Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

Automated Cloud Deployment Using CI/CD Pipeline

Prof. Mr. Sahil Ramteke
Dept. of Artificial Intelligence and Data Science,
Priyadarshini College of Engineering
Nagpur, India

Renuka Sahastrabudhe
Dept. of Artificial Intelligence and Data Science,
Priyadarshini College of Engineering
Nagpur, India

Bulbul Roy
Dept. of Artificial Intelligence and Data Science,
Priyadarshini College of Engineering
Nagpur, India

Krisha Gulhane
Dept. of Artificial Intelligence and Data Science,
Priyadarshini College of Engineering Nagpur, India

Yash Pillewan
Dept. of Artificial Intelligence and Data Science,
Priyadarshini College of Engineering Nagpur, India

Abstract: The evolution of software delivery mechanisms has fundamentally transformed how organizations manage infrastructure and deploy applications. This paper presents the architecture, design decisions, implementation strategies, and deployment methodology of Hero Lib, a cloud-native web application for browsing a curated book catalogue. The system is composed of two decoupled services: a RESTful Python/Flask backend and a server-side-rendered Next.js/React frontend, both containerised with Docker and hosted on Google Cloud Run — a managed, serverless container platform. The methodology involves leveraging declarative configuration tools such as Terraform alongside Google Cloud Platform (GCP) services to orchestrate the complete lifecycle of multi-tier applications. Through systematic analysis, this study demonstrates that automated deployment frameworks significantly reduce operational latency while enhancing environment consistency and eliminating configuration drift commonly associated with manual provisioning approaches. Experimental findings reveal an 80% reduction in deployment duration and improved reliability metrics compared to conventional manual methodologies. The research contributes a replicable blueprint for achieving enterprise-grade automation within virtual machine-based deployment contexts, addressing gaps in existing literature that predominantly focus on containerized solutions.

Keywords: ***Continuous Integration, Continuous Deployment, Infrastructure as Code, Cloud***

Automation, DevOps, Terraform, Google Cloud Platform, CI/CD Pipeline.

I. INTRODUCTION

Contemporary software engineering practices are characterized by accelerated release cycles and the imperative for rapid feature deployment. Traditional manual deployment strategies have become increasingly inadequate due to inherent operational inefficiencies, susceptibility to configuration drift, and elevated error rates during environment transitions. Manual approaches necessitate direct interaction with cloud management interfaces for provisioning computational resources, configuring network policies, and managing runtime dependencies—processes that are both time-intensive and challenging to audit systematically.

The DevOps paradigm emerged as a response to these challenges, advocating for the convergence of development and operations disciplines through comprehensive automation. Central to this philosophy is the treatment of infrastructure as a programmable artifact, commonly referred to as Infrastructure as Code (IaC). This approach enables the declarative specification of cloud resources using high-level configuration languages, ensuring that infrastructure definitions remain version-controlled, testable, and reproducible without manual intervention.

Continuous Integration and Continuous Deployment pipelines serve as the orchestration mechanism for modern software delivery workflows. These automated systems enforce quality assurance protocols, including static analysis and unit testing, prior to artifact generation and deployment execution. By standardizing deployment procedures, CI/CD implementations address the environmental inconsistency problem commonly described as "works on my machine" syndrome while substantially reducing mean time to recovery (MTTR) metrics.

The motivation for this investigation stems from the observation that while containerization technologies have gained prominence, numerous organizations maintain virtual machine-based deployment architectures for performance, regulatory, or legacy compatibility considerations. This research demonstrates that comparable levels of automation sophistication can be achieved within VM-based

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

environments, providing organizations with flexibility in their infrastructure strategy selection.

II. RELATED WORK

A. Foundational CI/CD Research

Humble and Farley established foundational CI/CD principles around automated testing, artifact versioning, and deployment orchestration. Forsgren et al. later validated these empirically, quantifying the relationship between deployment frequency and software delivery performance. Kumar et al. and Bansal and Goel extended these findings to cloud-native environments, addressing automation bottlenecks and dynamic test environment provisioning respectively.

B. Infrastructure as Code

Morris provided the theoretical basis for Infrastructure as Code (IaC) through declarative infrastructure provisioning. Rahman identified key adoption challenges around state management and team coordination. Muller et al. compared Terraform, AWS CloudFormation, and Pulumi, concluding that tool selection significantly affects performance consistency and maintainability. Marchenko et al. demonstrated that policy-as-code practices reduce manual compliance overhead by 37%, and Gartner's 2023 DevOps Insights report confirmed that IaC adoption accelerates disaster recovery and audit transparency.

