International Journal of Scientific Research & Engineering Trends
Volume 12, Issue 2, Mar-Apr-2026, ISSN (Online): 2395-566X

Cloud-Based Web Application Deployment Platform

Rajani Devi K, Gowri Sankar R, Gayathri Reddy R, Harsha Vardhan V, Srikanth T
Department of Artificial Intelligence & Data Science, Vasireddy Venkatadri Institute of Technology, Andhra Pradesh, India

Abstract- — In the modern software development landscape, countless developers—particularly students, beginners, and hobbyists—build innovative web applications but fail to deploy them to the internet due to the complexity of traditional deployment processes. Deploying an application requires extensive knowledge of cloud platforms such as AWS, GCP, or Cloudflare, involving technical hurdles including renting and configuring cloud instances, purchasing domains, setting up web servers, and managing infrastructure. This steep learning curve creates a significant barrier to entry, causing many developers to abandon their fully-functional applications at the development stage without ever making them publicly accessible, thereby limiting innovation visibility and preventing developers from building their portfolios.

Keywords— Modern deployment barriers, cloud complexity, infrastructure setup, hosting challenges, domain management, server configuration, steep learning curve, beginner developers, student projects, hobbyist apps, deployment friction, inaccessible cloud platforms, technical overhead, innovation stagnation, portfolio limitations, unreleased applications, DevOps difficulty.

I. INTRODUCTION

The rapid growth of web applications has intensified the demand for efficient, reliable, and accessible deployment solutions. Traditionally, deploying web applications requires manual configuration of servers, environment setup, and continuous infrastructure management.

A significant proportion of developers, particularly students and beginners, build functional web applications but never make them publicly accessible due to the steep learning curve associated with cloud platforms such as AWS, GCP, and Cloudflare. The Cloud-Based Web Application Deployment Platform is designed to eliminate these barriers by automating the entire deployment workflow through a single, user-friendly interface. The system allows developers to submit their application source code via a Git repository URL, after which the platform automatically builds the project using containerized environments and deploys the generated files to storage.

II. LITERATURE REVIEW

A review of existing deployment technologies and platforms reveals several approaches to automated application hosting, each with distinct capabilities and limitations.

Cloud-Based Deployment Platforms

Vercel is a widely adopted cloud platform that enables developers to deploy frontend applications quickly by connecting GitHub repositories directly to the platform. It provides automatic builds, scalable hosting, and global CDN delivery. However, many of its advanced features rely on proprietary managed services, making it difficult for developers to understand or replicate the underlying deployment mechanisms.

Automated Web Hosting Platforms

Netlify simplifies web application deployment by supporting automatic builds, continuous deployment from Git repositories, and serverless functions. Although it provides a user-friendly experience, it relies heavily on managed infrastructure that may limit customization and developer control. Neither Vercel nor Netlify provides automated LLM-based code analysis prior to deployment.

Automated Web Hosting Platforms

Netlify simplifies web application deployment by supporting automatic builds, continuous deployment from Git repositories, and serverless functions. Although it provides a user-friendly experience, it relies heavily on managed infrastructure that may limit customization and developer control. Neither Vercel nor Netlify provides automated LLM- based code analysis prior to deployment.

Container-Based Deployment Systems

Containerization technologies such as Docker have become widely adopted for application deployment due to their ability to create isolated and reproducible build environments. Containers package applications along with their dependencies, ensuring behavioral consistency across environments. Modern deployment platforms leverage containerized build processes

to improve automation and reliability; however, managing container orchestration requires significant DevOps expertise.

Cloud Object Storage for Deployment Artifacts
Cloud storage services such as Amazon S3 are commonly used to store application assets and deployment artifacts. These services provide scalable storage, high availability, and efficient content delivery for static files. Deployment platforms typically store build outputs in object storage systems before serving them to end users. Integration with deployment pipelines and reverse proxy mechanisms is essential to effectively deliver application content.

III. PROPOSED SYSTEM

System Overview
The Cloud-Based Web Application Deployment Platform introduces a fully automated deployment workflow designed specifically for developers who lack cloud infrastructure expertise. The platform accepts single input a Git repository URL and autonomously handles repository cloning, dependency installation, containerized build execution, artifact storage, reverse proxy configuration, and deployment URL generation.

System Architecture

Module 2 – Deployment Request The platform follows a distributed, microservices-oriented architecture comprising the following core components: API Server: Receives deployment requests from the frontend and orchestrates communication between system components.

Build Server: Clones the Git repository, installs dependencies, and executes the build command inside an isolated Docker container managed by AWS ECS. Cloud Storage (Amazon S3): Stores generated build artifacts under a unique project-ID-based directory structure. Reverse Proxy Server: Routes incoming user requests to the correct deployment directory in S3 storage. Socket Server: Provides real-time deployment log streaming from the build server to the frontend interface.

