written by Tom McAdam on 18 December 2012 in engineering

small is beautiful

Over the last few months, we've been very busy adding a host of new features to our products, and making some systems changes to support them. As part of that, we've split up applications into smaller, more manageable pieces. If you've got your buzzword bingo scorecard in front of you, I'll give you one for free. This all sounds a lot like service-oriented architecture, doesn't it? Don't worry, we're not doing anything crazy with the likes of SOAP: we're generally exposing services over HTTP using JSON, in a RESTful manner, and sometimes we involve a queue where streams of updates are involved.

why?

We already had shared code for dealing with the likes of getting data from Twitter, Facebook and other external services, but used to run these in-process in whatever service needed them. There are a number of problems with this as we use the same data sources across more and more services, which are mitigated by moving connectivity to them to separate services.

Firstly, some APIs implement rate limits. By centralising access to these APIs we're better able to control how we use them, and make more efficient use of any limits. Where we get the same data across multiple projects, we're not wasting valuable API requests in doing so. In a similar vein, we can centralise and share accounts across many applications.

Secondly, some of the data we use is available from multiple sources. Take, for example, Twitter. We can get tweets from Twitter themselves, Datasift and Gnip, amongst others. It's really useful to be able to take that out of the hands of each application using Twitter data, which doesn't really care where it's getting its tweets from, and being able to configure it dynamically, across one or all products.

Thirdly, by pushing streaming updates onto a queue, we're able to cope with downtime of a component processing the updates without loss of data, since it'll just pick up from where it left off.

On the operations side, we also have smaller, single purpose processes so they're easier to monitor and understand. Better understanding extends to engineering, too; each component is smaller so it's easier for an engineer to keep everything they need to know in their head!

Finally, exposing data through JSON over HTTP means that we can quickly build new products on top of existing data very easily. And if we decide the data is worth sharing with others, we've already built an API that will let us do just that.

the cost

This doesn't come without some additional complexity, of course. Extra components need building, configuring and monitoring. We're using Puppet for configuration management and a combination of Nagios, Nimrod and PagerDuty for monitoring and alerting, although we're working on moving to sensu from Nagios (more from Adam on that soon!).

the future

We're also looking at applying this to Atlas: running many discrete ingest processes, rather one monolithic beast, as Fred mentioned previously. This will help both at runtime, in being able to manage different ingest processes independently, but also, and importantly, make the codebase easier to navigate!