# Software Architecture for Sentiment Analysis System

Submitted by:
Aditya Srivastava (2201013)
Deepanika Gupta (2201062)
Kartik Saini (2201103)

## Contents

**1 Software Architecture Style** **2**
1.1 I. Choice of Software Architecture: Service-Oriented Architecture (SOA) 2
1.1.1 A. Justification of SOA . . . . . . . . . . . . . . . . . . . . . . . . . 2
1.1.2 B. Why SOA is the Best Choice? . . . . . . . . . . . . . . . . . . . . 2

**2 II. Application Components** **2**

**3 III. Hosting & Deployment Strategy** **3**
3.1 A. Hosting Environment . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
3.2 B. Deployment Strategy . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

**4 IV. User & Component Interaction** **4**

## 1 Software Architecture Style

### 1.1 I. Choice of Software Architecture: Service-Oriented Architecture (SOA)

#### *1.1.1 A. Justification of SOA*

- **Independent Services:** Each core functionality (Sentiment Analysis, Toxicity Detection, Emotion Analysis) is designed as an independent service.
- **Shared Database:** All services store processed results in a common Analysis Database.
- **API-Based Communication:** Services communicate via REST APIs, ensuring modularity.
- **Modular Deployment & Fault Isolation:** If one service fails (e.g., Emotion Analysis), other services (Sentiment Analysis, Toxicity Detection) remain unaffected, ensuring high availability.
- **Scalability & Flexibility:** Services can be independently scaled and updated without affecting others.

#### *1.1.2 B. Why SOA is the Best Choice?*

- **Scalability:** Services can be independently scaled depending on the demand.
- **Maintainability:** Since services are loosely coupled, updating or debugging one service does not affect others.
- **Performance:** API Gateway optimizes request distribution.
- **Flexibility & Technology Stack Independence:** Different services can use different technologies (e.g., Django for APIs, PyTorch/TensorFlow for NLP tasks).
- **Easy Third-Party Integrations:** Can easily integrate external APIs (like Twitter API) without modifying the core system.

## 2 II. Application Components

- Frontend
- API Gateway
- Tweet Fetcher Service
- Sentiment Analysis Service
- Toxicity Analysis Service
- Emotion Analysis Service
- Leaderboard Service
- Admin Management Service

- Analysis Database
- Admin Database

## 3 III. Hosting & Deployment Strategy

### 3.1 A. Hosting Environment

Each component in the **Service-Oriented Architecture (SOA)** system will be hosted on **AWS cloud-based services** to ensure **scalability, maintainability, and performance**.

| **Component** | **Hosting Platform** | **Justification** |
|---|---|---|
| **Frontend (React.js UI)** | AWS S3 (Static Website Hosting) | Highly available and cost-efficient static file hosting. |
| **API Gateway (NGINX)** | AWS EC2 Instance | Routes API requests to backend services and enforces security policies. |
| **Tweet Fetcher Script** | AWS EC2 Instance | Runs continuously to fetch and process tweets from the Twitter API. |
| **Sentiment, Toxicity, Emotion, Leaderboard, and Admin Management Services** | AWS ECS (Elastic Container Service) using Docker | Each service runs in a Docker container, ensuring scalability and isolation. |
| **Message Queue for Tweet Fetcher** | AWS SQS / Kafka | Manages request distribution and prevents API rate limits from Twitter. |
| **Caching System** | Redis (AWS ElastiCache) | Caches fetched tweets to reduce API calls and improve performance. |
| **Database (Analysis & Admin Data)** | AWS RDS (PostgreSQL) or MongoDB (Atlas on AWS) | Stores processed tweet analysis, leaderboard data, and admin credentials. |

### 3.2 B. Deployment Strategy

The deployment strategy outlines the steps and policies for setting up and maintaining the infrastructure, ensuring smooth communication between components, and optimizing performance. Below are the key steps:

**1. Server Provisioning:**

- Launch AWS EC2 instances for the API Gateway and Tweet Fetcher script.
- Set up AWS ECS for the services using Docker containers.
- Configure AWS S3 for static file hosting.
- Deploy AWS RDS for structured data storage.

**2. API Configuration & Communication Setup:**

- Configure API Gateway (NGINX) to route API requests to appropriate backend services.
- Implement security policies, including authentication, and rate limiting.
- Set up AWS SQS/Kafka to handle message queues for asynchronous processing.

**3. Containerization & Service Deployment:**

- Use Docker to package sentiment analysis, toxicity analysis, emotion analysis, and leaderboard management services.
- Deploy containers using AWS ECS with auto-scaling policies.

**4. Database & Caching Configuration:**

- Configure AWS RDS with proper indexing and backup strategies.
- Set up Redis (AWS ElastiCache) to store frequently accessed data for performance optimization.

**5. Monitoring & Logging:**

- Use AWS CloudWatch for real-time monitoring of services.
- Set up logging for API Gateway, ECS services, and database queries.
- Implement alerts for failure detection and performance degradation.

