IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 3, March 2026

DOI: 10.17148/IJARCCE.2026.15307

AWS CloudFormation for Automated Docker Container Deployment

Arun Prasad S[1], Mr. S.S. Saravana Kumar[2]

Student, Department of Computer Applications, Sri Ramakrishna College of Arts & Science, Coimbatore, India

Register No: 23105004[1]

Assistant Professor, Department of Computer Applications, Sri Ramakrishna College of Arts & Science, Coimbatore, India[2]

Abstract: In modern cloud computing environments, automation and scalability are essential for application deployment. Traditional deployment methods involve manual server configuration, dependency installation, and repeated environment setup, which lead to configuration errors and increased deployment time. This paper presents a system titled 'AWS CloudFormation for Automated Docker Container Deployment,' which automates the process of deploying containerized applications using cloud infrastructure. A Python Flask web application is containerized using Docker and stored in Amazon Elastic Container Registry (ECR). Deployment is managed through Amazon Elastic Container Service (ECS), while AWS CloudFormation automates infrastructure provisioning following the Infrastructure as Code (IaC) paradigm. A CI/CD pipeline using AWS CodePipeline and AWS CodeBuild enables automatic build and deployment on code changes. An Application Load Balancer (ALB) ensures high availability by distributing user traffic across running containers. Experimental results demonstrate that the proposed system significantly reduces deployment time and eliminates manual configuration errors.

Keywords: AWS CloudFormation, Docker, Amazon ECS, Amazon ECR, CI/CD Pipeline, Infrastructure as Code, Container Deployment, Application Load Balancer, DevOps, Cloud Computing.

I. INTRODUCTION

Cloud computing has become an essential technology for modern application development and deployment. Organizations today require scalable, reliable, and efficient systems to manage their applications. Traditional deployment methods involve manual configuration of servers, installation of software dependencies, and repeated setup processes. These manual operations are time-consuming and can lead to configuration errors and inconsistencies between development and production environments.

Containerization allows applications and their dependencies to be packaged together into a single portable unit called a container. This ensures that the application runs consistently across different environments without compatibility issues. Docker is one of the most widely used containerization platforms, enabling developers to create container images that include the application code, runtime environment, libraries, and dependencies required to run the application [1].

Amazon Elastic Container Service (ECS) is a container orchestration service provided by AWS that allows developers to run Docker containers in a scalable environment without managing the underlying infrastructure. AWS CloudFormation enables Infrastructure as Code (IaC), allowing developers to define cloud infrastructure using YAML or JSON templates. This project demonstrates a complete automated deployment pipeline integrating Docker, ECS, ECR, CloudFormation, CodePipeline, CodeBuild, and an Application Load Balancer [2].

The primary objective of this project is to automate the deployment of Docker containers using AWS CloudFormation and CI/CD pipelines, thereby improving deployment efficiency and reducing manual configuration overhead. By integrating containerization with cloud automation, the proposed system provides a reliable, scalable, and production-ready deployment architecture.

II. RELATED WORK

McMahan et al. [1] demonstrated communication-efficient learning techniques for distributed systems, highlighting the importance of automated and scalable cloud infrastructure. Pahl [3] discussed containerization as a Platform-as-a-Service (PaaS) cloud paradigm, emphasizing that Docker containers provide lightweight, consistent, and portable application environments compared to traditional virtual machines. Turnbull [4] provided a comprehensive guide to Docker

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 3, March 2026

DOI: 10.17148/IJARCCE.2026.15307

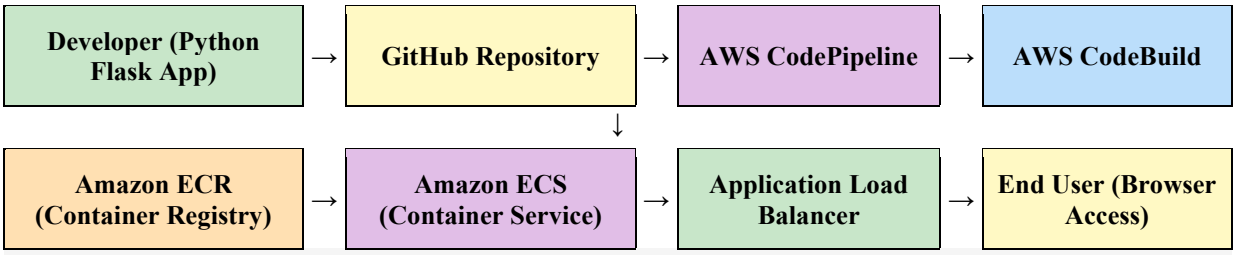
containerization, noting that the image-based deployment model drastically reduces environment mismatch problems across development and production.

