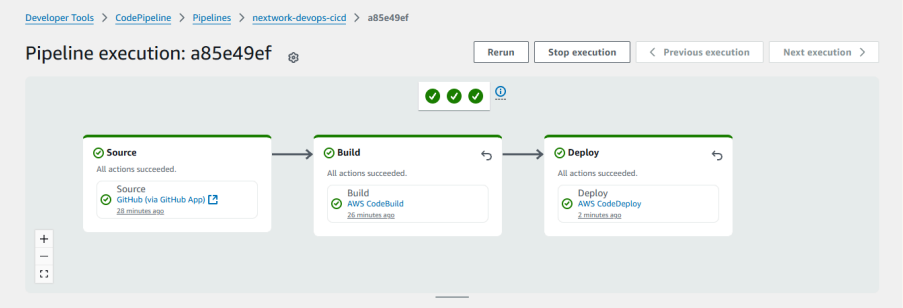
nextwork.org

Build a CI/CD Pipeline with AWS

Louis Moyo



L
Louis Moyo
NextWork Student

Introducing Today's Project!

In this project, I will demonstrate an end-to-end CI/CD pipeline on AWS using CodePipeline, CodeBuild, and CodeDeploy connected to a GitHub repo. I'll set up a source stage that auto-triggers on commits, a build stage that compiles, tests, and packages artefacts via buildspec.yml, and a deploy stage that ships to the target environment with health checks. I'm doing this to learn how to automate releases, make builds repeatable, and reduce manual errors. I'll practise IAM least-privilege, environment variables/parameters, artefact versioning, and rollback strategies (including automatic rollback on failed deployments). By the end I'll have a reusable pipeline template, screenshots, and a concise README that proves I can deliver changes safely, quickly, and reliably-ready to show clients.

Key tools and concepts

Services I used were: GitHub + CodeConnections, AWS CodePipeline, CodeBuild, CodeDeploy, EC2, S3, IAM, Security Groups, CloudWatch Logs, and the CodeDeploy agent. Key concepts I learnt include: CI/CD pipelines, artifacts, buildspec.yml and appspec.yml, instance roles and least-privilege IAM, webhooks, health checks, rollback on failure, troubleshooting with logs, curl and systemctl, and opening port 80 via security groups.

Project reflection

This project took me approximately 5 hours to complete. The most challenging part was diagnosing deploy connectivity and IAM issues. It was most rewarding to see an automatic deploy and a clean manual rollback work end to end.

L
Louis Moyo
NextWork Student

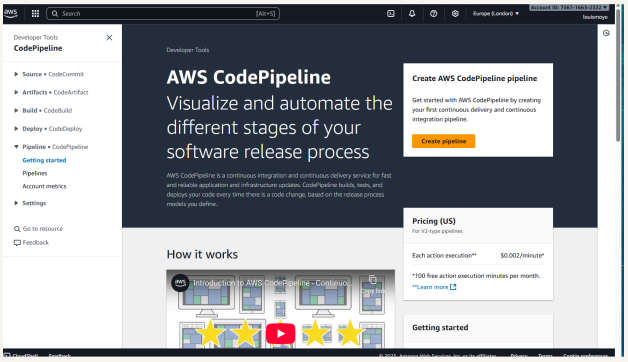
Starting a CI/CD Pipeline

AWS CodePipeline is a managed CI/CD service that automates moving code from commit to production. It watches your source (e.g., GitHub), triggers builds/tests in CodeBuild, and deploys via CodeDeploy/ECS/CloudFormation. Pipelines are stages with actions, artefacts, and optional approvals. You get event-driven triggers, visibility, and safe rollbacks-delivering faster, consistent, low-error releases on every push.

CodePipeline offers three execution modes based on how you want overlapping runs handled: Superseded - a new run cancels any in-progress run. Best when you only care about the latest commit and want to avoid wasted build minutes. (Chosen here.) Queued - runs line up FIFO and each waits for the current one to finish. Good for serial/stateful deploys or long integration tests. Parallel - multiple runs execute at once. Useful for independent branches, multi-service repos, or high-throughput CI with isolated environments.