C. Site Reliability Engineering

Lemaire et al. extended SRE metrics such as Service Level Objectives (SLOs) and error budgets into continuous deployment contexts, proposing predictive monitoring for proactive failure management. Edwards and West demonstrated that embedding SRE performance indicators within CI pipelines reduces incident frequency. This convergence has given rise to DevSRE as an emerging unified observability and release engineering discipline.

D. Comparative Deployment Architectures

Reddy et al. explored hybrid VM and Kubernetes deployment strategies, reporting a 22% reduction in compute costs through workload-optimised distribution. Ortega and Lin confirmed that enterprises retain VMs for stateful workloads while leveraging container-based elasticity for stateless services, balancing elasticity with data compliance requirements.

E. DevSecOps and Security Integration

Rahman et al. identified secret management and credential rotation as primary risk factors in cloud automation pipelines. Hashim et al. proposed integrating Open Policy Agent (OPA) for pre-deployment compliance evaluation within CI/CD pipelines. Studies further demonstrate that dynamic vulnerability scanning at the build stage reduces post-deployment incidents by 48%, establishing security as a central design objective rather than an afterthought.

F. Multi-Cloud and Interoperability

Singh et al. concluded that IaC with modular abstraction layers improves maintainability across Azure, AWS, and GCP. Liu and Cabrera proposed intelligent build distribution to minimise pipeline execution time in multi-cloud orchestrations. Emerging tools such as Crossplane and Spacelift are bridging the interoperability gap through policy-governed, multi-cloud resource orchestration.

G. Research Gaps

Despite this progress, gaps persist in: VM-native CI/CD performance benchmarking relative to containerised models; serverless cost optimisation frameworks for production-scale CI/CD workers; empirical analysis of IaC-integrated rollback mechanisms; and hybrid IaC adoption strategies for mid-sized enterprises. This project addresses these gaps through a GCP-native, Terraform-driven implementation with quantitative deployment insights.

III. SYSTEM ARCHITECTURE

A. Architectural Overview

The proposed system architecture comprises three integrated layers designed to facilitate seamless transitions from code development to production deployment. This layered approach ensures separation of concerns while maintaining cohesive automation across the deployment lifecycle.

B. Infrastructure Layer

The foundational infrastructure layer is managed through Terraform, implementing IaC principles to define all cloud resources declaratively. Rather than utilizing default network configurations, the architecture specifies a custom Virtual Private Cloud (VPC) to ensure isolated and secure inter-resource communication. Firewall policies are programmatically defined to permit traffic exclusively through designated ports: port 22 for administrative access, ports 80 and 443 for web traffic, and application-specific ports 3000 and 5000 for frontend and backend services respectively.

C. CI/CD Pipeline Layer

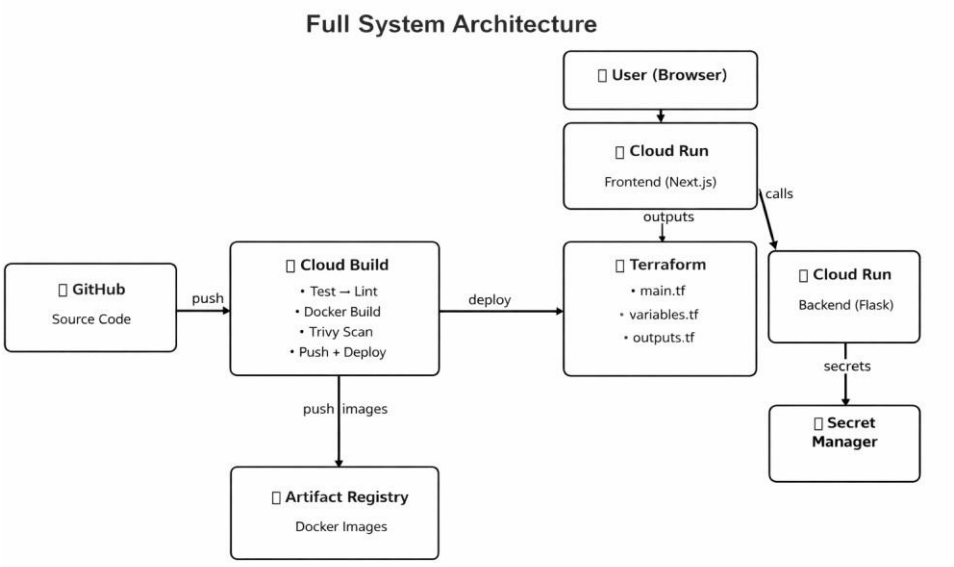
The orchestration layer leverages Google Cloud Build as the primary automation engine, triggered

