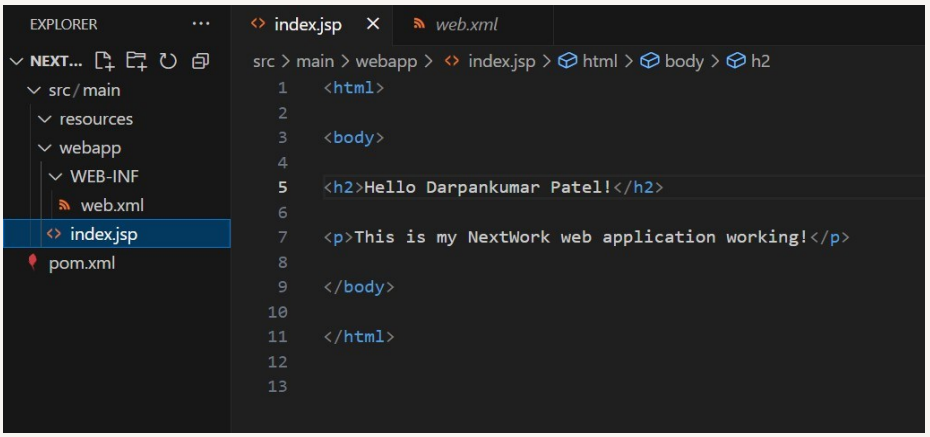
Set Up a Web App Using AWS and VS Code



Introducing Today's Project!

Today, I’m starting Project 1 of the 7 Day DevOps Challenge, where I’ll be setting the foundation for a CI/CD pipeline by launching and running a simple web app in the cloud. What I’m Trying to Do Today In this project, I’m going to: Launch an EC2 instance (a virtual server) on AWS. Connect to it remotely using VS Code and SSH. Install Java and Maven, which are necessary for building Java-based applications. Generate a basic web application using Maven. Run the app on the EC2 server and access it from my browser. By the end of today’s project, I’ll have a working cloudhosted web app running on an Amazon EC2 instance. This is the first step toward building an automated DevOps pipeline. What I Need Before Starting An AWS account
Visual Studio Code with the Remote - SSH extension A stable internet connection Around 2 hours of focused time Let’s get started with Step 1: Launching the EC2 instance!

Key tools and concepts

Services I used were Amazon EC2 for hosting the web app, VS Code with Remote SSH for remote editing, and Linux terminal tools like `nano`, `cd`, `ls`, and `pwd` for navigating and editing files directly on the server. Key concepts I learnt include SSH connectivity, how to deploy and manage a web app on a remote server, the structure of a Java web application (including `index.jsp`, `src`, and `webapp` folders), and the differences between editing code with an IDE vs a terminal-based text editor like `nano`.

Project reflection

One thing I didn't expect in this project was how seamless it would be to see real-time changes across both the terminal and VS Code using an SSH connection—it was surprising and satisfying to watch updates made in `nano` appear instantly in the VS Code editor.

This project took me approximately 2 hours to complete. The most challenging part was configuring the Remote-SSH connection to the EC2 instance and making sure all Git commands were correctly reflected in my GitHub repository.

It was most rewarding to see the changes instantly reflected across both the terminal and VS Code, and to realize how powerful SSH connections are for remote development.

This project is part one of a series of DevOps projects where I'm building a CI/CD pipeline! I'll be working on the next project in the coming days to keep up the momentum and deepen my understanding of continuous integration and deployment using tools like GitHub Actions, Jenkins, and AWS services. I'm excited to continue learning how these pieces fit together in a real-world workflow.

Launching an EC2 instance

I started this project by launching an EC2 instance because it provides a flexible and scalable virtual server in the cloud, which is ideal for hosting my web app. With EC2, I can easily customize the instance's CPU, memory, storage, and network settings to match the requirements of my project. This allows me to have a secure and powerful environment for developing and deploying my web app without the need for physical hardware.

