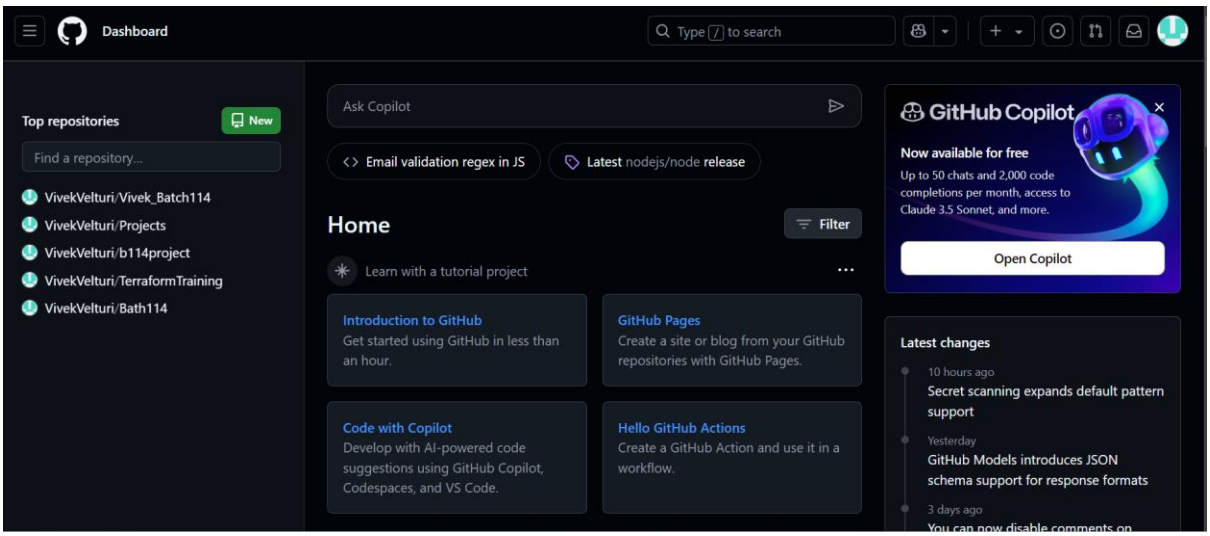
In this step, you're going to:

1. Set up a GitHub account.
2. Create a GitHub repository.

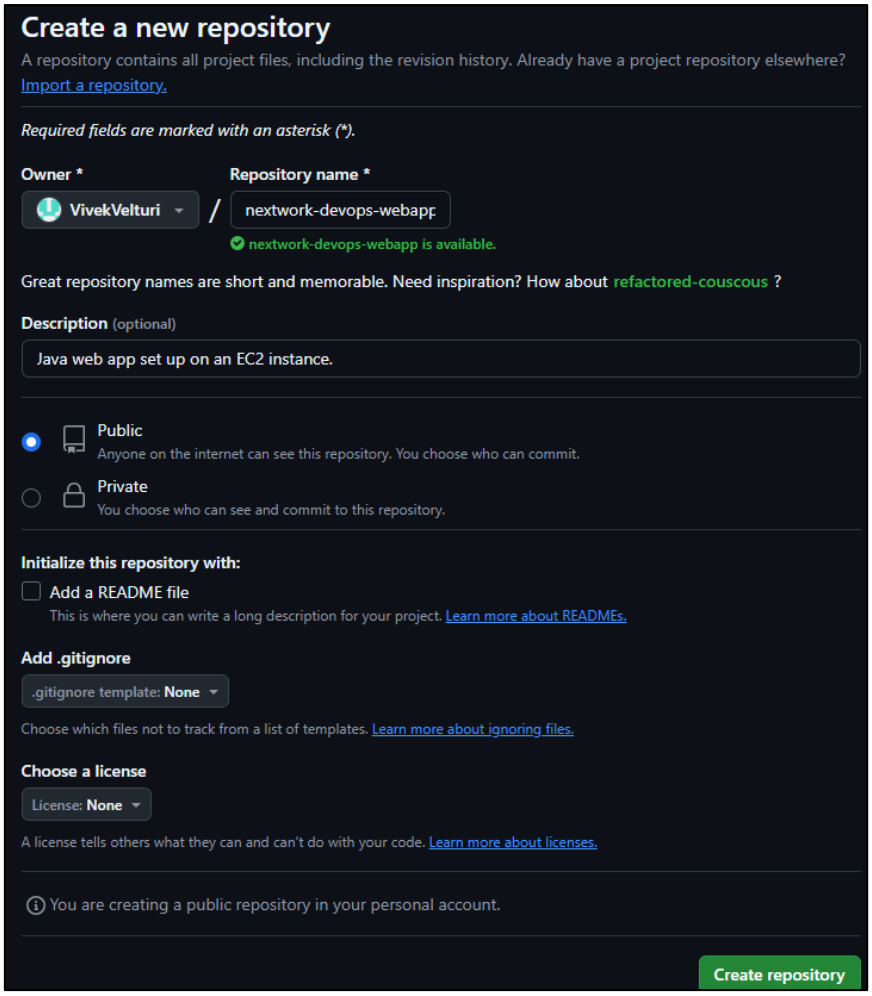
If you already have a GitHub account:

- Sign in to GitHub.
- Scroll to the heading, Set **up a new repository** in this step.
- After signing in to GitHub, click on the **+** icon next to your GitHub profile icon at the top right-hand corner.
- Select **New repository** 

