# AWS Cloudformation stack setup

# Table of Content

**CONTENTS**

Student’s Declaration…………………………………………………………………...i

Abstract…………………...……………………………………………………………ii

List Of Tables …………..……………………………………………………………..iii

Chapter 1- INTRODUCTION …..…………………………………………………….7

* 1. Introduction to Cloud Computing
  2. Manual Work and Automation in Cloud
  3. Objective
  4. Detail of software used

Chapter-2 INTRODUCTION TO THE PROJECT....…………………………….........12

2.1 Company’s Vision and Mission

2.2 Origin and growth of the company

2.3 Various departments and their functions

Chapter-3 BRIEF DESCRIPTION OF WORK DONE.………………………….........18

3.1 Activities and Equipment Handled

3.2 Challenges Faced and Solutions

3.3 Learning Outcomes

3.4 Data Analysis

Chapter-4 Review of Literature ....……………………………………………….........21

4.1 Technical Feasibility

4.2 Economical Feasibility

4.3 Operational Feasibility

Chapter 5- Implementation of project ……...……………………………….................23

5.1 Create a GitHub Repository

5.2 Create the CloudFormation Template

5.3 Deploy the CloudFormation Stack

Chapter-6 Results and Discussions ……………….……………………………………33

Final Chapter: Conclusion and Future Scope …………………………………………..38

References………...……………………………………………………………………..40

# Chapter-1 INTRODUCTION

## 1.1 Introduction to Cloud Computing

Cloud computing is the delivery of various services over the internet, including storage, computing power, databases, networking, software, and more. Instead of owning and maintaining physical data centers and servers, organizations can access these resources on-demand from cloud providers like Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform.

### **Importance in Modern IT Infrastructure**

Cloud computing has revolutionized the IT landscape, enabling organizations to scale their operations rapidly, reduce costs, and innovate faster. By leveraging cloud services, companies can focus on their core business activities without the burden of managing complex IT infrastructure.

### **Benefits of Cloud Computing**

* **Cost Efficiency:** Pay only for the resources you use, reducing capital expenditure on hardware and software.
* **Scalability:** Easily scale up or down based on demand, ensuring optimal resource utilization.
* **Flexibility:** Access a wide range of services and tools to support diverse business needs.
* **Disaster Recovery:** Enhance data security and disaster recovery capabilities with cloud-based solutions.
* **Collaboration:** Enable seamless collaboration among teams with cloud-based applications and data sharing.

## 1.2 Manual Work and Automation in Cloud Computing

### **Traditional Manual IT Processes**

In traditional IT environments, many processes are manual, requiring significant time and effort. Tasks such as server provisioning, software deployment, and configuration management often involve manual interventions, leading to potential errors, delays, and inefficiencies.

### **Introduction to Automation**

Automation in cloud computing refers to the use of tools and services to perform tasks with minimal human intervention. By automating repetitive and time-consuming processes, organizations can achieve higher efficiency, consistency, and reliability in their operations.

### **How Automation Enhances Cloud Computing**

Automation in the cloud streamlines operations, reduces the risk of human error, and allows IT teams to focus on more strategic initiatives. Automated processes can include infrastructure provisioning, application deployment, monitoring, and scaling, among others. This not only speeds up workflows but also ensures consistent and reliable outcomes.

## 1.3 Objective

In this project, the aim is to streamline the setup process by implementing **a CloudFormation stack** that **automates the creation of a CodePipeline** and **all related resources**. This approach will significantly decrease the time required for setting up individual CodePipelines across multiple environments, enhancing efficiency and ensuring consistency in deployment processes. This approach minimizes the chances of errors and constant improvements by automating repetitive tasks. By creating a streamlined process, applications can be built and deployed effortlessly, reducing manual effort and enhancing efficiency. Automation aims to make the entire development and deployment process smooth and fast, significantly cutting down the time required to bring an application from development to production.

Additionally, the project places a strong emphasis on security. By using AWS Identity and Access Management (IAM) users, it ensures that permissions are carefully controlled and restricted, allowing only authorized users to perform specific actions within the application development and deployment process. This focus on security helps safeguard the process and maintains the integrity of the application throughout its lifecycle.

CI/CD of WebApp helps to develop and deploy continuously so that there is less scope of improvement and error as well. It helps in creating a well-tailored process, so that no work is executed repeatedly and applications can be built and deployed easily. It aims to make the whole process of application development and deployment automated and effortless. It aims to lower the time taken to execute the application that is on the verge of creation. It also focuses on security of the whole process with the help of IAM users in AWS which restricts what permissions are allowed to which user in order to operate on the process of application development and deployment.

### **Scope**

This project is extensible and has way too many possibilities, there are many other tools and stages that can be added with this project to make it even more dynamic and end-to-end. The scope for the same is limitless because this is a concept and there are limitless possibilities to integrate it with. Overall, the scope encompasses the development and deployment of a web application, with a focus on user satisfaction, efficiency through automation with CI/CD, and contributing to the advancement of software development practices through sharing experiences and insights.

## 1.4 Detail of software used

**Key AWS Services in Automation**

### **AWS CloudFormation**

**Overview:** AWS CloudFormation is a service that provides a way to model and set up Amazon Web Services resources using infrastructure as code (IaC). This allows users to **create, update, and delet**e a collection of related AWS resources in a predictable and orderly fashion.

**Benefits:**

* Simplifies infrastructure management by using templates to define and provision resources.
* Ensures consistency across different environments.
* Reduces manual errors and operational overhead.

**Use Cases:**

* Automated resource provisioning.
* Version control for infrastructure configurations.
* Replicating environments for development, testing, and production.

### **Amazon S3 (Simple Storage Service)**

**Overview:** Amazon S3 is a scalable object storage service that provides secure, durable, and highly available storage. It is commonly used for backup, archival, big data analytics, and as a primary storage for cloud-native applications.

