The Etsy Saga: From Silos to Happy to Billions of Pageviews a Month

Monday, January 9, 2012 at 9:10AM

Todd Hoff in Example

Seldom do we get to hear stories of the bumps and bruses earned by a popular website during its formative years. Ross Snyder, a Sr. Software Engineer at Etsy, changes that with an engaging talk he gave at Surge 2011: Scaling Etsy: What Went Wrong, What Went Right.

Ross gives a detailed and honest account of how Etsy went from a raw startup in 2005, to a startup struggling with their success in 2007, to the mean, handmade, super scaling, ops driven machine they've become in 2011.

There's lots to learn from this illuminating story of transformation:

Origin Story

- Etsy is a market for handmade goods. They connect buyers and sellers, similar to eBay, they don't handle goods.
- Started: June 2005, 3 guys in an apartment.

- Today: 250 employees, billions of pageviews a month. All in 6 years.

2007 - The Architecture Then

- Ubuntu, PostgreSQL, lighttpd, PHP, Python
- Business logic was all in PostgreSQL stored procedures. A very common architecture at one time.
- Front-end interacted with the database using stored procedure calls wrapped in PHP.
- Large central DB, with some partitioning by feature. The partitioning bought them some time.
- Site uptime wasn’t great which costs money for an e-commerce site.
- Regular maintenance windows meant the site would come down regularly.
- Big bang software deployments would often result in outages.

2008 - The Failed Sprouter Experiment

- Still a startup at this stage, 20-30 people tops.
- Conway’s Law: When a team makes a product the product ends up resembling the team that made it.
    - The team looked like: Dev, DBA, Ops. Devs write code, DBAs write stored procs, Ops deployed code. A silo organization with lots of walls between teams.
- To fix their scalability problems they created middleware layer called Sprouter - Stored procedure router.
- Sprouter was a Python daemon sitting between the webservers and the database. You give it some arguments, it calls the stored procedure, and returned the results. In theory it could some caching and sharding, but these features weren't implemented.
- The hope was the Sprouter would be easier to scale than the

database, because scaling an architecture built around stored procedures is nearly impossible.
- Ready in fall of 08. Deprecated in 09, it didn't work.
- Sprouter was still a silo architecture, which resembled the team that they had.
    - Created a lot of developer friction as DBAs were on the hook for all database work which is a huge schedule dependency. Remember, only DBAs could do database work.
    - Ops still had to deal with dependencies after it was deprecated. It turns out Sprouter was dependent on an earlier version of Python, which means they couldn't upgrade Python. So there's a lot of operational overhead.
    - Made deploys worse because Sprouter cached stored procedure identifiers, which were invalid during an upgrade. Deploys were an outage nearly every time.
    - Without sharding, the database was still a single point of failure.

2008 - The Great Etsy Culture Shift

- Chad Dickerson was hired as the CTO and later became the CEO. He brought in a new culture and the previous culture was no longer at Etsy.
- Removing the old culture also removed:
    - Devotion to PostgreSQL and stored procedures.
    - Fear of developers coding in SQL.
    - Fear of developers touching the product in general.
    - The idea that of infrequent and large deployments are the way to move code into production.
    - A general NIH attitude.

Spring 2009 - The Way Forward: Part 1

- DevOps to the Rescue
  - DevOps are Ops people who are also engineers and engineers who are also Ops.
  - Silos became bad.
  - Trust, cooperation, transparency, and shared responsibility became good.
  - We're all in this together. Let's help each other to get stuff done.

- Stabilize the site:
  - **Install/improve metrics and monitoring**.
  - Upgrade database vertically as far as possible. Still not enough, but bought some breathing room.
  - Give developers production **access to running code** so they can help fix problems. Developed a tool called StatsD. Though not all developers have root on production machines.

Continuous Deployment - The Way Forward: Part 2

- Added **continuous deployment**. Any engineer in Etsy can deploy the whole site to production at anytime. Happens 25 times a day because it's so easy. It's a **one button deploy**.
- Developed a tool called Deployinator. **Chef** is used for configuration management.
- Big win is that **small change sets** are going out all the time, not large deployments. If things go wrong they can quickly figure out what went wrong and fix it.
- Added tests that run on the code to verify the code works before deployment.
- The team **talks to each** other all the time on **IRC**.
- **QA is performed by developers**. There's no separate QA group or

phase in the process. Developers are in charge of making sure the code is solid. Small change sets make this easier.
- Database schema changes are the outlier in the happy continuous deployment story. They have one day a week which is **schema change day**. Feature flags are used to turn on and off features, so they flip the flag and the schema is in place they turn the flag on.
- An A/B testing scheme allows features to be just turned on for admins or to be released to a certain percentage of users. They create a **feature flag**, commit a bunch of code, and then turn on the flag so only developers can see it.
- Every developer is on call for **support for a week out of a year**. They will double up around the holidays. Ops has their own rotations so a developer and ops are always on call.

Spring 2009 - The Death of Sprouter - The Way Forward: Part 3

- Use an **Object Relational Mapper** to get around Sprouter. They wrote their own.
- Front-end PHP code now (again) talks directly to the database through the ORM.
- Using the ORM shifted some work to the webservers which are easy to scale.
- ORM does some front-end caching.
- ORM's are notorious bottlenecks, but they haven't hit that problem yet, but they are aware of the potential.

The Coming of the Shard - The Way Forward: Part 4

- With Sprouter neutralized, the database is still a single point of failure.

- As a lot of people from **Flickr** went to Etsy, so they adopted the sharding scheme from Flickr.
- Based on **MySQL** as a simple datastore. **Battle tested** from Flickr.
- Scales database **horizontally** to infinity and beyond.
- **Master-master** replication is used to remove SPOF.

Spring 2011 - Sprouter Turned Off

- Sprouter was released using a stop everything strategy. Feature development virtually halted while people are working on infrastructure instead and big deploys explode in your face.
- As evidence for how the culture changed, a couple of people worked on Sprouter's replacement on the side and they **gradually phased** it on over time. They couldn’t afford to focus just on Sprouter, even though they might have liked to.
- Finally in the Spring of 2011 Sprouter was removed from the code base.

2011 - The Etsy Architecture Now

- CentOS, MySQL, Apache, PHP.
- **Phasing out PostreSQL**. Still not done. Migration is on a feature by feature basis.
- **Python is gone**. They need to plug people in so they standardized on PHP and MySQL.

Lessons Learned

- **Open and trusting** is much better than closed and afraid.
- If you are doing something **clever it’s probably wrong**. If you are going to scale don’t take chances on untested front-end to database interaction approaches. Etsy is an e-commerce site, they aren’t

sending rockets to the moon. What they are doing isn't that super controversial, but it's not that common either.
- Architectural decisions made today will have a large impact long after you are gone.
- No hole is so deep that proven scaling strategies can't dig you out.
- There was no path forward. It was not planned out. It happened gradually over a few years after the cultural shift started.

While Etsy didn't quite live up to their NIH pledge, as they built most everything themselves, the culture/team observation, about how the team structure dictates software structure, is a deep one. As above so below, the ancients held. To see it so clearly drawn out in practice is remarkable. Interestingly, the transformation we see across the board in web development teams--agility, insurgency, small groups, independent action, loose coordination, goal driven-- are paralleled in the battle strategy and tactics of 4th Generation Warfare. We see centralization give way to a distributed core. It appears complex ever moving landscapes trigger a parallel evolution.

Related Articles

- Etsy Blog
- Flickr Architecture
- Optimizing for developer happiness
- Pushing: Facebook, Flickr, Etsy
- The Changing Face of War: Into the Fourth Generation