A service role gets created automatically so CodePipeline can assume it and act on your behalf without storing long-term keys. With this role, CodePipeline can read/write S3 artefacts, start/monitor CodeBuild, trigger CodeDeploy/ECS/CloudFormation, publish events (EventBridge/SNS), and use KMS to encrypt/decrypt artefacts. The trust policy lets codepipeline.amazonaws.com assume the role; the permissions policy grants least-privilege access to your pipeline resources. All actions are auditable in CloudTrail and you can tighten policies later.

Louis Moyo
NextWork Student

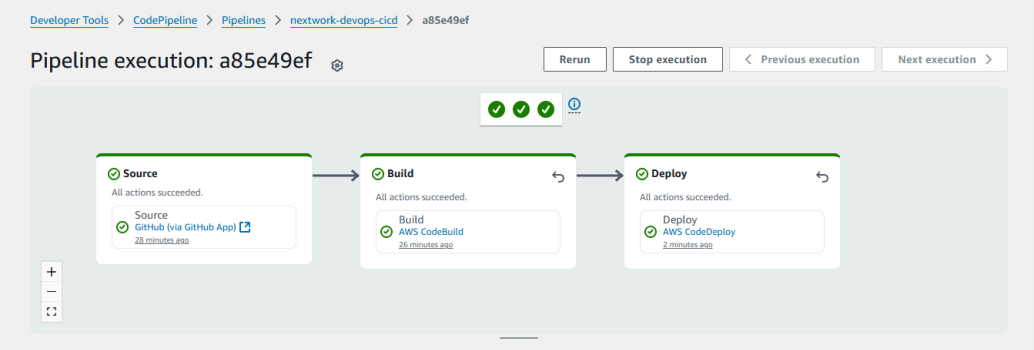


Louis Moyo
NextWork Student

CI/CD Stages

The three stages I've set up in my CI/CD pipeline are Source, Build, and Deploy. While setting up each part, I learnt about Source-GitHub App connections, branch discipline, webhooks, artefact formats. Build-buildspec phases, env vars, IAM/caching, producing a deterministic BuildArtifact, Deploy-appspec hooks, health checks, in-place vs blue/green, instance roles, and automatic rollback.

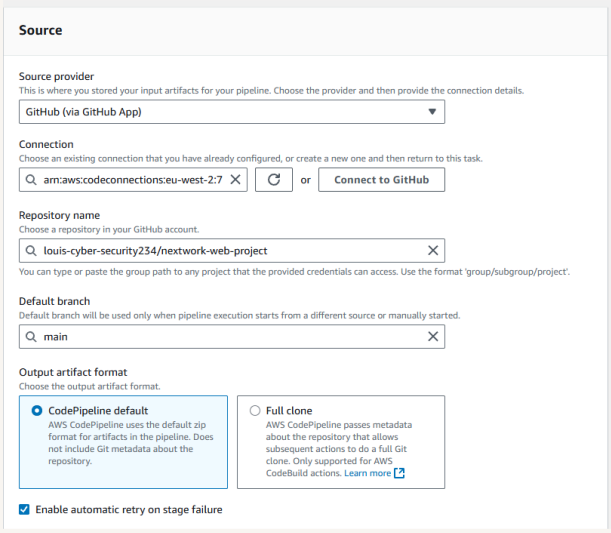
CodePipeline organises the three stages into Source >Build > Deploy. In each stage, you can see more details on status (not started/in progress/succeeded/failed), start/end time and duration, the trigger and action provider, the exact commit/revision, named input/output artefacts, configured resources (repo/branch, CodeBuild project, CodeDeploy app/group), deep links to logs/events, error messages, and Retry/Rerun controls.



Source Stage

In the Source stage, the default branch tells CodePipeline which branch to watch and fetch from. Webhooks/polling are tied to that branch, so pushes there trigger the pipeline, other branches won't unless you add more source actions or change it later. When you start a run manually, the latest commit on this branch is used (unless you override the source revision). Picking a single branch (e.g., main) ensures consistent, safe builds and prevents accidental deploys from feature branches.