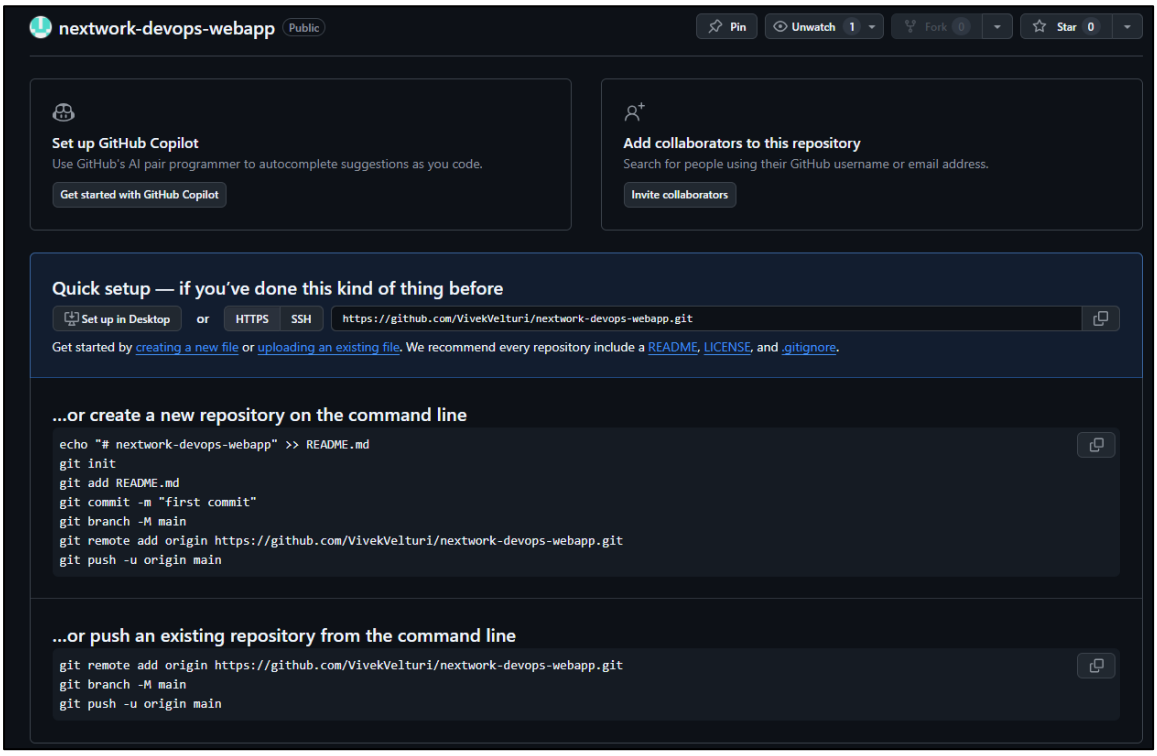
.



- Under **Repository name**, enter nextwork-devops-webapp
- For the **Description**, enter Java web app set up on an EC2 instance.
- Choose your visibility settings. We'd recommend selecting **Public** to make your repository available for the world to see.



- Select **Create repository.**



Commit and Push Your Changes to GithHub

Now we have a place in the cloud that will store our code and track the changes we make.

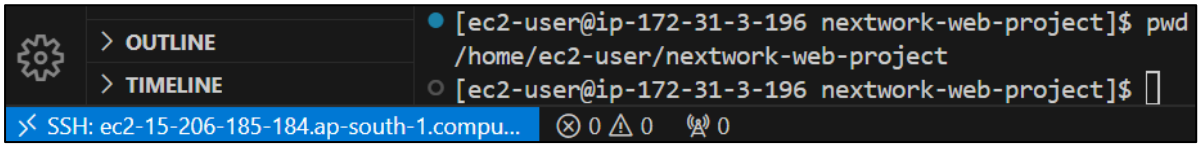
But... this storage folder (your GitHub repo) still doesn't know where your web app files are.

Let's connect our GitHub repo with our web app project stored in your EC2 instance.

In this step, you're going to:

1. Set up a local git repo in your web app folder.
2. Connect your local repo with your GitHub repo.

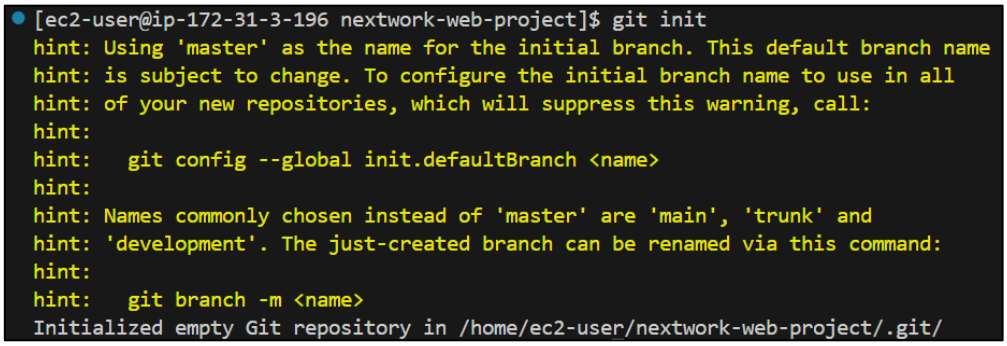
- Head back to your VSCode remote window. Make sure it's still SSH connected to your EC2 instance by checking the bottom left corner.
- Check that you are in the right folder by running this command in your terminal: pwd



What does pwd do?
pwd stands for **print working directory**, and this command asks your server "where am I right now?" The terminal will show you the exact location of the directory (folder) you're in.

