Things change fast in the AWS ecosystem. If you want to keep up to date on new features, products, functionality, and pricing I suggest that you follow the Amazon Web Services blog: http://aws.typepad.com/. This is one of the best places to get up to the second information about the AWS universe.

Who Is Deploying SQL Server in EC2?

A question I hear a lot is, “Who is putting SQL Server into EC2?” Sometimes there’s a hint of incredulity in that question, hinting that people can’t seriously be deploying SQL Server into EC2. That’s far from the truth: many companies, large and small, are deploying SQL Server into Amazon’s cloud, with varying degrees of success.

Back to the question of who’s deploying SQL Server in EC2…

I work with a start up building a platform on a Microsoft stack. The front end is hosted on mobile devices, the middle tier is written in .NET, and the database is SQL Server. They’re just starting out and they don’t have the time to spend specing out servers, ordering them, and waiting for delivery. It’s easy to spin up servers to test new features and functionality and if a feature doesn’t catch on the servers are turned back off.

Traditional hosting facilities didn’t offer the flexibility or response times that they were looking for, so they chose Amazon EC2.

Another company I work with is a ISV transitioning to a Software as a Service (SaaS) model. The ISV works in an industry where there is a lot of specialized knowledge and technical ability, but that knowledge and ability doesn’t have anything to do with keeping a server up and running. In the past they’ve added additional maintenance routines to the software that they sell to customers, but sometimes customers don’t have anywhere to host this mission critical software. The ISV has hosted a few customers on their own servers, but they don’t want to keep buying new hardware in response to customer demands – what happens when a...
Look out!

There are many things to worry about, here are a few:

1. Costs
With AWS, you get billed for the resources you use. When the VM is on, you’re getting charged. When your storage is provisioned, you’re getting charged.

2. Noisy Neighbors
AWS is built around virtualization. Be careful, other VMs could be using more than their share of resources.

3. Failures
Equipment failures are inevitable. You must plan for them.

4. Drive limits
There is a finite limit on the amount of storage a single server can have. Plan accordingly and split the load.

5. Bandwidth limits
You can use ten gigabit ethernet locally, but Windows instances in AWS are limited to one gigabit connectivity.

6. Instance Sizes
There’s a hard limit on the biggest size of an EC2 instance. You can’t get a 64 core monster server in AWS.

How much?
You can use the AWS calculator to figure out just how much your existing infrastructure will cost to move to the cloud.

What Are Some Of The Problems?

There are, of course, problems with every way that you can possibly deploy SQL Server. When you deploy on premises, you have to purchase hardware, wait for it to arrive, set it up, and then you have to maintain it. When you host your database with another hosting provider, you’re limited by the amount of server space they have available. When you host in EC2 there are a different set of problems.

Cost

One of the perceived problems with deploying into the cloud is the cost. You’re renting something by the hour (like that “friend” your uncle brought to Thanksgiving last year); and renting something by the hour can get really expensive when you want that something to be up and running all the time.

How much would it cost to keep a reasonable SQL Server running 365 days a year at $10,862.40 assuming that you’re using a Double Extra Large Hi-Memory instance (that’s 4 x 2.66 GHz Xeons and 68.4GB of memory) for twice the price. Yeah, that’s expensive. Of course, you can also reserve the instance and just buy it outright for a fraction of that cost, but who wants to do that?

What would a similar server cost you from HP? You can get an HP ProLiant DL380 G7 with a pair of fancy pants Intel E5640 CPUs (that’s right around the same speed as the EC2 instance), 32GB of RAM, and 1.2TB of in chassis storage for about $8,600. That price doesn’t include an OS or any other licensing costs. It also doesn’t include power, cooling, or even a spare server sitting around ready to pick up the load if your primary server fails.

Storage isn’t tremendously expensive in Amazon’s cloud – 10 cents per GB per month of provisioned storage. Over the course of a year, 1 terabyte of storage is only $102.40 a month, and that storage is fairly redundant within a single data center.