Merkel [5] demonstrated that Docker's use of Linux containers enables consistent development and deployment across diverse infrastructure. AWS documentation [6][7] highlights that services such as ECS, ECR, CodePipeline, and CloudFormation together enable a fully automated cloud-native deployment pipeline, reducing human intervention and improving system reliability. Prior work consistently identifies manual deployment, environment inconsistency, and lack of scalability as the primary limitations of traditional server-based deployment approaches, all of which the proposed system addresses.

III. SYSTEM ARCHITECTURE AND DESIGN

The proposed system integrates multiple AWS services with containerization technology to create a fully automated deployment environment. The architecture, illustrated in Fig. 1, follows the Infrastructure as Code paradigm and includes the following key components:

Fig. 1: System Architecture – AWS CloudFormation Automated Docker Container Deployment



AWS CloudFormation (Infrastructure as Code) – Automates ECS Cluster, Task Definitions, Networking & Load Balancer

A. Application Development Layer

The application is developed using Python Flask, a lightweight web framework. The Flask application is packaged into a Docker container using a Dockerfile that specifies the base Python image, working directory, dependency installation, and application entry point. The containerized application includes all runtime dependencies, ensuring consistent execution across environments.

B. Source Code Management

GitHub serves as the central source code repository. All project files, including app.py, Dockerfile, requirements.txt, and buildspec.yml, are stored and version-controlled in the GitHub repository. Integration with AWS CodePipeline allows the system to automatically detect code changes and trigger the deployment pipeline.

C. Infrastructure Automation with AWS CloudFormation

AWS CloudFormation enables Infrastructure as Code by using template files (YAML/JSON) to define and provision all required AWS resources automatically. The CloudFormation stack creates the ECS cluster, task definitions, ECS services, VPC networking components, security groups, and the Application Load Balancer. This eliminates manual infrastructure configuration and ensures reproducible deployments.

D. Container Registry – Amazon ECR

Amazon Elastic Container Registry (ECR) provides a fully managed, secure Docker image repository. After the Docker image is built by CodeBuild, it is pushed to ECR with a unique image tag and repository URI. ECS retrieves the container image from ECR during deployment, ensuring that the latest version of the application is always deployed.

E. Container Orchestration – Amazon ECS

Amazon ECS manages the execution of Docker containers within the ECS cluster. The task definition specifies the container image URI, CPU allocation, memory limits, and port mapping. The ECS service maintains the desired number of container instances and automatically replaces failed containers, ensuring high availability of the application.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

