IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 3, March 2026

DOI: 10.17148/IJARCCE.2026.15367

DEPLOY A MULTI-REGION APPLICATION WITH DOCKER AND AWS ROUTE 53

Pon Vignesh P[1], Gowri Lakshmi K S[2]

III BCA, Department of Computer Applications, Sri Ramakrishna College of Arts & Science (Autonomous),

Coimbatore – 641006, Tamil Nadu, India[1]

Assistant Professor / Guide, Department of Computer Applications,

Sri Ramakrishna College of Arts & Science (Autonomous), Coimbatore – 641006, Tamil Nadu, India[2]

Abstract: The modern digital economy demands "always-on" application availability, where even brief downtime can result in significant financial loss and eroded consumer trust. Deploying a multi-region application using Docker and AWS Route 53 represents a transition from traditional high availability to a "global-first" architecture. While standard cloud deployments often rely on multiple Availability Zones within a single geographic region, a truly resilient system must account for regional-level failures such as large-scale power outages, natural disasters, or fiber optic interruptions. Docker enables organizations to package microservices into immutable, portable units that function identically across regions — whether hosted in Virginia, Dublin, or Sydney — eliminating "environmental drift" that historically plagued cross-site disaster recovery.

The technical foundation rests on deploying Amazon Elastic Container Service (ECS) or Amazon Elastic Kubernetes Service (EKS) clusters across geographically distinct AWS regions. Each region acts as a fully functional, autonomous silo with its own VPC, subnets, and Application Load Balancers (ALBs). Docker images are stored in Amazon Elastic Container Registry (ECR) with cross-region replication, ensuring low deployment latency. AWS Route 53 manages DNS and intelligently routes users to the nearest healthy region, providing sub-second response times and high fault tolerance.

Keywords: Docker, AWS Route 53, Multi-Region Deployment, Cloud Computing, High Availability, Containerization, DNS Routing, Fault Tolerance

I. INTRODUCTION

In the current era of global digitalization, the demand for "always-on" application availability has transitioned from a competitive advantage to a fundamental business requirement. Deploying a multi-region application using Docker and AWS Route 53 represents a sophisticated architectural response to the inherent vulnerabilities of single-site hosting.

While traditional cloud deployments rely on multiple Availability Zones within a single geographic area, they remain susceptible to regional-scale disruptions such as massive power grid failures, undersea cable cuts, or extreme weather events. By distributing containerized workloads across geographically distinct AWS regions — such as North America, Europe, and Asia — organizations can achieve fault tolerance that ensures service continuity even during catastrophic infrastructure outages.

The synergy between Docker's containerization and AWS Route 53's global traffic management provides a robust framework for both disaster recovery and performance optimization. Docker provides the unit of isolation needed to manage complex microservices at scale, while Route 53 acts as the "intelligent director," steering users to the most appropriate regional endpoint based on health, proximity, and network latency.

1.1 System Specification

This project focuses on creating a highly available and scalable web application using Docker containerization and AWS cloud infrastructure deployed across multiple AWS regions to ensure low latency, high performance, and fault tolerance. The architecture integrates:

- Docker for containerized application deployment
- AWS EC2 instances for hosting containers
- Application Load Balancer for distributing traffic
- AWS Route 53 for DNS management and traffic routing
- Multiple AWS regions for redundancy and disaster recovery

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 3, March 2026

DOI: 10.17148/IJARCCE.2026.15367

Hardware Requirements

Component	Requirement
Processor	Intel Core i5 / AMD Ryzen 5 or higher
RAM	Minimum 8 GB (Recommended 16 GB)
Storage	256 GB SSD or higher
Network	Broadband Internet connection
Architecture	64-bit system

Table 1: Development System Hardware Requirements

Cloud Infrastructure (AWS EC2 Configuration)

Component	Specification
Instance Type	t2.micro / t3.micro
vCPU	1
RAM	1 GB
Storage	8–30 GB EBS
Network Performance	Low to Moderate

Table 2: AWS EC2 Instance Configuration

II. SYSTEM STUDY

System study involves analyzing the existing system, identifying its limitations, and understanding the requirements for developing a new system. In modern web applications, users expect fast response time, continuous availability, and reliable performance. However, traditional application deployment methods often fail to meet these expectations, especially when hosted on a single server or region.

2.1 Existing System

The existing system refers to the traditional method of deploying web applications where the application is hosted on a single server or a single cloud region. All users access the application through the same server regardless of their geographic location. Applications are deployed directly on servers using Apache or Nginx, with all components running on the same machine or within the same data center.

