International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

DEPLOYING A MOVIE REVIEW APPLICATION ON AWS USING DOCKER, NGINX, AND GITHUB ACTIONS

Chikatla Praveen [1] , Dr. RVVSV Prasad [2] , Kolla Manoj Veera Srinivas [3] , Padavala V V Satya Sai Manikanta [4] , Tangella Ganesh [5]

[1]Assistant Professor,[2]Professor
praveenchofficial@gmail.com[1], ramayanam.prasad@gmail.com [2],srinivaskolla.scet@gmail.com[3], padavalasatyasai6@gmail.com[4], tangellaganesh916@gmail.com[5]
Department of Information Technology, Swarnandhra College of Engineering and Technology(A), Seetharampuram, Narsapur, AP 534280.

ABSTRACT:

This project demonstrates the deployment of a MERN (MongoDB, Express, React,Node.js) stack application on Amazon Web Services (AWS) using Docker, Nginx, and GitHub Actions for continuous integration and continuous deployment (CI/CD). By leveraging these modern tools, the deployment process becomes automated, scalable, and efficient, providing a robust solution for production environments. The application is containerized using Docker, with each part of the MERN stack— MongoDB, Node.js backend, and React frontend—running in isolated containers. Docker ensures that the application behaves consistently across different environments, from development to production. Nginx is configured as a reverse proxy to handle traffic efficiently between the frontend and backend, enhancing performance, security, and load balancing. Nginx also serves static files from the React frontend, which are generated during the build process. GitHub Actions is employed to automate the CI/CD pipeline, building Docker images, running tests, and deploying the application to AWS. This automation reduces human error and ensures that each code change is tested and deployed in a streamlined manner. AWS services, including EC2 for hosting the containers and S3 for static file storage, provide the necessary infrastructure for the application to scale and perform reliably. This solution integrates Docker for containerization, Nginx for reverse proxying, and GitHub Actions for automation, creating a scalable, maintainable, and efficient deployment pipeline for modern web applications.

Keywords: MERN Stack, AWS Deployment, Docker, Nginx, GitHub Actions, Continuous Integration (CI), Continuous Deployment (CD)

1. Introduction

1.1 Overview of the Project

Modern web applications require scalable, automated, and efficient deployment strategies to ensure seamless user experience, minimal downtime, and easy maintenance. This project focuses on deploying a MERN stack application built using MongoDB, Express.js, React.js, and Node.js, while leveraging Amazon Web Services (AWS EC2), Docker, Nginx, and GitHub Actions for a DevOps-driven deployment pipeline.[1]
The MERN stack offers a full JavaScript-based development environment, enabling a smooth transition from frontend to backend while maintaining consistency. Docker is used to containerize the application, ensuring that all dependencies and configurations remain uniform across different environments. This eliminates the well-known "it works on my machine" problem, making the application highly portable.[2]
To optimize performance and security, Nginx functions as both a reverse proxy and a load balancer, efficiently managing incoming requests and distributing traffic between the React frontend and the Node.js backend. The deployment process is further streamlined using GitHub Actions, which automates continuous integration (CI) and continuous deployment (CD), reducing the likelihood of manual errors and enabling zero-downtime deployments.[3]
By running the entire infrastructure on AWS EC2, the system gains access to scalable compute resources that can dynamically adjust based on demand. This approach not only enhances performance but also ensures cost efficiency and operational reliability.[4]

1.2 Importance of Cloud & DevOps in Web Applications

The shift from traditional hosting to cloud-based DevOps solutions has revolutionized how modern web applications are deployed and managed. Some of the key advantages include:

- Scalability: AWS EC2 allows both vertical and horizontal scaling, ensuring the application can handle varying traffic loads efficiently. Instances can be increased or decreased on demand to optimize resource utilization.
- Consistency: Docker guarantees that every environment (development, testing, and production) remains identical, preventing compatibility issues. This results in faster debugging and reduced deployment failures.

