Structuring a Scalable Systems Design QA

Example: Design News Feed/Twitter

Twitter: Example Functional Requirements

- User Account Management
- User should be able to post tweets
  - 140 characters text
  - Media
  - Hashtags
- User should be able to follow/unfollow other users
- Users should be able to visualize feed of tweets
- Users should be able to like, comment on tweets
- Trending Hashtags
- Analytics
- Search

Twitter: Example Design Constraints

- Number of users: 400 million
- Number of tweets generated per second: 6000 /sec
- Number of feed views per second: 300,000 /sec

Twitter: Example Microservices

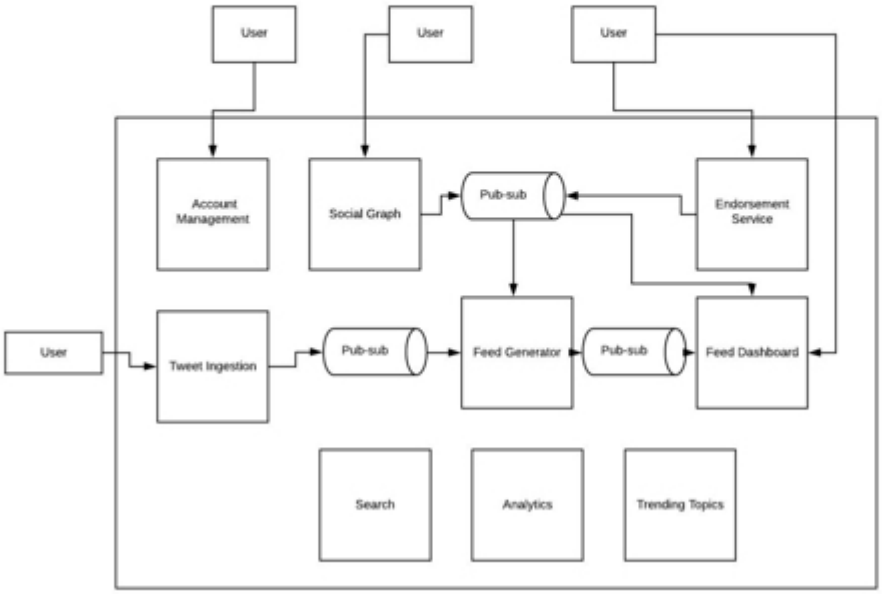
- Account management service
- Tweet Ingestion service
- Social graph
- Feed Generator
- Feed Dashboard
- Endorsement service
- Analytics
- Search
- Trending Hashtags

Clearly, a breadth oriented problem

Microservices to focus for Twitter

- Tweet Ingestion service
- Social Graph
- Feed Generator
- Feed Dashboard
- Endorsement service

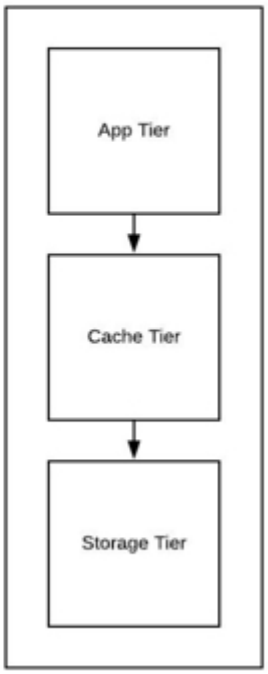
Twitter: Logical Architecture Block Diagram



Tweet Ingestion

Tweet Generation Service: Logic

- Tiers: App server tier, Cache or In-memory tier, Storage tier
- Data Model: K-V pair
  - Text Data:
    - K: <user id/tweet id>: V: text + other properties
  - Media Data
    - K: <user id/tweet id>: V: bytestream
- APIs
  - REST APIs:
  - insertTweetText(user id, text content) for text data (K-V API)
  - insertTweetMediaChunk(user id, bytestream, offset, length) for media data
- How to store in Cache tier -
  - Hashmap for Text Data
- How to store in Storage tier
  - as a row oriented K-V store for Text data
  - Filesystem for Media Data

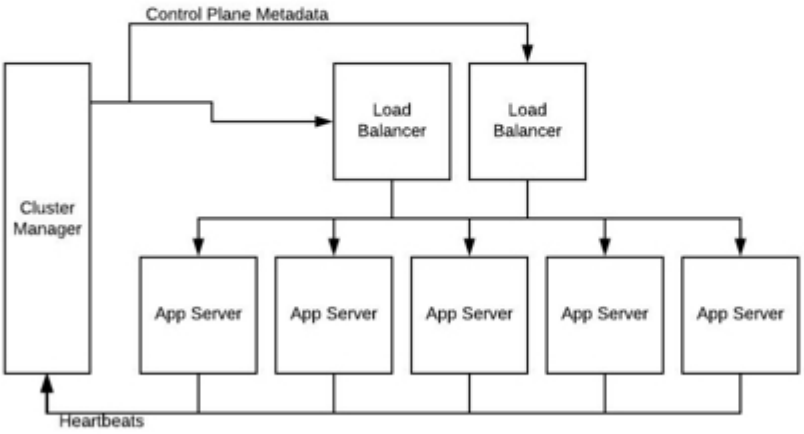


Tweet Generation Microservice: Need to Scale?