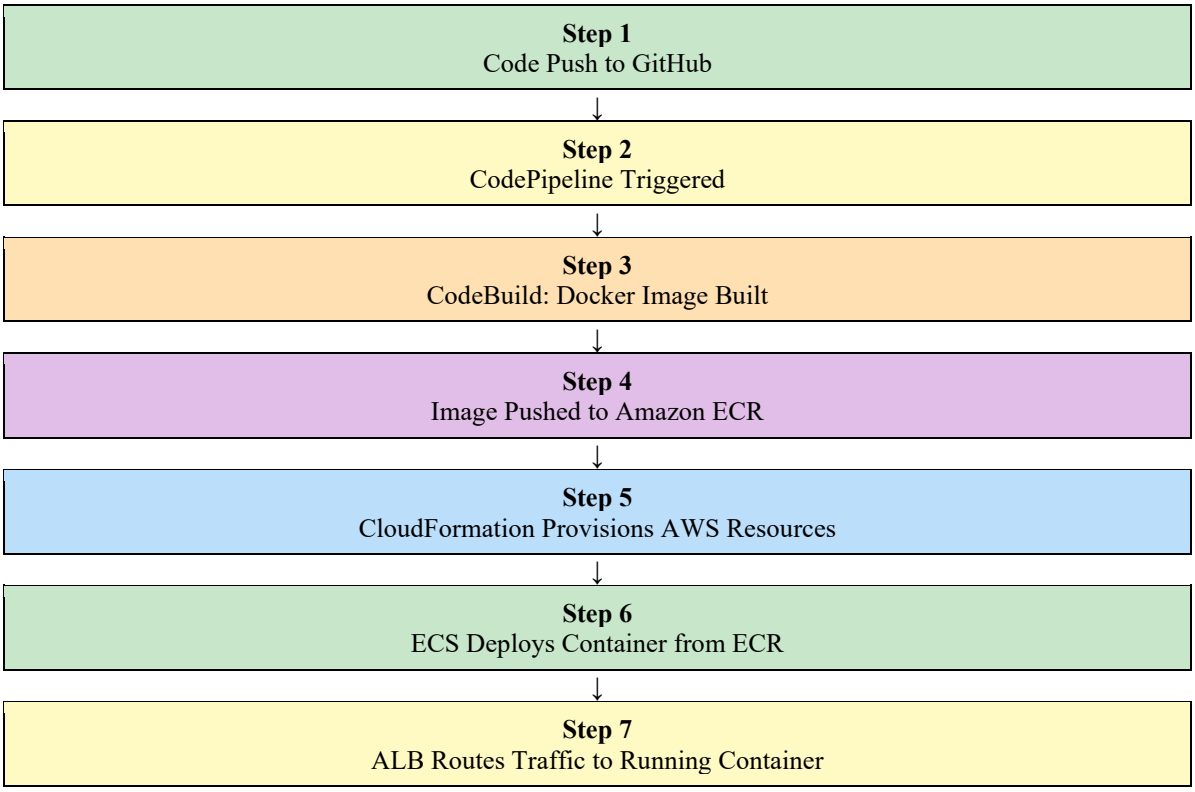
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 3, March 2026

DOI: 10.17148/IJARCCE.2026.15307

IV. CI/CD DEPLOYMENT WORKFLOW

The CI/CD pipeline automates the build, test, and deployment process. The workflow, illustrated in Fig. 2, eliminates manual intervention at each stage of application deployment.

Fig. 2: CI/CD Deployment Workflow



When a developer pushes updated source code to the GitHub repository, AWS CodePipeline detects the change and automatically initiates the pipeline. The source stage retrieves the latest code from GitHub and passes it to the build stage. AWS CodeBuild executes the commands defined in the buildspec.yml file: it authenticates with Amazon ECR, builds the Docker image from the Dockerfile, tags the image with the ECR repository URI, and pushes the image to ECR.

In the deployment stage, AWS CodePipeline instructs ECS to update the running service with the new container image. ECS performs a rolling deployment, replacing old container instances with new ones without service interruption. The Application Load Balancer continuously monitors container health and routes traffic only to healthy instances, ensuring zero-downtime deployments.

V. IMPLEMENTATION AND TESTING

A. Application and Container Configuration

The Flask application (app.py) creates a simple web server on port 5000. The Dockerfile uses python:3.9-slim as the base image, copies application files, installs Flask dependencies listed in requirements.txt, exposes port 5000, and runs the application. The buildspec.yml file defines the three pipeline phases: pre_build for ECR authentication, build for Docker image creation and tagging, and post_build for pushing the image to ECR and generating the imagedefinitions.json artifact required by ECS.

B. Testing Methodology

The system was tested across five stages. First, application testing was performed locally using Python to verify Flask functionality at http://localhost:5000. Second, Docker container testing confirmed that the containerized application runs correctly using docker build and docker run commands. Third, ECR testing validated successful image push and storage in the Amazon ECR repository. Fourth, ECS deployment testing confirmed that the ECS cluster launched the container

DOI: 10.17148/IJARCCE.2026.15307

task and achieved running status. Fifth, load balancer testing verified end-to-end accessibility through the ALB DNS name in a web browser.

