

Cultured Perl: Perl and the Amazon cloud, Part 1

Learn the basics of Amazon's S3 and SimpleDB services by designing a simple photo-sharing site

Skill Level: Intermediate

Teodor Zlatanov (tzz@lifelogs.com)

Programmer

Gold Software Systems

31 Mar 2009

This [five-part series](#) walks you through building a simple photo-sharing Web site using Perl and Apache to access Amazon's Simple Storage Service (S3) and SimpleDB. In this installment, get a feel for the benefits and drawbacks of S3 and SimpleDB by taking a tour of their architectures and starting to design your photo-sharing site.

So you want to learn about two of Amazon's Web services: Amazon S3 (Simple Storage Service) and Amazon SimpleDB. What better way to learn than a hands-on experience? In this case, you'll build a simple photo sharing site.

The goal is not to build a well-designed site; that's been done many times. Besides, putting together a Web site is *hard*, and the technical side is only part of the equation, so please don't send me complaints like "d00d your teh worst" because sharethewindbeneathmydonkey.com didn't make you a million in the first week. (But if it does, remember me as the one who got you started.)

To get the most from this series

This series requires beginner-level knowledge of HTTP and HTML, as well as intermediate-level knowledge of JavaScript and Perl (inside an Apache mod_perl process). Some knowledge of relational databases, disk storage, and networking will be helpful. The series will get increasingly more technical, so see the [Resources](#) section if you need help with any of those topics.

I'll use `share.lifelogs.com` in this series as the domain name. Let's take a look at Amazon S3.

Amazon S3 overview

I've been a UNIX® administrator for awhile now, and I can tell you that backups and file storage are not simple services. If acronyms such as SAN, NAS, LUN, LVM, RAID, JBOD, IDE, and SCSI don't mean anything to you, then be glad. If they do, you've surely whimpered quietly into your tear-stained napkin at lunch and hoped for a better way to manage data after, say, the third month of restoring from corrupted four-year-old DLT backups. Not that I've ever done that.

IBM and Amazon Web Services

Cloud computing provides a way to develop applications in a virtual environment, where computing capacity, bandwidth, storage, security and reliability aren't issues—you don't need to install the software on your own system. In a virtual computing environment, you can develop, deploy, and manage applications, paying only for the time and capacity you use, while scaling up or down to accommodate changing needs or business requirements.

IBM has partnered with Amazon Web Services to give you access to IBM software products in the Amazon Elastic Compute Cloud (EC2) virtual environment. Our software offerings on EC2 include:

- DB2® Express-C 9.5
- Informix® Dynamic Server Developer Edition 11.5
- WebSphere® Portal Server and Lotus® Web Content Management Standard Edition
- WebSphere sMash

This is product-level code, with all features and options enabled. Get more information and download the Amazon Machine Images for these products on the IBM developerWorks [Cloud Computing Resource Center](#).

For more cloud computing resources, see the [Cloud Computing for Developers](#) space on developerWorks.

Amazon's S3 (Simple Storage Service) is a distributed storage system. If you're willing to trust Amazon with your data, it makes life quite a bit easier. Of course, you can always run your own backups to be sure. (Security might also be an issue: putting data into S3 means you have to use S3's access control system, which might not fit your authentication and authorization requirements. Check the S3 documentation listed in [Resources](#) for details.)

So what do you get with S3? S3 uses a user key (a long, random-looking string) and

a user password (another random-looking string) to let you store and retrieve files. You get charged according to Amazon's S3 pricing, which you can find on their Web site. It's not too expensive; when compared to the costs of keeping your own NAS or SAN or local disks, S3 is quite reasonable.

As of early 2009, S3 data is hosted at two Amazon data centers (the US and EU centers) with good network connectivity. If you want to serve your data to large audiences outside the US and EU, you should run tests with a service such as Gomez or Keynote, which are designed to determine worldwide performance. Even within the US and EU, if your business depends on serving data quickly and reliably, you should set up daily performance tests through such a service.

The major problem with a distributed storage system is its *update latency*. This is the time between the content owner's actions and when those actions propagate. But simple time between actions and propagation isn't the only potential worry; the propagation may not be uniform, so your customers may see different content at different times. Amazon guarantees consistency at the server, meaning that your customers will not see corrupted data, but you should bear this in mind as you evaluate S3. When you upload, modify, or delete an image, don't expect the changes to take effect immediately.

