Amazon Simple Storage Service

開発者ガイド

API Version 2006-03-01

AWS, Inc.

Amazon Simple Storage Service: 開発者ガイド

AWS, Inc.

Amazon S3 へようこそ

これは *Amazon Simple Storage Service 開発者ガイド*です。このガイドでは、バケットとオブジェクト、およびそれらのリソースで作業するための Amazon S3 アプリケーションプログラミングインターフェイスの使用方法など、Amazon S3 の中心的概念を説明します。また、リクエストを送信してバケットを作成する方法、オブジェクトを保存および取得する方法、リソースへのアクセス許可を管理する方法について説明します。さらに、アクセスコントロールと認証プロセスについても説明します。アクセスコントロールでは、Amazon S3 内のオブジェクトおよびバケットにアクセスできるユーザーとアクセスの種類 (READ、WRITE など) を定義します。認証プロセスは、アマゾン ウェブ サービス (AWS) にアクセスを試みるユーザーの身元を確認します。

Amazon Simple Storage Service (Amazon S3) は、クラウドにデータを保存できるウェブサービスです。その後、データをダウンロードしたり、Amazon Elastic Compute Cloud といった他の AWS サービスを使用してデータを利用したりすることができます。 (『Amazon Elastic Compute Cloud ユーザーガイド』を参照してください。)

知りたい情報

情報	関連するセクション
一般的な製品の概要と料金体系	「Amazon Simple Storage Service (Amazon S3) 」
Amazon S3 の実践的入門	『Amazon Simple Storage Service 入門ガイド』
Amazon S3 の主要な用語と概念	Amazon S3 のご紹介 (p. 2)
バケットの操作方法	Amazon S3 バケットの使用 (p. 87)
オブジェクトの操作方法	Amazon S3 オブジェクトの使用 (p. 107)
リクエストの実行方法	リクエストの実行 (p. 11)
リソースへのアクセスの管理方法	アクセスコントロール (p. 294)

Amazon S3 のご紹介

Topics

この「Amazon Simple Storage Service のご紹介」では、このウェブサービスの要点について詳しく説明します。このセクションを読むことで、本サービスの内容と、ビジネスへの利用方法についてご理解いただけます。

Amazon S3 の概要

Amazon Simple Storage Service はインターネット用のストレージサービスです。開発者がウェブスケールのコンピューティングを簡単に利用できるように設計されています。

Amazon S3 のウェブサービスインターフェイスはシンプルで、いつでも、ウェブのどこからでも容量に関係なくデータを格納および取得できます。これにより、すべての開発者が、スケーラブルで信頼性が高く、かつ高速で安価なデータストレージインフラストラクチャを利用できるようになります。このインフラストラクチャは、Amazon が使用しているウェブサイトのグローバルネットワークと同じものです。このサービスの目的は規模のメリットを最大化して開発者に提供することです。

Amazon S3 のメリット

Amazon S3 は、シンプルさと堅牢性を重視し、必要な機能に絞って提供しています。Amazon S3 サービスには次のようなメリットがあります。

- バケットの作成–データを格納するバケットを作成し、名前を付けます。バケットとは、Amazon S3 におけるデータストレージ用の基本的なコンテナです。

- バケットへのデータの格納 – 膨大な量のデータをバケットに格納します。 必要な数のオブジェクトを Amazon S3 バケットにアップロードします。各オブジェクトに最大 5 TB のデータを格納できます。各オブジェクトの格納と取得には、開発者が設定した一意のキーを使用します。
- データのダウンロード – データをダウンロードするか、他のユーザーがダウンロードできるようにします。 データはいつでもダウンロードできます。他のユーザーにデータのダウンロードを許可することもできます。
- アクセス許可 – Amazon S3 バケットにおけるデータのアップロードまたはダウンロードを、他のユーザーに対して許可または拒否します。アップロードおよびダウンロードのアクセス許可を 3 タイプのユーザーに付与します。データが不正アクセスに侵されないように認証機能もついています。
- 標準的なインターフェイス – 標準ベースの REST および SOAP インターフェイスを使用し、あらゆるインターネット開発ツールキットが使えるように設計されています。

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

Amazon S3 の概念

Topics

このセクションでは、Amazon S3 を効果的に使用するために理解しておく必要のある重要な概念と用語について説明します。使用頻度が高いものから順に説明します。

バケット

バケットとは、 &S3 に格納されるオブジェクトのコンテナです。すべてのオブジェクトはバケット内に格納されます。例えば、`photos/puppy.jpg` という名前のオブジェクトが `johnsmith` バケットに格納される場合、URL `http://johnsmith.s3.amazonaws.com/photos/puppy.jpg` を使ってアドレスを解決できます。

バケットの用途には、Amazon S3 の最上位の名前空間を形成する、ストレージとデータ転送の課金アカウントを特定する、アクセスコントロールに使用する、使用状況レポートの集計単位として使用するなど、さまざまなものがあります。

特定のリージョンに作成されるようにバケットを設定できます。詳細については、「Buckets and Regions (p. 90)」を参照してください。また、オブジェクトが追加されるたびに Amazon S3 で一意なバージョン ID を生成してオブジェクトに割り当てるようにすることもできます。詳細については、「Versioning (p. 388)」を参照してください。

バケットの詳細については、「Amazon S3 バケットの使用 (p. 87)」を参照してください。

オブジェクト

オブジェクトは、Amazon S3 に格納される基本エンティティです。オブジェクトは、オブジェクトデータとメタデータで構成されます。データ部分を Amazon S3 から見ることはできません。メタデータは、オブジェクトを記述する、名前と値の一連のペアです。これには最終更新日などのデフォルトメタデータや、Content-Type などの標準 HTTP メタデータが含まれます。開発者が、オブジェクトの格納時にカスタムメタデータを指定することもできます。

オブジェクトは、キー (名前) とバージョン ID によってバケット内で一意に識別されます。詳細については、「Keys (p. 4)」と「Versioning (p. 388)」を参照してください。

キー

キーとは、バケット内のオブジェクトの固有の識別子です。バケット内のすべてのオブジェクトは、厳密に 1 個のキーを持ちます。バケット、キー、バージョン ID の組み合わせによって各オブジェクトが一意に識別されるため、 Amazon S3 は「バケット + キー + バージョン」とオブジェクト自体の基本的なデータマップであると考えることができます。Amazon S3 内の各オブジェクトは、ウェブサービスエンドポイント、バケット名、キー、およびオプションでバージョンを組み合わせることで一意にアクセスすることができます。例えば、http://doc.s3.amazonaws.com/2006-03-01/AmazonS3.wsdl という URL では、「doc」がバケットの名前で、「2006-03-01/AmazonS3.wsdl」がキーです。

リージョン

作成したバケットを Amazon S3 でどこに格納するか、地理的なリージョンを選択できます。レイテンシーを最適化し、コストを最小限に抑えて規制用件に対応できるリージョンを選ぶとよいでしょう。Amazon S3 は、現在次のリージョンをサポートしています。

- US Standard Uses Amazon S3 servers in the United States

  Provides eventual consistency for all requests. This region automatically routes requests to facilities in Northern Virginia or the Pacific Northwest using network maps.
- US West (Oregon) Region Uses Amazon S3 servers in Oregon

  Provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.
- US West (Northern California) Region Uses Amazon S3 servers in Northern California

  Provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.
- EU (Ireland) Region Uses Amazon S3 servers in Ireland

  Provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.
- Asia Pacific (Singapore) Region Uses Amazon S3 servers in Singapore

  Provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.
- Asia Pacific (Sydney) Region Uses Amazon S3 servers in Sydney

  Provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.
- Asia Pacific (Tokyo) Region Uses Amazon S3 servers in Tokyo

  Provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.

