S3 を利用して保管を容易にする

Amazon の Simple Storage Service を利用してクラウドの世界に入る

Andrew Glover
Co-Founder
ThirstyHead.com

2009年 4月 07日

概要: Amazon の S3 (Simple Storage Service) は誰もが利用できるサービスであり、Web アプリケーション用のデジタル資産 (画像、動画、音楽、文書など) の保管に利用することができます。S3 には RESTful な API が用意されており、プログラムでこの API を利用すると、S3 サービスとやり取りすることができます。この記事で、オープンソースの JetS3t ライブラリーの使い方を学び、データの保管と取得に Amazon の S3 クラウド・サービスを活用しましょう。

はじめに

クラウドとは、疎結合された一群のコンピューターを連携動作させることで何らかのタスクまたはサービスを実行し、あたかもそうしたタスクやサービスが 1 つのエンティティーによって実現されているように見せる抽象的な概念です。クラウドの背景にあるアーキテクチャーも抽象的です。つまり各クラウド・プロバイダーは、そのプロバイダーにとって適切と思えるサービスを自由に設計することができます。クラウドが何らかのサービスをユーザーに提供するという意味で、SaaS (Software as a Service) もクラウドに関連する概念です。クラウド・モデルによって、ユーザーにとってのコストが下がる可能性があります。ソフトウェアやそのソフトウェアを実行するためのハードウェアはサービスのプロバイダーが既に購入しており、ユーザーはそれらを購入する必要がないからです。

流行語の誕生

クラウド・コンピューティングという言葉は決して新しいものではありません (Amazon は 2006年にクラウド・サービスの提供を開始していました)。しかし Google と Amazon のクラウド・サービスがますます注目を集めるようになるにつれ、2008年にはクラウド・コンピューティングが流行語となりました。Google の App Engine を利用すると、Google のインフラ上で Web アプリケーションを作成したりホストしたりすることができます。Amazon Web サービスには S3 の他に、Amazon のインフラの上でアプリケーションをホストできる EC2 (Elastic Cloud Compute) というコンピューティング Web サービスがあります。他の会社も Amazon と Google に対抗する意図を表明しています。例えば Microsoft® は Azure によって、さらには Sun Microsystems でさえ対抗の意志を表明しています (ただし、この記事の執筆時点では Sun のクラウド・コンピューティング製品はまだ正式に市場導入されていません)。IBM® も最近、特定の製品群を Amazon の EC2 環境で利用できるようにすると発表しました。

例えば Amazon の S3 サービスを考えてみましょう。その名前からもわかるように、S3 は誰もが利用できるサービスであり、このサービスを利用すれば Web アプリケーションに使用するデジタル資産 (画像、動画、音楽、文書など) を保管することができます。S3 を使用する場合、S3 はデジタル資産が保管されているハード・ドライブを持つ、インターネット上に配置されたマシンのように見えます。実際には、(さまざまな地域に分散して配置された) 多数のマシンがデジタル資産 (あるいはデジタル資産の一部) を保管しています。また Amazon は、データの保管と取得のためのサービス・リクエストに関するすべての複雑な処理を行います。ユーザーは Amazon のサーバーに資産を保管する場合、少額の料金 (毎月ギガバイト当たり約 15 セント) を支払います。また Amazon のサーバーからのデータ転送に対しても、少額の料金を支払います。

Amazon の S3 サービスでは RESTful な API を公開しているので、S3 にアクセスするコードをゼロから作らなくても、この API を利用することで HTTP での通信をサポートする任意の言語を使って S3 にアクセスすることができます。JetS3t プロジェクトはオープンソースの Java ライブラリーであり、S3 の RESTful な API を扱うための詳細を抽象化し、この API を通常の Java のメソッドやクラスとして公開します。作成するコードは少ないに越したことはありません。また、誰か他の人による苦労の成果を借用することも非常に重要なことです。この記事を読むとわかるように、JetS3t を利用することで S3 と Java 言語の処理が非常に容易になり、最終的に作業が非常に効率的になります。