**Benefits:**

* Highly scalable and reliable storage solution.
* Cost-effective with a pay-as-you-go pricing model.
* Integrated with a wide range of AWS services and third-party tools.

**Use Cases:**

* Storing and retrieving any amount of data from anywhere.
* Hosting static websites.
* Backup and disaster recovery.

### **AWS CodePipeline**

**Overview:** AWS CodePipeline is a continuous integration and continuous delivery (CI/CD) service for fast and reliable application and infrastructure updates. It automates the build, test, and deploy phases of your release process every time there is a code change.

**Benefits:**

* Streamlines the software release process.
* Increases development speed and efficiency.
* Integrates with various AWS services and third-party tools.

**Use Cases:**

* Automating software release pipelines.
* Ensuring consistent and reliable deployments.
* Integrating with version control systems like GitHub.

### **Integration with GitHub**

**Overview:** GitHub is a web-based platform for version control and collaboration. It allows multiple people to work on projects simultaneously, track changes, and manage different versions of code.

**Benefits:**

* Facilitates collaboration among developers.
* Integrates seamlessly with AWS services for CI/CD.
* Provides robust version control and code management features.

**Use Cases:**

* Source code management.
* Collaborative development.
* Integration with CI/CD pipelines.

# Chapter-2 INTRODUCTION TO THE PROJECT

## 2.1 Company’s Vision and Mission

**VISION**

Firminiq aims to revolutionize the way businesses operate in the digital age by providing innovative, technology-driven solutions. Our vision is to empower organizations to achieve unprecedented levels of efficiency, productivity, and growth through the seamless integration of cutting-edge technologies.

**MISSION**

Our mission is to deliver high-quality, customizable digital solutions that cater to each client's unique needs. We are committed to fostering long-term partnerships by consistently exceeding expectations and providing exceptional value. Our focus is on continuous innovation, operational excellence, and customer satisfaction.

## 2.2 Origin and growth of the company

**Origin**:

Firminiq was founded in 2015 by a group of technology enthusiasts with a shared passion for digital transformation. The founders recognized a growing need for businesses to adapt to the rapidly changing technological landscape and sought to provide comprehensive solutions to meet this demand.

**Growth**:

Since its inception, Firminiq has experienced remarkable growth. Initially starting as a small startup, the company has expanded its operations globally, serving clients across various industries. Our growth trajectory has been fueled by our dedication to innovation, a deep understanding of market trends, and an unwavering commitment to client success.

## 2.3 Various departments and their functions

**1. Research and Development (R&D):**

**Function**: Focuses on the development of new technologies and the improvement of existing products. The R&D team ensures that Firminiq stays ahead of industry trends and continuously innovates.

**Key Activities**: Conducting market research, developing prototypes, testing new technologies, and collaborating with other departments to integrate new solutions.

**2. Sales and Marketing:**

**Function**: Responsible for promoting Firminiq's products and services, acquiring new clients, and maintaining relationships with existing clients.

**Key Activities:** Market analysis, advertising campaigns, sales strategy development, client presentations, and customer relationship management.

**3. Operations:**

**Function**: Manages the day-to-day activities of the company, ensuring that all departments operate smoothly and efficiently.

**Key Activities:** Resource allocation, process optimization, logistics management, and quality control.

**4. Customer Support:**

**Function**: Provides assistance to clients, addressing their concerns and ensuring they derive maximum value from Firminiq's products and services.

**Key Activities:** Technical support, customer service, troubleshooting, and feedback collection.

**5. Human Resources (HR):**

**Function**: Manages employee-related functions including recruitment, training, and development.

**Key Activities:** Hiring, onboarding, performance management, employee engagement, and compliance with labor laws.

**6. Finance:**

**Function**: Oversees the financial health of the company, managing budgets, investments, and financial planning.

**Key Activities:** Accounting, financial reporting, budgeting, and financial analysis.

**7. IT and Infrastructure:**

**Function**: Maintains the company's IT infrastructure, ensuring all systems and networks are secure and operational.

**Key Activities:** Network management, cybersecurity, software development, and system maintenance.

Organization Chart of the Company

A screenshot of a computer

Description automatically generated

#### **UI/UX Design**

**Function:** The UI/UX Design department focuses on creating intuitive and engaging user interfaces and experiences for Firminiq's products and services. This team ensures that the end-users have a seamless and enjoyable interaction with the company's digital solutions.

**Key Activities:**

* **User Research:** Conducting user research to understand user needs, behaviors, and preferences.
* **Wireframing and Prototyping:** Creating wireframes and prototypes to visualize design concepts.
* **User Interface Design:** Designing visually appealing and functional user interfaces.
* **User Experience Design:** Developing user experience strategies to enhance usability and satisfaction.
* **Usability Testing:** Conducting usability tests to gather feedback and make necessary improvements.

#### **Mobile Development**

**Function:** The Mobile Development department is responsible for creating and maintaining mobile applications for various platforms, including iOS and Android. This team ensures that Firminiq's mobile solutions are robust, user-friendly, and aligned with client requirements.

**Key Activities:**

* **App Development:** Writing code and developing mobile applications for different platforms.
* **Platform Optimization:** Ensuring applications are optimized for performance across various devices.
* **Integration:** Integrating mobile applications with backend services and third-party APIs.
* **Testing and Debugging:** Performing rigorous testing to identify and fix bugs.
* **App Store Deployment:** Managing the deployment of applications to app stores and handling updates.

#### **Web Development**

**Function:** The Web Development department is responsible for building and maintaining websites and web applications. This team ensures that Firminiq's web solutions are scalable, secure, and provide a seamless user experience.

