NoSQL and the architecture of Twitter "from a scalability point of view"

Twitter used Redis instance in the architecture to keep all the tweets for a particular user, this instance contains his home timeline.

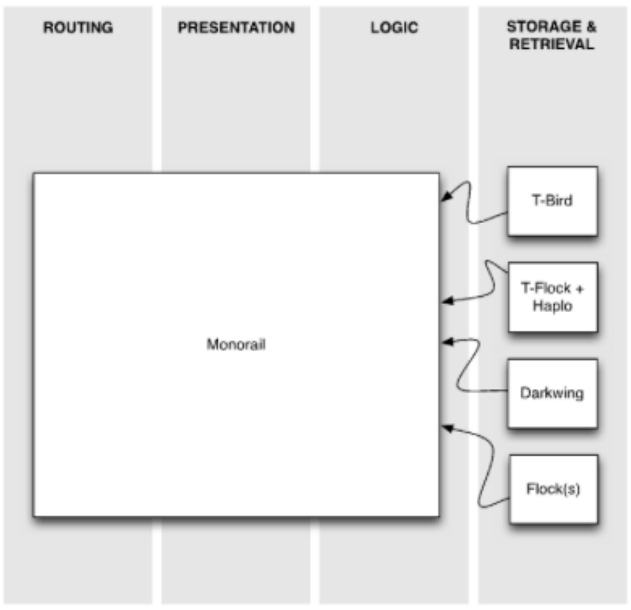
Redis means REmote DIctionary Server, it is open-source and it supports different kinds of abstract data structures, such as strings, lists, maps, sets, sorted sets, hyperloglogs, bitmaps and spatial indexes. Redis has built-in replication, transactions and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster
[https://en.wikipedia.org/wiki/Redis]

In twitter, Redis instance uses a native list structure, store a data object contain Tweet ID, User ID and Bits. [1]

While Cache is ~3% of Twitter infrastructure, it is critical. Redis protects Twitter backing stores from heavy read traffic and allows for storing objects which have heavy hydration costs.[2]

Also, Redis is cluster-oriented, Redis Clusters are series of caches throw the data center [1], Twitter used Redis to create a write based system "Fanout approach: do a lot of processing when tweets arrive to figure out where tweets should go", so when tweet come in, they start re-caching every follower home timeline, and when some user of the followers opens the web, they need just to order one operation (Find that home timeline) and reply it to the user.
Using Redis Twitter optimize the speed of fanout. [1] [3]

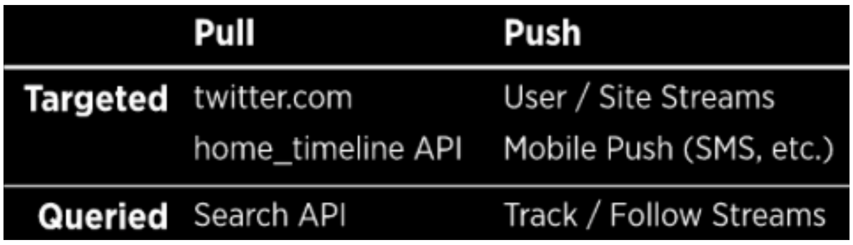
Twitter used to be a standard 4-layer Architecture with one big Monorail app that used to do everything [1]:



Then twitter break this application up to achieve some goals [1]:

- Evolve from being solely a web stack
- Isolate responsibilities and concerns
- Site speed and reliability
- Developer innovation speed

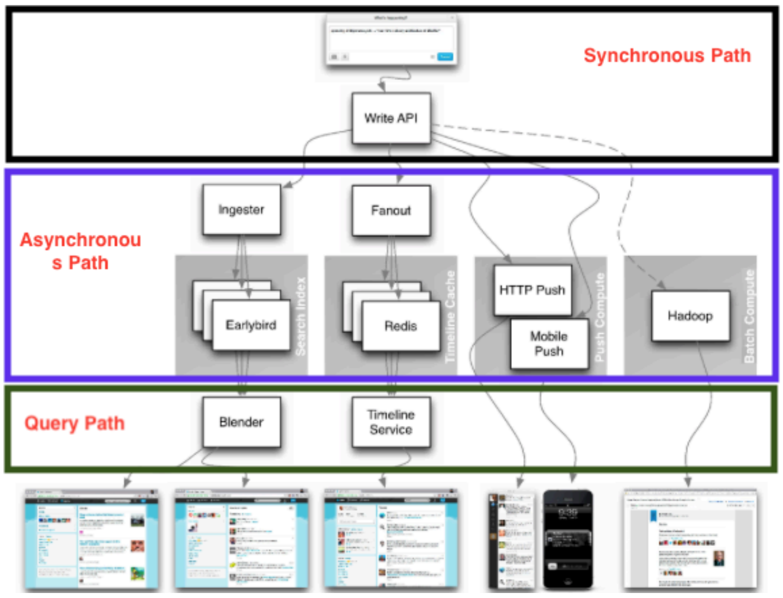
And they separated timeline services so now no change will affect all services, there are 4 types of timelines in twitter "addressed in the image below [1]".