- If you're not in nextwork-web-project, use cd to navigate your terminal into your web app project.
- Now let's tell Git that we'd like to track changes made inside this project folder using the following command: git init

What does git init do?
To start using Git for your project, you need to create a **local repository** on your computer. When you run git init inside a directory e.g. nextwork-web-project, it sets up the directory as a local Git repository which means changes are now tracked for version control.
What's a local repository?
The local repository is where you use Git **directly on your own EC2 instance.** The edits you make in your local repo is only visible to you and isn't shared with anyone else yet
This is different to the GitHub repository, which is the remote/cloud version of your repo that others can see.



I got a bunch of yellow text when I ran this command
This yellow text is just Git giving you a heads-up about naming your main branch **master** and suggesting that you can choose a different name like 'main' or 'development' if you want.
What is a main branch?
You can think of Git **branches** as parallel versions or 'alternate universes' of the same project. For example, if you wanted to test a change to your code, you can set up a new branch that lets you diverge from the original/main version of your code (called **master**) so you can experiment with new features or test bug fixes safely. We won't create new branches in this project and we'll save all new changes directly to master, but it's best practise to make all changes in a separate branch and then merge them into master when they're ready.

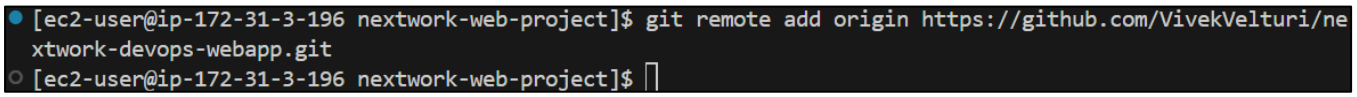
- Head back to your GitHub repository's page.
- In the blue section of the page titled **Quick setup — if you've done this kind of thing before**, copy the HTTPS URL to your repository page. It will look like https://github.com/username/nextwork-devops.git

Now let's connect your local project folder with your Github repo!

- Head back to your terminal in VSCode.
- Run this command. Don't forget to replace **[YOUR GITHUB REPO LINK]** with the link you've just copied.

git remote add origin [YOUR GITHUB REPO LINK]

in my case: git remote add origin https://github.com/VivekVelturi/nextwork-devops-webapp.git



What does 'remote add origin' mean?
Your local and GitHub repositories aren't automatically linked, so you'll need to connect the two so that updates made in your local repo can also reflect in your GitHub repo.
When you set remote add origin, you're telling Git where your GitHub repository is located. Think of **origin** as a bookmark for your GitHub project's URL, so you don't have to type it out every time you want to send your changes there.

Next, we'll save our changes and push them into GitHub.

- Run this command in your terminal:

  git add .

What does this c ommand do?

git add . stages all (marked by the '.') files in nextwork-web-project to be saved in the next version of your project.

What does staging mean?

When you stage changes, you're telling Git to put together all your modified files for a final review before you commit them. This is incredibly handy because you get to see all your edits in one spot, which means its much easier to check if there were are mistakes or unwanted changes before you commit.

- Run this command next in your terminal:

  git commit -m "Updated index.jsp with new content"

```
[ec2-user@ip-172-31-3-196 nextwork-web-project]$ git commit -m "Updated index.jsp with new content"
[master (root-commit) 1bcd97f] Updated index.jsp with new content
 Committer: EC2 Default User <ec2-user@ip-172-31-3-196.ap-south-1.compute.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 3 files changed, 39 insertions(+)
 create mode 100644 pom.xml
 create mode 100644 src/main/webapp/WEB-INF/web.xml
 create mode 100644 src/main/webapp/index.jsp
```

What does this command do?

git commit -m "Updated index.jsp with new content"saves the staged changes as a snapshot in your project's history. This means your project's version control history has just saved your latest changes in a new version. -m flag lets you leave a **message** describing what the commit is about, making it easier to review what changed in this version.

- Finally, run this command:

  git push -u origin master

What does this command do?

git push -u origin master uploads i.e. 'pushes' your committed changes to origin, which you've bookmarked as your GitHub repo. 'master' tells Git that these updates should be pushed to the master branch of your GitHub repo. By using -u you're also setting an 'upstream' for your local branch, which means you're telling Git to remember to push to master by default. Next time, you can simply run git push without needing to define origin and master.

```
[ec2-user@ip-172-31-3-196 nextwork-web-project]$ git push -u origin master
Username for 'https://github.com':
```

- Git can't push your work to the Github repository yet. It's now asking for a username!

Why is Git asking for my username?
Git needs to double check that you have the right to push any changes to the remote origin your local repo is connected with. To do this, Git is now authenticating your identity by asking for your GitHub credentials.

