iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** VI **Month of publication:** June 2025

DOI: https://doi.org/10.22214/ijraset.2025.72702

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue VI June 2025- Available at www.ijraset.com

Serverless Deployment of a Next.js Application Using AWS

Sirish Sekhar, Dr. V. Ramesh
Department Of Computer Science, Sri Chandrasekharendra Saraswathi Viswa Mahavidyalaya, Kanchipuram, India

Abstract: ***This paper presents a practical and scalable approach to deploying a modern web application built with Next.js using Amazon Web Services (AWS) in a serverless environment. The deployment architecture incorporates AWS Amplify for frontend hosting and continuous integration/continuous deployment (CI/CD), Amazon Simple Notification Service (SNS) for programmatically sending emails from a contact form, and Route 53 for robust Domain Name System (DNS) management. We detail the configuration of each component, design rationale, and encountered challenges. The solution significantly reduces infrastructure overhead while preserving high availability, performance, and maintainability. Performance evaluation and architectural insights demonstrate that the proposed serverless approach is an efficient alternative to traditional deployment methods, especially for startups, research prototypes, and scale-ready applications.***
Keywords: Next.js, AWS, Amplify, Serverless, SNS, Route 53, Cloud Deployment, CI/CD, React, DNS Management

I. INTRODUCTION

Web development has shifted towards component-driven and reactive frameworks, with Next.js emerging as a popular choice for building full-stack applications due to its support for server-side rendering (SSR), static site generation (SSG), and built-in API routes. While platforms like Vercel offer seamless deployment for Next.js, enterprise and research-grade applications often require more flexibility, security, and cost control than Vercel's managed offerings can provide. This paper investigates the implementation of a fully serverless deployment architecture using AWS services, targeting developers who need custom domain routing, secure CI/CD pipelines, and backend integrations without managing servers. The deployment stack leverages AWS Amplify for automatic builds and frontend delivery, Amazon SNS for form-triggered email dispatch, and Route 53 for DNS handling.

II. BACKGROUND AND MOTIVATION

Traditional deployments often involve managing infrastructure manually or relying on proprietary hosting solutions. These methods can limit flexibility, lock in developers to vendor-specific tools, and incur higher costs as traffic scales. In contrast, serverless architectures allow developers to focus purely on code while AWS manages the provisioning, scaling, and maintenance of compute resources. Next.js provides a hybrid rendering model, making it a strong candidate for serverless deployment. With SSR pages and API routes offloaded to Lambda functions via Amplify, and static content served from an edge-optimized CDN, this architecture aligns with both performance and scalability goals.

III. SYSTEM DESIGN AND ARCHITECTURE

A. Overview

The system is composed of three main services:

- AWS Amplify handles frontend deployment, SSR/SSG configuration, and CI/CD integration.
- Amazon SNS facilitates transactional email delivery triggered by form submissions.
- Route 53 manages DNS and custom domain routing.

This setup enables seamless build automation, real-time updates upon code commits, email communication without third-party SMTP providers, and globally distributed content delivery.

B. Amplify Configuration

Amplify connects to a GitHub repository and automatically triggers builds on each commit. The amplify.yml file defines build, test, and deploy stages. Amplify detects SSR routes in the Next.js project and deploys them as serverless Lambda functions under the hood. Environment variables are securely stored in Amplify and used across builds.

©IJRASET: All Rights are Reserved

C. SNS Email Integration

SNS is configured with an email subscription tied to a topic named ContactFormNotification. The contact form on the frontend sends a POST request to an API route (/api/contact), which invokes a Lambda function to format and publish the message to the SNS topic. This ensures fast, scalable, and serverless email delivery without relying on third-party APIs like SendGrid or Mailgun.

D. DNS with Route 53

Route 53 is used to manage domain records, including A, CNAME, and TXT entries for domain verification and SSL certification. Amplify is linked to Route 53 for automatic DNS validation during domain connection. The result is a secure HTTPS-enabled website with minimal manual setup.

