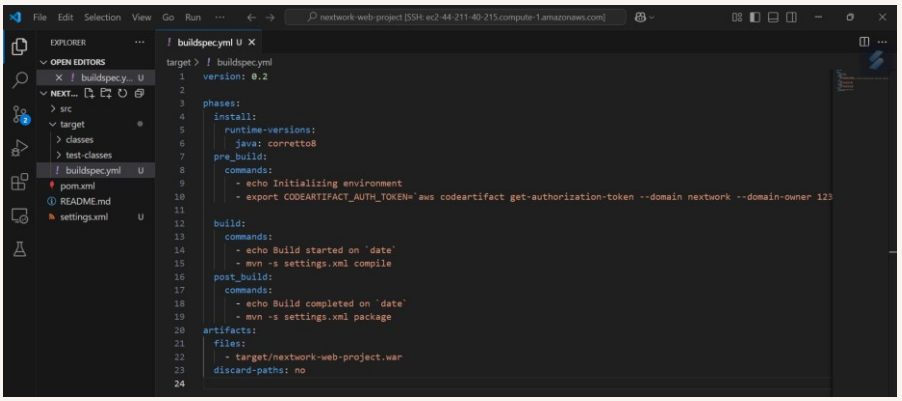
Darpan Patel

Continuous Integration with CodeBuild



Darpan Patel

Introducing Today's Project!

In this project, I will demonstrate how to set up Continuous Integration using AWS CodeBuild. I'm doing this project to learn how to automate the build and test process for my application every time code is pushed to a repository. This includes Creating and configuring a CodeBuild project from scratch. Connecting CodeBuild to a GitHub repository. Writing a `buildspec.yml` file to define the build and test steps. Running automated tests as part of the CI workflow. By doing this, I'll gain hands-on experience with one of the key components of a CI/CD pipeline, which is essential for delivering high-quality software faster and more reliably.

Key tools and concepts

Services I used were AWS CodeBuild for continuous integration and automated testing, Amazon S3 for artifact storage, and GitHub for source code version control and triggering builds. Key concepts I learnt include the importance of automating testing within a CI/CD pipeline to catch errors early, how to write and integrate simple test scripts into build processes, managing buildspec.yml to control build stages and commands, using exit codes to control build success or failure, and how continuous integration helps maintain code quality and streamline deployments.

Project reflection

This project took me approximately 4 to 6 hours to complete. The most challenging part was correctly integrating the test script into the buildspec.yml file and troubleshooting permission issues to make the script executable in the build environment. It was most rewarding to see the automated tests run

successfully within CodeBuild, giving me confidence that my CI pipeline was robust and reliable.

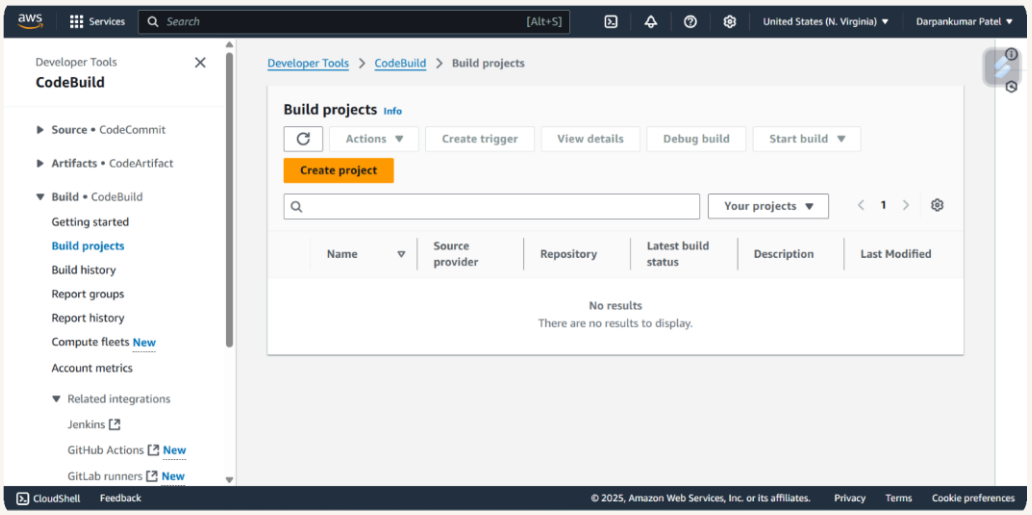
This project is part four of a series of DevOps projects where I'm building a CI/CD pipeline! I'll be working on the next project within the next few days to keep the momentum going, continuing to deepen my skills in deployment automation and infrastructure as code. Staying consistent helps me build a strong, practical foundation

