Scenario: Hosting a Web Application on AWS for IT Professionals

Name : Dineshbabu M.R
ACE no - ACE12507

Scenario Overview

Your organization plans to hoost a web application on AWS. The application includes:

1. A frontend built using React.
2. A backend API built with Python (Flask/Django).
3. A MySQL database for storing data.

The architecture should:

Use highly available and scalable AWS services.
Secure the application with best practices.
Ensure minimal downtime.

1. **A frontend built using React:-**

- Host your React application on **Amazon S3** as a static website. This is a cost-effective, scalable solution for serving static files (HTML, CSS, JS).
- Use **Amazon CloudFront** as a content delivery network (CDN) to cache and distribute content globally with low latency.
- **AWS Certificate Manager (ACM)** can be used to manage SSL/TLS certificates for your domain to ensure secure communication between users and your frontend.

2**. A backend API built with Python (Flask/Django).**

- Host the backend API on **Amazon Elastic Beanstalk**. It's a fully managed service that automatically handles the deployment, scaling, and monitoring of the application.
    - **Scaling:** Elastic Beanstalk can scale the app automatically based on demand, ensuring minimal downtime during traffic spikes.

    - **Load Balancing:** Elastic Beanstalk uses an **Elastic Load Balancer (ELB)** to distribute traffic evenly across EC2 instances.
- Optionally, use **Amazon EC2** with an **Auto Scaling Group (ASG)** if you need more control over the infrastructure.

3. **A MySQL database for storing data:-**

Use **Amazon RDS** for MySQL. It's a managed relational database service that provides automatic backups, patching, and scaling.

- **High Availability (HA):** Set up **Multi-AZ deployment** for automatic failover in case of instance failure. This ensures minimal downtime and high availability.
- **Read Replicas:** If you need to handle read-heavy workloads, you can configure **read replicas** to offload read traffic from the primary database instance.

1. Frontend:-

- React app deployed on **Amazon S3** (static website).
- CloudFront as CDN.
- SSL/TLS encryption via ACM.

2. Backend API:

- Flask/Django app hosted on **Elastic Beanstalk** with EC2 instances behind an **Elastic Load Balancer (ELB)**.
- **Auto Scaling** for scalability.

3. Database:

- **Amazon RDS MySQL** with Multi-AZ and Read Replicas.
- Backups and automatic failover enabled.