Despite the cost, Amazon’s cloud is still incredibly popular with many businesses. Why? Simple: it’s easy to start with a small amount of resources and rapidly scale in response to customer demands. If something doesn’t pan out, you just stop the servers and delete the storage and you don’t have to worry about it anymore. The costs go away as soon as you stop using the compute cycles.

Noisy Neighbors

I used to live in an apartment. It wasn’t a great apartment. In fact, it wasn’t even a nice apartment. I could hear everything that my neighbors did. After a while, I knew about

customer leaves? The ISV hasn’t made the move to cloud computing yet, they’re feeling out the different options available, but EC2 provides a level of flexibility that they won’t get through traditional IT solutions.
You may not be alone...

their girlfriends, their love of Super Tecmo Bowl, and I learned that they liked to listen to loud music on week nights when they didn’t have to work the next day.

My noisy neighbors made it difficult for me to get things done. When you’re sharing a host, noisy neighbors can make it difficult to get things done, too. This sort of thing can happen on a server right now – when one program is aggressive writing to disk, read performance will suffer. If you’re sharing a physical server with another guest operating system, you have no idea what that other OS is doing. It could be sitting there doing nothing, it could be chugging along doing 3d rendering, or it could be a BitTorrent server. You have no idea what’s going on in the next room and how it could be affecting your performance.

Unfortunately, there’s not a lot that you can do about noisy neighbors apart from moving. In the real world, you move to a new apartment or buy a house. It takes time and money to get a new place, but it’s feasible. In EC2 it’s a lot easier to get a new place: you just upgrade your instance. You pay more by the hour, but it’s easy to get a bigger place.

Crashes In General

To be fair, you can’t do much about crashes if you’re using someone else’s hosting service; when the data center loses power your server is going to go down. So, how do you protect yourself from crashes?

Redundancy, of course.

Unfortunately, redundancy gets expensive fast. If you want a redundant mirror for that 10k a year server, you need a second 10k a year server. If you want multiple servers spread across the country, you’re paying for each one. Thankfully the cost of the servers includes the Windows license, but it’s still a large cost for many people to stomach.

SQL Server has a number of options to help you keep your servers up and running in case something horrible happens. Traditional database mirroring will work in EC2. You are free to implement whatever Rube Goldberg machine you want. It’s important to keep in mind that you have little control over your storage (there are no SAN snapshots or SAN mirroring that you can control) and there is no dedicated network that you can rely on. All of your connectivity goes over the open infrastructure within Amazon’s data centers. You’re fighting for bandwidth with Amazon, Netflix, Heroku, and everyone else in the cloud.

Limited Number Of Drives

How many drives can you connect to an installation of Windows locally? 26 or so, right? That’s how many letters there are in the alphabet. You can go beyond that using mount points to boost the amount of storage you have. For most purposes, though, you effectively can attach an unlimited number of drives to a Windows machine. Certainly more than you’ll need. SANs make it easy to create huge volumes, after all.

In EC2, that’s not quite the case. You can only attach 16 drives (Amazon call them volumes) to a server. It’s important to keep this limitation in mind when designing your strategy to move into Amazon’s cloud. If you need to store more than 16 terabytes of data, you’ll need to find a way to spread that data out across multiple EC2 instances. Considering some of what we’ll cover later on, this is going to be a really good idea.

Limited Network Throughput

Did you know that there’s a pretty solid limit on network throughput in EC2? I didn’t either until I start performing some storage benchmarks (more on this soon). It turns out that there’s a single active gigabit connection into each host. That’s right: a gigabit connection per host. You might be sharing that server with a lot of other servers. If there are 4 other SQL Servers sharing the same gigabit connection, you could end up with some very poor performance. Amazon recently added instances with 10 gigabit network cards, but they’re limited to Linux machines in the US-East region right now. In the future there might be instances with 10 gig connections that run Windows, but there’s nothing in EC2 right now to make the path between a Windows server and the storage any faster. To keep up to date on which instances have 10 gigabit ethernet, visit the list of EC2 Instance Types and search for “I/O Performance: Very High”.

In short, if your applications are chatty, you’re going to need to learn to break them apart into many smaller pieces to keep throughput high.

