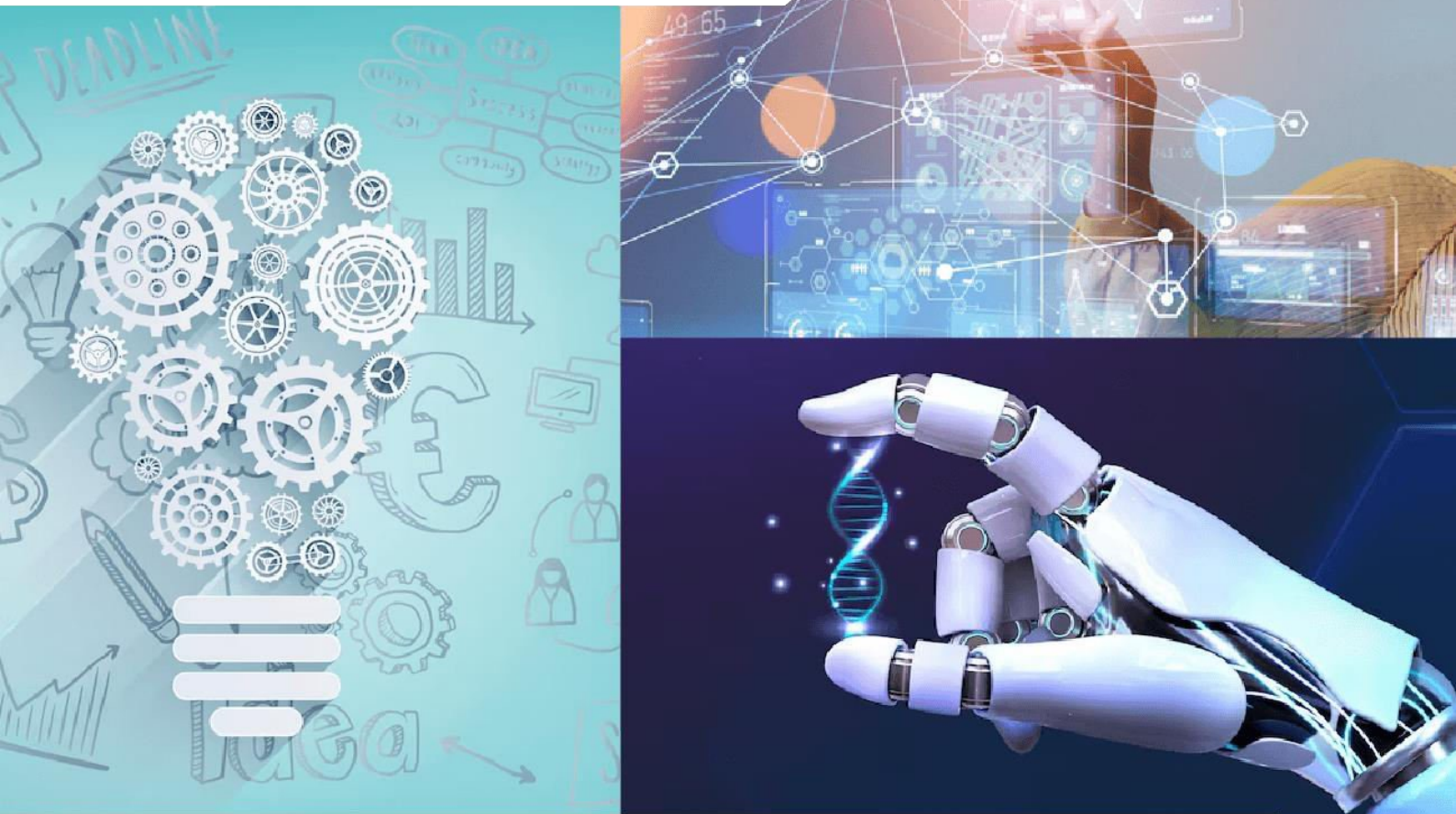




International Journal of Advanced Research in Education and Technology (IJARETY)

Volume 13, Issue 2, March - April 2026

Impact Factor: 8.152



Infrastructure as Code for Enterprise Deployments Using AWS Cloudformation and CLI

Pathipati Chandra Sekhar¹, P. Hima Varshith Reddy², Rahul Jason Rajaputi³, Snigdha Muthyala⁴

Assistant Professor, Department of Computer Science & Engineering, Guru Nanak Institute of Technology,
Ibrahimpattanam, RR District, Telangana, India¹.