Setting up a CodeBuild Project

CodeBuild is a continuous integration service, which means it automatically builds and tests your code every time you make a change or push it to your version control system (like GitHub). It helps ensure that your application is always working as expected by catching bugs early in the development process. Engineering teams use it because it removes the need to manually build, test, and package applications. It speeds up development, reduces human error, and gives immediate feedback when something breaks. This lets teams move faster, collaborate better, and release more reliable software. With services like CodeBuild, you don’t have to manage build servers—everything is automated, scalable, and only runs when needed, making it cost-effective and efficient.

My CodeBuild project's source configuration means the location from which CodeBuild will pull the application's source code to build, test, and package it. It defines where the code lives so that CodeBuild knows what to work on during each build process. I selected GitHub as my source provider because my web application code is stored in a GitHub repository. By connecting CodeBuild directly to GitHub, I enable automatic integration—so whenever code is pushed to the repository,
CodeBuild can automatically fetch the latest version and start the build process.

Darpan Patel

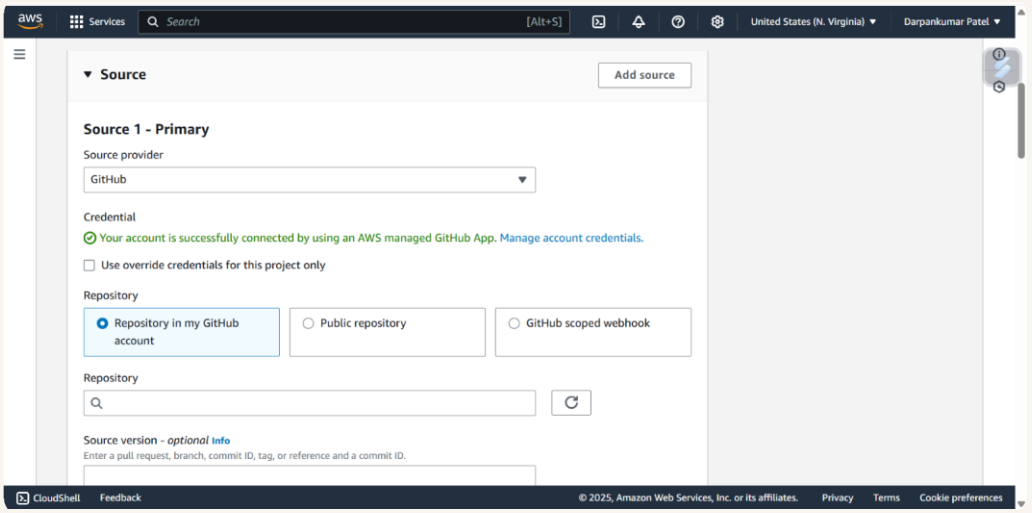


Connecting CodeBuild with GitHub

There are multiple credential types for GitHub, like personal access tokens and OAuth apps. I used GitHub App because it is the most secure and easiest to manage. With GitHub App, AWS handles the authentication and token management behind the scenes, which reduces the risk of credential leaks and eliminates the need for manual token rotation. It also integrates seamlessly with CodeBuild and supports fine-grained permission controls, making it the recommended choice for most CI/CD workflows.

The service that helped connect our AWS environment with GitHub is AWS CodeConnections. It acts as a secure bridge that manages all the authentication and authorization between AWS and GitHub. Instead of manually handling sensitive credentials like personal access tokens or SSH keys, CodeConnections automates this process by using GitHub Apps and securely managing permissions. This simplifies the connection setup, enhances security, and allows CodeBuild to seamlessly access the code repository for continuous integration tasks.

Darpan Patel



CodeBuild Configurations

Environment

My CodeBuild project's Environment configuration means I defined how and where my build will run. I chose On-demand provisioning, so AWS only allocates resources when a build is triggered, keeping it cost-effective. I used a Managed image, which provides a pre-configured environment maintained by AWS, saving time and effort. For Compute type, I selected EC2, offering more flexibility and compatibility for our Java-based project. I set the Operating System to Amazon
Linux, chose the Standard runtime, and picked the image `aws/codebuild/amazonlinux-x86_64-standard:corretto8` to support Java Corretto 8, which matches our app's requirements. Lastly, I let AWS create a new service role to handle permissions, ensuring secure access to resources during the build process. This setup balances efficiency, compatibility, and ease of use, making it ideal for a scalable and maintainable CI pipeline.

