AWS CI/CD Pipeline with Security Best Practices: Node.js Application Using GitHub

*Author: Tatenda Manyepa*

# Step-by-Step Implementation

Table of Contents

[Step-by-Step Implementation 1](#_Toc192867412)

[Step 1: Set Up AWS IAM Roles and Permissions 2](#_Toc192867413)

[Step 2: Configure Source Control Using GitHub 6](#_Toc192867414)

[Step 3: Configure the Deploy Stage in AWS CodePipeline 7](#_Toc192867415)

[Step 4: Create an S3 Bucket for Your Application Artefacts 9](#_Toc192867416)

[Step 5: Configure CodePipeline Stages 10](#_Toc192867417)

[Step 6: Create A CodeBuild Project 11](#_Toc192867418)

[Step 7: Link the CodeBuild Project to CodePipeline 12](#_Toc192867419)

[Step 8: Build the Node.js Application 14](#_Toc192867420)

[Step 9: To Automate AWS CodePipeline Deployment 15](#_Toc192867421)

[Successful deployment: The Application 17](#_Toc192867422)

[Step 7: Trigger and Monitor Deployments 17](#_Toc192867423)

[To Monitor the Execution: 18](#_Toc192867424)

[Using AWS Secrets Manager in the CI/CD Pipeline 19](#_Toc192867425)

[Future Enhancements 20](#_Toc192867426)

## Step 1: Set Up AWS IAM Roles and Permissions

Go to AWS Management Console > IAM > Roles > Create Role.

Select Trusted Entity, choose AWS Service and select EC2 or CodeBuild.

Proceed to Add Permissions and attach the following policies:

*AWSCodePipelineFullAccess*

*AWSCodeBuildAdminAccess*

*AWSCodeDeployFullAccess*

*AmazonS3FullAccess (for storing pipeline artifacts)*

Click Next, name the role CodePipelineRole, and create the role.

A screenshot of a computer

Description automatically generated

After creating the role, go to IAM Roles > CodePipelineRole > Trust relationships and click Edit Trust Policy.

Replace the existing JSON with the following to allow CodePipeline to assume the role:

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Principal": {

"Service": "codepipeline.amazonaws.com"

},

"Action": "sts:AssumeRole"

}

]

}

A screenshot of a computer

Description automatically generated

Create additional roles as required

A screenshot of a computer

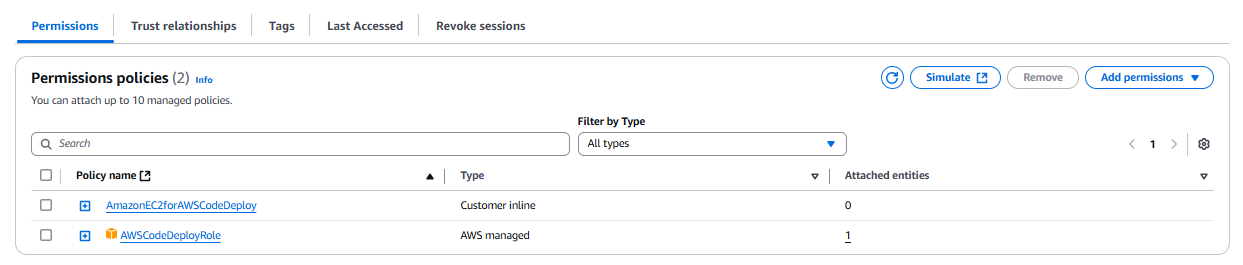
AI-generated content may be incorrect.

CodeBuildRole permissions

A screenshot of a computer

AI-generated content may be incorrect.

CodeDeployRole permissions



EC2InstanceRole permissions

A screenshot of a computer

AI-generated content may be incorrect.

SecretsManagerPipelineRole permissions

A screenshot of a computer

AI-generated content may be incorrect.

Enable **Multi-Factor Authentication (MFA)** for your root account and limit permissions to follow the principle of **least privilege**.

A screenshot of a dashboard

Description automatically generated

## Step 2: Configure Source Control Using GitHub

Go to AWS CodePipeline > Create Pipeline.

Enter a pipeline name, such as NodeJsAppPipeline.

Choose New Service Role or select the previously created CodePipelineRole.

For Source Provider, select GitHub.

Click Connect to GitHub and authorise AWS to access your GitHub account.

