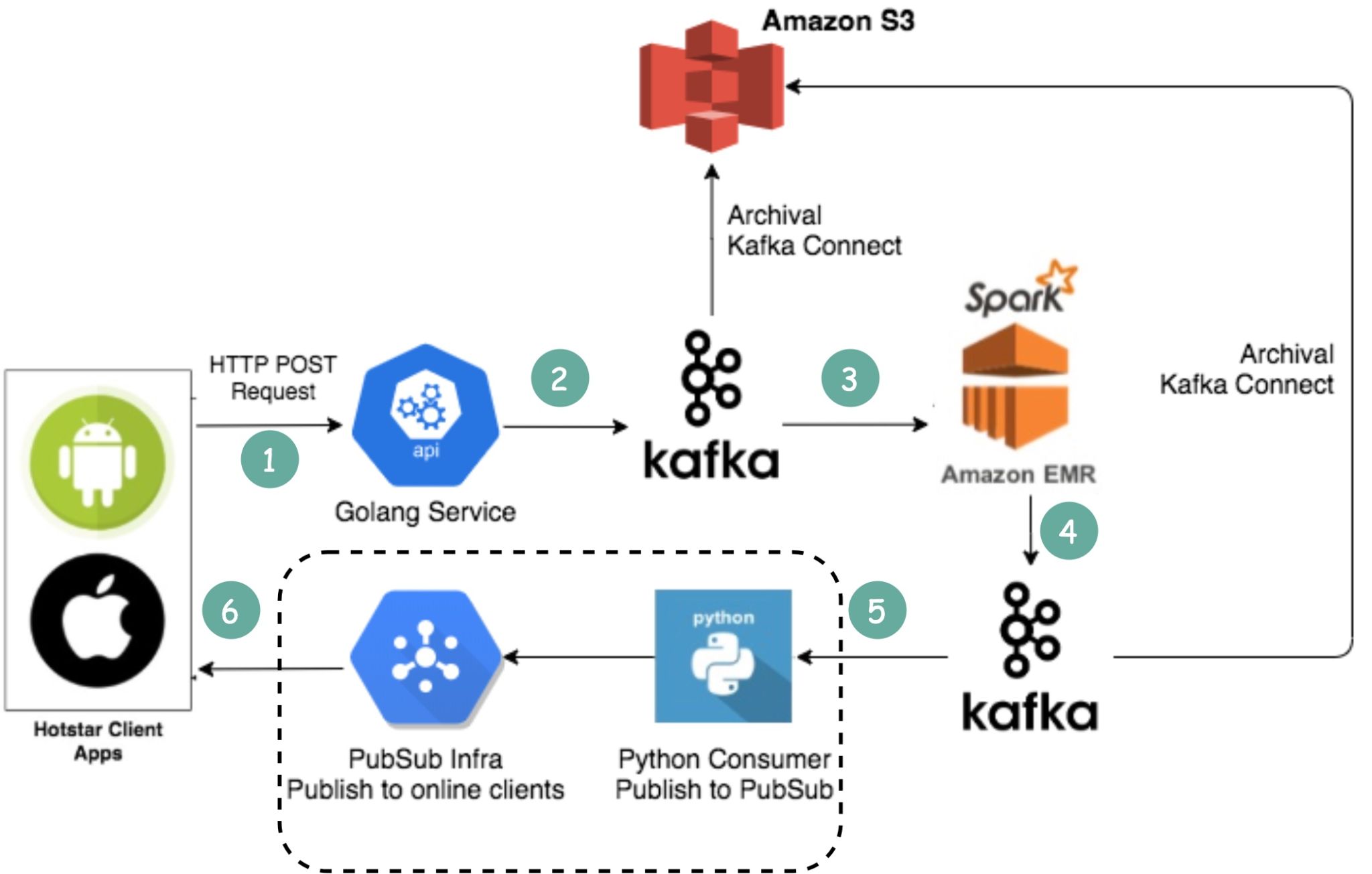
How does Disney Hotstar capture 5 Billion Emojis during a tournament?

  
  
[Dedeepya Bonthu](https://www.linkedin.com/in/ACoAABlYigQBarNH5vgqqtzMCr9kJEyQq0k0VKw) wrote an excellent engineering blog that captures this nicely. Here is my understanding of how the system works.  
  
1. Clients send emojis through standard HTTP requests. You can think of Golang Service as a typical Web Server. Golang is chosen because it supports concurrency well. Threads in GoLang are lightweight.   
  
2. Since the write volume is very high, Kafka (message queue) is used as a buffer.  
  
3. Emoji data are aggregated by a streaming processing service called Spark. It aggregates data every 2 seconds, which is configurable. There is a trade-off to be made based on the interval. A shorter interval means emojis are delivered to other clients faster but it also means more computing resources are needed.  
  
4. Aggregated data is written to another Kafka.   
  
5. The PubSub consumers pull aggregated emoji data from Kafka.   
  
6. Emojis are delivered to other clients in real-time through the PubSub infrastructure.   
  
The PubSub infrastructure is interesting. Hotstar considered the following protocols: Socketio, NATS, MQTT, and gRPC, and settled with MQTT. For those who are interested in the tradeoff discussion, see [2].  
  
A similar design is adopted by LinkedIn which streams a million likes/sec [3].  
  
Over to you: What are some of the off-the-shelf Pub-Sub services available? Is there anything you would do differently in this design?  
  
Sources:   
[1] Capturing A Billion Emo(j)i-ons: <https://lnkd.in/e24qZK2s>  
[2] Building Pubsub for 50M concurrent socket connections: <https://lnkd.in/eKHqFeef>  
[3] Streaming a Million Likes/Second: Real-Time Interactions on Live Video: <https://lnkd.in/eUthHjv4>  
  
—  
If you enjoy these posts, you might like my ByteByteGo System Design newsletter. Subscribe here: <https://bit.ly/3FEGliw>