AgileCart with CI/CD Pipeline

Team Atom

Aditi Rajesh Limkar

Tejas Guduru

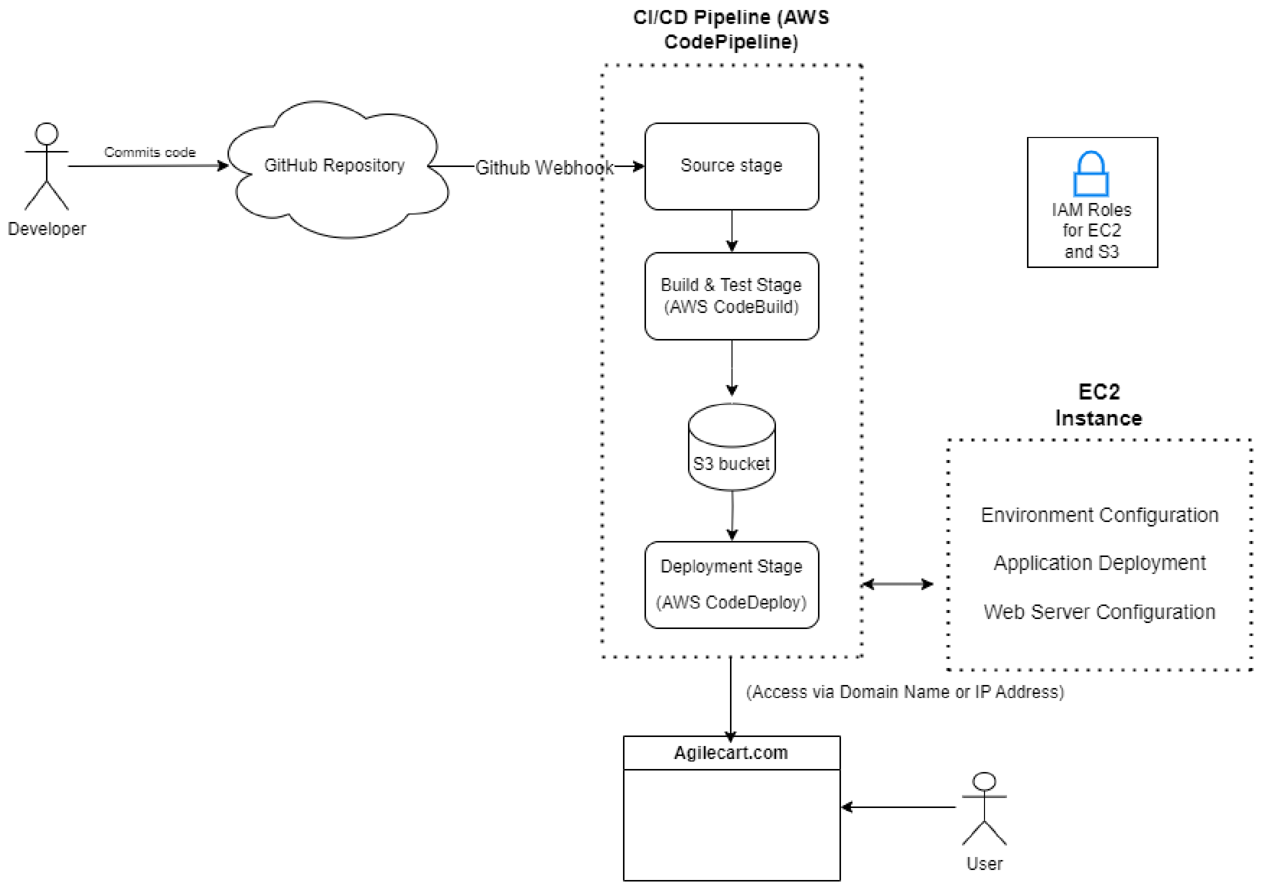
Sri Moulika Yetukuri

Prabodh Wesley Rondla

Abstract:

The blog post focuses on the transformative journey of implementing a Continuous Integration and Continuous Deployment (CI/CD) pipeline, a pivotal element in modern software development. This initiative was fueled by the knowledge and resources acquired in recent courses, with the primary goal of automating the software delivery process. The adoption of CI/CD practices significantly streamlined the team's workflow, leading to a more rapid, reliable, and efficient process. This advancement was particularly beneficial for our project, a burgeoning shopping website developed using the MERN stack, encompassing React for the front end, Express and Node.js for the back end, and MongoDB as the database. This online shopping platform, envisioned to evolve into a full-fledged e-commerce site, offers functionalities like adding or removing items from the cart, checkout, payment integration with Stripe, and a comprehensive user management system. The blog details how the CI/CD pipeline accelerated the deployment of updates and new features, enhancing the overall efficiency and potential of our shopping app.

