How We Built Filmgrain, Part 1 of 2

By Andre on June 25, 2013

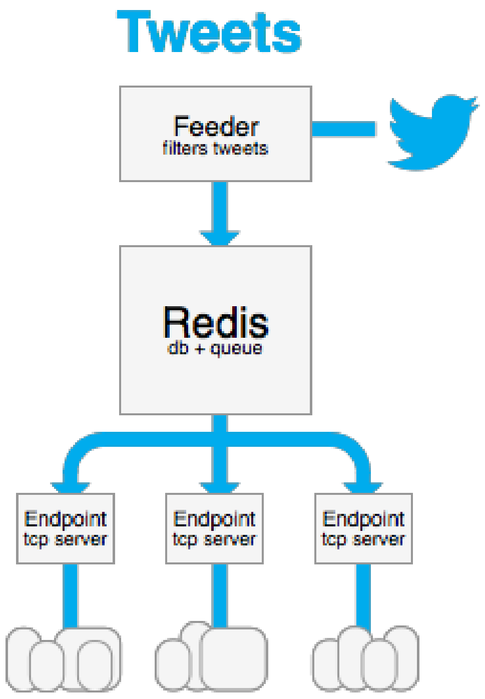
Filmgrain enables users to see what movies are popular on Twitter and watch what people are tweeting about those movies in real time. We had to do some out-of-the-box thinking on our backend and the way apps communicate with it in order to make this functionality possible.

The Backend Architecture

Modularity has many benefits that are pretty obvious to most programmers. Unfortunately, many backends aren't built with modularity in mind from the very beginning and so end up becoming monolithic monstrosities.

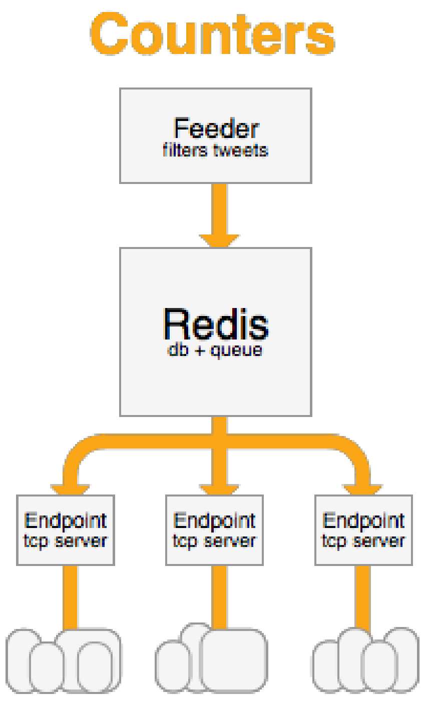
To build our backend, we broke up self contained components into their own processes and use Redis [1] to manage all of the communication. This means each component can run on its own machine and be brought up and down without disrupting the entire system. This also provides fault-tolerance and is scale-friendly.

Redis [2] is great. We use it as an in-memory cache, a database, and a message queue. Since we get all of these critical functionalities from one piece of software, we've been able to keep the number of different technologies we use down to a minimum.



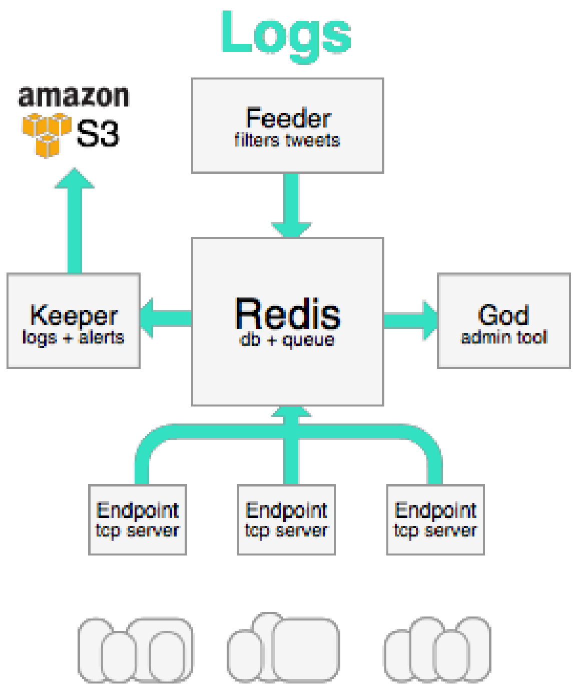
The first independent component of the Filmgrain backend that we built is what we call the Feeder. The Feeder listens on Twitter's Streaming API and publishes filtered tweets into Redis for the other components to consume.

The second component we built is the endpoints that mobile clients connect to. These endpoints are subscribed to the tweet messages in Redis and so receive new tweets in real time. This communication between the Feeder and the endpoints is done using Redis's publish/subscribe [3] functionality. Finally, the endpoints push these new tweets to the mobile clients as fast as they are received.



Filmgrain ranks movies by the number of times it has been tweeted about in the past hour. We use Redis to keep track of these counts. Every time we see a new tweet about a movie, that movie's key is incremented. These counts are synchronized with the mobile clients every few seconds.

To make sure tweets are actually about the movie, we use heuristics including keyword matching. It's not perfect yet, but it will improve.



The third component we built is called the Keeper. The Keeper is responsible for watching the logs.

Our logging system works through Redis's publish/subscribe functionality, just like our Tweet system. The Feeder and the endpoints all publish their logs into Redis where Keeper listens and is responsible for three things. First, it uploads the logs regularly to S3. Second, it makes sure that the logs are actually flowing and alerts us via SMS if they aren't. This could mean Redis is having issues, Twitter's Streaming API is down, or we've broken something. Finally, any time Keeper sees an exception in the logs, it emails it to us.

It’s All About Redis

We're relying heavily on Redis to make Filmgrain possible. By letting Redis stand between each of the independent components that make up our backend, not only can any one of them can go down without effecting the others' operation, but we can bring up more and more endpoints as we need to without restarting the entire system. And, since each endpoint is only a single additional Redis client that then supports hundreds or thousands of concurrent mobile clients, we have plenty of room to scale.

Also, while Redis is currently a single point of failure, we’re content to know that we have only *one* single point of failure. (Many backends wouldn’t survive any number of different technologies that they use failing.) It will also be relatively simple for us to reduce this risk going forward.

Managing It All

Along with focusing on modularity, we decided to make monitoring and deployment comfortable from the start. We felt this was important because automating things not only pays huge dividends going forward, if you don't do it from the start, it only gets more difficult to add later. To this end, we wrote an administration component we called God. God can do everything--it can bring up and down machines, it can deploy our code, it can monitor the status of all of the processes, and it can tap into the log stream. Having God from the start means that as we continue to build Filmgrain, God will be a part of every decision we make. This guarantees we will always have an up to date automated way of administrating our backend.

What’s In Part 2?

Now that we’ve covered how we built our realtime backend, part 2 will discuss building our TCP API in order to bring the benefits of a realtime infrastructure down to the mobile clients. We’ll have part 2 up next week.

Discuss this post on Hacker News [4]

Discuss this post on reddit [5]

1. http://redis.io/
2. http://redis.io/
3. http://redis.io/topics/pubsub
4. https://news.ycombinator.com/item?id=5940433
5. http://www.reddit.com/r/programming/comments/1h1mn9/how_we_built_our_realtime_app_filmgrain_part_1/