3. before the Move into the Cloud

1. Know Your Software Licenses

With many cloud environments in operation today, you pay for resources by the CPU-hour. The cheapest virtual machine in the Amazon Cloud, for example, costs $0.10 for each hour you leave the instance in operation. If it is up for 10 hours and then shut down, you pay just $1.00—even if that is the only use you make of the Amazon cloud for that month.

In a real world, you might have the following operating scenario:
• From midnight to 9 a.m., run your application on two application servers for redundancy's sake.
• From 9 a.m. to 5 p.m., launch six additional application servers to support business-hour demand.
• For the evening hours through midnight, reduce the system down to four application servers.

Adding all that up, you pay for 110 hours of computing time. If you were using physical servers, you would have to purchase and run eight servers the entire time.
Unfortunately, not all software vendors offer licensing terms that match how you pay for the cloud. Traditional software licenses are often based on the number of CPUs. An organization that uses 10 application servers must pay for 10 application server licenses—even if 5 of them are shut down during the late night hours.
So, when moving to the cloud, you must understand your licensing terms and, in particular:
• Does the software license support usage-based costs (CPU-hour, user, etc.)?
• Does the software allow operation in virtualized environments?

The ideal cloud-licensing model is open source. In fact, the flexibility of open source licensing has made the Amazon cloud possible. If you can remove licensing concerns altogether from your cloud deployments, you are free to focus on the other challenges of moving to the cloud.

The best licensing model for the cloud is one that charges by the CPU-hour. As the cloud catches on, more and more software vendors are offering terms that support hourly license charges. Microsoft, Valtira, Red Hat, Vertica, Sun, and many other companies

have adopted per-CPU-hour terms to support cloud computing. Oracle promotes their availability in the cloud but unfortunately retains traditional licensing terms.

Software that offers per-user licensing can work adequately in the cloud as well. The challenge with such software is often how it audits your licensing. You may run the risk of violating your terms, the license may be tied to a specific MAC or IP address, or the software license management system may not be smart enough to support a cloud environment and unreasonably prevent you from scaling it in the cloud.

The worst-case scenario* in the cloud is software that offers per-CPU licensing terms. As with some per-user systems, such software may come with license management tools that make life difficult.

2.The Shift to a Cloud Cost Model

You pay for resources in the cloud as you use them. For Amazon, that model is by the CPU-hour. For other clouds, such as GoGrid, it's by the RAM hour. Let's look at how you can anticipate costs using the example resource demands described earlier (two application servers from midnight until 9 a.m., eight from 9 a.m. until 5 p.m., and four from 5 p.m. until midnight).
Suppose your core infrastructure is:
$0.10/CPU-hour: one load balancer
$0.40/CPU-hour: two application servers
$0.80/CPU-hour: two database servers
Each day you would pay:
$2.40 + $44.00 + $38.40 = $84.80
Your annual hosting costs would come to $30,952.00—not including software licensing fees, cloud infrastructure management tools, or labor.

How to Approach Cost Comparisons:

The best way to compare costs in the cloud to other models is to determine the total cost of ownership over the course of your hardware depreciation period. Depending on the organization, a hardware depreciation period is generally two or three years. To get an accurate picture of your total cost of ownership for a cloud environment, you must consider the following cost elements:

• Estimated costs of virtual server usage over three years.
• Estimated licensing fees to support virtual server usage over three years.
• Estimated costs for cloud infrastructure management tools, if any, over three years.
• Estimated labor costs for creating machine images, managing the infrastructure, and responding to issues over three years.
• Any third-party setup costs for the environment.

In comparison, you must examine the following elements of your alternatives:

• What are your up-front costs (setup fees, physical space investment, hardware purchases, and license fee purchases)?
• What labor is required to set up the infrastructure?
• What are the costs associated with running the infrastructure (hosting space, electricity, insurance)?
• What is the labor costs associated with supporting the hardware and network infrastructure?
• What are the ongoing license subscription/upgrade fees? Maintenance fees?

A Sample Cloud ROI Analysis:

Let's perform an ROI analysis of a specific infrastructure that compares building it internally to launching it in the cloud.