Architecture:



The architecture of our project's deployment pipeline starts with a developer pushing code to a GitHub repository. A key component in this process is an active GitHub webhook, which detects code pushes and triggers our automated pipeline. Once activated, AWS CodePipeline assumes control, managing the deployment process stages, including code retrieval, building, testing, and deploying. This service integrates with various AWS services for each stage. During the build stage, the code is compiled and packaged, preparing it for the next steps. Here, Amazon S3 plays a critical role as it stores the build artifacts, which can include compiled code, executables, configuration files, and other necessary components for deployment. Finally, the deployment stage involves deploying the code to an EC2 instance. This is facilitated by specific IAM roles attached to the EC2 instance, granting it access to the S3 bucket and enabling deployment through the CodeDeploy agent. This setup ensures a smooth and secure deployment process, integrating various AWS services for an efficient CI/CD pipeline.

Features:

Continuous Integration and Continuous Deployment (CI/CD) are crucial practices in modern software development that significantly enhance the efficiency and quality of a product. Here are some of the most important things that CI/CD can do, along with an example of how they might be used in a typical session:

- Utilizing Amazon EC2 instances for our build environment offers scalability and flexibility. It allows for customizable configurations to match the specific requirements of our project, such as computing power or memory.
- The pipeline automatically tests and integrates new code submissions. This ensures that any changes made are compatible with the existing code and do not introduce new bugs.
- Our pipeline is set up for continuous deployment. Once the code passes the automated tests, it gets deployed automatically, allowing for quick and consistent updates to your application.
- Integrating various AWS services, such as AWS CodePipeline and S3 for artifact storage, provides a robust and scalable infrastructure for our CI/CD pipeline. This integration allows for efficient management of different stages of the deployment process.
- The setup likely includes monitoring mechanisms to track the health of the application and the performance of the pipeline. This provides valuable feedback to the development team for ongoing improvement.
- While not explicitly mentioned, incorporating security best practices within the pipeline, especially given the use of cloud services, is crucial. This may include using secure configurations for the EC2 instances and ensuring that the code complies with security standards.
- The use of EC2 instances means our CI/CD environment can be scaled up or down based on demand and can be customized to suit the specific needs of our project, offering a balance between performance and cost.

How it works:

The CI/CD pipeline for our project leverages a combination of technologies and services, primarily centered around AWS, to automate and streamline the software development process. Here's a deeper dive into how each component plays its part:

Initiation with Code Push: The CI/CD process is set in motion when developers commit their updates to the codebase, which are tracked by a GitHub repository. This action triggers an automated sequence.

Role of GitHub Webhook: Integral to this system is the GitHub webhook. It functions as a real-time alert mechanism, detecting when new code has been pushed to the repository and signaling the start of the CI/CD pipeline.

AWS CodePipeline's Central Role: AWS CodePipeline is the core orchestrator of the process. It seamlessly coordinates the flow from code retrieval to deployment, ensuring each step is executed upon successful completion of the previous one.

Building on EC2: The compilation and preparation of the code are conducted on an Amazon EC2 instance. This instance is configured to handle the specific requirements of the build process, offering both power and flexibility.

Amazon S3's Storage Function: Post-build, the artifacts generated – such as compiled code and configuration files – are stored in an Amazon S3 bucket. S3 provides a secure and accessible location for these essential components.

Automated Testing Phase: An automated testing phase is integral to the pipeline. This includes a variety of tests designed to validate the integrity and functionality of the new code, ensuring seamless integration with the existing codebase.

Deployment Strategy: The deployment phase involves transferring the verified code to a production EC2 instance. This step is automated, reducing manual intervention and enhancing the efficiency of the process.

Utilizing IAM for Secure Access: The pipeline employs AWS Identity and Access Management (IAM) roles to manage permissions. These roles ensure secure access to necessary AWS resources during the build and deployment stages.