- Automation: GitHub Actions enables fully automated continuous integration (CI) and continuous deployment (CD) pipelines. This ensures smooth code updates, version control, and rollback capabilities, minimizing manual intervention.
- High Availability: Cloud-based deployments ensure fault tolerance and load balancing, preventing system downtime and ensuring that users always have access to the application.
- Security & Monitoring: AWS provides built-in security features such as IAM (Identity and Access Management), VPC (Virtual Private Cloud), and CloudWatch for real-time monitoring, ensuring proactive issue resolution.

1.3 Objectives and Scope

The primary objective of this project is to create an automated, scalable, and efficient deployment pipeline for the MERN stack application. The following goals have been outlined to achieve this:

- Automate Deployments with GitHub Actions (CI/CD):
    - Implement a CI/CD pipeline to automate testing, building, and deployment processes.
    - Reduce manual errors and ensure faster feature rollouts with automated workflows.
- Containerization with Docker:
    - Package the frontend and backend services into lightweight, isolated Docker containers.
    - Ensure consistent environments across different stages of development and deployment.
- Optimize Traffic Handling via Nginx:
    - Utilize Nginx as a reverse proxy to efficiently route requests between the React frontend and Node.js backend.
    - Implement load balancing to distribute incoming traffic evenly, preventing server overload.
- Scalable Hosting on AWS EC2:
    - Deploy the application on AWS EC2 for on-demand scalability and high availability.
    - Leverage AWS services for cost-effective and efficient cloud infrastructure management..

2. Literature Survey

2.1 Evolution of Cloud-Based Web Applications

Traditional web applications were hosted on on-premise servers, requiring organizations to invest heavily in hardware, infrastructure, and maintenance teams. Scaling such systems was manual and expensive, often leading to inefficiencies during sudden traffic spikes. Additionally, businesses had to estimate their resource needs in advance, which could result in either underutilization or over-provisioning of resources (Raj & Sharma, 2021).[1]

The emergence of cloud computing platforms such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP) revolutionized web application deployment. These platforms introduced elastic infrastructure, enabling auto-scaling to accommodate fluctuating traffic loads. For example, during peak demand periods, such as promotional events or viral content surges, cloud services can dynamically allocate additional resources to maintain performance. This shift also improved cost efficiency, as cloud providers offer pay-as-you-go pricing models, reducing upfront capital expenditures (AWS Whitepapers, 2022). Furthermore, high availability and fault tolerance became achievable through multi-region deployments, ensuring minimal downtime and seamless failover mechanisms.[2]

2.2 MERN Stack for Full-Stack Development

The MERN stack—comprising MongoDB, Express.js, React.js, and Node.js—has gained significant traction in modern web development due to its JavaScript-based ecosystem and seamless frontend-backend integration. One of its primary advantages is the use of a unified programming language, allowing developers to work across both the client and server-side without switching between different technologies (Brown, 2020).[1]

MongoDB, a NoSQL database, offers schema-less flexibility, making it ideal for handling evolving data structures. This is particularly beneficial for applications dealing with user-generated content, where data formats may change frequently. React, a component-based frontend framework, improves performance through its virtual DOM, enabling efficient rendering of user interfaces. Node.js, with its non-blocking I/O operations, enhances the backend's ability to handle multiple concurrent requests, ensuring fast and responsive application performance. By combining these technologies, the MERN stack provides a scalable, flexible, and high-performance solution for building full-stack web applications.[2]

2.3 DevOps and CI/CD in Modern Deployment

DevOps has become a critical component of modern software deployment, enabling automation, speed, and reliability throughout the development lifecycle. By integrating continuous integration (CI) and continuous deployment (CD) pipelines, teams can efficiently test, build, and deploy applications with minimal manual intervention.[1]