Technology Stack
Layer — Technology
Frontend — React.js, Tailwind CSS Backend — Node.js, Express.js Containerization — Docker
Cloud Storage — Amazon S3 Infrastructure — AWS ECS, AWS ECR Real-Time Communication — Socket.io Routing — Reverse Proxy

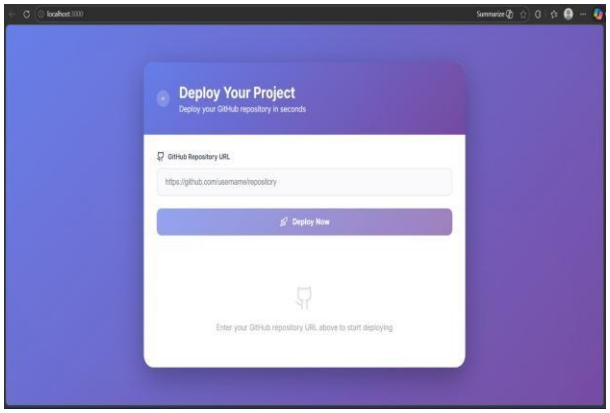
Modules
Module 1 – User Interface & Repository Submission: Provides the deployment card interface where developers submit GitHub repository URLs and monitor deployment progress.

Processing: The API server processes deployment requests and routes the repository URL to the build server for execution.
Module 3 – Build Server & Containerized Environment: Clones the repository, installs dependencies, and executes the build command inside an isolated Docker container.

Module 4 – Artifact Storage: Uploads generated build outputs (HTML, CSS, JS, assets) to Amazon S3 under a unique project ID directory. Module 5 – Reverse Proxy Routing: Routes incoming requests to the correct deployment artifacts stored in cloud storage via a reverse proxy server. Module 6 – Deployment Status Monitoring: A Socket.io-based server streams real-time build logs and deployment status updates to the frontend interface.

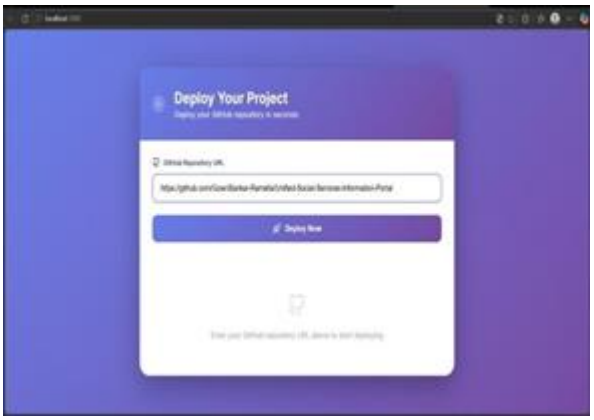
IV. SYSTEM DESIGN & IMPLEMENTATION

User Interface & Repository Submission
The platform homepage presents a minimalist deployment card where the developer enters a GitHub repository URL and clicks the Deploy Now button. The interface is intentionally simple, requiring zero infrastructure configuration from the user. Upon submission, the frontend validates the input and dispatches a deployment request to the backend API server, which initiates the full deployment pipeline.



Deployment Execution
Upon receiving the repository URL, the API server triggers the build server, which executes the following sequential operations: Clone the Git repository to the build environment. Install all project dependencies (npm install / yarn).Execute the production build command (npm run build).Generate

production-ready output files (HTML, CSS, JS bundles, static assets).Upload all generated artifacts to the designated Amazon S3 directory.Each build is executed inside an isolated Docker container managed by AWS ECS, ensuring that build environments are reproducible, consistent, and independent across concurrent deployments.

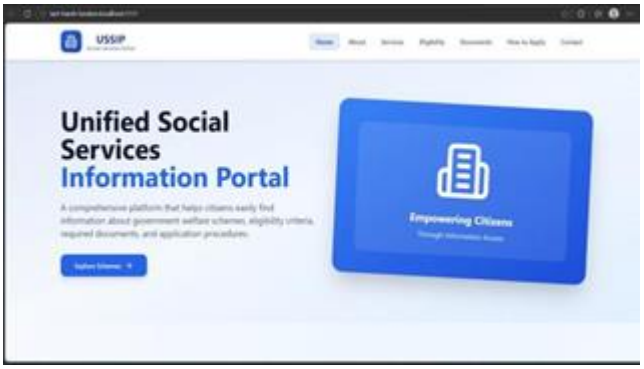


Deployment Logs & Monitoring

The platform streams real-time deployment logs to the user interface via Socket.io. These logs include build step progress, file generation details, S3 upload status, and a final deployment success or failure indicator. This transparency enables developers to identify and diagnose build failures without navigating external cloud consoles.

Accessing the Deployed Application

Upon successful deployment, the platform generates a unique Preview URL for each project. This URL is routed through the reverse proxy server, which maps incoming requests to the corresponding S3 artifact directory using the project ID. The deployed application is immediately accessible as a live website without any additional hosting configuration.



V. MODULE INTERACTION & TESTING

Module Interaction

The deployment workflow follows a well-defined inter-module communication sequence. The User Interface Module captures the repository URL and dispatches it to the API Server Module, which forwards the request to the Build Server Module. The Build Server clones the repository, builds the application, and uploads artifacts to the Cloud Storage Module. The Reverse Proxy Module then routes user access requests to the appropriate S3 directory. Simultaneously, the Deployment Monitoring Module streams real-time status updates back to the frontend via Socket.io, ensuring the developer receives continuous feedback throughout the process.