Limited Instance Sizes

We just briefly touched on this in the last section – not every size of instance is available in every region. Typically, the US-East region gets the newest fanciest servers, at least as far as I can tell. Other regions slowly get new hardware over time. The name of the instance type (m1.small, m2.4xlarge) will stay the same over time, but it may not be
Terminology:

Volupat est ipsum quis est eu pede laoreet elementum lectus

AWS
Amazon Web Services. This is the overarching banner of Amazon’s cloud offering.

EC2
Elastic Compute Cloud. This is where you create and administer virtual machines.

EBS
Elastic Block Storage. Your SAN in the sky.

S3
Simple Storage Service. Cheap, durable, and slow. It’s a great place to keep data around for a long time.

Region
AWS is split into geographic regions (US East, US West, EU, Asia Singapore, Asia Tokyo).

Availability Zone
Availability Zones are data centers inside of regions. Availability Zones have independent power and connectivity and should be used for failover.

Storage Performance

If you want to keep data around for any length of time, it’s important to persist it somewhere. Persisting your data means that you have to write it and, if you aren’t writing it to your users’ computers you have to write it to the database. Hiding beneath all of that fancy database software is something as simple as a pile of disks for storing data in rows and tables. Making sure the disks are working as fast as they can is critical to keeping a database running at peak performance.

Server Performance

Just like any other computer, there are a finite number of resources available on any host. Likewise, the VMs that you spin up have a finite amount of resources available to them. Unlike your local data center, you have no control over which VMs share the same host.

Normally you could put different machines together and stagger their functionality around the clock so that guests that did bulk processing at night would be on the same host as machines that record orders during business hours. When everything is under your control, it’s very easy to make sure that any host isn’t overcommitted during a certain time window.

The only way that you can guarantee performance is to allocate larger instances that take up all of the physical resources of the host but cost more. By doing this you’re able to eliminate the effect of noisy neighbors. It’s important to understand how to get the best performance out of your resources.

Just How Fast is Gigabit Ethernet?

The question isn’t rhetorical. It’s important to understand how fast gigabit ethernet is. Gigabit ethernet can really only transfer, at most, 120 megabytes per second. That’s the theoretical maximum assuming that everything along the line is operating perfectly and there is no latency or additional hops to make.

We’re not likely to see this in the real world. You’re not likely to see this in your own server room and you’re sure as hell not going to see it in EC2.

Why is your storage going to be slow? You have keep in mind that not everything involved may be able to push data at 120 MB/s; the drives could be slow, fragment, or you could be sharing space with someone else who is doing a lot of reading and writing. Other people could be reading from the same host as you or they could be streaming movies from the host to a customer in the suburbs.

In an attempt to find an answer, I conducted several tests on an EC2 instance over the course of several days to answer the question, “Just how fast is gigabit ethernet?”

Storage Performance

Gigabit ethernet sucks. If disk I/O is your absolute bottleneck, you only have two options – you need to write to as many drives as possible. We’ll find out why that isn’t entirely possible in EC2. The other option is to make the connection between your database server and the storage as fast as possible. Since you can’t get super fast storage in EC2, you’ll have to find another way to make your database fast.
EC2 Storage Test Results:

**CrystalDiskMark: Quick Format, Reads**

<table>
<thead>
<tr>
<th>Read: Sequential</th>
<th>Read: 512K</th>
<th>Read: 4K</th>
<th>Read: 4K QD32</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 EBS</td>
<td>2 EBS</td>
<td>3 EBS</td>
<td>4 EBS</td>
</tr>
</tbody>
</table>

![Read Performance (MB/s)](image)

Right away, we can see that performance quickly spikes for sequential reads and writes. While the results for read decrease, this isn’t a reflect of the storage itself so much as it is a reflection of trying to cram a lot of data through a tiny ethernet pipe. Overall, the quick formatted drives don’t show a lot of performance improvement no matter how many spindles we throw at the problem.

**Typical Performance**

A typical, server grade hard drive is very different from a typical commodity hard drive.