- Enter your Github username, and press **Enter** on your keyboard.
- Next, enter your password. You'll notice that as you type this out, nothing shows on your terminal. This is totally expected - your terminal is hiding your input for your privacy. Press **Enter** on your keyboard when you've typed out your password, even if you don't see it printed out in your terminal.
- Git is letting us know that it can't actually accept our password.

```
Username for 'https://github.com': VivekVelturi
Password for 'https://VivekVelturi@github.com':
remote: Support for password authentication was removed on August 13, 2021.
remote: Please see https://docs.github.com/get-started/getting-started-with-git/about-remote-repositories
#cloning-with-https-urls for information on currently recommended modes of authentication.
fatal: Authentication failed for 'https://github.com/VivekVelturi/nextwork-devops-webapp.git/'
```

What does this mean?
GitHub phased out password authentication to connect with repositories over HTTPS - there are too many security risks and passwords can get intercepted over the internet.You need to use a personal access token instead, which is a more secure method for logging in and interacting with your repos.
What is a token?
A token in GitHub is a unique string of characters that looks like a random password. For example, a GitHub token might look like **ghp_xHJNmL16GHSZSV88hjP5bQ24PRTg2s3Xk9ll**. As you can imagine, tokens are great for security because they're unique to every token and would be very hard to guess.

Set up a GitHub token

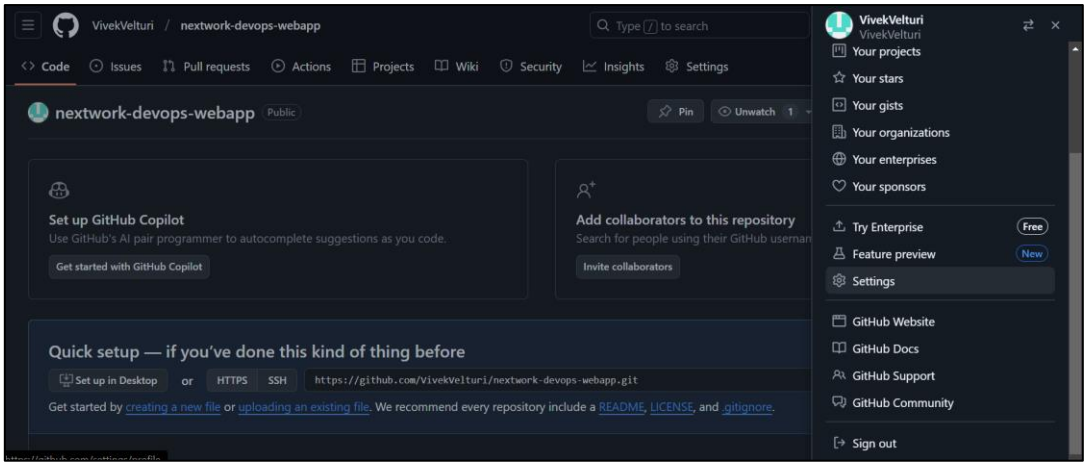
Now that we know passwords won't work for authentication, we'll have to find a replacement.

Let's generate an authentication token on GitHub!

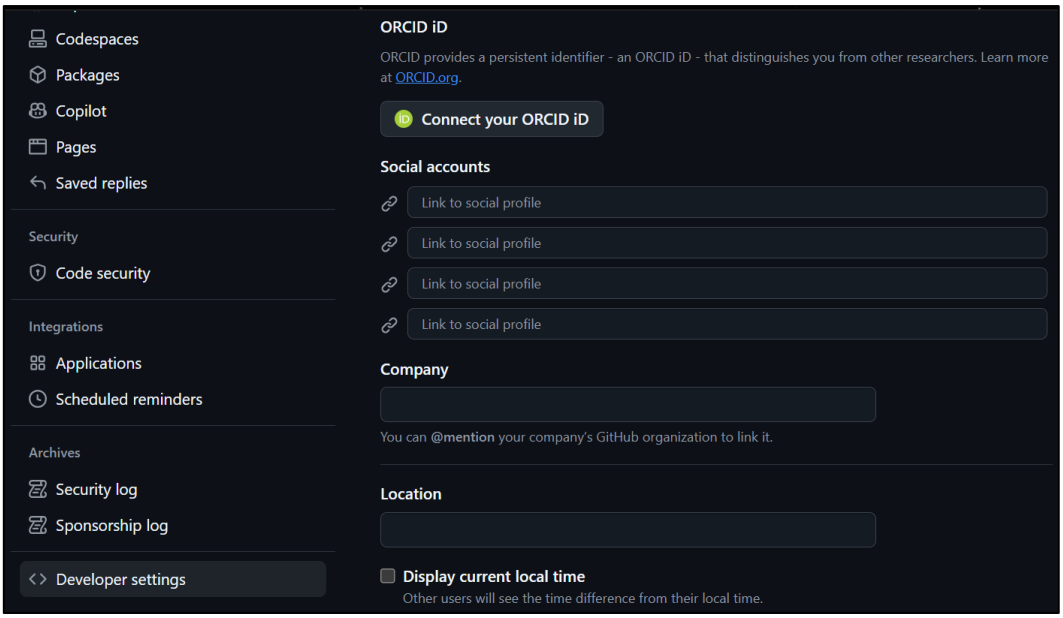
In this step, you're going to:

1. Set up a token on GitHub.
2. Use the generated token to access your GitHub repo from your local repo.

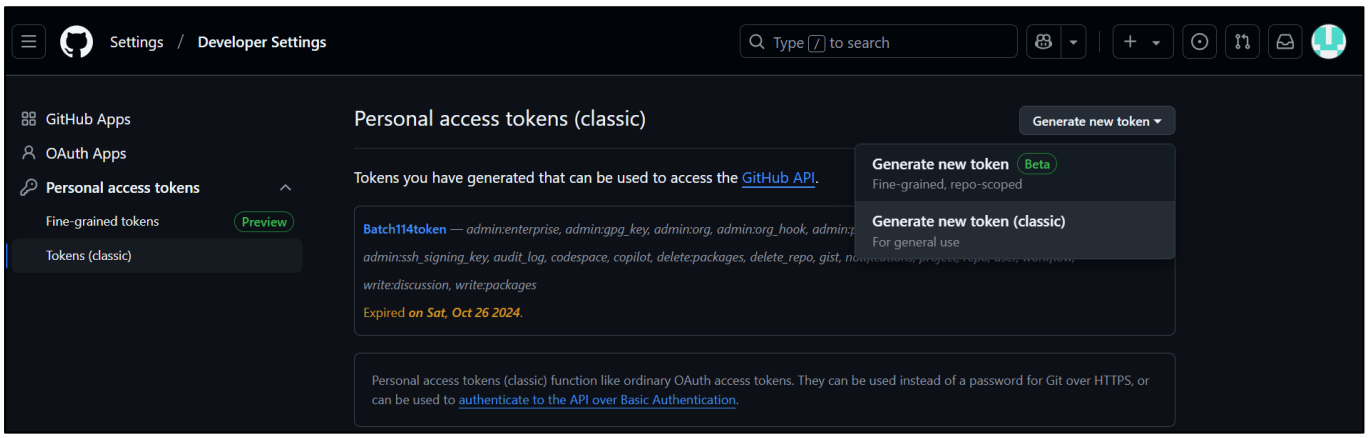
- Head back into your browser with GitHub open.
- Select your profile icon and select **Settings**



- Select **Developer settings** at the very bottom of the left hand navigation panel.



- Select **Personal access tokens**.
- Select **Tokens (classic)**.
- Select **Generate new token**.
- Select **Generate new token (classic)**.

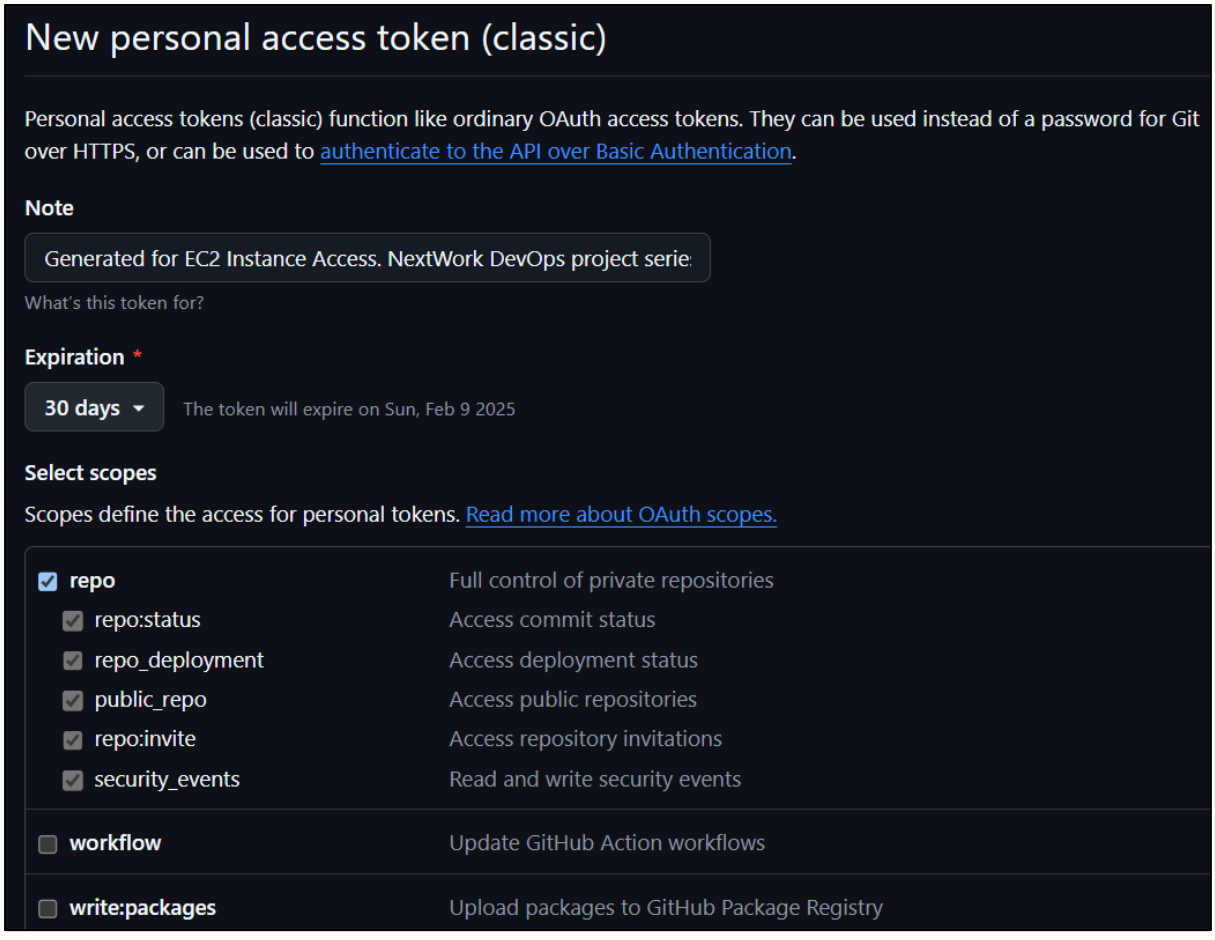


- If prompted for providing password, provide you GithHub Password.
- Give your token a descriptive note, like Generated for EC2 Instance Access. NextWork DevOps project series.
- Keep **30 days** as the token expiration limit.