Student, Department of Computer Science & Engineering, Guru Nanak Institute of Technology, Ibrahimpattanam, RR
District, Telangana, India^{2,3,4}

ABSTRACT: Cloud computing has become the backbone of modern enterprise IT, offering scalability, speed, and cost savings that manual approaches simply cannot match. But as cloud environments grow larger and more complex, managing infrastructure by hand quickly becomes a liability — deployments drift, configurations become inconsistent, and reproducing an environment reliably is nearly impossible. This paper presents a structured approach to automating enterprise cloud infrastructure using Infrastructure as Code (IaC) with AWS CloudFormation and the AWS Command Line Interface (CLI). We designed and implemented a fully declarative, version-controlled deployment model capable of provisioning VPCs, EC2 instances, IAM roles, security groups, and load-balancing components in a consistent, repeatable way. The system supports blue/green deployments, automated rollback, and continuous drift detection. Results show a clear reduction in deployment time and human error compared to manual provisioning, while security and compliance standards are maintained across environments. This work demonstrates that AWS CloudFormation paired with CLI automation is a practical, enterprise-ready foundation for scalable cloud infrastructure management.

KEYWORDS: AWS CloudFormation, Infrastructure as Code (IaC), AWS CLI, Enterprise Deployment, Cloud Automation, DevOps, YAML Templates, EC2, VPC, IAM, Drift Detection, Blue/Green Deployment.

I. INTRODUCTION

As enterprises grow in the cloud, infrastructure becomes increasingly complex to manage manually. Multiple services like EC2, S3, VPCs, and IAM are often configured differently over time. This inconsistency leads to errors, inefficiencies, and difficulty in maintenance. Infrastructure as Code (IaC) addresses this challenge by defining resources in structured configuration files. These files enable teams to version, review, and reuse infrastructure just like application code.

AWS CloudFormation provides a native solution for implementing IaC using YAML or JSON templates. It ensures resources are provisioned automatically in the correct dependency order. The AWS CLI further enhances this by enabling fully automated deployments. Entire application stacks can be deployed with a single command. This project demonstrates a complete framework for managing networking, compute, security, and monitoring through CloudFormation and automation pipelines.

Getting this right at enterprise scale is harder than it looks. The following four challenges come up consistently in real-world IaC adoption and shaped how this project was designed:

1. **Heterogeneity:** Enterprise cloud environments include many AWS services, each with unique configurations and complexities. CloudFormation templates must use parameters, conditions, and nested stacks to manage this diversity efficiently without duplicating logic.
2. **Configuration Drift:** Manual changes made outside of templates can cause infrastructure to differ from its defined state over time. CloudFormation drift detection helps identify these differences, but it must be regularly used as part of the workflow.
3. **Security and Compliance:** Security settings like IAM policies, encryption, and access rules must remain consistent across all resources. Embedding these controls into templates and validating them with tools ensures compliance is maintained.

4. **Lifecycle Management:** Infrastructure continuously evolves through updates, replacements, and occasional failures. A strong deployment strategy with change sets, rollback mechanisms, and monitoring is essential to manage this lifecycle reliably.

II. METHODOLOGY

2.1 Methodologies

2.1.1 Module Overview

The system is built around seven core functional modules:

- CloudFormation Template Management
- Infrastructure Provisioning
- Networking Configuration
- Security and Access Management
- AWS CLI Automation
- Monitoring and Resource Management
- Output and Validation

1. CloudFormation Template Management

All infrastructure is defined in YAML CloudFormation templates with sections like Parameters, Resources, Outputs, and Metadata. Parameterization allows the same template to be reused across environments by simply changing input values.

2. Infrastructure Provisioning

CloudFormation provisions resources automatically in the correct dependency order, preventing deployment errors. It also uses change sets to preview updates before applying them to the live environment.

3. Networking Configuration

The networking setup includes a VPC with public and private subnets, along with gateways and routing controls. Security groups and network ACLs are used together to manage traffic at both instance and subnet levels.

4. Security and Access Management

IAM roles and policies follow the principle of least privilege and are defined directly in the templates. Sensitive data is securely stored in AWS Systems Manager Parameter Store and never exposed in plain text.

