AWS Cloud Architecture Design for an E■Commerce Web Application

In this project, I designed an AWS Cloud Architecture for an E■commerce web application. My goal was to build something practical that could handle real-world situations such as sudden traffic spikes during sales. While working on it, I wanted to focus on three things: making it scalable, keeping it secure, and ensuring it's easy to monitor and maintain. AWS services like EC2, RDS, S3, and CloudFront gave me the flexibility to create an architecture that balances cost and performance. This project also helped me understand how cloud design decisions affect user experience and business reliability.

1. AWS Architecture Description

I decided to use a three■tier setup: Web, Application, and Database layers. This made it easier to organize and scale different parts of the system independently. Here's how I used each AWS service:

1. Amazon EC2: Used for backend logic and API processing with Auto Scaling across multiple Availability Zones.
2. Elastic Load Balancer (ELB): Distributes user traffic evenly among EC2 instances to prevent overload.
3. Amazon RDS (MySQL): Stores user and order data with managed backups and Multi■AZ for high availability.
4. Amazon S3: Stores static assets like product images and stylesheets to offload the web servers.
5. Amazon CloudFront: Delivers content faster for users in different countries by caching S3 content.
6. Amazon Route 53: Handles domain management and routes users to healthy endpoints.

2. Security Strategy

1. IAM Roles and Policies: Created separate roles for EC2, S3, and administrators with least privilege access. Enabled MFA for admin accounts.
2. Security Groups: Restricted inbound rules so that only HTTP and HTTPS traffic passed through the load balancer. SSH access allowed only from my IP.
3. Encryption: Enabled encryption for RDS and S3 using AWS KMS. All data transfers use HTTPS.
4. Logging: Enabled CloudTrail and AWS Config to track configuration and access changes.

3. Amazon CloudWatch Monitoring Plan

1. EC2: CPU and memory usage, network throughput with alerts if CPU > 80%.
2. RDS: Connection count and query latency to monitor database stress.
3. ELB: Latency and request count for traffic distribution insights.
4. CloudFront: Cache hit ratio for edge performance.
5. S3: Bucket size and request count for storage and cost tracking.

4. Cost Optimization Recommendations

1. Auto Scaling and Spot Instances to reduce unused server costs.
2. S3 Lifecycle Policies to move old data to Glacier storage.

3 Savings Plans for consistent services like RDS.
4 Resource Cleanup using Trusted Advisor and removing unused resources.

5. Disaster Recovery Considerations

1 Automated Backups and Snapshots for RDS.
2 S3 Cross■Region Replication for critical static files.
3 Multi■AZ Deployment for EC2 and RDS for high availability.
4 CloudFormation Templates for quick environment recreation.

Conclusion

This project gave me hands■on understanding of designing and maintaining a cloud■based E■commerce application. By building the architecture, I learned how AWS tools improve scalability, security, and cost management. The experience taught me how design decisions directly impact user satisfaction and system reliability.