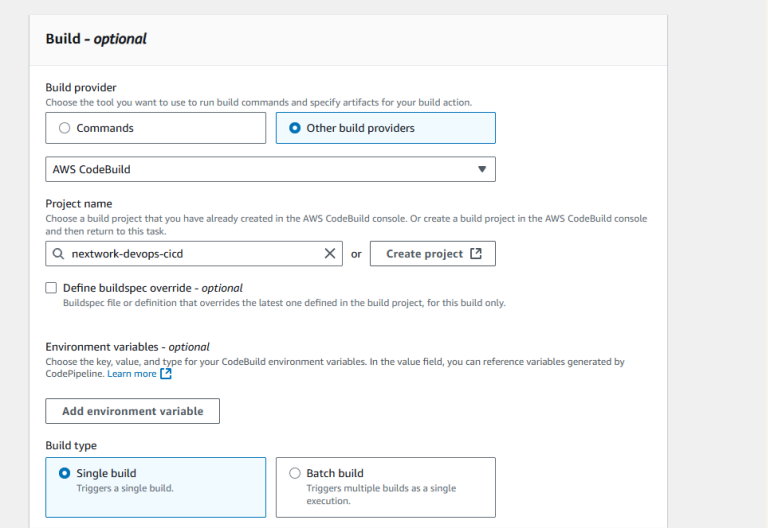
The source stage is also where you enable webhook events, which make the pipeline event-driven. On each push to the chosen branch, GitHub sends a signed callback and CodePipeline immediately starts a new execution with that exact commit. Result: near-instant feedback, no manual runs, less polling latency/cost, and only real code changes trigger builds. Branch/path filters help avoid accidental deploys, and every trigger is auditable-improving traceability and release confidence.


Source
Source provider
This is where you stored your input artifacts for your pipeline. Choose the provider and then provide the connection details.
GitHub (via GitHub App)
Connection
Choose an existing connection that you have already configured, or create a new one and then return to this task.
arn:aws:codeconnections:eu-west-2:7
or
Connect to GitHub
Repository name
Choose a repository in your GitHub account.
louis-cyber-security234/nextwork-web-project
You can type or paste the group path to any project that the provided credentials can access. Use the format 'group/subgroup/project'.
Default branch
Default branch will be used only when pipeline execution starts from a different source or manually started.
main
Output artifact format
Choose the output artifact format.
CodePipeline default
AWS CodePipeline uses the default zip format for artifacts in the pipeline. Does not include Git metadata about the repository.
Full clone
AWS CodePipeline passes metadata about the repository that allows subsequent actions to do a full Git clone. Only supported for AWS CodeBuild actions. Learn more
Enable automatic retry on stage failure

Louis Moyo
NextWork Student

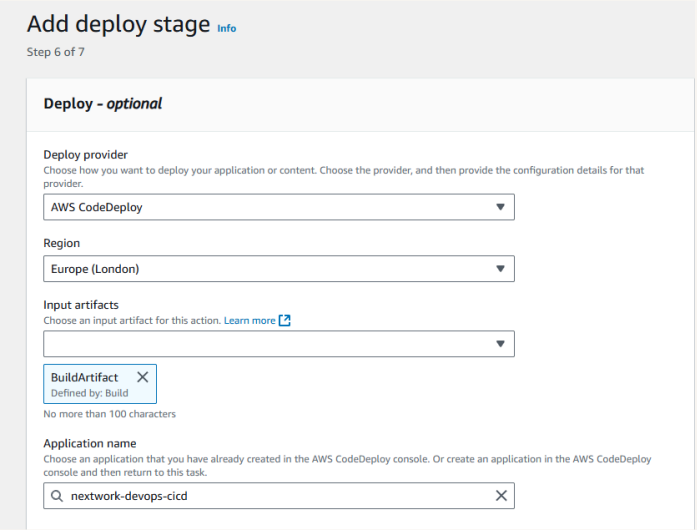
Build Stage

The Build stage sets up CodeBuild to compile, test, and package the code from Source. I configured the nextwork-devops-cicd CodeBuild project to run buildspec.yml with default env/settings and to produce a build output. The input artifact for the build stage is SourceArtifact (the ZIP emitted by the Source stage).



