



Infrastructure options for Social Media Applications

Table of Contents

Introduction.....	2
Marketing within a Social Network.....	2
Application Life Cycles.....	3
The 'traditional' hosting model.....	5
The 'public cloud' hosting model.....	8
The data centre as a server.....	10
Billing	10
Conclusions.....	12





Introduction

Social networking sites have existed since the late 1990's, but it's only over the past few years that we have really seen them become truly mainstream. MySpace was set up in 2003 and was rapidly adopted by record companies to promote bands, by allowing you to 'friend' your favourite artists. In 2005, when MySpace recorded more page hits than Google, other big businesses started to take note.

Marketing within a Social Network

Due to the free subscription basis of most social networking sites, business usage tends to revolve around the marketing of products or services – anything from simple banner ads, to the more complex, loyalty-based relationships built up by record and film companies between their artists and site users. The sites themselves display adverts used to generate revenue for the site vendor, using data collected from user's own profiles and friends lists, to direct content-based advertising, targeting the common interests of the site's owner and their readership. Publicly displaying details of their likes and dislikes, the owners of social networking sites, through this 'self branding', have become a significant resource for marketers.

With the rise of Facebook as the most popular social networking site, the advertising model has taken on some subtle changes. The ability to install applications (or 'apps') has led to various different ways for businesses to harvest data about the users, and to generate direct revenue. Here are a few of the main ones:

Cost Per Click (CPC) – This commonly revolves around a quiz application, where the user has to register to proceed, providing their email address and sometimes other details too. The quiz results (often just a thinly veiled marketing questionnaire) will be collected, along with the supplied contact details. Advertisers pay per completed quiz result for use as lead generation.

Branding within an application – This usually involves site users making lists of their favourite things – cars, bands, mp3 players, etc. It creates increased exposure for featured products as the results are displayed





in a user's friends newsfeed, and can be seen to improve a brand's credibility, and thus desirability, if it's regularly featured in a top spot.

Virtual Currency – This is based around a game which can be played entirely for free, but which takes an huge amount of time to progress in - usually something along the lines of building a virtual farm, city, gangster empire, etc. To save time, you can buy game currency, energy points, or whatever else is needed to quickly improve your status within the game. The sale of virtual assets for a small sum of real money is called a **microtransaction**. Some of games also unlock resources if you subscribe to services, such as video rentals, provided by various businesses affiliated with the application developer.

While the individual payments involved in each of these approaches is extremely small, they rely on the game or other application 'going viral' and being picked up by potentially millions of other users. To do this, they all leverage the 'social' aspect of the site and include a prompt to recommend it to a friend, in the hope that a large number of people will start using it. If the application is a success, the net profit generated can be staggering.

Application Life Cycles

Once development and testing of an application has been completed, it's usual to trial it with a **beta release**, targeted at a limited number of users. During this period, the use of the application will often be restricted to a pre-determined number of maximum users while it is monitored in a 'live' environment, allowing any bugs encountered to be fixed before release to a larger audience. Depending on the company's requirements, and the type of application, this period could be months, weeks or even as short as days. Alternately, a decision may be taken to go live without any beta testing.

Due to the fickle nature of the internet, it can be extremely difficult to gauge what applications will 'hit' and which will 'miss'. Some of the biggest hits have come about from some very simple, seemingly trivial ideas, whereas some flashy, high concept applications can end up failing spectacularly. For this reason, once the



decision is made to '**go live**', the application should be closely monitored for the first few weeks after **launch day**.

The life span of internet applications tend to operate like fads, rather than traditional software life-cycles. Games and applications can be huge one week and then forgotten the next, as users move onto the next big thing. So these first weeks are particularly important, because if the application 'hits', it can result in an exponential **period of rapid growth**, resulting in potentially millions of users, and it's important the application developer is ready to make the most of this window of opportunity.

If the take-up is much lower than expected, and the application fails to hit within a pre-decided post launch period, it's often preferable to pull its release entirely and concentrate on the release of your new project. Alternately, you can choose to restrict the size of the user base (as per the beta release) so it can be left to run, relatively unmonitored, without the fear that it's going to be affected by a sudden and unexpected increase in users.

Once the growth period levels out, the application enters a period of **maturity**, where the user base remains fairly static. Depending on the type of application, and whether the developers are able to expand and refine it to retain user interest, this period may last anything from months to years. The market for apps is still too new for the upper limits of this life span to be known yet, but with continual development there is certainly no reason why an application shouldn't remain popular for several years.

