Realtime publishers

The Definitive Guide™ To

Windows Application and Server Backup 2.0

sponsored by

AppAssure
HOME OF BACKUP 2.0

Don Jones

Copyright Statement

Chapter 9: Keeping Your Backups—Storage Architecture

Storage has long been a difficult companion for backups. The first backups were stacks of punched cards, although magnetic tape quickly came onto the scene to store more data and permit somewhat faster recovery. Ever since, we've struggled with where to put our backups. Buying a new terabyte file server inevitably meant buying another terabyte of backup capacity; some companies used—and still use—file filtering technologies to reduce the amount of data on their file servers, primarily to help control the amount of data that has to be backed up.

That's the ugly thing that happens when technology—and its limitations—start to drive the business rather than the business driving the technology. Sure, keeping errant MP3 files off your file servers might be a good idea for any number of reasons, but in general shouldn't users be able to put *any* business-related data onto a file server without worrying that it might not be backed up? Isn't *all* our business data worth backing up?

This is where storage comes into the Backup 2.0 picture. For all the great things that Backup 2.0 can do in terms of backing up our data and allowing fast and flexible restore operations, it's useless if it needs more space than we can give it.

Where Backups Go—Storage in Backup 1.0

Let's consider backup storage in the 1.0 world, which traditionally involved magnetic tape. Today, even Backup 1.0 solutions are a bit more flexible, allowing multiple forms of storage and even a hierarchy of storage. Is the 1.0 concept of backup storage suitable for a 2.0 world?

What Backed-Up Data Looks Like

In a Backup 1.0 world, backed-up data reflects the intrinsic nature of the backup itself. In other words, think *snapshots.* A typical backup file consists of a single file, or perhaps a collection of related files, that contain data streamed from a source. Figure 9.1 illustrates this process—which you probably watch happen so often that you've stopped thinking about it.

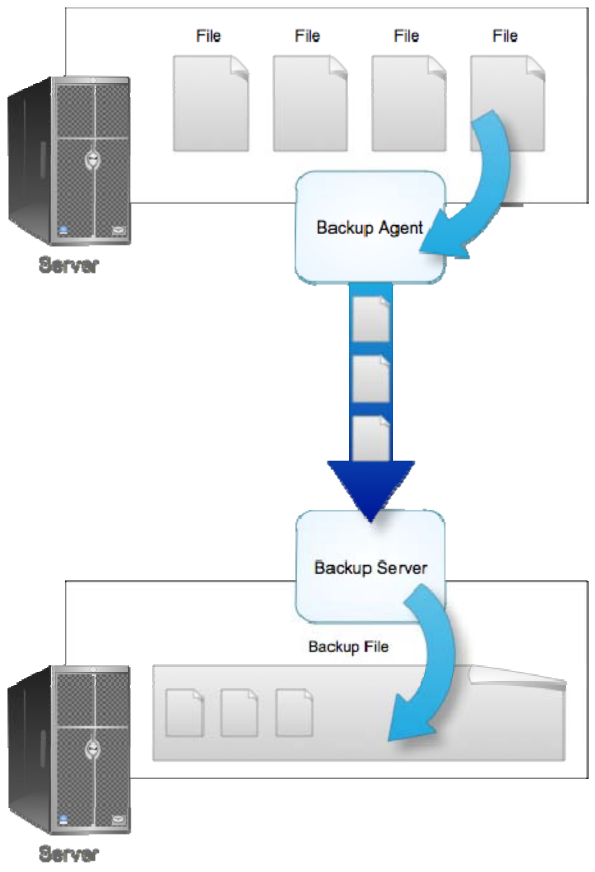


Figure 9.1: Traditional backup process.

So there are really two problems here: First, we're backing up data as it exists at a specific point in time—making a snapshot, in other words, which is the intrinsic problem with Backup 1.0. The other problem relates to how we're storing the data: We're essentially dumping everything into one logical structure. Even though that may be split across multiple physical files, it's a single, usually flat, logical entity. There's a reason for that, and it's called *magnetic tape.* Magnetic tape is a sequential-access storage device; it doesn't support random access the way a hard disk does. Because most backups are written to tape, it makes sense to store them in a single structure that can be sequentially written to, and read from, magnetic tape. And the problem with sequential access—and magnetic tape—is that it's incredibly slow compared with random access.

When Is Sequential Access Fast?

One area where you regularly use sequential access and get really fast access speeds is with optical media—CD-ROMs, DVD-ROMs, and so forth. Those discs are written much like an old vinyl record, with a single track, and the laser has to follow that single path sequentially to get to whatever data you want. The reason it's so fast is simply that the disc is spinning really, really fast (around 1600 RPM for a DVD)—so the laser can move from the outermost area of the disc to the innermost in a flash.

Tape, unfortunately, doesn't support that kind of speed. Although modern tape drives can wind and unwind tape awfully rapidly, they still can't do so fast enough to simulate random access the way a DVD-ROM can. The tape is more delicate than a DVD disc, and there's usually a lot more of it, since tapes have to store many gigabytes (600GB isn't unusual, and 1600GB is considered state-of-the-art right now) of data, while a DVD only stores up to 8GB or so (even a Blu-Ray disc only stores about 50 to 60GB).

Modern Backup 1.0 solutions typically keep additional supporting files, indexes of a sort, that help them quickly locate a given bit of data within that monolithic backup file structure. Thus, when the backup file *is* located on random-access storage, a backup solution can typically dive in and find a given mailbox message, or whatever, pretty quickly. The problem is that most of us *don't* store our backup files on random-access storage. In fact, in many cases, that backup file is being written directly to tape, where it will *always* be fairly slow to retrieve.

