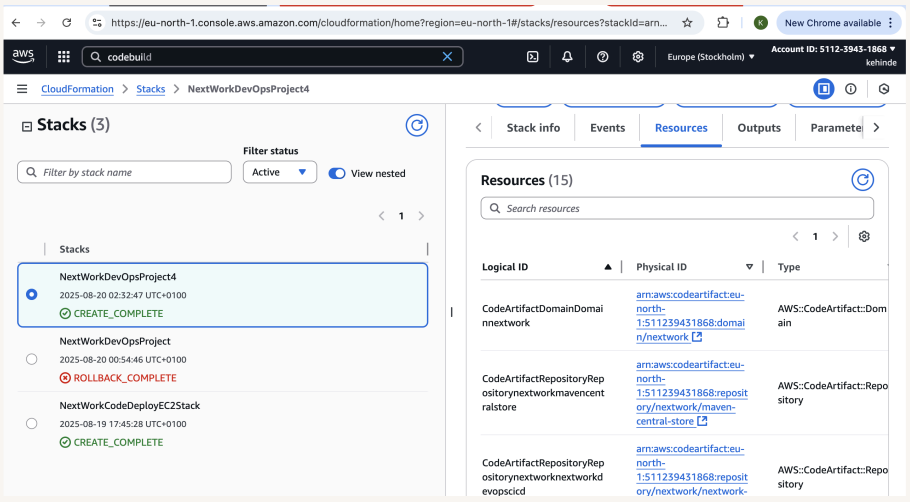
nextwork.org

# Infrastructure as Code with CloudFormation

Kehinde Abiuwa



# Introducing Today's Project!

In this project, I will demonstrate how to rebuild my CI/CD infrastructure step by step with AWS CloudFormation—starting from a dev instance all the way to a CodeDeploy deployment group—and then capture it all in a reusable template. I'm doing this project to learn how Infrastructure as Code makes it easier to set up, fix, and extend AWS resources consistently, while also deepening my understanding of how CI/CD services connect together.

## Key tools and concepts

Services I used were AWS CloudFormation (for Infrastructure as Code), AWS CodeArtifact (for storing build artifacts), AWS CodeDeploy (for automated deployments), AWS CodeBuild (for building and packaging my app), Amazon EC2 (to run my app servers), Amazon S3 (to store artifacts and deployment files), and AWS IAM (to manage permissions). Key concepts I learnt include Infrastructure as Code (IaC) with CloudFormation, using build and deployment scripts, working with the appspec.yml and buildspec.yml files, tagging EC2 instances for deployments, understanding CodeDeploy agents, setting up IAM roles and policies for least privilege, troubleshooting common CloudFormation errors like circular dependencies, and enabling rollbacks for safer deployments.

## Project reflection

This project took me approximately 60 minutes. The most challenging part was troubleshooting CloudFormation errors like circular dependencies and fixing broken references in the template. It was most rewarding to see the CI/CD pipeline come together and successfully deploy my web app automatically.

This project is part six of a series of DevOps projects where I'm building a CI/CD pipeline! I'll be working on the next project right away

**Kehinde Abiuwa**
NextWork Student



# Generating a CloudFormation Template

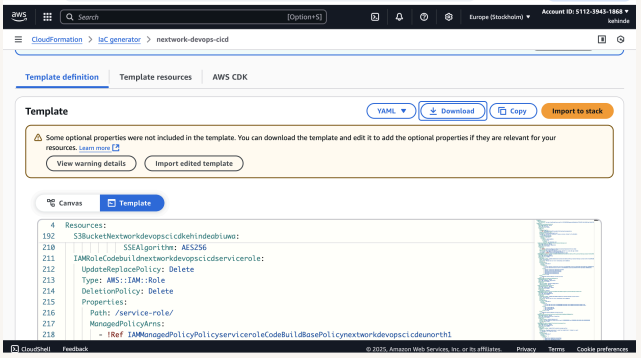
The IaC Generator is a tool that automatically creates Infrastructure as Code templates for you based on your existing resources. It works in a three-step process where it first scans and identifies the AWS resources you already have, then generates a draft CloudFormation template that represents them, and finally lets you review and refine that template before deploying or reusing it.