From maturity, particularly if it's not updated or developed further, the application will eventually succumb to a **gradual decrease** in its user base. Once the user base has decreased to levels comparable to its initial beta launch, the application maybe considered to have entered a **niche phase**, where it's only being used by a stable group of core users, and isn't being actively developed. If the application developer is happy to allow it to remain online, this period could last years. However, once it's decided it's no longer profitable to maintain the application, it will be taken offline and **discontinued**. It's usually common courtesy to give the remaining users some prior warning before discontinuing the service, often done by displaying a message





within the application itself. This is also a good time to attempt a form of up-selling to newer applications, often with the temptation of a loyalty reward – if it's a game, this will often be virtual currency.

Some social networks, such as MySpace, host all applications on their own servers. Developing applications for such 'in house' networks doesn't confront you with any hosting considerations - at least where the application itself is concerned.

However, the big trend at the moment, particularly with Facebook, is for applications to be hosted externally, by the application developer or one of their affiliates. The application is then loaded into a template page – called a **canvas page** – and authenticated against the users Facebook login details using **OAuth**. This permits a single login to all applications via the users account, without the application developer needing to know the user's username or password details. While the front-end of this system is handled by Facebook, the application now needs to be hosted somewhere.

So, what are your hosting options?

The 'traditional' hosting model

This involves hosting the application on one or more physical servers. For a very small, niche usage application, you could potentially use a single-server solution, but a minimum configuration would usually involve at least a web server and a separate database server, each optimised for their specific role.

However, this isn't going to leave much room for growth. A more realistic solution would involve two or more web servers, sat behind a high-availability load-balancer, and two data bases servers with data replication enabled. Each of these servers would typically use single or dual Quad-core Xeon processors, with 8GB RAM for each of the web servers and 16GB RAM for each of the database servers, and 2 x 1TB hard drives in a RAID 1 configuration. The solution would also include some form of off-server backups.

The **Capital Expenditure** (CapEx) for such a solution can vary significantly, depending on whether you buy





and configure your own hardware, and then host it within a co-location centre, or purchase the solution through a single vendor who will configure the hardware and provide the hosting.

The first option has considerable initial outlay, as servers have to be purchased and configured before they can be hosted. Purchasing from a reputable server supplier (such as Dell or HP), the hardware for a solution of the type detailed above would run to several thousand pounds. The exact figure would depend upon the number of servers and their specification, but this could easily be somewhere between £5-8k and much higher. You will then need to host your servers in a co-location centre – a data centre that allows third-party hosting. Co-location contracts usually don't include, as standard, any hands-on technical support, except server reboots, and are unlikely to include any backup services. Some data centres may offer these services as additional extras, but this will increase your monthly/yearly costs. Alternately, you may want to source your own backup solution and have your own technical staff on standby to deal with any support issues. But, again, this will increase your setup and operating costs.

The second option, using a single vendor, has several advantages. The vendor will usually have some stock machines racked in a data centre, ready to be configured and deployed, so your hosting environment can be brought online much more quickly. And your contract will usually include some form of backup solution, technical support and full hardware support.

Both hosting options will usually involve one-year minimum contracts, and may include additional fees for extras such as bandwidth, managed support, etc. These contracts usually have the option to be prepaid for the year, or billed monthly. As no hardware needs to be purchased up front, the second option provides the lower CapEx, although it may reduce some of your options regarding specific hardware selection. Yearly costs in both instances will run to tens of thousands of pounds.

Web server – While the default Linux web server, **Apache**, is quite capable of handling huge volumes of web traffic, a popular alternative is **Nginx** (pronounced “engine X”). This light-weight web server was originally developed to handle websites run by the Russian portal Rambler, but has been gaining increasingly



popularity due to its ability to handle more concurrent connections with a much lower memory and CPU usage. This doesn't guarantee switching to Nginx will automatically make your application run faster, but it is something you might want to test during the development stage. It is currently the fourth most popular web server, used on 8.77% of all active sites.

Now, let's look at some 'hit' and 'miss' scenarios for application launches using the hosting model described above.

Hit - You release a game that rapidly becomes successful. By the end of the launch day you have 10,000 users. By day two you have 15,000 users, by day three you have 40,000 users... By this point, if your hosting provider hasn't been able to rapidly deploy some additional servers for you, the servers will be running at full capacity, and the application will probably be suffering from periods of downtime as they struggle to cope.

If the rate of uptake continues to increase at this rate, you will need to add multiple additional servers to your cluster daily to have any chance of meeting demand. In this situation, it's highly likely that you won't be able to match this rapid growth by continually deploying new physical servers across data centres, and your application will finally run out of resources and grind to a halt.