This particular example assumes that two application servers easily support standard demand. It also assumes, however, that the business has a peak period of traffic on the 15th day of each month that lasts for 24 hours. Serving this peak capacity at the same performance levels as standard capacity requires an extra four servers.

If you're starting from scratch, you will minimally need the following equipment for your IT shop:

• Half rack at a reliable ISP with sufficient bandwidth to support your needs
• Two good firewalls
• One hardware load balancer
• Two good GB Ethernet switches
• Six solid, commodity business servers

For the cloud option, you will need just a few virtual instances:

• One medium 32-bit instance
• Four large 64-bit instances during standard usage, scaled to meet peak demand at 8

In addition, you need software and services. Assuming an entirely open source environment, your software and services costs will consist of time to set up the

environments, monitoring services, support contracts, and labor for managing the environment. Table 3-1 lays out all of the expected up-front and ongoing costs.

To complete the analysis, you need to understand your organization's depreciation schedule and cost of capital. For hardware, the depreciation schedule is typically two or three years. For the purposes of this example, I will use the more conservative three-year schedule. This schedule essentially defines the expected lifetime of the hardware and frames how you combine monthly costs with one-time costs.

Costs initiated with different infrastructures:

	Internal(initial)	Cloud(initial)	Internal (montly)	Cloud (montly)
Rack	**$3,000**	**$0**	**$500**	**$0**
Switches	**$2,000**	**$0**	**$0**	**$0**
Load balancer	**$20,000**	**$0**	**$0**	**$73**
Servers	**$24,000**	**$0**	**$0**	**$1,206**
Firewalls	**$3,000**	**$0**	**$0**	**$0**
24/7 support	**$0**	**$0**	**$0**	**$400**
Management s/w	**$0**	**$0**	**$100**	**$730**
Expected labour	**$1,200**	**$1,200**	**$1,200**	**$600**
Degraded performance	**$0**	**$0**	**$100**	**$0**
Totals	**$53,200**	**$1,200**	**$1,900**	**$3,009**

The cost of capital for most organizations lies in the 10% to 20% range. In theory, the cost of capital represents what your money is worth to you if you were to invest it somewhere else. For example, if you have $10,000, a 10% cost of capital essentially says that you know you can grow that $10,000 to $11,046.69 (with the cost of capital being compounded monthly) after a year through one or more standard investment mechanisms. Another way to look at it is that gaining access to $10,000 in capital will cost you $11,046.69 after a year. Either way, the cost of $10,000 to your business is 10%, calculated monthly.

The financial expression is that you want to know the present value of all of your cash outflows over the course of the depreciation period.

For each option, calculate the present value of your monthly payments and add in any upfront costs:

Internal: = (–PV(10%/12,36,1900,0)) + 53200 = $112,083.34

Cloud: = (–PV(10%/12,36,3900,0)) + 1200 = $94,452.63

Not only is the cloud cheaper, but the payment structure of the cloud versus up-front investment saves you several thousand dollars.

The final point to note is that you can use the $112,083.34 and $94,452.63 numbers to help you understand how much money these systems need to generate over three years in order to be profitable.

Where the Cloud Saves Money:

If you always have the same steady amount of usage, you won't see most of the key cost benefits of being in the cloud.

Cost savings in the cloud become significant, and even reach absurd levels, as your variance increases between peak and average capacity and between average and low capacity. My company has an extreme example of a customer that has very few unique visitors each day for most of the year. For about 15 minutes each quarter, however, they have the equivalent of nearly 10 million unique visitors each month hitting the website. Obviously, purchasing hardware to support their peak usage (in other words, for 1 hour per year) would be insanely Wasteful.

A common and relatively straightforward set of cost savings lies in the management of nonproduction environments—staging, testing, development. In the cloud, you can replicate your entire production infrastructure for a few days of testing and then turn it off.

3. Service Levels for Cloud Applications

A service level agreement (SLA) that identifies key metrics (service levels) that the customer can reasonably expect from the service. The ability to understand and to fully trust the availability, reliability, and performance of the cloud is the key conceptual block for many technologists interested in moving into the cloud.

Availability:

Availability describes how often a service can be used over a defined period of time. For example, if a website is accessible to the general public for 710 hours out of a 720-hour month, we say it has a 98.6% availability rating for that month.

