we write about the things we build and the things we consume

written by Chris Jackson on 21 September 2015 in Atlas, engineering

apis vs feeds

what's in a name?

Here at MetaBroadcast we handle a lot of data. A lot of HTTP APIs, and a lot of feeds. What's the difference? Does it matter?

Lots of people think feeds are the old, and probably transmitted via FTP, while APIs are the shiny, new better thing that lives the web. API is a marketing bullet point and a feature of the trade show booth. Feed is the dirty reality, hidden until the contract is signed.

Or maybe feeds hark back to the days of blog aggregators. You can do nearly anything with a media RSS or an atom feed, right?

get this right, save time

The reality is that choosing a feed vs HTTP API is an absolutely crucial decision. At MetaBroadcast, much of what we do is engineering. Unfortunately this is an area where we engineers are often so wrapped up in their own concerns, that we totally forget what our users need, or how the world actually works.

There's a need for APIs and feeds in most systems. The choice has got very little to do with the latest technology, and is completely orthogonal to whether the format JSON, a character separator, some kind of XML, or some other very clever idea.

HTTP APIs are great for interactive things. For example, if you're making an app for an antique shop, you'll be needing an API call to get data for a single antique. You'll also want an API that shows a selection of antiques to browse, and/or a search API.

Conversely, feeds are great for moving data between places. Imagine your shop works with a style guide service, that pairs antiques with the perfect paints colours. Ideally they would offer an API that looks up the paints per antique, exactly as you would like to display it. More likely, you want to process their data to your needs, so you're going to need to sync it to your server periodically. This is where a feed works very well indeed. The sender generates the feed at a time of their choice. The receiver writes simple code to loop through the content and handle it.

apis for everything?

Imagine if you try to do this syncing with an API. For the system receiving the data, you'll have to write code that crawls the API - calling some index API first, then each piece of content in turn. If there are a number of different data types in the
database (eg antiques and paints) you might have to do this several times, or in a complex tree of calls.

If you are lucky the amount of data to be synced is small, and does not change very often. If you're unlucky, the amount of data is larger than you can reasonably crawl before it changes. For example, imagine a large database of antiques, where new items are added regularly and a small proportion of older items are also updated, eg to add extra photos or a better description. It takes you a day to crawl every antique in the database, but new antiques need to be shown within a few minutes of being added. You'll need code to update new items frequently, and also to crawl all items less frequently.

Meanwhile, the system producing the data is also struggling. Suddenly someone started making thousands of extra calls per hour to the API. Most of the calls were for data that hasn't changed in months. They're being made just in case a change happened. Everybody — computers and engineers alike, got very busy, while only achieving a simple data sync.

As you can see, trying to sync via an API is very hard. The only exception is to make some kind of API designed for syncing, that lists just the changes*

**good apis are hard**

So far so good. A few things to bear in mind, but unless you've seen a number of large software systems before, you've probably decided APIs are the perfect for pretty most situation. Just make sure they're good APIs, right? Enter the real world, and consider these factors:

1. Your company almost certainly needs to work with other companies. That tends to be how business works!
2. Their APIs probably are not quite what you need. Your APIs are probably not what they need.
3. Making an API that provides what everyone wants is possible, but blooming hard. You'll certainly find that the API calls you need to provide are not in the same shape as your data. To do it well, you have to denormalise, cope with eventual consistency, and accept that a cache will do little to disguise slow code or the wrong architecture.

**so, what do we learn?**

**DO** make good clean HTTP APIs that provide exactly the data an interactive user of your system requires.

**DO** use feeds as a practical, real world way to sync data between systems and partners.

**DON'T** use APIs for non-interactive integrations, where you actually wanted a feed.
DON'T make a basic API that is ignorant of user needs, looks exactly like your database structure, and think you did anything whatsoever to make the world a better place.

* APIs that pass changes between systems are REALLY hard, by the way, and that's a whole different topic for the future.

If you enjoyed the read, drop us a comment below or share the article, follow us on Twitter or subscribe to our #MetaBeers newsletter. Before you go, grab a PDF of the article, and let us know if it's time we worked together.