Tape Drives

That is ultimately the downside of tape-based storage: their speed. Capacity can be an issue, too, but that's a question of spending more money: 16 petabyte (PB) tape libraries are widely available. Tape drives are typically measured in how much data they can move *in an hour,* and even at the high end of the current technology—around 450GB/hour—that's a snail's pace, especially when someone's anxiously waiting for a file, or when the boss is tapping his foot waiting for you to bring a failed server back online. Keep in mind that hour isn't the complete recovery time: Once the files are pulled off tape, you still have to have a backup solution go through the file and look for the exact bits of data you need, and write them back to the server they came from (or to an alternative location). Heck, even finding the right tape—especially if it's offsite—can take a long time.

You have to use some caution, too: Tape drive speeds are usually measured in *compressed* data. So a 430GB/hour tape drive isn't *really* moving 430GB/hour of raw data, it's moving that much data *after compression.* So the data will still need to be uncompressed before it can be used. Capacities like 1600GB (1.6TB) are often expressed with compression assumed, and typically a 50% compression ratio is assumed. That means native capacities may be something like 800GB, and the 1.6TB number assumes you're achieving 50% compression, which means on a 430GB/hour drive, you'd be moving 215GB/hour of raw data. It's fun math.

Some companies get frustrated enough with these speeds and capacities that they create a sort of hierarchical storage plan: Backup servers write their files to local hard disk space, then stream that backup file to tape after the backup is complete. Disks can move data a *lot* faster than a tape drive (think 6Gbps for disks versus something like .5Gbps for tape—making disks at least twelve times faster, not to mention capable of random access). An advantage of this approach is that the latest backup file—or even the latest few backup files, if you have the disk space—are handy on faster disk-based storage if you need them. Of course, if you need something older, you're still back to tape.

Disk Storage

It's not unusual, in the Backup 1.0 world, to see administrators considering purely disk-based storage solutions. A 1TB SATA disk drive is far cheaper than a 146GB tape drive. It's also far faster. Why not just rely exclusively on disk-to-disk backup?

Two reasons: Disks are more fragile than tape, and disks are less portable than tape. Disk drives don't do well with shock when being transported; tape doesn't care. Disks are more susceptible to magnetic fields than tapes. Disks can only survive in a narrower temperature range compared with a tape. For an effective backup plan, you need to move copies off-site, and tape remains the best way to do that.

That's why the Backup 1.0 world often settles for disk *and* tape, as Figure 9.2 shows.

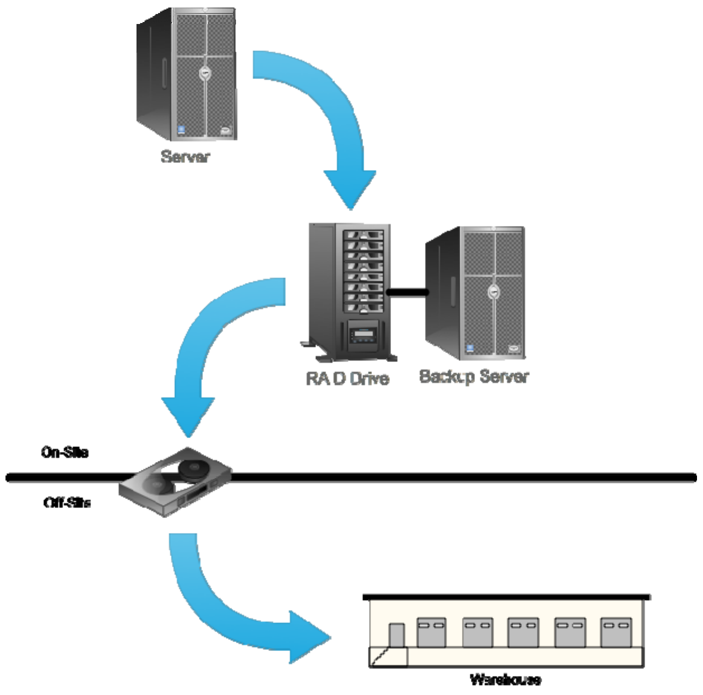


Figure 9.2: Tiered, or hierarchical, backup storage.

This approach gives you fast access to recent backup files, and the ability to easily and safely carry copies of those backups off-site for safekeeping. In fact, this tiered, or *hierarchical* backup storage approach has been effective in solving some of the traditional problems with Backup 1.0, such as recovery speed. It's still constrained by the inherent snapshot-based nature of Backup 1.0, but this kind of storage approach may serve us well for Backup 2.0.

Rethinking Storage for Backup 2.0

Rethink. I use that word deliberately because, like backups themselves, we in the IT world have been doing things the same way for a long time out of sheer inertia. Let's start, in fact, by re-stating our mission for Backup 2.0:

> Backups should prevent us from losing any data or losing any work, and ensure that we always have access to our data with as little downtime as possible.

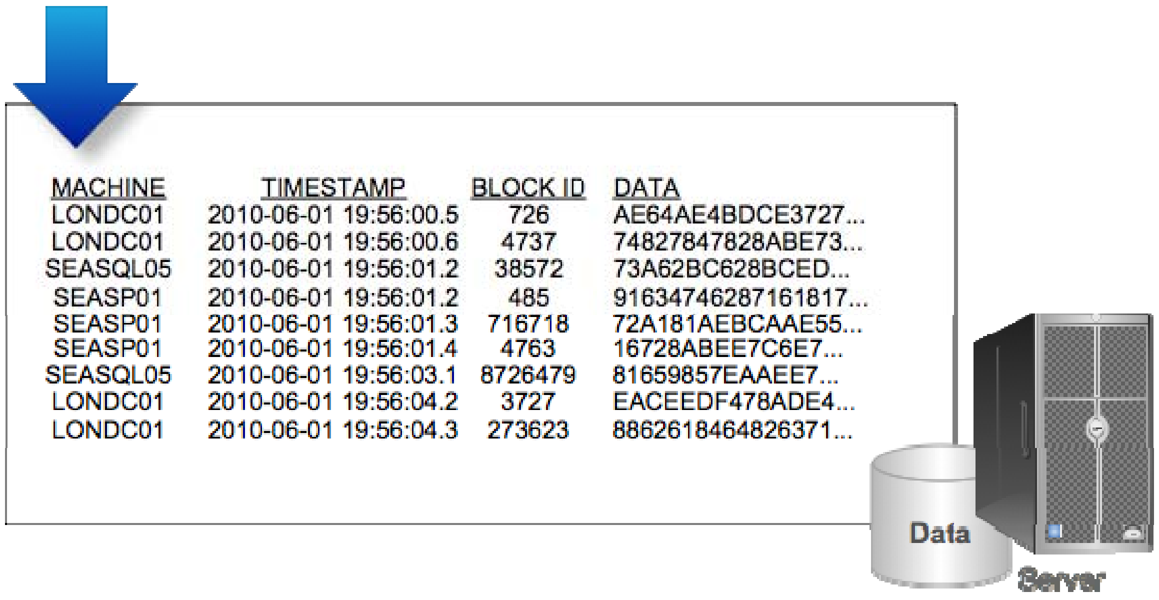
That's going to drive a few priorities for backup storage. We need to keep our backups:

- On fast, random-access storage, because only that offers the "as little downtime as possible" criteria that we're looking for.
- Off site, as well, because fast random-access storage isn't 100% reliable.
- Potentially for a long time, since we may need to roll back to a fairly old point in time, or retrieve data from an old point in time.

Let's not use words like "disk" and "tape" right now; let's focus less on the technology for a moment and focus instead on *capability.* Sure, tape might be one way to get our data off-site—but there might also be other ways, so let's explore those. Disks are great random-access storage devices, but let's focus on that requirement—fast, random-access storage—over the implementation or the medium.

What Backed-Up Data Should Look Like

In a Backup 2.0 world, we're constantly collecting changed disk blocks from multiple different computers. We need to store each disk block, along with an identifier of where it came from, and the timestamp of when we collected it. But we can't lose sight of the fact that we want to be able to retrieve *all* the disk blocks associated with a particular machine, up to a particular timestamp, on demand. In other words, sticking all this data into a tape-centric sequential file structure isn't going to serve our needs very well. It's more likely that these disk blocks will need to go into some kind of database—Figure 9.3 illustrates what I'm talking about. Let's not worry for the moment about how we'll store that database—although disk storage is a safe bet—but focus instead on the capabilities this database gives us.



MACHINE	TIMESTAMP	BLOCK ID	DATA
LONDC01	2010-06-01 19:56:00.5	726	AE64AE4BDCE3727...
LONDC01	2010-06-01 19:56:00.6	4737	74827847828ABE73...
SEASQL05	2010-06-01 19:56:01.2	38572	73A62BC628BCED...
SEASP01	2010-06-01 19:56:01.2	485	91634746287161817...
SEASP01	2010-06-01 19:56:01.3	716718	72A181AEBCAAE55...
SEASP01	2010-06-01 19:56:01.4	4763	16728ABEE7C6E7...
SEASQL05	2010-06-01 19:56:03.1	8726479	81659857EAAEE7...
LONDC01	2010-06-01 19:56:04.2	3727	EACEEDF478ADE4...
LONDC01	2010-06-01 19:56:04.3	273623	8862618464826371...

Figure 9.3: Creating a backup 2.0 backup database.

This type of structure allows any given disk block change to be recalled very quickly, and allows all the blocks for an entire machine up to a certain point in time to be retrieved very quickly. This type of approach has a single downside: It has the potential to take up *gobs* of disk space. This is, in fact, where Backup 2.0 goes from being great on paper to possibly being not-so-great in practice: If you add up the total disk space involved in *every disk block change* across all your servers—can you come up with enough space to store all of that? What about getting it off site for safety? What about keeping a lot of it for a long period of time? Given the sheer amount of data involved, can Backup 2.0 meet our three objectives for backup storage?

- Fast, random-access storage
- Off site storage
- Long archival period

We'll have to see. This is where we need to start really looking at the storage technology—or technologies—that this is going to use.

Physical Storage: Tapes, SANs, NAS, the Cloud, and More

Let's get back to physical storage media, and let's be honest with each other. There are only two practical kinds of mass-storage devices available to us today: disk and tape. We're going to have to find a way to make them work, because it's all we've got.

But disks in particular come in a wide variety of implementations: Relatively cheap local storage inside a server (or in an attached storage expansion cabinet of some kind), inexpensive network-attached storage (NAS) appliances, and somewhat-more-expensive storage area networks (SANs). All of these offer RAID capabilities to help protect us in the event one or two drives crash on us, and all of these are well-understood, robust, mature technologies. iSCSI in particular is making NAS and SAN storage less expensive, better-performing, more flexible, and so on.

Tapes—well, tapes are tapes. Most businesses have autoloaders or tape libraries that can automatically churn through several tapes to store tons and tons of data, and some tape libraries are even directly-connected to a network (although it's still common to have a tape drive or library hanging off the back of a dedicated server or tape library controller of some kind).

You can start dragging "clouds" into the conversation because cloud storage is becoming fashionable, but it's just more hard drives and tapes located somewhere else; it isn't really a different *kind* of storage.

What really matters is speed, cost, and location. Remember, we have three requirements for our backup storage:

- Fast, random-access storage
- Off site storage
- Long archival period

The first requirement addresses speed. The second addresses location. The third impacts cost the most (after all, if we were happy only being able to recover to last night, we wouldn't need to store very much backup data, and so the potential cost of our Backup 2.0 solution wouldn't be very high).

As with Backup 1.0, the answer is going to be a mix of backup technologies. But, as we did with Backup 1.0 in the first place, we need to re-think what a hierarchical backup storage setup might look like—and we need to consider all our modern options.

Rethinking Hierarchical Backup Storage

Let's start by accepting the fact that our Backup 2.0 solution is going to be backing up data to disk. That's going to get us our fast, random-access storage. We can decide how much we want to spend on disk space, and that'll constrain how long of an archive we can maintain in fast, random-access storage. Figure 9.4 shows the initial phase of our storage architecture.