Continuous Monitoring: Continuous monitoring mechanisms are in place to oversee the entire process. This includes tracking the performance of the pipeline and the status of the deployed application.

Manual Intervention in Rollbacks: In the current setup, reverting to a previous version of the application in case of deployment issues is a manual process, as an automated rollback mechanism is not implemented.

In essence, our CI/CD pipeline is a well-orchestrated sequence of events, tightly integrated with AWS services, that automates the process of taking code from version control (GitHub) all the way to a deployed state on an EC2 instance, with rigorous testing and quality checks along the way.

Try It Yourself:

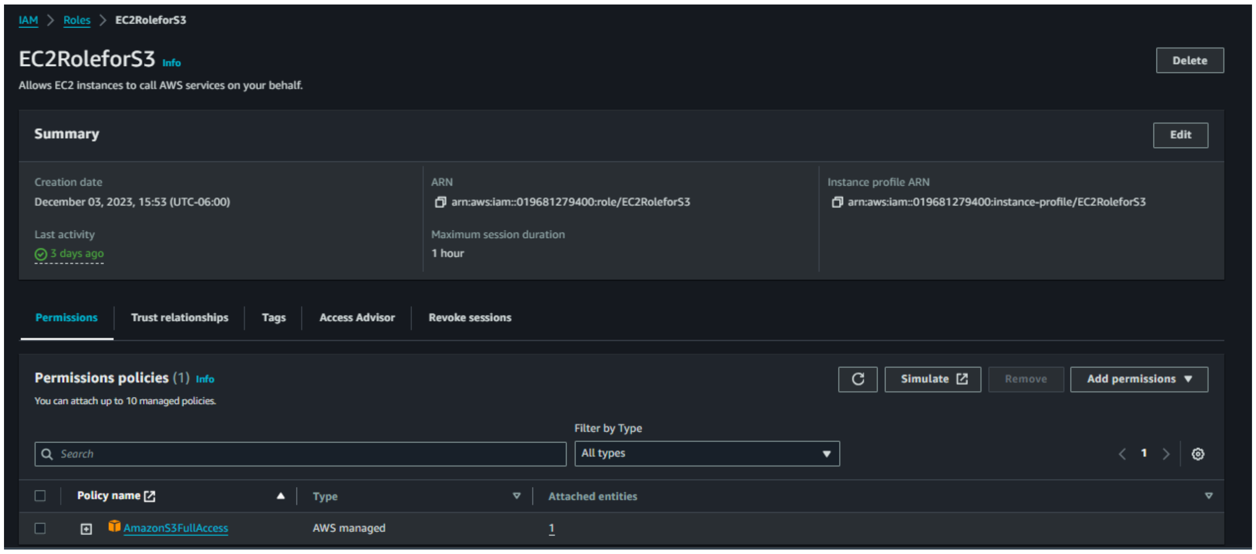
Creating a CI/CD pipeline involves several steps, each crucial for automating and streamlining software deployment. Here's a guide to set up this process:

Step 1:

- Select a Project and Set Up GitHub
- A project needs to be chosen for deployment. This could be an existing project or a new one intended for development.
- The code should be hosted on a GitHub repository. For existing projects, ensure they are already on GitHub. For new projects, a sample project can be cloned from a repository.
- Example clone command: git clone [repository-url]

Step 2:

- Create IAM Roles
- An IAM role for the EC2 instance must be created, including a policy that allows S3 access.



EC2RoleforS3

Allows EC2 instances to call AWS services on your behalf.

Summary

Creation date: December 03, 2023, 15:53 (UTC-06:00)

Last activity: 3 days ago

ARN: arn:aws:iam::019681279400:role/EC2RoleforS3

Maximum session duration: 1 hour

Instance profile ARN: arn:aws:iam::019681279400:instance-profile/EC2RoleforS3

Permissions policies (1)

You can attach up to 10 managed policies.

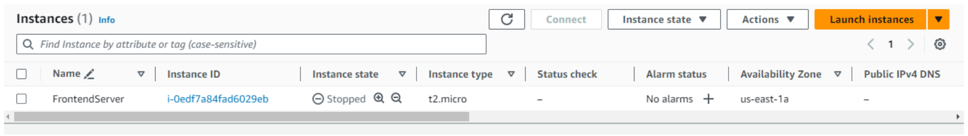
Filter by Type: All types

Policy name	Type	Attached entities
AmazonS3FullAccess	AWS managed	1

- Another IAM role for AWS CodeDeploy should be created and attached with the AmazonEC2RoleforAWSCodeDeploy policy.