There are Perl libraries for S3 access on the CPAN (see [Resources](#)). Net::Amazon::S3 is a good option, but there are many others listed on Amazon's S3 resource page. We won't need to use them, because our S3 integration uses S3 features to bypass any Perl code when content is uploaded. (In addition, there are many good tools for accessing S3—such as JungleDisk or the Firefox S3Fox add-on—that make it easy and convenient to manage your data without Perl.)

An example of Amazon S3

Now, onward to what you get with S3. Files (called *objects* by S3) are stored in *buckets*. In each bucket, a file name (its *key*) has to be unique. You can give files attributes like "color" or "language," but those are not part of the file name.

Let's say you store the picture of the American flag as "images/flag.png" in the bucket "us.images.share.lifelogs.com" and the picture of the German flag as "images/flag.png" in the bucket "de.images.share.lifelogs.com" (they are named the same but in different buckets). Your users can then request `http://us.images.share.lifelogs.com.s3.amazonaws.com/images/flag.png` to get the American flag or `http://de.images.share.lifelogs.com.s3.amazonaws.com/images/flag.png` to get the German flag. Furthermore, you can alias `de.images.share.lifelogs.com` to `de.images.share.lifelogs.com.s3.amazonaws.com` in DNS (do the same for `us.images.share.lifelogs.com`), so users will just have to request `http://us.images.share.lifelogs.com/images/flag.png` or

<http://de.images.share.lifelogs.com/images/flag.png> to get the flags.

Note that bucket names have to be unique across all Amazon S3 accounts, so names like "test" and "default" are no good. Qualify the bucket name with the full domain name if possible. It makes identifying the bucket and using it in DNS easier. Also, bucket names are pretty limited, so don't try to write a novel in them. Stick to the same characters you'd use for a domain name.

S3 is a complex service, so I encourage you to look at the S3 home page before you go on.

Amazon SimpleDB overview

This is the part where professional speakers and college professors yell out a random phrase to wake up that snoring guy in the front row who was up until 3 AM last night doing tequila shots: DATABASES ARE IMPORTANT!

Are you awake?

Sorted, filtered, aggregated, averaged, analyzed, the flood of raw data we face every day can become an unmanageable stream of information. Hosting these databases is a full-time job for IT professionals. They require space, power, backups, and many other resources. Using hosted databases such as SimpleDB may be worthwhile for your business as a financial decision; I will only be explaining the technical side.

A simple database example is a to-do list pinned to the fridge: each item is on a line by itself, and there's a check mark next to some and perhaps others are crossed out. In a traditional relational database, this might be modeled as two tables with two columns each:

Table 1. todo_foreign table

item	statuscode (FK to status.statuscode, default 0)
call Mom	0
call IRS	2
get milk	1

Table 2. Status table

statuscode	statusdesc
0	active
1	done
2	deleted

"But wait," you say. "What about the date when the item was completed or deleted, who modified it, and what are the data types? This is, after all, why we train wise, powerful database administrators (DBAs). They know all about normal forms and foreign keys and SQL. Surely, you need one of them to look over your design now, right?"

Yes, thank you, Mr. Smarty Pants, but leave my simple example alone. Console yourself later by cuddling up with a copy of "Secrets Of The SQL and RDBMS Gods For Dummies."

Amazon SimpleDB is a widely distributed key-attribute database. It's definitely not for every business, and there are tight restrictions on performance and scalability. Attributes are limited to 1KB each, so your to-do items can't have a name longer than one kilobyte.

Security is also an issue; SimpleDB's access control system is similar to S3's. A social site such as the simple one you will assemble in this series can thrive with SimpleDB as the database back end. Still, you should assess your business requirements, budget, and data storage needs to find out if SimpleDB will fit them all.

The S3 update latency issue I mentioned affects SimpleDB as well. Your updates do not take immediate effect everywhere.

Using our simple database example, the SimpleDB structure would be:

Table 3. SimpleDB todo structure

Item	Status
call Mom	active
call IRS	deleted
get milk	done

So far, so good. This is simpler than the first example, isn't it? Oh, but let's add another item:

get cow	active
---------	--------

Do you see that the status is duplicated? The word `active` is now stored twice in the database. This can be expensive for large tables in terms of storage and performance. On the other hand, each SimpleDB row is self-sufficient by design. When you get that row, you've got everything it contains. You don't need to look up the status description. With the update latency of SimpleDB, this matters.