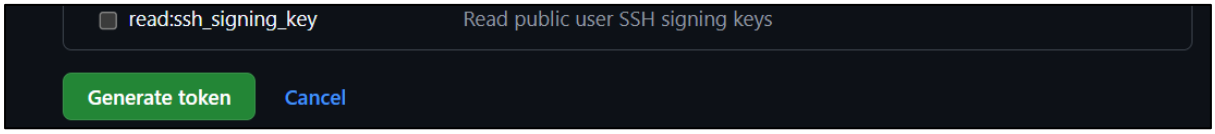
What is a token expiration limit?
A token expiration limit is how long your personal access token would work for. After this time period, the token expires and no longer grants access, so you'll need to generate a new token.

- Select the checkbox next to **repo**.

What do all these scopes mean?
We use **scopes** to decide what kind of permissions your token will grant. Each scope you pick gives the token the ability to do even more things with your GitHub account. In our case, we picked the **repo** scope, which means the token can even access and control private repositories in your account.



- Scroll Down and Select **Generate token.**



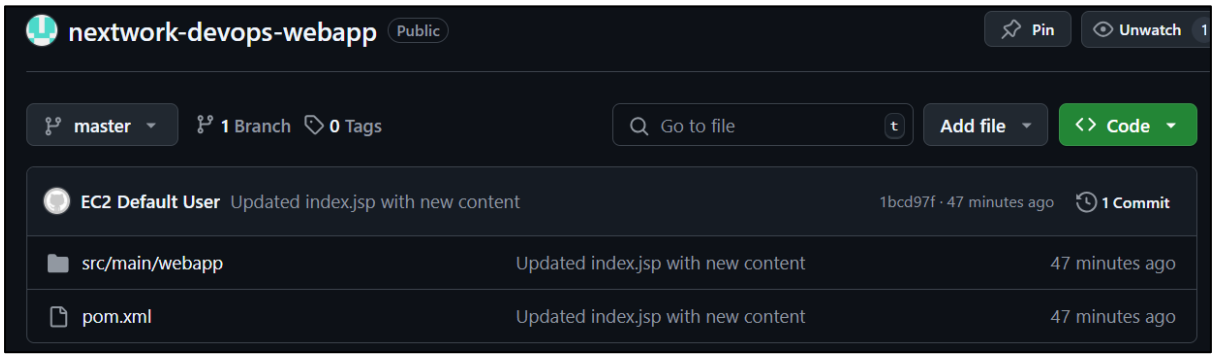
- Nice, a new token (a long string of random letters) is generated! **(not uploading the screenshot due to privacy)**
- **Make sure to copy your token now. Keep it safe somewhere else, you won't be able to see your token once you close this tab.**
- Switch back to your VS Code terminal.
- Run git push -u origin master again, which will trigger Git to ask for your GitHub username.
- Enter your username again.
- When Git asks for your password, **paste in your token instead.** Your terminal won't show your password for privacy reasons, so press **Enter** on your keyboard once you've pasted your token (even if you don't see it on screen)

```
[ec2-user@ip-172-31-3-196 nextwork-web-project]$ git push -u origin master
Username for 'https://github.com': VivekVelturi
Password for 'https://VivekVelturi@github.com':
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (9/9), 1.07 KiB | 1.07 MiB/s, done.
Total 9 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/VivekVelturi/nextwork-devops-webapp.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
```

- Looks like Github recognises your token and pushed your changes to your repository.

What does the message in the terminal mean?
This message appears when you successfully push changes to a GitHub repository - nice work. It shows the progress of transferring objects (like files and commits), how many objects were processed, and tells you that your local branch is now tracking the remote branch after the push.

- Head to your GitHub repository in your web browser.
- Refresh the page, and you'll see your web app files in the repository, along with the commit message you wrote.



When you add, commit and push your changes, you might notice the terminal automatically sets two other things - your name and email address - before it asks for your GitHub username.

- Run git log to see your history of commits, which also mentions the commit author's name.

```
[ec2-user@ip-172-31-3-196 nextwork-web-project]$ git log
commit 1bcd97f8e7f6229824d1043413c4023ec495e2bc (HEAD -> master, origin/master)
Author: EC2 Default User <ec2-user@ip-172-31-3-196.ap-south-1.compute.internal>
Date:   Fri Jan 10 08:51:11 2025 +0000

    Updated index.jsp with new content
```

- **EC2 Default User** isn't really your name, and the EC2 instance's IPv4 DNS is not your email. Let's configure your local Git identity so Git isn't using these default values.
- Run these commands in your terminal to manually set your name and email. Don't forget to replace **"Your name"** with your name (keep the quote marks), and **you@nextwork.org** with your email address.

  git config --global user.name "Your Name"
  in my case git config --global user.name "VivekVelturi"
  git config --global user.email you@nextwork.org
  in my case git config --global user.email vivek.velturi@gmail.com

```
[ec2-user@ip-172-31-3-196 nextwork-web-project]$ git config --global user.name "VivekVelturi"
[ec2-user@ip-172-31-3-196 nextwork-web-project]$ git config --global user.email vivek.velturi@gmail.com
[ec2-user@ip-172-31-3-196 nextwork-web-project]$
```

You've set up your local Git identity, which means Git can associate your changes in the local repo to your name and email.