S3 の詳細

論理的に見ると、S3 は 1 つの巨大なハード・ドライブに見えるグローバルな SAN (Storage Area Network) であり、デジタル資産の保管や取得に利用することができます。ただし技術的に見ると、Amazon のアーキテクチャーは SAN とは少しばかり異なります。S3 を介しての保管、取得の対象となる資産は「オブジェクト」と呼ばれます。オブジェクトは「バケット」に保管されます。これを頭の中でイメージするためにはハード・ドライブを考えます。オブジェクトはファイルに相当し、バケットはフォルダー (またはディレクトリー) に相当します。そしてハード・ドライブの場合とまったく同じように、オブジェクトとバケットの検索は URI (Uniform Resource Identifier) を使って行います。

例えば私のハード・ドライブには whitepaper.pdf という名前のファイルがあり、このファイルはホーム・ディレクトリーの documents という名前のフォルダーの中にあります。従って、この .pdf ファイルの URI は /home/aglover/documents/whitepaper.pdf です。S3 の場合、URI は少し異なります。第 1 に、バケットが存在するのは最上位レベルのみであり、ハード・ドライブ上のフォルダー (またはディレクトリー) の場合のようにネストさせることはできません。第 2 に、バケットはインターネットでの命名規則に従う必要があります。つまり、バケット名ではピリオドの隣にダッシュを使うことはできず、名前にアンダーバーを含めてもいけない、といった規則があります。そして最後に、バケット名は Amazon のドメイン (s3.amazonaws.com) の中の公開 URI の一部になるため、S3 全体のなかで一意でなければなりません。(幸いなことに、アカウントごとに持てるバケットは 100 個までに制限されるため、良さそうな名前を何百も先行登録して売りつける人達がいるとは思えません。)

DNS のマジック

S3 資産の URL に関してあまり心配する必要はありません。DNS (Domain Name System) と CNAME (canonical name) レコードのマジックのおかげで、S3 の URL に対してカスタムの

URL を 割り当てることができるのです。こうすることで、アプリケーションが S3 を利用しているという事実を隠すことができます。

S3 では、バケットは URI のルートの働きをします。つまりバケットの名前は URI の一部になり、S3 の中のオブジェクトを指します。例えば agdocs という名前のバケットと whitepaper.pdf という名前のオブジェクトがあった場合、その URI は http://agdocs.s3.amazonaws.com/whitepaper.pdf になります。

また S3 には、ハード・ドライブ上のファイルやフォルダーの場合と同様、バケットとオブジェクトに関してその所有者とアクセス権を指定できる機能が用意されています。S3 でオブジェクトまたはバケットを定義する際には、誰がどのように (例えば、読み書き権限など) S3 資産にアクセスできるかに関するアクセス制御ポリシーを指定することができます。従ってオブジェクトに対していくつかの方法でアクセスすることができ、RESTful な API はそのうちの 1 つにすぎません。

S3 と JetS3t の使い方入門

S3 を使い始めるためにはアカウントが必要です。S3 は無料ではないため、アカウントを作成する際には支払い方法 (クレジットカードの番号など) を Amazon に提示する必要があります。ただし心配する必要はありません。セットアップは無料であり、使用量に対してのみ支払えばよいのです。この記事で説明する例に対する料金は非常に少額であり、1 ドルにも満たないはずです。

アカウントを作成するプロセスの一部として、クレデンシャルも作成する必要があります (つまりアクセス・キーと秘密鍵が必要です。これはユーザー名とパスワードと考えることができます)。(x.509 証明書を取得することもできますが、Amazon の SOAP API を使う場合以外には必要ありません。) どのようなアクセス情報を扱う場合にも言えることですが、絶対に秘密鍵を「秘密」にする必要があります。誰かが皆さんのクレデンシャルを手に入れ、それを使って S3 にアクセスすると、請求は皆さんに行きます。そのため、バケットやオブジェクトを作成する際には必ずすべてを非公開にし、外部からのアクセスには皆さんが明示的に許可を与えるようにします。