A Fujitsu MBE2 RC series drive will, on average, be able to perform around 980 IOPS when streaming writes and around 990 streaming reads. To put that into perspective, it’s around 252 MB/s. This is a typical server grade rotational hard disk.

A consumer grade SSD, on the other hand, can perform around 360 MB/s for sequential writes and 441 MB/s for sequential reads. At these speeds, the SSD max out at 49,000 IOPs.

EBS volumes, by contrast, perform similarly to the drives you find in an average desktop computer.

As with many things, the more money you spend on hardware, the better the performance you will see.

**Formatting Drives**

Typically you ask your SAN admin to provide 140GB of RAID 10 and you call it a day. In EC2 you need to make sure that you use the correct block size (64k) as well as striping the drives yourself. The only RAID level available is RAID 0. If you need redundancy, you’ll need to engineer it somewhere outside of the storage tier. Thankfully, SQL Server has many features that make it easy to easily provide the redundancy and availability that we’re used to.

**Testing Methods**

For all of the tests, I used the same m2.4xlarge instance. I installed it from a standard Windows Server 2008R2 SP1 AMI that Amazon supply. Nothing was changed with the operating system. This Windows was about as vanilla as it gets.

For my tests with the drives, I used EBS volumes that were allocated in complete 1TB chunks. I made two rounds of tests. In both cases, the drives were formatted with a 64k sector size. The first round of tests was done with drives that were quick formatted. The second round of tests were performed with drives that were not quick formatted. FYI, formatting 10TB of drives in EC2 takes about 2 days. I used a Large Windows instance to format the drives at a cost of about $26 in CPU time plus another $57.20 in I/O cost just to format the drives.

The first set of tests I ran involved using CrystalDiskMark. I used the default configuration and ran it across the quick formatted drives. Drives were configured with 1 EBS volume, 2 EBS volumes, 3 EBS volumes, and 4 EBS volumes. I conducted a second set of CrystalDiskMark tests using fully formatted drives with 1, 2, 3, 4, and 10 EBS volumes in a single drive.

The second set of tests was more complex and involved using SQLIO to simulate sequential and random reads from SQL Server over a period of time. All tests were performed on a single EBS volume, two striped EBS volumes, four striped EBS volumes, and a pair of striped four EBS volumes.

Each round of SQLIO tests ran for approximately 20 minutes per test and used a 20GB file.
The First Touch Penalty

Amazon mention in their documentation that there is a penalty, and potentially a large one, for the first write to a block of storage. When you look at the numbers below you'll realize that they weren’t kidding. Amazon further go on to claim that if you benchmark the storage (as I did) you'll be horrified by the results (as I was) but that you shouldn't worry because the first write penalty is only really obvious during benchmarking and will be amortized away for time in the real world.

If your workload consists almost entirely of writing new data, you’re going to see a first touch penalty until the drive is full. It won’t end. Writes will suck until your drive is full. Then you’ll add more drives and the suck will continue. This is compounded for relational databases where the currently accepted best practice is to ensure sequential writes by clustering on an arbitrarily increasing integer.

The only way to guarantee acceptable performance is to format your drives and write to them block by agonizing block until the storage is fully formatted.

The other alternative is to design your data storage so that you minimize the effects of the first touch penalty by aggregating writes across several servers and using intermediate caching layers and asynchronous writes to ensure that client applications are not aware of any first touch performance problems.

EC2 Storage Test Results:
CrystalDiskMark: Quick Format, Writes

Write: Sequential
Write: 512K
Write: 4K
Write: 4K QD32

Queue Depth

Queue depth is a setting that allows the hard disk controller to queue up reads and writes and, potentially, re-order them if it determines that there is a faster way to perform the instructions in the queue. By default the queue depth is set to 1 - all reads and writes happen in the order they arrive. Changing the queue depth affects read performance, write performance, and data durability. You never know if that storage device in the cloud has any kind of battery backed cache.

Thin Provisioning

A common technique in virtualized environments, as well as enterprise grade SANs, is to use something called thin provisioning. Rather than allocate all of the requested space at once, the storage device only marks the space as requested but it doesn’t actually write anything to the disks until it’s actually told to do so. This is a good thing.

