# Twitter-like Scalable Backend System - Technical Documentation (Java Spring Boot)

## 1. System Architecture Overview

### 1.1 Objective

Design a scalable backend system similar to Twitter that can support millions of users with a focus on high availability, low latency, and modular architecture.

### 1.2 High-Level Architecture

Clients (Web, Mobile)  
 ↓  
API Gateway (ALB / Spring Cloud Gateway)  
 ↓  
Spring Boot Microservices (Fargate / EC2):  
 - Auth Service  
 - User Service  
 - Tweet Service  
 - Follow Service  
 - Timeline Service  
 - Notification Service

// Additonal services:

- Search Service (Elasticsearch)

- Analytics Service

- Content Moderation Service

- Rate Limiting Service  
 ↓  
Databases:  
 - PostgreSQL (Amazon RDS)  
 - Cassandra / DynamoDB  
 ↓  
Event Queue:  
 - Apache Kafka (Amazon MSK)  
 ↓  
Feed Workers (EC2 Auto Scaling Group)  
 ↓  
Cache:  
 - Redis (Amazon ElastiCache)  
 ↓  
Media:  
 - Amazon S3 / MinIO

### 1.3 Deployment Strategy

* Stateless services run on **AWS Fargate**
* Heavy background jobs and feed generation run on **EC2 Auto Scaling Group**
* Persistent storage via **RDS**, **ElastiCache**, **MSK**, and **S3**

## 2. Database Design

### 2.1 Relational Schema (PostgreSQL)

**Users Table**

CREATE TABLE users (  
 id BIGSERIAL PRIMARY KEY,  
 username VARCHAR(50) UNIQUE NOT NULL,  
 email VARCHAR(255) UNIQUE NOT NULL,  
 password\_hash TEXT NOT NULL,  
 created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  
);

**Tweets Table**

CREATE TABLE tweets (  
 id BIGSERIAL PRIMARY KEY,  
 user\_id BIGINT REFERENCES users(id),  
 content TEXT NOT NULL,  
 created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  
);

**Follows Table**

CREATE TABLE follows (  
 follower\_id BIGINT REFERENCES users(id),  
 followee\_id BIGINT REFERENCES users(id),  
 created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  
 PRIMARY KEY(follower\_id, followee\_id)  
);

**Likes Table**

CREATE TABLE likes (  
 user\_id BIGINT REFERENCES users(id),  
 tweet\_id BIGINT REFERENCES tweets(id),  
 created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  
 PRIMARY KEY(user\_id, tweet\_id)  
);

### 2.2 NoSQL Schema (Cassandra / DynamoDB)

**User Feed Table**

Table: user\_feed  
Partition Key: user\_id  
Sort Key: timestamp DESC  
Attributes: tweet\_id, tweet\_content, author\_id

## 3. API Design (Spring Boot + REST)

### 3.1 User APIs

| Method | Endpoint | Description |
| --- | --- | --- |
| POST | /auth/register | Register user |
| POST | /auth/login | Authenticate user |
| GET | /users/{id} | Get user profile |
| POST | /follow/{id} | Follow user |
| DELETE | /unfollow/{id} | Unfollow user |

### 3.2 Tweet APIs

| Method | Endpoint | Description |
| --- | --- | --- |
| POST | /tweets | Create tweet |
| GET | /tweets/{id} | Get tweet |
| DELETE | /tweets/{id} | Delete tweet |
| POST | /tweets/{id}/like | Like a tweet |
| DELETE | /tweets/{id}/like | Unlike a tweet |

### 3.3 Timeline APIs

| Method | Endpoint | Description |
| --- | --- | --- |
| GET | /timeline/home | Get home timeline |
| GET | /timeline/user/{id} | Get user’s tweets |

## 4. Core Features Implementation

### 4.1 User Registration and Authentication

* Spring Security + JWT + BCrypt
* Secure storage of password hashes
* Stateless authentication via tokens

### 4.2 Posting Tweets

* Tweets stored in PostgreSQL
* Kafka event emitted to timeline workers
* Worker pushes tweets to followers’ feeds (fan-out)

### 4.3 Feed Generation

* **Hybrid Model**:
  + Fan-out on write for users with <10K followers
  + Fan-out on read for influencers/celebrities
* Timeline data cached in Redis (ElastiCache)

### 4.4 Likes and Follows

* Like and follow events trigger Kafka messages
* Notification service consumes events and inserts into user\_notification table

### 4.5 Notification Service

* Kafka-based pub/sub
* Users notified of likes, mentions, follows
* Push to frontend via WebSocket or polling

### 4.6 Media Upload

* Spring Boot handles metadata
* Upload via pre-signed URL to Amazon S3

## 5. Caching Strategy

| Data | Tool | TTL |
| --- | --- | --- |
| Home Timeline | Redis | 1-5 min |
| Tweets | Redis | 10 min |
| User Profiles | Redis | 30 min |

## 6. Monitoring & Observability

* **AWS CloudWatch**: logs, metrics, alerts
* **AWS X-Ray**: distributed tracing
* **Prometheus + Grafana**: service metrics (optional)

## 7. Deployment & Scaling

### 7.1 Fargate Services

* Auth, Tweet, User, Follow, Notification APIs
* Scales automatically per demand

### 7.2 EC2 Services

* Feed generation workers
* Kafka consumers

### 7.3 Managed Services

* PostgreSQL (RDS)
* Kafka (MSK)
* Redis (ElastiCache)
* S3 (Media)

## 8. Conclusion

This Spring Boot-based backend is modular, scalable, and production-ready for a social media app like Twitter. It leverages AWS Fargate where serverless scaling is ideal, and EC2 where fine-tuned control is needed for high-load background processing.

The system follows modern architecture practices with event-driven design, hybrid feed modeling, and strong observability, ensuring it can support millions of users reliably and efficiently.