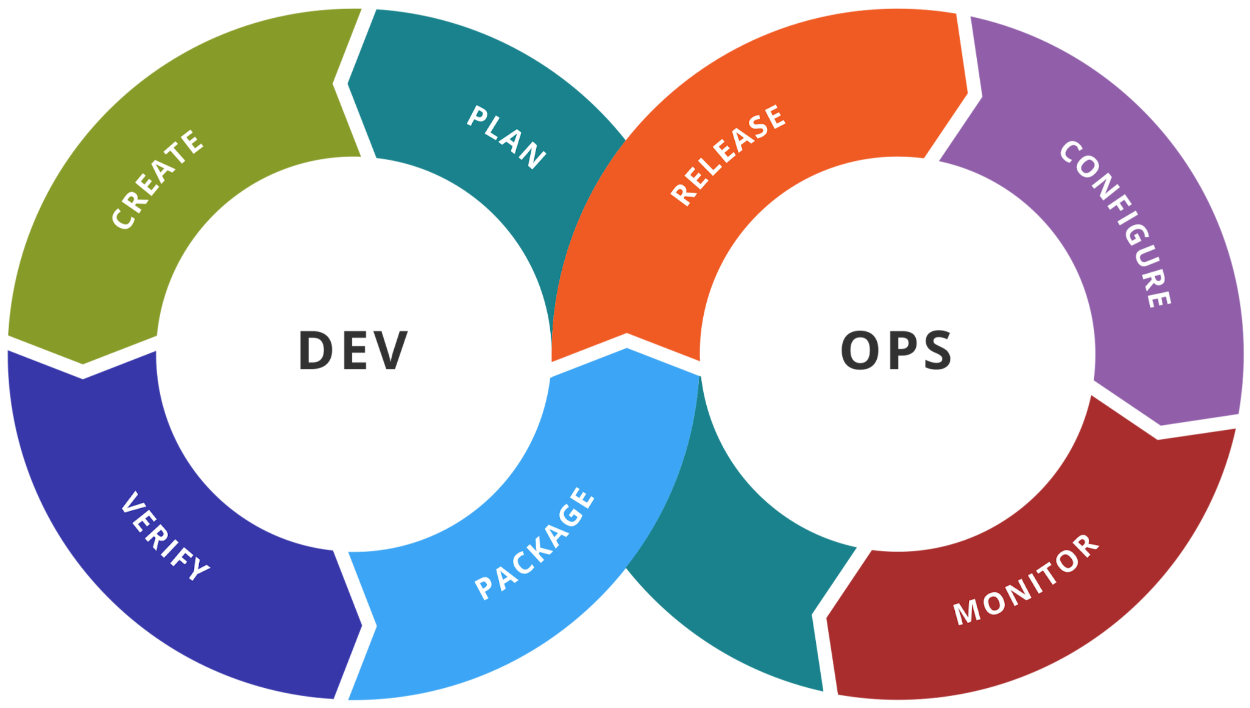
Final Project Document: Automated DevOps Pipeline for CI/CD

By Cameron Souza



Overview of the Project

The purpose of this project is to automate the process of deploying code updates by implementing a robust Continuous Integration and Continuous Deployment (CI/CD) pipeline. In traditional deployment workflows, updates are often performed manually, requiring multiple steps such as pushing code changes, testing, and deploying to the hosting environment. These manual processes are time-consuming, error-prone, and can lead to inconsistencies in deployment, especially in fast-paced development cycles where updates are frequent. To address these challenges, I built an automated workflow leveraging AWS services to streamline the deployment process, reduce human intervention, and ensure that updates are delivered consistently, quickly, and reliably.

The project integrates GitHub as the version control system to manage and track changes to the application source code. AWS CodePipeline serves as the orchestration tool, automating the workflow by connecting the various stages of the pipeline, such as source retrieval, build execution, and deployment. AWS CodeBuild is used to simulate the build process, validating the integrity of the application code before deployment. Finally, Amazon S3 is configured as the hosting environment for the static website, providing a scalable and cost-effective solution for delivering the application.

The end result is a fully automated system where any changes made to files in the GitHub repository automatically trigger the pipeline. AWS CodePipeline fetches the updated source code, runs it through the build stage using AWS CodeBuild, and then deploys the changes to the S3 bucket. This ensures that the website hosted on S3 is always up-to-date with the latest changes from the GitHub repository. By automating the deployment process, this project not only eliminates manual effort but also minimizes the risk of deployment errors, enhances productivity, and ensures a seamless, reliable experience for both developers and end users.

Project Goals

The main objective of this project is to demonstrate the automation of deployment workflows through the implementation of a Continuous Integration and Continuous Deployment (CI/CD) pipeline. In modern software development, deployment automation plays a critical role in ensuring consistency, reliability, and efficiency when delivering code updates to production environments. By automating deployment workflows, this project eliminates manual intervention, reduces the likelihood of human errors, and accelerates the delivery process, making it suitable for scenarios that require frequent updates.

At the core of this project is a CI/CD pipeline that continuously monitors a GitHub repository for changes. GitHub serves as the version control system where the source code for the static website is managed, ensuring that all changes are tracked, versioned, and easily accessible. Whenever an update is made to the repository—such as modifying an HTML file—the pipeline is triggered automatically. AWS CodePipeline acts as the central orchestrator, retrieving the updated source files from GitHub and managing the workflow through a series of well-defined stages.

