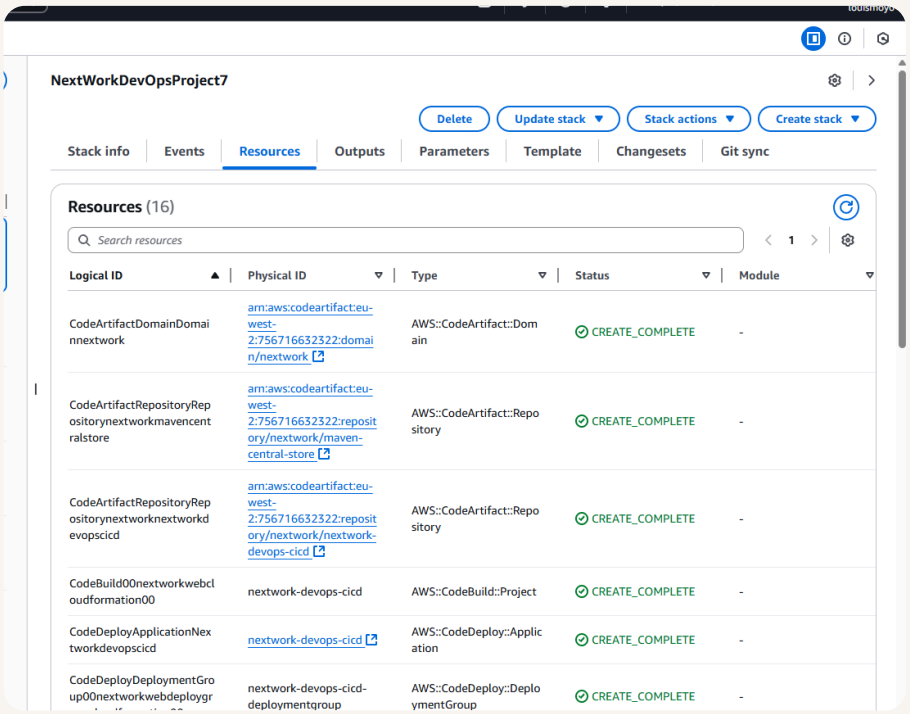
nextwork.org

Infrastructure as Code with CloudFormation

Louis Moyo



Louis Moyo
NextWork Student

Introducing Today's Project!

In this project, I will demonstrate how to use AWS CloudFormation to turn my CI/CD infrastructure into code. I'll rebuild resources like the development instance, CodeArtifact repository, S3 bucket, CodeBuild project, and CodeDeploy deployment group using CloudFormation templates. I'm doing this project to learn how Infrastructure as Code makes deployments consistent, reduces manual errors, and allows me to recreate my CI/CD environment quickly and reliably.

Key tools and concepts

Services I used were CloudFormation, S3, CodeBuild, CodeDeploy, CodeStar Connections, IAM, and CloudWatch Logs. Key concepts I learnt include IaC templating, parameters/refs, stack rollbacks, resource dependencies, bucket policies, least-privilege IAM, build artefacts, and troubleshooting YAML/indentation issues.

Project reflection

This project took me approximately4 hours to complete. The most challenging part was fixing CFN indentation/nesting and IAM links. It was most rewarding to see CREATE_COMPLETE and verify all resources.

L

Louis Moyo
NextWork Student

This project is part six of a series building a CI/CD pipeline! I'll work next on CodePipeline (GitHub > CodeBuild > CodeDeploy), tests, approvals, and notifications.