5. AWS CLI Automation

The AWS CLI is used to automate stack creation, updates, and deletion through scripted commands. This approach integrates easily with CI/CD pipelines for fully automated deployments.

6. Monitoring and Resource Management

Stack events and CloudWatch alarms are used to monitor deployments and resource health in real time. AWS Config and drift detection ensure resources remain compliant and unchanged from their defined state.

7. Output and Validation

CloudFormation outputs important resource details like IP addresses and DNS names after deployment. These outputs are used to verify that the application is running correctly and accessible.

III. MODELING AND ANALYSIS

System Architecture

The system is centered on a developer workstation where CloudFormation templates and CLI scripts are created and stored in a version-controlled repository. The AWS CLI deploys these templates through CloudFormation, while monitoring ensures compliance and reduces errors.

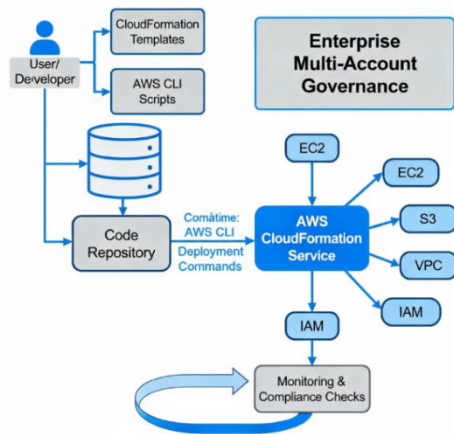


Fig 3.1: System Architecture

Design Engineering uses UML diagrams to communicate how the system is structured and how its components interact. Each diagram below captures a different dimension of the project from high-level actor interactions down to deployment flow and resource relationships.

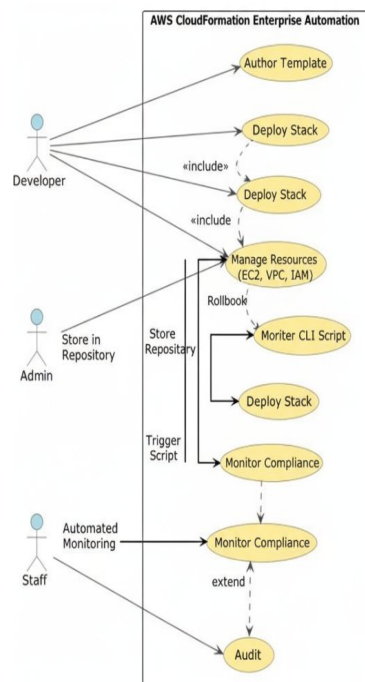


Fig 3.2: Use Case Diagram

The Use Case Diagram shows the interactions between the Developer and Operations Engineer with the system. It includes actions like template creation, deployment, monitoring, and validation through the AWS CLI.

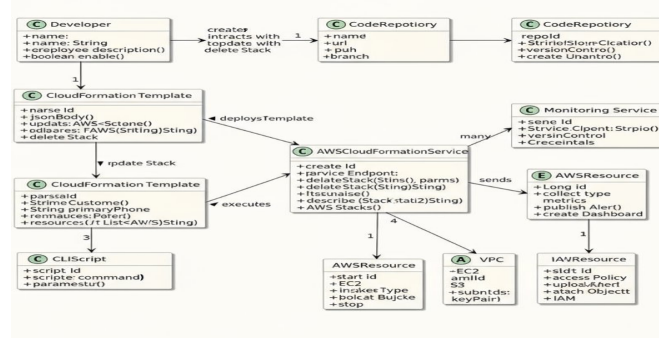


Fig 3.3: Class Diagram

The Class Diagram represents key components such as templates, CLI scripts, stacks, and AWS resources. It shows how these elements are related and how resources are managed within the system.

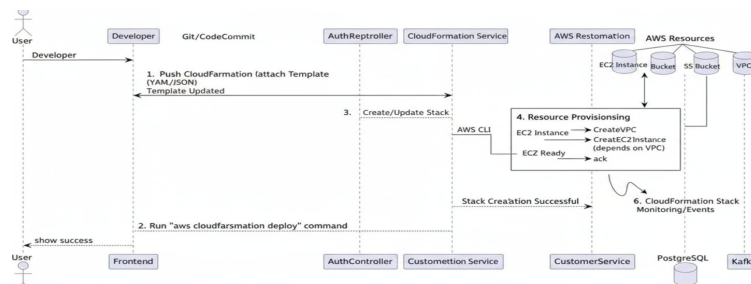


Fig 3.4: Sequence Diagram

The Sequence Diagram explains the step-by-step deployment process from template creation to resource provisioning. It also highlights change set execution and rollback in case of failures.