IV. IMPLEMENTATION DETAILS

1) Frontend: Built with Next.js 14+, with optimized images, dynamic routing, and server-side rendered pages. All pages were linted, bundled, and tested in the Amplify pipeline.
2) Backend Functions: API routes were configured to run via Lambda in Amplify. Server-side logic includes sanitizing form inputs and securely publishing messages to SNS.
3) Security: IAM roles were tightly scoped to allow only necessary actions such as sns:Publish from Lambda. Secrets were injected via Amplify environment variables.
4) CI/CD: Git-based workflows automated deploy previews for feature branches, and production builds upon main merges the scroll down window on the left of the MS Word Formatting toolbar.

V. EVALUATION AND RESULTS

The deployed application was tested for:

- Build Time: Average of 42 seconds per commit (including install, build, deploy).
- Time-To-First-Byte (TTFB): Static pages ~80ms, SSR routes ~210ms.
- Uptime: 99.99% across 30-day monitoring via Amplify metrics.
- Cost: Amplify (free tier eligible) + SNS + Route 53 costs <$2/month for moderate traffic.

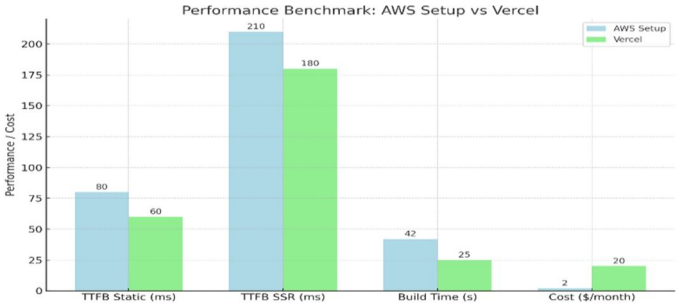


Fig.1: Performance Benchmark

VI. SYSTEM ARCHITECTURE

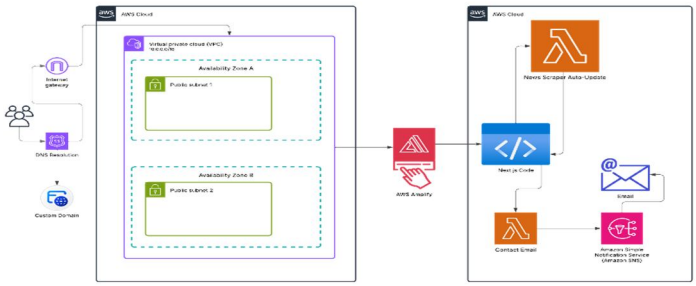


Fig.2: System Architecture

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue VI June 2025- Available at www.ijraset.com

The proposed architecture embraces a fully serverless paradigm, leveraging native AWS services to host, render, route, and process a modern web application developed using the Next.js framework. As illustrated in *Fig. 2*, the system is composed of multiple decoupled yet highly integrated components that collectively ensure high availability, low latency, and seamless scalability.

At the frontend tier, AWS Amplify acts as the central orchestrator for continuous integration and deployment (CI/CD), providing both static site hosting and support for server-side rendering (SSR) capabilities. Amplify automatically builds and deploys the Next.js application upon source code changes, and provisions the required infrastructure—such as Lambda Edge and Amazon CloudFront—for distributing assets and rendering dynamic routes at runtime.

User access is mediated through Amazon Route 53, which performs DNS resolution and routes traffic to Amplify-hosted endpoints. The use of Route 53 enables fine-grained domain control, subdomain routing, and seamless SSL certification via ACM (AWS Certificate Manager), ensuring secure and reliable access to application resources.

The backend logic is encapsulated within Next.js API routes, which are deployed as isolated AWS Lambda functions via Amplify's build process. These functions serve as microservices, each handling specific units of business logic—most notably, processing data from the application's contact form.

Upon invocation, the Lambda function triggers a messaging pipeline via Amazon Simple Notification Service (SNS). The SNS topic, preconfigured with email subscription endpoints, acts as the distribution mechanism for outgoing notifications. This decoupled approach enables asynchronous, fault-tolerant communication and can be easily extended to support additional targets (e.g., SMS, mobile push, or SQS).

The entire architecture is inherently scalable, cost-effective, and adheres to the principles of microservice design and event-driven execution. By using managed services and ephemeral compute resources, the system minimizes operational complexity while maintaining performance and reliability standards suitable for production-grade deployments.