**Key Activities:**

* **Front-End Development:** Creating responsive and interactive user interfaces using HTML, CSS, and JavaScript.
* **Back-End Development:** Developing server-side logic, databases, and APIs to support web applications.
* **Content Management:** Implementing and managing content management systems (CMS) for easy content updates.
* **Web Security:** Ensuring websites are secure from threats and vulnerabilities.
* **Performance Optimization:** Optimizing website performance for faster load times and better user experience.

#### **AWS (Amazon Web Services)**

**Function:** The AWS department manages Firminiq's cloud infrastructure, leveraging Amazon Web Services to provide scalable and reliable cloud solutions. This team ensures that the company's cloud operations are efficient and cost-effective.

**Key Activities:**

* **Cloud Architecture:** Designing and implementing cloud architectures using AWS services.
* **Resource Management:** Managing cloud resources to ensure optimal performance and cost-efficiency.
* **Security and Compliance:** Ensuring cloud environments meet security and compliance standards.
* **Automation:** Automating cloud operations to improve efficiency and reduce manual efforts.
* **Monitoring:** Monitoring cloud infrastructure to detect and address issues proactively.

#### **DevOps**

**Function:** The DevOps department bridges the gap between development and operations, focusing on automating and streamlining the software delivery process. This team ensures continuous integration, continuous delivery, and infrastructure automation.

**Key Activities:**

* **CI/CD Pipeline:** Implementing continuous integration and continuous delivery pipelines for automated deployments.
* **Infrastructure as Code:** Managing infrastructure using code to automate provisioning and configuration.
* **Configuration Management:** Ensuring consistency and efficiency in configuration across environments.
* **Monitoring and Logging:** Setting up monitoring and logging to track system performance and identify issues.
* **Collaboration:** Facilitating collaboration between development and operations teams for smoother workflows.

#### **Monitoring & Analytics**

**Function:** The Monitoring & Analytics department focuses on tracking and analyzing the performance of Firminiq's digital solutions. This team provides insights and data to drive informed decision-making and continuous improvement.

**Key Activities:**

* **Performance Monitoring:** Monitoring the performance of applications and infrastructure to ensure optimal operation.
* **Data Collection:** Collecting data from various sources for analysis.
* **Analytics:** Analyzing data to identify trends, patterns, and insights.
* **Reporting:** Creating reports and dashboards to present data and findings to stakeholders.
* **Alerting:** Setting up alerts to notify teams of potential issues or anomalies.

#### **Quality Engineering**

**Function:** The Quality Engineering department is responsible for ensuring that Firminiq's products and services meet the highest standards of quality. This team focuses on testing, quality assurance, and continuous improvement.

**Key Activities:**

* **Test Planning:** Developing test plans and strategies to cover all aspects of product testing.
* **Manual Testing:** Conducting manual tests to identify bugs and issues.
* **Automated Testing:** Implementing automated tests to increase testing efficiency and coverage.
* **Performance Testing:** Testing the performance and scalability of applications.
* **Quality Assurance:** Ensuring that products meet defined quality standards before release.
* **Continuous Improvement:** Identifying areas for improvement and implementing changes to enhance product quality.

# Chapter-3 BRIEF DESCRIPTION OF WORK DONE

### **Weekly Report**

#### **Week 1: AWS CloudFormation Setup**

* **Objective**: Set up a CloudFormation stack to automate the creation of AWS resources.
* **Tasks Completed**:
  + Created and configured a GitHub repository to store the CloudFormation templates.
  + Developed a CloudFormation template to automate the creation of an S3 bucket and a CodePipeline.
  + Implemented automation to trigger the CodePipeline, pull data from the GitHub repository, and store it in the S3 bucket.
* **Outcome**: Streamlined the setup process, reducing setup time from several hours to mere minutes, enhancing efficiency and ensuring consistency in deployment processes.

#### **Week 2: AWS IAM Policy Management**

* **Objective**: Create and manage IAM policies for the frontend team.
* **Tasks Completed**:
  + Reviewed and created read policies for the frontend team, ensuring appropriate permissions and security.
  + Managed and updated existing IAM policies, addressing the issue of policies reaching the maximum version limit.
* **Outcome**: Improved security and access management, ensuring that only authorized users have the necessary permissions.

#### **Week 3: AWS CLI Scripts for Security Group Management**

* **Objective**: Automate the management of security group rules using AWS CLI.
* **Tasks Completed**:
  + Developed AWS CLI scripts to add and update inbound rules with descriptions.
  + Tested and validated the scripts to ensure they functioned as expected, streamlining the process of managing security group rules.
* **Outcome**: Enhanced efficiency in managing security group rules, reducing the potential for manual errors and ensuring consistent rule application.

#### **Week 4: Resource Tracking and Monitoring Automation**

* **Objective**: Automate the tracking and monitoring of resource creation and deletion.
* **Tasks Completed**:
  + Created a script in AWS CloudShell to run daily, gather information on creating and deleting various AWS resources (EC2, AMI, snapshot, S3, IAM roles, CodePipelines, CloudFormation).
  + Implemented a Lambda function to track S3 bucket creations and deletions that occurred today, uploading this information into an S3 bucket.
* **Outcome**: Automated the resource tracking process, providing up-to-date information on resource changes and improving monitoring and reporting capabilities.