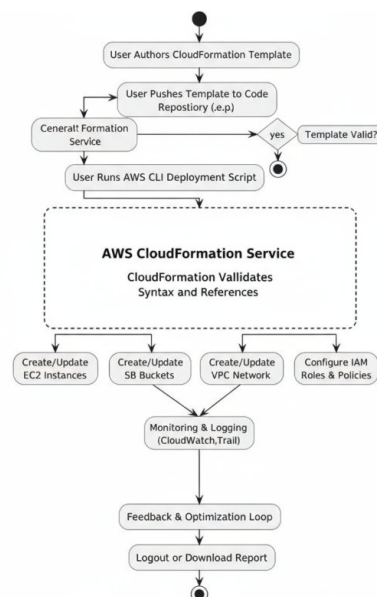


Fig 3.5: Activity Diagram

The Activity Diagram illustrates the overall workflow from template creation to final validation. It shows decision points like successful deployment or rollback paths.

IV. RESULTS AND DISCUSSION

The system was implemented and tested by deploying a complete multi-tier infrastructure stack on AWS. All screenshots below were captured during live deployment sessions. The stack included a VPC with public subnets, an EC2 instance running a web server, an IAM instance profile, and the associated security groups all provisioned entirely through CloudFormation with no manual console interaction.

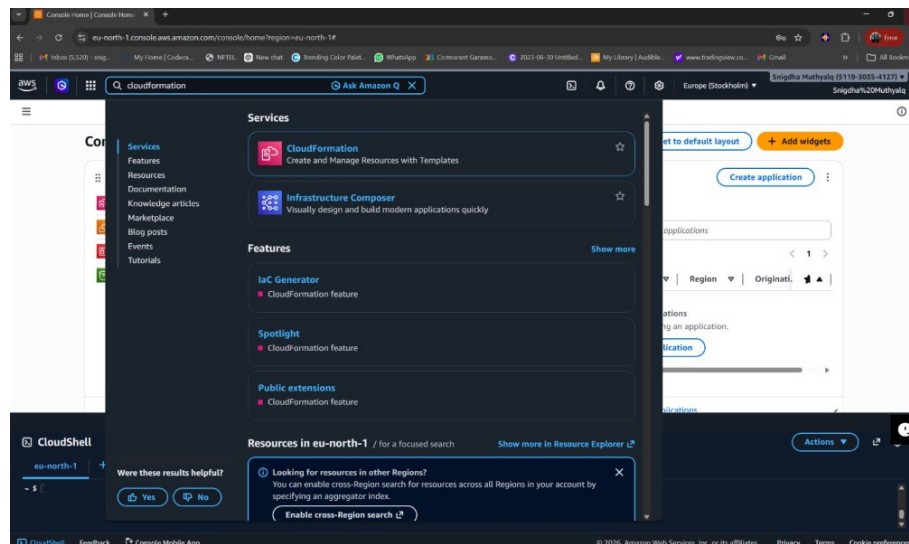


Fig 4.1: Cloud Formation Service Console

The Cloud Formation console is the entry point for managing stacks. From here, users can create new stacks using a template file, use the Infrastructure Composer to build templates visually, or import existing resources. For this project, all deployments were initiated through the CLI rather than the console, but the console was used to monitor progress and review stack events.

