AWS: the good, the bad and the ugly

Posted on December 18, 2012 by seldo

Here at awe.sm we have been hosted from the beginning on Amazon Web Services (AWS). Over the past 3 years we've learned a lot about what it's good at, what it's not so good at, and have formulated our own set of best practices for running a high-availability, high-performance system, which differ in some important ways from Amazon's own advice.

We're going to talk about two related things:

1. For people who have heard of AWS but haven't started using it yet, we thought we'd lay out both the **benefits and the challenges** we've encountered.
2. For those already using AWS, we go into some detail of the **best practices** we've picked up for running a high-performance service like ours, where uptime is one of the highest priorities.

It's not an exaggeration to say that AWS has radically changed the economics of running a technology startup, but so slowly and gradually that it crept up on the industry. Nobody realizes how many companies are using Amazon's Elastic Compute Cloud (EC2) somewhere in their stack until it has an outage, and suddenly it seems like half the Internet goes away. But it's not like Amazon got lucky: they have an awesome product. Everybody uses AWS because EC2 has radically simplified running software, by hugely lowering the amount you need to know about hardware in order to do so, and the amount of money you need to get started.

EC2 is a new way of running software

The first and most important thing to know about EC2 is that it is not merely a virtualized hosting service. A better way of thinking about it is as employing a **fractional system and network administrator**: instead of employing one very expensive person to do a whole lot of automation for you, you instead pay a little bit more for every box you own, and you have whole classes of problems abstracted away. Power and network topology, hardware costs and vendor differences, network storage systems — these are things you used to have to think about back in 2004 (or regretted not thinking about). With AWS and its growing crop of competitors, you no longer do — or at least not until you get much bigger.

Far and away, the biggest difference and advantage of using EC2 is **flexibility**. We can spin up a new box very, very quickly — about 5 minutes from thinking "I need some hardware" to logging into the shell for the first time, ready to go. This lets us do some things that just a few years ago would have been crazy, for example:

as easy as resetting the load balancer, and going forward with the new system means just shutting down the old boxes. Running twice as much hardware as you need, but just for 24 hours, is cheap and simple.

- our failure plan for some non-critical systems, where perhaps up to an hour of occasional downtime is acceptable, is to monitor the box, and if it fails, **spin up a new box and restore the system** from backups
- we can **scale up in response to load events, rather than in advance of them**: when your monitoring detects high load, you can spin up additional capacity, and it can be ready in time to handle the *current* load event — not the next one.
- we can **not worry about pre-launch capacity calculations**: we spin up what seems at a gut level like enough hardware, launch, and then if we find we've got it wrong, spin boxes up or down as necessary. This kind of iteration at the hardware level is one of the greatest features of AWS, and is only possible because they can provision (and de-provision) instances near-instantly.

EC2 makes strong financial sense for startups

The most obvious cost advantage of AWS is that it has **literally zero setup costs**: you use the same Amazon account you use to order random junk off the Internet, click a button, and start paying for servers, by the hour. You only pay for the boxes when they run, and you only pay for storage that's actually in use, so your startup costs are minimal, and it **encourages experimentation at the hardware level**: spin up 10x more capacity than you need, run load tests, and then spin them back down until you really need them. That's not just convenient, that's revolutionary. As with many other aspects of AWS, such a large quantitative difference becomes a qualitative one.

As I've already alluded to, AWS also **dramatically lowers operational overhead**. Until 2012, more than two years after we started the company, we had no dedicated operational staff at all. This was in retrospect a bad call — we should have hired one in 2011, maybe even earlier. But we still have only one full-time ops guy managing our whole fleet of over a hundred boxes. That's a pretty great ratio. The multiplier effect of (mostly) not needing to care about networking, power, etc. etc. is enormous and, once you've got used to it, under-appreciated.

Of course, it's not a total home-run. AWS is definitely more expensive than vanilla colo hosting. But they do a lot to take the sting out of that by routinely lowering their prices — 18% in October, 10% in March just this year. You should also factor in the money you save by having the ability to forgo running spare hardware in favor of spinning it up on demand. Finally, for long-term use, you can pay some money up-front and save a ton of money — over 50% — by using reserved instances. (At awe.sm, we are kind of crazy about reliability, so we run a lot of redundant hardware, and reserved instances have been a big win for us)

But EC2 has some problems

This is where the love letter ends, and the Amazon rep who was prepping a pull-quote for their front page starts to reconsider. While we love EC2 and couldn't have got where we are without it, it's important to be honest that not everything is sunshine and roses. EC2 has serious performance and reliability limitations that it's important to be aware of, and build into your planning.

co-located, but (in theory) isolated from each other in terms of networking, power, etc.. There's a few important things we've learned about this region-zone pattern:

1. **Virtual hardware doesn't last as long as real hardware**. Our average observed lifetime for a virtual machine on EC2 over the last 3 years has been about 200 days. After that, the chances of it being "retired" rise hugely. And Amazon's "retirement" process is unpredictable: sometime they'll notify you ten days in advance that a box is going to be shut down; sometimes the retirement notification email arrives 2 hours after the box has already failed. Rapidly-failing hardware is not too big a deal — it's easy to spin up fresh hardware, after all — but it's important to be aware of it, and invest in deployment automation early, to limit the amount of time you need to burn replacing boxes all the time.