Deploy Stage

The Deploy stage is where the built artefact is released to the target environment. I configured AWS CodeDeploy as the provider, set BuildArtifact as the input, selected my existing CodeDeploy application and deployment group (EC2/Auto Scaling), and used the repo's appspec.yml for lifecycle hooks. I also enabled automatic rollback (and retry) so any failed deployment reverts to the last good version with minimal downtime.



Success!

Since my CI/CD pipeline gets triggered by pushes to main, I edited src/main/webapp/index.jsp and added: If you see this line, that means your latest changes are automatically deployed into production by CodePipeline! I saved, ran git add ., git commit -m "Test CodePipeline: update index.jsp", and git push origin main. That push triggered Source > Build > Deploy and confirmed the pipeline deployed the change end-to-end.

The moment I pushed the code change to main, the webhook fired and CodePipeline started a new execution automatically. In Source, I could see my commit message and ID, and the link opened the exact GitHub diff. Build then picked up that revision, ran the buildspec, and produced the artifact. Deploy used that artifact to update the EC2 host via CodeDeploy, running the lifecycle hooks. All stages turned green-clearly reflecting the same commit message across Source/Build/Deploy for traceability.

Once my pipeline executed successfully, I checked: Executions showed Source, Build, Deploy green Source details matched my GitHub commit ID/message Build logs ended SUCCEEDED with BuildArtifact Deploy lifecycle events showed Succeeded Visiting the instance Public IPv4 DNS returned HTTP 200 and displayed my new line.


Not secure
ec2-3-9-12-2.eu-west-2.compute.amazonaws.com
Hello Louis!
This is my NextWork web application working!
This line is the edit!

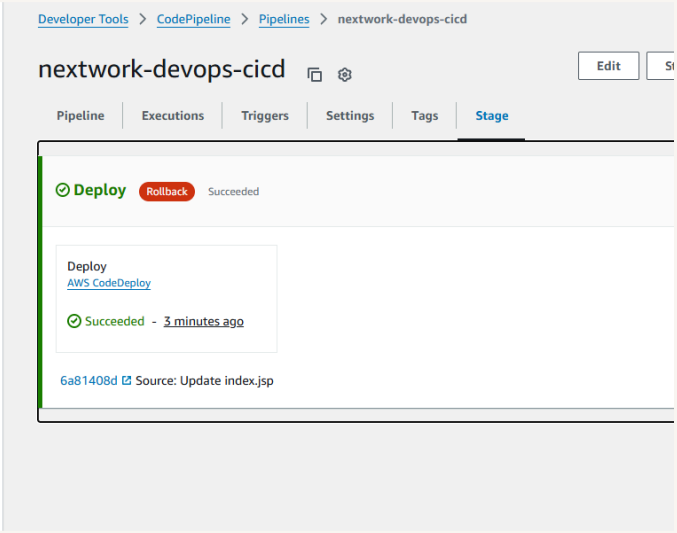
Testing the Pipeline

In a project extension, I initiated a rollback on the Deploy stage - using Start rollback in CodePipeline to redeploy the last successful CodeDeploy revision while keeping Source > Build unchanged. Automatic rollback is important for rapid recovery (lower MTTR), limiting blast radius when bad commits slip through, and ensuring service stability without manual server fixes. It provides a safe, auditable path to restore a known-good version while you investigate and patch the issue.

During the rollback, the Source and Build stages are unaffected because a rollback in CodeDeploy only redeploys a previously known-good application revision. The latest commit and its built artifact still exist as the current outputs of Source > Build. Deploy simply selects an older revision to ship. I verified this by comparing IDs: Source > Build show the latest commit SHA and BuildArtifact, while Deploy shows an earlier revision/commit message from the previous successful deployment.

After rollback, the live web app reverted to the prior version. The extra paragraph This line is the edit! was gone. A hard refresh and curl -I http://<Public-IPv4-DNS> both returned 200 and the HTML source matched the pre-change markup. This confirmed the deploy stage rolled back while the app served the previous revision.

Louis Moyo
NextWork Student



The place to learn & showcase your skills

Check out nextwork.org for more projects