The first stage of the pipeline involves retrieving the source files directly from the GitHub repository, which is connected to AWS CodePipeline as the source provider. Once the updated files are fetched, the pipeline seamlessly transitions into the build stage, where AWS CodeBuild processes the source code. Although this project focuses on deploying a static HTML website, CodeBuild simulates a build process that mimics real-world scenarios where additional tasks—such as code compilation, testing, and optimization—are required before deployment. By incorporating CodeBuild, the project demonstrates the flexibility of the CI/CD pipeline to accommodate more complex workflows if needed in future enhancements.

Following the build stage, AWS CodePipeline advances to the deployment stage, where the processed files are deployed to an Amazon S3 bucket configured for static website hosting. S3 serves as the hosting environment, providing a scalable, highly available, and cost-effective solution for delivering static content. The deployment process ensures that the latest changes in the repository are immediately reflected on the hosted website. The pipeline operates autonomously and handles the transfer of updated source files to S3 without requiring any manual input, effectively streamlining the deployment workflow.

A critical component of this project is the real-time monitoring of pipeline execution. AWS CodePipeline provides detailed logs and status updates for each stage of the pipeline, allowing developers to monitor the workflow's progress and identify any issues quickly. If an error occurs at any stage—such as failure to retrieve source files, build issues, or deployment problems—the pipeline generates clear logs and notifications, making it easy to pinpoint and troubleshoot the root cause. This feature ensures that errors can be addressed promptly, minimizing downtime and enhancing the reliability of the deployment process.

The final outcome of this project is the successful deployment of a static HTML website to Amazon S3. The automated pipeline ensures that any updates made to the source code in the GitHub repository are efficiently processed and reflected on the hosted site in near real-time. This workflow highlights key benefits such as rapid deployment, reduced manual effort, and improved consistency in delivering changes. By demonstrating the integration of GitHub, AWS CodePipeline, CodeBuild, and S3, the project provides a practical example of how modern CI/CD practices can streamline software deployment processes.

In summary, this project not only automates the end-to-end deployment workflow but also showcases the pipeline's ability to monitor changes, process updates efficiently, troubleshoot errors effectively, and deliver a fully functional static website. The implementation highlights the value of automation in modern development environments, paving the way for scalable, reliable, and maintainable deployment solutions.

Technologies Used

This project leverages a suite of AWS services to orchestrate, host, and process an automated CI/CD workflow, ensuring seamless integration, scalability, and reliability throughout the deployment process. By utilizing cloud-based tools, the project eliminates manual intervention, reduces errors, and enables efficient and automated updates to the static website.

At the heart of the orchestration process is AWS CodePipeline, which serves as the central automation tool to connect the various stages of the workflow. CodePipeline manages the flow of code updates, ensuring that each step—from retrieving the latest changes, processing them through a build phase, and deploying them—is executed sequentially and reliably. By integrating tightly with other AWS services, CodePipeline streamlines the CI/CD process, providing real-time status updates, logs, and error handling. This ensures that updates made to the source code are immediately picked up, processed, and deployed without manual input.

The build process within the pipeline is handled by AWS CodeBuild. While this project does not require code compilation—since it deploys a simple static HTML page—CodeBuild is still used to simulate a build stage, highlighting its flexibility and versatility. In a real-world scenario, CodeBuild could perform tasks such as code compilation, dependency installation, testing, and packaging. By including CodeBuild, the project demonstrates how the pipeline can be extended for more complex workflows that involve dynamic applications, backend services, or multi-step build processes. Even for a static website, this build step reinforces the importance of modularity and scalability in CI/CD pipelines.

Amazon S3 serves as the deployment environment and hosting solution for the project. S3 is configured to host the static web page, offering a scalable, durable, and cost-effective method for serving web content. As an object storage service, S3 allows seamless integration with CodePipeline, enabling updated source files to be directly uploaded to the designated S3 bucket. Once the files are deployed, S3 automatically updates the hosted website, ensuring the most recent changes are live. This approach is particularly effective for static websites, where rapid updates and high availability are critical.

The project uses GitHub as the version control system, where the application source code resides. GitHub plays a vital role in the workflow by triggering the CI/CD pipeline upon detecting changes in the repository. This ensures that any modifications made to the source code, such as updating HTML, CSS, or other website files, are instantly picked up by AWS CodePipeline. By integrating GitHub as the source provider, the project demonstrates how modern version control practices enhance collaboration, code tracking, and deployment automation.

To ensure security and proper access management throughout the workflow, AWS IAM (Identity and Access Management) is used to configure permissions for the pipeline and associated AWS services. IAM policies and roles are carefully defined to grant CodePipeline, CodeBuild, and S3 the appropriate level of access while adhering to the principle of least privilege. This ensures that the pipeline can retrieve source code from GitHub, execute build steps in CodeBuild, and deploy files to the S3 bucket securely. Proper IAM configuration is essential for preventing unauthorized access, securing sensitive resources, and maintaining the integrity of the CI/CD workflow.