アクセス・キーと秘密鍵が入手できると JetS3t をダウンロードすることができます。そして JetS3t を駆使し、RESTful な API を介して S3 とやり取りすることができます。

JetS3t を使ってプログラムで S3 にサイン・インするプロセスは 2 つのステップで構成されています。第 1 のステップでは `AWSCredentials` オブジェクトを作成し、このオブジェクトを `S3Service` オブジェクトに渡す必要があります。`AWSCredentials` オブジェクトは極めて単純なオブジェクトで、アクセス・キーと秘密鍵を `String` として受け取ります。`S3Service` オブジェクトは実際にはインターフェース型です。S3 には RESTful な API と SOAP API の両方が用意されているため、JetS3t ライブラリーには `RestS3Service` と `SoapS3Service` という 2 つの実装型が用意されています。この記事の目的としては (そして実際に S3 のすべてとは言わないまでも大部分を利用する場合は)、単純さの点で RESTful な API の方が選択肢としては適切です。

接続された `RestS3Service` インスタンスの作成は簡単です (リスト 1)。

リスト 1. JetS3t の `RestS3Service` のインスタンスを作成する

```
def awsAccessKey = "blahblah"
def awsSecretKey = "blah-blah"
def awsCredentials = new AWSCredentials(awsAccessKey, awsSecretKey)

def s3Service = new RestS3Service(awsCredentials)
```

ここからが面白い部分です。例えばバケットを作成し、そのバケットにムービーを追加したら、特別な有効期限付きの URL を取得します。この説明だけ聞くと、この例はビジネス・プロセスのように聞こえるのではないでしょうか。実際、ムービーなどの制限付きの資産のリリースに関するビジネス・プロセスなのです。

バケットを作成する

私の架空のムービー・ビジネスのために、bc50i というバケットを作成します。JetS3t を使うと、このプロセスは簡単です。`S3Service` 型を使う場合、いくつかの選択肢があります。私が好きな方法は `getOrCreateBucket` 呼び出しを使う方法です (リスト 2)。名前からわかるように、このメソッドを呼び出すと、(`S3Bucket` 型のインスタンスで表現される) バケットのインスタンスが返されるか、あるいは S3 の中にバケットが作成されます。

リスト 2. S3 サーバー上でバケットを作成する

```
def bucket = s3Service.getOrCreateBucket("bc50i")
```

この単純なコード・サンプルを軽く見てはいけません。JetS3t ライブラリーは非常に広範な機能をカバーしています。例えば、`listAllBuckets` 呼び出しを使って `S3Service` のインスタンスを要求するだけで、すぐに自分が持っているバケットの数を確認することができます。このメソッドによって `S3Bucket` インスタンスの配列が返されます。また、バケットの任意のインスタンスに対して、そのバケットの名前と作成日を要求することもできます。もっと重要な点として、そのバケットに関連付けられたアクセス権を JetS3t の `AccessControlList` 型を使って制御することができます。例えば bc50i バケットのインスタンスを取得し、このバケットを誰もが読み書きできるようにすることができます (リスト 3)。

リスト 3. バケットに対するアクセス制御リストを変更する

```
def bucket.acl = AccessControlList.REST_CANNED_PUBLIC_READ_WRITE
```

もちろん、この API を利用してバケットを削除することもできます。Amazon ではさらに、どの地域にバケットを作成するかを指定することさえできます。実際のデータの保管場所に関する複雑な処理を Amazon が行ってくれるため、バケット (そしてその中のすべてのオブジェクト) を (現在利用可能な選択肢である) 米国またはヨーロッパのどちらに置くかを Amazon に指示するだけでよいのです。