Step 3:

- Launch and Configure EC2 Instance



Instances (1)

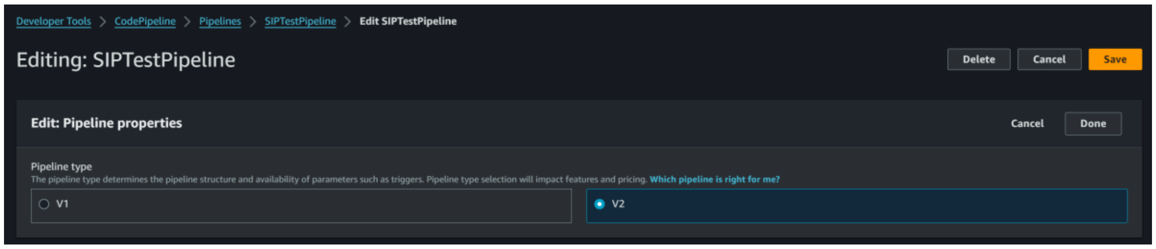
Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
FrontendServer	i-0edf7a84fad6029eb	Stopped	t2.micro	–	No alarms	us-east-1a	–

- An EC2 instance, preferably an Ubuntu Server (or any preferred OS), should be launched.
- The IAM role created for EC2 (for S3 access) should be attached.
- Using EC2 Instance Connect or SSH, the instance needs to be accessed. Subsequent steps include updating the server, creating a new user, and installing the AWS CodeDeploy Agent.

Step 4:

- Create AWS CodePipeline

- A new pipeline is created in AWS CodePipeline.



Developer Tools > CodePipeline > Pipelines > SIPTestPipeline > Edit SIPTestPipeline

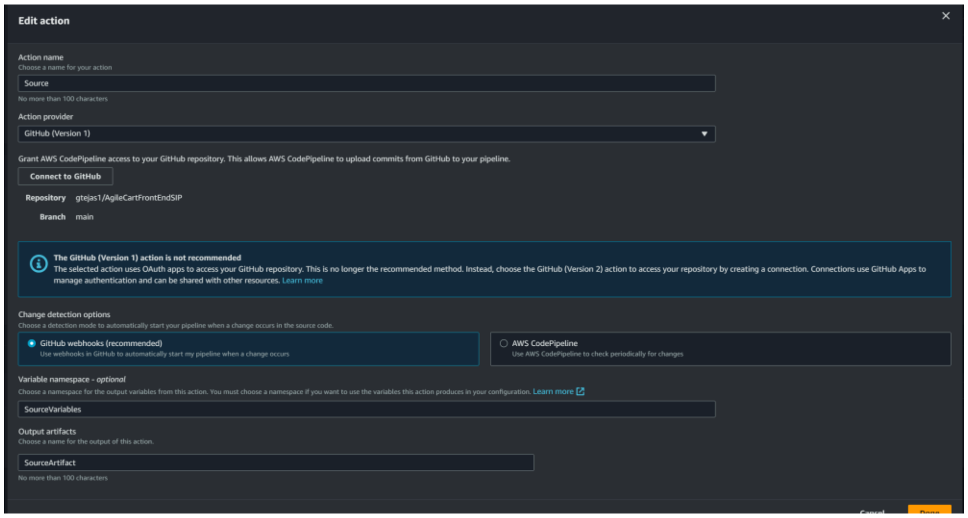
Editing: SIPTestPipeline

Edit: Pipeline properties

Pipeline type
The pipeline type determines the pipeline structure and availability of parameters such as triggers. Pipeline type selection will impact features and pricing. Which pipeline is right for me?

V1 [✓ V2]

- This pipeline is connected to the GitHub repository hosting the project.



Edit action

Action name
Choose a name for your action
Source
No more than 100 characters

Action provider
GitHub (Version 1)

Grant AWS CodePipeline access to your GitHub repository. This allows AWS CodePipeline to upload commits from GitHub to your pipeline.