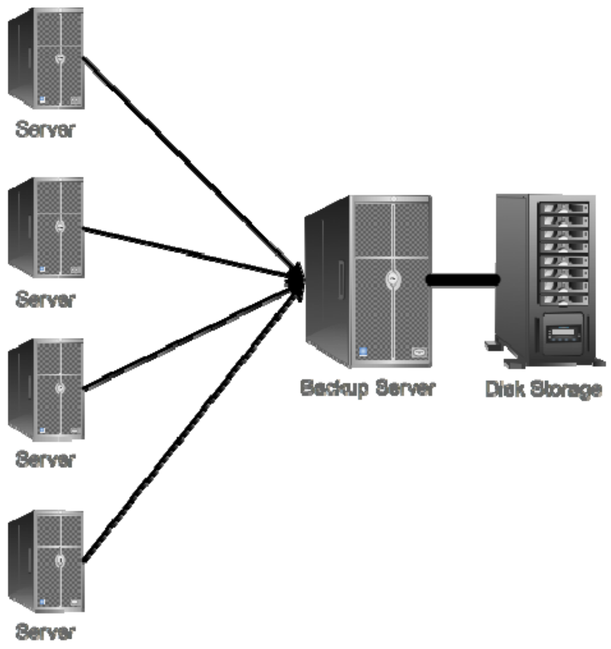


Figure 9.4: Initial storage architecture.

We have four servers that we want to back up, and a central backup server—along with some disk storage—is holding it all. Again, our ability to provide more disk space is the only limitation on how much data we can store on-site, meaning the more disk space we can throw at the problem, the longer we can keep backups on-site, and the further in the past we can dive when we need to recover something.

Backup <> Archive

Don't think of your backup system as an *archive.* In other words, if there are specific pieces of data—quarterly financials, last year's taxes, personnel records, and so on—that you need to permanently archive (or even archive for a number of years), your backup solution isn't the place to do it. Those archives should be someplace else—often on tape, which is pretty durable and doesn't consume power when it isn't being used. You might burn files to optical media, which is equally durable and low-power. But just because you need to keep customer records for, say, 10 years, doesn't mean your backup system needs to be capable of storing 10 years of data.

My general rule is this: I figure out how far in the past I can go in a backup system until I start running into data I would never, ever restore. That's all the data my backup storage systems need to hold. In many cases, I find that to be a month or two, possibly even as long as a quarter, but anything further back tends to be completely useless, so there's no need to back it up. *Archive* it, perhaps, but it doesn't need to live on the live backup system.

Your service level agreements (SLAs) with your users play an important role here. "We can retrieve anything up to 3 months old, and we can do it in an hour or so," is a good SLA. "Anything older than that, we can search the quarterly archives for—but that may take a few days, because we have to bring them in from off-site, and there's a cost associated with doing so." That's a good next step in an SLA: Establishing what can be done, and what it will cost, and how long it will take.

Our next step is to get a copy of our backups offsite. *Not* for long-term storage but because we want to be covered in the event that our data center—containing our backup server—burns down or something awful. We want to get a copy of our backup data off-site fairly frequently, but we don't need to keep copies off-site for more than our on-site backup data retention period—say, 3 months or so.

One way to do so is, of course, tape, as Figure 9.5 shows.

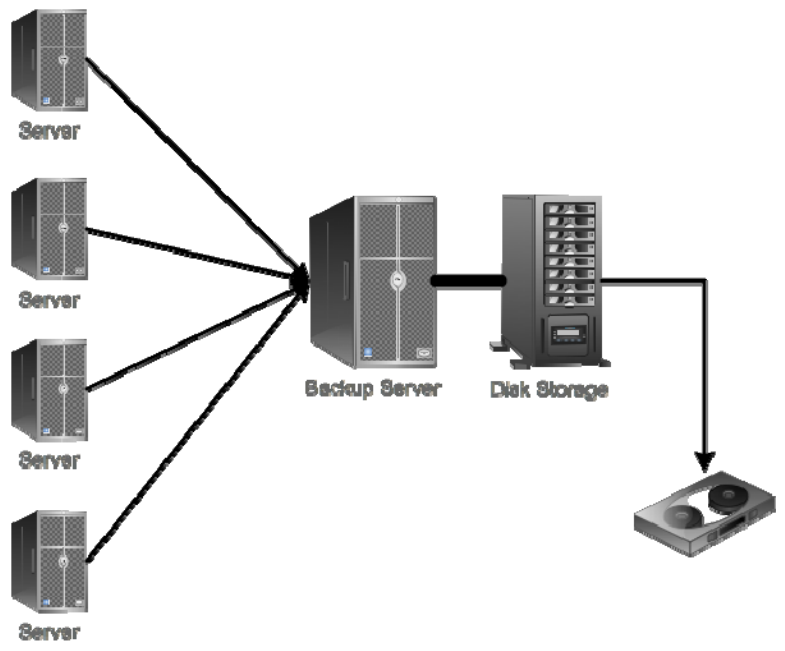


Figure 9.5: Moving data to tape.

As I've acknowledged, tape is a great way for getting data off-site in bulk. The trick is that I don't need my tapes to be an *archive*—I'm just using them to cover myself in the event of a backup server disk failure. So if I'm only keeping 3 months of data (in my example) in the backup server, then my tapes only need to be good for about 3 months. Tapes do represent a snapshot of my backup server's data, so if I do lose my backup server, then I'm at-risk for losing data because I've lost whatever backups occurred between the last tape backup and the backup server failure.

Another option, then, is to migrate data to the cloud to achieve my off-site protection. Figure 9.6 shows what this might look like.