I also enabled SSH

SSH, or Secure Shell, is a secure protocol used to access and control remote servers. It verifies that you have the correct private key to match the public key stored on your EC2 instance, ensuring only authorized users can connect. Once verified, SSH creates an encrypted connection between your computer and the EC2 server, keeping all commands and data transfers secure and private.

Key pairs

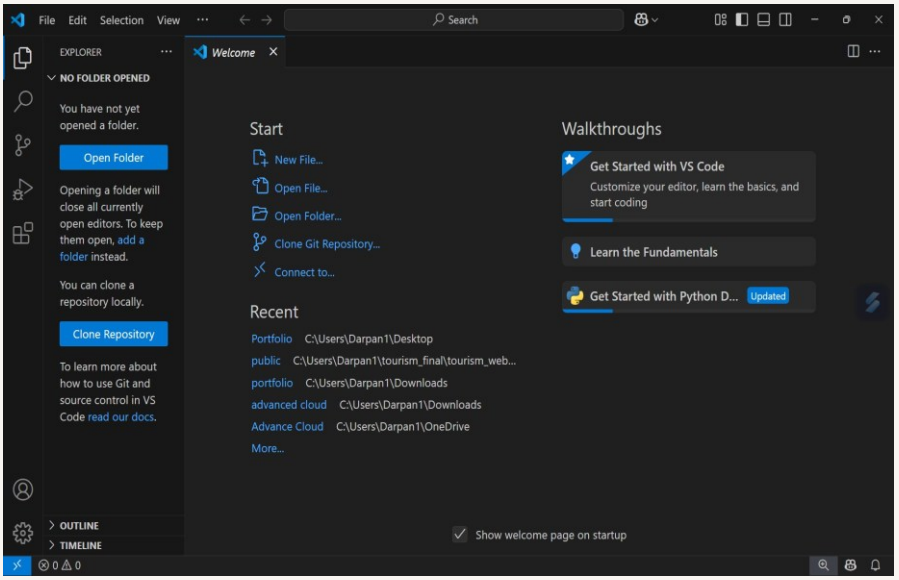
I created a key pair for secure access to my EC2 instance. The key pair consists of a public key, stored by AWS, and a private key, which I download. This private key, saved in .pem format, ensures that only I can securely access my EC2 instance. I named the key pair "nextwork-keypair" and used RSA encryption, which is a trusted and widely supported cryptographic algorithm. After generating the key pair, I saved the private key file in a "DevOps" folder on my desktop for easy access. This will be used later when I establish a secure SSH connection to my EC2 instance.

Once I set up my key pair, AWS automatically downloaded a private key file in .pem format to my local computer. This file is crucial for securely accessing my EC2 instance via SSH. I moved the file from my Downloads folder into a "DevOps" folder on my desktop to keep it organized and easily accessible for future use.

Set up VS Code

VS Code is a lightweight yet powerful source-code editor developed by Microsoft. It supports development in many programming languages like Java, Python, and JavaScript, and includes features like debugging, Git integration, and extensions. In this project, VS Code is used to securely connect to your EC2 instance, write and edit code, and run commands directly on the remote server via its integrated terminal.

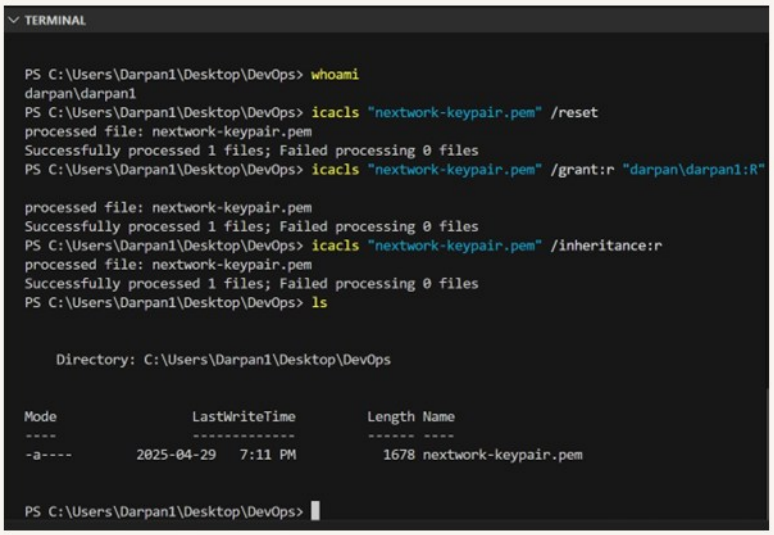
I installed VS Code to connect securely to my EC2 instance, write and edit the web app’s code, and run terminal commands directly on the server. This allows me to manage the development environment from my local computer while working entirely in the cloud.