Test Strategies & Types

Unit Testing: Individual components repository submission, build execution, artifact storage are tested in isolation. Integration Testing: Verifies correct inter-module communication, such as API server–build server handoff and S3 upload confirmation. System Testing: End-to-end validation of the complete workflow from URL submission to live preview URL generation.

User Acceptance Testing: Validates platform usability with developer users to confirm intuitive operation and correct deployment outcomes.

Sample Test Cases

TC1 — Valid GitHub URL submitted — Deployment initiates successfully TC2 — Deploy Now button clicked — API server receives the request TC3 — Build server clones repository — Source code downloaded successfully

TC4 — Build process executes — Application build files are generated TC5 — Artifacts uploaded to S3 — Files stored in cloud storage

TC6 — Preview URL generated — App accessible via the URL

TC7 — Deployment logs viewed — Real-time logs displayed TC8 — Invalid URL submitted — Error message displayed

Bug Fixes & Iterations

Build Process Failures: Repositories failed to build due to missing dependencies. Resolved by pre- installing common build toolchains in the base Docker image.

Deployment Log Synchronization: Delays in log display were resolved by optimizing the Socket.io event emission frequency and frontend listener configuration.

Artifact Upload Errors: Incorrect S3 directory paths caused upload failures. Fixed by enforcing a standardized project-ID-based path structure.

VI. CHALLENGES & PROPOSED SOLUTIONS

Repository Handling & Build Environment Setup

Challenge: Different projects use varying frameworks, dependency managers, and build configurations, making it difficult to guarantee a consistent build environment. Solution: Docker containers are used to create isolated, reproducible build environments. Each build runs independently with its own dependency installation cycle, ensuring consistency across diverse project types.

Managing Concurrent Build Processes

Challenge: Executing multiple builds simultaneously risks resource contention and build interference.

Solution: The platform leverages AWS ECS for container orchestration, enabling each deployment to run in a fully isolated container instance, preventing interference between concurrent build processes.

Cloud Storage Integration

Challenge: Managing structured artifact storage across multiple deployments requires a consistent and scalable naming convention.

Solution: Each project's build output is stored in a unique S3 directory keyed by project ID, enabling the reverse proxy to accurately retrieve and serve deployment files.

Reverse Proxy Routing

Challenge: Mapping incoming user requests to the correct deployed application required an efficient and scalable routing mechanism. Solution: A reverse proxy server was implemented to parse the deployment URL, extract the project ID, and forward the request to the appropriate S3 path.

Real-Time Deployment Monitoring

Challenge: Providing live build feedback was critical for user experience; without it, developers had no visibility into the deployment process.

Solution: A Socket.io-based communication layer was implemented between the build server and frontend, enabling continuous log streaming without page refreshes or polling overhead.

VII. CONCLUSION & FUTURE SCOPE

Conclusion

The Cloud-Based Web Application Deployment Platform successfully delivers a simplified, automated approach to deploying web applications. By accepting a single Git repository URL as input, the platform orchestrates the entire deployment lifecycle—from repository cloning and containerized build execution to cloud artifact storage and live URL generation—without requiring any cloud platform knowledge from the developer.

The integration of Docker containerization, AWS ECS, Amazon S3, Socket.io real-time monitoring, and a reverse proxy routing mechanism demonstrates how distributed system components can be synthesized into a coherent, production-grade deployment platform. The system bridges the gap between developer productivity and infrastructure complexity, enabling a broader population of developers to publish and share their web applications publicly.

Future Scope

Custom Domain Support: Allow users to connect their own domain names to deployed applications.User Authentication & Project Management: Enable personal accounts for managing multiple deployments and build histories.

Improved Scalability: Introduce horizontal scaling with multiple build server instances and load balancing. Multi-Framework Support: Extend build pipeline support to backend frameworks and full-stack applications. Deployment Analytics: Provide deployment history, performance metrics, uptime monitoring, and usage dashboards.

LLM Code Review Integration: Fully operationalize the AI-powered code review engine with actionable remediation suggestions prior to deployment approval.

REFERENCES

1. Vercel Inc. (2023). Vercel Documentation: Framework Presets and Deployment.
2. Netlify Inc. (2023). Netlify Documentation: Continuous Deployment from Git.
3. Merkel, D. (2014). Docker: Lightweight Linux Containers for Consistent Development and Deployment. Linux Journal.
4. Amazon Web Services. (2023). Amazon S3 Developer Guide: Storing and Retrieving Objects.
5. Amazon Web Services. (2023). Amazon ECS Documentation: Running Containers at Scale.
6. Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, Omega, and Kubernetes. ACM Queue.
7. Newman, S. (2015). Building Microservices: Designing Fine-Grained Systems. O'Reilly Media.
8. Fette, I., & Melnikov, A. (2011). The WebSocket Protocol. IETF RFC 6455.
9. Richardson, L., & Ruby, S. (2007). RESTful Web Services. O'Reilly Media.
10. Rajput, A. (2022). Cloud-Native Application Deployment: Patterns and Practices. IEEE Software.