**🐦 Deep Dive: Twitter System Design (X Platform)**

**✅ 1. Functional Requirements**

* Post tweets (text, image, video)
* Follow/unfollow users
* Timeline generation (home feed, profile feed)
* Likes, retweets, replies
* Notifications
* Real-time updates (new tweets, likes, etc.)
* Search (hashtags, users, tweets)
* Trending topics

**🔒 Non-Functional Requirements**

* High availability
* Low latency
* Eventual consistency
* Scalability (100M+ DAU)
* Fault tolerance
* Rate limiting
* Spam/abuse detection

**🧱 2. High-Level Architecture**

+-----------+ +-------------+ +---------------+

| Client | <---> | API Gateway| <----> | Auth & Rate |

+-----------+ +-------------+ | Limit Service |

+-------------+

|

v

+--------------------+

| Tweet Service |

+--------------------+

|

v

+----------------------+ +------------------------+

| Timeline Service | <-> | Feed Cache (Redis) |

+----------------------+ +------------------------+

|

v

+--------------------------+

| Social Graph Service |

+--------------------------+

|

v

+------------------+ +-------------------+

| User Service | | Notification Svc |

+------------------+ +-------------------+

|

+-------------------+

| Media Service |

+-------------------+

|

+----------------+

| Object Store |

| (S3 + CDN) |

+----------------+

+-------------------+

| Search Index |

| (ElasticSearch) |

+-------------------+

+-------------------+

| Kafka Streams |

+-------------------+

+-----------+ +-------------+ +---------------+

| Client | <---> | API Gateway| <----> | Auth & Rate |

+-----------+ +-------------+ | Limit Service |

+-------------+

|

v

+--------------------+

| Tweet Service |

+--------------------+

|

v

+----------------------+ +------------------------+

| Timeline Service | <-> | Feed Cache (Redis) |

+----------------------+ +------------------------+

|

v

+--------------------------+

| Social Graph Service |

+--------------------------+

|

v

+------------------+ +-------------------+

| User Service | | Notification Svc |

+------------------+ +-------------------+

| |

+-------------------+ +--------------------+

| Media Service | | Notification Store |

+-------------------+ +--------------------+

| |

+----------------+ |

| Object Store | |

| (S3 + CDN) | |

+----------------+ |

v

+--------------------------+

| Notification Dispatcher |

+--------------------------+

|

+-----------------+------------------+

| |

+---------------+ +---------------+

| Web Push/WS | | Mobile Push |

+---------------+ +---------------+

+-------------------+

| Search Index |

| (ElasticSearch) |

+-------------------+

+-------------------+

| Kafka Streams |

+-------------------+

**🧪 3. Core Services & Components**

**✉️ Tweet Service**

* Stores tweets (text/media metadata)
* Persists to database and pushes to Kafka for fan-out
* Tweets are immutable

**Schema:**

Table: tweets

- tweet\_id (UUID)

- user\_id

- content

- media\_url

- timestamp

- likes, retweets

**DB:** Write-optimized store (Cassandra, HBase)

**🧾 Timeline Service**

* Generates user feed based on follows
* Uses **fan-out-on-write** for celebrities, **fan-out-on-read** for normal users

**Caching:** Redis sorted sets by timestamp per user:

Key: timeline:<user\_id> => Sorted Set of tweet\_ids

**Interview Tip:** Explain **hybrid fan-out strategy** and how to handle stale data in cache.

**🕸️ Social Graph Service**

* Maintains follow/follower relations
* Exposes API: getFollowers(user\_id) / getFollowees(user\_id)

**Storage:**

* SQL for follower table
* Redis cache for hot users

**🔔 Notification Service**

* Handles likes, mentions, retweets, replies, follows
* Event-driven architecture via Kafka
* Generates both real-time and batch notifications
* Sends via WebSocket, Web Push, Mobile Push

**Components:**

* **Notification Listener (Kafka Consumer)**
  + Listens to events: like, retweet, mention, follow
  + Writes notification objects into store
* **Notification Store**
  + Stores user notifications for historical view
  + TTL-based Redis for recent + SQL/NoSQL for full history
* **Notification Dispatcher**
  + Checks user presence (via Redis)
  + If online → push via WebSocket or Web Push
  + If offline → queue for mobile push (FCM/APNs)
* **User Preferences**
  + Respect Do Not Disturb, mute, blocked users
    - Table: notifications
    - - notification\_id (UUID)
    - - user\_id (target)
    - - type (like, follow, mention, etc.)
    - - actor\_id
    - - tweet\_id (optional)
    - - created\_at
    - - is\_read

**📸 Media Service**

* Uploads/serves images/videos (async)
* Uses S3 for storage, CloudFront (CDN) for delivery
* Generates thumbnails + previews

**🔍 Search & Trends**

* Elasticsearch for text search on tweets and users
* Background jobs to index tweets
* Trending topics = real-time analytics on hashtags

**🔐 Auth & Rate Limiter**

* OAuth2 / JWT for sessions
* Redis token bucket for per-user rate limiting

**📩 Kafka Event Bus**

* Decouples services
* Event types: new\_tweet, follow, like, notification
* Used for real-time updates, logging, analytics