Louis Moyo
NextWork Student

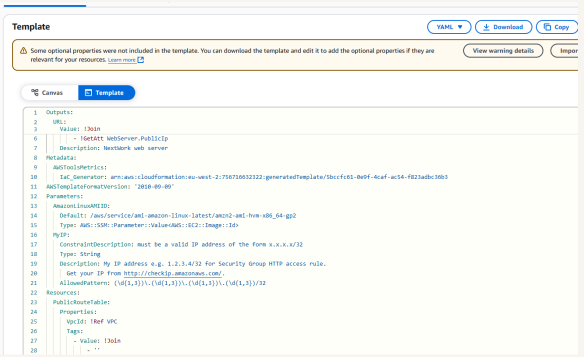
Generating a CloudFormation Template

The IaC Generator is a feature in AWS CloudFormation that automatically creates Infrastructure as Code templates for me. Instead of writing YAML or JSON from scratch, it scans my AWS account, discovers existing resources like EC2 instances, S3 buckets, IAM roles, and networking components, and generates a template describing them. It works in a three-step process where I first scan my account to inventory resources, then select the ones I want grouped into a template, and finally import that template back into CloudFormation to manage or redeploy the resources consistently. This makes building CloudFormation stacks faster, less error-prone, and more repeatable.

A CloudFormation template is a JSON or YAML file that defines the AWS resources I want to create and manage in my account. Instead of manually provisioning services one by one through the AWS console, I can declare everything as code in a template, and CloudFormation will automatically create, configure, and link those resources together in a stack. This makes deployments faster, consistent, and repeatable while reducing the chance of human error.

The resources I couldn't add to my template were the CodeBuild project and the CodeDeploy deployment group, because they require specific configuration details (like environment settings and deployment targets) that the IaC generator can't automatically capture. These need to be defined manually in the CloudFormation template.

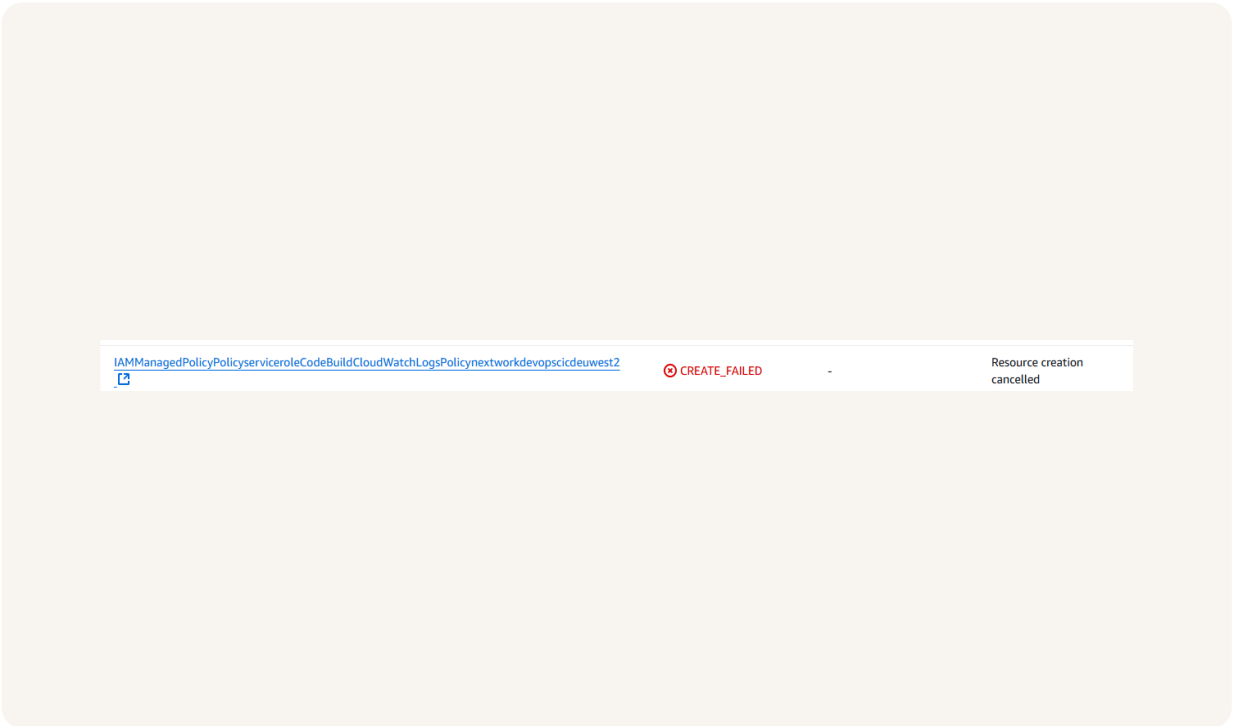
Louis Moyo
NextWork Student



Template Testing

Before testing my template, I deleted all existing CI/CD resources in my AWS account - including the CodeDeploy application, CodeBuild project, CodeArtifact domain and repository, S3 bucket, EC2 instance, IAM roles, and IAM policies. I did this because CloudFormation deployments fail if a resource with the same name already exists, and removing them ensures a clean environment for testing while avoiding duplication and potential errors.