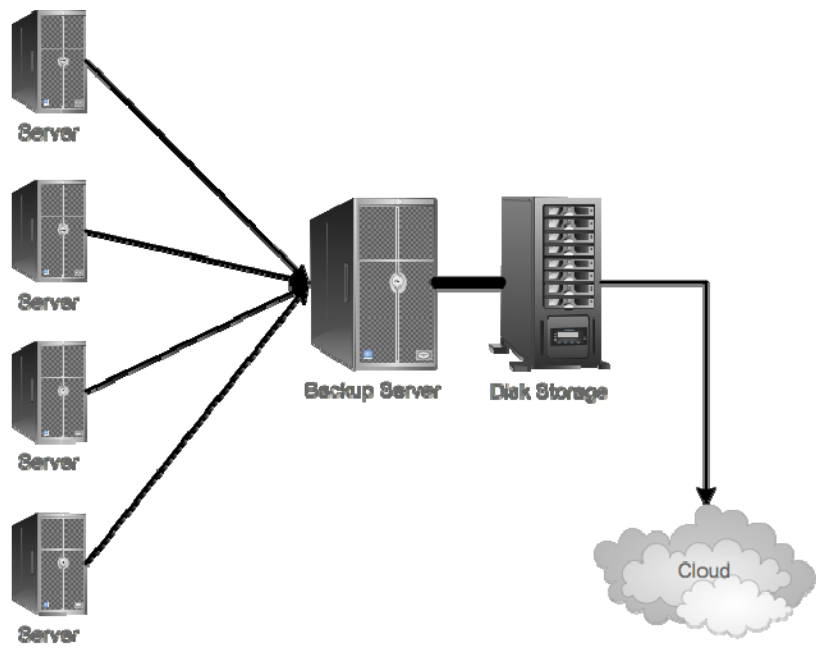


Figure 9.6: Moving data into the cloud.

This accomplishes the goal of getting the data off-site, and it isn't as impractical as it may sound at first. To begin with, cloud-based storage is getting cheaper all the time: Compare it with the cost of making tape backups and storing them off-site, and you might find that the two techniques aren't worlds apart in terms of price. Solutions already exist to move data off-site in near-real-time, using advanced compression and bandwidth-throttling techniques that help conserve WAN bandwidth. This type of backup would be more up-to-date than a tape backup, so less data is at risk—meaning there might be justification for somewhat higher pricing than traditional tapes. There are even solution providers out there who can take your backed-up disk images and move them to their own virtual machine host servers *also* located in the cloud. That can provide an exceptionally quick recovery time, if that's something you feel you'll need for your business.

You could also construct your own "private" cloud, replicating backup content from multiple backup servers to your own "backup of the backup" server—perhaps located in a different office. Figure 9.7 illustrates this idea.

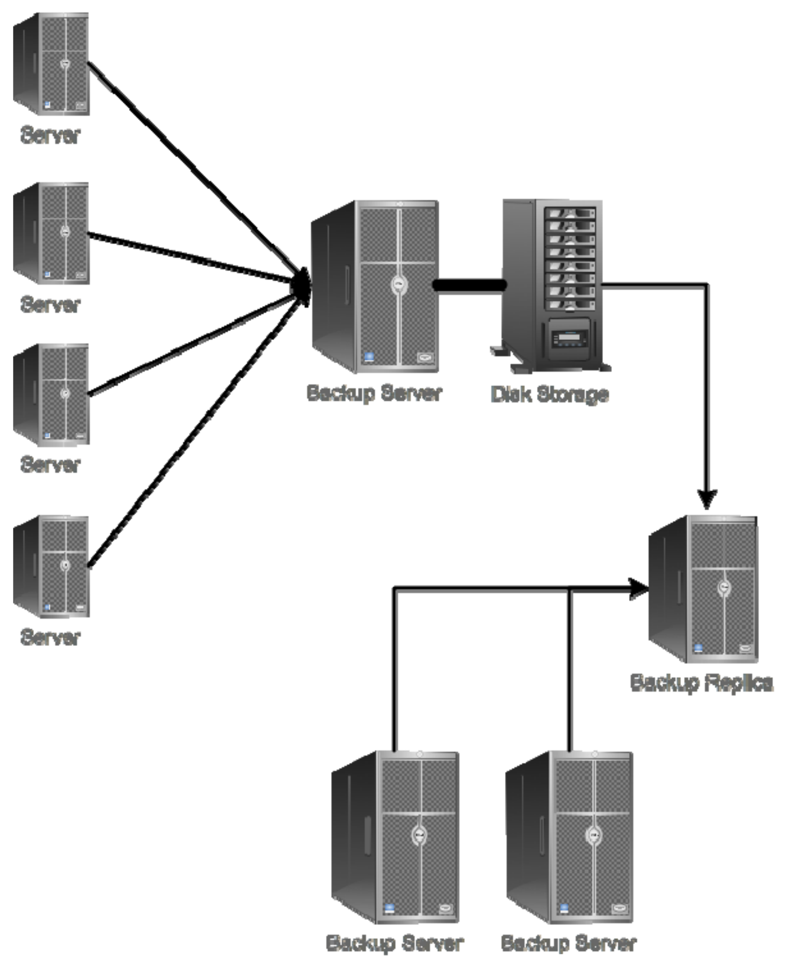


Figure 9.7: "Private cloud" backup.

The backup replica would actually be using *Backup 2.0 techniques* to back up the front-line backup servers, meaning very little data would be at-risk at any given time. If a server fails, its backup server can recover it; if a backup server fails, the central backup replica could recover *it.* Your costs go up, but your ability to survive multiple disasters also increases, so you're getting something for the money.

Having an off-site replica can greatly improve your situation in the event of a total location failure, such as a flood, fire, loss of utility power, meteor strike, and so on. The backup replica can be used to rapidly re-construct all your lost servers, perhaps as virtual machines, in another location. *Rapidly* is the key, there: You won't be spinning data off of a dozen tapes for hours and hours, you'll be immediately building virtual machine disk images and getting them running on a virtual machine host—perhaps even one you've leased for this particular disaster.

The backup replica could also serve as a source for archival activities. Using Backup-2.0 techniques, the backup replica itself would be able to mount disk images, meaning you could grab files, folders, databases, and what-have-you and copy them to tape to create a *permanent* archive of whatever data your business has that *needs* to be permanently archived. Figure 9.8 illustrates this addition.