NB: The figures above based on the case study for Facebook game 'FarmVille', which attracted 6 million active daily users within the first two weeks, and grew to 10 million within 6 weeks. It's doubtful any data centre is going to be able to scale up your cluster of physical servers quickly enough to meet that level of demand.

Miss – You release a game which has only minimal uptake. By the end of launch day you have zero users. Two weeks later you have fewer than 100 active users. Unless something pretty dramatic happens, your project has just flat-lined. However, you still have a year long contract with your hosting supplier, and potentially additional CapEx costs created by hardware and staffing. Unless you have several new applications to roll out quickly, that are successful, you're going to struggle to cover your costs.





The 'public cloud' hosting model

The '**cloud**' is a buzzword that many people have heard, but relatively few of them could give you a clear explanation of what it actual means. Put simply, the cloud refers to internet-based computing, where applications and data are stored and accessed on-demand. These applications and data are stored, not on your home/office computer, but within the 'cloud'.

The 'cloud' is essentially just a synonym for the internet, and derives its name from telecommunication and networking flowcharts, where a cloud symbol is often used to represent the internet or telecom network.

In practice, this means that data stored in the cloud is stored on servers owned by the provider of a cloud service. For example, in the traditional computing model, if you want to write a letter, you need to install some word processing software, write the letter and then save it to the hard drive on your PC. Using the cloud model, we might log onto a word processor application (e.g. Google's 'docs' application) via a web browser, write the letter and then save it on the provider's servers (in the case of 'docs', it will be saved somewhere within Google's vast farms of servers). The letter can then be accessed anywhere in the world, on-demand, via a web browser, by logging onto your 'docs' account.

This is a very simple example, but it illustrates the basic concept.

A cloud can be public or private. A **private cloud** is a propriety network that provides web-based services to a limited number of clients. This provides the customer assurance as to where their data is being held. Some large companies set up private clouds to host their own systems – providing some of the flexibility of cloud but the security of dedicated resources.

A **public cloud** is available to anyone on the internet, normally with no information as to where a given application is physically hosted. Currently, **Amazon Web Services (AWS)** are the biggest provider of public cloud services. They are also the biggest hosting provider of Facebook applications.





As an example of cloud hosting, let's see what our previous hosting model would look like using Amazon's services:

The web servers would be hosted on **Amazon Elastic Computing Cloud** (Amazon EC2) service. This allows you to deploy any number of virtual servers – called '**instances**' - from your web-based AWS Management Console. You can launch these using one of several pre-configured templated images or, more usefully, you can create a **Amazon Machine Image** (AMI), containing your required applications, configuration settings, libraries, etc, allowing you to deploy multiple, identical server instances within minutes. We can configure an image with our web server requirements and then deploy two of them. If we find we need more, we can then use the AMI to deploy more within minutes.

As we're hoping our application will be popular, we going to assume there will be a lot of traffic going to these servers, so we will bring these instances up behind Amazon's **Elastic Load Balancing** (ELB). This service monitors the health of EC2 instances, distributing traffic across the instances to try and maintain a manageable load across all instances.

Our database will be handled by the **Amazon Relational Database Service** (Amazon RDS). This is a virtualized database server, which is fully MySQL compatible. Any database designed for the MySQL engine, should work unmodified with RDS. To match our physical model, we will bring up two of them and setup replication.

We will use the **Amazon Simple Storage Service** (Amazon S3) for our backups. This service allows you to write, read and delete objects containing from 1 byte to 5 gigabytes of data each. You can store an unlimited number of objects. Each object is stored in a container called a **bucket**, which can be retrieved via a developer-assigned key.





The data centre as a server

While **virtualization** is not a prerequisite for hosting cloud services, it is the foundation of the cloud platform's ability to be hugely scalable (or 'elastic') and to meet the demands of web-scale computing. Google has stated that data centres powering cloud services should be considered and managed as a single computing entity, rather than as a room full of separate servers. This approach begins to make sense when you consider the scale of the resources available within an entire data centre. Let's assume each server within the cloud service has 1TB of disk space available and 16GB RAM – if there are 2000 servers available, this gives the cloud platform access to 1.6TB of RAM and 2000 TB (or 2 Petabytes) of disk space. Virtualized instances can be deployed across multiple physical servers, and the instances themselves can grow across multiple servers to allow access to far greater resources than an single machine would allow. While each individual machine has the usual hardware limits, treating the data centre as a server offers the illusion of near infinite resources.