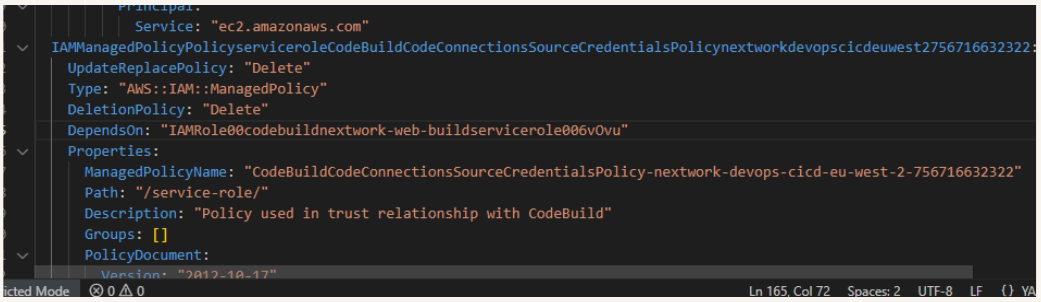
I tested my template by uploading it to CloudFormation with rollback enabled and IAM resource creation allowed. The result of my first test was a CREATE_FAILED error on the IAM managed policies, because CloudFormation attempted to attach them before the IAM role was fully created. This showed me I need to add a DependsOn property to ensure the role is provisioned before the policies are attached.



DependsOn

To resolve the error, I added the DependsOn attribute to each IAM policy so CloudFormation would wait for the IAM role to be created before attaching them. The DependsOn attribute means one resource must finish before another can start, enforcing the correct order of creation. This prevented the earlier race condition and allowed my stack to deploy the IAM role and policies successfully without errors.

The DependsOn line was added to four different parts of my template: for the CodeBuild Base Policy, for the CodeBuild CloudWatch Logs Policy, for the CodeBuild CodeConnections Policy, and for the CodeArtifact policy. For CodeArtifact policy, I made it depend on both the CodeBuild service role and the EC2 instance role so CloudFormation waits for both before attaching the policy.

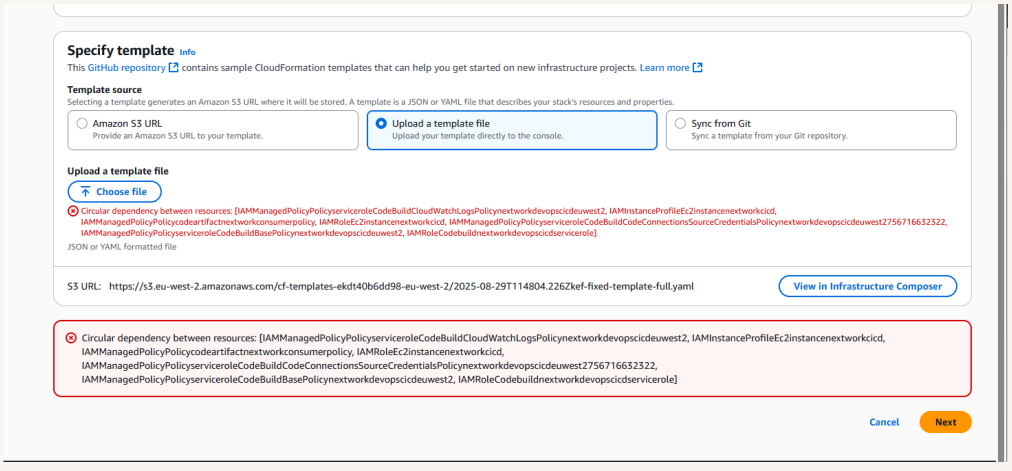


Louis Moyo
NextWork Student

Circular Dependencies

I gave my CloudFormation template another test! But this time I ran into a circular dependency error. CloudFormation told me that some of my IAM roles and IAM policies were referencing each other in a loop. This meant the policies were waiting for the roles to be created first, while the roles were also trying to reference the policies, so neither could be created successfully.