More SimpleDB to-do lists

Let's say you add a new status code, `waituntiltomorrow`, and apply it to one item in the `todo_foreign` table (Table 1, with the foreign keys). So, now you have two updates (one to the status table, one to `todo_foreign`). If the status table (Table 2) update happens *after* the `todo_foreign` update, you'll have inconsistent data. Remember, SimpleDB doesn't guarantee that your updates will make it out immediately in the order you make them, so besides the performance penalty you'll pay for doing two lookups (one for the item, one for the status code description), you may also have inconsistent data.

Here's the key to SimpleDB: forget about the columns in `todo_simple` (Table 2). SimpleDB doesn't have columns! It has attributes for each row. Those attributes are not static, so you can add and delete them at will. You want your to-do items to have a creation and a deletion date? Just give them those attributes. In `todo_foreign`, that would require two columns; the deletion date might be null to indicate that the item is still active. Let's add one more column for the date the item was done. Or maybe it should be a status code only, and use the deleted date as the done date. What do we do?

The SimpleDB way is to just do what you need. You need a creation date? Make a `created_date` attribute. A deletion date? Assign that attribute only to items that have been deleted. The *presence* of the attribute tells us that it applies to this item.

Stop thinking in terms of columns. SimpleDB rows are more like Perl's hashes. Every key is a string. Every value is a string or an array of strings. Let's try our design again:

Listing 1. `todo_freeform`

```
{ item: "call Mom" }
{ item: "call IRS", deleted_date: "2009-03-01" }
{ item: "get milk", done_date: "2009-03-02" }
```

Note that SimpleDB has an implicit key called `ItemName`, which in this case would be the to-do item as a string, like so:

Listing 2. SimpleDB todo list

```
"call Mom" { }
"call IRS" { deleted_date: "2009-03-01" }
"get milk" { done_date: "2009-03-02" }
```

SimpleDB doesn't allow an object without attributes, so give all objects a `created_date` attribute, like so:

Listing 3. SimpleDB todo list with `created_date` added

```
"call Mom" { created_date: "2009-02-01" }  
"call IRS" { created_date: "2009-02-01", deleted_date: "2009-03-01" }  
"get milk" { created_date: "2009-02-01", done_date: "2009-03-02" }
```

"But wait," you cry, "Everything really is a string? Data is not rigidly typed? Aaaaah! Doom!"

Yes. Everything is a string. Isn't it wonderful?

Oh, and you can add a `delete_reason` attribute to any deleted item in this table three months after it goes live. It won't break anything, and only new code that knows about it has to use it.

I'll pause here for dramatic effect while the DBAs take a couple of aspirin. Meanwhile, the Perl programmers are giving them glasses of water just because, well, that's the kind of nice people we are.

Moving on with the example. The important part now is figuring out the queries that will give us active, deleted, or done items. This is really simple; you can look at the SimpleDB documentation for all the query options. We'll use the SELECT language. There's also a QUERY language, but SELECT is closer to SQL and thus easier to understand for most readers.

Listing 4. `todo_freeform` queries

```
-- get active  
select * from todo_freeform where done_date is null and deleted_date is  
null  
-- get deleted  
select * from todo_freeform where deleted_date is not null  
-- get done  
select * from todo_freeform where done_date is not null
```

There you go. Now let's put SimpleDB and S3 together.

Integrating the services and sharing photos

The next question you're probably asking is, how can I connect SimpleDB and S3? (They are not innately connected except for the access control model they use.) Easy: you can simply store the bucket and name for an S3 object in SimpleDB. In any case, enough with the to-do list; let's start designing the photo-sharing site.

The site needs to store photos in S3 and user comments in SimpleDB. What about user accounts? We need to live with the distributed nature of SimpleDB, which means we will have invalid users sometimes (such as when the user has not been pushed out but the row that refers to that user has). We'll keep users in SimpleDB

for this application, though. There will be no dependency on any external databases, as the goal here is to be able to set up the site anywhere quickly, with just some Perl glue running under `mod_perl` and with the real action happening on S3 and SimpleDB.