This setup works for small-scale applications but becomes inefficient when user traffic increases. The deployment process is usually manual and time-consuming. If the server fails, the entire application becomes unavailable.

2.2 Drawbacks of Existing System

- Single Point of Failure: If the server fails, the entire system becomes unavailable
- Limited Scalability: Cannot easily handle increasing user traffic
- High Downtime: Server maintenance or hardware failure causes service interruption
- Slow Response Time: Users far from the server experience higher latency
- Manual Deployment Process: Prone to configuration errors
- Lack of Fault Tolerance: No automatic backup or failover system
- Difficult Maintenance: Complex and error-prone as application grows

2.3 Proposed System

The proposed system overcomes the limitations of traditional deployment using modern cloud computing technologies. It combines Docker containerization with Amazon Web Services (AWS) to create a highly reliable, scalable, and efficient multi-region application deployment system.

The application is first packaged into Docker containers and deployed on AWS EC2 instances located in multiple geographic regions. Each region hosts a copy of the application inside Docker containers. AWS Route 53 acts as a DNS service that manages user traffic, routing requests to the nearest or healthiest region. Load balancing distributes incoming

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 3, March 2026

DOI: 10.17148/IJARCCE.2026.15367

requests across multiple servers, and automatic failover ensures uninterrupted service even when a region becomes unavailable.

2.4 Features

- High Availability: System remains available even if one region fails
- Docker Containerization: Consistent execution across all environments
- Multi-Region Deployment: Improves reliability and ensures continuous service
- DNS Traffic Management: AWS Route 53 directs users to the nearest server
- Latency-Based Routing: Requests routed to the server with lowest latency
- Load Balancing: Traffic distributed across multiple servers
- Automatic Failover: Traffic redirected to healthy regions automatically
- Scalability: Easily scale by adding more containers or servers
- Cost Efficiency: Pay-as-you-use cloud pricing model
- Security Management: AWS IAM roles and security groups protect the system

III. SYSTEM DESIGN AND DEVELOPMENT

System design focuses on planning the architecture, selecting technologies, and implementing the system. The proposed system follows a multi-region cloud deployment model where the application is containerized using Docker and deployed in multiple AWS regions, with AWS Route 53 managing DNS and traffic routing.
The architecture consists of the following main components: User Client (Web Browser), AWS Route 53 (DNS Service), Application Load Balancer, AWS EC2 Instances, Docker Containers, and Monitoring Services (CloudWatch). When a user accesses the application, the request is sent to Route 53, which determines the best region to serve the request, then forwards it to the load balancer, which distributes traffic to the appropriate EC2 instance.

3.1 System Architecture

The multi-region architecture follows a layered approach:

- Client Layer: Web browser or mobile app sending HTTP/HTTPS requests
- DNS Layer: AWS Route 53 receives requests and routes based on latency and health
- Load Balancer Layer: Application Load Balancer distributes traffic to EC2 instances
- Compute Layer: AWS EC2 instances running Docker containers per region
- Container Layer: Docker containers hosting the application with all dependencies
- Monitoring Layer: AWS CloudWatch for performance tracking and alerting

3.2 File Design

Project files are organized in a structured manner to support Docker containerization and cloud deployment:

- index.html, style.css, script.js — Frontend application files
- app.py / server.js — Backend application logic
- Dockerfile — Instructions to build the Docker image
- docker-compose.yml — Manages multiple containers
- Environment variable files and server configuration files
- Deployment scripts and infrastructure configuration files
- Log files for debugging and monitoring

3.3 Database Design

The database is designed to store application data, user information, logs, configuration details, and system records. It supports high availability, reliability, and scalability across multiple AWS regions.

Field Name	Data Type	Description
User_ID	Integer (PK)	Unique identifier
Name	Varchar	User name
Email	Varchar	User email
Password	Varchar	Encrypted password
Created_Date	Date	Account creation date

Table 3: User Table Structure

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 3, March 2026

DOI: 10.17148/IJARCCE.2026.15367

IV. SYSTEM DEVELOPMENT

System development involves building the containerized web application using Docker and deploying it on AWS cloud infrastructure with multi-region support. The development process follows these steps:

Step 1: Application Development
The web application is developed using HTML, CSS, and JavaScript for the frontend and Node.js or Python for the backend. The application is tested locally before containerization.