My first terminal commands

A terminal is a text-based interface that lets you interact directly with your computer's operating system by typing commands. Instead of clicking through menus, you can give your computer precise instructions quickly and efficiently. The first command I ran for this project is: cd ~/Desktop/DevOps This command moved me into the **DevOps** folder on my desktop, where my `.pem` key file is stored. This step is essential for securely connecting to my EC2 instance using SSH.

I also updated my private key's permissions by using the icacls command in PowerShell since chmod isn’t supported on Windows. I ran the following commands: icacls "nextwork-keypair.pem" /reset icacls "nextwork-keypair.pem" /grant:r "darpan\darpan1:R" icacls "nextwork-keypair.pem" /inheritance:r icacls "nextworkkeypair.pem" /inheritance:r These commands reset the file's permissions, granted read-only access to my user account, and removed inherited permissions. This ensures that only I can access the .pem file, keeping the private key secure for connecting to my EC2 instance via SSH.

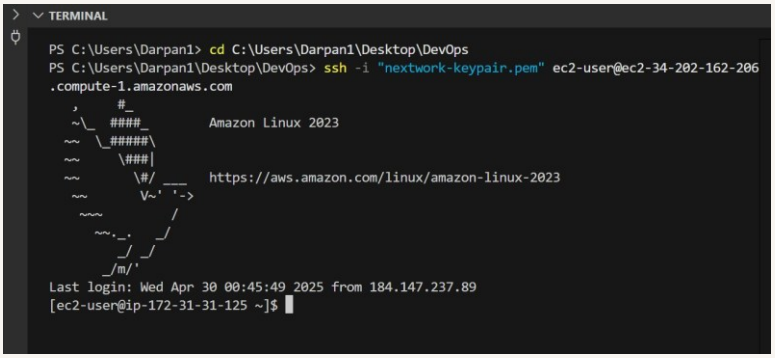


SSH connection to EC2 instance

To connect to my EC2 instance, I ran the command: ssh -i "nextwork-keypair.pem" ec2-user@ec2-34-202-162-206.compute-1.amazonaws.com I replaced [PATH TO YOUR .PEM FILE] with the path to my private key (C:\Users\Darpan1\Desktop\DevOps\nextwork-keypair.pem) and [YOUR PUBLIC IPV4 DNS] with my instance's Public DNS (ec2-34-202-162-206.compute-1.amazonaws.com). After running the command, my terminal prompted me to confirm if I trusted the server, showing a unique "fingerprint" code. I typed yes to continue. Once connected, I saw my terminal prompt change to something like ec2-user@ipxxx-xxx-xxx-xxx, confirming that I was successfully connected to my EC2 instance via SSH!

This command required an IPv4 address

A server's IPV4 DNS is the publicly accessible domain name that maps to the server's IP address, allowing other systems—like your local machine—to connect to it over the internet. In the context of AWS EC2, it's automatically generated and looks something like `ec2-34-202-162-206.compute-1.amazonaws.com`. You use this DNS when establishing an SSH connection to the server.



Maven & Java

Apache Maven is a build automation tool primarily used for Java projects. It simplifies the process of building, packaging, and managing dependencies for Java applications. Maven helps developers by automating tasks like compiling code, running tests, generating documentation, and handling libraries or external dependencies. It also provides a project structure and standardizes how projects are built, making it easier to work across different teams and environments. With Maven, you can focus more on developing the application rather than manually managing these repetitive tasks.

