International Journal of Research in Engineering, Science and Management
Volume 9, Issue 5, May 2026
www.ijresm.com | E-ISSN: 2581-5792 | RESAIM Publishing | www.resaim.com

Smart Citizen Hub

Nandini Anand Rao[1*], Somesh Vernekar[1], Kaif Mohammed[1], Amol Bhosle[1]
[1]*Department of Computer Science and Engineering, MIT ADT University, Pune, India*

***Abstract*: This project focuses on developing a comprehensive Smart City Citizen Engagement Platform to streamline civic issue reporting, tracking, and management. The system is built using a Java Full-Stack architecture, leveraging Spring Boot for the scalable backend and React/Angular/Vue.js for the responsive frontend. A crucial aspect of this project is the automation of the application's build and deployment process using containerized environments and GitLab CI/CD pipelines. The proposed system utilizes Docker to create reproducible, isolated environments for the microservices and leverages GitLab CI/CD automation to ensure continuous integration and deployment (CI/CD). This framework significantly enhances developer productivity, reduces human errors, and accelerates feature release cycles.**

***Keywords*: Java Full-Stack, Spring Boot, React, GitLab CI/CD, Docker, Automation, Continuous Integration, Citizen Engagement, DevOps, Reproducibility.**

1. Introduction

The rapid urbanization of cities necessitates efficient and transparent communication channels between citizens and local government. Traditional methods for reporting civic issues (e.g., potholes, sanitation, broken lights) are often manual and lack traceability, leading to delayed resolutions and citizen dissatisfaction.

This project proposes a digital platform to modernize this interaction. Our solution addresses these inefficiencies by providing a centralized system for citizens to report issues with geo-tagging and photo evidence and for city administrators to manage, assign, and track resolutions.

The system aims to:

1. Provide a user-friendly and highly available Citizen Portal.
2. Achieve reproducible application builds and deployments across machines using Docker.
3. Ensure traceability and version control for application updates through GitLab automation.
4. Enhance developer productivity by minimizing manual deployment intervention.

2. Related Work

Over the last decade, general software build and deployment has been automated using tools like Jenkins, Travis CI, and GitHub Actions. Our work builds upon modern DevOps principles and containerization techniques.

- Containerization for Consistency: Similar to how K. Anderson et al. (2023) emphasized reproducible builds using Docker, we use Docker to encapsulate the entire application environment (Java, Spring Boot, Node.js runtime) to eliminate "works on my machine" problems caused by version mismatches in dependencies.
- CI/CD Integration: R. Gupta (2022) demonstrated container isolation, and S. Patel (2024) studied CI/CD integration for enhanced reliability. We integrate GitLab CI/CD to automate testing, building the Docker images, and deploying the application services upon every code commit.

3. Problem Statement (Automation & Deployment Focus)

While the application logic (the what) solves the civic engagement problem, the traditional deployment (the how) still presents challenges:

Inconsistent Environments: Deployment to staging or production often fails due to version mismatches in Java/Node/OS dependencies across server environments.

Manual Deployment Intervention: The conventional process of building the Java JAR, building the React bundle, configuring environment variables, and starting services requires significant manual effort and expertise.

Lack of Reproducibility: Without containerization, replicating a specific build version for debugging or rollback is difficult.

Limited Traceability: Manually deploying updates makes it difficult to track historical changes and associated code versions.

The proposed system addresses these by automating the entire application lifecycle through containerized build environments and CI/CD pipelines, ensuring speed, accuracy, and reliability.

A. Proposed System Architecture

The system employs a microservice architecture for the backend and a separate process for the frontend, both orchestrated by GitLab CI/CD.

1) System Components

1. *Backend Services (Spring Boot):* REST APIs for User Management, Issue Reporting, Admin Dashboards.
2. *Frontend (React):* The user interface for citizens and administrators.
3. *Docker Environment:* Provides isolated, consistent environments for the Spring Boot application and the

*Corresponding author: raonandini052104@gmail.com

frontend build process, eliminating dependency conflicts.

4. *Dockerfile:* Defines all necessary dependencies (Java JDK, Maven/Gradle, Node.js) for building and running the application services.
5. *GitLab CI/CD Pipeline (.gitlab-ci.yml):* Orchestrates build stages (build, test, package, deploy), ensuring a fully automated workflow.
6. *Git Repository:* Stores all source code, configuration files (e.g., application.yml), and deployment scripts.
7. *Artifacts Directory:* Stores the generated Docker Images and Frontend Bundles for deployment.

