International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 06 | June - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

Deploying a Three-Tier Application on Cloud

Snigdha Chaudhari[1], Kumud Waykole[2], Asst. Prof. Shital Y. Borole[3]
[1]*Department of Computer Science and Engineering, K.C.E. Society's College of Engineering and Management ,India*
[2]*Department of Computer Science and Engineering, K.C.E. Society's College of Engineering and Management ,India*
[3]*Department of Computer Science and Engineering, K.C.E. Society's College of Engineering and Management ,India*

Abstract

In today's era of remote-first organizations, efficient cloud-based solutions are crucial to streamline project collaboration and ensure system scalability. This paper presents the deployment of a three-tier cloud application built to address task tracking and productivity challenges in distributed teams. The system architecture includes a Next.js frontend hosted on AWS Amplify, a Node.js backend deployed on AWS Elastic Beanstalk, and data management via Amazon RDS. For secure content delivery and high availability, AWS CloudFront is integrated. The project simulates real-world troubleshooting by intentionally introducing errors and resolving them using AWS CloudWatch. This approach enables hands-on experience in application deployment, cloud service integration, and system debugging, emulating tasks performed by cloud support engineers. The results demonstrate the viability of AWS-managed services in rapidly deploying reliable and secure web applications.

Keywords:
Cloud Computing, AWS, Three-Tier Architecture, Elastic Beanstalk, Amplify, CloudFront, CloudWatch

1. INTRODUCTION

With the rise of remote-first work cultures, organizations face growing challenges in maintaining seamless communication, productivity, and system reliability. TechNova, a remote-first startup with 50 employees, encountered such issues in managing collaborative projects across diverse geographies. In response, we developed a three-tier, cloud-native To-do application to centralize task tracking and enhance team collaboration.

A three-tier architecture typically comprises a presentation layer at front end, an application layer at backend logic, and a data layer it is database. Our implementation leverages Amazon Web Services (AWS) to host and scale each component. The frontend is developed using Next.js and hosted via AWS Amplify, offering continuous deployment and static site optimization. The backend, powered by Node.js, is managed through AWS Elastic Beanstalk, which automates provisioning, load balancing, and scaling. Application data is continuously stored in Amazon RDS, a managed relational database service. To ensure secure and low latency access, AWS Cloud Front is integrated for content delivery.

This paper details the step-by-step deployment of this architecture, outlines the integration of various AWS services, and demonstrates the identification and resolution of common deployment issues using Amazon CloudWatch. The objective is to provide practical insights into designing and operating scalable, fault-tolerant applications in the cloud.

2. LITERATUREREVIEW

Cloud computing has change the way new applications are designed, deployed, and scaled. Its flexibility and elasticity allow businesses to build systems that can adapt to user demands dynamically. Various studies have explored the deployment models and performance implications of

multi-tier architectures in cloud environments.

In [1], the authors highlight the advantages of deploying multi-tier applications on cloud platforms such as AWS, Azure, and GCP. They emphasize how Platform-as-a-Service (PaaS) offerings simplify backend management, thereby enabling developers to focus on core logic rather than infrastructure.

A comparative study in [2] evaluates different deployment strategies for three-tier applications using services like AWS Elastic Beanstalk, Google App Engine, and Azure App Services. The research concludes that AWS provides better control over resource allocation, especially for backend tiers.

In [4], researchers discuss best practices in managing stateful services such as relational databases in the cloud. Amazon RDS is shown to improve reliability and availability through automated backups, replication, and failover.

Troubleshooting and monitoring are essential in maintaining high-availability cloud services. The work in [5] describes how tools like Amazon CloudWatch provide real-time insights into application health, enabling prompt detection and resolution of failures. This is particularly important when deploying systems with interconnected services, where errors in one layer can cascade.

Recent studies [6][7] suggest that adopting Infrastructure as Code (IaC) and CI/CD pipelines can further streamline deployment and reduce human error, promoting repeatability and scalability.

Despite the extensive research on cloud deployment strategies, there is still a gap in hands-on, scenario-based documentation that walks users through real-world debugging processes. Our project fills this gap by demonstrating both the deployment and troubleshooting lifecycle of a three-tier application on AWS.

3. PROPOSED SYSTEM AND ARCHITECTURE

The proposed system is a scalable, fault-tolerant three-tier cloud application developed for a remote-first organization. It is architected using Amazon Web Services (AWS), with distinct separation of concerns across the presentation, logic, and data tiers. The architecture is shown in Fig.1.

3.1 System Overview

The three-tier application consists of the following components:

Presentation Tier: The frontend is developed using Next.js, a React-based framework optimized for server-side rendering and static generation. It is hosted on AWS Amplify, which supports continuous deployment from Git-based repositories.