If, for example, Google's spider is down for 24 hours but you can still search and get results, would you consider Google to be down?

Most people consider a system to have high availability if it has a documented expectation of 99.99% to 99.999% availability. At 99.999% availability, the system can be inaccessible for at most five minutes and 15 seconds over the course of an entire year.

How to estimate the availability of your system:

Determining expected availability thus involves two variables:

• The likelihood you will encounter a failure in the system during the measurement period.

• How much downtime you would expect in the event the system fails.

The mathematic formulation of the availability of a component is:

$$a = (p - (c \times d))/p$$

where: a = expected availability

c = the % of likelihood that you will encounter a server loss in a given period

d = expected downtime from the loss of the server

p = the measurement period

So, if your 486 has a 40% chance of failure and you will be down for 24 hours, your 486 uptime is (8760 - (40%×24))/8760, or just shy of 99.9%.

The availability of a system is the total time of a period minus the sum of all expected downtime during that period, all divided by the total time in the period:

$$a = (p - \mathrm{SUM}(c1 \times d1 : cn \times \mathrm{dn}))/p$$

If your cable provider generally experiences two outages each year lasting two hours each, the Internet connection's availability is:

(8760 – (200%×2))/8760 = 99.95%

Thus, your overall availability is:

(8760 - ((40%×24) + (200%×2)))/8760 = 99.84%

When you have two or more physical components representing a logical component, the expected downtime of the logical component is the expected amount of time you would expect all of the physical components to be down simultaneously. In other words, the $c \times d$ formula for downtime calculation becomes slightly more complex:

$$(C \times d^{n})/ (p^{(n-1)})$$

where n is the level of redundancy in the system. As a result, when n = 1, the formula reduces as you would expect to its simpler form:

$(c \times d^{n})/(p^{(n-1)}) = (c \times d)/(p0) = c \times d$

The redundancy of two 486 boxes—and this requires seamless failover to be considered one logical component—now provides a much better availability rating:

$(8760 - ((40\% \times (24^{2}))/(8760^{(2-1)}))/8760 = 99.999\%$

What constitutes availability?

In particular, you need to define the following availability criteria:

• What features must be accessible in order for the system to qualify as available? In the Google example earlier, we would consider Google to be available as long as you can search the site and get results; what the spider is doing is irrelevant.

• Should you include planned downtime? For example, if your architecture is fully capable of supporting 99.999% availability but you prefer to take the environment down 1 hour each week for system maintenance, you could advertise the environment as having 99.999% availability. If, however, you are trying to communicate to end users what they should expect in the way of uptime, saying such an environment has 99.999% availability is misleading.

• What percentage of the time will your environment remain available?

A website, for example, might state its availability requirements as the home page and all child pages will be accessible from outside the corporate network between the hours of 7 a.m. and 9 p.m. 99.999% of the time.

Cloud service availability:

Perhaps the most jarring idea is that many failures you might consider rare in a traditional data center are common in the cloud. This apparent lack of reliability is balanced by the fact that many of the failures you might consider catastrophic are mundane in the cloud.

An EC2 instance is utterly unreliable when compared to the expected availability of a low-end server with component redundancies. It is very rare to lose such a physical server with no warning at all. Instead, one component will typically fail (or even warn you that it is about to fail) and is replaced by a redundant component that enables you to recover without any downtime. In many other clouds, such as Rackspace and GoGrid, you see the same levels of reliability. In the Amazon cloud, however, your instances will eventually fail without warning. It is a 100% guaranteed certainty.

Amazon Web Services service levels:

Most competitors offer strong service levels in the cloud. Although Amazon has provided its customers with an SLA for S3 for some time, it has only recently added a

formal SLA to EC2. The S3 service level promises that S3 will respond to service requests 99.5% of the time in each calendar month. EC2, on the other hand, defines a more complex availability service level. In particular, EC2 promises 99.95% availability of at least two availability zones within a region.

• You need S3 to be available to launch an EC2 instance. If S3 has a 99.5% availability, you will be able to launch new EC2 instances only 99.5% of the time, regardless of how well EC2 is living up to—or exceeding—its promises. This drawback also applies to snapshots and creating volumes because you cannot take a snapshot or create a volume from a snapshot if S3 is down.
• EC2 is living up to its service level as long as two availability zones in the same region are available 99.95% of the time. EC2 can technically live up to this service level with entire availability zones constantly going down.
• You need to architect your application to be able to reliably support the demands on it.