Make sure you use Github (via GitHub App) to connect to GitHub. That way, there will not be any need to enable Webhook Integration manually. Also ensure that AWS Connector for GitHub is installed under Applications in GitHub.

Select your repository and branch (e.g., main) that will trigger the pipeline and create the pipeline.

A screenshot of a computer

AI-generated content may be incorrect.

## Step 3: Configure the Deploy Stage in AWS CodePipeline

Go to AWS CodeDeploy > Open the AWS Management Console > Search for AWS CodeDeploy and click on it.

Create a New Application > Click Create Application.

Fill in the details as follows:

Application Name: NodeJsAppDeploy (or any name you prefer)

Compute Platform: Select EC2/On-Premises

Click Create Application

A screenshot of a computer

AI-generated content may be incorrect.

Create an Instance Profile for CodeDeployRole, attach the CodeDeployRole IAM Role to the Instance Profile and verify that the Instance Profile was created.

A screen shot of a computer

AI-generated content may be incorrect.

Create an EC2 instance and associate the CodeDeployRole to the instance.

A close-up of a computer screen

AI-generated content may be incorrect.

SSH into the EC2 instance

A screen shot of a computer

AI-generated content may be incorrect.

Install, Enable and Start the AWS CodeDeploy Agent

A screenshot of a computer program

AI-generated content may be incorrect.

Create a Deployment Group under applications in CodeDeploy

A screenshot of a computer

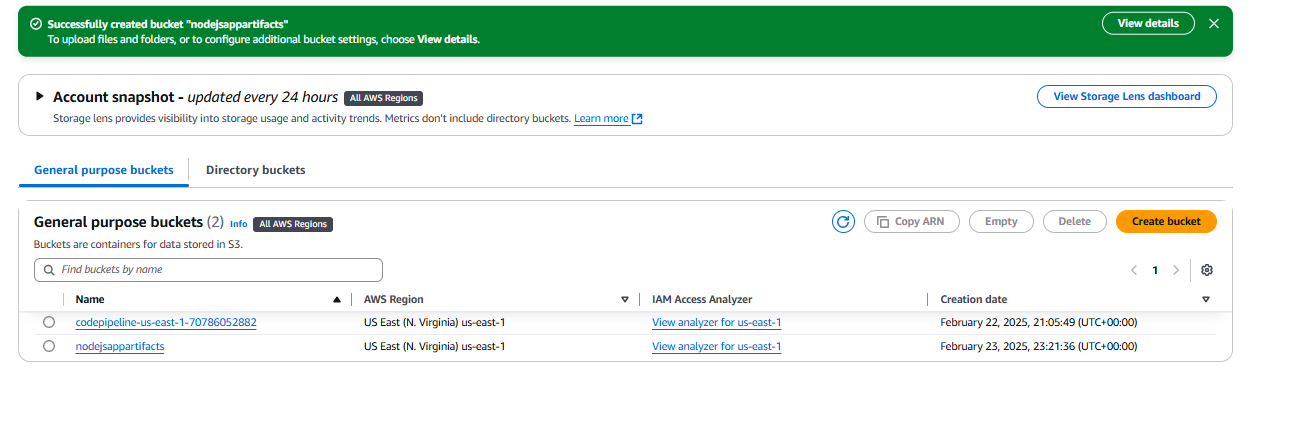
AI-generated content may be incorrect.

## Step 4: Create an S3 Bucket for Your Application Artefacts

Go to AWS S3 > Buckets > Create Bucket.

Name it something like nodejsappartifacts and choose the same region (us-east-1).

Allow public access only if needed. Otherwise, leave it private.



Upload the application file into the S3 bucket

A screenshot of a computer

AI-generated content may be incorrect.

## Step 5: Configure CodePipeline Stages

Go to AWS Console > CodePipeline > Pipelines.

Select your pipeline and click Edit.

In the Source stage, select:

Amazon S3 as the provider.

Choose the S3 bucket where you uploaded the ZIP file (e.g., nodejsappartifacts).

In S3 object key, enter the ZIP file name: e.g. musician-app-master.zip.

Click done and save the pipeline.

A white rectangular object with a black border

AI-generated content may be incorrect.

## Step 6: Create A CodeBuild Project

Go to AWS CodeBuild Console:

Open the AWS Management Console > CodeBuild > Build projects > Click Create build project.

Configure Project:

Project name: NodeJSMusicianApp (This must match the pipeline reference exactly).

