1. Patterns
   1. User Feed/Timeline (real-time)
      1. Redis
   2. User Feed real-time update
      1. Pub-Sub architecture
   3. Store Tweets
      1. Cassandra
   4. Messaging apps (Real time message)
      1. WebSockets
   5. Notification System (Async)
      1. Kafka, RabbitMQ
   6. Search systems (Read)
      1. ElasticSearch
2. Problems & Solutions
   1. Parallel process related data across multiple servers
      1. MapReduce algorithm
3. Metrics
   1. Availability
      1. 99.9% or more
   2. Load
      1. 100 Million DAU
4. Process
   1. High availability
      1. E.g. Social Media apps, Google search
      2. Replication
      3. Failover mechanism
   2. Eventual Consistency
      1. E.g. Likes on post
      2. Cassandra, DynamoDB (Highly districted systems)
      3. Achieved through Kafka??!
   3. Strong consistency
      1. E.g. Banking apps
      2. ACID compliant database (SQL)
   4. Low Latency
      1. E.g .Real time systems (chat apps, online gaming)
      2. WebSocket, UDP, Edge Caching (like CDNs for static content)
   5. Asychronous / Event Driven systems
      1. E.g. : Notification system
      2. Message Queue
5. Learnings
   1. Error codes :
      1. 200 : OK (HTTP request)
      2. 301 : Redirection (URL shortener redirection service)
      3. 401 : Unauthorized
      4. 429 : Too many requests (rate limiter)
   2. Authentications
      1. Token based : JWT, OAuth2
6. Common Services
   1. Media Service :
      1. Handle image/video uploads
      2. E.g. video uploads on FB, insta, twitter.
      3. Post upload processing : Async, serverless
      4. Workflow :
         1. 🎥 A user uploads a video to your service.
         2. ✅ The upload is saved in **S3 (Simple Storage Service).**
         3. 📢 S3 sends an event (like "new video uploaded").
         4. 📨 **EventBridge** catches this event and routes it.
         5. ⚙️ EventBridge triggers a **Lambda worker**. (serverless, for small piece of code, like functions, limits like not more than 15 min, payload not more than 256 KB, etc.)
         6. 🛠️ The Lambda worker encodes the video (e.g., converts to different resolutions like 1080p, 720p).
         7. 📦 The encoded videos are saved back to S3, ready to be served via a **CDN**.
   2. Post service :
      1. Handles post creation (text/image/video)
      2. E.g. When user create a post on social media
      3. Uses : *Media service +* intermediary things *+ Workers*
      4. Workflow:
         1. Media upload : ***Media Service***
         2. Saves **Metadata** (for calculations/decision making) to a database (e.g. **MySQL**)
            1. E.g. {*post\_id, user\_id, timestamp, visibility, media\_s3\_link, caption}*
         3. On creation complete, publishes event to **Kafka**.
            1. → *post\_created(user\_id, post\_id, timestamp)*
   3. Workers :
      1. Consumes **events from Kafka** and does some work.
      2. E.g.
         1. Fan-out workers (social media post creation)
            1. Consumes *post\_created* events from Kafka
            2. For each friend/follower of the posting user:
            3. o **Push** *post\_id* to their feed in Feed cache(**Redis)** and Feed DB (**Cassandra)**
            4. o Or mark for pull-based fetching if the user is a celebrity
            5. Stored as “feed:{user\_id}” → list of post IDs (with TTL to save memory for inactive users)
   4. Feed Generator Service:
      1. Generate user feed from list of posts
      2. Type : Consumer service (uses already assembled list of posts)
      3. Workflow :
         1. Fetch already assembled feed for a user (active user) from **Redis cache.**
         2. If not in cache(passive user), fetch already assembled feed from **Feed store (Cassandra)**.
         3. For each post\_id, it fetches **full post details** from the **Post DB**.
            1. e.g., caption, image URL, user info, like counts, comments
         4. Finally, it assembles, orders (rank based on recency, relevance, interaction, diversification, etc.) and returns the top N posts as feed to a user.