Billing

The biggest difference between physical hosting and public cloud computing is the billing model. Public cloud services treat **Software as a Service** (SaaS), using a **utility billing** model to charge for services on a per usage basis. The most basic EC2 instance (a '**small instance**') can be deployed for as little as 5p an hour – one hour being the minimum deployment period. While an instance this small isn't really sufficient by itself for live business use, higher specification instances can be launched for 45-77 p/h. Additional services, such as S3, RDS and bandwidth changes, are extra, of course, but it does mean a fully-featured solution can be deployed for several pounds per hour, without any up front expenditure. This allows you to convert much of your CapEx to **Operating Expenditure** (OpEx), as your up-front payments are greatly reduced.

Utility billing also allows instances to be deployed for several hours testing, and then switched off again, and only be billed a few pounds for the uptime. This means you can test your application or server configuration, without the risk of running it on a live server, or the workload/cost of setting up a separate development





server. For example, if you wanted to compare how well your application runs on Nginx vs Apache, you could use the AMI for your web server and deploy two instances – one running Nginx and one running Apache, and load test your application on them for several hours, gathering your testing metrics, and then take them off line.

Under this billing model, a modest system – such as the one detailed – could cost end up costing hundreds of pounds per year, compared with the tens of thousands of pounds the physical solution would cost. Of course, if your app is incredibly successful, and you have to deploy hundreds of instances, and additional bandwidth charges, the year end figures may not be much different and the public cloud approach could prove more expensive – but at least these costs didn't need to all be paid up front, allowing you time to earn money as you spend it.

Now that we've had a look at how the cloud and AWS work, let's take another look at our earlier 'hit' and 'miss' scenarios, but this time hosted in the cloud:

Hit – By end of day one we're seeing a constant load across both web servers, as we hit the 10,000 user mark. Using our AMI, we deploy another two web servers. On day two we see the load increase again, so deploy ten additional web servers... and so on. Database servers can be deployed, with the same ease and speed, if we find our two RDS instances are struggling.

Miss – By end of day one our game has no users. By the end of week two we have fewer than 100 users. At this point we will probably want to consider cutting back services – e.g. removing one of the web server instances and load-balancing, and just running a single database server. If things don't improve, we might then consider either discontinuing the project completely or moving it to a shared account on an existing physical server.



Conclusions

As illustrated in the examples above, the biggest advantage of cloud based hosting in comparison with a physical solution, is rapid scalability. As shown in the 'hit' scenario, it would be incredibly difficult to scale up physical services to match such rapid growth. The physical solution cannot compete with the ability to deploy, in some cases, 100 servers in half an hour – something which is easily possible within the Amazon public cloud environment. The cloud solution also requires much less CapEx, which is obviously attractive to any business, but particularly so to small, start-up businesses or solo developers. The CapEx is converted into OpEx, which can be paid while you earn. This makes it far easier, at least financially, to start developing and hosting applications.

The physical solution poses the system architect with a dilemma – do you massively scale your infrastructure to anticipate a huge demand, or deploy a more modest system and put in place some contingency plans to expand it, should the need arise. This would be done with the knowledge that, however much you spend and however big you make your system, it may not be enough to meet the levels of rapid growth that can accompany the launch of a successful application. In either case, you risk either overspending or under-provisioning. The first case can leave you considerably out of pocket, with less than expected income to cover the costs. In the latter case an otherwise successful launch can be stopped dead in its tracks as the hosting infrastructure fails to match the rate of growth.

Very few businesses will exist solely in the public cloud. In many instances a business will use a **hybrid solution**, utilizing a mixture of traditional dedicated servers and private and public cloud servers for different aspects of their business. A corporate website and database servers might be kept 'in house' on a dedicated server platform, with rapid growth sites and storage hosted on cloud-based servers. A private cloud might be used for an Intranet or data processing, depending upon the needs of the business.

While the cloud solution isn't perfect for every hosting situation, given the fickle nature of social networking





site applications, and the difficulty in predicting which ones will hit and which ones will miss, it does offer some clear advantages for this particular market, and the flexibility to expand or shrink your resources at will.

For further information on the hosting options available, contact our team on **0845 4210 444**.

ForLinux Limited, Innovation House, Beacon Hill Office Park, Cafferata Way, Newark, Nottinghamshire, NG24 2TN

T: 01636 881200

E: info@forlinux.co.uk

W: www.forlinux.co.uk



INVESTOR IN PEOPLE

Registered in England & Wales Company No. 04227715