Maven is required in this project because it helps streamline the process of building and managing the Java-based web application. It automates tasks like dependency management, compilation, testing, and packaging, ensuring that the project is organized and efficient. Additionally, Maven provides a standardized project structure, making it easier to set up and maintain the web app's foundational components. By using Maven, we can focus on writing the application code while Maven handles the underlying configuration and dependency management, saving time and reducing the risk of errors.

Java is a widely used programming language and platform that allows developers to build various types of applications, including web apps, mobile apps, enterprise

systems, and more. It is known for its platform independence, meaning that applications built in Java can run on any system that supports the Java Virtual Machine (JVM), making it highly versatile and popular for large-scale applications. Java also provides a robust set of libraries, tools, and frameworks that help developers create scalable and maintainable software.

Java is required in this project because Apache Maven, the tool we're using to build and organize our web app, depends on Java to function. Maven uses Java to compile code, manage dependencies, and execute various tasks during the build process.

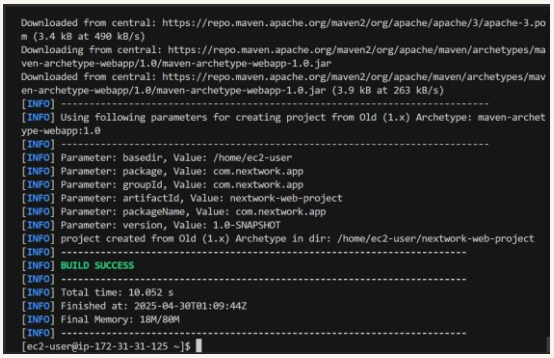
Since I am building a Java -based web app, Java is essential for running Maven commands and for the app's execution. Additionally, Java provides the platform and libraries necessary to create a scalable, efficient web application.

Create the Application

I generated a Java web app using the command: mvn archetype:generate \ DgroupId=com.nextwork.app \ -DartifactId=nextwork-web-project \ -DarchetypeArtifactId=maven-archetype-webapp \ -DinteractiveMode=false This command told Maven to generate a new web application using the maven-archetypewebapp template. By specifying the groupId and artifactId, I created a unique identity for the project (com.nextwork.app and nextwork-web-project). The -DinteractiveMode=false part ensured that the process ran automatically without prompting me for input, which streamlined the setup. The result is a basic Java web app structure, ready for development!

I installed Remote - SSH, which is an extension in VS Code that allows me to securely connect to remote servers over SSH. I installed it to enable seamless interaction with my EC2 instance directly from VS Code, so I can edit and manage my Java web app's files as if they were on my local machine. This will make development and file navigation much more efficient, leveraging VS Code's IDE features while working on the EC2 instance.

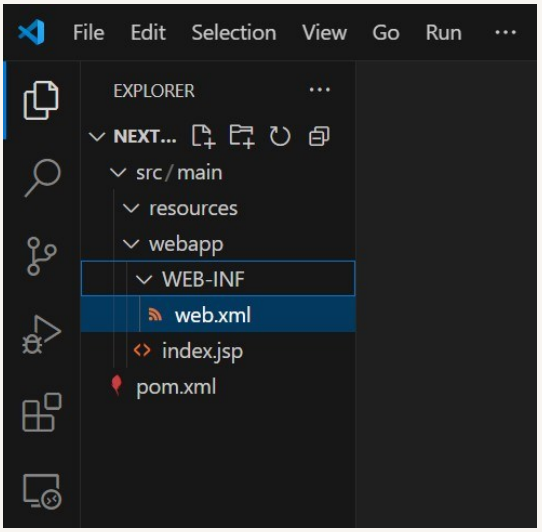
Configuration details required to set up a remote connection include the following: 1. Host: The Public IPv4 DNS of the EC2 instance, which specifies the remote server (e.g., `ec2-34-202-162-206.compute-1.amazonaws.com`). 2. IdentityFile: The path to the private key file used for SSH authentication, such as `/path/to/nextworkkeypair.pem`. 3. User: The username used for logging into the EC2 instance, typically `ec2-user` for Amazon Linux instances. These details ensure that VS Code connects to the correct EC2 instance and uses the appropriate credentials for a secure connection.