Repository gtejas1/AgileCartFrontEndSIP
Branch main

The GitHub (Version 1) action is not recommended
The selected action uses OAuth apps to access your GitHub repository. This is no longer the recommended method. Instead, choose the GitHub (Version 2) action to access your repository by creating a connection. Connections use GitHub Apps to manage authentication and can be shared with other resources. Learn more

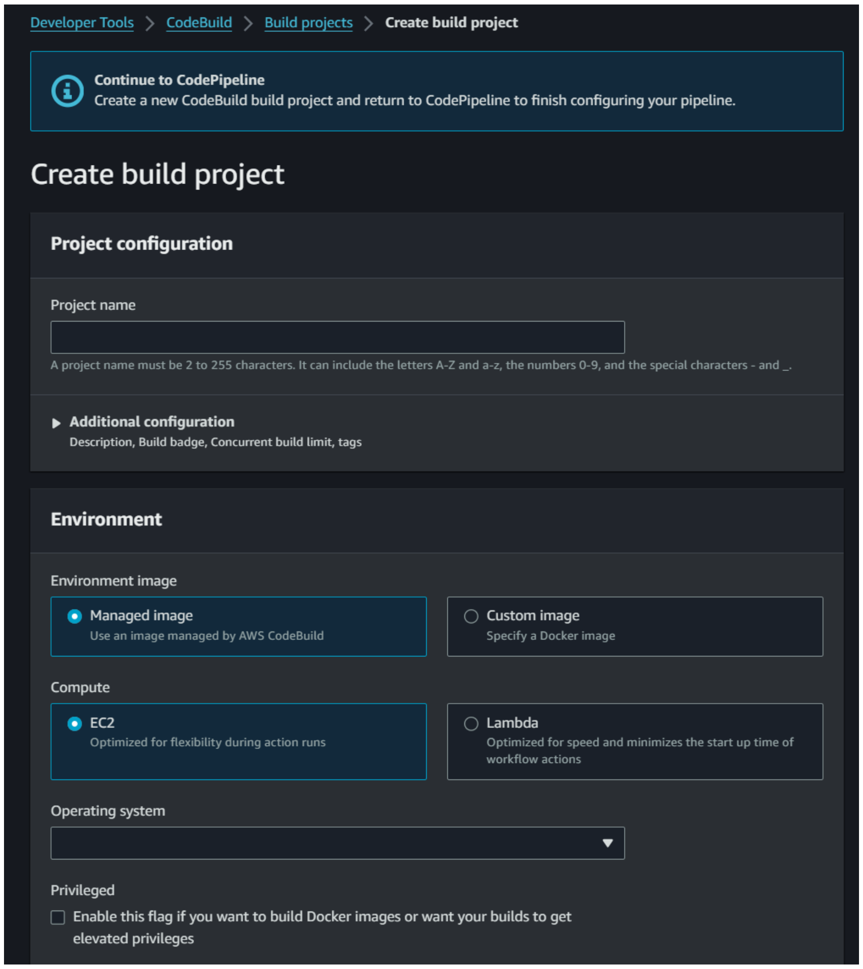
Change detection options
Choose a detection mode to automatically start your pipeline when a change occurs in the source code.
[✓ GitHub webhooks (recommended)] Use webhooks in GitHub to automatically start my pipeline when a change occurs
AWS CodePipeline Use AWS CodePipeline to check periodically for changes

Variable namespace - *optional*
Choose a namespace for the output variables from this action. You must choose a namespace if you want to use the variables this action produces in your configuration. Learn more
SourceVariables

Output artifacts
Choose a name for the output of this action.
SourceArtifact
No more than 100 characters

Step 5:

- Establish AWS CodeBuild
- A new build project is set up in AWS CodeBuild.


Developer Tools > CodeBuild > Build projects > Create build project

Continue to CodePipeline
Create a new CodeBuild build project and return to CodePipeline to finish configuring your pipeline.

Create build project

Project configuration

Project name

A project name must be 2 to 255 characters. It can include the letters A-Z and a-z, the numbers 0-9, and the special characters - and _.