Description: Build project for Node.js Musician App.

Source provider: Select Amazon S3.

Bucket: Choose your nodejsappartifacts bucket.

S3 object key: musician-app-master.zip.

Environment:

Environment image: Use an AWS-managed image.

Operating system: Amazon Linux 2.

Runtime: Standard.

Image: aws/codebuild/amazonlinux2-x86\_64-standard:5.0.

Environment type: Linux.

Service role: Choose Create a service role in your account.

Buildspec:

Build specifications: Use a buildspec file (If your app has a buildspec.yml in the source code, select this option).

Artifacts:

Type: Amazon S3.

Bucket name: nodejsappartifacts.

Name: BuildArtifact.zip.

Encryption: Use default settings.

Logs:

Enable CloudWatch logs.

Create the Project.

A screenshot of a computer

AI-generated content may be incorrect.

## Step 7: Link the CodeBuild Project to CodePipeline

Go to AWS CodePipeline > Select your pipeline NodeJsAppPipeline > Edit.

In the Build stage:

Click Edit action.

Under Project name, select NodeJSMusicianApp from the dropdown list.

Click Done and Save pipeline.

A screenshot of a computer

AI-generated content may be incorrect.

## Step 8: Build the Node.js Application

Created an appspec.yml file with the following parameters:

A screenshot of a computer program

AI-generated content may be incorrect.

Run the pipeline

A screenshot of a computer

AI-generated content may be incorrect.

## Step 9: To Automate AWS CodePipeline Deployment

**Set Up Source Control Trigger**

In AWS CodePipeline, configure the source stage to monitor a repository:

GitHub: Set up a webhook to trigger deployments on code commits.

AWS CodeCommit: Enable automatic triggers when changes are pushed.

Amazon S3: Use event notifications for new object uploads (if using S3 as a source).

**Define a CI/CD Workflow**

Create a buildspec.yml file in the repository for AWS CodeBuild to define the build steps:

version: 0.2

phases:

install:

runtime-versions:

nodejs: 18

pre\_build:

commands:

- echo "Installing dependencies..."

- npm install

build:

commands:

- echo "Building the application..."

- npm run build

post\_build:

commands:

- echo "Uploading artifacts..."

- aws s3 cp ./build s3://my-app-bucket --recursive

artifacts:

files:

- '\*\*/\*'

Ensure CodeBuild is integrated into the pipeline.

Automate Deployment Using AWS CodeDeploy

Configure AWS CodeDeploy to deploy the application to EC2 instances, ECS, or Lambda.

Add an appspec.yml file for deployment automation:

*version: 0.0*

*os: linux*

*files:*

*- source: /*

*destination: /var/www/html*

*hooks:*

*ApplicationStart:*

*- location: scripts/start\_server.sh*

*timeout: 300*

*runas: root*

Add Notifications and Monitoring

Enable Amazon SNS notifications for pipeline status updates.

Use Amazon CloudWatch Logs and AWS X-Ray for monitoring.

Automate Testing and Approvals

Add an automated testing stage using AWS CodeBuild or Selenium for UI testing.

Implement manual approval steps if required (e.g., security review before deployment).

### Successful deployment: The Application

A screenshot of a computer

AI-generated content may be incorrect.

## Step 7: Trigger and Monitor Deployments

Commit a Change to GitHub and push a code update to the main branch.

A screenshot of a computer program

AI-generated content may be incorrect.

Go to the CLI and run the following commands

*Git add routes*

*Git commit -m “Add healthcheck route”*

*Git push*

CodePipeline detects the change and triggers the pipeline.

A screenshot of a computer

AI-generated content may be incorrect.

Successful deployment

A screenshot of a computer

AI-generated content may be incorrect.

### To Monitor the Execution:

Go to AWS CodePipeline to track the pipeline execution.

Go to AWS CodeDeploy to verify successful deployment.

Check logs in CloudWatch Logs if issues occur.

## Using AWS Secrets Manager in the CI/CD Pipeline

To enhance security in the CI/CD pipeline, AWS Secrets Manager was integrated to securely store and manage sensitive credentials, such as database passwords and API keys. This implementation prevented hardcoding secrets in the repository and ensured they were retrieved securely during deployment.