- South America (Sao Paulo) Region Uses Amazon S3 servers in Sao Paulo

  Provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.

明示的に別のリージョンに移動する場合を除き、特定のリージョンに保存されたオブジェクトは、そのリージョンから移動されることはありません。例えば、欧州 (アイルランド) リージョンに格納されたオブジェクトは、ずっとそのリージョンに置かれたままです。

Amazon S3 のデータ整合性モデル

単一のキーに対する更新はアトミックです。例えば、既存のキーに PUT を実行すると、後続の読み取りで古いデータが返されたり更新されたデータが返されたりする可能性がありますが、壊れたデータや部分的なデータが書き込まれることはありません。

Amazon S3 は、Amazon のデータセンターに配置された複数のサーバー間でデータを複製することにより、高い可用性を実現します。PUT リクエストが成功した場合は、データが安全に格納されています。ただし、変更に関する情報は Amazon S3 内でレプリケーションされる必要があり、これにはしばらくかかります。レプリケーションでは、次の動作を確認することができます。

- 新しいオブジェクトを Amazon S3 に書き込み、すぐに読み取りを試みます。変更が完全に反映されるまで、Amazon S3は「キーが存在しません」というレポートを表示する場合があります。
- 新しいオブジェクトを Amazon S3 に書き込み、すぐにバケット内のキーのリストを表示します。変更が完全に反映されるまで、オブジェクトがリストに表示されないことがあります。
- 既存のオブジェクトを置換し、すぐにそのオブジェクトの読み取りを試みます。変更が完全に反映されるまで、Amazon S3 は古いデータを返す場合があります。
- 既存のオブジェクトを削除し、すぐにそのオブジェクトの読み取りを試みます。削除が完全に反映されるまで、Amazon S3 は削除済みのデータを返す場合があります。
- 既存のオブジェクトを削除し、すぐにバケット内のキーのリストを表示します。削除が完全に反映されるまで、Amazon S3 は削除済みのオブジェクトをリストに表示する場合があります。

米国スタンダードリージョンでは、すべてのリクエストに対して結果整合性が保証されます。それ以外のリージョンでは、新しいオブジェクトの PUT については「書き込み後の読み込み」整合性、PUT および DELETE の上書きについては結果整合性を提供します。

Note

現在のところ、Amazon S3 はオブジェクトのロックをサポートしていません。同じキーに対して 2 つの PUT リクエストが同時に行われた場合、最新のタイムスタンプを持つリクエストが優先されます。これが問題になる場合は、アプリケーション内にオブジェクトロックメカニズムを構築する必要があります。

更新はキーベースであり、複数キーにまたがるアトミックな更新を行う方法はありません。例えば、ご自分で機能をアプリケーション設計に組み込まない限り、別のキーの更新に依存してキーを更新することはできません。

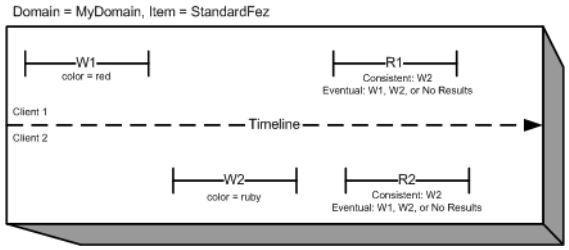
次の表は、結果的に整合性のある読み込みと、整合性のある読み込みの特徴をまとめたものです。

結果的に整合性のある読み込み	整合性のある読み込み
古いデータを読み込むことがある	古いデータを読み込まない
読み込みのレイテンシーは最小	読み込みのレイテンシーが大きくなることがある
読み込みスループットは最大	読み込みスループットが小さくなることがある

アプリケーションの同時実行

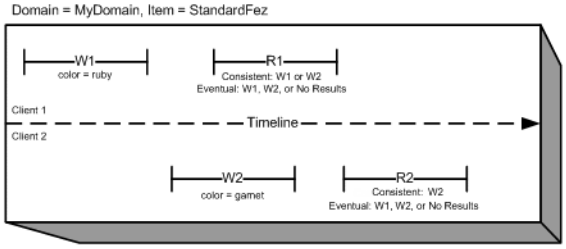
このセクションでは、複数のクライアントが同じアイテムに書き込む場合の、結果的に整合性のある読み込みと、整合性のある読み込みリクエストの例を示します。

次の例では、R1 (読み取り 1) と R2 (読み取り 2) の開始前に W1 (書き込み 1) と W2 (書き込み 2) が完了しています。整合性のある読み込みの場合、R1 と R2 の両方が `color = ruby` を返します。結果的に整合性のある読み込みの場合、経過時間により、R1 と R2 は `color = red` または `color = ruby` を返すか、結果を返さないこともあります。

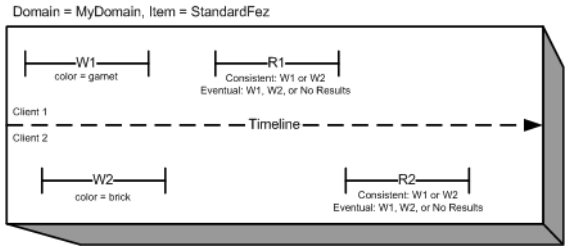


次の例では、R1 の開始前に、W2 は完了していません。このため、整合性のある読み込みと結果的に整合性のある読み込みのどちらの場合にも、R1 は `color = ruby` または `color = garnet` を返す可能性があります。また、経過時間によっては、結果的に整合性のある読み込みが結果を返さないこともあります。

整合性のある読み込みの場合、R2 は `color = garnet` を返します。結果的に整合性のある読み込みの場合、経過時間により、R2 は `color = ruby` または `color = garnet` を返すか、結果を返さないこともあります。



最後の例では、クライアント 2 は Amazon S3 が W1 の成功を返す前に W2 を実行するため、最終的な値の結果は不明です (`color = garnet` または `color = brick`)。それ以降の読み込み (整合性のある読み込みまたは結果的に整合性のある読み込み) ではどちらかの値を返す可能性があります。また、経過時間によっては、結果的に整合性のある読み込みが結果を返さないこともあります。