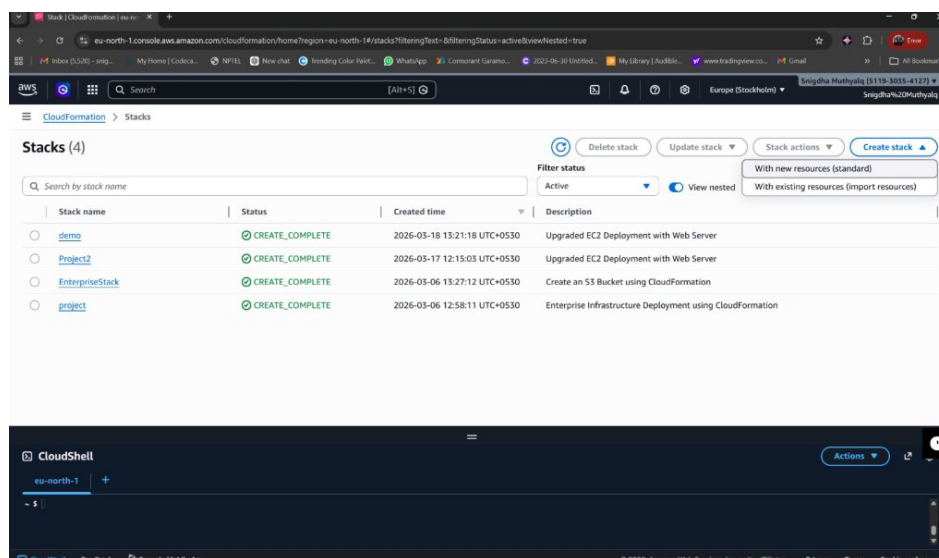


Fig 4.2: Stack Creation Dashboard

This view shows the stack list after multiple deployments. Every stack shows CREATE_COMPLETE status, confirming that all resources were provisioned without errors.

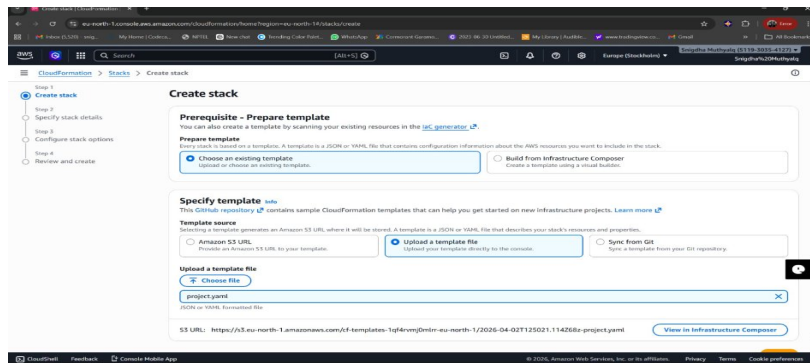


Fig 4.3: YAML Template Upload

The stack wizard accepts a local YAML file as the template source. The file contains all resource definitions, parameters, and outputs for the deployment. Uploading through the wizard is one option; the CLI approach used in this project packages the template and uploads it to S3 automatically as part of the deploy command, making the file path the only input needed.

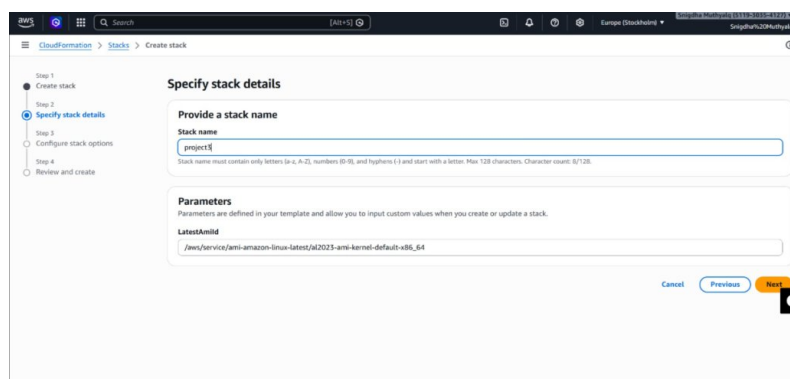


Fig 4.4: Stack Deployment Parameters

Before execution, CloudFormation prompts for parameter values, in this case the stack name and the AMI ID for the EC2 instance. These inputs are the only things that change between environments; the template itself stays the same. This is the core of parameterized IaC: one template, multiple environments, zero duplication of infrastructure logic.