Additional configuration
Description, Build badge, Concurrent build limit, tags

Environment

Environment image

[✓ Managed image — Use an image managed by AWS CodeBuild] Custom image — Specify a Docker image

Compute

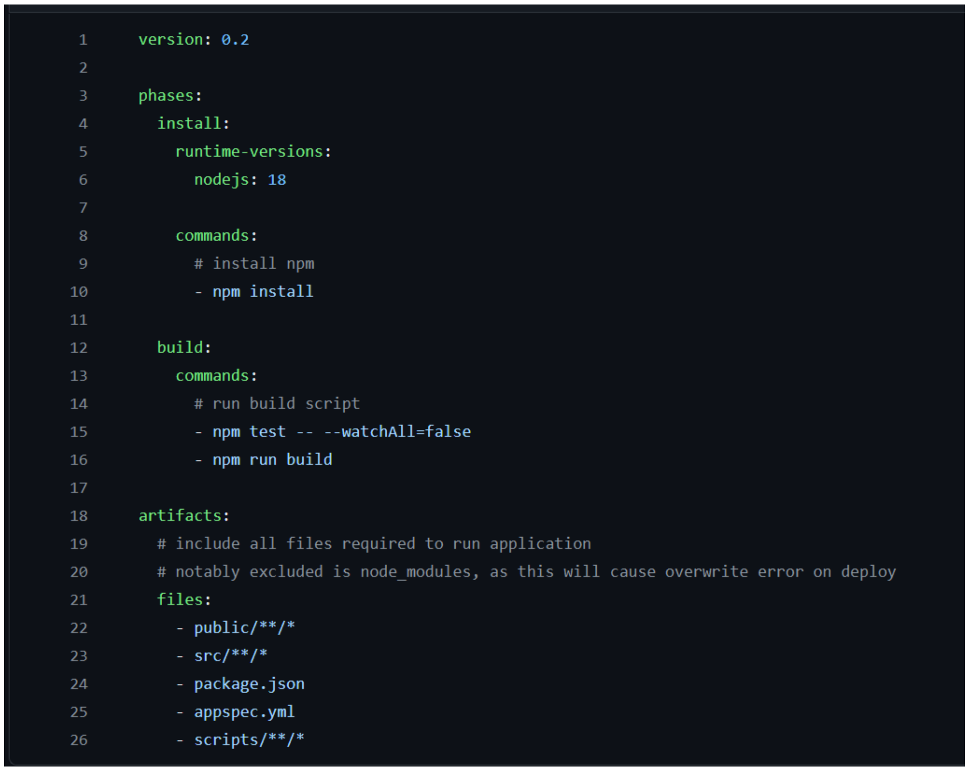
[✓ EC2 — Optimized for flexibility during action runs] Lambda — Optimized for speed and minimizes the start up time of workflow actions

Operating system

Privileged

☐ Enable this flag if you want to build Docker images or want your builds to get elevated privileges

- Build specifications are defined, either directly in the AWS management console or by adding a buildspec.yml file to the project's root directory.



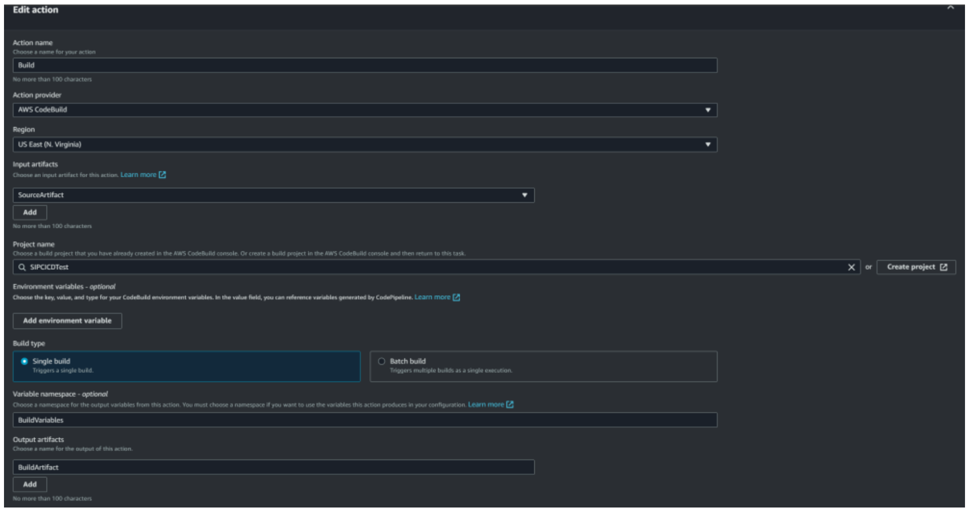
```yaml
version: 0.2

phases:
  install:
    runtime-versions:
      nodejs: 18

    commands:
      # install npm
      - npm install

  build:
    commands:
      # run build script
      - npm test -- --watchAll=false
      - npm run build

artifacts:
  # include all files required to run application
  # notably excluded is node_modules, as this will cause overwrite error on deploy
  files:
    - public/**/*
    - src/**/*
    - package.json
    - appspec.yml
    - scripts/**/*
```