In summary, this project highlights the power of AWS services in automating deployment workflows. AWS CodePipeline acts as the orchestrator, coordinating the stages of the CI/CD process. AWS CodeBuild, even in a simplified form, demonstrates the build stage's versatility and scalability. Amazon S3 provides a highly reliable and efficient hosting environment for the static web page, while GitHub serves as the version control system to trigger updates. Finally, AWS IAM ensures that all components of the pipeline operate securely and efficiently, reinforcing best practices for access control and security in cloud-based workflows.

Step-by-Step Instructions to Build the CI/CD Pipeline

The following steps explain how to replicate the project. This guide provides detailed explanations of the tasks, commands, and configurations that were completed.

Step 1: Creating the Application Code

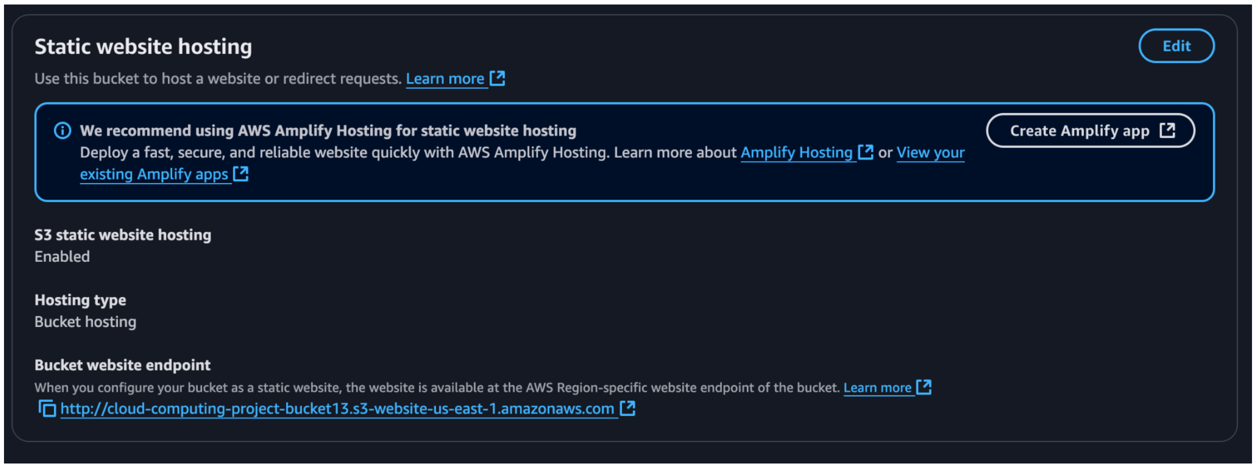
The first step was to create a simple static web page. I created a file named index.html with the following content:

```html
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>My Static Webpage</title>
</head>
<body style="display: flex; justify-content: center; align-items: center; height: 100vh; margin: 0;
background-color: #f0f0f0;">
   <h1 style="font-family: Arial, sans-serif; color: #333;">Hello! I hope you enjoyed the project</h1>
</body>
</html>
```

I saved this file locally and uploaded it to a new GitHub repository named cloud-devops-pipeline. The repository served as the source for the pipeline.

Step 2: Setting Up an S3 Bucket for Static Website Hosting

To host the static website, I created a new S3 bucket in the AWS Management Console. I named the bucket cloud-devops-static-site and enabled Static Website Hosting under the bucket


Static website hosting

Use this bucket to host a website or redirect requests. Learn more

We recommend using AWS Amplify Hosting for static website hosting
Deploy a fast, secure, and reliable website quickly with AWS Amplify Hosting. Learn more about Amplify Hosting or View your existing Amplify apps

S3 static website hosting
Enabled

Hosting type
Bucket hosting

Bucket website endpoint
When you configure your bucket as a static website, the website is available at the AWS Region-specific website endpoint of the bucket. Learn more
http://cloud-computing-project-bucket13.s3-website-us-east-1.amazonaws.com

properties. I specified the index document as index.html to ensure that the correct file is served when the website is accessed.

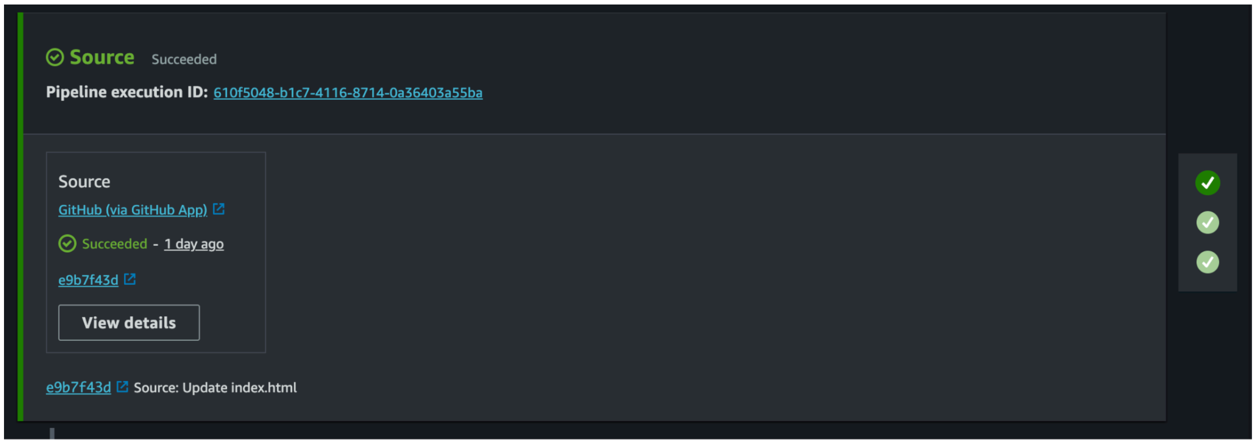
To allow public access to the website, I updated the bucket policy with the following configuration:

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "PublicReadGetObject",
   "Effect": "Allow",
   "Principal": "*",
   "Action": "s3:GetObject",
   "Resource": "arn:aws:s3:::cloud-devops-static-site/*"
  }
 ]
}
```

This policy allows the contents of the bucket to be read by anyone accessing the static website.

Step 3: Configuring AWS CodePipeline

The next step was to set up AWS CodePipeline to automate the deployment workflow. I created a new pipeline named static-website-pipeline. In the Source stage, I configured GitHub as the source provider and connected the repository cloud-devops-pipeline. I selected the main branch to monitor for changes.



Source Succeeded

Pipeline execution ID: 610f5048-b1c7-4116-8714-0a36403a55ba

Source

GitHub (via GitHub App)

Succeeded - 1 day ago

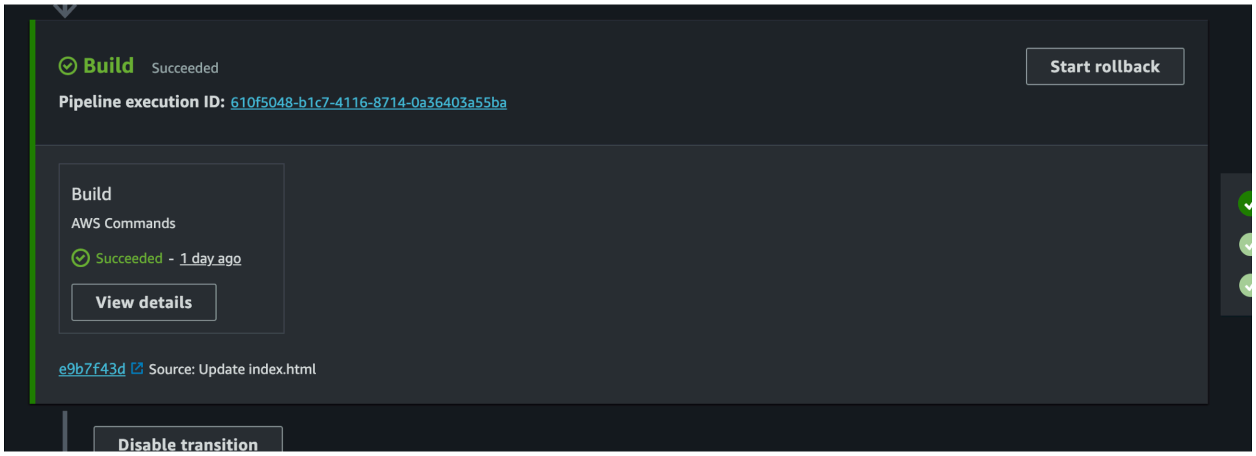
e9b7f43d

e9b7f43d Source: Update index.html

In the Build stage, I used AWS CodeBuild to simulate a build process. Since this is a static website, no actual build steps were needed. To satisfy the build phase, I added an inline command to upload the index.html file to the S3 bucket with the correct Content-Type. The following command was used:

```
aws s3 cp index.html s3://cloud-devops-static-site/index.html --content-type "text/html"
```

This command ensures that the file is uploaded to S3 and that the Content-Type metadata is explicitly set to text/html.



Pipeline execution ID: 610f5048-b1c7-4116-8714-0a36403a55ba

Succeeded - 1 day ago

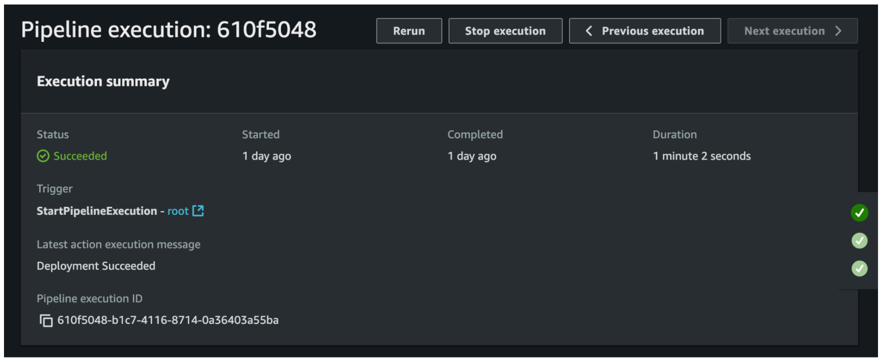
e9b7f43d Source: Update index.html

Step 4: Testing the Pipeline

To test the pipeline, I made a small update to the index.html file. I changed the text inside the <h1> tag to “Update Success” After saving the file, I pushed the update to the GitHub repository using the following commands:

```
git add index.html
git commit -m "Updated project message"
git push origin main
```

Once the changes were pushed, AWS CodePipeline detected the update and triggered the pipeline automatically. The Source stage retrieved the updated file from GitHub. The Build stage executed the inline command to upload the file to the S3 bucket. Finally, the Deployment stage completed successfully, and the updated version of the website became visible at the S3 Static Website Hosting URL.


Pipeline execution: 610f5048

Execution summary

Status	Started	Completed	Duration
Succeeded	1 day ago	1 day ago	1 minute 2 seconds

Trigger
StartPipelineExecution - root

Latest action execution message
Deployment Succeeded

Pipeline execution ID
610f5048-b1c7-4116-8714-0a36403a55ba

Results and Verification

After the pipeline completed successfully, I verified that the website had been updated by accessing the Static Website Hosting URL provided by the Amazon S3 bucket. By navigating to the URL, I was able to see the updated message displayed on the web page, which directly reflected the changes I made in the GitHub repository. This step was essential to confirm that the CI/CD pipeline was functioning as intended and that updates to the repository were seamlessly picked up, processed, and deployed to the hosting environment. The immediate availability of the updated content demonstrated the reliability and efficiency of the automated workflow.

To ensure the pipeline's functionality was thoroughly validated, I proceeded to review the AWS CodePipeline logs within the AWS Management Console. I checked each stage of the pipeline—Source, Build, and Deploy—to confirm that they had all been executed successfully.

1. In the Source stage, I verified that the pipeline correctly retrieved the updated files from the connected GitHub repository. This confirmed that the integration between GitHub and AWS CodePipeline was properly configured and responsive to changes in the repository.

2. In the Build stage, I ensured that AWS CodeBuild completed its execution without errors, even though no code compilation was required. Logs indicated that the build environment successfully processed the incoming files, demonstrating the versatility of the build stage, which could accommodate future enhancements such as testing or optimization.

3. In the Deploy stage, I validated that the processed files were correctly transferred to the Amazon S3 bucket designated for website hosting. This step confirmed the seamless integration between CodePipeline and S3, ensuring that the deployment workflow was reliable and efficient.

Additionally, to ensure the content was being served correctly, I checked the metadata of the index.html file in the S3 bucket. Proper metadata configuration is crucial for browser rendering, as it informs the browser of the file type. Specifically, I confirmed that the Content-Type attribute for the index.html file was set to text/html. This configuration ensures that the file is recognized and displayed as an HTML document rather than as plain text or any other format. Maintaining this setting is particularly important for static website hosting, as incorrect metadata could lead to rendering errors and an inconsistent user experience.

To further validate the deployment, I refreshed the Static Website Hosting URL multiple times to ensure that the changes persisted and were not affected by browser caching or propagation delays. This step reinforced confidence that the S3 bucket was serving the latest version of the website files. I also tested accessibility across different browsers and devices to confirm that the updates were consistent and rendered correctly regardless of the platform.

By completing these verification steps—reviewing the CodePipeline logs, validating the deployment stages, and confirming the Content-Type metadata—I ensured that the pipeline was functioning correctly and reliably. The updated content displayed on the static website served as clear evidence that changes made in the GitHub repository triggered the pipeline, passed through each stage without errors, and were successfully deployed to the hosting environment. This comprehensive validation demonstrated the robustness of the CI/CD pipeline and highlighted the value of automation in reducing manual effort while ensuring consistent, error-free deployments.

Challenges and Solutions

During the project, I encountered an issue where the static website would download the index.html file instead of rendering it directly in the browser. This problem occurred because the Content-Type metadata for the file was not being set correctly during deployment. By default, when files are uploaded to an S3 bucket without specifying metadata, Amazon S3 may assign a generic content type such as application/octet-stream, which prompts browsers to treat the file as a binary object and download it instead of displaying it. This was a critical issue, as the index.html file serves as the main entry point for the static website, and improper metadata configuration prevented the website from functioning as intended.

To resolve this issue, I explicitly set the correct Content-Type for the index.html file during deployment. I modified the upload command in the pipeline to include the --content-type "text/html" option when using the AWS CLI aws s3 cp command. The updated command looked like this:

```
aws s3 cp index.html s3://<bucket-name>/ --content-type "text/html"
```

By explicitly specifying the content type as text/html, I ensured that the file would be correctly interpreted as an HTML document by web browsers. This small but crucial adjustment allowed the browser to render the web page instead of prompting a download. After re-running the deployment and verifying the S3 bucket's metadata settings, the website began to display correctly, confirming that the issue had been resolved.

Another challenge I encountered was related to permissions on the S3 bucket. Initially, when I tried to access the static website hosting URL, I received an Access Denied error. This occurred because the default configuration of an Amazon S3 bucket restricts public access for security purposes. While this default setting is a best practice for protecting sensitive data, it needs to be adjusted for hosting a public static website.

To resolve this issue, I updated the S3 bucket policy to allow public read access to the objects within the bucket. I created a JSON-based bucket policy that granted s3:GetObject permissions to all users, ensuring the website files could be accessed by anyone through the browser. The policy looked like this:

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "PublicReadGetObject",
   "Effect": "Allow",
   "Principal": "*",
   "Action": "s3:GetObject",
   "Resource": "arn:aws:s3:::<bucket-name>/*"
  }
 ]
}
```

After applying this bucket policy, I verified that the block public access settings were adjusted to allow public access where necessary. I then tested the Static Website Hosting URL to confirm that the index.html file and any additional resources were accessible. Once the policy took effect, the website became publicly viewable, and the permissions issue was fully resolved.