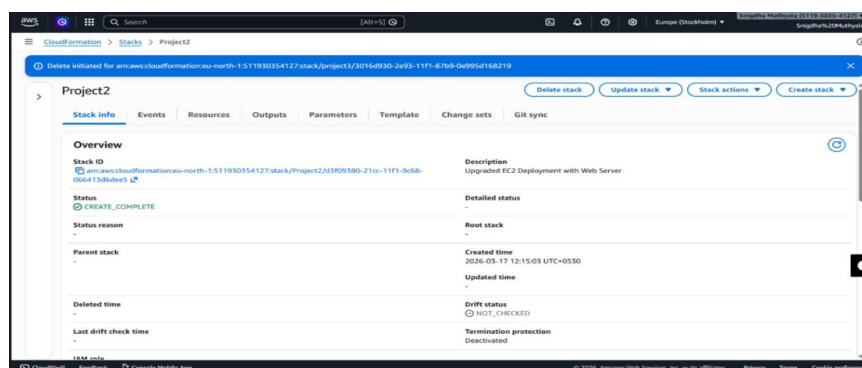


Fig 4.5: Stack Status — CREATE_COMPLETE

The stack detail view confirms CREATE_COMPLETE status along with the stack ID, creation timestamp, and drift status. No drift was detected, which is expected immediately after a fresh deployment. The event log below this view would show each resource being created in sequence, which is useful for debugging if any step had failed.

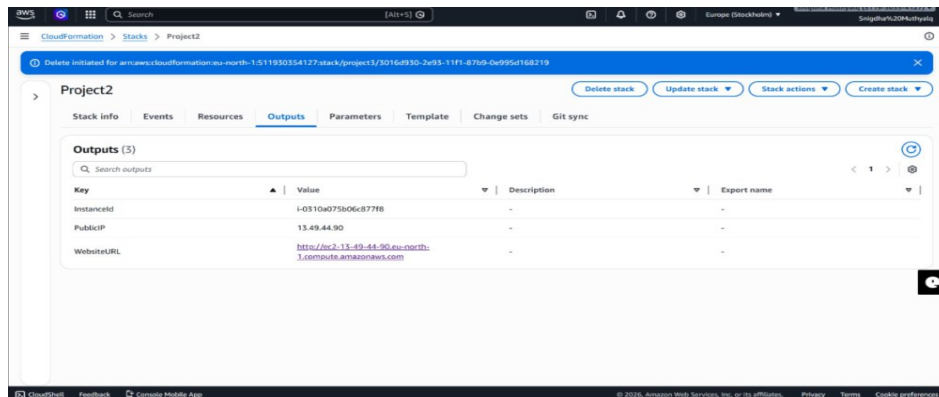


Fig 4.6: Provisioned Resources List

The Resources tab lists every AWS component the stack created, along with its physical resource ID and current status. In this deployment, the list includes the VPC, public subnet, Internet Gateway, route table, security group, IAM instance profile, and EC2 instance all created in a single stack with no manual steps between them.

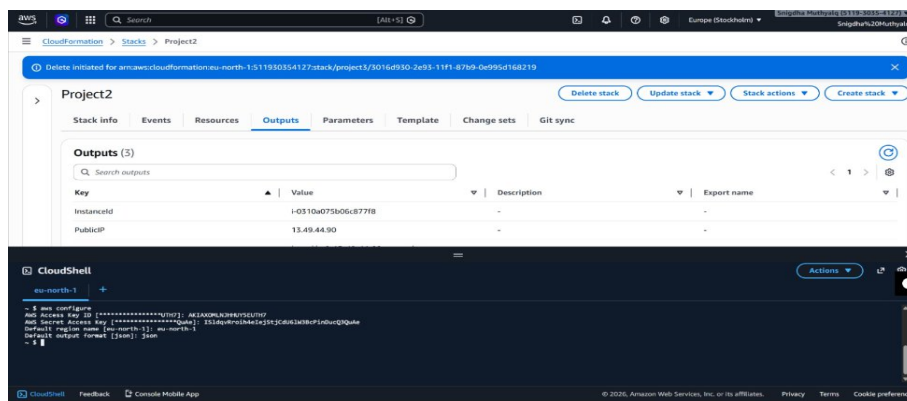


Fig 4.7: AWS CLI Configuration

Before any CLI-driven deployment, the `aws configure` command sets up credentials and region preferences locally. The configuration stores the Access Key ID, Secret Access Key, default region, and output format. These credentials determine which AWS account and region all subsequent CLI commands target, making environment switching as simple as updating the profile.