Expected availability in the cloud:

The key differentiator between downtime in the cloud and downtime in a physical environment lies in how much easier it is to create an infrastructure that will recover rapidly when something negative happens.

Let's compare a very simple setup that includes a single load balancer, two application servers, and a database engine. When implementing this architecture in a physical data center, you will typically locate this infrastructure in a single rack at a hosting provider using fairly mundane servers with component redundancy and hardware load balancing. Ignoring network failure rates and the possibility of a data center catastrophe, you will probably achieve availability that looks like these calculations:

Load balancer

99.999% (people use hardware load balancers partly to minimize component failure)

Application server

$(8760 - ((30\% \times (24^2))/8760))/8760 = 99.999\%$

Database server

$(8760 - (24 \times 30\%))/8760 = 99.92\%$

Overall system

$(8760 - ((24 \times 30\%) + (24 \times (((30\% \times (24^2))/8760)) + (24 \times 30\%)))/8760 = 99.84\%$

In the cloud, the calculation is quite different. The load balancer is simply an individual instance, and individual server instances are much less reliable. Furthermore, the downtime for these nodes is much less:

Load balancer
(8760 - (.17×80%))/8760 = 99.998%
Application server
(8760 - (17%×((.17^2)/8760)))/8760 = 99.9999%
Database server
(8760 - (.5×80%))/8760 = 99.995%
Overall system
(8760 - ((.17×80%) + (17%×((.17^2)/8760)) + (.5×80%)))/8760 = 99.994%
This calculation assumes you are using tools to perform automated recovery of your cloud infrastructure.

Reliability:

Reliability is often related to availability, but it's a slightly different concept. Specifically, reliability refers to how well you can trust a system to protect data integrity and execute its transactions.

A system that is frequently not available is clearly not reliable. A highly available system, however, can still be unreliable if you don't trust the data it presents.

Much of the reliability of your system depends on how you write the code that runs it. The cloud presents a few issues outside the scope of your application code that can impact your system's reliability. Within the cloud, the most significant of these issues is how you manage persistent data.

Because virtual instances tend to have lower availability than their physical counterparts, the chance for data corruption is higher in the cloud than it is in a traditional data center. In particular, any time you lose a server, the following factors become real concerns:

- You will lose any data stored on that instance that ha not been backed up somewhere.
- Your block storage devices have a chance of becoming corrupted

For now, you should remember two rules of thumb for dealing with reliability concerns in the cloud:

• Don't store EC2 persistent data in an instance's ephemeral mounts.
• Snapshot your block volumes regularly.

Performance

When performing a high-performance transactional application for deployment in a physical data center applies to deployment in the cloud. Standard best practices apply:
• Design your application so logic can be spread across multiple servers.
• If you are not clustering your database, segment database access so database reads can run against slaves while writes execute against the master.
• Leverage the threading and/or process forking capabilities of your underlying platform to take advantage of as much of each individual CPU core as possible.

Clustering versus independent nodes:

Depending on the nature of your application, your choke points may be at the application server layer or the database.
• Use a load balancer to automatically split sessions across an indeterminate number of independent nodes.
• Use a load balancer to route traffic to nodes within a clustered application server. Through clustering, application server nodes communicate with each other to share state information.

Clustering has the advantage of enabling you to keep state information within the application server tier; it has the disadvantage of being more complex and ultimately limiting your long-term scalability. Another challenge with true clustering is that many clustering architectures rely on multicasting.

EC2 performance constraints:

When you select a particular EC2 machine, the CPU power and RAM execute more or less as you would expect from a physical server. Network speeds are outstanding. Differences show up in network and disk I/O performance.
The three different kinds of data storage have three very different performance profiles:

• Block storage has exactly the kind of performance you would expect for a SAN with other applications competing for its use across a GB Ethernet connection. It has the most reliable performance of any of the options, and that performance is reliably solid.
• S3 is slow, slow, and slow (relatively speaking). Applications should not rely on S3 for real time access.
• Local storage is entirely unpredictable.