- Need to scale for storage ? yes
  - Amount of storage in cache and storage tiers: (Number of tweets/sec)*size of K-V pair per tweet*(plan for 2-3 years)
- Need to scale for throughput ? somewhat
  - Number of update API calls per second: 6000 per sec
- Need to scale for API parallelization ?
  - For text data, no
  - For media data, can be parallelized to improve tweet media ingestion latency
- Availability ? yes
- Geo-location based distribution ? no

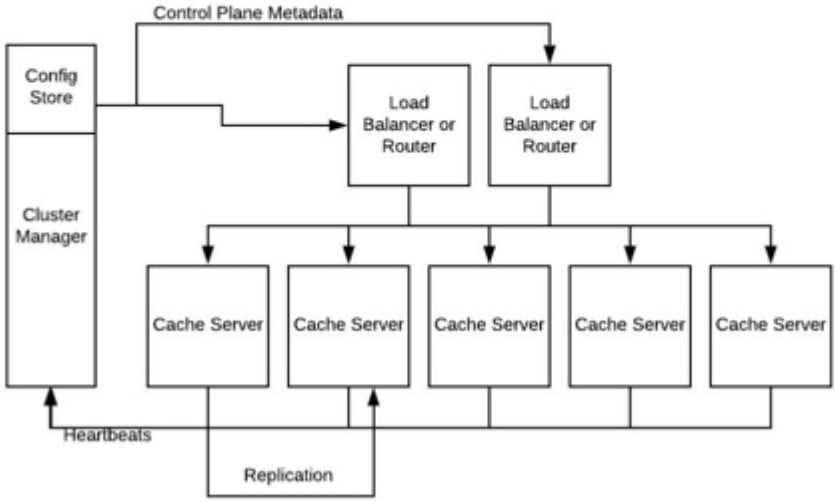
Tweet Generation Microservice: Dist. Architecture

Architectural layout for app tier



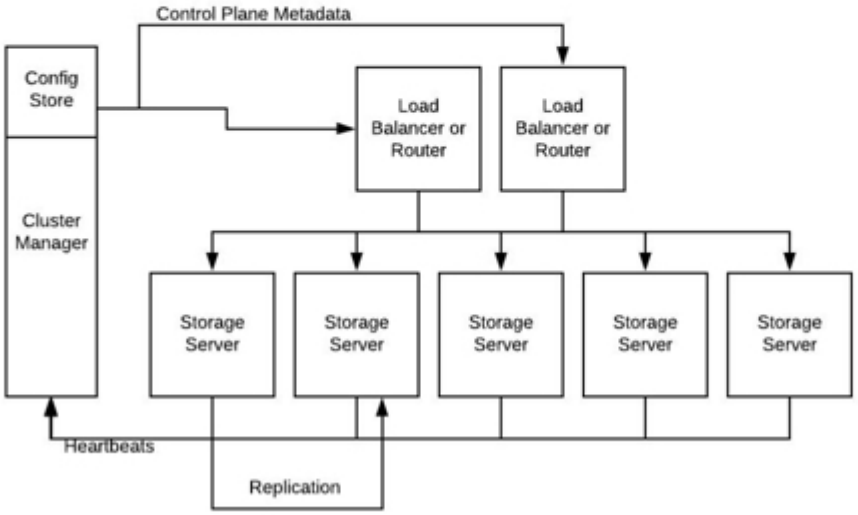
Tweet Generation Microservice: Dist. Architecture

Architectural layout for cache tier



Tweet Generation Microservice: Dist. Architecture

Architectural layout for storage tier



Tweet Generation Microservice: Dist. Architecture

- Sharding
  - Horizontal sharding for Text Data
  - Horizontal + Vertical sharding for Media Data
    - Media data stored as chunks per tweet
    - Sharding key: [Key of tweet][0-1MB] -> shard X-> servers
- Placement of shards in servers
  - Consistent hashing
- Replication
  - Yes for availability as well as for throughput
- CP or AP?
  - CP:
  - Either master-slave or quorum

Social Graph

Social Graph Service: Logic

- Tiers: App server tier, Cache or In-memory tier, Storage tier
- Data Model: K-V pair
  - Follower Graph
    - K: user id: V: list of {followers, relevance}
  - Followee Graph
    - K: user id: V: list of {followees, relevance}
- APIs
  - insertFollowerEdge
  - insertFolloweeEdge
- How to store in in-memory tier:
  - Adjacency list
- How to store in storage tier:
  - Store as individual vertices and edges

Social Graph Service: Need to Scale?