First, you'll need a photo table. Records might look like this:

Listing 5. Photo table records, `share_photos`

```
"http://developer.amazonwebservices.com/connect/images/amazon/logo_aws.gif"
{ user: "ted", name: "Amazon Logo" }

"http://images.share.lifelogs.com/funny.jpg"
{ user: "bob", name: "Funny Picture", s3bucket:
  "images.share.lifelogs.com" }
```

Next, a users table:

Listing 6. Users table, `share_users`

```
"ted" { given: "Ted", family: "Zlatanov" }
"bob" { given: "Bob", family: "Leech" }
```

And comments:

Listing 7. Comments, `share_comments`

```
"random-string"
{
  url: "http://images.share.lifelogs.com/funny.jpg",
  comment: "Ha ha",
  posted_when: "2009-03-01T19:00:00+05"
}

"random-string2"
{
  user: "ted",
  url:
  "http://developer.amazonwebservices.com/connect/images/amazon/logo_aws.gif",
  comment: "No it doesn't",
  posted_when: "2009-03-01T20:00:01+05"
}

"random-string3"
{
  url:
  "http://developer.amazonwebservices.com/connect/images/amazon/logo_aws.gif",
  comment: "No it doesn't",
  reply_to: "random-string2",
  posted_when: "2009-03-01T20:00:01+05"
}
```

Worth noting

Google offers at least some comparable services; this series is not in any way a comparison between Google's services and Amazon's services. You can find plenty of comparisons online. This series also does not discuss other Amazon offerings, such as the Elastic Compute Cloud (EC2), even though they are interesting and useful and could certainly help in putting together a Web site presence. Finally, there are other distributed key-value databases like CouchDB that compare favorably to SimpleDB; I highly recommend investigating them as well.

We'll thread comments by looking at the `reply_to` key. Every post will have a random string as the unique key.

Please notice that we've established some conventions here:

- The lack of a `user` attribute means the comment is anonymous.
- The photo URL unites all the comments, so changing the photo URL will not be allowed.
- S3 objects will have a URL, too, with a bucket name to identify them as S3 objects.
- Duplicate photo URLs will not be allowed, because the URL is the key of the object.

This is not the final version of the table design (remember, SimpleDB is very flexible), but this is enough to get started.

Wrapup

You've now seen the benefits and drawbacks of S3 and SimpleDB in some detail. Though not by any means complete, this discussion should help you decide if Amazon's S3 and SimpleDB are right for your project.

After providing a simple to-do list example in SimpleDB, you saw how to start designing the data layout of the user, photo, and comment tables. In Part 2, I set up the Web site (Apache with `mod_perl`) and the libraries to work with S3 and SimpleDB.

Resources

Learn

- Service offering details and developer resources are on Amazon.com:
 - [Amazon S3](#)
 - [Amazon SimpleDB](#)
- [mod_perl](#) brings together the full power of Perl and the Apache HTTP server.
- [Prototype](#) is a JavaScript framework that makes it easier to develop dynamic Web applications via a toolkit for class-driven development and the "nicest" Ajax library on Earth. And here's an excellent article on using it: "[Developer Notes for prototype.js](#)" by Sergio Pereira.
- [Outages](#) are an important consideration because S3 and SimpleDB are Web-based services and can suffer outages.
- Cloud computing with Amazon services is a hot topic. This series is about accessing the services using Perl, but for a broader overview of the offerings, read the series "[Cloud computing with Amazon Web Services](#)" (developerWorks, July 2008 - February 2009).
- In the [developerWorks Linux zone](#), find more resources for Linux developers, and scan our [most popular articles and tutorials](#).
- See all [Linux tips](#) and [Linux tutorials](#) on developerWorks.
- Stay current with [developerWorks technical events and Webcasts](#).

Get products and technologies

- At the [CPAN \(Comprehensive Perl Archive Network\)](#) site you can find modules—scads and scads of modules—and module documentation.
- [S3Fox, the Amazon S3 Firefox Organizer](#), the S3 management add-on for Firefox, puts an easy-to-use front end on S3.
- [Order the SEK for Linux](#), a two-DVD set containing the latest IBM trial software for Linux from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.
- With [IBM trial software](#), available for download directly from developerWorks, build your next development project on Linux.

Discuss

- Get involved in the [developerWorks community](#) through blogs, forums, podcasts, and spaces.

About the author

Teodor Zlatanov

Teodor Zlatanov emerged with an M.S. in computer engineering from Boston University in 1999. He has worked as a programmer since 1992, using Perl, Java, C, and C++. His interests are in open source work on text parsing, database architectures, user interfaces, and UNIX system administration.

Trademarks

IBM, the IBM logo, ibm.com, DB2, developerWorks, Lotus, Rational, Tivoli, and WebSphere are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. See the current list of [IBM trademarks](#).

Linux is a trademark of Linus Torvalds in the United States, other countries, or both. UNIX is a registered trademark of The Open Group in the United States and other countries.