Edit action

Action name
Choose a name for your action
Build
No more than 100 characters

Action provider
AWS CodeBuild

Region
US East (N. Virginia)

Input artifacts
Choose an input artifact for this action. Learn more
SourceArtifact
Add
No more than 100 characters

Project name
Choose a build project that you have already created in the AWS CodeBuild console. Or create a build project in the AWS CodeBuild console and then return to this task.
SIPCICDTest or Create project

Environment variables - *optional*
Choose the key, value, and type for your CodeBuild environment variables. In the value field, you can reference variables generated by CodePipeline. Learn more
Add environment variable

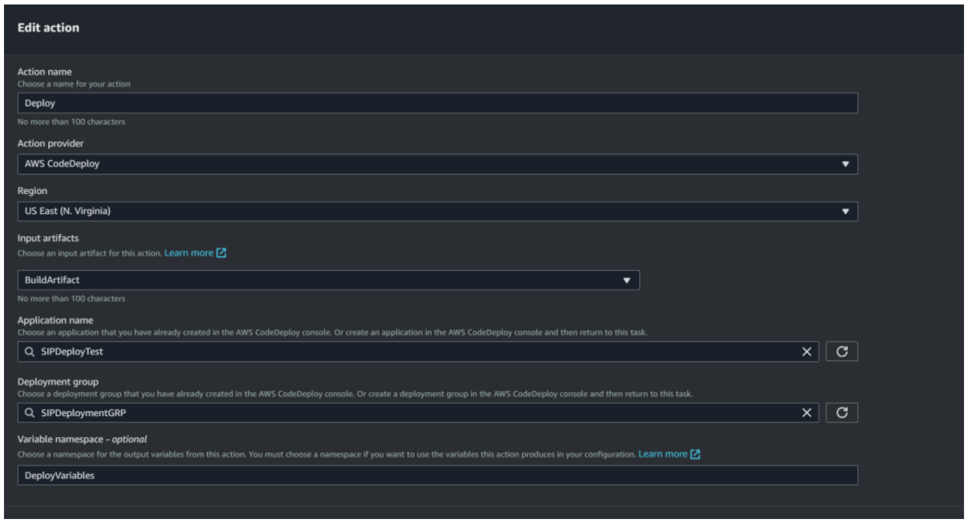
Build type
[✓ Single build] Triggers a single build.
Batch build Triggers multiple builds as a single execution.

Variable namespace - *optional*
Choose a namespace for the output variables from this action. You must choose a namespace if you want to use the variables this action produces in your configuration. Learn more
BuildVariables

Output artifacts
Choose a name for the output of this action.
BuildArtifact
Add
No more than 100 characters

Step 6:

- Implement AWS CodeDeploy
- In AWS CodeDeploy, a new application and deployment group are created. The IAM role for CodeDeploy is selected.
- The project must include an appspec.yml file in its root directory to guide AWS CodeDeploy during the deployment process.