2. **You need to be in more than one zone, and redundant across zones**. It's been our experience that **you are more likely to lose an entire zone than to lose an individual box**. So when you're planning failure scenarios, having a master and a slave in the same zone is as useless as having no slave at all — if you've lost the master, it's probably because that zone is unavailable. And if your system has a single point of failure, your replacement plan cannot rely on being able to retrieve backups or configuration information from the "dead" box — if the zone is unavailable, you won't be able to even see the box, far less retrieve data.

3. **Multi-zone failures happen, so if you can afford it, go multi-region too**. US-east, the most popular (because oldest and cheapest) AWS region, had region-wide failures in June 2012, in March 2012, and most spectacularly in April 2011, which was nicknamed cloudpocalypse. Our take on this — and we're probably making no friends at AWS saying so — is that AWS region-wide instability seem to frequently have the same root cause, which brings me to our next point.

To maintain high uptime, we have stopped trusting EBS

This is where we differ sharply from Amazon's marketing and best-practices advice. Elastic Block Store (EBS) is fundamental to the way AWS expects you to use EC2: it wants you to host all your data on EBS volumes, and when instances fail, you can switch the EBS volume over to the new hardware, in no time and with no fuss. It wants you to use EBS snapshots for database backup and restoration. It wants you to host the operating system itself on EBS, known as "EBS-backed instances". In our admittedly anecdotal experience, EBS presented us with several major challenges:

- **I/O rates on EBS volumes are poor**. I/O rates on virtualized hardware will necessarily suck relative to bare metal, but in our experience EBS has been significantly worse than local drives on the virtual host (what Amazon calls "ephemeral storage"). EBS volumes are essentially network drives, and have all the performance you would expect of a network drive — i.e. not great. AWS have attempted to address this with provisioned IOPS, which are essentially higher-performance EBS volumes, but they're expensive enough to be an unattractive trade-off.

- **EBS fails at the region level, not on a per-volume basis**. In our experience, EBS has had two modes of behaviour: all volumes operational, or all volumes unavailable. Of the three region-wide EC2 failures in us-east that I mentioned earlier, two were related to EBS issues cascading out of one zone into the others. If your

- **The failure mode of EBS on Ubuntu is extremely severe**: because EBS volumes are network drives masquerading as block devices, they break abstractions in the Linux operating system. This has led to really terrible failure scenarios for us, where a failing EBS volume causes an entire box to lock up, leaving it inaccessible and affecting even operations that don't have any direct requirement of disk activity.

For these reasons, and our strong focus on uptime, **we abandoned EBS entirely**, starting about six months ago, at some considerable cost in operational complexity (mostly around how we do backups and restores). So far, it has been absolutely worth it in terms of observed external uptime.

Beware: other AWS services rely on EBS

Because some AWS value-added services are built on EBS, they fail when EBS fails. This is true of Elastic Load Balancer (ELB), Relational Database Service (RDS), Elastic Beanstalk and others. And EBS — in our experience — seems to nearly always lie at the core of major failures at Amazon. So if EBS fails and you need to suddenly balance traffic to another region, you can't — because your load balancer also runs on EBS. And you can't launch new hardware anyway, because the console app that AWS provides to launch it also runs on EBS. So we love EC2 (and really really love S3), but **we don't use any other AWS value-added services**. A side-benefit of this approach is that we can switch relatively simply to other hosting providers, and are not locked too closely to AWS.

Lessons learned

If we were starting awe.sm again tomorrow, I would use AWS without thinking twice. For a startup with a small team and a small budget that needs to react quickly, AWS is a no-brainer. Similar IaaS providers like Joyent and Rackspace are catching up, though: we have good friends at both those companies, and are looking forward to working with them. As we grow from over 100 to over 1000 boxes **it's going to be necessary to diversify to those other providers**, and/or somebody like Carpathia who use AWS Direct Connect to provide hosting that has extremely low latency to AWS, making a hybrid stack easier.

I hope this has been of use to you. If your company needs social analytics — and it probably does, even if you don't know it — you should check us out. And if you want to join our crazy crew, we're always hiring good people.

This entry was posted in Engineering. Bookmark the permalink.

59 Responses to *AWS: the good, the bad and the ugly*

richtaur *says:*
December 18, 2012 at 12:37 pm

Excellent article, Laurie! We're using AWS for all of our web stuff these days too, so good to know where the trouble areas are.

Brian Whalley *says:*
December 18, 2012 at 12:41 pm