機能

Topics

このセクションでは、Amazon S3 の重要な特徴について説明します。

低冗長化ストレージ

データの格納に Amazon S3 の低冗長化ストレージ (RRS) オプションを使用できます。RRS は、それほど重要でない再生可能なデータを、Amazon S3 の標準ストレージの冗長性よりも低レベルの冗長性で保存することで、お客様がコストを削減できるようにします。どこか他の場所で堅牢に格納されているコンテンツを配信または共有するため、またはサムネイル、コード変換されたメディアもしくは簡単に再生可能なその他の加工データを格納するために、費用対効果および可用性の高いソリューションを提供します。RRS オプションは、複数の施設の複数のデバイス上にオブジェクトを保存し、通常のディスクドライブの 400 倍の堅牢性を提供しますが、標準的な Amazon S3 ストレージほど何度もオブジェクトをレプリケーションするわけではないため、費用対効果が高くなっています。

RRS は、年間 99.99% のオブジェクトの耐久性を提供します。この耐久性レベルは、年間平均予測オブジェクト喪失率 0.01% に該当します。

RRS の利用料金は標準的な Amazon S3 ストレージよりも割安です。料金表については、「Amazon S3 の詳細ページ」を参照してください。

詳細については、「低冗長化ストレージの使用 (p. 386)」を参照してください。

バケットポリシー

バケットポリシーを使用すると、Amazon S3 オペレーション、リクエスタ、リソース、リクエストの情報 (例: IP アドレス) を含めたさまざまな条件に基づいて、バケットとオブジェクトへのアクセスを一元的に制御できます。ポリシーは *アクセスポリシー言語* で表現され、アクセス許可を一元的に管理できます。バケットにアタッチされたアクセス許可は、そのバケット内のすべてのオブジェクトに適用されます。

会社だけでなく個人でもバケットポリシーを利用できます。会社は Amazon S3 に登録するときに、アカウントを作成します。それ以降、会社とアカウントは同義になります。アカウントは、会社 (およびその従業員) が作成した Amazon のリソースに対して経済面での責任を負います。アカウントは、バケットポリシーにアクセス許可を与えたり、さまざまな条件に基づいて従業員にアクセス許可を割り当てることができます。例えば、ユーザーに書き込みアクセスを与えるポリシーを次のように作成できます。

- 特定の S3 バケットに対して
- アカウントの社内ネットワークから
- 営業時間内
- アカウントのカスタムアプリケーションから (ユーザーエージェント文字列によって識別)

あるアプリケーションに制限付きの読み取りアクセスと書き込みアクセスを付与し、他のアプリケーションにはバケットの作成および削除を許可することもできます。1 つのバケットに複数の営業所がそれぞれの日次レポートを格納することもできます。この場合、各営業所はその営業所の IP アドレス範囲からのみ特定の名前セット (例: 「Nevada/*」、「Utah/*」) にのみ書き込むことができるようにします。

アクセス許可を個別のオブジェクトに追加 (付与) できるアクセスコントロールリスト (以下で説明) とは異なり、ポリシーはバケット内のすべて (または一部) のオブジェクトに対してアクセス許可を追加または拒否できます。1 つのリクエストで、バケット内の複数のオブジェクトのアクセス許可を設定できます。Amazon リソースネーム (ARN) やその他の値に対してワイルドカード (正規表現演算子のようなもの) を使用できるので、よく使用するプレフィックスで始まるオブジェクト群や、*html* のような拡張子で終了するオブジェクト群へのアクセスを管理できます。

バケット所有者のみが、ポリシーをバケットに関連付けることができます。ポリシーはアクセスポリシー言語で記述され、次の項目に基づいてリクエストを*許可*または*拒否*します。

- Amazon S3 バケットオペレーション (例: `PUT ?acl`) 、およびオブジェクトオペレーション (`PUT Object`、`GET Object` など)
- リクエスタ
- ポリシーに指定された条件

アカウントでは、`GetObject`、`GetObjectVersion`、`DeleteObject`、`DeleteBucket` など、特定の Amazon S3 オペレーションに基づいてアクセスをコントロールできます。

条件には、IP アドレス、CIDR 表記での IP アドレス範囲、日付、ユーザーエージェント、HTTP Referrer、トランスポート (HTTP および HTTPS) などがあります。

詳細については、「Using Bucket Policies (p. 328)」を参照してください。

AWS Identity and Access Management

例えば、IAM を Amazon S3 とともに使用すると、自分の AWS アカウントが所有する Amazon S3 バケットの特定の部分に対して個々のユーザーまたはユーザーグループに付与するアクセス権のタイプをコントロールできるようになります。

IAM の詳細については、以下のものを参照してください。

- Identity and Access Management (IAM)
- IAM 入門ガイド
- IAM を使用する

アクセスコントロールリスト

詳細については、「ACL の使用 (p. 347)」を参照してください

バージョニング

詳細については、「オブジェクトのバージョニング (p. 111)」を参照してください

オペレーション

以下は API を使用して実行する一般的なオペレーションです。

一般的なオペレーション

- バケットの作成 – オブジェクトを格納する独自のバケットを作成し、名前を指定します。
- オブジェクトへの書き込み – オブジェクトを作成または上書きしてデータを格納します。オブジェクトに書き込むときは、バケットの名前空間内で一意のキーを指定します。このとき、オブジェクトに対して必要なアクセスコントロールも指定することをお勧めします。
- オブジェクトの読み取り – データを読み戻します。HTTP または BitTorrent 経由でデータをダウンロードできます。
- オブジェクトの削除 – データの一部を削除します。
- キーのリスト作成 – いずれかのバケットに含まれているキーのリストを作成します。プレフィックスに基づいてキーのリストをフィルタできます。

この機能およびその他すべての機能については、このガイドの後半で詳しく説明します。

Amazon S3 アプリケーションプログラミングインターフェイス (API)

Amazon S3 は、プログラミング言語に依存しないアーキテクチャーとして設計されており、サポートされているインターフェイスを使用してオブジェクトを格納し、取得します。

Amazon S3 には、REST および SOAP インターフェイスが用意されています。この 2 つは似ていますが、いくつかの相違点があります。例えば、REST インターフェイスでは、メタデータは HTTP ヘッダーで返されます。4 KB (本文を含まない) 以下の HTTP リクエストしかサポートされないため、使用できるメタデータの量が制限されます。

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

REST インターフェイス

REST API は、Amazon S3 に対する HTTP インターフェイスです。REST を使用すると、標準 HTTP リクエストを使用してバケットとオブジェクトを作成、取得、および削除できます。

REST API を使用する場合、HTTP をサポートする任意のツールキットを使用できます。匿名で読み取り可能なオブジェクトであれば、ブラウザを使用して取得することもできます。

REST API は標準の HTTP ヘッダーとステータスコードを使用するため、標準のブラウザとツールキットが予期したとおりに機能します。一部のエリアでは、HTTP に機能が追加されています (例えば、アクセスコントロールをサポートするヘッダーを追加しました) 。このように新機能を追加する場合、できるだけ標準 HTTP 書式の使用法に合致するように最善を尽くしました。

SOAP インターフェイス

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

SOAP API には、ドキュメントリテラルエンコードを使用した SOAP 1.1 インターフェイスが用意されています。SOAP を使用する最も一般的なのは、WSDL をダウンロードし（「http://doc.s3.amazonaws.com/2006-03-01/AmazonS3.wsdl」にアクセス）、Apache Axis や Microsoft .NET などの SOAP ツールキットを使用してバインドを作成した後、そのバインドを使用して Amazon S3 を呼び出すコードを記述する方法です。