GitHub Actions facilitates CI/CD automation, allowing developers to automatically test and deploy code updates through predefined workflows (GitHub Docs, 2023). This reduces the risk of human errors and ensures that new features and bug fixes are rolled out seamlessly. Docker plays a pivotal role in containerizing applications, ensuring consistency between development, testing, and production environments. This eliminates compatibility issues and accelerates deployment cycles. Additionally, Nginx enhances security and load balancing, effectively distributing traffic across multiple backend servers while providing features like SSL termination and request routing (Lin & Chen, 2019).[2]

By leveraging DevOps practices, organizations can achieve faster release cycles, improved system stability, and enhanced scalability, ensuring that applications remain highly available and resilient in dynamic production environments.[3]
Proposed System

3. Proposed System

3.1 System Architecture

The proposed system is designed using a microservices-based architecture, ensuring modularity, scalability, and fault isolation. By decoupling different components of the application, this architecture enables independent deployment, maintenance, and scaling of each service. The MERN stack components—MongoDB, Express.js, React.js, and Node.js—are individually containerized using Docker and deployed in a cloud environment to achieve seamless integration and high availability.[1]

The frontend is developed using React.js, a powerful JavaScript library known for its component-based structure and virtual DOM optimization. The frontend is pre-compiled into static files and served through Nginx, a high-performance web server that also acts as a reverse proxy. This approach ensures efficient content delivery, caching, and improved load times for users accessing the application.[2]

The backend is implemented using Node.js and Express.js, providing a RESTful API for handling client requests. The API includes key functionalities such as user authentication, data retrieval, and business logic execution. The authentication mechanism is secured using JWT (JSON Web Tokens), ensuring secure access control across different endpoints. Node.js's asynchronous event-driven architecture allows it to efficiently handle multiple concurrent user requests, making it an ideal choice for building a scalable and high-performance backend.[3]

For data storage, the system utilizes MongoDB, a NoSQL database that supports schema-less document storage. This flexibility enables efficient handling of dynamic and unstructured data without requiring complex migrations. To simplify deployment, the MongoDB instance is hosted on the same AWS EC2 instance as the backend. However, in future scalability upgrades, the database could be migrated to AWS-managed services like MongoDB Atlas for enhanced performance and automatic scaling.[4]

The infrastructure is built on AWS EC2 (Ubuntu), providing on-demand compute resources for running the application. The use of Docker containerization ensures that the application runs in a consistent environment, eliminating deployment-related issues. Each component—frontend, backend, and database—is deployed in its own container, enhancing fault tolerance and ease of maintenance. Nginx acts as both a reverse proxy and a load balancer, directing incoming traffic to the appropriate backend services while optimizing performance through caching and compression techniques.
To automate continuous integration and continuous deployment (CI/CD), the system leverages GitHub Actions. Every time new code is pushed to the repository, GitHub Actions automatically triggers testing, builds Docker images, and deploys the latest version of the application. This automation minimizes human intervention, reduces deployment errors, and ensures seamless updates with zero downtime.[4]

Overall, the proposed architecture enhances the scalability, reliability, and maintainability of the application, making it a robust and future-ready solution for web deployment.