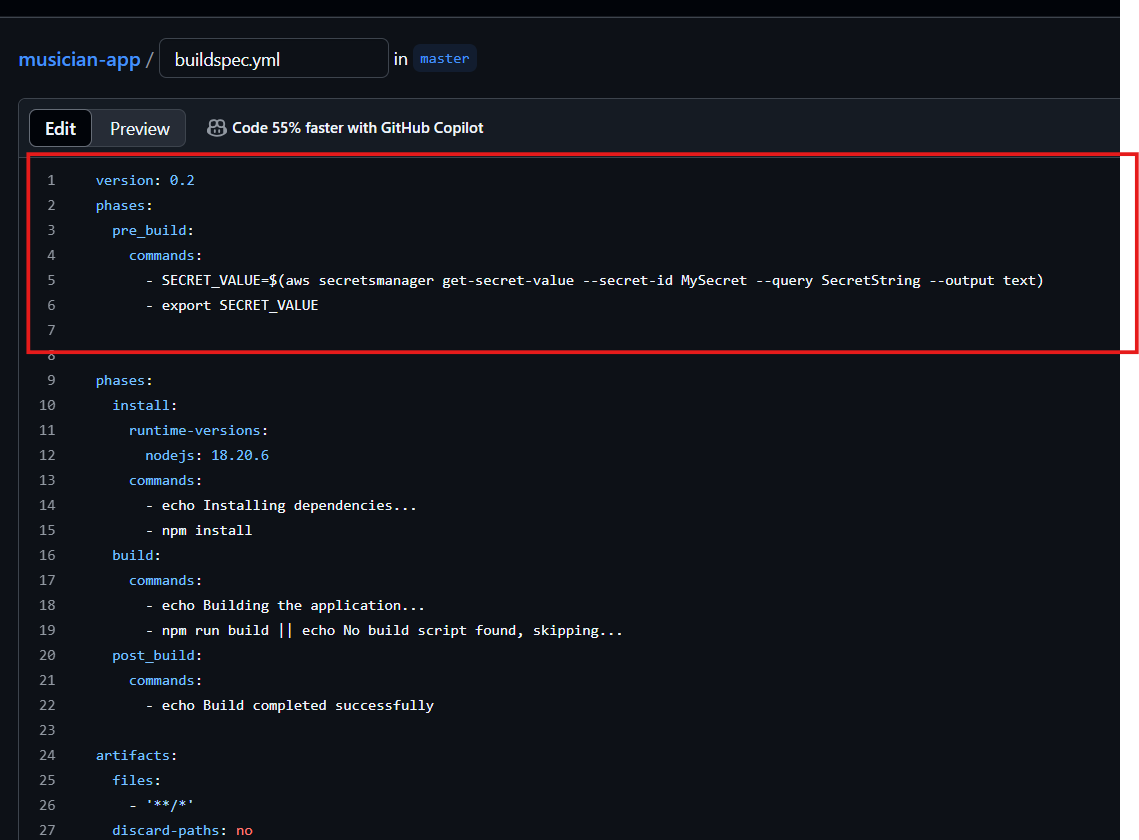
Storing Secrets in AWS Secrets Manager:

Create a secret in AWS Secrets Manager to store sensitive environment variables.

Configure IAM permissions to allow AWS CodeBuild to retrieve the secrets securely.

Retrieving Secrets in the Build & Deploy Stages:

Update the buildspec.yml file in AWS CodeBuild to use the AWS CLI to fetch secrets dynamically:



The retrieved secret was injected as an environment variable, ensuring secure access to credentials during build and deployment.

Using Secrets in the Application:

During deployment, the application referenced the environment variables populated from AWS Secrets Manager.

Ensured that IAM policies only granted read access to the necessary services.

## Future Enhancements

To further optimise and enhance the solution, the following improvements can be considered:

**Containerised Deployment** – Utilise Docker and deploy the Node.js application on Amazon ECS (Fargate) or AWS Lambda for a more scalable, serverless, and efficient deployment model.

**Infrastructure as Code (IaC) Expansion** – Extend the use of Terraform or AWS CloudFormation to manage additional infrastructure components and automate deployments.

**Enhanced Monitoring and Security** – Implement AWS Config, AWS Security Hub, and IAM best practices to strengthen security and compliance.

**Auto-scaling and Load Balancing** – Integrate Application Load Balancer (ALB) and Auto Scaling Groups to handle variable workloads dynamically.

These enhancements would improve scalability, security, and operational efficiency, ensuring the solution remains adaptable to evolving business needs.