	Pull	Push
Targeted	twitter.com home_timeline API	User / Site Streams Mobile Push (SMS, etc.)
Queried	Search API	Track / Follow Streams

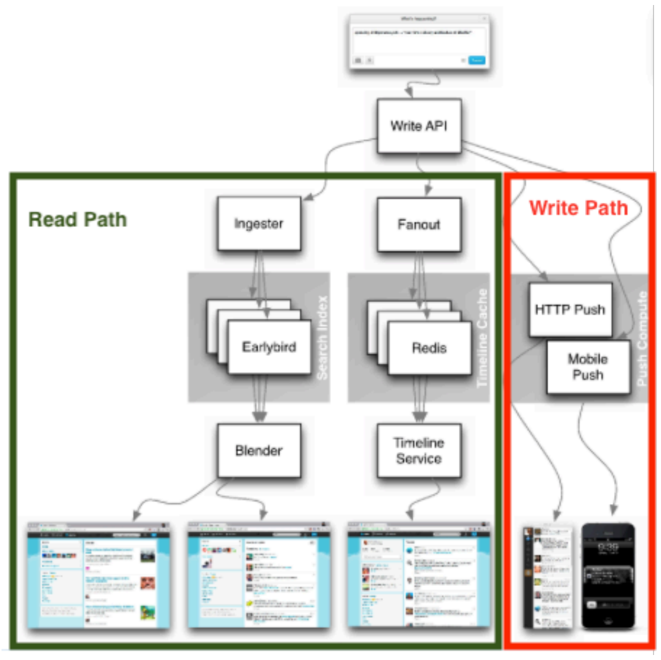
There are few ways to address how twitter broke up their architecture, First [1]:

- Synchronous Path "return to the user as quickly as possible"
- Asynchronous Path "start fanout and Ingester processes"
- Query Path "How you can ask for the data from twitter"



Second [1]:

- Read Path "what happen when read request issue"
- Write Path "tweet need to be written in the structure"



Twitter also use Gizzard (Storage Mechanism) "an open source sharding framework to create custom fault-tolerant, distributed databases, it manages partitioning data across arbitrary backend data stores, that allows it to be accessed efficiently" [https://en.wikipedia.org/wiki/Gizzard_(Scala_framework)]

Twitter front all of Redis at twitter with the same Gizzard library. So, they rely in Gizzard for partitioning and replication.

In Twitter, there are a Redis instances for each particular user, they keep in it all tweets for this user, they replicate these instances because if they lost one machine they don't want to lose the full timeline.

Redis instances are highly replicated through the data center, so every single time they do insertion in timeline Redis instance, they actually insert in three other ones, they replicate the entire write path to secondary data center, they only replicate the tweet stream and then they re-run logic on the other data centers. [1][3]

From the other hand, twitter logically partition their caches by users, Tweets, timelines, etc. and in general, every cache cluster is tuned for a particular need. Based on the type of the cluster, they handle between 10M to 50M QPS, and run between hundreds to thousands of instances. [2]

References

[1]:
Raffi Krikorian, Real-Time Delivery Architecture at Twitter
“https://www.infoq.com/presentations/Real-Time-Delivery-Twitter”

[2]:
Mazdak Hashemi, The Infrastructure Behind Twitter: Scale
“https://blog.twitter.com/engineering/en_us/topics/infrastructure/2017/the-infrastructure-behind-twitter-scale.html”

[3]:
The Architecture Twitter Uses to Deal With 150M Active Users, 300K QPS, A 22 MB/S Firehose, And Send Tweets in Under 5 Seconds
“http://highscalability.com/blog/2013/7/8/the-architecture-twitter-uses-to-deal-with-150m-active-users.html”

[https://en.wikipedia.org/wiki/Redis]

[https://en.wikipedia.org/wiki/Gizzard_(Scala_framework)]