PayPlus

Software Architect Hands-On Exam

Objective:
To assess the candidate's ability to design, build, and deploy a secure, scalable chat application on AWS infrastructure using various AWS services. The exam will evaluate the candidate's expertise in system architecture, server-side development, security, database integration, version control, and logging.
Duration: 5 hours

Section 1: System Design on AWS

Task:
Design a scalable and secure chat system architecture using AWS services. The system should meet the following requirements:

Compute: Use EC2 instances for hosting the Node.js application.
Messaging: Implement SQS (Simple Queue Service) to handle message queuing between the chat rooms.
Email Notifications: Use SES (Simple Email Service) to send notifications or alerts.
Database: Utilize RDS (Relational Database Service) for storing chat messages, user data, and room details.
Scalability: Ensure the design supports auto-scaling based on traffic.
Security: Include VPC, Security Groups, and IAM roles/policies to secure the infrastructure.
Logging: Design a logging system to track all requests and responses within the application for monitoring and debugging purposes.

Deliverables:

- A detailed architecture diagram.
- A brief explanation of how each AWS service is used.
- Security considerations and how they are addressed in the design.
- Logging strategy, including the services or tools used for logging.

Section 2: Chat Application Development

Task:
Develop a chat application with the following features:

Private Rooms: Users can create and join private chat rooms.
Message Storage: All messages should be saved in an RDS database as a backup.
Channel Management: Users with appropriate permissions can manage rooms (e.g., add/remove users, delete messages).

API and Webhooks:
Provide an API for managing channels (e.g., creating rooms, adding users).
Implement webhooks to notify external systems when a message is posted in a room.
Logging: Implement a logging system that records all requests and responses, including user actions and API calls.
Admin Management: Build simple admin management tool to manage all chats and users.

Deliverables:

- Node.js application code.
- API documentation (e.g., Swagger or Postman collection).
- SQL scripts for database setup.
- Logging configuration files and sample logs.

Section 3: Security Implementation

Task:
Secure the chat application by implementing the following:

Route Protection: Ensure that only authenticated and authorized users can access specific routes (e.g., channel management).
Input Validation: Prevent SQL injection and XSS attacks by validating user inputs.
Encryption: Store sensitive data (e.g., passwords) securely using encryption.
External API Security: Use API keys or OAuth for external API access.
Logging Security: Ensure that sensitive information is not logged and that logs are protected from unauthorized access.

Deliverables:

- A security report detailing the measures implemented.
- Code snippets demonstrating key security implementations.
- Logs demonstrating secure handling of requests and responses.

Section 4: Deployment, Testing, and Version Control

Task:
Deploy the chat application on AWS and perform the following tasks:

Deployment: Set up the application on EC2 instances and configure the necessary AWS services.
Functionality Testing: Verify that all features (e.g., private rooms, message storage, API) work as expected.

Security Testing: Test the application's security, including route protection, input validation, and encryption.
Version Control: Push the code to a GitHub or Bitbucket repository, ensuring proper commit messages and version history.

Deliverables:

- A live URL where the application can be accessed.
- A brief report on the tests performed and their outcomes.
- A link to the GitHub or Bitbucket repository with the project code.

Evaluation Criteria:
System Design: Completeness, scalability, and security of the design.
Development: Code quality, functionality, adherence to requirements, and effective logging.
Security: Effectiveness of security measures implemented.
Deployment: Successful deployment and functionality in a live environment.
Version Control: Proper use of version control, including meaningful commit messages and organized repository structure.