AWS SCENARIO QUESTION

NAME: APARNAA

ACE NO: ACE12502

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

Ans:

To host your web application on AWS with high availability, scalability, and security, you can leverage several AWS services to meet these requirements. Below is an architecture outline and the recommended services and configurations for your web application:

1. Frontend: React Application

- **Amazon S3**: Host the static assets of the React frontend (HTML, CSS, JavaScript) on **Amazon S3** (Simple Storage Service). S3 offers high availability and scalability for serving static web content.
    - **Enable Static Website Hosting**: Configure S3 to host the frontend as a static website.
    - **CloudFront**: Use **Amazon CloudFront** (Content Delivery Network) to cache and distribute content closer to users globally, improving performance and lowering latency.

2. Backend API: Python (Flask/Django)

- **Amazon EC2**: Deploy the backend API (Flask or Django) on **Amazon EC2** instances. Use EC2 for more control over the server environment, scaling, and management.
    - **Auto Scaling**: Set up **Auto Scaling** groups to automatically scale the EC2 instances based on demand. This ensures your API can handle varying traffic loads.
    - **Elastic Load Balancer (ELB)**: Place an **Application Load Balancer (ALB)** in front of your EC2 instances to distribute incoming traffic evenly and ensure high availability.
    - **Security Groups**: Configure **Security Groups** for the EC2 instances to restrict traffic to only the necessary ports and IPs.

Alternatively, if you want a more serverless approach, you can use **AWS Lambda** in combination with **API Gateway** to host your backend without managing servers.

3. Database: MySQL

- **Amazon RDS**: Use **Amazon RDS** (Relational Database Service) to manage the MySQL database. RDS provides automated backups, patching, and scaling capabilities.
    - **Multi-AZ Deployment**: For high availability, enable **Multi-AZ** deployments in RDS. This creates a primary DB instance and a synchronous standby replica in a different Availability Zone.
    - **Read Replicas**: For better performance, especially for read-heavy workloads, you can configure **Read Replicas** in RDS to offload read queries from the primary instance.
    - **VPC Security**: Ensure your RDS instance is inside a **VPC** (Virtual Private Cloud) and use **Security Groups** to control access between the application and database.

4. Networking & Security

- **VPC (Virtual Private Cloud)**: Set up your application in a **VPC** to isolate it from other networks, control routing, and provide secure communication between components.
    - **Subnets**: Create public and private subnets for different parts of the application:

        - **Public Subnet**: For the EC2 instances running your API and the Load Balancer.
        - **Private Subnet**: For the RDS instance, ensuring it's not directly accessible from the internet.
    - **Security Groups**: Use security groups to define rules for controlling inbound and outbound traffic to your EC2 instances and RDS.
    - **IAM Roles**: Apply **IAM roles** to grant the necessary permissions for your EC2 instances and Lambda functions to access other AWS services securely.

5. Monitoring & Logging

- **Amazon CloudWatch**: Use **Amazon CloudWatch** for monitoring the performance and health of your EC2 instances, RDS, and other AWS services. Set up **CloudWatch Alarms** to notify you of any issues.
- **AWS CloudTrail**: Enable **AWS CloudTrail** to track API activity and ensure compliance and security auditing.
- **Elastic Load Balancer Logs**: Enable logging on the ALB to track incoming requests and errors.

6. Backup & Recovery

- **Amazon RDS Snapshots**: Schedule regular **RDS snapshots** to back up your MySQL database.
- **S3 Backup**: Use **S3** to back up application code and important assets.
- **Cross-Region Replication**: For extra redundancy, you can configure **S3 Cross-Region Replication** to replicate static files to another region.

7. Security Best Practices

- **SSL/TLS Encryption**: Use **AWS Certificate Manager (ACM)** to manage and deploy SSL/TLS certificates for securing communications between the client and the server.
    - Configure your **ALB** to terminate SSL and forward traffic to the backend via HTTPS.
- **WAF (Web Application Firewall)**: Implement **AWS WAF** to protect your web application from common web exploits such as SQL injection, cross-site scripting (XSS), and more.

- **IAM Policies and Least Privilege**: Follow the principle of least privilege when setting up IAM roles and policies for access control.
- **Secrets Management**: Use **AWS Secrets Manager** or **AWS Systems Manager Parameter Store** to securely manage sensitive data like database credentials or API keys.

8. CI/CD Pipeline

- **AWS CodePipeline/CodeBuild**: Set up a **CI/CD pipeline** using **AWS CodePipeline** and **CodeBuild** for automated code deployments and testing. This ensures continuous delivery and quick updates to your application.