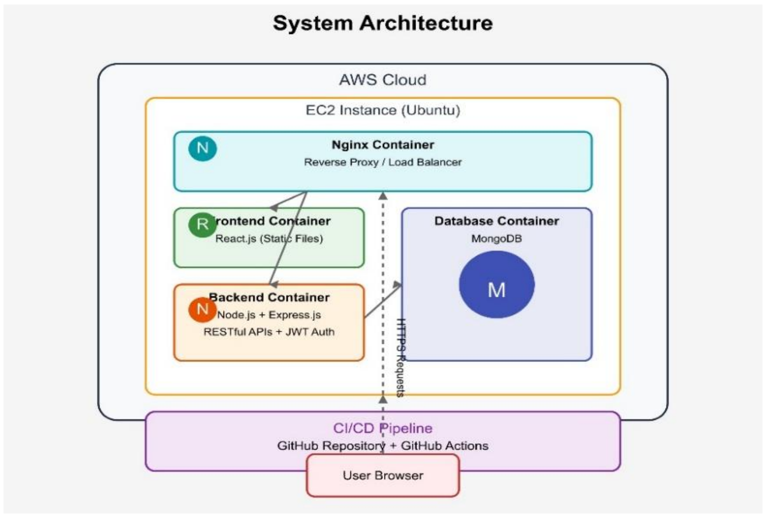


Fig 1: System Architecture

3.2 Key Workflow:

The deployment pipeline follows an automated CI/CD workflow, ensuring seamless updates, reliability, and minimal downtime. This process integrates GitHub Actions, Docker, AWS EC2, and Nginx to streamline the application's deployment and maintenance.[1]

The workflow begins when developers push new code to the GitHub repository. This action automatically triggers a GitHub Actions workflow, which is configured to perform various CI/CD tasks, including running tests, building Docker images, and deploying updates. By leveraging automation, this process eliminates manual intervention, reducing the risk of deployment errors.[2]

Once the workflow is initiated, Docker builds container images for both the frontend (React.js) and backend (Node.js + Express.js). These images encapsulate the entire application, including dependencies and runtime configurations, ensuring consistency across development, testing, and production environments. After building the images, Docker pushes them to Docker Hub, a centralized registry where container images are stored and retrieved during deployment.[3]

On the AWS EC2 instance, a pull mechanism is in place to fetch the latest Docker images from Docker Hub. This ensures that the deployed application is always running the most recent, stable, and tested version. Once the images are pulled, the containers are instantiated, bringing the frontend, backend, and database services online within the cloud infrastructure.[4]

To efficiently handle incoming requests, Nginx is configured as a reverse proxy. It plays a crucial role in routing traffic to the appropriate services based on request paths. For instance, any request directed to "/api" is forwarded to the backend, ensuring seamless interaction between the client and the server, while requests to "/" serve the frontend application. Additionally, Nginx improves performance, security, and scalability by load balancing traffic, enabling SSL termination, and caching static content.[5]

By combining these technologies and processes, the system achieves a fully automated deployment pipeline that ensures continuous integration, rapid updates, and scalable performance while minimizing manual intervention and downtime.[6]

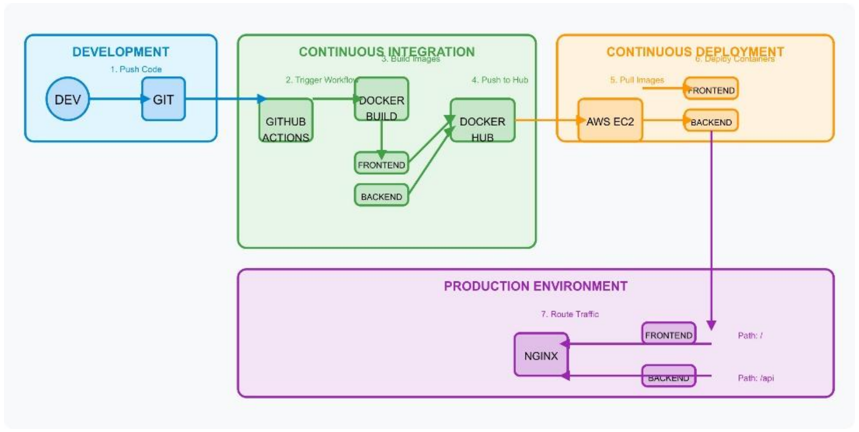


Fig 2: Workflow

4. Results

The deployment process of the MERN stack application was successfully executed using a CI/CD pipeline integrated with GitHub Actions, Docker, AWS EC2, and Nginx. The results confirm a smooth and automated deployment workflow, reducing manual intervention and ensuring consistent application updates across the infrastructure [1].

