Design a scalable service like Twitter

Step 1: Requirement Gathering

1. Who can read/post a tweet?
2. What does a tweet contain: text, photo, video?
3. Can a user follow another user?
4. Can a user like a tweet?
5. What is in the user feed?
6. Is user feed a list of tweets in chronological order?
7. Can a user search tweets?
8. Are we designing client/server interaction? Backend architecture or both?
9. How many users are there?
10. How man daily active users are there?

Step 2 System Interface Definition

Define what APIs are expected from the system.

Step 3 Back-of-the-envelope Capacity Estimation

1. Scale of the system: number of new tweets per day, number of tweet views
2. Storage needed: photos and videos in the tweets?
3. Network bandwidth: how to manage traffic and balance load between servers?

This would help with scaling, partitioning, load balancing and caching.

Step 4 Define the Data Model

Identify various entities of the system, and how will these entities interact with each other.

Different aspect of data management like storage, transfer and encryption.

1. User: userId, name, email, dob, lastLogin
2. Tweet: tweetId, content, tweetLocation, numberOfLikes, timeStamp
3. UserFollows: userId1, userId2
4. FavouriteTweets: userId, tweetId, timeStamp
5. What database system is required?

This would help with partitioning and management.

Step 5 High-level Design

1. Multiple application servers to serve all the read/write requires with load balancer in front of them for traffic distribution.
2. Assuming more read traffic than write traffic, should separate servers from handling reads vs writes.
3. An efficient database to store all the tweets and support a huge number of reads.

4. A distributed file storage system for storing photos (videos)
5. A search index and infrastructure to enable searching of tweets

Step 6 Detailed Design for Selected Components

1. How to partition data to distribute it to multiple databases?
2. What issue would it cause if we store all the data of a user on the same database?
3. How to handle high-traffic users, i.e., celebrities?
4. How to store data to optimise scanning latest tweets?
5. How much and at which layer should cache be used to speed things up?
6. What components need better load balancing?

Step 7 Identify and Resolve Bottlenecks

1. Single point of failure in the system? How to mitigate it?
2. Do we have enough replicas of the data to ensure if we lose a few servers we could still serve the users?
3. Do we have enough copies of difference services running, such that a few failures will not cause total system down?
4. How to monitor the performance of the system?

Reference: [hackernoon.com]