- Need to scale for storage ? yes
  - Graph of scale of 400 million members
- Need to scale for throughput ? may be, not very high
  - Number of edges created per second: not very high
- Need to scale for API parallelization ? no
  - K-V workloads, constant latency
- Availability ? yes
- Geo-location based distribution ? no

Social Graph Service: Dist. Architecture

Architectural layout for every tier same as previous microservice

Social Graph Service: Dist. Architecture

- Sharding
  - Horizontal sharding
- Placement of shards in servers
  - Consistent hashing
- Replication
  - Yes for availability as well as for throughput
- CP or AP?
  - CP: master slave or quorum

Feed Generator

Feed Generator Service: Logic

- Tiers: App server tier, Shares Social Graph cache
- APIs
  - consumeTweetsFromTweetIngestion
  - consumeEndorsements
  - generateFeed(User Id, List of tweets with endorsements)
- Algorithm in APIs
  - Consume tweets from Tweet Ingestion service
  - Use social graph to curate feeds per follower
  - Consume endorsements per tweet and curate feeds per follower
  - Publish curated feeds per follower

Feed Generator Service: Need for Scale

- Need to scale for storage ? N/A
- Need to scale for throughput ? no
- Need to scale for API parallelization ? yes
  - Curating APIs have to be fast
- Availability ? yes
- Geo-location based distribution ? no

Feed Generator Service: Dist. Architecture

Architectural layout for every tier same as previous microservices

Step 4c for Feed Generator Service

- Sharding
  - Scatter gather processing
- Placement of shards in servers
  - Consistent hashing
- Replication
  - Yes for availability as well as for throughput
- CP or AP? N/A

Feed Dashboard

Feed Dashboard Service: Logic

- Tiers: App server tier, In-memory Tier, Storage tier (just for recovery purposes)
- Data Model: K-V pair
  - K: <User id>: V: *list of tweet text data ordered by timestamp and relevance*
- How to store in In-memory tier -
  - Hashmap and ringbuffer
- APIs
  - REST API: read(User id)
  - comsumeTweetsFromPubsub(User Id, list of tweets)
    - Consume tweets from feed generator
- Algorithm in APIs
  - Simple K-V workload
  - Not much meat

Feed Dashboard Service: Need to Scale?

- Need to scale for storage ? yes
  - Number of users: 400 million
- Need to scale for throughput ? yes
  - Number of update API calls per second: 300,000 per sec
- Need to scale for API parallelization ? no
  - APIs themselves are constant latency, so no need to parallelize
- Availability ? yes
- Geo-location based distribution ? yes

Feed Dashboard Service: Dist. Architecture

Architectural layout for every tier same as previous microservices

Feed Dashboard Service: Dist. Architecture

- Sharding
  - Horizontal sharding by user id
- Placement of shards in servers
  - Consistent hashing
- Replication
  - Yes for availability as well as for throughput
- CP or AP? No hard and fast requirement

Endorsement

Endorsement Service: Logic

- Tiers: App server tier, In-memory Tier, Storage tier
- Data Model: K-V pair
  - K: Endorsement id: V: User, Tweet, Type, Comment
- APIs
  - REST API: createNewEndorsement(K)
- How to store in In-memory tier -
  - Hashmap
- How to store in storage
  - Row oriented storage
- APIs
  - REST API: read(K)
- Algorithm in APIs
  - Simple K-V workload
  - Not much meat

Endorsement Service: Need to Scale?

- Need to scale for storage ? yes
  - Need to estimate rate of endorsements
- Need to scale for throughput ? may be
  - Need to estimate rate of endorsements
- Need to scale for API parallelization ? no
  - APIs themselves are constant latency, so no need to parallelize
- Availability ? yes
- Geo-location based distribution ? no

Endorsement Service: Dist. Architecture

Architectural layout for every tier same as previous microservices

Endorsement Service: Dist. Architecture

- Sharding
  - Horizontal sharding
- Placement of shards in servers
  - Consistent hashing
- Replication
  - Yes for availability as well as for throughput
- CP or AP? CP