Amazon S3 の料金

Amazon S3 の料金は、アプリケーションのストレージ要件を考慮しなくてすむように設定されています。多くのストレージプロバイダでは、あらかじめ決まった量のストレージとネットワーク転送容量を購入する必要があります。その容量を超えると、サービスが停止されるか、高額な超過料金を支払う必要があります。その容量を超えない場合でも、全量を使用したものとして支払うことになります。

Amazon S3 では、実際に使用した分だけが請求されます。隠れた料金や超過料金はありません。つまり、開発者は、Amazon のインフラストラクチャのコストメリットを享受しながら、ビジネスの成長に合わせたコスト変動サービスを利用できるということです。

Amazon S3 にデータを格納する前に、サービスに登録し、毎月末に行われる請求の支払い方法を指定する必要があります。サービスの使用を始めるためのセットアップ料金はありません。月末に、指定した支払い方法に当月の利用料金が自動的に請求されます。

Amazon S3 ストレージの支払いについては、Amazon Simple Storage Service 製品詳細ページを参照してください。

関連サービス

Amazon S3 にロードしたデータは、用意されている他のサービスでも利用できます。ご利用頻度が高くなると思われるサービスは次のとおりです。

- Amazon Elastic Compute Cloud – このウェブサービスは、クラウド内で仮想計算リソースを提供します。詳細については、Amazon ElasticCompute Cloud の製品詳細ページを参照してください。
- Amazon Elastic MapReduce – このウェブサービスでは、ビジネス、研究者、データアナリスト、および開発者が、簡単に、そして費用対効果の高い方法で、莫大な量のデータを処理できます。これはホストされた Hadoop フレームワークを利用しています。Hadoop フレームワークは、 Amazon EC2 および Amazon S3 の、ウェブスケールのインフラストラクチャ上で稼動します。詳細については、Amazon Elastic MapReduce 製品詳細ページを参照してください。
- AWS Import/Export – AWS Import/Export では、RAID ドライブなどの記憶装置を Amazon に郵送すると、Amazon S3 へのデータ（テラバイト）のアップロードを代行します。詳細については、『AWS Import/Export Developer Guide』を参照してください。

リクエストの実行

Topics
- アクセスキーについて (p. 11)
- エンドポイントのリクエスト (p. 13)
- AWS SDK を使用したリクエストの実行 (p. 14)
- REST API を使用したリクエストの実行 (p. 49)

Amazon S3 は REST サービスです。Amazon S3 にリクエストを送信するには、REST API か、またはこの基盤である Amazon S3 REST API をラップする AWS SDK (「Sample Code and Libraries」を参照) ラッパーライブラリを使用できます。これにより、プログラミング作業も簡素化されます。

Amazon S3 とのすべてのやり取りは認証されるか匿名で行われます。認証は、アマゾン ウェブ サービス (AWS) 製品にアクセスを試みるリクエスタの ID を検証するプロセスです。認証リクエストには、リクエストの送信元を認証する署名値を含める必要があります。署名値の一部は、リクエスタの AWS アクセスキー (アクセスキー ID とシークレットアクセスキー) から生成されます。アクセスキーの取得の詳細については、「How Do I Get Security Credentials?」(『*AWS General Reference*』) を参照してください。

AWS SDK を使用している場合、指定したキーから、ライブラリによって署名が計算されます。ただし、アプリケーションで直接 REST API を呼び出す場合、署名を計算するコードを作成し、それをリクエストに追加する必要があります。

アクセスキーについて

以下のセクションでは、認証リクエストの実行に使用できるアクセスキーの種類について説明します。

AWS アカウントアクセスキー

アカウントアクセスキーを使用することで、アカウントが所有する AWS リソースにフルアクセスすることができます。アクセスキーの例を次に示します。

- アクセスキー ID (半角英数字で構成された 20 文字)。例: AKIAIOSFODNN7EXAMPLE
- シークレットアクセスキー (40 文字の文字列)。例: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

アクセスキー ID によって AWS アカウントが一意に識別されます。このアクセスキーを使用して認証リクエストを Amazon S3 に送信できます。

IAM ユーザーアクセスキー

お客様の会社用に 1 つの AWS アカウントを作成することができます。ただし、組織の AWS リソースへのアクセス権を必要とする社員が組織内に複数いる場合があります。AWS アカウントアクセスキーを共有するとセキュリティは低くなり、また、各社員に個別の AWS アカウントを作成することは実用的ではない場合があります。さらに、バケットやオブジェクトなどのリソースは別のアカウントが所有しているので、簡単に共有できません。リソースを共有するには、追加の作業でアクセス許可を付与する必要があります。

この場合、AWS Identity and Access Management (IAM) を使用して、お客様の AWS アカウントの下にユーザーと各自のアクセスキーを作成し、該当するリソースアクセス許可を付与する IAM ユーザーポリシーをアタッチします。このようなユーザーをより効率的に管理するために、IAM では、ユーザーのグループを作成し、グループ内のユーザー全員に適用されるグループレベルのアクセス許可を付与できます。

お客様が AWS で作成および管理するユーザーは、IAM ユーザーと呼ばれます。親アカウントは、ユーザーに AWS へのアクセスを許可するかどうかを制御します。IAM ユーザーが作成するリソースは、親 AWS アカウントが管理し、支払いを行います。これらの IAM ユーザーは、各自のセキュリティ認証情報を使用して Amazon S3 に認証リクエストを送信できます。お客様の AWS アカウントの下でユーザーを作成、管理する方法の詳細については、AWS Identity and Access Management 製品詳細ページを参照してください。

一時的セキュリティ認証情報

IAM では、お客様は、IAM ユーザーと各自のアクセスキーを作成できるだけでなく、任意のユーザーに一時的なセキュリティ認証情報 (一時的なアクセスキーとセキュリティトークン) を付与して、AWS サービスおよびリソースにアクセスできるようにすることもできます。また、AWS ではなく、使用しているシステムでユーザーを管理することもできます。このようなユーザーはフェデレーションユーザーと呼ばれます。また、このユーザーには、AWS リソースにアクセスするために作成したアプリケーションも含まれます。

IAM には、一時的なセキュリティ認証情報をリクエストするための AWS Security Token Service API が用意されています。一時的なセキュリティ認証情報をリクエストするには、AWS STS API または AWS SDK を使用できます。この API は一時的なセキュリティ認証情報 (アクセスキー ID およびシークレットアクセスキー) とセキュリティトークンを返します。これらの認証情報は、リクエスト時に指定した有効期間の間のみ有効です。アクセスキー ID およびシークレットキーの使用方法は、AWS アカウントまたは IAM ユーザーのアクセスキーを使用してリクエストを送信するときの使用方法と同様です。また、Amazon S3 に送信する各リクエストにトークンを含める必要があります。

