

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

FIRST Semester 2025- 2026

CSIW ZG628T DISSERTATION

Dissertation Outline Evaluation

Title of Dissertation: Automated CI-CD Pipeline Implementation for Linux-Based Applications Using AWS

BITS ID No. 2021WA86442

Name of Student: SIDDHESH DNYANESHWAR SHEJOLE

E-mail ID of the student: 2021wa86442@wilp.bits-pilani.ac.in

Name of Supervisor: Seshu Sharma

Designation of Supervisor: Lead Administrator

Qualification and Experience: B.Tech and 13+ years of experience in Linux & VMware

E- mail ID of Supervisor: seshu.sharma@wipro.com

Name of First Examiner: Pratyush Srivastava

Designation of First Examiner: Lead Administrator

Qualification and Experience: B.Tech and 9+ years of experience in Linux. (Wipro)

E- mail ID of First Examiner: Pratyush.Srivastava1@wipro.com

Name of Second Examiner: Shreya Tekade

Designation of Second Examiner: Project engineer

Qualification and Experience: B.E. and 3+ years of experience (Wipro)

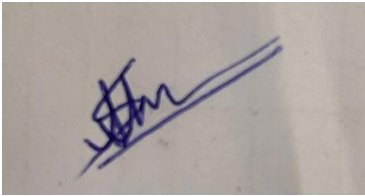
E- mail ID of Second Examiner: shreya.tekade1@wipro.com

Supervisor's rating of the Technical Quality of this Dissertation Outline

EXCELLENT / GOOD / FAIR/ POOR (Please specify): EXCELLENT

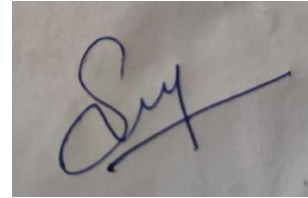
Supervisor's suggestions and remarks about the outline:

Its project oriented and meets the business requirements. which can be delivered and maintained on schedule.



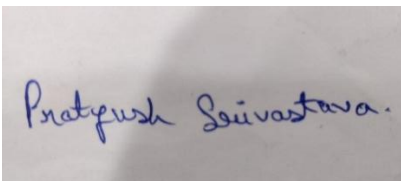
Signature of Student

31-07-2025



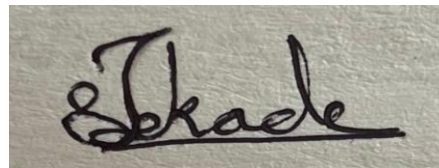
Signature of Supervisor

31-07-2025



Signature of First Examiner

31-07-2025



Signature of Second Examiner

31-07-2025

Overview of the Project

The objective of this project is to design and implement a complete Continuous Integration and Continuous Deployment (CI/CD) pipeline tailored for Linux-based applications using Amazon Web Services (AWS). The system automates the process of building, testing, and deploying code to a virtual machine hosted on AWS (EC2), eliminating the need for manual intervention. This project is built around AWS-native services, including CodeCommit (for source control), CodeBuild (for compiling and testing), CodePipeline (for managing CI/CD flow), EC2 (as a Linux server for deployment), CloudWatch (for logging and monitoring), and SNS (for notification services).

By building this pipeline, the project aims to demonstrate how cloud-native CI/CD solutions can be easily adopted for small-scale Linux applications. It also provides hands-on exposure to real-world DevOps practices and familiarizes learners with tools that are widely used in industry.

Key Features of the Project

1. Cloud-Native CI/CD Automation

Leverages AWS-managed services to build a fully automated CI/CD workflow from source to deployment.

2. Linux-Based Application Hosting

Deploys and runs the web application (e.g., Flask/Node.js) on a Linux EC2 instance, simulating production-like environments.

3. End-to-End Integration

Integrates source control (CodeCommit), build automation (CodeBuild), deployment (CodePipeline), and logging/alerts (CloudWatch and SNS).

4. Cost-Effective Setup

Designed to operate entirely within the AWS Free Tier, making it ideal for students and small teams with limited resources.

5. Health Monitoring and Logging

Includes post-deployment health checks and centralized log management via AWS CloudWatch.

6. Real-Time Notifications

Configures AWS Simple Notification Service (SNS) for immediate alerts on build and deployment status.

7. Modular and Scalable Design

The pipeline can be easily extended to support new applications, environments, or cloud services in future phases.

8. Educational Value

Offers a practical learning opportunity for students and professionals aiming to understand DevOps, cloud computing, and pipeline automation.

TABLE OF CONTENTS

Serial No.	Title
1	Abstract
2	Background of the Project
3	Project Overview
4	Scope of Dissertation Work
5	Expanded Methodology
6	Customer/End-User Benefits
7	Detailed 16-Week Plan of Work
8	References

1. Abstract

This dissertation presents the implementation of a Continuous Integration and Continuous Deployment (CI/CD) pipeline for Linux-based applications using Amazon Web Services (AWS). The project focuses on automating the build, test, and deployment stages using AWS-native services such as CodeCommit, CodeBuild, CodePipeline, EC2, CloudWatch, and SNS. The goal is to eliminate manual intervention, reduce errors, and achieve faster delivery cycles while maintaining scalability and cost-effectiveness through the AWS Free Tier. This project offers a practical solution for DevOps adoption in small-scale academic and enterprise environments.