## 3.1. Activities and Equipment Handled

As a Cloud DevOps Trainee, I was responsible for a variety of tasks and activities that primarily involved working with AWS services, GitHub, and Excel. Here’s a detailed breakdown:

* **AWS EC2 (Elastic Compute Cloud)**: Managed virtual servers to run applications, with tasks including instance setup, monitoring, and scaling.
* **AWS S3 (Simple Storage Service)**: Handled storage and retrieval of data, including setting up buckets, managing permissions, and implementing lifecycle policies.
* **AWS Lambda**: Developed and deployed serverless functions to automate various tasks, focusing on event-driven programming.
* **AWS CloudFormation**: Created and managed infrastructure as code, automating the provisioning of AWS resources using templates.
* **AWS CodePipeline**: Implemented CI/CD pipelines to automate the build, test, and deploy phases of application development.
* **GitHub**: Managed source code repositories, collaborated with team members through pull requests and issue tracking, and integrated GitHub with AWS CodePipeline.
* **Excel**: Utilized for data analysis, tracking project progress, and reporting purposes.

## 3.2. Challenges Faced and Solutions

Throughout my training, I encountered several challenges, which I tackled with specific strategies:

* **YAML Code Not Working**: YAML syntax errors or misconfigurations were common. To resolve these, I used YAML validators and thoroughly reviewed documentation to ensure correct formatting and parameter usage.
* **Access Issues**: Frequently faced permission errors due to improper IAM role configurations. I resolved these by carefully managing IAM policies, conducting regular access reviews, and ensuring the principle of least privilege.
* **Data Backup**: Ensured regular backups of all critical data using AWS S3 and automated snapshot tools for EC2 instances. Implemented lifecycle rules to archive or delete old data.
* **Cost Optimization**: Monitored AWS costs using AWS Cost Explorer and implemented cost-saving measures such as reserved instances, auto-scaling policies, and identifying unused resources for termination.
* **Automating Manual Work**: Reduced manual intervention by scripting repetitive tasks using AWS Lambda and integrating various AWS services through automation tools like CloudFormation and CodePipeline.

## 3.3. Learning Outcomes

By the end of my training, I achieved significant learning outcomes:

* **CloudFormation Stack Implementation**: Developed a CloudFormation stack to automate the creation of a CodePipeline and associated resources. This approach streamlined the setup of CI/CD pipelines across multiple environments, ensuring consistency and reducing setup time.
* **Efficiency and Consistency**: Enhanced the efficiency of deployment processes by automating infrastructure provisioning and configuration management, leading to consistent and repeatable environments.
* **Enhanced Problem-Solving Skills**: Gained proficiency in troubleshooting and resolving complex issues related to cloud infrastructure and deployment pipelines.
* **Data Analysis Skills**: Leveraged Excel and AWS tools to perform data analysis, providing insights into resource utilization, performance metrics, and cost analysis.

## 3.4. Data Analysis

As part of my role, I conducted data analysis to derive meaningful insights and inform decision-making processes. Key activities included:

* **Resource Utilization Tracking**: Analyzed usage patterns of AWS resources to optimize performance and cost-efficiency.
* **Performance Metrics**: Monitored and reported on key performance indicators (KPIs) of applications and services deployed on AWS.
* **Cost Analysis**: Evaluated spending trends and identified areas for cost optimization, helping to implement cost-saving strategies effectively.

# Chapter-4 Review of Literature

## 4.1 Technical Feasibility

The technical requirements for this project involve accessing AWS services, grasping CI/CD principles and practices, and having a development environment that works well with AWS services. Upon examination, AWS tools exhibit strong capabilities for implementing CI/CD, providing smooth integration and automated functions. Compatibility with existing development frameworks and workflows reduces disruptions during integration, while AWS infrastructure's ability to scale enables accommodating growth and handling increased workloads. Nonetheless, there are potential risks such as compatibility issues, security vulnerabilities, and resource constraints that must be managed. *Williams, Phillip. “A survey on security in internet of things with a focus on the impact of emerging technologies.” Internet of Things, vol. 19, no. 100564, 2022, p. 24. Science Direct, https://www.sciencedirect.com/science/article/pii/S2542660522000592.*

Effective measures like thorough testing, security protocols, and regular monitoring of AWS services can help mitigate these risks. Having skilled personnel and the necessary ha

A diagram of a product management system

Description automatically generated

Figure 1: CICD workflow

## 4.2 Economical Feasibility

Integrating AWS tools into our software development workflow entails initial costs for subscription fees, and usage fees as it is pay-as-you-go services. There’s this one thing to notice that AWS has great services with minimal costing which is a favorable situation for the project. The scalability of AWS infrastructure offers flexibility to adapt to changing demands, potentially mitigating the need for costly infrastructure upgrades. Despite economic risks, such as unexpected cost overruns, regular monitoring and optimization can ensure economic viability. Investing in skill development and training further enhances long-term sustainability, maximising the economic benefits of AWS integration. Overall, while upfront investments are necessary, the potential for increased efficiency and scalability makes integrating AWS tools economically feasible and advantageous in the long run.

## 4.3 Operational Feasibility

This involves evaluating the readiness of personnel to adopt new tools and processes, the alignment of existing workflows with AWS services, and the likelihood of encountering resistance to change. Learning from comprehensive documentation will be essential to ensure that team members can proficiently use AWS tools and grasp their role within the CI/CD pipeline. Moreover, the flexibility of our development processes to incorporate AWS integration will be pivotal in minimising disruptions and maximising the benefits of automation and scalability. Addressing any reluctance to embrace change through effective communication and engaging stakeholders will be crucial in garnering support for the integration endeavour. Despite potential operational hurdles, the anticipated enhancements in efficiency, collaboration, and software quality render the integration of AWS tools highly feasible and advantageous for our organisation's operations.

# Chapter-5 Implementation of project

## 5.1 Create a GitHub Repository

**Step 1: GitHub**

**1 Log in to GitHub**

2 **Create a new repository**

**3 Clone the repository to your local machine**:

**Step 2: Add Your Files**

**1 Create your files**

2 **Commit and push the changes**:

A black background with white text

Description automatically generated

Figure 3: Committing the changes

A screenshot of a computer

Description automatically generated

Figure 2: Repository data

**Step 3: Create a GitHub Personal Access Token**

* Go to your GitHub account settings and navigate to Developer settings > Personal access tokens.
* Click Generate new token.
* Select scopes: repo and admin:repo\_hook.
* Click Generate token and save the token somewhere secure.