バケットにオブジェクトを追加する

JetS3t の API を使って S3 オブジェクトを作成するためには、バケットを操作すればよいだけです。また JetS3t ライブラリーは賢く作られており、S3 バケットの中にあるファイルのコンテンツ・タイプに関する複雑な処理も一部行うことができます。例えば、私が nerfwars2.mp4 というムービーを S3 にアップロードし、顧客が期間限定で見られるようにしたいとします。そのための S3 オブジェクトを作成するには、通常の `java.io.File` 型を作成して `S3Object` 型をバケットに関連付ければよいだけです (リスト 4)。

リスト 4. S3 オブジェクトを作成する

```
def s3obj = new S3Object(bucket, new File("/path/to/nerfwars2.mp4"))
```

`S3Object` をファイルとバケットに関連付けて初期化したら、あとはこのオブジェクトを `putObject` メソッドを使ってアップロードすればよいだけです (リスト 5)。

リスト 5. ムービーのアップロードは簡単です

```
s3Service.putObject(bucket, s3obj)
```

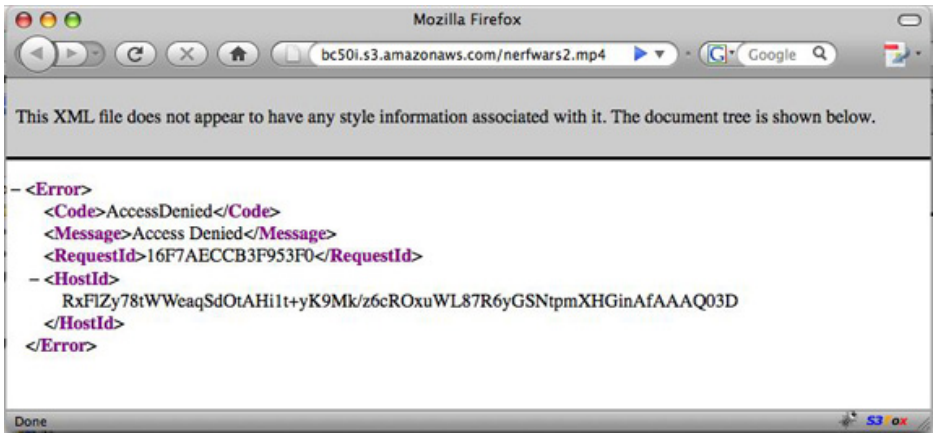
リスト 5 のコードを作成したら作業は終わりです。このムービーは今や Amazon のサーバー上にあり、このムービーに対するキーはこのムービーの名前です。もちろん、このオブジェクトを何か別の名前で呼ぶ必要がある場合には、この名前を変更することもできます。実際、オブジェクトを作成する際には、JetS3t の API (そして関連する Amazon S3 の RESTful な API) によって、少し追加の情報を公開することができます。もちろんアクセス制御リストを提供することもできます。S3 内部のすべてのオブジェクトは追加のメタデータを持つことができ、API を使ってこれらのメタデータを作成することができます。そして後から、S3 API (そしてその派生としての JetS3t) を使って任意のオブジェクトにクエリーを実行し、そのメタデータを取得することができます。

オブジェクトに対する URL を作成する

この時点で、この S3 インスタンスには中にムービーが入ったバケットがあります。このムービーは実際に http://bc50i.s3.amazonaws.com/nerfwars2.mp4 という URI にあります。しかし私以外は誰もこのムービーを取得することができません。(そしてこの場合、私はプログラムを使わない限りこのムービーにアクセスすることができません。なぜなら、すべてのものに対してデフォルトのアクセス制御設定がされており、クレデンシャルを持たないアクセスはすべて拒否されるからです。) ここでの目標は、アクセスに対する課金を開始するまでの限られた期間、特定の顧客が新しいムービーを見られるようにすることです (これも S3 の機能を使うと容易に行えます)。

