Problem

Design a Test Architecture for a simpler version of Instagram

A user can share photos and can also follow other users. The ‘News Feed’ for each user will consist of top photos of all the people the user follows.

**Requirements**

We’ll focus on the following set of requirements while designing the Instagram:

**Functional Requirements**

1. Users will be located only in India and will be using this application over their mobile phones only. Connections could be 2G, 3G, 4G or Wifi (Public or Private)
2. Users should be able to upload/download/view photos
3. Users can perform searches based on photo titles
4. Users can follow other users
5. The system should be able to generate and display a user’s News Feed consisting of top photos from all the people the user follows.

**Non-functional Requirements**

1. Our service needs to be highly available
2. The acceptable latency of the system is 200ms for News Feed generation
3. Consistency can take a hit (in the interest of availability), if a user doesn’t see a photo for a while; it should be fine
4. The system should be highly reliable; any uploaded photo or video should never be lost

**Scalability Requirements**

1. 500M Total Users
2. 1M Daily Actives
3. 2M Photos uploaded per day

APIs

1. POST /login - Consumes a user name and a password to allow the user to login. Responds with a session cookie that needs to be sent in successive API calls
2. GET /user/:id - Gets user meta-data. Example - User name, Profile Pic URL, Number of followers, Number of Instagram posts, Number of other users followed by the user
3. GET /feed/:user\_id - Gets the News Feed for the user. Each element on the news feed contains - Element ID, list of image URLs for that Feed Element, Upload Time-stamp for each image, Like count on each image, User name for the user who uploaded that image. Feed Elements are sorted by recency and paginated. Refer Step 7 for how the images are downloaded on the API client
4. GET /storage/:user\_id - Gets the temp credentials for the photo storage. The photo storage is sharded by users hence the user\_id parameter in the API call. This API responds with a temporary access token valid for 30 mins and it has PUT access to a specific namespace in a AWS S3 Bucket. Response contains - AWS S3 Bucket URL, Temporary Access Token along with Token Validity Timestamp
5. PUT /photo - The API client can use this API to post upto 10 photos at a time to the S3 Bucket URL (received from API #4). Each Photo needs to be <1 MB in size and the API responds back with the Full S3 URL for each successfully uploaded image. Behind the scenes, this S3 Bucket is mapped to a CDN Distribution allowing low latency access to photos
6. PUT /url/:user\_id - The URLs received from the above step are posted to this URL allowing the server to associate the Photo URLs to a given user to trigger news feed generation as an async workflow.
7. The API client can make direct API calls to the CDN URLs to download the photos for the news feed and profile pic

Architecture Components (In the request path)

1. API Client - Native Android and iOS Apps
2. DNS Service - Route 53
3. Load Balancer - Application Load Balancer with TLS Termination
4. Target Groups (Web Server Tier) - Nginx
5. Application Layer (API Tier) - Docker Containers in HA mode containing API Services hosted in an async framework. API Tier has auto-scaling enabled based on the API requests
6. Photo Storage - AWS S3 Bucket with Cross Region Replication Enabled
7. Photo Serving - AWS CloudFront (CDN)
8. Database Server - Replicated MySQL with Primary and Replica instances
9. Message Bus - Kafka Brokers in HA mode with Zookeeper
10. Async Worker Tier - Docker containers running Python code picking up Feed Generation jobs from a Kafka Queue. Worker Tier has auto-scaling enabled based on the pending job count

1. Different components connectivity -
   1. Api call from one component to another - is the request received?
      1. Is the request received and proper response supplied?
      2. Proper response from one component to another - one user should his photos & not someone else’
      3. Slow network - Gateway in front of components - APIs are timing out - give a proper response
      4. Authentication flow check (session token)
      5. Database Testing - user uploads the photo- it might happen that API is sending proper response but due to some issue database is not storing the photo
2. Instagram Core functionality to be tested
3. Load Testing - Auto scaling can be tested
4. Performance testing - with no .of threads(users) , one can check the response time of different APIs
5. Calculation check - Between components we can have proxy servers
   1. Summation of all request in a timeframe should match the overall request under different status (Pending, Success, Rejected)