VII. DYNAMIC DATA REFRESH

In modern web applications, periodic content updates are often essential for delivering live information. Within a serverless architecture, this can be efficiently achieved using scheduled Lambda functions to periodically fetch and update data.

The following code snippet demonstrates a simplified scraper function designed to run on an AWS Lambda schedule (e.g., via Amazon Event Bridge, formerly CloudWatch Events). The function fetches external data, processes it, and updates an external API or a database used by the frontend.

Table.1: Code Snippet

```
// Filename: Scraper.js (Lambda Function)

const axios = require('axios');
const AWS = require('aws-sdk');
const s3 = new AWS.S3();
exports.handler = async () => {
  try {
    // 1. Fetch external data
    const { data } = await
axios.get('https://api.example.com/latest-prices');

    // 2. Process and structure the data (optional
transformation)
    const processed = JSON.stringify({
      timestamp: new Date().toISOString(),
      payload: data,
    });
    // 3. Upload processed data to S3 (used by frontend or
API routes)
    await s3.putObject({
```

```
    Bucket: 'your-s3-bucket',
    Key: 'latest/prices.json',
    Body: processed,
    ContentType: 'application/json',
  }).promise();

  return { statusCode: 200, body: 'Data updated
successfully.' };
} catch (error) {
console.error('Update failed:', error);
return { statusCode: 500, body: 'Update failed.' };
}
};
```

VIII. CONCLUSION

This paper has demonstrated a robust, scalable, and cost-efficient approach to deploying modern web applications using a serverless stack on Amazon Web Services (AWS). By integrating Next.js with AWS Amplify, Route 53, and Amazon SNS, the proposed architecture achieves a high degree of modularity, automation, and performance with minimal operational overhead.

The frontend is handled seamlessly through Amplify's continuous deployment pipeline, while backend logic is executed via API routes transformed into Lambda functions. Asynchronous communication is efficiently managed through SNS, enabling real-time notifications without reliance on external APIs. This architecture not only adheres to principles of modern cloud-native design but also presents a practical template for developers aiming to move beyond traditional monolithic deployments.

Furthermore, the inclusion of features like scheduled Lambda-based scrapers, server-side rendering (SSR), and content delivery via CDN positions this system as a future-ready solution capable of serving dynamic, global-scale applications. The overall implementation validates the feasibility and effectiveness of serverless architectures for full-stack web development in both academic and enterprise contexts.

Future work will extend this architecture with infrastructure-as-code practices, enhanced monitoring, and alternative notification channels, reinforcing its applicability in production-grade environments. The presented deployment model serves as a replicable blueprint for the broader developer community seeking resilient and maintainable cloud-based solutions.