A screenshot of a computer

Description automatically generated

Figure 4: GitHub Personal Access Token

## 5.2 Create the CloudFormation Template

**Overview**

Here's a CloudFormation template that sets up a CodePipeline with related resources such as S3 bucket for storing the artifacts, IAM roles, and a sample CodeBuild project.

AWSTemplateFormatVersion: '2010-09-09'

Description: Stack for deploying a resources using CodePipeline and GitHub

Parameters:

  GitHubToken:

    Type: String

    NoEcho: true

    Description: "GitHub OAuth Token"

  GitHubUser:

    Type: String

    Description: "GitHub Username"

  GitHubRepo:

    Type: String

    Description: "GitHub Repository Name"

  GitHubBranch:

    Type: String

    Default: "main"

    Description: "GitHub Branch Name"

Resources:

  WebsiteBucket:

    Type: 'AWS::S3::Bucket'

    Properties:

      BucketName: !Sub "${AWS::StackName}-website-bucket"

      WebsiteConfiguration:

        IndexDocument: index.html

        ErrorDocument: error.html

      PublicAccessBlockConfiguration:

        BlockPublicAcls: false

        IgnorePublicAcls: false

        BlockPublicPolicy: false

        RestrictPublicBuckets: false

  CodePipelineServiceRole:

    Type: 'AWS::IAM::Role'

    Properties:

      AssumeRolePolicyDocument:

        Version: '2012-10-17'

        Statement:

          - Effect: Allow

            Principal:

              Service: codepipeline.amazonaws.com

            Action: sts:AssumeRole

      Path: "/"

      Policies:

        - PolicyName: CodePipelineServicePolicy

          PolicyDocument:

            Version: '2012-10-17'

            Statement:

              - Effect: Allow

                Action:

                  - "s3:CreateBucket"

                  - "s3:DeleteBucket"

                  - "s3:PutObject"

                  - "s3:GetObject"

                  - "s3:DeleteObject"

                  - "s3:ListBucket"

                Resource: "\*"

  CodePipeline:

    Type: 'AWS::CodePipeline::Pipeline'

    Properties:

      RoleArn: !GetAtt CodePipelineServiceRole.Arn

      ArtifactStore:

        Type: S3

        Location: !Ref WebsiteBucket

      Stages:

        - Name: Source

          Actions:

            - Name: Source

              ActionTypeId:

                Category: Source

                Owner: ThirdParty

                Provider: GitHub

                Version: 1

              OutputArtifacts:

                - Name: SourceOutput

              Configuration:

                Owner: !Ref GitHubUser

                Repo: !Ref GitHubRepo

                Branch: !Ref GitHubBranch

                OAuthToken: !Ref GitHubToken

              RunOrder: 1

        - Name: Deploy

          Actions:

            - Name: Deploy

              ActionTypeId:

                Category: Deploy

                Owner: AWS

                Provider: S3

                Version: 1

              InputArtifacts:

                - Name: SourceOutput

              Configuration:

                BucketName: !Ref WebsiteBucket

                Extract: true

              RunOrder: 1

Outputs:

  WebsiteURL:

    Value: !GetAtt WebsiteBucket.WebsiteURL

    Description: URL for the static website

By using this template you can also create a static website by uploading the all the resources in github & enabling static website S3 bucket.

This AWS CloudFormation template sets up a stack to create a resources using CodePipeline and GitHub. It includes resources for an S3 bucket to host the website, an IAM role for CodePipeline, and the pipeline itself.

**Parameters**

These are the inputs required when creating the stack:

**GitHubToken**: A secure string parameter (hidden during input) that stores the GitHub OAuth token used for accessing the repository.

**GitHubUser**: A string parameter for the GitHub username.

**GitHubRepo**: A string parameter for the GitHub repository name.

**GitHubBranch**: A string parameter for the branch name in the GitHub repository, defaulting to "main".

**Resources**

These are the AWS resources created by the template:

1. **WebsiteBucket** (S3 Bucket): Bucket name

Type: AWS::S3::Bucket

**Properties**:

**BucketName**: Dynamically generated using the stack name.

**WebsiteConfiguration**: Specifies the index and error documents for the static website.

**PublicAccessBlockConfiguration**: Configures the bucket to allow public access by disabling various public access blocks.

2. **CodePipelineServiceRole** (IAM Role): creating a role for permission.

Type: AWS::IAM::Role

**Properties**:

**AssumeRolePolicyDocument**: Allows the CodePipeline service to assume this role.

**Policies**: Grants permissions to interact with S3, including creating, deleting, and managing objects and buckets.

3. **CodePipeline** (Pipeline)

**Type**: AWS::CodePipeline::Pipeline

**Properties**:

**RoleArn**: The ARN of the IAM role for the pipeline.

**ArtifactStore**: Specifies an S3 bucket (WebsiteBucket) to store artifacts.

**Stages**: Defines the stages of the pipeline:

**Source Stage**: Retrieves the source code from the specified GitHub repository.

**Actions**:

**Name**: Source

**ActionTypeId**: Specifies this action as a third-party source action using GitHub.

**OutputArtifacts**: Defines the output artifact from this stage.

**Configuration**: Configures GitHub repository details, including owner, repo name, branch, and OAuth token.

**Deploy Stage**: Deploys the retrieved source code to the S3 bucket.

**Actions**:

**Name**: Deploy

**ActionTypeId**: Specifies this action as a deploy action using S3.

**InputArtifacts**: Defines the input artifact for this stage.

**Configuration**: Configures the S3 bucket details and extraction settings.

**Outputs**

**WebsiteURL**: Outputs the URL of the static website hosted in the S3 bucket.

This CloudFormation template automates the setup of a pipeline for deploying a static website hosted on S3, using CodePipeline to integrate with GitHub for source control. The pipeline retrieves the website files from a specified GitHub repository and deploys them to an S3 bucket configured for website hosting.