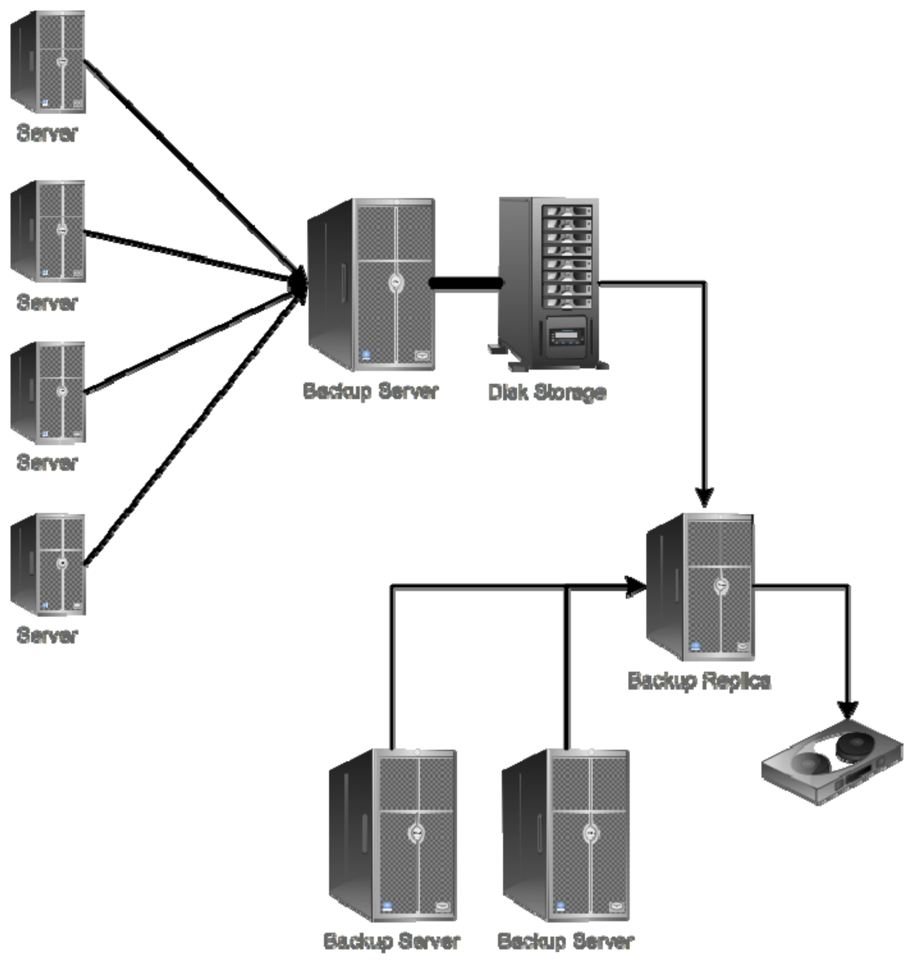


Figure 9.8: Adding tape archival to the mix.

Again, the purpose of these tapes is to permanently archive files that need to be retained beyond your “fast restore” capabilities. Typically, archived files don’t need to be accessed very often, and when they do need to be accessed, it’s for reference or “read-only” purposes. Optical discs might also be suitable as an archival medium, and you could store copies on-site as well as off-site, if desired.

The point of all this is that you can achieve whatever business needs you have. The keys are to clearly differentiate different needs:

- Use fast disk storage to create a "fast recovery" capability. Establish a reasonable period of time that you'll retain in fast recovery, or tier-1 storage.
- Get that data off-site quickly and frequently. Ideally, get it off-site to another "fast recovery" solution. Less-ideally, get it off-site in snapshot form—likely on tape.
- Periodically migrate files to slower, tier-2 storage for long-term archival purposes. Again, archive files aren't needed in the event of a failure, they're needed as a long-term or permanent reference. Archiving isn't part of your backup storage architecture, although your backups may be a starting point for creating archive copies. You don't choose to retain tape copies for 5 years because you think you'll need to recover those files and begin making changes to them; you do so because someone might need to refer to those files at some distant point in the future.

Backup <> Archive, Redux

A backup, for me, is something you keep because you might want to return your production environment (or a bit of it) to the point in time that the backup represents. You may want to recover an accidentally-deleted file, for example, or you might want to restore a failed server, or you may want to recover a crashed database. Backups are for when *things go wrong.*

Archives are things you keep because the data in them has inherent value. You'll never want to make it the "working copy" again, but you might have need to refer to it. Archives aren't used because something went wrong, but because you need to look at something up from the past.

When I run across companies who "save backup tapes for a year" (or more), I die a little bit inside. Does anyone seriously contemplate rolling back the production environment to a year ago? No, they do not. What they've done is created an archive out of a backup—and likely, that "archive" contains data *they will never need to refer to,* like copies of operating system (OS) or application files.

One reason I'm told that disk storage isn't suitable for backup storage is because it's too expensive to keep enough data far enough into the past. My problem with that is that you *shouldn't be keeping backups for that far into the past*—at some point, the backup has no value as a live, production system, and it's turned into an archive. When it does so, it should be trimmed down to *the data you'll actually need to refer to,* and written to second- or third-tier storage.

Carefully consider your business' specific needs with regards to both backups and archives. That'll help you properly plan for the right amount of disk space, storage tiers, off-site storage, and so on.