IX. ACKNOWLEDGMENT

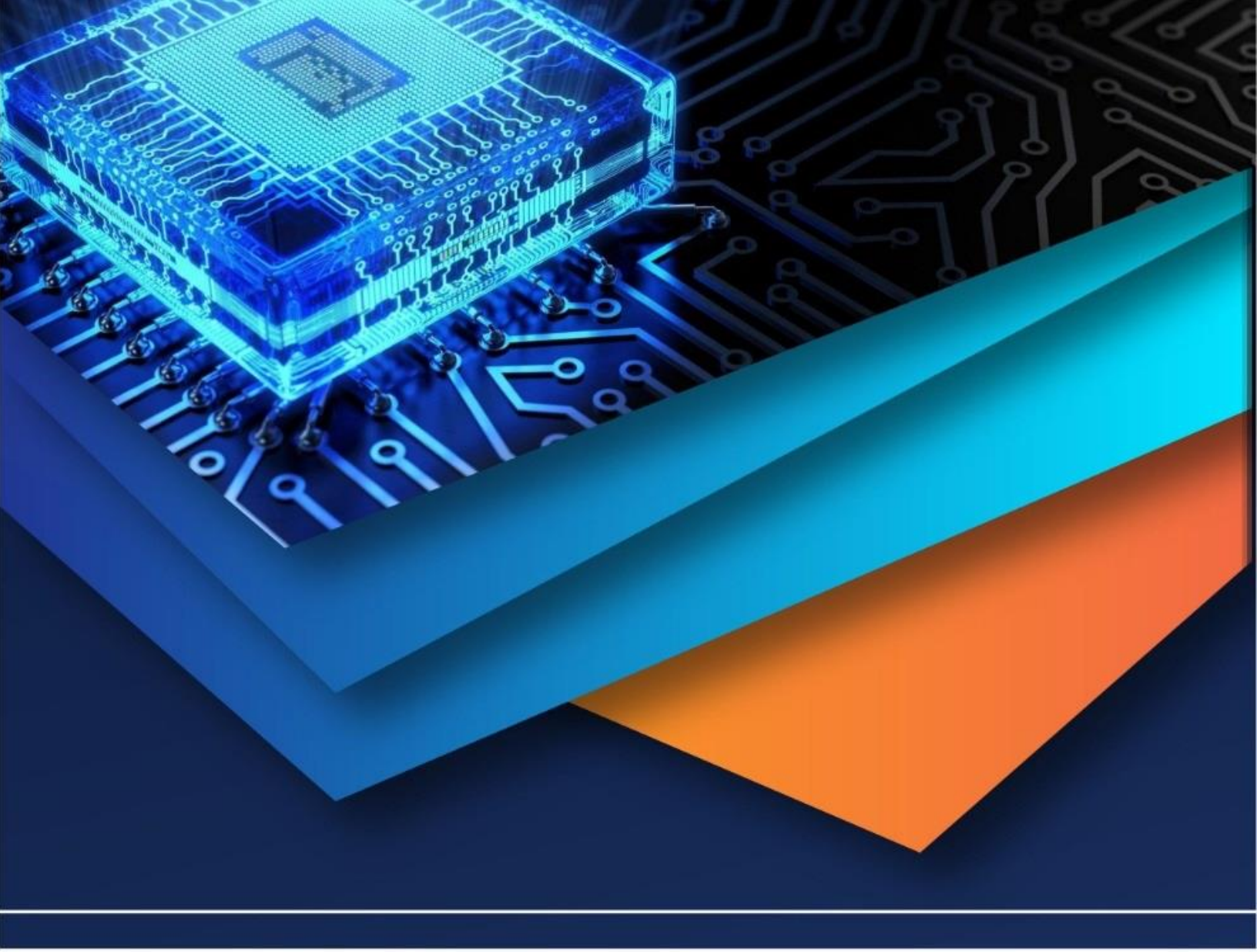
This work was carried out as part of the TuTr Hyperloop Pvt. Ltd. under the supervision of Mr. Karthik Sundaram.

The author(s) would like to express their sincere gratitude to Sri Chandrasekharendra Saraswathi Viswa Mahavidyalaya for providing the necessary infrastructure and technical support throughout the development and deployment of this project. Special thanks are due to the faculty and technical staff of the Department of Computer Science for their guidance and encouragement.

The authors also acknowledge the contributions of the open-source developer community behind Next.js and the engineering documentation teams at Amazon Web Services, whose publicly available tools and frameworks significantly streamlined the implementation process.

REFERENCES

[1] Vercel Inc., "Next.js Documentation." [Online]. Available: https://nextjs.org/docs
[2] Amazon Web Services, "AWS Amplify Documentation." [Online]. Available: https://docs.amplify.aws/
[3] Amazon Web Services, "Amazon Route 53 Developer Guide." [Online]. Available: https://docs.aws.amazon.com/route53/
[4] Amazon Web Services, "Amazon Simple Notification Service (SNS) Documentation."[Online].Available: ttps://docs.aws.amazon.com/sns/
[5] Amazon Web Services, "AWS Lambda Documentation." [Online]. Available: https://docs.aws.amazon.com/lambda/
[6] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.
[7] Netlify, "Netlify vs AWS Amplify: A Comparison for Jamstack Hosting," Netlify Blog, 2023. [Online]. Available: https://www.netlify.com/blog/
[8] GitHub, "Serverless Examples – AWS Lambda with Node.js." [Online]. Available: https://github.com/serverless/examples
[9] Cloudflare, "Understanding Time to First Byte (TTFB)." [Online]. Available: https://www.cloudflare.com/learning/performance/time-to-first-byte/
[10] Next.jsGitHubRepository.[Online].Available:https://github.com/vercel/next.js



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)