Long ago, in the dark ages, we had to format our own hard drives. This was an agonizing process that usually involved putting a manufacturer’s floppy disk (or CD) into the computer, running a specialized utility, and then finding something to do for the next 8 hours. Thin provisioning gets around this.

It takes a long time to format a hard disk and sequentially write a large number of 0s to it. Thin provisioning and quick formatting can be useful - it speeds up your ability to get your server off the ground and handling requests from users.
EC2 Storage Test Results:

CrystalDiskMark: Full Format

First Thoughts

There are some things that we can conclude about the performance of disks in Amazon’s platform. First and foremost: format your drives. Don’t just use a quick format; use the slow format. The downside is that it takes time to format disks and time is money, especially in the cloud. By using a full format, disk performance improved by 30% for 4K writes with a queue depth of 32 and improved, on the whole, by around 15%.

It Isn’t A Race Car

But you didn’t expect it to be one, did you?

After going through the CrystalDiskMark tests, I decided to be a bit more scientific about the process and use SQLIO to use the disks in a way that’s more consistent with how SQL Server would operate.

You could make assumptions about how the storage is going to perform when we look at SQLIO; the actual results may surprise you.

Disk performance improved, overall, by 15 per cent

There were a few anomalies – specifically the 4k write performance spike with 4 EBS volumes and the sequential read performance spike for 2 and 4 EBS volumes. The only conclusion I can come to is that reads were cached and did not hit disk.

A full format consists of writing a lot of zeroes to the drive. This isn’t for the faint of heart, pressed for time, or weak of budget.

It’s also a fast way to consistently good disk performance in AWS. If there were a better way, I’d tell you.
The SQLIO tests show a different story. Right away we can see that sequential I/O performance peaks with a maximum of two volumes, but random reads and writes keep scaling right up until the tests stop – the more spindles that were used, the more random I/O that we could perform. And here’s where things get interesting again. Looking at the graphs you can see that sequential I/O falls apart a little bit north of 1000 IOPS. That’s not a limit of the underlying disks, we’re just maxing out the ethernet connection. When there’s a finite pipe limiting how fast we can read, of course the number of IOPS is going to stop pretty quickly.

Looking at the throughput in terms of the amount of data (rather than the number of reads and writes) that we can move through EC2, it’s very easy to see where we hit the ceiling. With a theoretical maximum of 120 MB/s, it’s easy to see that sequential read and write max out when two volumes are use. It’s clear, though, that random reads and writes continue to scale as we add more drives. I suspect that if I had been able to add more spindles, the random write performance would have continued to scale up to 120 MB/s.

What Does This Mean For SQL Server In AWS?

You can get the random I/O performance that you want, but sequential read/write performance is not going to be good. Taking that a step further, here’s what you don’t want to do in EC2:

• Data warehousing
• Large ETL
• Bulk data processing that requires sequential reads
• Anything else that requires a lot of sequential read and write

On the flip side, what should you use SQL Server in EC2 for?

• OLTP
• Off site processing that isn’t time sensitive
• Distributed write loads (sharding)

SQL Server hosted in EC2 is going to be very effective for highly random loads and especially for loads where data can be kept in memory (currently less than 68.4 GB).
Ephemeral Storage

Fast, Temporary Instance Level Storage

Don’t use it. It goes away and may never come back. Well, it goes away when your instance reboots.

Because it’s the drives that live in your server chassis. You can’t predict where your server is going to be when you turn it back on again, so you can’t predict if that data is going to stay with you or not. This is a best practice for virtualization, if you think about it. When you’re setting things up using VMware or Hyper-V you don’t want to attach long-term storage that’s in the server chassis, it makes your virtualization more brittle. Using local storage, if anything happens (a motherboard fails, a drive fails, a CPU fan goes out, or memory fails) you’ve lost that host. You can’t bring that VM up on another server because the OS is still on hard drives in that broken server.

Local storage is great for temporary (ephemeral) things. Make sure that you take that into account when you’re designing your infrastructure. Fast, cheap, temporary storage is great for fast, cheap, temporary operations like TempDB or local scratch disks.