A CloudFormation template is a YAML or JSON file that describes AWS resources and their configuration so they can be created, updated, or deleted automatically as Infrastructure as Code. The resources that I could add to my template include the EC2 instance, its network interface, security group, subnet, attached volume, and VPC, along with IAM roles like the instance role and the CodeBuild service role. These related resources are recommended so the generated template has all the dependencies needed for the EC2 instance and CI/CD environment to work properly.

The resources I couldn't add to my template were my CodeBuild project and CodeDeploy deployment group because the IaC generator can't automatically capture resources that need detailed configuration (like build environment settings) or sensitive permissions. These must be added to the CloudFormation template manually.

**Kehinde Abiuwa**
NextWork Student





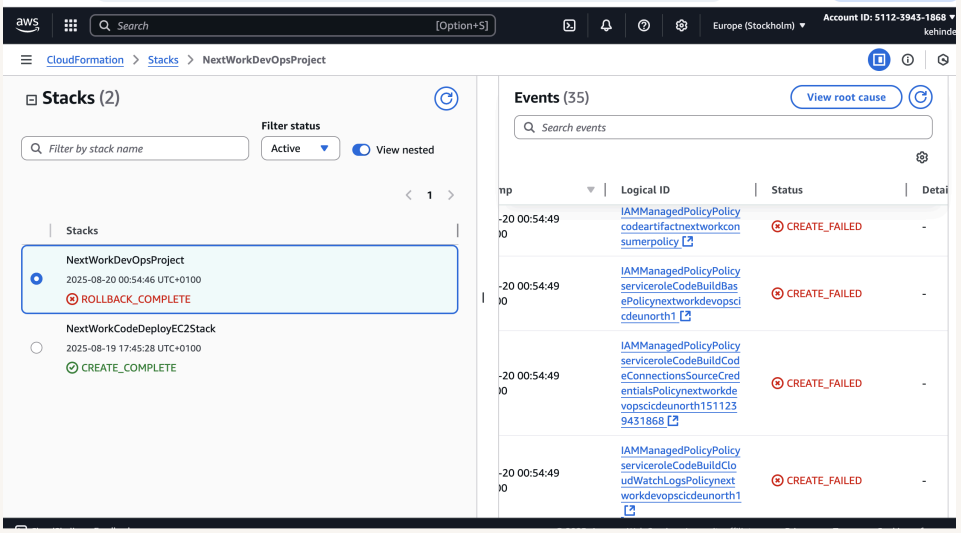
**Kehinde Abiuwa**
NextWork Student



# Template Testing

Before testing my template, I cleaned up (deleted) my existing CI/CD resources because I wanted to avoid conflicts with resources that already existed and to make sure my CloudFormation template could recreate the entire infrastructure from scratch.