## 5.3 Deploy the CloudFormation Stack

**Upload your Yaml Template in a S3 bucket & Copy the URL as shown in the figurethe.**

A screenshot of a computer

Description automatically generated

Figure 5: Yaml Template

**Open Cloudformation Stack**

* **Paste the URL**

A screenshot of a computer

Description automatically generated

Figure 6: Creating a Stack

**Specify Stack Details.**

A screenshot of a computer

Description automatically generated

Figure 6: Stack Details

**After that Create a Stack. If there is any issue Troubleshoot it**

**Troubleshooting issue if your template is not working:**

Here are a few steps you can take to troubleshoot the issue:

1. **Review the CloudFormation Template**: Double-check your CloudFormation template for any syntax errors or discrepancies in parameter definitions.
2. **Check Input Parameters**: Ensure that the input parameters you're providing to CloudFormation match the expected format defined in your template. Pay close attention to data types, such as strings, numbers, lists, and maps.
3. **Validate Input Values**: Validate that the values you're providing for each parameter are within the allowed range and meet any constraints specified in the template.
4. **Look for Specific Error Messages**: If there's an error message accompanying the issue, it can provide valuable insights into what's causing the problem. Look for any specific guidance on what aspect of the input didn't match the expected format.
5. **Review AWS Documentation**: Refer to the AWS CloudFormation documentation for the resource you're trying to create. It often contains examples and detailed explanations of the expected input format.
6. **Test with Minimal Configuration**: If possible, try simplifying your CloudFormation template and input parameters to isolate the issue. Gradually add complexity back in until you identify the root cause.
7. **Seek Community Support**: If you're still unable to resolve the issue, consider reaching out to the AWS community forums or support for assistance. Others may have encountered similar issues and can provide helpful insights or solutions.

By following these steps and carefully reviewing your CloudFormation template and input parameters, you should be able to identify and address the formatting issue causing the creation problem.

# Chapter-6 Results and Discussions

The project demonstrates a successful implementation of a CI/CD pipeline using AWS CodePipeline. The following key points highlight the effectiveness of the setup: which orchestrates the stages from source retrieval to deployment. The process involved creating an S3 bucket to store the website files and configuring the CodePipeline to automate the workflow.

#### **Resources Created**

1. **CodePipeline**:
   * **Logical ID**: CodePipeline
   * **Physical ID**: my-project-test-CodePipeline-1350USW1QR92B
   * **Status**: CREATE\_COMPLETE
   * **Description**: This pipeline is configured to automate the stages from source to deployment, ensuring continuous integration and delivery.
2. **IAM Role**:
   * **Logical ID**: CodePipelineServiceRole
   * **Physical ID**: my-project-test-CodePipelineServiceRole-tkpkB8KYKZK
   * **Status**: CREATE\_COMPLETE
   * **Description**: This role provides necessary permissions for the CodePipeline to interact with other AWS services securely.
3. **S3 Bucket**:
   * **Logical ID**: WebsiteBucket
   * **Physical ID**: my-project-test-website-bucket
   * **Status**: CREATE\_COMPLETE
   * **Description**: This bucket stores the website files and facilitates their deployment via the pipeline.

#### **Pipeline Execution**

* **Source Stage**:
  + **Status**: Succeeded
  + **Details**: The source stage successfully retrieved the latest code from the configured source repository in GitHub. This confirms that the integration between CodePipeline and the version control system is functioning correctly.
* **Deploy Stage**:
  + **Status**: Succeeded
  + **Details**: The deployment stage successfully uploaded the files to the S3 bucket. This indicates that the deployment configuration is correct, and the website files are properly stored and updated in the S3 bucket.

#### **S3 Bucket Contents**

The S3 bucket named **my-project-test-website-bucket** contains the following files:

* **error.html**
  + **Type**: HTML
  + **Size**: 14.0 B
  + **Last Modified**: June 13, 2024
* **index.html**
  + **Type**: HTML
  + **Size**: 50.0 B
  + **Last Modified**: June 13, 2024
* **my-project-test-Code/**
  + **Type**: Folder
  + **Last Modified**: June 13, 2024
* **README.md**
  + **Type**: Markdown
  + **Size**: 42.0 B
  + **Last Modified**: June 13, 2024

These files confirm that the website content and necessary documentation are correctly stored in the S3 bucket, and ready to be served.

#### **Discussion**

The project demonstrates a successful implementation of a CI/CD pipeline using AWS CodePipeline. The following key points highlight the effectiveness of the setup:

1. **Automation**: The pipeline automates the deployment process, reducing manual intervention and potential errors.
2. **Integration**: Seamless integration with GitHub ensures that the latest code changes are automatically deployed.
3. **Security**: Using IAM roles ensures that the pipeline operates with the least privilege principle, enhancing security.
4. **Scalability**: The use of S3 for hosting the website files makes it easy to scale the website as needed.

Overall, the project showcases a robust and efficient deployment process, leveraging AWS services to maintain continuous delivery and integration. This setup is ideal for maintaining and deploying web applications with minimal downtime and maximum reliability.

: The pipeline automates the deployment process, reducing manual intervention and potential errors.

**Integration**: Seamless integration with GitHub ensures that the latest code changes are automatically deployed.

**Security**: Using IAM roles ensures that the pipeline operates with the least privilege principle, enhancing security.

**Scalability**: The use of S3 for hosting the website files makes it easy to scale the website as needed.

Overall, the project showcases a robust and efficient deployment process, leveraging AWS services to maintain continuous delivery and integration. This setup is ideal for maintaining and deploying web applications with minimal downtime and maximum reliability.