IAM ユーザーは、これらの一時的なセキュリティ認証情報をリクエストした後、自分で使用することも、フェデレーションユーザーまたはアプリケーションに渡すこともできます。フェデレーションユーザー用に一時的なセキュリティ認証情報をリクエストする場合、ユーザー名と、その一時的なセキュリティ認証情報に関連付けるアクセス許可を定義した IAM ポリシーを指定する必要があります。フェデレーションユーザーに、一時的な認証情報をリクエストした親 IAM ユーザーよりも多くの権限が付与されることはありません。

これらの一時的なセキュリティ認証情報は、Amazon S3 にリクエストするときに使用できます。API ライブラリによって、これらの認証情報を使用して必要な署名値が計算されて、リクエストが認証されます。失効した認証情報を使用してリクエストを送信すると、 Amazon S3 はリクエストを拒否します。

REST API リクエストで一時的なセキュリティ認証情報を使用してリクエストに署名する方法については、「Signing and Authenticating REST Requests (p. 52)」を参照してください。AWS SDK を使用し

てリクエストを送信する方法については、「AWS SDK を使用したリクエストの実行 (p. 14)」を参照してください。

IAM による一時的なセキュリティ認証情報のサポートについては、『IAM を使用する』の「*Granting Temporary Access to Your AWS Resources*」を参照してください。

Amazon S3 リソースにアクセスする際のセキュリティを高めるには、バケットポリシーを設定して多要素認証 (MFA) を要求することができます。詳細については、「MFA 認証を要求するバケットポリシーの追加 (p. 335)」を参照してください。Amazon S3 リソースにアクセスするために MFA を要求した後に、これらのリソースにアクセスできる唯一の方法は、MFA キーで作成された一時的な認証情報を提供することです。詳細については、『*IAM を使用する*』の「AWS Multi-Factor Authentication」(詳細ページ) と「Configuring MFA-Protected API Access」を参照してください。

エンドポイントのリクエスト

サービスの定義済みエンドポイントに対して REST リクエストを送信します。「*Amazon Web Services General Reference*」には、AWS サービスとそれに対応するエンドポイントが記載されています。Amazon S3 のリージョンとエンドポイントについては、『AWS General Reference』の「Regions and Endpoints」を参照してください。

AWS SDK を使用したリクエストの実行

Topics

認証リクエストを Amazon S3 に送信するには、AWS SDK を使用するか、アプリケーションで直接 REST API を呼び出します。AWS SDK API は、指定された認証情報を使用して認証用の署名を計算します。アプリケーションで直接 REST API を使用する場合、リクエストの認証に使用する署名の計算に必要となるコードを記述する必要があります。利用可能な AWS SDK のリストについては、「Sample Code & Libraries」を参照してください。

AWS アカウントまたは IAM ユーザーの認証情報を使用したリクエストの実行

Topics

AWS アカウントまたは IAM ユーザーのセキュリティ認証情報を使用して、認証リクエストを Amazon S3 に送信できます。このセクションでは、AWS SDK for Java、AWS SDK for .NET、および AWS SDK for PHP を使用して認証リクエストを送信する方法の例を示します。利用可能な AWS SDK のリストについては、「Sample Code & Libraries」を参照してください。

AWS アカウントまたは IAM ユーザーの認証情報を使用したリクエストの実行 – AWS SDK for Java

以下のタスクは、Java クラスで AWS アカウントの認証情報または IAM ユーザーの認証情報を使用して、認証リクエストを送信する手順を示しています。

AWS アカウントまたは IAM ユーザーの認証情報を使用したリクエストの実行

1	AWS アカウントまたは IAM ユーザーの認証情報 (アクセスキー ID、シークレットアクセスキー) を指定して、`AmazonS3Client` クラスのインスタンスを作成します。
2	`AmazonS3Client` メソッドの 1 つを実行して、 &S3 にリクエストを送信します。クライアントにより、認証情報から必要な署名値が生成され、Amazon S3 に送信されるリクエストにその署名値が追加されます。

以下の Java コード例は、前述のタスクを実装したものです。

```
AWSCredentials myCredentials = new BasicAWSCredentials(
                                  myAccessKeyID, mySecretKey);
AmazonS3 s3client = new AmazonS3Client(myCredentials);

// Send sample request (list objects in a given bucket).
ObjectListing objectListing = s3client.listObjects(new
     ListObjectsRequest().withBucketName(bucketName));
```

Note

AmazonS3Client クラスは、セキュリティ認証情報を指定せずに作成できます。このクライアントを使用して送信されるリクエストは、署名なしの匿名リクエストです。公開されていないリソースに対して匿名リクエストを送信すると、Amazon S3 はエラーを返します。

使用例については、「Amazon S3 オブジェクトの使用 (p. 107)」および「Amazon S3 バケットの使用 (p. 87)」を参照してください。これらの例をテストするには、AWS アカウントまたは IAM ユーザーの認証情報を使用します。

例えば、バケットのオブジェクトキーをすべて表示する方法については、「AWS SDK for Java を使用したキーのリスト作成 (p. 243)」を参照してください。

関連リソース

- AWS dynamo と Explorer の使用 (p. 475)

AWS アカウントまたは IAM ユーザーの認証情報を使用したリクエストの実行 – AWS SDK for .NET

以下のタスクは、.NET クラスで AWS アカウントまたは IAM ユーザーの認証情報を使用して、認証リクエストを送信する手順を示しています。

AWS アカウントまたは IAM ユーザーの認証情報を使用したリクエストの実行

1	AWS アカウントまたは IAM ユーザーの認証情報 (アクセスキー ID、シークレットアクセスキー) を指定して、`AmazonS3Client` クラスのインスタンスを作成します。
2	`AmazonS3Client` メソッドの 1 つを実行して、 &S3 にリクエストを送信します。クライアントにより、認証情報から必要な署名が生成され、Amazon S3 に送信されるリクエストにその署名値が追加されます。

以下の C# コード例は、前述のタスクを実装したものです。この例を使用するには、指定された箇所を自身のアクセスキー ID とシークレットアクセスキーに置き換えます。

```
BasicAWSCredentials basicCredentials =
               new BasicAWSCredentials("*** Access Key ID ***",
                                       "*** Secret Access Key ***");
AmazonS3Client s3Client = new AmazonS3Client(basicCredentials);

// Send sample request (for example, list buckets).
var response = s3Client.ListBuckets();
```

Note

`AmazonS3Client` クライアントは、セキュリティ認証情報を指定せずに作成できます。このクライアントを使用して送信されるリクエストは、署名なしの匿名リクエストです。公開されていないリソースに対して匿名リクエストを送信すると、Amazon S3 はエラーを返します。

使用例については、「Amazon S3 オブジェクトの使用 (p. 107)」および「Amazon S3 バケットの使用 (p. 87)」を参照してください。これらの例をテストするには、AWS アカウントまたは IAM ユーザーの認証情報を使用します。

例えば、バケットのオブジェクトキーをすべて表示する方法については、「AWS SDK for .NET を使用したキーのリスト作成 (p. 246)」を参照してください。

関連リソース

- AWS dynamo と Explorer の使用 (p. 475)

AWS アカウントまたは IAM ユーザーの認証情報を使用したリクエストの実行 – AWS SDK for PHP

このトピックでは、AWS SDK for PHP のクラスで AWS アカウントまたは IAM ユーザーの認証情報を使用して、認証リクエストを送信する手順を示します。