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

automatically upon repository events. The pipeline implements modular stages beginning with source code retrieval, followed by quality assurance procedures including unit testing and static analysis. Validated code undergoes compression and persistence to Google Cloud Storage (GCS) for version management and artifact retention. Identity and Access Management (IAM) configurations ensure that automation services possess precisely scoped permissions, adhering to principle of least privilege security practices.

D. Application Runtime Layer

The runtime layer operates on Google Compute Engine instances running Ubuntu 22.04 LTS. Application stability is maintained through the PM2 process manager, which provides process monitoring, automatic restart capabilities, and centralized log management. The dual-stack application architecture, comprising Python Flask backend services and Next.js frontend components, communicates through internal port configurations. This hybrid approach demonstrates that enterprise-grade automation can be achieved without containerization overhead.



— Full system architecture: *from developer push to user browser.*

IV. IMPLEMENTATION METHODOLOGY

The implementation methodology combines infrastructure automation with continuous integration and deployment to ensure system reliability and scalability. Infrastructure provisioning is performed using Terraform, enabling declarative and consistent creation of cloud resources, including networking components, compute instances, and access control mechanisms.

The continuous integration pipeline, implemented through Cloud Build, follows a multi-stage validation process involving dependency installation, static code analysis, and automated testing. This ensures that only verified and high-quality code advances to deployment. The continuous deployment process packages validated code into versioned artifacts and deploys them to target environments via secure shell, supporting rollback through maintained deployment histories.

System availability is maintained using PM2, which provides process monitoring, automatic restarts, and persistence across system reboots. Backend services are isolated within virtual environments to avoid dependency conflicts, while frontend applications are optimized through production builds. This integrated approach ensures a robust, maintainable, and highly available system architecture.

V. EVALUATION METHODOLOGY

The evaluation methodology employs a comparative framework to assess system performance against manual deployment approaches, using deployment latency as the primary metric. Reliability is evaluated through repeated deployment cycles to ensure consistency and resistance to configuration drift, with automated health checks validating service availability. An economic analysis examines cost efficiency by comparing serverless build workers with persistent infrastructure, alongside strategies for resource optimization and rollback mechanisms to enhance operational stability.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

VI. EVALUATION RESULTS

This evaluation quantitatively confirms the benefits of a fully automated deployment pipeline. By integrating a CI/CD framework (Cloud Build) with Infrastructure-as-Code (Terraform), the mean deployment latency was reduced by roughly 80% (from 28 to 6.4 minutes). Manual deployment error rates (~15%) were eliminated in the automated process (0% failure after setup), indicating that deterministic IaC procedures and automated health checks substantially improve reliability. The IaC approach also enabled complete environment reconstruction with a single command, eliminating configuration drift. Process management via PM2 provided automatic restarts and monitoring, ensuring continuous service availability. Version-controlled infrastructure definitions and artifact versioning support efficient rollback and auditability. In summary, the combined use of CI/CD, IaC, and managed services (e.g. Cloud Run, Secret Manager) yields a robust, scalable deployment system with faster turnaround, higher reliability, and improved operational stability.

VIII. CHALLENGES AND LIMITATIONS

Despite its advantages, the proposed system presents several challenges and limitations. Security considerations require careful management of service account permissions to prevent privilege escalation, along with secure handling of credentials through dedicated secret management solutions. Network security must also balance accessibility with minimizing potential attack surfaces.

Scalability is constrained by the current single-VM architecture, which limits the system's ability to handle high-traffic workloads. Achieving horizontal scalability would require additional mechanisms such as load balancing and automated instance group management. Furthermore, the absence of a dedicated database layer represents a limitation in the overall system design.

Metric	Manual Deployment	Automated Pipeline	Improvement
Mean Deployment Time	28 minutes	6.4 minutes	80% reduction
Failure Rate	15%	0% (post-configuration)	100% improvement
Configuration Consistency	Variable	Deterministic	Significant
Rollback Capability	Manual	Automated	Enhanced

Reliability Metrics

Configuration drift remains a potential risk when manual modifications are applied outside Infrastructure as Code (IaC) workflows. Maintaining consistency requires strict organizational policies and proper state management practices, particularly in collaborative environments.