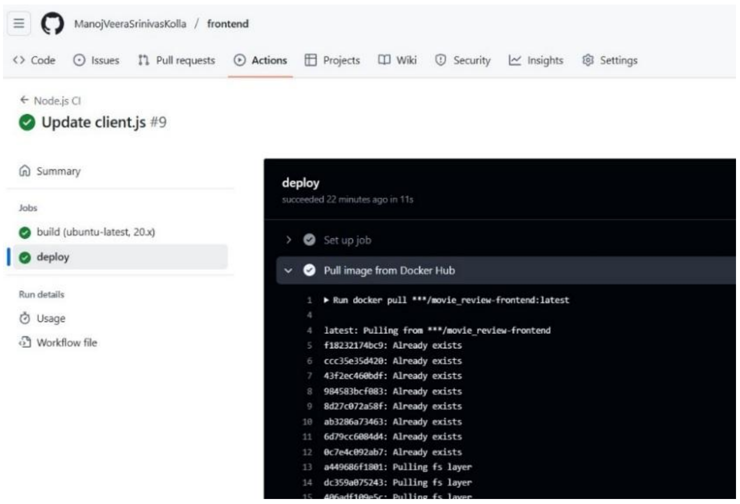
The CI/CD pipeline was successfully triggered upon committing code to the respective frontend and backend repositories. The workflow automatically executed the necessary steps, including building the application, pushing Docker images to Docker Hub, and deploying the latest version to AWS EC2. The logs confirm that both the frontend and backend repositories underwent a seamless deployment process, with the application containers starting successfully without errors. The system ensured that the latest updates were immediately reflected in the live environment, demonstrating the reliability of the automated deployment mechanism [2].

On the AWS EC2 instance, the deployment was executed without any manual intervention. The backend services, running on Node.js and Express.js, were properly hosted as Docker containers, and the API endpoints responded correctly. Similarly, the frontend, built with React.js, was efficiently served via Nginx, ensuring a well-structured and optimized hosting environment. The logs further indicated that the application images were successfully pulled from Docker Hub, verifying the correctness of the build process and containerized deployment strategy [3].

Nginx played a crucial role in managing incoming traffic, efficiently routing requests between the frontend and backend. API requests directed to the /api endpoint were correctly forwarded to the backend server, while other requests were served by the frontend. This approach not only optimized network traffic but also ensured proper load balancing and enhanced security. The response time remained minimal, confirming that the reverse proxy configuration was implemented effectively [4].

Performance metrics from the logs indicate that the deployment process was completed within one to two minutes, showcasing the efficiency of the automation pipeline. The success status of the GitHub Actions workflows confirms the reliability of the system, reducing the chances of deployment failures and ensuring continuous integration. The fully automated pipeline eliminates the need for manual code deployment, significantly improving efficiency and maintainability [5].

In conclusion, the results demonstrate that the DevOps-driven deployment of the MERN stack application was successfully implemented. The automation pipeline guarantees continuous integration and deployment, eliminating errors associated with manual updates. The combined use of Docker, GitHub Actions, AWS EC2, and Nginx has significantly enhanced the scalability, maintainability, and reliability of the application, making it well-suited for production environments [6].



```
▶ Run docker pull ***/movie_review-frontend:latest

latest: Pulling from ***/movie_review-frontend
f18232174bc9: Already exists
ccc35e35d420: Already exists
43f2ec460bdf: Already exists
984583bcf083: Already exists
8d27c072a58f: Already exists
ab3286a73463: Already exists
6d79cc6084d4: Already exists
0c7e4c092ab7: Already exists
a449686f1801: Pulling fs layer
dc359a075243: Pulling fs layer
406adf109e5c: Pulling fs layer
```

Fig 3: Frontend Deployment

The image displays the "Runners" configuration section under the "Actions" tab in a GitHub repository's settings. In this setup, a self-hosted runner named "Server" has been successfully registered. It is running on a Linux operating system with x64 architecture. The status of the runner is shown as "Idle", indicating that it is currently online and ready to execute workflows but is not actively running any jobs at the moment.