High Availability Options

Clustering, The Sad Trombone Of The Cloud

You can’t cluster in EC2. Don’t try. There’s no shared storage. EBS volumes can only be attached to a single server at a time. You could share the drives, but you really shouldn’t do that. You could use some Windows based SAN system that mounts drives and pretends it’s a SAN, but without any guarantees of performance or longevity, why would you want to?

We’ll just leave this one as: You can’t cluster in EC2.

Availability Zones And Regions

In case you didn’t know, Amazon’s magical cloud is divided up into regions. Each region is made up of multiple availability zones. They’re all right next door to each other, but they’re in separate buildings with separate internet access and separate power.

Availability zones exist to protect users from localized failure. Hopefully, if there’s a problem in a data center, it’s only going to be a problem in one data center (one availability zone). If that zone goes down, the others should be up and running. If you want to keep your server up and running through local failures, you need to make sure you keep a copy of your data in multiple availability zones. If you want to keep your business up and running in the event of a regional failure (the smoking crater scenario), you need to keep copies of your data in multiple regions.

- Region – national level.
- Availability zone – local level.

Mirroring SQL Server In EC2

One of the easier ways to keep things up and running in the event of a failure is to use SQL Server’s mirroring. Using synchronous mirroring with a witness is going to put you in a great place in terms of disaster recovery. It’s both possible and advisable to put the primary in one zone, the secondary in another, and the witness in a third zone. We can feasibly survive some pretty heavy duty hardship by configuring SQL Server mirroring in multiple availability zones. The applications talking to SQL Server need to be able to fail over between the mirrors but that should be pretty easy for today’s modern developers to handle, right guys?

SQL Server Replication In EC2

Replication is always an option and is probably a better option when you want some kind of geographic fail over. Instead of diving into contentious issues around designing a replication topology, I’m going to hit on the pros and cons of replication in the cloud.

We’ve already seen that there significant limits to network throughput in EC2. We’re limited to, at most, a gigabit of storage throughput. We’re also limited by our ability to write to disk in 64k chunks – only a little bit.

Replication

SQL Server replication uses a pub-sub model. The publishing database asynchronously sends transactions to a subscriber. Replication uses a distributor database, and all three (pub, sub, distributor) need to be operational for replication to work.
more than 1000 IOPS are possible at a 64k write size. These limitations become a problem for transactional replication. In an on premise set up, transaction log performance is frequently a problem when replication is involved. If you think that the performance of the transaction log won’t be an issue in the cloud, think again. The I/O characteristics of Elastic Block Storage guarantee that the performance of the transaction log will become a problem under heavy load.

Replication latency is another concern for DBAs in EC2. Not only do reads and writes occur slower, but that read/write slowness means that transactions will be committed to the subscriber further and further behind the publisher. Of course most applications aren’t writing constantly, but it’s important to consider what might happen if the subscriber isn’t able to commit the load that’s being written throughout the business day - will it be able to catch up after hours? If it can’t catch up today, will it be able to catch up tomorrow?

Crashes happen in EC2, entire regions have gone offline. Unfortunately, at any kind of scale something is bound to be broken right now. What do you do when that happens? What do you do when that happens to the database? (You are running the distributor on a separate server, right?) What do you do when the distributor goes offline and never comes back up, its drives corrupted irreparably?

I don’t have the answers, but that’s something your recovery model must take into account when you’re deploying into the cloud.

**Backups In The Cloud**

You need to keep your data safe. Taking backups and hoping for the best isn’t going to cut it once you’ve moved all of your data up into the cloud. You need a retention and recovery strategy.

**EBS Data Protection**

Amazon’s Elastic Block Storage (EBS) looks like some kind of SAN drives that you can attach to your server. When you look more closely at the disks themselves, they look like drives attached to some kind of storage device powered by Red Hat Enterprise Linux. I’m overusing the phrase “some kind” because you just don’t know what’s back there.

There are, however, some certainties around EBS. An EBS volume lives within a single availability zone. There are multiple drives involved and multiple block level copies of your data. There could be a failure of a back end drive and you’ll never know about it.