These two challenges highlighted the importance of understanding both metadata management and permissions configuration when working with Amazon S3. The content-type issue reinforced the need for careful attention to file metadata during deployments to ensure proper browser interpretation. Meanwhile, the permissions problem underscored the significance of balancing security with accessibility when hosting public content. Resolving these issues not only enhanced my technical knowledge of AWS services but also emphasized the importance of thorough testing and validation at every stage of the deployment process. By addressing these challenges systematically, I ensured that the pipeline functioned reliably and the static website was accessible to users as intended.

Conclusion

This project successfully demonstrates the implementation of a fully automated Continuous Integration and Continuous Deployment (CI/CD) pipeline using a suite of AWS services. By integrating GitHub for version control, AWS CodePipeline for orchestration, and Amazon S3 for hosting, I was able to create an end-to-end automated workflow that eliminates manual intervention and ensures code updates are seamlessly retrieved, processed, and deployed. The project not only streamlines the deployment process but also showcases the transformative impact of automation on modern software development workflows.

The pipeline begins with GitHub, where the application source code resides. GitHub serves as the single source of truth, enabling version control, code collaboration, and tracking of changes. By connecting the repository to AWS CodePipeline, any updates made to the codebase—such as changes to the index.html file—are automatically detected and trigger the pipeline. This integration ensures that the deployment process begins immediately when changes are pushed to the repository, creating a highly responsive and efficient workflow.

AWS CodePipeline serves as the core orchestrator of the deployment process, managing each stage of the CI/CD workflow. The pipeline is structured into distinct stages—Source, Build, and Deploy—each of which plays a crucial role in ensuring the reliability and consistency of the deployment process. In the Source stage, CodePipeline retrieves the updated source code from the connected GitHub repository. This step ensures that the pipeline always works with the latest version of the codebase.

The Build stage is handled by AWS CodeBuild, which simulates the process of preparing the application for deployment. Although the focus of this project is on deploying a static HTML website that does not require compilation, CodeBuild remains an essential component of the pipeline. It demonstrates the pipeline's extensibility and readiness to handle more complex workflows, such as compiling source code, running tests, or packaging applications. Including a build stage highlights how the pipeline can be easily adapted to accommodate dynamic applications, backend services, or additional testing requirements in the future.

In the final Deploy stage, the updated files are transferred to Amazon S3, which serves as the hosting environment for the static website. Amazon S3 was configured to support Static Website Hosting, ensuring that the website is accessible via a unique URL. This deployment step completes the pipeline, ensuring that updates to the source code are automatically reflected on the hosted website. The ability to seamlessly deploy changes to S3 demonstrates the power of AWS integrations and highlights the efficiency of cloud-based deployment solutions.

The successful implementation of this automated pipeline underscores the critical role of automation in modern software development workflows. Traditional deployment processes often involve manual tasks, such as copying files, configuring servers, or validating changes, which can introduce errors and inefficiencies. By automating these steps through a CI/CD pipeline, this project eliminates manual effort, reduces the risk of errors, and accelerates the deployment process. Automation ensures that code changes are deployed quickly, consistently, and reliably, enabling development teams to focus on innovation rather than routine tasks.

Furthermore, this project provides a scalable template that can be extended to more complex applications in the future. While this implementation focuses on a simple static website, the same principles and tools can be applied to more advanced scenarios, such as deploying containerized applications using AWS ECS, orchestrating microservices, or implementing automated testing frameworks. The modular nature of the pipeline allows for easy customization, making it suitable for projects of varying sizes and complexities.