Saving Space

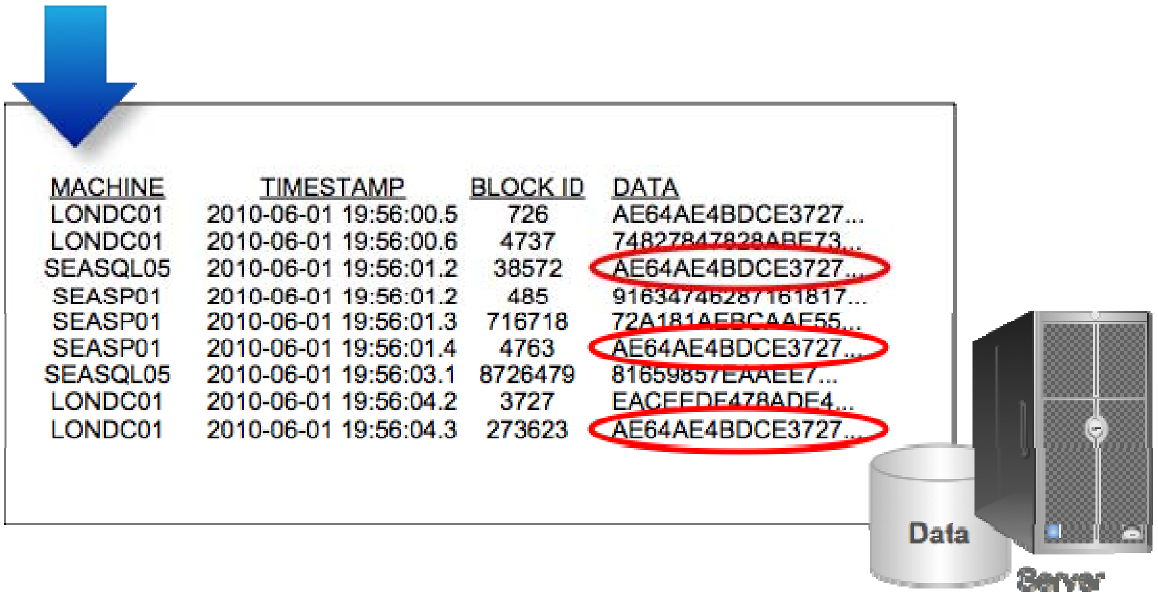
Of course, disk space is still not free, and so the less of it we have to use for our backups, the better. That's why a Backup 2.0 solution should also include two major features—compression and de-duplication—to reduce disk costs as much as possible. Less disk usage on the backup server means less storage used at every successive tier—whether you're moving data off-site on tapes, to a cloud provider, or to an off-site "private cloud."

De-Duplication of Data

De-duplication is a hot new topic in the world of storage management, and there's no reason Backup 2.0 shouldn't be on that bandwagon. The idea is simple: Rather than storing multiple copies of the exact same data, your store it once, along with some notes about all the identical copies that would normally exist.

You would probably be appalled at the amount of duplicate data on your servers. Every OS file, for example, is duplicated across each server—no need to store them all independently when one copy will do. Sometimes, the OS will write a block of data with the same actual contents as were already on disk—no need to store that disk block as a change because there's already a copy of the same data in storage.

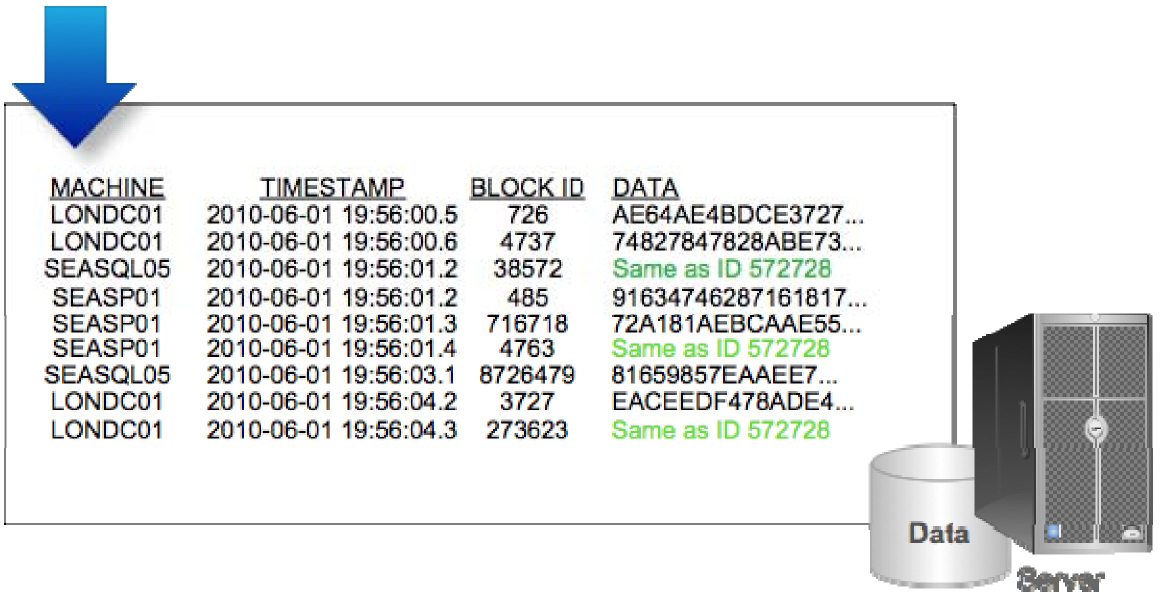
Without de-duplication, our backup storage database could look something like Figure 9.9, with multiple blocks of duplicated data both between servers and even within the same servers.



MACHINE	TIMESTAMP	BLOCK ID	DATA
LONDC01	2010-06-01 19:56:00.5	726	AE64AE4BDCE3727...
LONDC01	2010-06-01 19:56:00.6	4737	74827847828ABE73...
SEASQL05	2010-06-01 19:56:01.2	38572	AE64AE4BDCE3727...
SEASP01	2010-06-01 19:56:01.2	485	91634746287161817...
SEASP01	2010-06-01 19:56:01.3	716718	72A181AEBCAAE55...
SEASP01	2010-06-01 19:56:01.4	4763	AE64AE4BDCE3727...
SEASQL05	2010-06-01 19:56:03.1	8726479	81659857EAAEE7...
LONDC01	2010-06-01 19:56:04.2	3727	EACEEDE478ADE4...
LONDC01	2010-06-01 19:56:04.3	273623	AE64AE4BDCE3727...

