Design a Instagram feed

Q : What are the feature that we need to design?
A : Create post (image/video/reel)
  Follow/Unfollow
  See his/her own feed.

Q: How many users can we follow ?
A : Unlimited.

Q: Daily Active Users?
A : 20 Million

Back of the Envelope calculations:
DAU : 20 million

Assumptions :
QPS :
Read --> Feed read. (1 Req)
Write --> POST (1 Req)
User opens/refresh his/her own feed -> 10 times a day.
User post --> 1 picture a day. (1 MB)

write : Read --> 1 : 10

Storage unit : (Assuming for 5 years)

QPS --> Read QPS , Write QPS ?
Storage Unit ?

Functional Requirements :
1. User can create a Post (Img/video/Reel)
2. User can follow another user.
3. User can see his/her own feed.

Non-functional :
1. Availibility is more important then consistency.
2. Scalable (It could handle Billions of users)

Social media interaction :
Facebook, Instagram, LinkedIN

Graph DB --> Neo4j (NO- SQL)

Facebook : Friend --> Bi-directional connection
U1 <--> U2 (Un-directed graphs)

Instagram : Follow --> Uni-directional connection
U1 --> U2 (Directed graph)

LinkedIn : Hybrid Approach (Connection / Follow)

API End Points :

1. Posting something (User --> img/video/reel)
POST /create/post/{userId}
Body : {
    caption,
    content (img/video/reel),
    timestamp
  }

Response : 200 ok.

2. Following a user (Directed Graph)

Idempotent APIs --> Dedupe (double request)
Even if double requst comes, our api does not do anything wrong.

/create/post/{userID} { }
/create/post/{userID} { }
Request signature is identical

API --> double calls --> treat as one Idempotent APIs

Req : headers --> UUID, IdempotentID

2. Follow a user. (Idempotent by default)
u1 --> u2 (follow) --> Graph DB --> Neo4j
PUT : /user/{userId}
{

}

Reponse : 200 ok

3. View our feed.
GET : fetch/feed/{userId}
Response:
{
  2
}

High level overview :

1. user posting something (img/video/reel)

1. used Simple key-value store to store our metadata
2. used Amazon S3 to store content.

2. User following other users.

1. Many to Many relationships.
2. Can be easily model by graph.
3. Thats why we are using graph DB.

3. User viewing his/her own feed

user --> post (to whom you follow)
--> Reverse chronological order.

Flow :
1. Fetch the user's following list.
2. Fetch all the new post from this list. (metadata)
3. Finally, sort the list in reverse chronological order

Instagram --> Read (far more then write)

Data fetching should be fast.

Write --> can be slower but,
Read --> should be smarter.

2 types of mode :
Pull Based
Push Based.

Pull based Model : (Read slow)
Cons : slow
Pros : Better for users that are inactive.

Push based Model : (Read is fast)
user --> post --> pre compute in my follower's feed cache

Pre calculation / pre compute model.

Pros : read is faster.
cons : Bad for inactive user.
2. Hot-key problem / Celebrity problem

Virat Kohli --> Instagram account (250 M followers)

Let's use Push based approach :

1. User posting something
Flow :
1. We will fetch the followers list of the current user.
2. then we will add the postId, to the feed cache.
3. We will store a list of pair of <userId, postId>.

2. When user see his/her own feed.

Everything is pre-computed here.

Problems ?
1 . Both approach has some disadvantages.

Hybrid approach

Every user has a boolean variable --> isPreComputed.
Virat Kohli --> 250 M

follower <= 1 Million (less followers --> push based approach)

People with more followers --> pull based approach
People with less followers --> push based approach

2. We can limit the number
of post appear in feed cache.
200 people data. (90 % people)
fetch next 200 --> Pagination

i have many followers (500K followers)

Reason to use Kafka ?
1. Cache/Db throughput is less.
2. To make our Api call Async2