Create the Application

Using VS Code's file explorer, I could see the full directory structure of the `nextworkweb-project` folder on the EC2 instance. This included key components such as: The `src` directory, which holds the core source code of the web application. - Inside `src/main/webapp`, I could see web-facing files like HTML, CSS, JavaScript, and JSP that define the user interface and client-side behavior. - Inside `src/main/resources`, I found configuration files typically used for application settings and resource mapping. - The `pom.xml` file, which is the Maven Project Object Model file. It defines project dependencies, build configurations, and plugins that Maven uses to compile and manage the web application. This structure allowed me to easily navigate and start editing the web project directly from within VS Code.

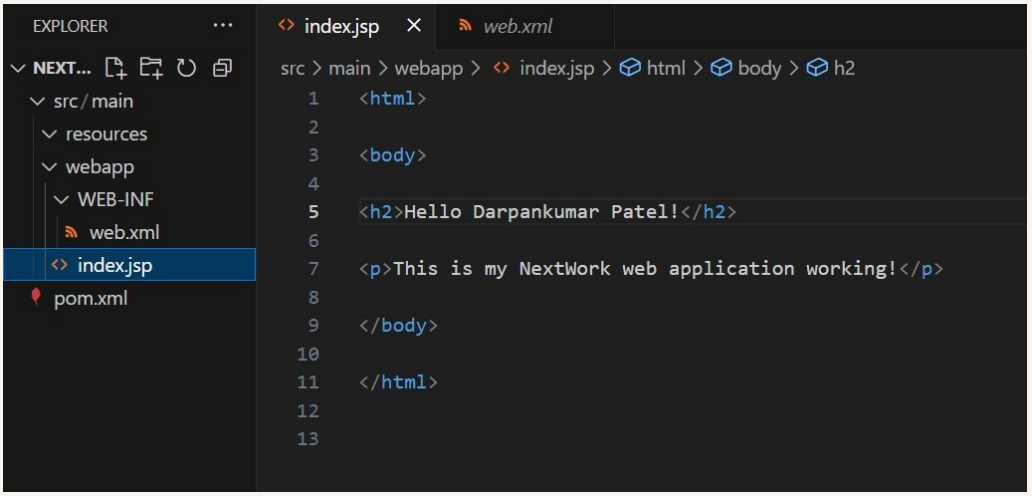
Two of the project folders created by Maven are src and webapp, which serve key roles in organizing the web application's source code and front-end components: The `src` (short for source) folder is the main directory for all source code and resources of the project. It typically follows a standard structure like `src/main/java` for Java classes, `src/main/resources` for configuration files, and `src/main/webapp` for web content. - The `webapp` folder, located inside `src/main`, contains all the webfacing files—such as HTML, CSS, JavaScript, and JSP (JavaServer Pages)—that make up the front end of the web application. These files define what users see and interact with in their browsers. Together, these folders ensure the project is neatly organized and ready for building and deployment with Maven.



Using Remote - SSH

index.jsp is a Java Server Page file used in Java web applications to create dynamic web content. Unlike a standard HTML file, which only displays static content, `index.jsp` can embed Java code directly within the HTML to generate content that changes based on user input, server data, or other dynamic factors. This allows for more interactive and personalized web experiences, making it a powerful tool for building modern web apps.

I edited index.jsp by opening the file in VS Code's editor view, replacing the placeholder content with a custom HTML snippet that included my name, and then saving the changes using Ctrl + S (or Command + S on Mac). The new content displayed a personalized greeting and a message confirming that the NextWork web application is working.



Using nano

Compared to using an IDE, editing 'index jsp in the terminal felt more manual and less visually intuitive. Without features like syntax highlighting, autocomplete, or a file tree, it was harder to navigate and catch errors quickly. I'd be more likely to use an IDE if I were working on a larger project, needed to debug code, or wanted a more user-friendly environment for frequent editing and testing.