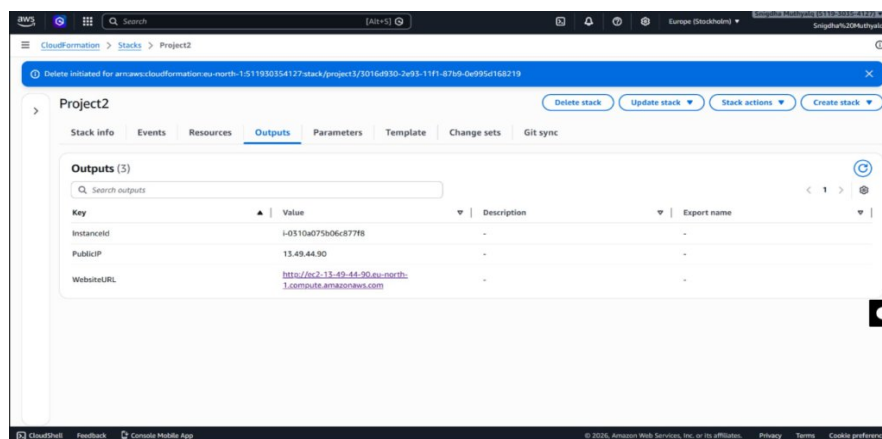


Fig 4.8: Stack Output — Live Web Server

DOI:10.15680/IJARETY.2026.1302124

The stack outputs include the EC2 instance ID, its public IP address, and the full URL to reach the deployed web server. Navigating to that URL in a browser returned the expected response page, confirming that the instance is running, the security group allows HTTP traffic on port 80, and the user data script that installed the web server executed successfully. This end-to-end validation closes the loop on the deployment.

V. CONCLUSION

This project demonstrates that Infrastructure as Code using AWS CloudFormation and CLI automation can effectively support enterprise cloud deployments. Parameterized templates ensure consistent provisioning with reduced errors and faster deployment compared to manual methods. Version control provides full tracking of changes, while rollback mechanisms maintain system stability during failures. Future improvements include CI/CD integration, multi-region support, and AI-based validation for enhanced automation and security.

Additionally, the approach improves scalability by allowing infrastructure to grow without increasing management complexity. It also enhances collaboration among teams by providing a standardized and reusable deployment framework.

REFERENCES

- [1] S. Davidson, L. Sun, B. Bhasker, L. Callot, and A. Deoras, "Benchmarking cloud infrastructure as code across multiple formats," Proc. IEEE Int. Conf. Cloud Eng., pp. 1-12, 2025.
- [2] C. Pahl, N. G. Gunduz, O. C. Sezen, A. Ghamgosar, and N. El Ioini, "Infrastructure as code: Technology review and research challenges," J. Cloud Comput., vol. 14, no. 2, pp. 45-68, 2025.
- [3] R. Ranjan, S. Dustdar, D. Georgakopoulos, and M. Villari, "Cloud resource orchestration programming," IEEE Trans. Cloud Comput., vol. 13, no. 1, pp. 87-102, 2025.
- [4] Deep Science Research Team, "Automation and orchestration in hybrid clouds," Cloud Syst. J., vol. 9, no. 3, pp. 55-70, 2025.
- [5] Orca Security Research Group, "Infrastructure as code security in cloud-native environments," IEEE Secur. Priv., vol. 22, no. 3, pp. 34-47, 2025.
- [6] R. K. Menon, S. Iyer, H. Zhao, and A. Fernandez, "Policy-driven infrastructure as code for secure cloud deployments," IEEE Trans. Cloud Comput., vol. 12, no. 2, pp. 101-115, 2024.
- [7] A. Santos, M. P. Ruiz, F. Ordonez, R. Lacuesta, and J. Lloret, "Automated application deployment on multi-access edge computing using infrastructure as code," IEEE Trans. Netw. Service Manage., vol. 20, no. 3, pp. 215-229, 2023.
- [8] M. Chiari et al., "Static analysis of infrastructure as code: A survey," ACM Comput. Surv., vol. 55, no. 1, pp. 1-34, 2022.
- [9] E. Ozdogan et al., "Systematic analysis of infrastructure as code technologies," J. Syst. Softw., vol. 182, pp. 111064, 2021.
- [10] A. Rahman et al., "A systematic mapping study of infrastructure as code," J. Syst. Softw., vol. 157, pp. 110391, 2019.

International Journal of Advanced Research in Education and Technology

ISSN: 2394-2975

Impact Factor: 8.152