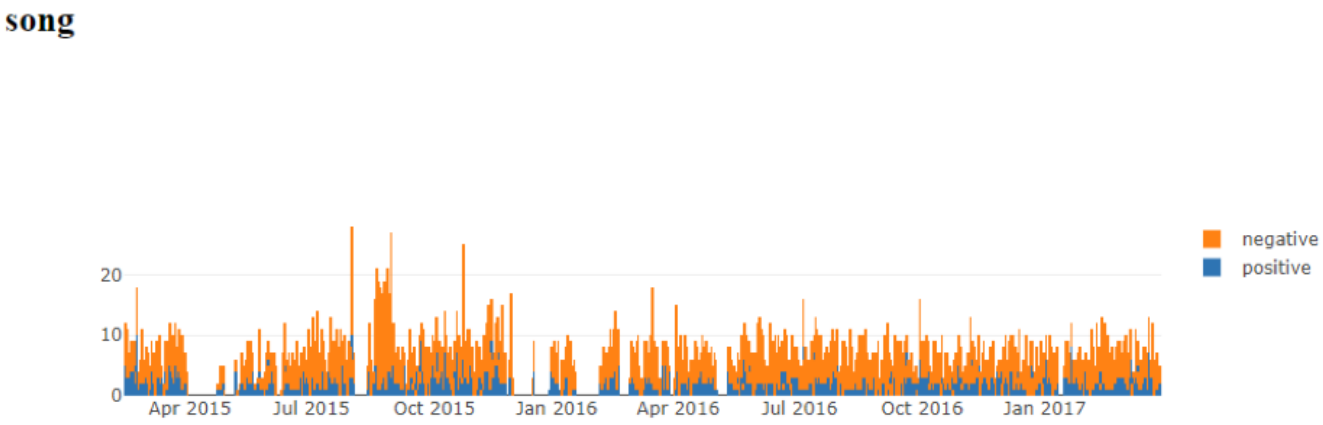
Keeping in training for Cloud Computing

12 February 2018 | **Software Engineering** | **Jaksa Vuckovic**, **Steve Freeman**
Reading time: 3 minutes

At Zuhlke, as part of our training commitment, our staff run “Topic Teams” to learn about tools and techniques that we think are important. The UK *Distributed Systems* team recently spent a few days experimenting with combinations of data, containers, and cloud services to refine their understanding of the trade offs between different cloud-based approaches.

Using the code and data from a previous pro-bono project, International Alert, our task was to generate a sentiment timeline for tweets: a time series showing the amount of positive and negative tweets per day for a given keyword, from a collection of about 250 million messages, for example:



This breaks down into two sub-problems: preparing sentiment analysis on all the tweets beforehand; and, supporting interactive queries by calculating daily averages and displaying the result.

The first was to use a naive master-worker pattern via REST. The master coordinates which partitions of the collection of tweets each worker will read, analyse, and record. We wrote services which we deployed using a combination of Docker and Kubernetes on Google Container Engine. This was straightforward to implement although, obviously, the design is not resilient to failure. It’s not a bad place to start when porting an existing system as we were.

The next step was to attempt to replace REST communication with middleware, in this case trying each of Gridgain, Hazelcast, Akka, and JGroups. It turns out that each of these need special plugins to Kubernetes for cluster discovery, which proved to be too large a task for the time we had available. We've been caught before by subtle differences between running in a container and running in a host, particularly around networking, so this was not a surprise. It's what you'd expect from middleware that creates resilience by running different instances on different boxes. Again, it's a reasonable approach when porting from a previous installation and trying to understand the issues.

Finally, we took the obvious path which was to use the service from the cloud provider, in this case Google Dataflow. This covers all of the system issues such as distribution, fault tolerance, and consistency. As always, there were quirks to navigate but in this situation it was the best solution because it let us focus on the problem rather the infrastructure.

For us, this exercise reinforced the lesson that "going Cloud" is not just a matter of replacing your data centre with someone else's, although that might be worth doing anyway. Cloud is also about learning how to repartition your systems to use the most appropriate infrastructure for each component. Sometimes we need to create a host, sometimes a container will do, and sometimes just functions. For example, we would expect that the higher up the functionality stack we go, the more we can focus our time on *our* functionality, but also the more we lock into one cloud vendor with their associated costs. The great thing about the Cloud is these trade-offs are visible so we can make informed decisions and, if circumstances or our understanding changes, we can adjust that balance without having to reconfigure our datacentre.