Darpan Patel

Artifacts

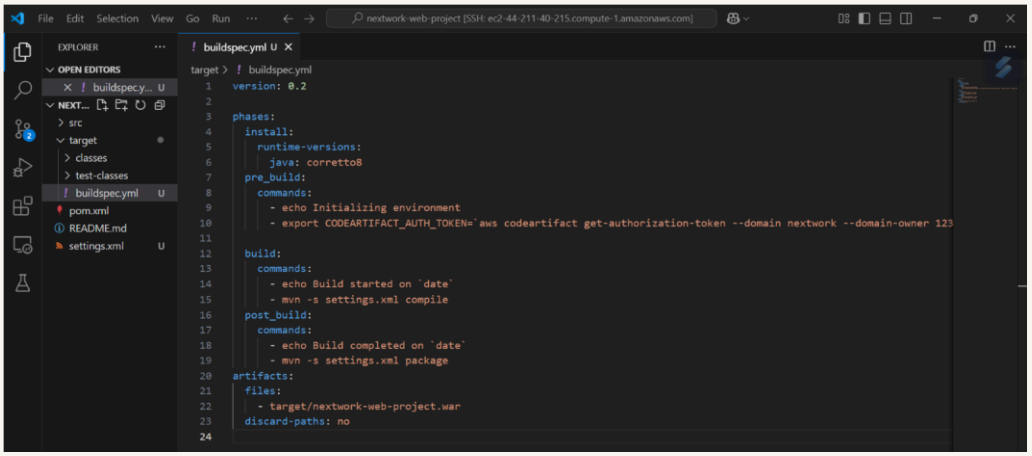
Build artifacts are the output files generated after a successful build. They're important because they contain the actual application package that will be deployed. My build process will create a `.war` file (Web Application Archive), which bundles all necessary files to run the Java web app. To store them, I created an S3 bucket named `nextwork-devops-cicd-darpan-patel`, where CodeBuild will upload the `.war` file after the build finishes. This ensures the artifact is safely stored, easily accessible, and ready for deployment.

buildspec.yml

My first build failed because CodeBuild couldn't find the `buildspec.yml` file in the root of my repository. A `buildspec.yml` file is needed because it defines the build instructions CodeBuild should follow—what to install, how to build the application, and what to output as artifacts. Without it, CodeBuild has no guidance on how to process the source code, which results in a failure during the `DOWNLOAD_SOURCE` phase with a `YAML_FILE_ERROR`. Creating this file is essential to move forward with a successful build.

The first two phases in my `buildspec.yml` file are `install` and `pre_build`. The `install` phase sets up the build environment by specifying Java 8 (Corretto 8) as the runtime. The `pre_build` phase prepares the environment by initializing and retrieving a CodeArtifact authentication token using AWS CLI, allowing Maven to securely access dependencies. The third phase in my `buildspec.yml` file is `build`, where the actual compilation of the Java code takes place using `mvn -s settings.xml compile`. The fourth phase is `post_build`, which finalizes the build process by packaging the compiled code into a WAR file with `mvn -s settings.xml package`. Each phase contributes to a clean and automated build process essential for continuous integration and deployment.