**6. Testing & Optimization:**

- Conduct load testing using AWS Load Balancer to optimize performance.
- Perform integration tests to verify communication between services.
- Regularly update and optimize configurations based on monitoring insights.

## 4 IV. User & Component Interaction

End users access the services through the React.js frontend, which is hosted on AWS S3 for static website hosting. When users interact with the web application, requests are sent to the API Gateway (NGINX), which routes them to appropriate backend services running on AWS ECS (Docker containers). These services handle sentiment analysis, toxicity analysis, emotion analysis, leaderboard management, and admin functionalities, communicating with a database (AWS RDS/MongoDB Atlas) for persistent storage and a caching system (Redis on AWS ElastiCache) for optimized performance. Additionally, a message queue (AWS SQS/Kafka) manages tweet processing to prevent request limits to Tweet Fetcher. The entire infrastructure is secured using HTTPS, authentication mechanisms, and AWS security policies to ensure reliability and scalability.

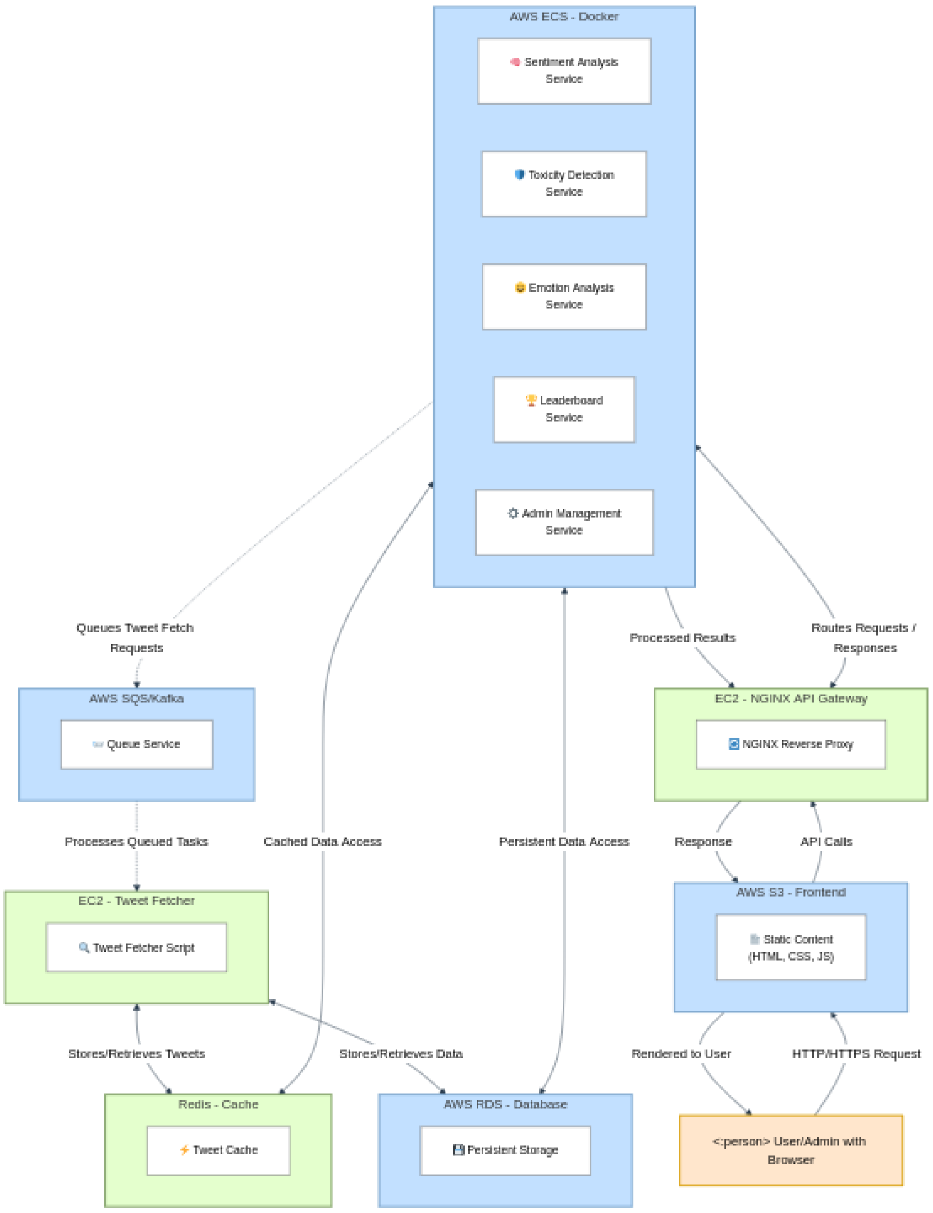

Figure 1: System Architecture: User-Application Interaction & Backend Services