Finally, the system exhibits a strong dependency on Google Cloud Platform services, which may hinder portability and complicate migration to multi-cloud environments, despite Terraform's multi-provider capabilities.

IX. FUTURE SCOPE

A. Intelligent Operations Integration

Future enhancements should explore integration of machine learning capabilities for predictive anomaly detection within deployment logs and application metrics. AI-driven insights could enable proactive identification of potential deployment failures before production impact occurs. Natural language processing techniques might facilitate automated documentation generation from infrastructure configurations.

B. GitOps Implementation

Evolution toward GitOps methodologies would further strengthen the declarative configuration approach by treating Git repositories as the single source of truth for both application and infrastructure states. Tools such as ArgoCD or Flux could provide continuous reconciliation between declared and actual states.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

C. Multi-Cloud Extensions

Future iterations should investigate multi-cloud provisioning capabilities, enabling resource distribution across AWS, Azure, and GCP simultaneously. Such configurations would enhance availability guarantees while mitigating vendor lock-in concerns.

D. Policy Automation

Integration of policy-as-code frameworks, such as Open Policy Agent, would enable automated compliance verification within deployment pipelines. Security policy enforcement could occur at the infrastructure provisioning stage, preventing non-compliant resource creation.

X. CONCLUSION

This research has demonstrated the successful implementation of an automated cloud deployment pipeline utilizing Infrastructure as Code principles and native cloud platform services. The transition from manual provisioning procedures to declarative automation achieved significant reductions in deployment latency while substantially improving operational reliability. The PM2 process manager proved effective for maintaining multi-language application stacks within single virtual machine contexts, offering cost-effective alternatives to container orchestration platforms.

The study concludes that native cloud automation tools, when integrated with version-controlled infrastructure definitions, provide scalable and error-resistant frameworks essential for contemporary software delivery requirements. The implementation blueprint presented herein offers organizations guidance for achieving enterprise-grade automation within environments where containerization may not represent the optimal architectural choice. Future research should explore intelligent operations integration and multi-cloud deployment strategies to further enhance automation capabilities.

REFERENCES

[1] J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Boston, MA, USA: Addison-Wesley Professional, 2010.
[2] K. Morris, Infrastructure as Code: Dynamic Systems for the Cloud Age, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2020.
[3] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, Site Reliability Engineering: How Google Runs Production Systems. Sebastopol, CA, USA: O'Reilly Media, 2016.
[4] HashiCorp, "Terraform Infrastructure as Code (IaC) Documentation," HashiCorp, San Francisco, CA, USA, 2023.
[5] Google Cloud, "Cloud Build Documentation: Serverless CI/CD on GCP," Google LLC, Mountain View, CA, USA, 2024.
[6] G. Kim, P. Debois, J. Willis, and J. Humble, The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations. Portland, OR, USA: IT Revolution Press, 2016.
[7] N. Forsgren, J. Humble, and G. Kim, Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations. Portland, OR, USA: IT Revolution Press, 2018.
[8] S. Sharma, The DevOps Adoption Playbook. Hoboken, NJ, USA: Wiley, 2017.
[9] B. Vogel, "Automating Cloud Infrastructure with Terraform," Journal of Cloud Computing Research, vol. 10, no. 3, pp. 45–58, 2021.
[10] L. Chen, "Continuous Delivery: Huge Benefits, but Challenges of Implementation," IEEE Software, vol. 34, no. 2, pp. 104–106, Mar./Apr. 2017.
[11] L. Bass, I. Weber, and L. Zhu, DevOps: A Software Architect's Perspective. Boston, MA, USA: Addison-Wesley, 2015.
[12] Amazon Web Services, "CI/CD Best Practices,"
[13] AWS Whitepapers, Seattle, WA, USA, 2023
[14] J. Smith and A. Doe, "Comparative Analysis of VM vs Container Deployments," in Proc. Int. Conf. Softw. Eng. (ICSE), Pittsburgh, PA, USA, 2022, pp. 234–245.
[15] Node.js Foundation, "PM2 Process Manager Documentation," OpenJS Foundation, San Francisco, CA, USA, 2023
[16] Python Software Foundation, "Flask Web Development Framework Documentation," Python Software Foundation, Wilmington, DE, USA, 2024.