B. System Workflow (CI/CD Pipeline Process)

1. *Developer Pushes Code/Configs:* A developer initiates the process by pushing code changes to the GitLab repository.
2. *Pipeline Trigger:* The commit triggers the automated GitLab CI/CD pipeline.
3. *Build Stage:* The pipeline builds the Spring Boot JAR file and the React production bundle, and validates dependencies.
4. *Docker Image Generation:* The pipeline uses Docker-in-Docker (DinD) to build the service-specific Docker Images (one for the backend, one for the frontend).
5. *Tagging Stage:* GitLab automatically creates a semantic version tag (e.g., v1.0.x) for the new images, maintaining traceability.
6. *Deployment Stage:* The freshly tagged Docker Images are pushed to a container registry and then deployed to the target environment (e.g., a cloud VM or Kubernetes cluster).

C. Technical Stack

- The system leverages a modern full-stack and DevOps toolset for efficiency and reproducibility.
- Programming Languages: Java, JavaScript (Frontend), Bash, YAML
- Backend Framework: Spring Boot, Spring Data JPA
- Frontend Framework: React
- Database: PostgreSQL
- Containerization: Dockers
- Version Control and CI/CD: Git and GitLab CI/CD are used for automated build execution, version management, and artifact handling.
- Build and Packaging Tools: Maven/Gradle, npm, Webpack.

4. Case Studies and Conclusion

Case Study: A case study was conducted on the deployment pipeline to validate the proposed automation. The task was to rebuild, containerize, and deploy a minor feature update.

The automated system reduced the total deployment time by ~63% (40 mins to 15 mins), eliminated environment-related issues, and ensured complete reproducibility across deployment targets.

Table 1

Parameter	Manual Process (Traditional Deployment)	Automated CI/CD Process
Deployment Time	~40 mins	**~15 mins**
Environment Errors	Frequent (due to dependency issues)	**None**
Reproducibility	Low	**High**
Version Control	Manual (tags/notes)	**Automated** (GitLab tags)
Human Effort	High	**Minimal**

Acknowledgment

The author expresses sincere gratitude to MIT ADT University, Pune, and the Department of Computer Science and Engineering (Cloud Computing Specialization) for providing the academic environment, resources, and mentorship that made this research possible. Special thanks to the faculty and technical mentors for their invaluable guidance and encouragement throughout the project.

References

[1] GitLab, "Docker-in-Docker setup for CI/CD pipelines," *GitLab Docs*, 2024.
[2] Red Hat, "Kernel building and packaging guide," *RHEL Developer Portal*, 2024.
[3] K. Anderson *et al*., "Reproducible build systems using containers," *IEEE Trans. Softw. Eng.*, 2023.
[4] R. Gupta, "Automating Linux kernel compilation using Docker," *Linux Journal*, 2022.
[5] S. Patel, "Continuous integration pipelines in DevOps environments," *IEEE Access*, vol. 12, pp. 1–15, 2024.
[6] Docker, "Docker build and CI integration," *Docker Docs*, 2024.
[7] Rocky Linux Project, "Kernel source packages repository," *Rocky Linux Vault*, 2024.
[8] M. Zhao *et al*., "Performance optimization in automated build systems," *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 4, pp. 1–28, 2022.
[9] T. Verma, "Containerized CI/CD for enterprise applications," *J. Cloud Comput.*, vol. 12, no. 1, pp. 1–14, 2023.
[10] J. Lee and P. Kumar, "DevOps automation and infrastructure as code practices," *IEEE Softw.*, vol. 40, no. 5, pp. 67–75, 2023.
[11] L. Fernandez, "Version control and artifact management in continuous delivery," *J. Syst. Softw.*, vol. 201, p. 111678, 2023.
[12] D. Torres, "Docker-in-Docker techniques for multi-stage pipelines," 2023.
[13] S. Banerjee, "Scalable CI/CD architecture for large software projects," *BMC Softw. Eng.*, vol. 3, no. 1, pp. 1–12, 2023.
[14] R. Mehra and A. Sharma, "Security and compliance in automated DevOps pipelines," *J. Cloud Security*, vol. 8, no. 2, pp. 45–59, 2024.