図 1 は、デフォルトのアクセス制御の実際を示しています。返される (そしてブラウザーに表示される) XML 文書によって、私がアクセスしようとした資産 (http://bc50i.s3.amazonaws.com/nerfwars2.mp4) へのアクセスが拒否されたことがわかります。

図 1. Amazon のセキュリティーの実際



S3 で公開されている便利な機能を利用すると、公開 URL を作成することができます。実際、S3 を使用すると、一定期間 (例えば 24 時間) のみ有効な公開 URL を作成することができます。そこで、先ほど S3 のサーバーに保管したムービーに対して、48 時間有効な URL を作成することにします。そしてこの URL を特定の顧客に提供し、彼らが自由にこのムービーをダウンロードして

見られるようにします (ただし彼らがこのムービーを 2 日以内にダウンロードするという前提です)。

有効期限付きの URL を S3 オブジェクトに対して作成するためには JetS3t の `createSignedGetUrl` メソッドを使います (このメソッドは `S3Service` 型の静的メソッドです)。このメソッドは引数として、バケット名、オブジェクトのキー (この場合はムービーの名前であることを覚えているでしょうか)、(JetS3t の `AWSCredentials` オブジェクトの形式での) いくつかのクレデンシャル、そして有効期限が切れる日を取ります。求めるバケットの名前とオブジェクトのキーがわかっている場合には、即座に URL を得ることができます (これを Groovy コードで示したものがリスト 6 です)。

リスト 6. 有効期限付きの URL を作成する

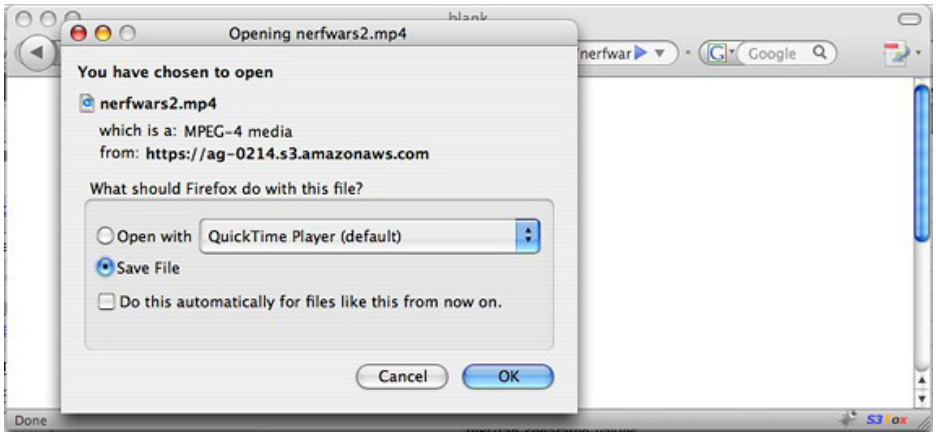
```
def now = new Date()
def url = S3Service.createSignedGetUrl(
   bucket.getName(), s3obj.key, awsCredentials, now + 2)
```

Groovy では、`+ 2` 構文を使うと 48 時間時間後の日付を非常に容易に指定することができます。これにより、URL は次のようになります (表示の都合で 2 行になっていますが、実際は 1 行です)。

```
https://bc50i.s3.amazonaws.com/nerfwars2.mp4?AWSAccessKeyId=
   1asd06A5MR2&Expires=1234738280&Signature=rZvk8Gkms%3D
```

このようにして作成された URL を使うと、ブラウザーからのリクエストが受け入れられます (図 2)。

図 2. URL によってダウンロードが実現される



このプロセスは非常に簡単だと思いませんか。ほんの数行のコードを作成するだけで、特別な URL を使わないとダウンロードできないセキュアな資産をクラウドの中に作成することができたのです。

有効期限付きのダウンロードに S3 を活用する

帯域幅とストレージに対する要求が一定ではない場合には S3 が非常に有効です。例えば、私が示しているビジネス・モデル (1 年をとおして特定の時点でムービーがリリースされるモデル) を考えてみてください。従来のストレージ・モデルでは、ラック上のどこかに大量のスペースを購入