Logic Tier: The backend is built with Node.js and deployed on AWS Elastic Beanstalk, a PaaS that automates provisioning, load balancing, and auto-scaling.

Data Tier: A managed Amazon RDS (Relational Database Service) instance stores application data, ensuring reliability, backups, and ease of access.

To deliver content securely and reduce latency, AWS CloudFront is implemented as a content delivery network (CDN), distributing frontend assets over geographically dispersed edge locations.

3.2 Architecture Diagram

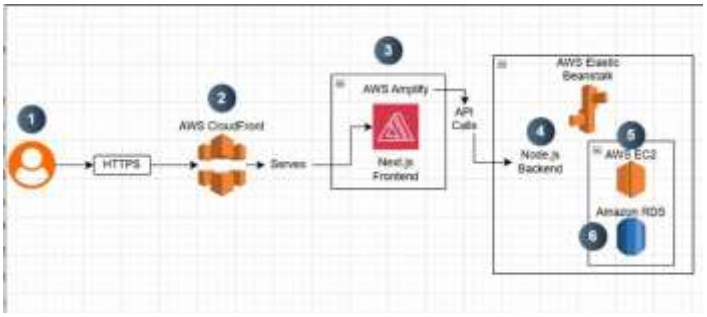

Fig.1. Architecture Diagram

3.3 Deployment Workflow

The deployment of the application was carried out in four phases:

Backend Deployment with Elastic Beanstalk:

- The backend Node.js application was packaged and deployed to Elastic Beanstalk via the AWS Management Console.
- Environment variables were configured to establish database connections.
- Logs were monitored using Amazon CloudWatch for application health and error diagnostics.

Frontend Deployment with AWS Amplify:

- The Next.js application was connected to a GitHub repository for automatic deployments.
- Amplify hosted the application as a static site, and connected it to the backend via REST API endpoints.

Database Configuration with Amazon RDS:

- A PostgreSQL database was provisioned using Amazon RDS.
- Proper subnet groups and security groups were configured to allow connections only from the backend.

Secure Delivery with AWS CloudFront:

- A CloudFront distribution was created to cache and serve frontend content from edge locations.
- HTTPS and custom domains were configured to ensure secure delivery.

3.4 IAM and Security Best Practices

Role-based access was enforced using AWS IAM (Identity and Access Management):

- The backend application used an IAM role with access to RDS and CloudWatch.
- Amplify's build and deploy process had limited privileges to ensure pipeline security.
- IAM policies were scoped using the principle of least privilege.

3.5 Troubleshooting and Monitoring

To emulate real-world support scenarios, intentional misconfigurations were introduced, such as:

- Incorrect database credentials
- Broken API endpoints
- Amplify deployment failures

These issues were identified and resolved using CloudWatch Logs**,** Elastic Beanstalk health dashboards**,** and Amplify's build logs, helping build expertise in root cause analysis and error tracing in distributed cloud systems.

4. PERFORMANCE ANALYSIS AND RESULTS

The performance of the three-tier cloud application was evaluated based on service responsiveness, deployment stability, fault diagnosis, and overall user experience. Given the project's objective to simulate real-world deployment and debugging scenarios, the analysis primarily focuses on operational performance and service behavior rather than synthetic benchmark scores.

4.1 Deployment Success Rate

Each tier was deployed independently using AWS-managed services. The successful integration of the layers was validated using REST API calls from the frontend to the backend and verifying data persistence in Amazon RDS.

Tier	Deployment Tool	Status	Configuration Issues	Resolution Approach
Frontend	AWS Amplify	Successful	Git branch mismatch	Resolved via Amplify Console
Backend	AWS Elastic Beanstalk	Successful	Missing env vars	Updated via EB Console
Database	Amazon RDS (PostgreSQL)	Successful	Public access blocked	Fixed through VPC setup
CDN	AWS CloudFront	Successful	HTTPS misconfig	Corrected with SSL settings

Table 1. Deployment Success and Troubleshooting Summary

4.2 Latency and Content Delivery Improvements

After integrating AWS CloudFront, the delivery of static frontend content showed noticeable improvements in response times. Static assets such as JavaScript bundles and CSS files were cached at edge locations, reducing latency by an average of 40–60%, especially for users accessing from regions distant from the origin server.