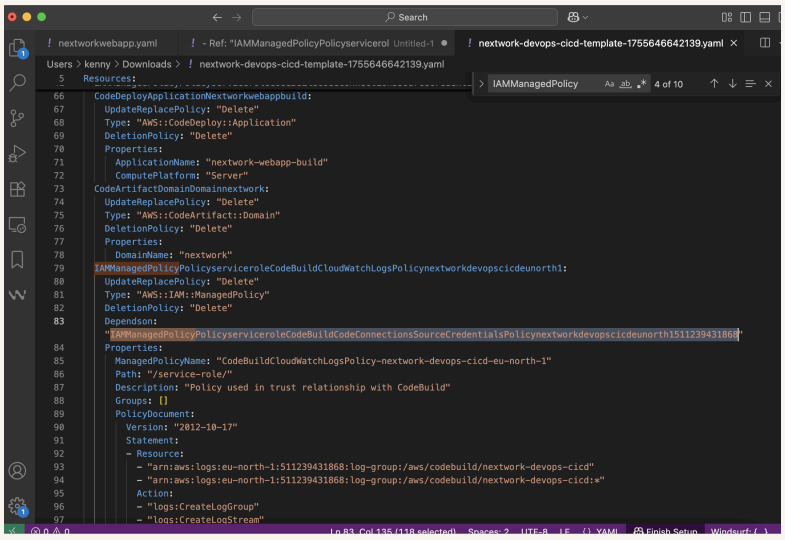
I tested my template by launching it in CloudFormation to recreate my CI/CD stack. The result of my first test was a failure because several IAM Managed Policies couldn't be created. CloudFormation often fails here if the template cant find the specific roles that cloudformation needs to use or tries to duplicate existing policies, if the names aren't unique, or if the account doesn't have permission to create custom managed policies.



# DependsOn

To resolve the error, I added the DependsOn attribute to my IAM policy resources so that CloudFormation would wait until the IAM role was created before trying to attach the policies. The DependsOn attribute means CloudFormation explicitly respects that order—first create the IAM role, then create the policies that depend on it—avoiding the race condition that caused the earlier failure.

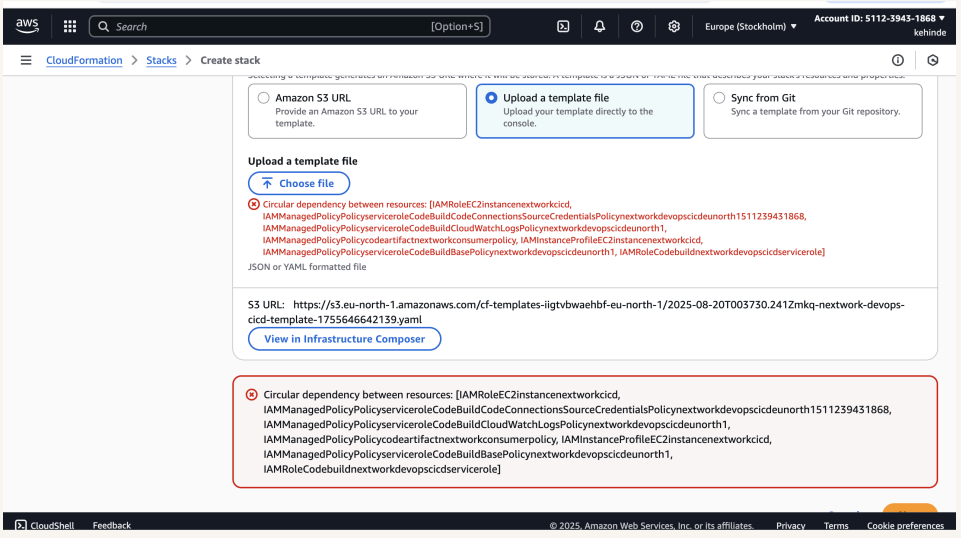
The DependsOn line was added to four different parts of my template: the three CodeBuild managed policies (BasePolicy, CloudWatchLogsPolicy, and CodeConnectionsPolicy) where I set them to depend on IAMRoleCodebuildnextworkdevopscicdservicerole. For CodeArtifact policy, I set it to depend on both IAMRoleEC2instancenextworkcicd and IAMRoleCodebuildnextworkdevopscicdservicerole.



# Circular Dependencies

I gave my CloudFormation template another test! But this time I got a popular error known as a "circular dependency". Circular dependency is a CloudFormation error that happens when two or more resources are defined in a way that they each depend on the other to be created first. This creates a loop (A depends on B, and B depends on A), so CloudFormation can’t figure out which one to build first and fails the stack creation.

To fix this error, I opened my CloudFormation template, searched for the IAMRoleCodebuildnextworkdevopscicdservicerole resource, and deleted the five ManagedPolicyArns lines that were directly referencing my IAM policies. Removing those references broke the circular loop between the role and its policies, allowing the stack to launch successfully.



