**SAVEETHA SCHOOL OF ENGINEERING**

**Saveetha Institute of Medical and Technical Sciences**

**Chennai – 602 105**

**DESIGN AND IMPLEMENTATION OF A SCALABLE AND COST-EFFECTIVE WEB APPLICATION ON AWS**

**A CAPSTONE PROJECT REPORT**

***Submitted to***

**SIMATS ENGINEERING**

****

**By**

**V. Jaya chandra Reddy (192211440)**

***Supervisor***

**Dr. V. GOKULA KRISHNAN**

**CSA1584 – CLOUD COMPUTING AND BIG DATA ANALYTICS FOR MEDICAL APPLICATIONS**

**June – 2024**

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **1** | **Introduction** | **3** |
| **2** | **Project Planning** | **3** |
| **3** | **System Architecture** | **4** |
| **4** | **Implementation** | **5** |
| **5** | **Evaluation** | **7** |
| **6** | **Conclusion** | **8** |
| **7** | **Future work** | **8** |
| **8** | **References** | **8** |

**DESIGN AND IMPLEMENTATION OF A SCALABLE AND COST-EFFECTIVE WEB APPLICATION ON AWS**

**1. Introduction:**

The objective of this project is to design and implement a scalable and cost-effective web application on Amazon Web Services (AWS). This report details the planning, architecture, implementation, and evaluation of the web application, leveraging various AWS services to ensure scalability, reliability, and cost-efficiency**.**

**2. Project Planning:**

**1) Requirements Gathering:**

* Some of the Functional Requirements are:
  + User authentication and authorization.
  + CRUD operations for user data.
  + Responsive user interface.
  + RESTful API for data access.
* Some of the Non-Functional Requirements are:
  + High availability and scalability.
  + Cost-effectiveness.
  + Security and compliance.
  + Performance and load balancing.

**2) Technology Stack:**

* Frontend:
  + HTML, CSS, JavaScript
  + React.js
* Backend:
  + Node.js with Express.js framework
  + RESTful API
* Database:
  + Amazon RDS (Relational Database Service) for MySQL
* Infrastructure:
  + Amazon EC2 (Elastic Compute Cloud)
  + Amazon S3 (Simple Storage Service) for static file storage
  + Amazon CloudFront for Content Delivery Network (CDN)
  + AWS Lambda for serverless computing (optional for certain tasks)
  + Amazon Route 53 for DNS management
  + AWS IAM (Identity and Access Management) for security

**3. System Architecture:**

**1) Architectural Overview:**

The web application follows a multi-tier architecture:

* **Presentation Layer:** Hosted on Amazon S3 and delivered via Amazon CloudFront.
* **Application Layer:** Deployed on Amazon EC2 instances within an Auto Scaling group.
* **Database Layer:** Managed by Amazon RDS with Multi-AZ (Availability Zone) deployment for high availability.

**2) AWS Services Utilized**

* **Amazon EC2:** For hosting the backend application. Auto Scaling ensures that the number of EC2 instances scales up or down based on demand.
* **Amazon S3:** For storing static assets such as images, CSS, and JavaScript files.
* **Amazon RDS:** For database management, providing automated backups, software patching, and multi-AZ replication.
* **Amazon CloudFront:** To distribute static and dynamic content globally with low latency.
* **AWS IAM:** For managing access and permissions to AWS resources.
* **AWS Lambda:** For handling serverless functions where appropriate.
* **Amazon Route 53:** For domain name registration and DNS management.

**4. Implementation:**

**1) Setting Up the Infrastructure:**

1. **EC2 Instances:**
   * Launch EC2 instances with appropriate AMIs (Amazon Machine Images).
   * Configure security groups to allow necessary inbound and outbound traffic.
   * Set up Auto Scaling groups and configure scaling policies.
2. **Amazon RDS:**
   * Create an RDS instance with multi-AZ deployment.
   * Set up automated backups and database snapshots.
3. **Amazon S3 and CloudFront:**
   * Create S3 buckets for static assets.
   * Configure CloudFront distributions for S3 buckets to enable caching and improve load times.
4. **AWS IAM:**
   * Define roles and policies to control access to AWS resources.
   * Implement least privilege principles to enhance security.
5. **Route 53:**
   * Register the domain and configure DNS settings.
   * Set up health checks and routing policies.

**2) Application Development:**

1. **Frontend:**
   * Develop a responsive user interface using React.js.
   * Implement authentication using JWT (JSON Web Tokens).
   * Integrate with the backend API.
2. **Backend:**
   * Set up Node.js with Express.js framework.
   * Implement RESTful API endpoints for CRUD operations.
   * Integrate with Amazon RDS for data persistence.
3. **Deployment:**
   * Use CI/CD pipeline (e.g., AWS Code Pipeline) for automated deployment.
   * Containerize the application using Docker (optional) for easier deployment and management.

**5. Evaluation:**

**1) Performance Testing:**

* Conduct load testing using AWS tools like AWS Performance Insights and third-party tools like Apache JMeter.
* Monitor application performance and adjust scaling policies as necessary.

**2) Cost Analysis:**

* Use AWS Cost Explorer and AWS Budgets to monitor and analyse costs.
* Identify cost-saving opportunities, such as reserved instances for EC2, and optimize resource usage.

**3) Security Assessment:**

* Perform security audits and penetration testing.
* Ensure compliance with industry standards and best practices.
* Regularly update IAM policies and review access controls.

**6. Conclusion:**

The project successfully demonstrates the design and implementation of a scalable and cost-effective web application on AWS. By leveraging AWS's robust infrastructure and services, the application achieves high availability, scalability, and cost-efficiency. Continuous monitoring and optimization ensure that the application remains performant and secure.

**7. Future Work:**

* **Enhanced Monitoring:** Implement more granular monitoring using AWS CloudWatch and custom metrics.
* **Serverless Architecture:** Explore more serverless components (e.g., AWS Lambda) for specific parts of the application.
* **Advanced Security Measures:** Incorporate advanced security measures such as AWS WAF (Web Application Firewall) and AWS Shield for DDoS protection.
* **Machine Learning Integration:** Use AWS Sage Maker for integrating machine learning capabilities into the application.

**8. References:**

 AWS Documentation: <https://aws.amazon.com/documentation/>

 React.js Documentation: https://reactjs.org/docs/getting-started.html

 Node.js Documentation: https://nodejs.org/en/docs/

 Amazon RDS Documentation: <https://docs.aws.amazon.com/rds/index.html>

 Amazon DynamoDB Documentation: <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html>

 Amazon CloudFront Documentation: <https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/Introduction.html>

 AWS IAM Documentation: <https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html>