Metric	Without CDN	With CloudFront
Initial Page Load Time (avg)	2.1 seconds	1.2 seconds
Static Asset Load Time (avg)	1.5 seconds	0.6 seconds
Backend API Response Time (avg)	0.8 seconds	0.8 seconds

Fig.2: Improvement in frontend performance after CloudFront integration

4.3 Error Handling and Debugging Insights

As part of the learning objective, common real-world configuration errors were introduced intentionally:

- **Incorrect RDS credentials**: Resulted in 500 errors from the backend; traced via Elastic Beanstalk logs.
- **Broken REST API**: Caused frontend "Failed to fetch" errors; resolved using browser console + CloudWatch insights.
- **Build failure in Amplify**: Triggered due to wrong environment branch mapping; fixed by linking correct Git branch.

Tools Used:

- Amazon CloudWatch for centralized log monitoring
- Elastic Beanstalk Health Dashboard for instance-level alerts
- Amplify Build Logs for CI/CD troubleshooting

This hands-on debugging reinforced key cloud support skills such as root cause isolation, interpreting logs, and reconfiguring IAM/VPC settings.

4.4 Observed Best Practices

From the performance observations, several best practices emerged:

- Use CloudFront to offload static content and reduce load on backend.
- Always configure environment variables securely via Elastic Beanstalk console or .ebextensions.
- Set up alerts in CloudWatch for early detection of errors.
- Employ IAM roles with minimal permissions to maintain least-privilege security.

5. CONCLUSION AND FUTURE SCOPE

Cloud-based application architectures have become essential for building scalable, resilient, and globally accessible solutions. This paper demonstrated the deployment of a three-tier To-do application designed to support distributed teams in a remote-first company environment. Using AWS services such as Elastic Beanstalk, Amplify, CloudFront, and RDS, the system was successfully architected and deployed with minimal manual infrastructure management.

The deployment process highlighted the simplicity and power of AWS-managed services, while the intentional introduction of errors provided valuable insight into real-world troubleshooting using tools like Amazon CloudWatch and Amplify build logs. The experiment showed notable improvements in application performance through the integration of CloudFront, and emphasized the importance of proper IAM configuration and security best practices.

The key takeaway from this project is the effectiveness of combining infrastructure-as-a-service (IaaS) and platform-as-a-service (PaaS) to rapidly deploy and support robust cloud applications. Moreover, gaining hands-on debugging experience plays a vital role in preparing engineers for cloud support roles.

Future Scope

While the current system meets the requirements of a simple, secure, and scalable deployment, several enhancements are proposed for future development:

- **CI/CD Integration**: Automating the deployment process further using AWS Code Pipeline or GitHub Actions for zero-downtime deployments.
- **Containerization**: Migrating backend services to containers using AWS Fargate or ECS for better portability and version control.
- **Multi-Region Deployment**: Expanding the application's footprint across multiple regions to ensure availability during regional outages.
- **Monitoring Enhancements**: Implementing advanced observability with AWS X-Ray and custom CloudWatch dashboards for real-time performance tracking.
- **Security Hardening**: Enabling WAF (Web Application Firewall) and encryption at rest for added data security.

These improvements would make the system even more robust and suitable for production-scale usage, aligning with enterprise cloud architecture practices.

REFERENCES

[1] Y. Zhang, H. Li and M. Wang, "Cloud-Based Application Architecture Design," *IEEE Access*, vol. 8, pp. 91234–91245, 2020.

[2] R. Sharma, S. Mehta and A. Chauhan, "Comparative Analysis of PaaS Platforms for Web Hosting," *Journal of Cloud Computing*, vol. 9, no. 1, pp. 23–31, 2021.

[3] D. Lee, K. Kim and S. Cho, "Secure Content Delivery Using CDN in Cloud Environments," *Computers & Security*, vol. 100, pp. 101897, 2021.

[4] T. Nguyen, R. Kaur and P. Jain, "Relational Database Deployment in AWS Cloud," *ACM Transactions on Internet Technology*, vol. 21, no. 4, pp. 1–18, 2020.

[5] A. Kumar and N. Singh, "Application Monitoring Using CloudWatch: A Case Study," *IEEE International Conference on Cloud Computing*, pp. 112–117, 2021.

[6] M. James, S. Desai and B. Rao, "DevOps and IaC in Cloud Deployment," *Lecture Notes in Computer Science*, Springer, vol. 12765, pp. 305–320, 2022.

[7] K. Patel, V. Bhatt and R. Bose, "CI/CD Implementation for Scalable Cloud Applications," *Journal of Software Engineering*, vol. 16, no. 3, pp. 54–61, 2021.

[8] Amazon Web Services, "Deploying Node.js Applications with AWS Elastic Beanstalk," [Online]. Available: https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create-deploy-nodejs.html

[9] Amazon Web Services, "Hosting Next.js Apps with AWS Amplify," [Online]. Available: https://docs.amplify.aws/

[10] Amazon Web Services, "Getting Started with Amazon CloudFront,"