Cheap off-site backup at Amazon Glacier

Description	File organization, compression, hashing and uploading to S3.
Period	Summer 2014
Languages & Libs	Bash
Tags	AWS, Encryption, Backups

In addition to a mirrored and check-summed ZFS based backup server, I wanted to have backups outside by premises to be safer against hazards such as burglary, fire and water damage. ZFS can already resist single disk failure and can repair silent data corruption, but for important memories that isn't sufficient level of protection. My ever-growing data set is currently 150k files, having a total size of 520 Gb. Amazon's Glacier seems to be the most cost efficient solution with sophisticated APIs and SDKs.

Currently the Glacier storage at EU Ireland is priced at 0.011 USD / Gb / month, which is about 0.10 EUR / Gb / year, or only 50 EUR / year for 500 Gb of data. However data retrieval cost calculations are quite non-trivial, because of numerous quotas and the pricing is based on monthly peak retrieval rate. The best reference for this is an Unofficial Amazon AWS Glacier Calculator, which tells us that to download 500 Gb in two weeks costs 11.51 USD for retrieval from Glacier storage and 59.88 USD for transfer cost, totalling at roughly 60 USD or 44 EUR. Halving the retrieval time doubles the retrieval cost, so it is important to split the data in smaller files and retrieve them in a carefully scheduled manner. This doesn't bother me too much because I hope I don't ever need to resort to this, and instead it is almost like write-only memory.

Pricing information can be expected to change quite rapidly, but I'm going to quote current prices for reference anyway. Perhaps the most popular service Dropbox quotes 500 EUR / year for 500 Gb, as can be seen from Figure 1. Unlimited Dropbox for Business is 15 USD / month / user, but a minimum of 5 users is required. Anyway Dropbox is meant for read-write data storage and sharing, not for "offline-like" backup-only solution. I'm unsure if it supports user-configured encryption keys, unless the user has to keep encrypted and decrypted copies of data on his/her computer.

Figure 1: Dropbox pricing for 500 Gb is about 500 EUR / year, 1000% more than Glacier. However it has ready-made desktop and mobile applications, no retrieval pricing and other advantages.

An other interesting service is Blackblaze, which offers unlimited storage for 50 USD / year, which is very competitive. Unfortunately they don't support Linux or Unix, which I want to use for main storage to get the benefits of ZFS. Also their clients are based on consumer-friendly synchronization principle, so there is less fine-tuned control on what is actually happening. They also compare themselves to their competitors, and quoted prices can be seen in Figure 2.

	Backblaze	Carbonite	Mozy	Crashplan	iDrive
Price for 1-Year	$50	$59	$59	$59	$49.50
Data backed up by default	All	Partial	Partial	Partial	Partial
Data storage limit	Unlimited	Unlimited	50 GB	Unlimited	150 GB

Figure 2: Blackblaze offers competitive pricing when compared to its consumer-friendly peers.

Given all these more or less ready-made solutions, I still ended up in choosing the Amazon Glacier service. I was lured in by the AWS Free Tier offer and the familiar by-engineers-for-engineers approach. It also gives me good first-hand experience on Simple Storage Service (S3), which's life cycle management I use to move files to their final destination: Amazon Glacier.

I naturally organize my files based on their topic and date, but on my pre-existing files I had to also split the files into smaller subsets. The initial step was to organize the data into about 50 - 100 GB folders, each of which is further split into about 10 GB partitions. Each file in these subsets is then SHA-1 checksummed and this file is stored into Dropbox and other services. Based on this file I can search files based on their (full) file name and their actual contents.