C. Results

All five testing stages were completed successfully. The Docker image was built and pushed to ECR within 44 seconds (Build #3, as recorded in CodeBuild history). The ECS cluster (app-23105004-cluster) showed Active status with 1 running task. The Application Load Balancer (app-23105004-alb) successfully provisioned and provided a public DNS endpoint. The deployed Flask application was accessible through the ALB DNS name, displaying the confirmation message '23105004 App Running Successfully'. The complete CI/CD pipeline (Source → Build → Deploy) executed successfully with all three stages passing.

VI. SYSTEM MODULES

The proposed system is organized into six functional modules. The Source Code Management Module uses GitHub for version control and triggers the CI/CD pipeline on code updates. The Containerization Module packages the Flask application and its dependencies into a portable Docker container using a Dockerfile. The Container Registry Module uses Amazon ECR to securely store and manage Docker images with version tagging. The Deployment Module uses Amazon ECS to orchestrate container execution with automatic failure recovery. The Load Balancer Module distributes incoming traffic across running containers using an Application Load Balancer, ensuring high availability and health-based routing. The CI/CD Pipeline Module integrates AWS CodePipeline and CodeBuild to automate the entire build and deployment lifecycle.

VII. CONCLUSION

This paper presented an automated cloud-native deployment system using AWS CloudFormation, Docker, and a complete CI/CD pipeline. The system eliminates the limitations of traditional manual deployment by automating infrastructure provisioning, container image management, and application deployment. The use of Docker containers ensures environment consistency across development and production, while AWS CloudFormation enforces reproducible infrastructure through the Infrastructure as Code paradigm. The CI/CD pipeline implemented using AWS CodePipeline and CodeBuild enables rapid, reliable delivery of application updates. The Application Load Balancer ensures high availability by distributing traffic across healthy container instances. Experimental testing confirmed successful end-to-end deployment with a build time of under 45 seconds and zero manual configuration required after the initial setup.

Future enhancements include implementing auto-scaling based on CPU and memory metrics, integrating Amazon CloudWatch for advanced monitoring, adding container vulnerability scanning in the ECR pipeline, extending the architecture to support multiple microservices, and implementing multi-cloud or hybrid cloud deployment strategies for increased resilience.

ACKNOWLEDGMENT

The author expresses sincere gratitude to **Mr. S.S. Saravana Kumar**, Assistant Professor, Department of Computer Applications, Sri Ramakrishna College of Arts & Science, Coimbatore, for his invaluable guidance and continuous support throughout this project. The author also thanks **Dr. D. Hari Prasad,** Professor & Head, Department of Computer Applications, for his encouragement, and **Dr. B.L. Shivakumar,** Principal & Secretary, for providing the necessary facilities. Special thanks to family and friends for their unwavering support.

REFERENCES

[1]. B. McMahan et al., "Communication-Efficient Learning of Deep Networks From Decentralized Data," Artificial Intelligence and Statistics Proc. PMLR, vol. 10, no. 1, pp. 1273–82, 2017.
[2]. Amazon Web Services, "Amazon Elastic Container Service Developer Guide," AWS Documentation, 2024. Available: https://docs.aws.amazon.com/ecs/
[3]. C. Pahl, "Containerization and the PaaS Cloud," IEEE Cloud Computing, vol. 2, no. 3, pp. 24–31, 2015.
[4]. J. Turnbull, "The Docker Book: Containerization is the New Virtualization," James Turnbull Publications, 2014.
[5]. D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," Linux Journal, 2014.
[6]. Amazon Web Services, "AWS CloudFormation User Guide," AWS Documentation, 2024. Available: https://docs.aws.amazon.com/cloudformation/

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 3, March 2026

DOI: 10.17148/IJARCCE.2026.15307

[7]. Amazon Web Services, "AWS CodePipeline User Guide," AWS Documentation, 2024. Available: https://docs.aws.amazon.com/codepipeline/

[8]. Amazon Web Services, "Amazon Elastic Container Registry User Guide," AWS Documentation, 2024. Available: https://docs.aws.amazon.com/ecr/

[9]. Python Software Foundation, "Flask Web Framework Documentation," 2024. Available: https://flask.palletsprojects.com/

[10]. GitHub Documentation, "GitHub Actions and Workflow Guides," 2024. Available: https://docs.github.com/