Note

このトピックでは、既に AWS SDK for PHP の使用と PHP サンプルの実行 (p. 479) の説明が実行されていて、AWS SDK for PHP が正しくインストールされていることを前提としています。

AWS アカウントまたは IAM ユーザーの認証情報を使用したリクエストの実行

1	自分の AWS 認証情報または IAM ユーザー認証情報と共に Aws\S3\S3Client クラスの factory() メソッドを使用して、Amazon S3 クライアントのインスタンスを作成します。
2	`Aws\S3\S3Client` メソッドの 1 つを実行して、Amazon S3 にリクエストを送信します。例えば、Aws\S3\S3Client::listBuckets() メソッドを使用して、アカウントの全バケットをリストするリクエストを送信できます。クライアント API により、認証情報から必要な署名が生成され、Amazon S3 に送信されるリクエストにその署名が追加されます。

以下の PHP コード例は前述のタスクを実装したもので、セキュリティ認証情報を使用してアカウントの全バケットをリストするリクエストをクライアントが実行する方法を示しています。

```
use Aws\S3\S3Client;

// Instantiate the S3 client with your AWS credentials
$s3 = S3Client::factory(array(
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));

$result = $s3->listBuckets();
```

Note

前述の例に示したようにアクセスキーを指定することもキーを省略することもできます。キーを省略できるのは、`AWS_ACCESS_KEY_ID` および `AWS_SECRET_ACCESS_KEY` 環境変数から取得した Amazon EC2 用 AWS Identity and Access Management (IAM) 役割のインスタンスまたは認証情報を使用する場合です。

使用例については、「Amazon S3 オブジェクトの使用 (p. 107)」および「Amazon S3 バケットの使用 (p. 87)」を参照してください。これらの例をテストするには、AWS アカウントまたは IAM ユーザーの認証情報を使用します。

バケット内のオブジェクトキーをリストする例については、「AWS SDK for PHP を使用したキーのリスト作成 (p. 250)」を参照してください。

関連リソース

- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client クラス」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::factory() メソッド」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::listBuckets() メソッド」

- 「AWS SDK for PHP – Amazon S3」
- 「AWS SDK for PHP」のドキュメント

AWS アカウントまたは IAM ユーザーの認証情報を使用したリクエストの実行 – AWS SDK for Ruby

以下のタスクは、AWS SDK for Ruby で AWS アカウントの認証情報または IAM ユーザーの認証情報を使用して、認証リクエストを送信する手順を示しています。

AWS アカウントまたは IAM ユーザーの認証情報を使用したリクエストの実行

1	AWS アカウントまたは IAM ユーザーの認証情報 (アクセスキー ID、シークレットアクセスキー) を指定して、`AWS::S3` クラスのインスタンスを作成します。
2	`AWS::S3` の `buckets` メソッドを使用してバケット内のオブジェクトを列挙することで、Amazon S3 に対するリクエストを実行します。クライアントにより、認証情報から必要な署名値が生成され、Amazon S3 に送信されるリクエストにその署名値が追加されます。

以下の Ruby コード例は、前述のタスクの例です。

```
# Get an instance of the S3 interface using the specified credentials configur
ation.
s3 = AWS::S3.new(
  :access_key_id => my_access_key_id,
  :secret_access_key => my_secret_key)

# Get a list of all object keys in a bucket.
bucket = s3.buckets[bucket_name].objects.collect(&:key)
puts bucket
```

Note

`AWS:S3` クライアントは、セキュリティ認証情報を指定せずに作成できます。このクライアントを使用して送信されるリクエストは、署名なしの匿名リクエストです。公開されていないリソースに対して匿名リクエストを送信すると、Amazon S3 はエラーを返します。

使用例については、「Amazon S3 オブジェクトの使用 (p. 107)」および「Amazon S3 バケットの使用 (p. 87)」を参照してください。これらの例をテストするには、AWS アカウントまたは IAM ユーザーの認証情報を使用します。

IAM ユーザーの一時的な認証情報を使用したリクエストの実行

Topics

AWS アカウントまたは IAM ユーザーは、一時的なセキュリティ認証情報をリクエストし、そのセキュリティ認証情報を使用して、認証リクエストを Amazon S3 に送信できます。このセクションでは、AWS SDK for Java、AWS SDK for .NET、および AWS SDK for PHP を使用して、一時的なセキュリティ認証情報を取得する方法と、その認証情報を使用して Amazon S3 にリクエストを認証してもらう方法について説明します。

IAM ユーザーの一時的な認証情報を使用したリクエストの実行 – AWS SDK for Java

IAM ユーザーまたは AWS アカウントは、AWS SDK for Java を使用して一時的なセキュリティ認証情報をリクエストし (「リクエストの実行 (p. 11)」を参照)、その認証情報を使用して、Amazon S3 にアクセスできます。これらの認証情報は、セッションの有効期間が過ぎると失効します。デフォルトでは、セッションの有効期間は 1 時間です。 IAM ユーザーの認証情報を使用する場合、一時的なセキュリティ認証情報をリクエストするときに、有効期間を 1 ~ 36 時間の間で指定できます。

IAM ユーザーの一時的なセキュリティ認証情報を使用したリクエストの実行

1	認証情報を指定して、AWS Security Token Service クライアントのインスタンスである `AWSSecurityTokenServiceClient` を作成します。
2	前の手順で作成した STS クライアントの `GetSessionToken` メソッドを呼び出して、セッションを開始します。`GetSessionTokenRequest` オブジェクトを使用して、このメソッドにセッション情報を指定します。 このメソッドは一時的なセキュリティ認証情報を返します。
3	Amazon S3 クライアントに一時的なセキュリティ認証情報を提供できるように、この認証情報を `BasicSessionCredentials` オブジェクトのインスタンスにパッケージ化します。
4	一時的なセキュリティ認証情報を渡して、`AmazonS3Client` クラスのインスタンスを作成します。 このクライアントを使用してリクエストを Amazon S3 に送信します。失効した認証情報を使用してリクエストを送信すると、Amazon S3 はエラーを返します。

以下の Java コード例は、前述のタスクを実装したものです。

```
// In real applications, the following code is part of your trusted code. It
has
// your security credentials you use to obtain temporary security credentials.
AWSCredentials credentials =
                 new BasicAWSCredentials("*** Access Key ID ***",
```

```
                                            "*** Secret Key ***");
AWSSecurityTokenServiceClient stsClient =
                        new AWSSecurityTokenServiceClient(credentials);

//
// Manually start a session.
GetSessionTokenRequest getSessionTokenRequest = new GetSessionTokenRequest();
// Following duration can be set only if temporary credentials are requested
by an IAM user.
getSessionTokenRequest.setDurationSeconds(7200);

GetSessionTokenResult sessionTokenResult =
                           stsClient.getSessionToken(getSessionTokenRequest);
Credentials sessionCredentials = sessionTokenResult.getCredentials();

// Package the temporary security credentials as
// a BasicSessionCredentials object, for an Amazon S3 client object to use.
BasicSessionCredentials basicSessionCredentials =
              new BasicSessionCredentials(sessionCredentials.getAccessKeyId(),

                                      sessionCredentials.getSecretAccessKey(),

                                       sessionCredentials.getSessionToken());

// The following will be part of your less trusted code. You provide temporary
 security
// credentials so it can send authenticated requests to Amazon S3.
// Create Amazon S3 client by passing in the basicSessionCredentials object.
AmazonS3Client s3 = new AmazonS3Client(basicSessionCredentials);

// Test. For example, get object keys in a bucket.
ObjectListing objects = s3.listObjects(bucketName);
```