**📊 4. Data Modeling**

**Tweets Table**

CREATE TABLE tweets (

tweet\_id UUID,

user\_id BIGINT,

content TEXT,

media\_urls LIST<TEXT>,

created\_at TIMESTAMP,

PRIMARY KEY (user\_id, created\_at)

) WITH CLUSTERING ORDER BY (created\_at DESC);

**Followers Table**

CREATE TABLE followers (

user\_id BIGINT,

follower\_id BIGINT,

PRIMARY KEY (user\_id, follower\_id)

);

**📤 5. Tweet Fan-out Logic**

**Fan-out on Write**

* When a user tweets:
  + Lookup followers from Social Graph
  + Push tweet\_id to each follower’s Redis timeline

**Fan-out on Read**

* For high-follower accounts (celebrities):
  + Fetch tweets on demand
  + Store tweet IDs in a separate celebrity timeline

**Optimization:**

* Use Kafka + batch workers to distribute fan-out
* Use TTL on timelines to save space

**🛠️ 6. Caching Strategy**

|  |  |  |
| --- | --- | --- |
| **Component** | **Cache Type** | **Tech** |
| User Timeline | Sorted Set | Redis |
| Social Graph | Hash/Set | Redis |
| Tweets | LRU Cache | Memcached |
| Trends | In-Memory | Redis |

**🛡️ 7. Security & Abuse Protection**

* Spam Detection with ML classifiers
* Rate limits on tweets, follows
* CAPTCHA for suspicious accounts
* Content moderation (flagging, filters)

**⚖️ 8. Scaling & Sharding**

|  |  |
| --- | --- |
| Component | Scaling Strategy |
| Tweets DB | Shard by user\_id |
| Timeline Cache | Shard Redis by user\_id hash |
| Kafka Topics | Partition by event\_type/user\_id |
| Search Index | ES shards by keyword/hash |
| Media Storage | S3 + CloudFront (global CDN) |

**🧠 9. Interview Insights**

* Push vs Pull Timeline architecture
* Eventual consistency & async fan-out
* Scaling timeline reads & writes
* Redis eviction strategy for hot timelines
* Designing for celebrity vs normal accounts
* Trade-offs in caching vs DB read
* Data freshness guarantees

**🔍 Deep Dive: Trade-offs & Interview Questions**

**🔁 1. Fan-out Strategies**

**Trade-offs:**

| **Strategy** | **Pros** | **Cons** |
| --- | --- | --- |
| Fan-out on write | Fast read, better UX for normal users | Costly for high-follower users, write amplification |
| Fan-out on read | Efficient writes, scalable for celeb accounts | Slower reads, higher latency, heavier DB/cache loads |
| Hybrid | Balance between performance and scalability | Complexity in classification & maintenance |

**Interview Questions:**

* How would you handle a user with 10M followers tweeting?
* How would you detect when to switch from fan-out-on-write to fan-out-on-read?

**🧠 2. Timeline Generation**

**Trade-offs:**

| **Strategy** | **Pros** | **Cons** |
| --- | --- | --- |
| Pre-computed timelines | Low latency, fast UX | Stale data, storage heavy, complexity |
| Real-time generation | Always fresh, simple storage | Higher latency, DB-heavy reads |

**Interview Questions:**

* How would you scale timeline delivery for 100M users?
* What’s your eviction strategy for Redis timelines?

**🧪 3. Caching Strategy**

**Trade-offs:**

| **Cache Policy** | **Use Case** | **Downside** |
| --- | --- | --- |
| TTL | Periodic refresh, time-bound | May serve slightly stale data |
| LRU | Popular tweets/profiles | Rarely accessed data gets evicted fast |
| Write-through | Consistent cache + DB | High write latency |
| Write-around | Avoid unnecessary writes to cache | Stale reads possible |

**Interview Questions:**

* How do you keep user timelines fresh?
* When do you invalidate a cache? How?

**⚙️ 4. Kafka Usage**

**Why Kafka?**

* Asynchronous processing
* Event-driven fan-out, notification, indexing
* Replayability for failure recovery

**Interview Questions:**

* Why Kafka over RabbitMQ?
* How would you ensure message order in Kafka?

**🔐 5. Security / Anti-Abuse**

**Systems:**

* Rate limiting with Redis token bucket
* Real-time spam detection on tweets (Kafka consumer + ML)
* Abuse reporting queues

**Interview Questions:**

* How do you prevent bot spamming?
* How to rate-limit across distributed systems?

**💬 6. Real-Time Updates**

**Tech:**

* WebSockets or long polling for live likes/retweets
* Event fan-out through Kafka
* Push Notification Queue

**Interview Questions:**

* How do you scale WebSocket connections?
* How would you push tweet likes in real time?

**📏 7. Data Modeling Trade-offs**

| **Tweet Table** | **Pros** | **Cons** |
| --- | --- | --- |
| Wide-column (Cassandra) | Fast writes, scale-out | Complex queries, eventual consistency |
| SQL (Postgres) | Rich querying, JOINs | Less write-scalable |