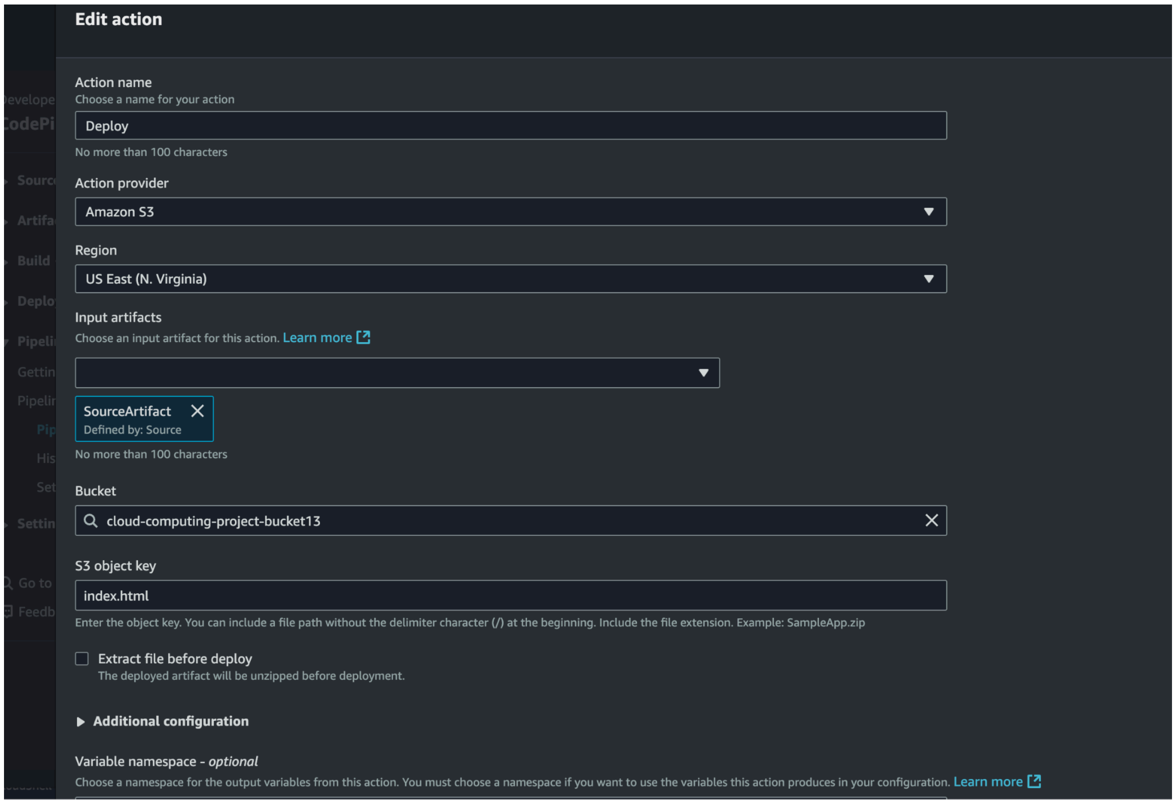
In conclusion, this project highlights the value of automation in streamlining software deployment workflows. By integrating GitHub, AWS CodePipeline, AWS CodeBuild, and Amazon S3, I was able to create a fully automated system that retrieves, builds, and deploys updates efficiently and reliably. The successful execution of this pipeline not only demonstrates its practical utility but also provides a foundation for building more sophisticated and scalable CI/CD workflows in the future. This project reinforces the importance of adopting automation to improve productivity, minimize errors, and deliver consistent results in modern software development environments.

Video Links

- Summary:
https://harvard.zoom.us/rec/share/GQtGnh6j9mTjLcCu2PLJ0BqEvcvVT5WP0QGvnrEzZOdZ0TG8zHswUAcSpE_zm6WL.aEE9IGBHnkK5_n_W?startTime=1734038587000

- Full Presentation:
https://harvard.zoom.us/rec/share/GQtGnh6j9mTjLcCu2PLJ0BqEvcvVT5WP0QGvnrEzZOdZ0TG8zHswUAcSpE_zm6WL.aEE9IGBHnkK5_n_W?startTime=1734038958000

Screenshots



Edit action

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

Action provider
Amazon S3

Region
US East (N. Virginia)

Input artifacts
Choose an input artifact for this action. Learn more
SourceArtifact
Defined by: Source
No more than 100 characters