Example

Note

AWS アカウント認証情報を使用して一時的なセキュリティ認証情報を取得する場合、取得した一時的なセキュリティ認証情報は 1 時間だけ有効です。セッションの有効期間を指定できるのは、IAM ユーザー認証情報を使用してセッションをリクエストする場合に限ります。

次の Java コード例では、指定したバケット内のオブジェクトキーをリストします。このコード例では、わかりやすいように、デフォルトの 1 時間のセッションの一時的なセキュリティ認証情報を取得し、それを使用して認証リクエストを Amazon S3 に送信します。

IAM ユーザーの認証情報を使用してサンプルをテストする場合、AWS アカウントに IAM ユーザーを作成する必要があります。IAM ユーザーを作成する方法については、『IAM 入門ガイド』の「*Set Up a Group, Grant Permissions, and Add Users*」を参照してください。

```
import java.io.IOException;
import com.amazonaws.auth.BasicSessionCredentials;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.securitytoken.AWSSecurityTokenServiceClient;
import com.amazonaws.services.securitytoken.model.Credentials;
import com.amazonaws.services.securitytoken.model.GetSessionTokenRequest;
import com.amazonaws.services.securitytoken.model.GetSessionTokenResult;
import com.amazonaws.services.s3.model.ObjectListing;

public class S3Sample {
 private static String bucketName = "*** Provide bucket name ***";

    public static void main(String[] args) throws IOException {
          PropertiesCredentials credentials = new PropertiesCredentials(
             S3Sample.class.getResourceAsStream("AwsCredentials.properties"));


        AWSSecurityTokenServiceClient stsClient =
                              new AWSSecurityTokenServiceClient(credentials);


        //
        // Start a session.
        GetSessionTokenRequest getSessionTokenRequest =
                                             new GetSessionTokenRequest();

        GetSessionTokenResult sessionTokenResult =
                           stsClient.getSessionToken(getSessionTokenRequest);

        Credentials sessionCredentials = sessionTokenResult.getCredentials();
        System.out.println("Session Credentials: "
                                             + sessionCredentials.toString());



        // Package the session credentials as a BasicSessionCredentials
        // object for an S3 client object to use.
        BasicSessionCredentials basicSessionCredentials =
             new BasicSessionCredentials(sessionCredentials.getAccessKeyId(),
```

```
                                    sessionCredentials.getSecretAccessKey(),
                                    sessionCredentials.getSessionToken());
        AmazonS3Client s3 = new AmazonS3Client(basicSessionCredentials);

        // Test. For example, get object keys for a given bucket.
        ObjectListing objects = s3.listObjects(bucketName);
        System.out.println("No. of Objects = " +
                                        objects.getObjectSummaries().size());

    }
}
```

関連リソース

- AWS dynamo と Explorer の使用 (p. 475)

IAM ユーザーの一時的な認証情報を使用したリクエストの実行 – AWS SDK for .NET

IAM ユーザーまたは AWS アカウントは、AWS SDK for .NET を使用して一時的なセキュリティ認証情報をリクエストし (「リクエストの実行 (p. 11)」を参照)、その認証情報を使用して、Amazon S3 にアクセスできます。これらの認証情報は、セッションの有効期間が過ぎると失効します。デフォルトでは、セッションの有効期間は 1 時間です。IAM ユーザーの認証情報を使用する場合、一時的なセキュリティ認証情報をリクエストするときに、有効期間を 1 ~ 36 時間の間で指定できます。

IAM ユーザーの一時的なセキュリティ認証情報を使用したリクエストの実行

1	認証情報を指定して、AWS Security Token Service クライアントのインスタンスである `AmazonSecurityTokenServiceClient` を作成します。
2	前の手順で作成した STS クライアントの `GetSessionToken` メソッドを呼び出して、セッションを開始します。`GetSessionTokenRequest` オブジェクトを使用して、このメソッドにセッション情報を指定します。 このメソッドは一時的なセキュリティ認証情報を返します。
3	`SessionAWSCredentials` オブジェクトのインスタンスに一時的なセキュリティ認証情報をパッケージ化します。このオブジェクトを使用して、 Amazon S3 クライアントに対して一時的なセキュリティ認証情報を提供します。
4	一時的なセキュリティ認証情報を渡して、`AmazonS3Client` クラスのインスタンスを作成します。 このクライアントを使用してリクエストを Amazon S3 に送信します。失効した認証情報を使用してリクエストを送信すると、Amazon S3 はエラーを返します。

以下の C# コード例は、前述のタスクを実装したものです。

```
// In real applications, the following code is part of your trusted code. It
has
// your security credentials you use to obtain temporary security credentials.
AmazonSecurityTokenServiceConfig config = new AmazonSecurityTokenServiceConfig();

 AmazonSecurityTokenServiceClient stsClient =
```

```
          new AmazonSecurityTokenServiceClient("*** Access Key ID ***",
                                               "*** Secret Access Key ***",
                                               config);

GetSessionTokenRequest getSessionTokenRequest = new GetSessionTokenRequest();
// Following duration can be set only if temporary credentials are requested
by an IAM user.
getSessionTokenRequest.DurationSeconds = 7200; // seconds.
Credentials credentials =
    stsClient.GetSessionToken(getSessionTokenRequest).GetSessionTokenResult.Cre
dentials;

SessionAWSCredentials sessionCredentials =
                          new SessionAWSCredentials(credentials.AccessKeyId,
                                                    credentials.SecretAccessKey,

                                                      credentials.SessionToken);

// The following will be part of your less trusted code. You provide temporary
 security
// credentials so it can send authenticated requests to Amazon S3.
// Create Amazon S3 client by passing in the basicSessionCredentials object.
AmazonS3Client s3Client = new AmazonS3Client(sessionCredentials);

// Test. For example, send request to list object key in a bucket.
var response = s3Client.ListObjects(bucketName);
```

Example

Note

AWS アカウントのセキュリティ認証情報を使用して一時的なセキュリティ認証情報を取得する場合、取得した一時的なセキュリティ認証情報は 1 時間だけ有効です。セッションの有効期間を指定できるのは、IAM ユーザー認証情報を使用してセッションをリクエストする場合に限ります。

次の C# コード例では、指定したバケット内のオブジェクトキーをリストします。このコード例では、わかりやすいように、デフォルトの 1 時間のセッションの一時的なセキュリティ認証情報を取得し、それを使用して認証リクエストを Amazon S3 に送信します。