**You can check all your resources created by the Cloudformation stack.**

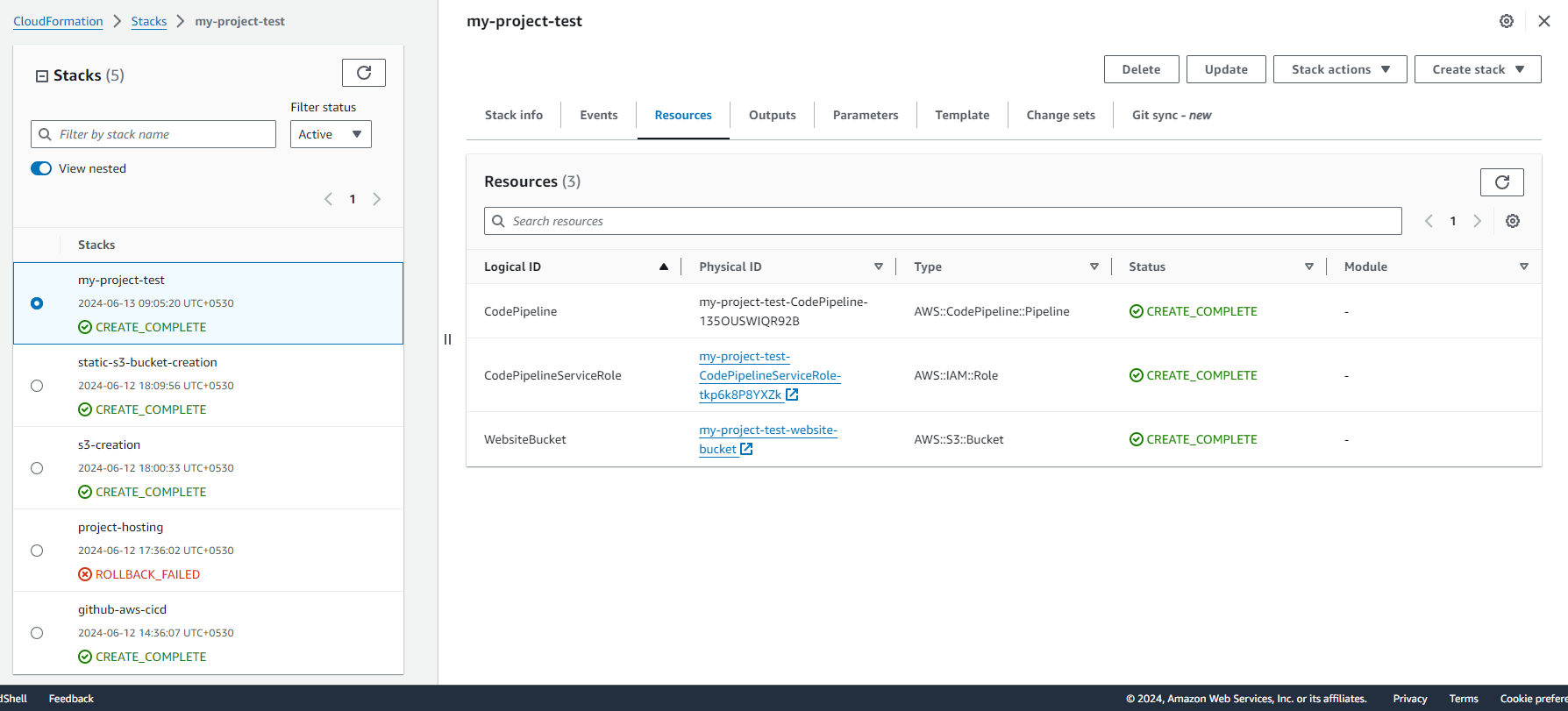


Figure 6: Showing Stack Resource Creation

**CodePipeline resources created:**

A screenshot of a computer

Description automatically generated

Figure 7: Showing CodePipeline creation

**S3 Bucket Creation & the resources updated which is stored in GitHub**

A screenshot of a computer

Description automatically generated

Figure 8: Showing S3 resource updated by GitHub

**IAM Resource Policy Creation**

A screenshot of a computer

Description automatically generated

Figure 9: CodePipeline policy creation using CloudFormation.

# Final Chapter: Conclusion and Future Scope

In this project, we have successfully demonstrated how to automate the deployment of a static website using AWS CloudFormation, CodePipeline, and GitHub. The CloudFormation template we developed provisions an S3 bucket configured for static website hosting and sets up an automated deployment pipeline. This pipeline integrates with GitHub to fetch the website's source code and deploy it to the S3 bucket whenever changes are pushed to the specified branch.

**Summary of Accomplishments**

**Automation**: We leveraged AWS CloudFormation to define the infrastructure as code, ensuring consistent and repeatable deployments.

**Integration**: By using AWS CodePipeline, we automated the build and deployment process, integrating seamlessly with GitHub for source control.

**Security**: The setup included secure handling of credentials, such as the GitHub OAuth token, and appropriate IAM roles and policies to limit permissions to what is necessary.

**Scalability**: The solution is designed to be scalable and can handle additional resources or more complex pipelines as needed.

**Future Scope**

The principles and techniques demonstrated in this project lay the foundation for further automation and enhancement of cloud-based infrastructures. Here are some potential future directions:

**Automating Additional Resources**: Beyond static websites, we can use CloudFormation to automate the deployment of various AWS resources such as databases, serverless functions, and containerized applications. This can help in creating a fully automated and scalable infrastructure.