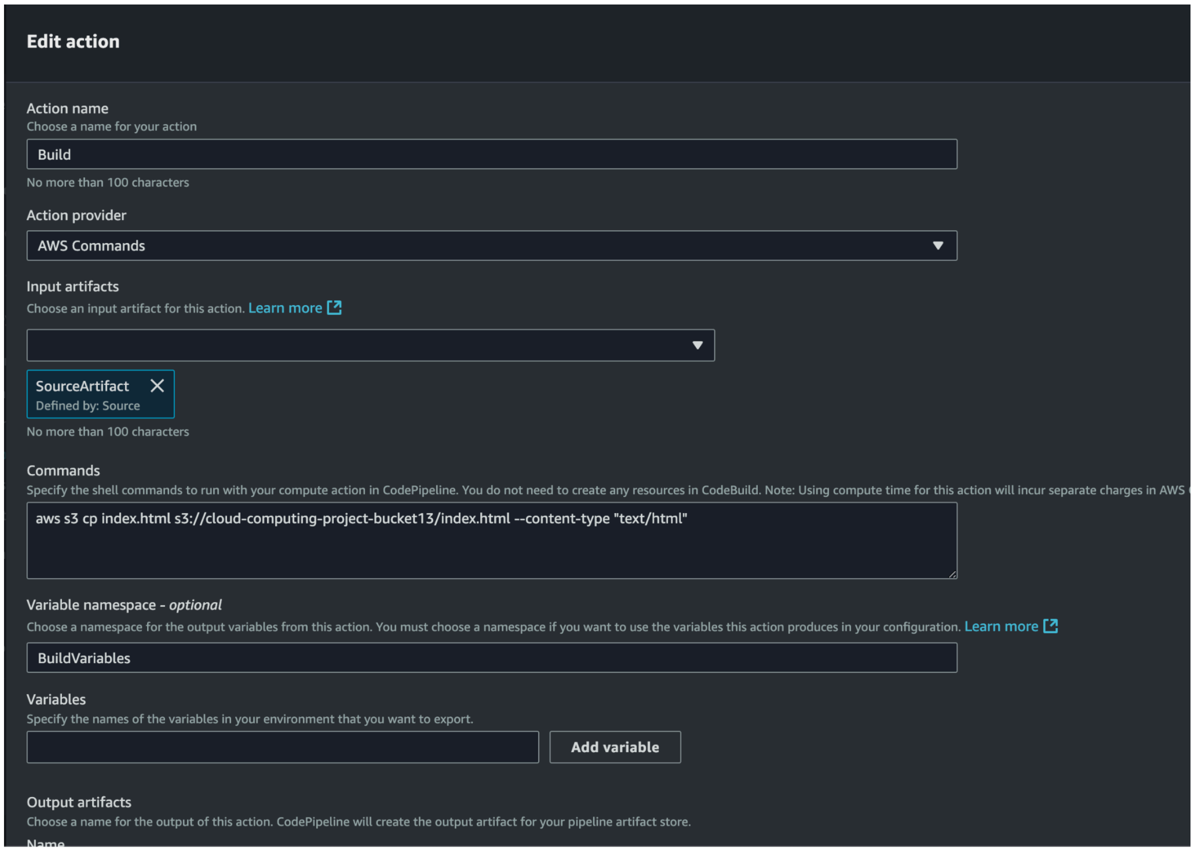
Bucket
cloud-computing-project-bucket13

S3 object key
index.html
Enter the object key. You can include a file path without the delimiter character (/) at the beginning. Include the file extension. Example: SampleApp.zip

Extract file before deploy
The deployed artifact will be unzipped before deployment.

Additional configuration

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



Edit action

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

Action provider
AWS Commands

Input artifacts
Choose an input artifact for this action. Learn more
SourceArtifact
Defined by: Source
No more than 100 characters

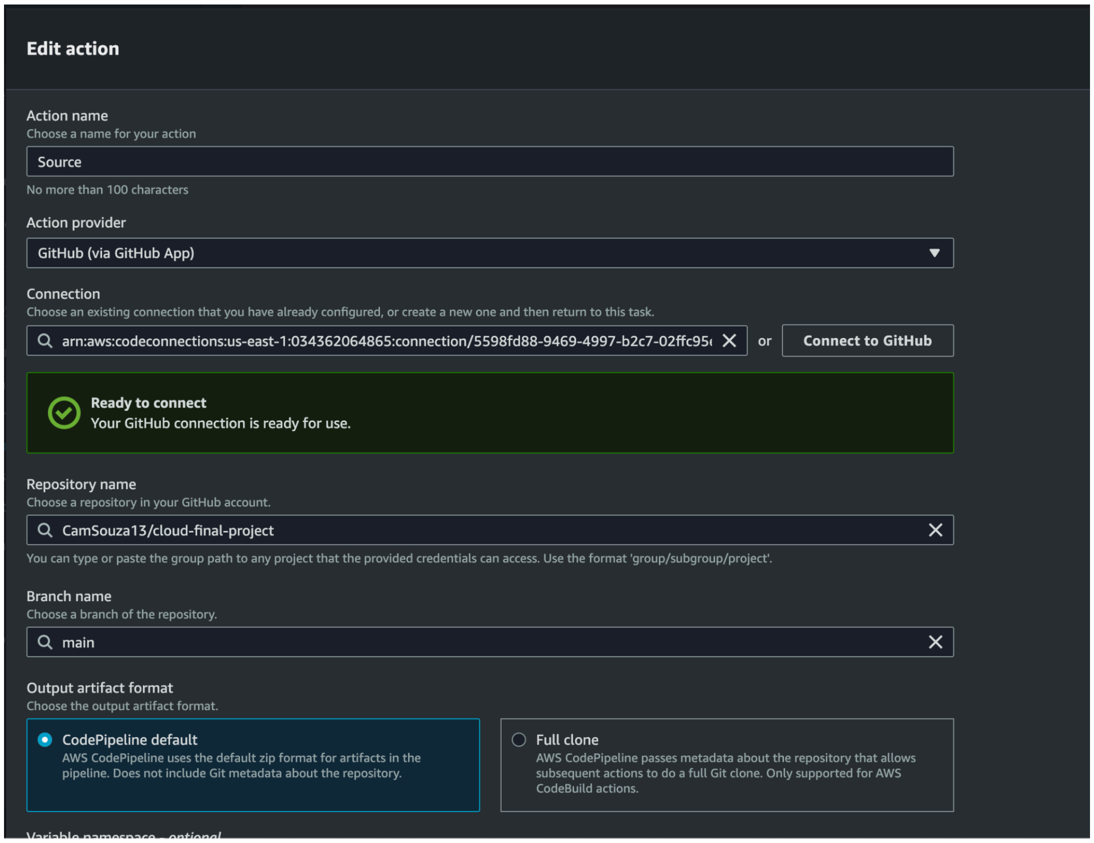
Commands
Specify the shell commands to run with your compute action in CodePipeline. You do not need to create any resources in CodeBuild. Note: Using compute time for this action will incur separate charges in AWS

```
aws s3 cp index.html s3://cloud-computing-project-bucket13/index.html --content-type "text/html"
```

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

Variables
Specify the names of the variables in your environment that you want to export.
Add variable

Output artifacts
Choose a name for the output of this action. CodePipeline will create the output artifact for your pipeline artifact store.
Name



Edit action

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

Action provider
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:us-east-1:034362064865:connection/5598fd88-9469-4997-b2c7-02ffc95
or

Ready to connect
Your GitHub connection is ready for use.

Repository name
Choose a repository in your GitHub account.
CamSouza13/cloud-final-project
You can type or paste the group path to any project that the provided credentials can access. Use the format 'group/subgroup/project'.

Branch name
Choose a branch of the repository.
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.