This setup is best practice for keeping a clear history of who made which changes ✅

Your Second Commit

Great sucess with getting your GitHub connection all set up.

we've learnt how to link your EC2 instance's files with a cloud repo, now let's see what happens when you make **new** changes to your web app files.

In this step, we'll edit index.jsp again using VSCode, and run commands that pushes those changes to your GitHub repository too.

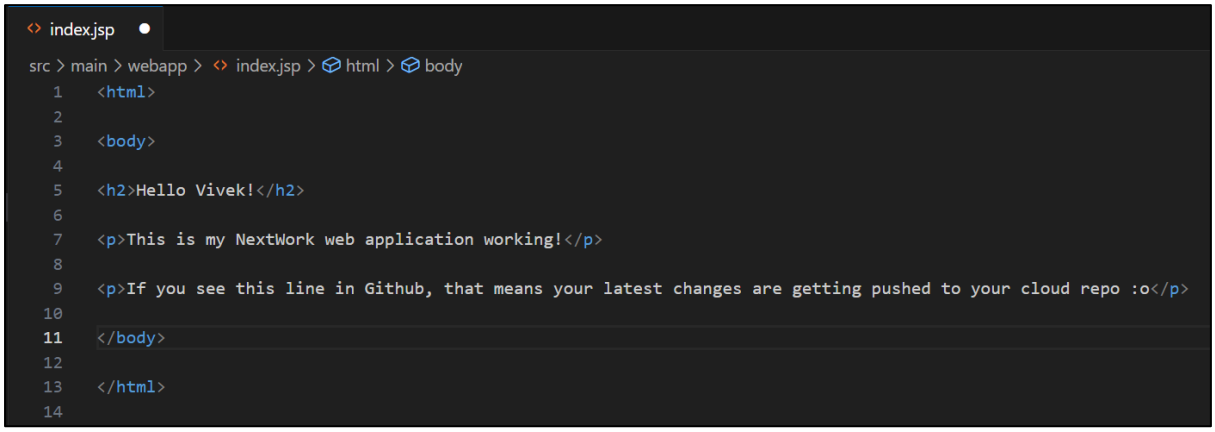
In this step, you're going to:

1. Make changes to your web app.
2. Commit and push those changes.

- Keep your GitHub page open, and switch back to your EC2 instance's VSCode window.
- Find **index.jsp** in your file navigator on the left hand panel.

- Find the line that says **This is my NextWork web application working!** and add this line below:

  If you see this line in Github, that means your latest changes are getting pushed to your cloud repo :o



- Save your changes by pressing Command/Ctrl + S on your keyboard while keeping your index.jsp editor open.
- Head back to your GitHub tab and click into the **src/main/webapp** folders to find index.jsp.
- Click into **index.jsp** - have there been any updates to index.jsp in your GitHub repo?

it's still looking the same...

You won't see your changes in GitHub yet, because saving changes in your VSCode environment only updates your **local** repository. Remember that the local repository in VSCode is separate from your GitHub repository in the cloud.

To make your changes visible in GitHub, you need to write commands that send (push) them from your local repository into your origin.

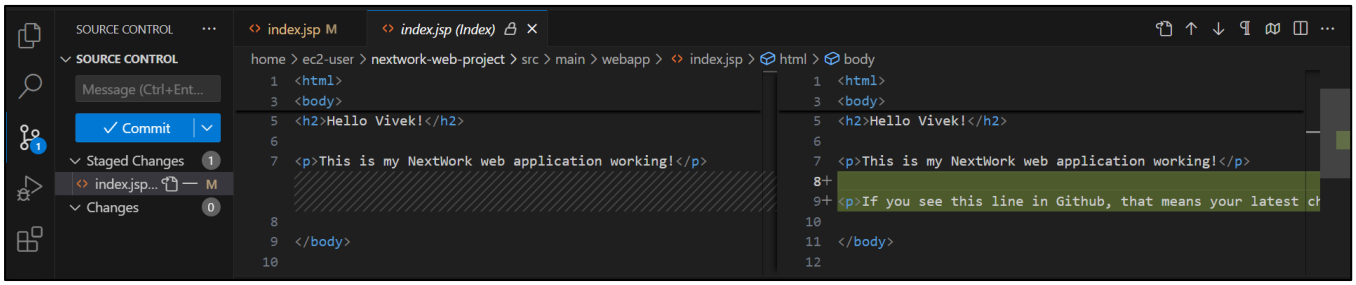
- Head back to your VSCode window.
- In the terminal, let's stage our changes: git add .
- Ready to see what changes are staged? Run git diff --staged next.

```
diff --git a/src/main/webapp/index.jsp b/src/main/webapp/index.jsp
index 4f61d04..6d4972f 100644
--- a/src/main/webapp/index.jsp
+++ b/src/main/webapp/index.jsp
@@ -6,6 +6,8 @@

 <p>This is my NextWork web application working!</p>

+<p>If you see this line in Github, that means your latest changes are getting pushed to your cloud repo :o</p>
+
 </body>

 </html>
~
~
~
~
```

What does this command do?
git diff --staged shows you the exact changes that have been staged compared to the last commit. Now you get to review your modifications in your code that you are about to save into your local repo's version history!