To fix this error, I removed the ManagedPolicyArns block from my CodeBuild IAM role. Those lines were directly referencing the IAM policies that themselves depended on the role, creating a circular dependency loop. By deleting them, CloudFormation now only creates the IAM role first, and then attaches the policies separately via their own DependsOn. This broke the cycle and let the stack deploy successfully.



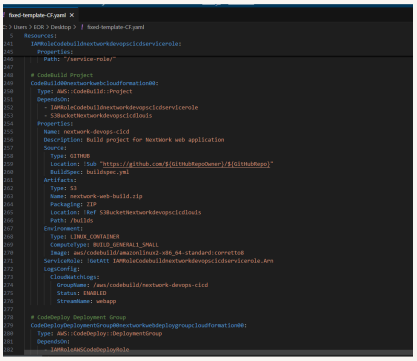
Manual Additions

In a project extension, I manually defined two more resources: 1) CodeBuild Project (AWS::CodeBuild::Project) – depends on the CodeBuild service role and the artifacts S3 bucket, pulls source from GitHub, runs buildspec.yml, uploads ZIP artifacts to S3, and sends logs to CloudWatch. 2) CodeDeploy Deployment Group (AWS::CodeDeploy::DeploymentGroup) – depends on the CodeDeploy app/role, uses CodeDeployDefault.AllAtOnce, and targets EC2 instances tagged role=webserver.

I also had to make sure the references were consistent in this template, so I edited the CodeBuild project to: set DependsOn to IAMRoleCodebuildnextworkdevopscicdservicerole and S3BucketNextworkdevopscicdlouis; set ServiceRole to !GetAtt IAMRoleCodebuildnextworkdevopscicdservicerole.Arn; point Artifacts.Location to !Ref S3BucketNextworkdevopscicdlouis; parameterize Source.Location with !Sub "https://github.com/${GitHubRepoOwner}/${GitHubRepo}"; and keep logs at /aws/codebuild/nextwork-devops-cicd.

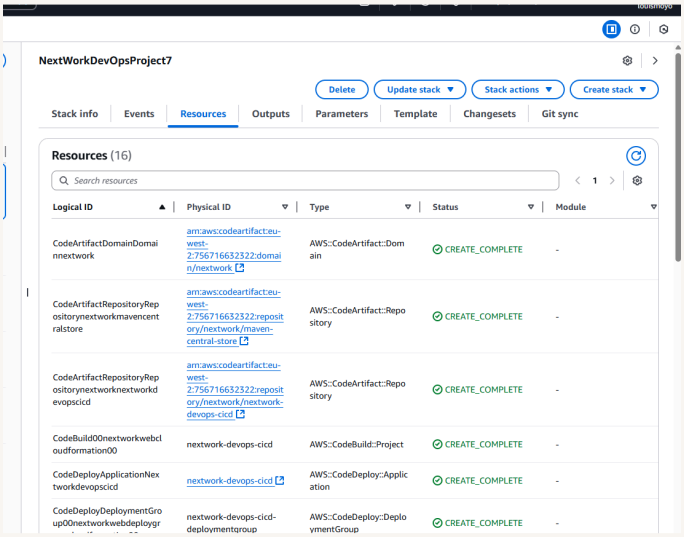
Parameters are input values you pass when creating a CloudFormation stack. They replace hard-coded strings with variables so the same template can be reused across repositories and environments. I added two string parameters-GitHubRepoOwner and GitHubRepo-with sensible defaults, then referenced them with !Sub (e.g., https://github.com/${GitHubRepoOwner}/${GitHubRepo}).

Louis Moyo
NextWork Student



Success!

I could verify all the deployed resources by visiting the CloudFormation stack's Resources tab. For items without links, I checked the resources directly.



The place to learn & showcase your skills

Check out nextwork.org for more projects