Edit action

Action name
Choose a name for your action
Deploy
No more than 100 characters

Action provider
AWS CodeDeploy

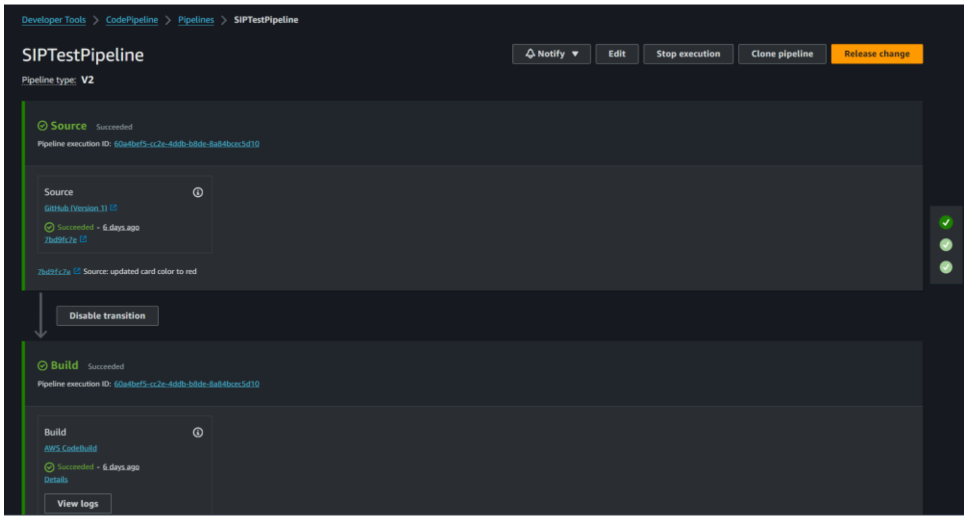
Region
US East (N. Virginia)

Input artifacts
Choose an input artifact for this action. Learn more
BuildArtifact
No more than 100 characters

Application name
Choose an application that you have already created in the AWS CodeDeploy console. Or create an application in the AWS CodeDeploy console and then return to this task.
SIPDeployTest

Deployment group
Choose a deployment group that you have already created in the AWS CodeDeploy console. Or create a deployment group in the AWS CodeDeploy console and then return to this task.
SIPDeploymentGRP

Variable namespace - *optional*
Choose a namespace for the output variables from this action. You must choose a namespace if you want to use the variables this action produces in your configuration. Learn more
DeployVariables



SIPTestPipeline
Pipeline type: V2

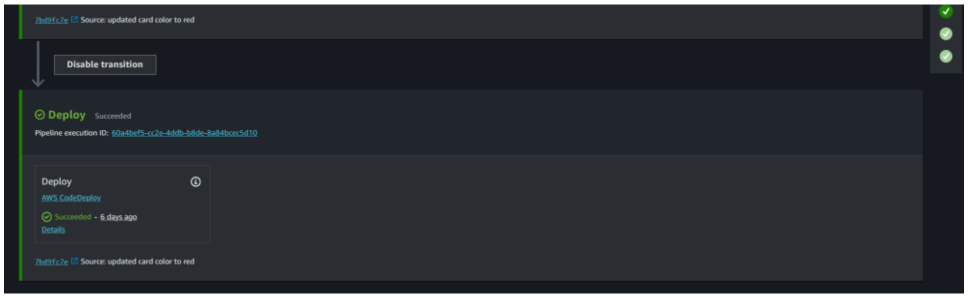
Source Succeeded
Pipeline execution ID: 60a4bef5-cc2e-4ddb-b8de-8a84bcec5d10

Source
GitHub (Version 1)
Succeeded - 6 days ago
7bd9fc7e

7bd9fc7e Source: updated card color to red

Build Succeeded
Pipeline execution ID: 60a4bef5-cc2e-4ddb-b8de-8a84bcec5d10

Build
AWS CodeBuild
Succeeded - 6 days ago
Details



7bd9fc7e Source: updated card color to red

Deploy Succeeded
Pipeline execution ID: 60a4bef5-cc2e-4ddb-b8de-8a84bcec5d10

Deploy
AWS CodeDeploy
Succeeded - 6 days ago
Details

7bd9fc7e Source: updated card color to red

Steps to Run the Application

- Project Setup: The project is cloned from the GitHub repository.
- Clone command: git clone [repository-url]
- Install Dependencies: In the project directory, necessary dependencies are installed.
- Example for a Node.js project: npm install
- Local Testing (Optional): The application can be run locally for testing.
- Example for a Node.js project: npm start

- Push Changes to GitHub: Changes are made and pushed to GitHub to initiate the CI/CD pipeline.
- Commands: git add ., git commit -m "Commit message", git push origin main
- Monitor Pipeline: AWS CodePipeline is monitored for the build, test, and deployment process.
- Accessing the Deployed Application: The application is accessed using the public IP address of the EC2 instance.

Debugging

- During deployment, if errors occur, the AWS console should be checked for details.
- Common issues include the absence of the CodeDeploy agent on the EC2 instance or the agent not running properly. It's crucial to install and start the CodeDeploy agent on the EC2 instance.
- This methodical approach facilitates the setup of a CI/CD pipeline, enabling automated software deployment on an AWS EC2 instance.