**Dynamic Websites**: While this project focused on a static website, the same principles can be applied to deploy dynamic websites. By incorporating services like Amazon EC2, AWS Lambda, and Amazon RDS, we can set up and manage complex, dynamic web applications.

**Enhanced CI/CD Pipelines**: We can extend the current pipeline to include additional stages such as testing, staging, and approval processes. This would ensure that code changes are thoroughly validated before reaching production.

**Infrastructure as Code Best Practices**: Adopting best practices for infrastructure as code (IaC), such as modularizing CloudFormation templates, using parameter stores and secrets managers, and implementing robust monitoring and logging, can further enhance the reliability and maintainability of the deployments.

**Multi-Cloud and Hybrid Deployments**: Explore integrating CloudFormation with other cloud providers or on-premises systems to create hybrid cloud environments. This would provide flexibility and leverage the strengths of multiple platforms.

**Serverless Architectures**: Implementing serverless architectures using AWS Lambda, API Gateway, and other serverless services can significantly reduce operational overhead and improve scalability.

By expanding on the foundational work done in this project, we can continue to innovate and streamline the process of managing and deploying web applications and other cloud-based resources. The potential for automation and efficiency gains is immense, positioning us well for future advancements in cloud computing and DevOps practices.

# REFERENCES

1. A. Bialek, M. Czarnecki, and K. Radomilowicz, "CI/CD with AWS CodePipeline and CloudFormation for Serverless Applications," *2020 11th International Conference on Cloud Computing (CLOUD)*, pp. 167-174, doi: 10.1109/CLOUD47715.2020.00032 (Covers CodePipeline, CloudFormation, and IAM for role creation).
2. S. Malik, A. Alam, and S. W. Shin, "A Secure and Efficient DevOps Approach for Continuous Delivery of Microservices using AWS CodePipeline," *2019 International Conference on Information Networking (ICOIN)*, pp. 508-513, doi: 10.1109/ICOIN48705.2019.8934853 (Focuses on secure DevOps with CodePipeline).
3. M. Palyan, A. Ghosh, and T. Kim, "Leveraging AWS CodePipeline for Continuous Delivery of Machine Learning Workflows," *2020 IEEE International Conference on Cloud Engineering (ICEC)*, pp. 1-10, doi: 10.1109/ICEC48113.2020.00281 (Highlights CodePipeline for machine learning deployments).
4. May 2024.

“Control traffic to your AWS resources using security groups - Amazon Virtual Private Cloud.” AWS Documentation,

https://docs.aws.amazon.com/vpc/latest/userguide/vpc-security-groups.html. Accessed 19

1. X. Ma, Z. Li, H. Zhao, and L. Zhang, "A Cloud-Based Continuous Integration and Delivery Approach for Microservices Architecture," *IEEE Access*, vol. 7, pp. 14222-14234, 2019, doi: 10.1109/ACCESS.2019.2892242 (Discusses CI/CD with CodePipeline for microservices).
2. “Use AWS CodePipeline with AWS CodeBuild to test code and run builds - AWS CodeBuild.” AWS Documentation, https://docs.aws.amazon.com/codebuild/latest/userguide/how-to-create-pipeline
3. A. Rani, M. A. Khan, and N. A. Khan, "A Review of Continuous Integration and Delivery (CI/CD) Tools and Techniques," *Journal of Software: Evolution and Process*, vol. 33, no. 4, pp. 1023-1052, 2021, doi: 10.1002/smsp.4843 (Reviews CI/CD tools including CodePipeline).
4. M. Naderi, *AWS Security Best Practices* (O'Reilly Media, Inc., 2021). (Covers IAM in detail and mentions CodePipeline and CloudFormation).
5. J. Beard, *Building Microservices: Designing Fine-Grained Systems* (O'Reilly Media, Inc., 2016). (This chapter might discuss deploying microservices with CodePipeline/CloudFormation).
6. AWS. “IAM users - AWS Identity and Access Management.” AWS Documentation, 2018, https://docs.aws.amazon.com/IAM/latest/UserGuide/id\_users.html. Accessed 19 May 2024.
7. AWS. “Overview of Amazon Web Services - Overview of Amazon Web Services.” AWS Documentation, 1 March 2024,
8. https://docs.aws.amazon.com/whitepapers/latest/aws-overview/introduction.html. Accessed 19
9. AWS CodePipeline Documentation: <https://docs.aws.amazon.com/codepipeline/> (Provides in-depth information on CodePipeline features).
10. AWS CloudFormation Documentation: <https://docs.aws.amazon.com/cloudformation/> (Offers comprehensive details on CloudFormation).
11. AWS IAM Documentation: <https://docs.aws.amazon.com/iam/> (Explains IAM roles and permissions).
12. AWS Blog: "Using AWS CodePipeline for Continuous Delivery of Serverless Applications": <https://docs.aws.amazon.com/codepipeline/latest/userguide/getting-started-codepipeline.html> (Provides a practical guide on using CodePipeline for serverless deployments).
13. GitHub: "aws-actions/cloudformation-aws-iam-policy-validator": <https://github.com/aws-actions/cloudformation-aws-iam-policy-validator> (This GitHub action helps validate IAM policies used with CloudFormation).
14. A. Jeffords, *High Scalability - No Servers in Sight* (O'Reilly Media, Inc., 2017). (This book might discuss serverless architecture deployment with CodePipeline/CloudFormation).
15. AWS Whitepaper: "AWS Well-Architected Framework: Continuous Delivery": <https://aws.amazon.com/blogs/devops/choosing-a-well-architected-ci-cd-approach-open-source-on-aws/> (Discusses best practices for CI/CD with AWS services).
16. AWS CodePipeline User Guide: <https://docs.aws.amazon.com/codepipeline/> (Another official resource for understanding CodePipeline features).