Darpan Patel



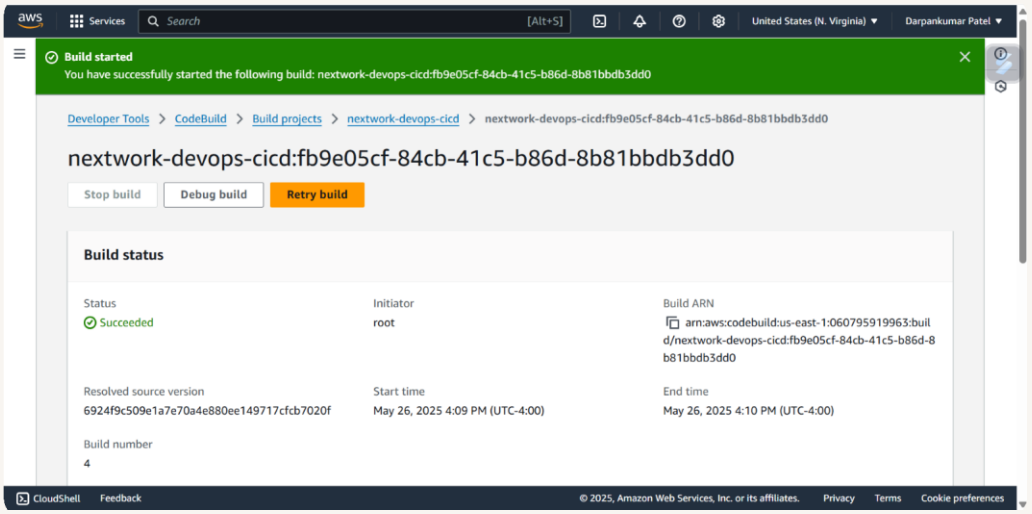
Success!

My second build also failed, but with a different error that said CodeBuild couldn’t access the `settings.xml` file or retrieve dependencies from CodeArtifact. This usually means that the IAM role used by CodeBuild does not have the proper permissions to interact with AWS CodeArtifact. To fix this, I updated the CodeBuild service role in the
IAM console and attached the required policies — either `AmazonCodeArtifactReadOnlyAccess` and `AmazonCodeArtifactAuthTokenAccess`, or a custom inline policy that includes permissions for `codeartifact:GetAuthorizationToken`, `codeartifact:GetRepositoryEndpoint`, `codeartifact:ReadFromRepository`, and `sts:GetServiceBearerToken`. After saving these changes, I retried the build with the updated permissions so CodeBuild can authenticate and access dependencies from CodeArtifact successfully.

To resolve the second error, I attached the `codeartifact-nextwork-consumer-policy` to my CodeBuild service role in the IAM console. This policy grants the necessary permissions for CodeBuild to access AWS CodeArtifact, which is

required to retrieve dependencies specified in the Maven build process. When I built my project again, I saw that the build succeeded, with all phases completing successfully—this confirmed that CodeBuild was now able to authenticate with CodeArtifact and download the required packages.

To verify the build, I checked the Amazon S3 bucket named nextwork-devops-cicd. I looked for the artifact file named nextwork-devops-cicd-artifact.zip inside the bucket.
Seeing the artifact tells me that the build process completed successfully and that the .war file, specifically nextwork-web-project.war, was compiled, packaged, and correctly uploaded—ready for deployment or the next CI/CD stage.



Darpan Patel

Automating Testing

In a project extension, I'm automating basic validation of my project's structure as part of the CI pipeline. I added a test script that checks: Whether the src directory exists – ensuring the core source code folder is present. If index.jsp is located at src/main/webapp/index.jsp – confirming that the main web application entry point is available. A simple placeholder test that always passes – demonstrating how custom tests can be added to the pipeline. These checks help validate that the project is correctly structured before proceeding to the build stage, reducing the chance of avoidable errors during deployment.

To add the test script to the build process, I modified the buildspec.yml file used by AWS CodeBuild. Specifically, I inserted commands in the build phase to: Print a marker indicating the start of the test phase Make the script executable using chmod +x runtests.sh Run the test script using ./run-tests.sh Print another marker to show the test phase completed before

proceeding to the build Here’s the relevant section I added to buildspec.yml: build: commands: - echo "====== BEGINNING TEST PHASE ======"
- echo "Running tests on `date`" - chmod +x run-tests.sh - ./run-tests.sh - echo "====== TEST PHASE COMPLETE ======" - echo "====== BEGINNING BUILD PHASE ======" - echo "Build started on `date`" - mvn -s settings.xml compile - echo "====== BUILD PHASE COMPLETE ======" This setup ensures CodeBuild runs the tests automatically before compiling and packaging the application. If the script exits with a non-zero status (indicating failure), CodeBuil

After pushing my code to GitHub, I ran a new build in AWS CodeBuild. I could see the test phase markers like `====== BEGINNING TEST PHASE ======` and `====== TEST PHASE COMPLETE ======` in the build logs. These logs confirmed that CodeBuild executed my `run-tests.sh` script automatically.

Darpan Patel

Additionally, the output showed `□ PASS` messages and concluded with `ALL TESTS PASSED`, verifying that the testing step was successfully automated as part of the CI pipeline.