# Manual Additions

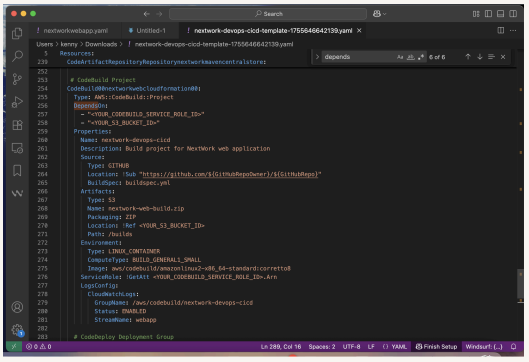
In a project extension, I manually defined two more resources: a CodeBuild project (to build and package my application artifact) and a CodeDeploy deployment group (to control where and how my application is deployed onto the target EC2 instances).

I also had to make sure the references were consistent in this template, so I edited the placeholders (<YOUR_CODEBUILD_SERVICE_ROLE_ID>, <YOUR_S3_BUCKET_ID>, <YOUR_CODEDEPLOY_ROLE_ID>, <YOUR_CODEDEPLOY_APPLICATION_ID>) and replaced them with the actual logical IDs already defined in my template. This ensured that !Ref and !GetAtt were pointing to the right resources (for example, using !Ref CodeDeployApplicationNextworkwebappbuild for the application and !GetAtt IAMRoleCodeDeployServiceRole.Arn for the role). Consistent references prevent circular dependencies and make sure CloudFormation knows the correct order to create each resource.

I also introduced Parameters, which are inputs you define in a CloudFormation template that let you pass in values (like GitHub usernames, branch names, or instance types) at deployment time. They make the template flexible and reusable across different environments without changing the code itself.

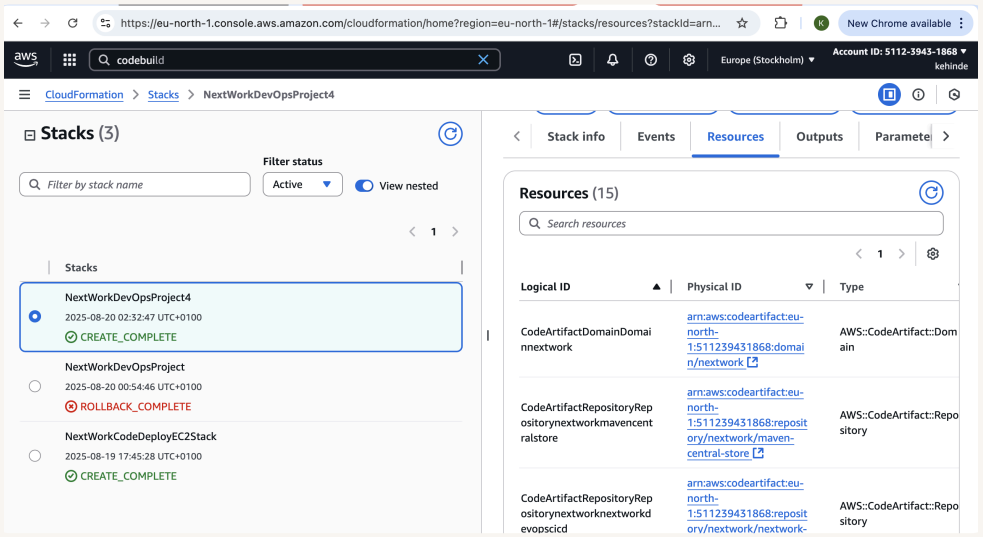
**Kehinde Abiuwa**
NextWork Student





# Success!

I could verify all the deployed resources by visiting the resources tab of the NextWorkDevOpsProject4 stack



# The place to learn & showcase your skills

Check out nextwork.org for more projects