2. Background of the Project

Traditional application deployment methods involve significant manual effort, making them error-prone and inefficient. In recent years, DevOps practices like CI/CD have emerged to solve these problems by automating the software lifecycle. Tools such as Jenkins and GitLab CI are commonly used but require manual server setup and maintenance. AWS offers fully managed CI/CD services, allowing developers to focus on applications rather than infrastructure. This project leverages that capability to build a lightweight, scalable, and production-ready Linux-based deployment system using AWS-native services, ideal for learners and organizations with limited DevOps experience.

3. Project Overview

The project aims to automate the delivery of a web application using the following AWS services:

- AWS CodeCommit for version control
- AWS CodeBuild to compile and test application code
- AWS CodePipeline for orchestrating the CI/CD flow
- EC2 (Amazon Linux) instance to host the deployed application
- Amazon CloudWatch for logging and performance monitoring
- Amazon SNS for email-based build and deployment alerts

The CI/CD pipeline is triggered when code is pushed to the version control repository, automatically building the application and deploying it to a Linux-based EC2 instance. This end-to-end solution provides a practical understanding of DevOps in the cloud.

4. Scope of Dissertation Work

- The scope of this project includes the following key activities:
- Setting up the AWS environment (IAM roles, EC2, Code services)
- Developing a sample Linux-compatible application (Flask or Node.js)
- Implementing a CI/CD pipeline using AWS CodePipeline
- Automating deployment to EC2 with Bash scripting
- Enabling logging and health monitoring using CloudWatch
- Setting up real-time notifications with SNS
- Conducting functional testing and performance analysis
- Documenting pipeline architecture, configuration files, logs, and outcomes

Excluded from scope: complex multi-region or multi-service deployments, high availability configurations, or containerization via ECS/EKS.

5. Expanded Methodology

5.1 Requirement Gathering

- Identify suitable application (Flask or Node.js)
- Define the CI/CD flow and integration points

5.2 Environment Setup

- Create an AWS account under the Free Tier
- Setup IAM users, roles, and security groups
- Launch a Linux-based EC2 instance and configure SSH access

5.3 Application Development

- Build a basic web application
- Set up Git and connect to AWS CodeCommit or GitHub repository

5.4 Continuous Integration

- Write `buildspec.yml` to define build steps
- Configure AWS CodeBuild to compile/test the application

5.5 Continuous Deployment

- Set up AWS CodePipeline for source → build → deploy
- Write a deployment script to copy files and restart the app on EC2

5.6 Monitoring and Notification

- Enable CloudWatch to capture logs and metrics
- Configure SNS to send email notifications on success/failure

5.7 Testing and Validation

- Simulate pipeline failures to test rollback capabilities
- Test for application availability using health checks

5.8 Documentation

- Record architecture diagrams, configuration files, test results, and code samples

6. Customer/End-User Benefits

- Speed: Automation reduces time-to-deploy from hours to minutes
- Reliability: Fewer manual steps mean fewer human errors
- Scalability: AWS services support scaling without extra setup
- Cost Efficiency: Entire solution runs within AWS Free Tier limits
- Industry Alignment: Reflects real-world practices used in DevOps and cloud teams
- Skill Development: Offers hands-on experience with cloud-native tools

7. Detailed 16-Week Plan of Work

Serial Number of Task	Tasks or Subtasks to be done (be precise and specific)	Planned duration in weeks	Specific Deliverable in terms of the project
1	Finalize topic and obtain supervisor approval	1 week	Approved dissertation outline
2	Set up AWS Free Tier account and configure IAM roles	1 week	Secure AWS account and IAM access
3	Launch EC2 Linux instance and install necessary tools	1 week	EC2 instance ready for deployment
4	Develop sample Linux-compatible web app (Flask/Node.js)	1 week	Working app on local system
5	Push code to GitHub or AWS CodeCommit	1 week	Source code available in repo
6	Create buildspec.yml and configure AWS CodeBuild	1 week	CI process successfully running
7	Set up AWS CodePipeline (Source → Build)	1 week	Basic CI pipeline functional
8	Write shell deployment script and auto-deploy to EC2	1 week	Code deployed to EC2 instance
9	Integrate post-deployment health check (e.g., curl)	1 week	Automated uptime validation
10	Configure SNS for email notifications	1 week	Alerts on build/deployment outcomes
11	Enable and test CloudWatch logging for build and EC2	1 week	Logs visible in CloudWatch console
12	Conduct full testing with success/failure/rollback cases	1 week	Test log with screenshots
13	Begin writing documentation and consolidate outputs	1 week	Draft of report structure
14	Analyze pipeline performance and document comparison	1 week	Metrics, graphs, analysis summary
15	Finalize report and create project presentation slides	1 week	Completed report and slides
16	Submit report and prepare for viva examination	1 week	Submitted project and viva-ready

8. References

- Amazon Web Services. (n.d.). *AWS CodePipeline – Continuous Delivery*. Retrieved from <https://aws.amazon.com/codepipeline>
- Amazon Web Services. (n.d.). *AWS CodeBuild – Fully Managed Build Service*. Retrieved from <https://aws.amazon.com/codebuild>
- Amazon Web Services. (n.d.). *Best Practices for CI/CD on AWS*. Retrieved from <https://docs.aws.amazon.com/codepipeline/latest/userguide/welcome.html>
- Rahman, M., & Williams, L. (2018). *Characterizing Continuous Integration Build Failures: An Exploratory Analysis*. *Empirical Software Engineering*, 23(3), 1751–1780. <https://doi.org/10.1007/s10664-017-9552-x>
- GitLab. (2024). *CI/CD Pipeline Configuration Reference*. Retrieved from <https://docs.gitlab.com/ee/ci>
- Jenkins.io. (2023). *Jenkins User Documentation*. Retrieved from <https://www.jenkins.io/doc>