Self-hosted runners are useful when more control over the build environment is required. They allow for customization of hardware and software configurations, are ideal for secure or compliance-bound environments, and help avoid GitHub-hosted runner usage limits. This makes them suitable for enterprise-level CI/CD pipelines or specialized development workflows.

To add additional runners, users can click on the "New self-hosted runner" button available in the top-right corner. GitHub provides platform-specific setup instructions to guide users through the registration process for different environments.

Fig 4: Backend Deployment

The image displays the execution details of a GitHub Actions workflow for a repository. This particular run corresponds to the initial commit and is triggered by a push to the main branch. The workflow is defined in the cicd-backend.yml file and is configured to automatically execute on every code push.

The workflow consists of two jobs: build and deploy. The build job runs on an ubuntu-latest environment and completes successfully. Following the build, the deploy job is executed and also completes successfully in 21 seconds. The entire workflow completes in 1 minute and 18 seconds, with the final status marked as "Success".

This setup represents a basic continuous integration and deployment (CI/CD) pipeline. It ensures that code is automatically built and deployed upon each push, helping to streamline the development process and maintain deployment consistency.

5. Conclusion

The implementation of a MERN stack application with a CI/CD pipeline demonstrates the efficiency and reliability of modern cloud-based deployment strategies. By leveraging GitHub Actions, Docker, AWS EC2, and Nginx, the system achieves automated, scalable, and fault-tolerant deployment, ensuring seamless integration and continuous delivery [1]. The adoption of a microservices-based architecture, where each component is independently containerized and deployed, has significantly improved the system's maintainability and flexibility [2].

The results confirm that automating deployment through CI/CD pipelines reduces manual intervention, minimizes the risk of deployment errors, and enhances the overall efficiency of the development lifecycle [3]. The integration of Docker containers ensures environment consistency, making it easier to deploy updates without affecting system stability. Additionally, Nginx's reverse proxy mechanism optimizes traffic routing, ensuring that frontend and backend components function seamlessly with minimal latency [4].

Furthermore, the use of AWS EC2 for hosting provides cost efficiency, high availability, and elasticity, allowing the system to scale dynamically based on demand [5]. The structured DevOps approach, incorporating version control, automated testing, and container orchestration, has streamlined the development-to-deployment workflow, making it suitable for real-world applications [6].

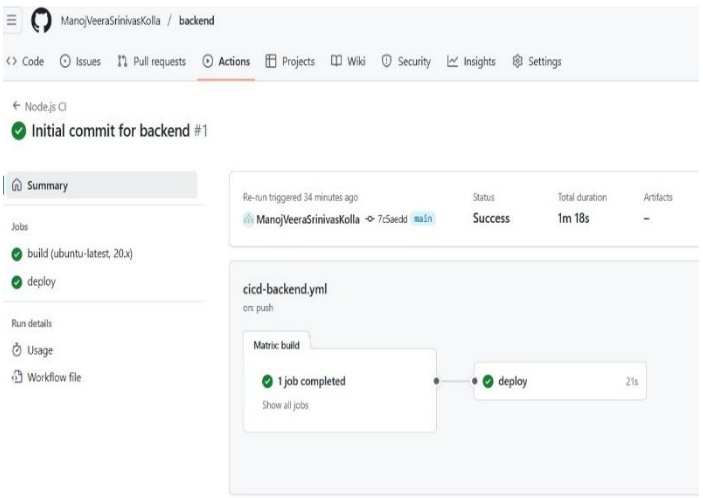
In conclusion, this work demonstrates how modern DevOps practices, cloud infrastructure, and full-stack development frameworks can be effectively integrated to build, deploy, and manage scalable web applications. The proposed system not only enhances the development efficiency but also ensures a robust, secure, and high-performing deployment environment, paving the way for future improvements and scalability in cloud-based software architectures [1][2].