Figure 9.9: Duplicated data in the backup storage database.

De-duplication saves us room, by eliminating the duplicated data and just keeping track of where the "master copy" of that data lives. Figure 9.10 shows what that might look like.



MACHINE	TIMESTAMP	BLOCK ID	DATA
LONDC01	2010-06-01 19:56:00.5	726	AE64AE4BDCE3727...
LONDC01	2010-06-01 19:56:00.6	4737	74827847828ABE73...
SEASQL05	2010-06-01 19:56:01.2	38572	Same as ID 572728
SEASP01	2010-06-01 19:56:01.2	485	9163474628716181 7...
SEASP01	2010-06-01 19:56:01.3	716718	72A181AEBCAAE55...
SEASP01	2010-06-01 19:56:01.4	4763	Same as ID 572728
SEASQL05	2010-06-01 19:56:03.1	8726479	81659857EAAEE7...
LONDC01	2010-06-01 19:56:04.2	3727	EACEEDF478ADE4...
LONDC01	2010-06-01 19:56:04.3	273623	Same as ID 572728

Figure 9.10: De-duplicated data in the database.

The idea is that the "pointer" to the actual copy of the data occupies less space than the data itself, so there's a net savings.

There are numerous techniques for de-duplicating data. At a high level, with a backup solution, you could simply look at disk blocks, which are usually relatively small. With larger disk blocks, such as an 8KB block, you might use a technique called *chunking,* where you break down each block into a smaller chunk—say, 1KB chunks. It'll be easier to find duplication at that level, meaning you can save space incrementally.

The tradeoff relates to the pointer size. If duplicated data will be replaced with a 12-byte pointer, there's no sense in looking for duplication at the 12-byte or smaller level because any space you saved would be completely offset by the pointer you had to create.

Note

In backup solutions, there are two kinds of de-duplication, which I'll call Source and Target. In Source de-duplication, the backup solution will de-duplicate data from a particular source, like a server, but it will allow duplicate data to exist between sources—like between two servers. You can save more space with Target de-duplication, which de-duplicates all data regardless of where it came from.

You want to look for a solution that operates at the smallest practical chunk size, which may be the size of a typical disk block. Some systems may only de-duplicate entire files, meaning you're much less likely to find as many duplicates and save space.

Compression of Data

Compression is, of course, a long-used technique for reducing storage requirements, and Backup 2.0 should make good use of it. In fact, a good solution will actually support many compression techniques, as different techniques work best across different kinds of data. *All* compression in a backup system is *lossless,* which means the original data can be completely re-constructed upon decompression. In fact, compression algorithms are really just a very advanced form of data de-duplication, functioning across smaller and larger data sets and across patterns as well as purely-duplicated data. For example, suppose a disk block contained this data, expressed in hexadecimal:

```
1730000AE4627DBCBCBCBCBCBC733475475475475
```

A compression algorithm might reduce that to:

```
173(0x4)AE4627D(6xBC)733(4x475)
```

The idea is to take repetitions over a certain length and replace them with some token that allows the pattern to be re-created during decompression. Shorter patterns might not be replaced at all, if the replacement token would be larger or even the same size as the pattern being replaced.

The nature of data itself is used by other compression algorithms. For example, a simple text document, stored as a plain-text file, typically uses a very limited character set. These characters can be expressed in 4 bits, and do not need an entire 8-bit byte—yet they are stored in full bytes on disk. Six characters might look like this:

```
00000110
00001000
00001101
00000110
```

The upper 4 bits of each byte are essentially wasted; a compression algorithm can “byte pack” these 4 bytes into 2 bytes, for a 50% reduction in space. This simply involves copying the lower 4 bits of one byte into the upper 4 bits of the previous byte:

```
10000110
01101101
```

So long as the decompression algorithm knows how to reverse this process, it works perfectly. Of course, this doesn’t work with more complex data, illustrating the need for a compression system to support many different kinds of algorithms.

Common lossless algorithms include:

- Run-length encoding (RLE)
- LZW (and variants LZ77 and LZ78), which are *dictionary coders*
- Dynamic Markov Compression (DMC)
- Various forms of entropy encoding

Resource

If you're interested, you can read about the specifics of these and other forms of compression—http://en.wikipedia.org/wiki/Data_compression#Lossless_data_compression is a good place to start, as it provides an index to some of the more commonly-encountered algorithms.

Between compression and de-duplication, a backup storage system can save a lot of space. It's relatively straightforward for compression algorithms to achieve 50% compression (which is why so many tape backup device manufacturers assume that rate of compression in their statistics); de-duplication can easily achieve an additional 20 to 30% reduction in space, meaning your backup storage can use as much as 80% less space than the data it is backing up actually consumes in production.

Working with Backed-Up Data

The last concern, of course, is being able to *use* your backed-up data, and the storage architecture obviously plays an important role. In a Backup 2.0 solution, *using* the data means being able to re-assemble the disk blocks for an entire server hard drive, or just a portion of it, quickly. Ideally, you should be able to push those blocks back to their original location, or to an alternative location (such as a virtual machine) for recovery purposes. You might also want the ability to mount those blocks as a working disk image so that you can browse it and retrieve data from it *without* having to push the blocks to a real server. Look for a solution that will give you flexibility in how you access and utilize your backed-up data.

Coming Up Next…

A lot of my discussion on Backup 2.0 thus far has focused on backup and restore: Making sure you can retrieve a single file, or a single mailbox, or a single other piece of data when you need to. Along the way, I've made mention of disaster recovery—which is what you'll be doing when an entire server, or an entire data center, crashes—but in the next chapter, I'm going to focus exclusively on that important topic. Disaster recovery is a distinct set of concerns from normal backup and restore, and it's another area where we'll really have to examine our assumptions, carefully reconsider our actual business objectives, and see what a "2.0" approach can offer.