The downside of EBS is that there’s no global availability of your data. There is no SAN replication, no global single back up of your data. If you want that, you’ll need to look into using Amazon S3 to provide greater safety.

**Be Safe With S3**

Amazon Simple Storage Service is a very simple storage service indeed. It’s a block level data store that refers to big binary chunks of data by name. It’s like a filesystem that exists everywhere. Sort of. The important part of S3 is that your data can be persisted at a very paranoid level over a very long period of time. In their own words S3 is “Designed to provide 99.999999% durability and 99.99% availability of objects over a given year.”

Why mention S3 at all? It’s not very fast and it’s not very easy to mount S3 storage to Windows. There are two things you can do with S3:

1. Store SQL Server backups.
2. Snapshot EBS volumes.

Storing SQL Server backups in S3 should be a no brainer for anyone.

S3 storage is cheaper than EBS storage and it’s far more durable. Since S3 is global, it’s also possible to push data up to S3 from one region and pull it down in another. Your servers will live on.

S3 can also be used to store snapshots of EBS volumes. Don’t be fooled by the word “snapshot”. The first time you take a snapshot you will likely make a very bad joke about how this isn’t a snapshot, it’s an oil painting. Don’t do that.

The first EBS snapshot to S3 is going to be slow because there is a block level copy of the entire EBS volume happening in the background. All of your data is zipping off into the internet to be safely stored in a bucket somewhere. The upside is that only the blocks of data that get changed are going to get snapshotted. If you perform frequent snapshots then very little data will need to be sent to S3, snapshots will be faster, and recovery should be a breeze. Keep in mind, of course, that these are not crash safe SAN snapshots, so they shouldn’t be used for live databases. I’m sure you can think of cunning ways to work with this.

**A Plan For Safe & Secure Backups In AWS**

What’s the safe way to manage backups in EC2? Carefully.

Were it up to me, here’s what you’d do:
1. Backup your data to an EBS volume.
2. Mount an S3 drive to your Server.
3. Copy the data to the S3 drive using xcopy with the /J flag

Of course, if it really were up to me, you’d have mirroring set up with a primary in one availability zone, a secondary in another, and the witness in a third. Oh, and you’d be using synchronous mirroring.

**Buffered Copy**

When you copy a file, there are a few different things that can happen. In an unbuffered copy, Windows will stream the bytes from one location to another. This is quick and painless. The other, more common, option is that Windows will perform a buffered copy. OS memory will be used and may not be released. This will cause memory pressure and poor SQL Server performance.

**If you can’t scale up, the only thing left is to scale out.**

**Scaling SQL Server In EC2**

**Scaling Up**

This is, after all, why we’re here, right? You’re not just reading this article because you want to learn about database backup best practices and how many IOPS some magical storage device can perform. You want to know how to give your SQL Server some more zip in EC2. There’s a finite limit to how much you can scale up SQL Server in EC2. That limit is currently 8 cores and 68.4GB of RAM. Those 8 cores currently are Intel Xeon X5550s that clock in at 2.66 GHz, but that might change by the time you’re reading this.

In EC2, there’s a limitation on how much we can scale up. Commodity hardware really does mean commodity hardware in this case. If you can’t scale up, the only thing left is to scale out.
Scaling Out

It’s incredibly easy to create more instances of SQL Server in EC2. Within a few clicks, you’re most of the way there. You can automate the process further using Amazon’s developer tools to script out additional configuration, attach local storage, and attach EBS storage. The developer tools make it a breeze to create and deploy your own custom images so you have pre-configured instances up and running. You can even use tools like Puppet or Chef to completely automate adding more servers to your environment.

In Conclusion

So, there you have it – it’s possible to get great performance for certain types of SQL Server operations in EC2. If you’re looking for a cheap way to set up a data warehouse, you should look elsewhere. You might be able to slap something together in EC2 that performs well enough, but there will be a lot of man power involved in making that solution work. If you’re looking for a quick way to scale your application up and out without worrying about buying hardware and making sure it’s being used appropriately, then you’ve come to the right place.