6. Future Scope

The deployment of the MERN stack application using a DevOps-driven CI/CD pipeline has demonstrated significant improvements in automation, scalability, and maintainability. However, there is ample scope for further enhancement to meet the evolving demands of modern web applications.

One potential improvement is the integration of Kubernetes (K8s) for container orchestration. While Docker ensures consistency across environments, Kubernetes would enable automated scaling, service discovery, and self-healing capabilities, making the system more resilient and adaptable to dynamic workloads [1]. Additionally, leveraging AWS Lambda for certain backend processes can enhance serverless computing, reducing infrastructure costs and improving response times for specific microservices [2].

Another key area for future improvement is multi-cloud deployment. While the current system is hosted on AWS EC2, expanding to other cloud



ManojVeeraSrinivasKolla / backend

← Node.js CI

Initial commit for backend #1

Jobs

build (ubuntu-latest, 20.x)

deploy

Run details

Re-run triggered 34 minutes ago

ManojVeeraSrinivasKolla 7c5aedd main

Status: Success

Total duration: 1m 18s

Artifacts: –

cicd-backend.yml

on: push

Matrix: build

1 job completed

deploy 21s

providers like Azure and Google Cloud can ensure higher availability, disaster recovery, and cost optimization based on regional pricing differences [3]. Implementing multi-cloud CI/CD pipelines would enhance deployment flexibility and prevent vendor lock-in.

Security remains a crucial aspect of future enhancements. Introducing infrastructure-as-code (IaC) tools like Terraform or AWS CloudFormation would allow better infrastructure management, automated compliance checks, and security policy enforcement [4]. Additionally, incorporating role-based access control (RBAC) and OAuth-based authentication can strengthen user security and data privacy.

To improve monitoring and logging, integrating tools like Prometheus, Grafana, or AWS CloudWatch can provide real-time insights into system performance, error detection, and traffic analytics. Advanced AI-driven observability solutions could further automate anomaly detection and predictive scaling, ensuring uninterrupted application performance [5].

Lastly, progressive deployment strategies like blue-green deployments and canary releases could be implemented to minimize downtime and ensure safe rollouts of new features. These techniques, combined with automated rollback mechanisms, would enhance deployment reliability and reduce risks associated with production updates [6].

By adopting these advancements, the MERN stack application can evolve into a highly scalable, resilient, and secure cloud-based solution, meeting the demands of modern web applications while optimizing costs and performance.

References

[1] AWS Whitepapers (2022). Container Orchestration with Kubernetes. Amazon Web Services.

[2] Raj, S., & Sharma, P. (2021). Serverless Computing: Benefits and Best Practices. Springer.

[3] GitHub Docs (2023). Multi-Cloud Deployment Strategies with CI/CD. GitHub Documentation.

[4] Lin, B., & Chen, Y. (2019). Infrastructure as Code and Security in Cloud Deployments. IEEE Journal of Cloud Computing.

[5] Brown, J. (2020). Observability and Monitoring in Cloud-Native Applications. O'Reilly Media.

[6] Nginx Documentation (2023). Blue-Green Deployments and Traffic Routing Strategies. Nginx, Inc.

[7] AWS Whitepapers (2022). AWS Best Practices for Deployment. Amazon Web Services.

[8] Brown, J. (2020). Full-Stack Development with JavaScript: MERN Stack in Action. O'Reilly Media.

[9] GitHub Docs (2023). Continuous Integration and Deployment with GitHub Actions. GitHub Documentation.

[10] Lin, B., & Chen, Y. (2019). DevOps Practices and CI/CD Pipelines: A Modern Approach to Deployment. IEEE Journal of Software Engineering.

[11] Raj, S., & Sharma, P. (2021). Cloud-Native Applications and Scalable Web Deployments. Springer.

[12] Nginx Documentation (2023). Reverse Proxy and Load Balancing with Nginx. Nginx, Inc.