Step 2: Docker Containerization
A Dockerfile is created defining the base OS image, dependencies, application code, and startup commands. The Docker image is built and tested locally, then pushed to AWS Elastic Container Registry (ECR) for cross-region access.

Step 3: Cloud Infrastructure Setup
AWS EC2 instances are launched in multiple regions (e.g., us-east-1 and ap-south-1). Application Load Balancers are configured per region to distribute traffic across container instances.

Step 4: DNS and Route 53 Configuration
AWS Route 53 is configured with latency-based routing policies and health checks. When a user enters the domain name, Route 53 analyzes the request and directs it to the nearest and healthiest server region.

Sample Code (app.js)

```
const express = require('express'); const app = express(); const REGION = process.env.AWS_REGION || "us-east-1"; app.get('/', (req, res) => { res.send(`Hello from region: ${REGION}`); }); app.listen(3000);
```

4.1 Testing and Implementation

Multiple types of testing were performed to ensure system reliability and performance:

- Unit Testing: Each module tested individually (Docker build, API endpoints)
- Integration Testing: Modules combined and tested together (containers with AWS services)
- System Testing: Complete system tested across multiple AWS regions
- Performance Testing: Response time and scalability measured under high traffic
- Security Testing: Authentication, IAM roles, and secure communication verified

4.2 Evaluation Results

The system was tested to evaluate performance, scalability, and response time. API requests were executed across multiple regions to verify functionality and routing.

Operation	Response Time	Scalability	Region
Data Retrieval	< 150 ms	High	us-east-1
Data Creation	< 200 ms	High	us-east-1
Data Update	< 180 ms	High	ap-south-1
Data Deletion	< 170 ms	High	ap-south-1
Failover Routing	< 30 sec	Automatic	Cross-Region

Table 4: System Performance Evaluation Results

Results confirm that the multi-region serverless architecture significantly reduces infrastructure overhead while maintaining efficient performance. The system automatically scales according to incoming request volume, making it suitable for modern global cloud-based applications.

 This work is licensed under a Creative Commons Attribution 4.0 International License

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 3, March 2026
DOI: 10.17148/IJARCCE.2026.15367

V. CONCLUSION

This paper presented the design and implementation of a multi-region application deployment system using Docker containerization and Amazon Web Services (AWS). The main objective was to develop a system that provides high availability, scalability, and improved performance for users across different geographic locations.

Docker was used to package the application and its dependencies into containers, ensuring consistent execution across environments. The containerized application was deployed on AWS EC2 instances in multiple regions, improving system reliability and reducing the risk of service interruption. AWS Route 53 played an important role in managing DNS routing and directing user requests to the nearest and healthiest server.

The multi-region deployment architecture provides several advantages including fault tolerance, faster response time, easy scalability, and better resource utilization. Even if one server or region becomes unavailable, the system continues operating by redirecting traffic to another active region. This project demonstrates how modern cloud technologies like Docker and AWS can be used to build reliable and scalable applications suitable for global deployment.

VI. FUTURE ENHANCEMENTS

The system can be further enhanced in several ways:

- Integration of Kubernetes for automated container orchestration and scaling
- Implementation of AWS Auto Scaling for dynamic resource management
- CI/CD pipeline using Jenkins or GitHub Actions for automated deployment
- Content Delivery Network (CDN) integration using AWS CloudFront
- Microservices architecture for better scalability and flexibility
- Database replication across regions for improved data availability
- Integration of serverless computing using AWS Lambda
- Advanced monitoring using Prometheus and Grafana
- AI/ML-based traffic prediction and resource optimization

REFERENCES

[1]. Amazon Web Services, "AWS Route 53 Developer Guide," AWS Documentation, 2023.
[2]. Amazon Web Services, "Docker on AWS — Deployment Guide," AWS Official Guide, 2023.
[3]. Docker Inc., "Docker Documentation," https://docs.docker.com/, 2023.
[4]. M. Roberts, "Serverless Architectures," Martin Fowler Blog, 2018.
[5]. P. Jamshidi et al., "Serverless Computing: A Survey of Opportunities and Challenges," IEEE Cloud Computing, 2020.
[6]. AWS Whitepaper, "Building Multi-Region Applications," Amazon Web Services, 2022.
[7]. R.S. Pressman, Software Engineering: A Practitioner's Approach, McGraw-Hill Education.
[8]. I. Sommerville, Software Engineering, Pearson Education.

 This work is licensed under a Creative Commons Attribution 4.0 International License