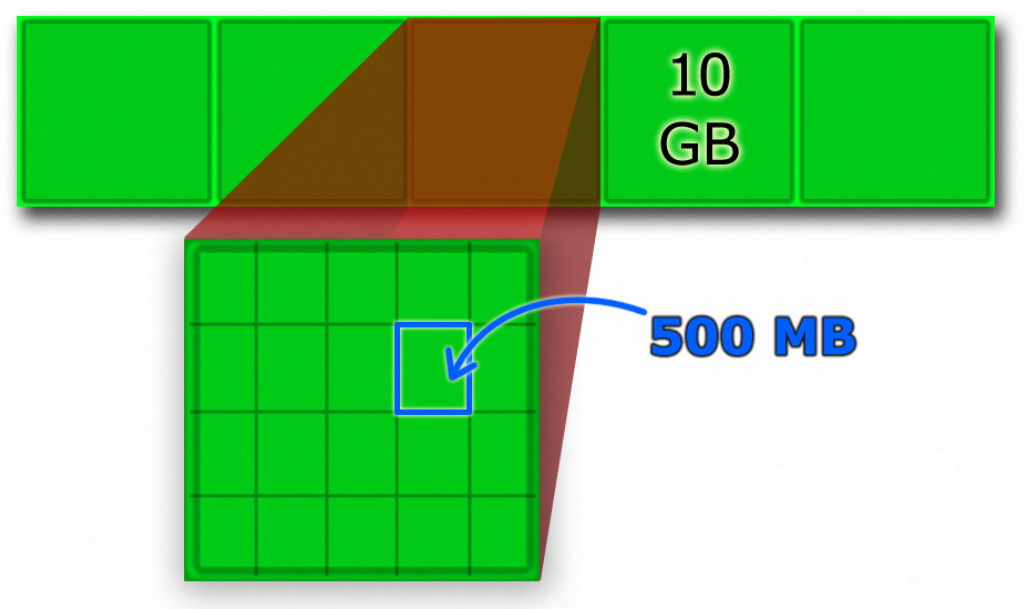

Figure 3: Files are divided into about 10 GB subsets, zipped and then splitted into 500 MB chunks. These chunks are then uploaded to S3/Glacier and checksummed.

After creating individual partitions, the actual compression and encryption can be done. The bash script is based on piping zip and OpenSSL Linux utilities, and the used encryption method is AES-256 in CBC mode with random salt. It is crucial not to forget the password for these backups, because there is no other recovery method than brute-force.

Since the data in question is mostly RAW camera files they aren't that compressible, the resulting zip files are roughly 10 GB each as well. Before uploading these are split into 500 MB chunks, which illustrated in Figure 3. This enables easier upload retries and fine-tuned Glacier retrieval rate. The resulting zip files and their 500 Mb or less chunks are also SHA-1 checksummed. Amazon reports so-called ETag hashes, which is based on MD5 hashes of uploaded parts. It can more-or-less prove that Amazon got the intact copy of the file over the internet.

My scripts utilize standard AWS CLI and s3md5 command line utilities, so no Perl or PHP coding is required (except the Python abort script to clean-up aborted multi-part uploads, which is based on this blog post and boto. The zip, encrypt, split and checksum commands look like this:

```
zip - -q -r path/to/subset | openssl aes-256-cbc -salt -k password > subset.zip.aes
split -d -bytes=500M subset.zip.aes subset.zip.aes_p
sha1sum subset.zip.aes* | tee subset.sha1.txt
ls subset.zip.aes_p* | xargs -n1 ./s3md5.sh 7 | tee subset.etag.txt
```

Beacuse I have 2 x 250 Gb SSD drives in RAID-0 configuration (via ZFS), even the Core i5 processor is able to calculate checksums in parallel at 1 GB/s (8 Gb/s)! Thus multiple passes over the data will be fast.

The current version of s3md5.sh can only parse individual files, so `xargs` is used to generate this sequence of program executions. The parameter "7" tells s3md5.sh that files are uploaded in 7 MB chunks. It also gave wrong results if the file happened to be evenly split into x MB chunks, in which case the last chunk is an empty zero-length string which needs to be ignored. I'll be sending a pull request on these patches later. This command can be used to confirm that the files will decrypt correctly:

```
cat subset.zip.aes_p* > subset.zip.aes.tmp
openssl aes-256-cbc -d -salt -in subset.zip.aes.tmp -k password > /dev/null
```

Naturally I have written nicer scripts to wrap these commands, to enable easy iteration over files etc. The actual upload uses the `aws s3 cp -storage-class REDUCED_REDUNDANCY` command, which is wrapped in trickle command to limit the upload bandwidth. The S3 bucket is configured to move files to Glacier storage after 1 day, so typically lower storage redundancy is sufficient. This provides 99.99% durability, where Glacier guarantees 99.999999999%. If the file goes missing within the one day period it can easily be re-uploaded (assuming this doesn't go unnoticed). Overall the AWS Glacier seems like a good solution, if you wan to have full control on all aspects of you storage "medium". However it takes extra care to properly organize your data and to be sure to use correct encryption keys, and hope that there is no in-RAM corruption if you aren't using ECC RAM (I'm not).

This project was initially set-up to archive my old files reliably and cheaply to the cloud, and it succeeds well in this task. After finishing the uploading process of 520 GB of data, I need to extend these scripts to handle automatic incremental backups of still-changing fresh files. A simple solution would be to use a Elastic Block Storage volume at Elastic Compute Cloud, and to use BitTorrent Sync P2P software to upload encrypted incremental ZIP files. This is a bit less of a concern because typically I still have a copy of the files on my laptop as well.