AWS SCENARIO

1.Scenario: Hosting a Web Application on AWS for IT Professionals

Scenario Overview

Your organization plans to host 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.

AWS Architecture for Hosting a Web Application

1. Architecture Overview

- The application consists
- Frontend: React
- Backend API: Python (Flask/Django)
- Database: MySQL

2. AWS Services Used

- Component: AWS Service Used
- Frontend (React): Amazon S3 + CloudFront
- Backend API (Python Flask/Django): Amazon EC2, AWS Faregate, or AWS Lambda (depending on deployment strategy)
- Database (MySQL): Amazon RDS (MySQL)
- Networking: AWS VPC, Elastic Load Balancer (ELB)

- Security: AWS IAM, AWS WAF, AWS Shield, Security Groups

CI/CD	AWS Code Pipeline, AWS Code Deploy, GitHub Actions
Monitoring	Amazon CloudWatch, AWS X-Ray, AWS CloudTrail

3. Deployment Architecture

- Frontend Deployment
- Host React application on Amazon S3 as a static website.
- Use Amazon CloudFront as a Content Delivery Network (CDN) to serve the React app globally with low latency.
- Enable AWS WAF for security against common web attacks.
- Backend Deployment
- Deploy Flask/Django on Amazon ECS (Fargate) for a serverless, auto-scaling solution. Alternatively, use EC2 instances with Auto Scaling for full control.
- Attach an Application Load Balancer (ALB) to distribute traffic evenly.
- Enable AWS Auto Scaling for handling traffic spikes.
- Database Deployment
- Use Amazon RDS (MySQL) for a managed relational database.
- Enable Multi-AZ Deployment for high availability.
- Enable read replicas for improved performance.
- Security Best Practices
- Use AWS IAM roles and policies to restrict access.
- Deploy AWS WAF to filter malicious traffic.
- Enable TLS encryption using AWS Certificate Manager.
- Implement VPC Security Groups and Network ACLs to restrict database access.
- CI/CD Pipeline
- Use AWS CodePipeline and CodeDeploy for automated deployments.
- Use GitHub Actions or Bitbucket Pipelines for integrating changes.
- Enable automated testing before deployments.
- Monitoring & Logging
- Use Amazon CloudWatch for application logs and performance monitoring.
- Enable AWS X-Ray for request tracing.
- Use AWS CloudTrail for security audits and tracking API calls.

4. High Availability & Scalability

- Feature: AWS Solution
- Load Balancing: Application Load Balancer (ALB)
- Auto Scaling: AWS Auto Scaling Groups
- Fault Tolerance: Multi-AZ RDS, AWS ECS Far gate
- Disaster Recovery: AWS Backup, Cross-Region Replication

5. Cost Optimization

- Use AWS Free Tier where possible.
- Enable auto-scaling to scale resources based on demand.
- Use AWS Compute Savings Plans for cost-effective EC2 and Far gate pricing.