Network & Hosting Architecture

The network and hosting architecture was one of the first decisions tackled as part of the project. Once the basic project requirements had been defined the next major task was to decide on the vendors and technologies which would be used to deliver the product.

# Platform

In terms of platform, I initially considered three major different platforms: Microsoft’s web stack, a Java web stack, and Ruby on Rails. Each of these possibilities would have profound impacts on the network and hosting architecture. After some preliminary prototyping with Clojure (a Lisp variant built on a Java framework) and Ruby on Rails, I decided that I would be biting off a significant amount of work in starting the project with a platform that I had not already mastered. So, I fell back on Microsoft technologies, which I use every day at the “day job”.

# Network Goals

With the platform decided on, I next defined what was most important to me in designing the network tier. I arrived with the following shortlist:

* Low level of hardware and network administration
* Flexibility to run any software necessary
* Low cost
* Good level of performance and availability for the price point

# Possible Providers Researched

I was immediately sure I wanted to host on some form of cloud-based infrastructure. The reasonable cost, the built-in functionality for load balancing and firewalls, the reduced administration demands, and the ability to quickly scale made it very attractive. And it would also be an excellent learning experience! The three major vendors I examined were:

## Microsoft Azure

Azure operates under a Platform-as-a-Service model – you never see or touch the underlying operating system, presumably some form of Windows. Azure fulfilled most of my network goals, but there was one disturbing downside that prevented me from pursuing it: developing against Azure is slightly different than normal Windows development, and the way the applications are coded, deployed and hosted makes them different than if they were on a “normal” server. I don’t really like this type of vendor lock-in, so I elected to give Azure a pass.

## Google App Engine

Google’s App Engine is also a Platform-as-a-Service model. I was initially quite excited about it, and played around with some of their tech demos, but my language and platform selection quickly ruled it out – Java and Python only.

## Amazon EC2

Amazon Web Services (AWS) was one of the original companies to introduce “cloud computing”, and as such their offerings are both a little more mature, and a little less refined. They follow an Infrastructure-as-a-Service model; you have complete control over the OS and applications installed on it, only the hardware is virtualized. This means there is much more administration that needs to be performed – the OS must be configured and kept patched and up to date, and necessary software must also be installed, configured and maintained. However, this also means that the user has greater control – if I need something out of the ordinary, or configured just so, I have the power to do that.

# Final Hosting Provider Selected

I opted for the greater flexibility offered by EC2. I was also impressed by the other services provided by Amazon Web Services – a content delivery network, cloud-based database capability, abstracted storage, abstracted queuing, monitoring and notification capabilities, the list goes on. AWS’ major negative, the increased administration needs, did not bother me greatly – I have experience in maintaining servers already. In the end I felt it the best all around solution:

* Flexibility of running a “bare” server
* Scaling up and out is simple
* Low cost
* No vendor lock-in during development
* Other valuable offerings for future needs: monitoring, content delivery network, cloud-based RDBMS, etc.

# Network Topology

With the vendor decided, I now focused on what my network topology would consist of. To be honest, my initial needs are very modest. The site would be fairly lowly trafficked, and most of the dynamic data could be aggressively cached as it changes fairly infrequently. The more time-sensitive and fluid data is all stored at eBay. I can get by initially with a single-tiered architecture, Figure 1 below.

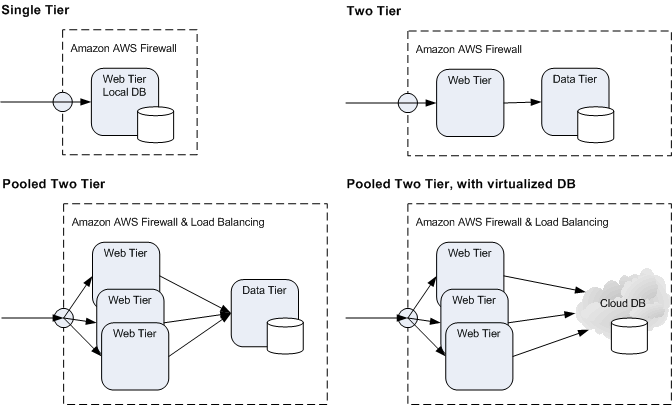
As traffic increases, the logical breakout point is to separate the database from the web tier, as shown in Figure 2. This would increase the overall throughput of the site, and leave it much less vulnerable to performance spikes. However, the database server performs much less work overall than the web server, so this scaling would probably only represent a 20% performance improvement.

As the number of stores increase, however, the site would run the risk of getting hit with a particularly popular store, which could end up dragging down all the others performance-wise. For this reason, it would make sense to starting scaling out the web tier, as shown in Figure 3. Higher traffic stores could be placed on their own web servers, and many low-traffic stores could cohabit on other web servers. Spikes such as the Christmas rush, or a Slashdot effect could also be dealt with in this manner, though it would be a manual effort.

Finally, at some point, traffic might start to exceed the abilities of the data tier server. At this point, there are two options – scaling up the data tier server by adding more and more resources (RAM, CPUs, etc.) or moving the data tier completely off a single node and hosting it in the cloud (Figure 4). Though this last option may be relatively expensive, it guarantees a massive level of throughput.

1.

2.



3.

4.

# Conclusion

I have chosen Amazon Web Service’s EC2 infrastructure as my hosting environment, given their unique blend of hardware abstraction and direct OS control. Current infrastructure needs are very light, but AWS EC2 offers several possible paths for scaling the site both up and out.

# References

Microsoft Windows Azure and Amazon EC2 on Collision Course  
<http://www.pcworld.com/article/199919/microsoft_windows_acure_and_amazon_ec2_on_collision_course.html?tk=rss_news>

Architecting for The Cloud: Best Practices  
<http://jineshvaria.s3.amazonaws.com/public/cloudbestpractices-jvaria.pdf>

Cloud Architectures  
<http://jineshvaria.s3.amazonaws.com/public/cloudarchitectures-varia.pdf>

Web Application Hosting in the AWS Cloud  
<http://media.amazonwebservices.com/AWS_Web_Hosting_Best_Practices.pdf>

Google App Engine  
<http://code.google.com/appengine/docs/whatisgoogleappengine.html>

Microsoft Azure  
<http://www.microsoft.com/windowsazure/products/>