Did you know you can view these changes using VSCode too?

- Select the **Source Control** icon on the side of your VSCode window.
- Under the **Saved Changes** heading, select index.jsp.
- You'll see your change in a new window that hightlights the new line you've added to index.jsp!



- Nice, these changes are what we want to save and send to GitHub, lets do that with these commands:

  git commit -m "Add new line to index.jsp"

  git push

  Note: If terminal isn't letting you to enter commands!
  (This doesn't apply to everyone,but) your terminal might stay stuck in the previous step to show you your staged changes. Enter q into the terminal to quit this view and return to running commands.

```
[ec2-user@ip-172-31-3-196 nextwork-web-project]$ git commit -m "Add new line to index.jsp"
[master 7aa9447] Add new line to index.jsp
 1 file changed, 2 insertions(+)
```

```
[ec2-user@ip-172-31-3-196 nextwork-web-project]$ git push
Username for 'https://github.com':
```

- We need to enter your username and token again to complete your push.

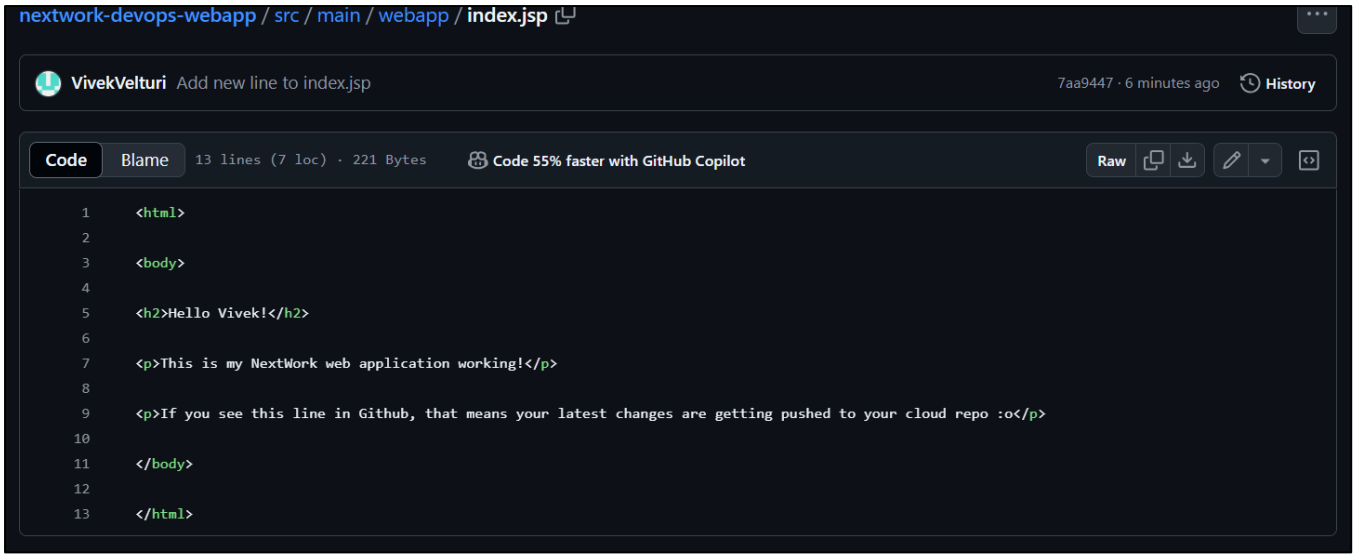
```
[ec2-user@ip-172-31-3-196 nextwork-web-project]$ git push
Username for 'https://github.com': VivekVelturi
Password for 'https://VivekVelturi@github.com':
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 606 bytes | 606.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/VivekVelturi/nextwork-devops-webapp.git
   1bcd97f..7aa9447  master -> master
```

I have to enter my username and password AGAIN?!
This is because we set up the connection to our GitHub repo using its HTTPS URL. Using HTTPS is straightforward compared to other options e.g. SSH, but it's also **stateless**, which means Git doesn't remember your credentials for security reasons. That's why Git asks for your GitHub credentials **every time** you pull from or push to your GitHub repo.

Tip: right after you enter your username and token again, you can run git config --global credential.helper store to ask Git to remember your credential details for next time!

```
[ec2-user@ip-172-31-3-196 nextwork-web-project]$ git config --global credential.helper store
[ec2-user@ip-172-31-3-196 nextwork-web-project]$
```

- Head back to your GitHub tab and refresh it - you will see your changes now.



Delete Your Resources

Delete your EC2 instance:

- In your AWS Management Console, head to **Amazon EC2** to delete your EC2 instance.
- Select **Instances** from the left hand navigation panel. Select the checkbox next to your instance.
- Click on the **Instance state** dropdown and select **Terminate (delete) instance.**
- Choose **Terminate (delete)**.

Delete your key pair:

Note: If you plan to continue this DevOps series (totally recommend!), you can keep your key pair and use the same credentials for the rest of this series. AWS does not charge for key pairs. Store your key pair in a safe place.

- Still in your EC2 console, select **Key Pair** from the left hand navigation panel.
- Select the checkbox next to your key pair.
- Select the **Actions** dropdown, and then **Delete.**
- Type Delete in the text field, and select **Delete.**

What about VSCode and GitHub?
VSCode and GitHub are absolutely free to install and use, we'd recommend keeping both. They're great tools for future projects and you'll need them again for the rest of this DevOps series.