する必要があります (あるいは独自のハードウェアと、そのハードウェアへのパイプを用意する必要があります)。そしてほとんど必ずと言っていいほど、短期間のうちに大量のダウンロードが行われた後、比較的アクティビティーの低い静かな期間が続きますが、そうした場合にも需要がどれだけあったかとは無関係に費用を支払う必要があります。しかし S3 の場合には、需要に応じたモデルを実現することができます。つまりビジネスは、ストレージと帯域幅を必要とする場合にのみ、その費用を支払えばよいのです。しかも、S3 のセキュリティー機能を利用すると、動画をダウンロードできる時間や、さらには誰がダウンロードできるのかに至るまで、細かく指定することができます。

こうした要件を S3 を利用して実現するための方法は非常に簡単です。限られた期間だけムービーをダウンロードできるように公開するための概要としては、以下の 4 つのステップが必要です。

1. S3 にサイン・インします。
2. バケットを作成します。
3. そのバケットに対して、必要な動画 (またはオブジェクト) を追加します。
4. 有効期限付きの URL をその動画に対して作成します。

それだけで終わりです。

賢明な選択

従来のストレージ・モデルに比べ、S3 での使用量に応じて支払うモデルは明らかな利点をいくつか持っています。例えば、私の音楽のコレクションを私自身のハード・ドライブに保管するためには、私は最初にハード・ドライブを購入しなければなりません (例えば 500GB のドライブを 130 ドルで購入する、など)。私は 500GB 近くものデータを保管する必要はないため、突き詰めて言えば、使用しない容量分に対して (非常に安価ですが) ギガバイト当たり約 25 セント支払うことになります。また、このハード・ドライブを動作させ、そのための使用電力に対して料金を支払わなければなりません。Amazon のサービスを利用すれば、すぐに古くなる資産に対して最初に 130 ドルを支払う必要はありません。私はギガバイト当たり 10 セント安い料金を支払えばよく、ストレージ用のハードウェアの管理や保守を行う必要がありません。同じ利点を企業の規模で考えてみてください。例えば Twitter は、百万を超えるユーザー・アカウント用の画像を S3 に保管しています。使用量に応じて支払うシステムにより、Twitter は、そうした画像を保管して提供するためのハードウェア・インフラの購入、またそうしたインフラの構成や保守に対して継続的に発生する作業や部品のコスト、といった高額の出費を回避しています。

クラウドの利点はそれだけではありません。低遅延や高可用性も実現することができます。推定では Amazon のクラウド・サービスに保管された資産は物理的に世界中に配置されるため、さまざまな場所で高速にコンテンツを提供することができます。しかも、資産はさまざまなマシンに分散されるため、一部のマシン (あるいはネットワークの一部) に障害が発生してもデータの高可用性は維持されます。

要約すると、Amazon の S3 のメリットは、低コスト、高可用性、そしてセキュリティー、という単純なものです。皆さんが SAN の専門家としてデジタル資産保管用のハードウェア資産の保守を楽しんでいるのでない限り、Amazon の方が皆さんよりも適切な仕事をしてくれるはずです。他の人から借用すれば済むにもかかわらず、最初からハードウェアに費用をかける理由は何もないのです (ハードウェアは時間の経過と共に価値がなくなることを忘れてはなりません)。

著者について

Andrew Glover

Andrew Glover は、開発者、著者、講演者、そして起業家です。また、easyb BDD (Behavior-Driven Development) フレームワークの創始者、そして「Continuous Integration」、「Groovy in Action」、「Java Testing Patterns」の 3 冊の本の共著者でもあります。彼は ThirstyHead.com で、Groovy、Grails、およびテスト関連のさまざまなクラスを教えています。Andrew がソフトウェア開発について定期的にブログを書いている thediscoblog.com で、最新情報を入手してください。