IAM ユーザーの認証情報を使用してサンプルをテストする場合、AWS アカウントに IAM ユーザーを作成する必要があります。IAM ユーザーを作成する方法については、『IAM 入門ガイド』の「*Set Up a Group, Grant Permissions, and Add Users*」を参照してください。

```
using System;
using System.Configuration;
using System.Collections.Specialized;
using Amazon.S3;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;
using Amazon.Runtime;
using Amazon.S3.Model;
using System.Collections.Generic;

namespace s3.amazon.com.docsamples.listingkeys
{
    class TempCred_ExplicitSessionStart
    {
        static string bucketName = "*** Provide bucket name ***";
        static AmazonS3 client;

        public static void Main(string[] args)
        {
            NameValueCollection appConfig = ConfigurationManager.AppSettings;
            string accessKeyID = appConfig["AWSAccessKey"];
            string secretAccessKeyID = appConfig["AWSSecretKey"];
            try
            {
                Console.WriteLine("Listing objects stored in a bucket");
                SessionAWSCredentials tempCredentials =
                     GetTemporaryCredentials(accessKeyID, secretAccessKeyID);

                // Create client by providing temporary security credentials.
                AmazonS3Client s3Client = new AmazonS3Client(tempCredentials);


                ListObjectsRequest listObjectRequest =
                                    new ListObjectsRequest();
                listObjectRequest.BucketName = bucketName;

                // Send request to Amazon S3.
             ListObjectsResponse response = s3Client.ListObjects(listObjectRe
quest);
                List<S3Object> objects = response.S3Objects;
```

```
                Console.WriteLine("Object count = {0}", objects.Count);

                Console.WriteLine("Press any key to continue...");
                Console.ReadKey();
            }
            catch (AmazonS3Exception s3Exception)
            {
                Console.WriteLine(s3Exception.Message,
                                  s3Exception.InnerException);
            }
            catch (AmazonSecurityTokenServiceException stsException)
            {
                Console.WriteLine(stsException.Message,
                                 stsException.InnerException);
            }
        }

        private static SessionAWSCredentials GetTemporaryCredentials(
                         string accessKeyId, string secretAccessKeyId)
        {
            AmazonSecurityTokenServiceClient stsClient =
                new AmazonSecurityTokenServiceClient(accessKeyId,
                                                     secretAccessKeyId);

            GetSessionTokenRequest getSessionTokenRequest =
                                               new GetSessionTokenRequest();
            getSessionTokenRequest.DurationSeconds = 7200; // seconds

            GetSessionTokenResponse sessionTokenResponse =
                          stsClient.GetSessionToken(getSessionTokenRequest);
            GetSessionTokenResult sessionTokenResult =
                                   sessionTokenResponse.GetSessionTokenResult;

            Credentials credentials = sessionTokenResult.Credentials;

            SessionAWSCredentials sessionCredentials = new
       SessionAWSCredentials.CreateStaticCredentials(credentials.AccessKeyId,
                                                  credentials.SecretAccessKey,

                                                    credentials.SessionToken);


            return sessionCredentials;
        }
    }
}
```

関連リソース

- AWS dynamo と Explorer の使用 (p. 475)

AWS アカウントまたは IAM ユーザーの一時的な認証情報を使用したリクエストの実行 – AWS SDK for PHP

このトピックでは、AWS SDK for PHP のクラスを使用して一時的セキュリティ認証情報をリクエストし、その認証情報を使用して Amazon S3 にアクセスする手順を示します。

Note

このトピックでは、既に AWS SDK for PHP の使用と PHP サンプルの実行 (p. 479) の説明が実行されていて、AWS SDK for PHP が正しくインストールされていることを前提としています。

IAM ユーザーまたは AWS アカウントは、AWS SDK for PHP を使用して一時的なセキュリティ認証情報をリクエストし（「リクエストの実行 (p. 11)」を参照）、その認証情報を使用して、Amazon S3 にアクセスできます。これらの認証情報は、セッションの有効期間が失効すると同時に失効します。デフォルトでは、セッションの有効期間は 1 時間です。IAM ユーザーの認証情報を使用する場合、一時的なセキュリティ認証情報をリクエストするときに、有効期間を 1 ~ 36 時間の間で指定できます。

AWS アカウントまたは IAM ユーザーの一時的なセキュリティ認証情報を使用したリクエストの実行

1	Aws\Sts\StsClient クラスの factory() メソッドを使用して、AWS Security Token Service（AWS STS）クライアントのインスタンスを作成します。
2	Aws\Sts\StsClient::getSessionToken() メソッドを使用してセッションを開始します。 このメソッドは一時的なセキュリティ認証情報を返します。
3	前のステップで取得した一時的なセキュリティ認証情報と共に Aws\S3\S3Client クラスの factory() メソッドを使用して、Amazon S3 クライアントのインスタンスを作成します。 呼び出す S3Client クラスのメソッドはいずれも、一時的なセキュリティ認証情報を使用して、認証リクエストを Amazon S3 に送信します。

以下の PHP コード例は、一時的なセキュリティ認証情報をリクエストし、その認証情報を使用して Amazon S3 にアクセスする方法を示しています。

```
use Aws\Sts\StsClient;
use Aws\S3\S3Client;

// In real applications, the following code is part of your trusted code.
// It has your security credentials that you use to obtain temporary
// security credentials.
$sts = StsClient::factory(array(
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));

$result = $sts->getSessionToken();

// The following will be part of your less trusted code. You provide temporary
// security credentials so it can send authenticated requests to Amazon S3.
// Create an Amazon S3 client using temporary security credentials.
$credentials = $result->get('Credentials');
$s3 = S3Client::factory(array(
    'key'    => $credentials['AccessKeyId'],
    'secret' => $credentials['SecretAccessKey'],
    'token'  => $credentials['SessionToken']
));

$result = $s3->listBuckets();
```

Note

AWS アカウントのセキュリティ認証情報を使用して一時的なセキュリティ認証情報を取得する場合、取得した一時的なセキュリティ認証情報は 1 時間だけ有効です。セッションの有効期間を指定できるのは、IAM ユーザー認証情報を使用してセッションをリクエストする場合に限ります。

Example （一時的なセキュリティ認証情報を使用した Amazon S3 リクエストの実行）

以下の PHP コード例では、一時的なセキュリティ認証情報を使用して、指定したバケットのオブジェクトキーをリストしています。このコード例では、デフォルトの 1 時間のセッションの一時的なセキュリティ認証情報を取得し、その認証情報を使用して認証リクエストを Amazon S3 に送信しています。PHP 例の実行については、このガイド内の「PHP サンプルの実行 (p. 480)」を参照してください。

IAM ユーザーの認証情報を使用してこのサンプルをテストする場合、AWS アカウントに IAM ユーザーを作成する必要があります。IAM ユーザーを作成する方法については、『*IAM 入門ガイド*』の「Set Up a Group, Grant Permissions, and Add Users」を参照してください。IAM ユーザー認証情報を使用してセッションをリクエストする場合のセッションの有効期間を設定する例については、「フェデレーションユーザーの一時的な認証情報を使用したリクエストの実行 – AWS SDK for PHP (p. 41)」を参照してください。

```
<?php

// Include the AWS SDK using the Composer autoloader.
require 'vendor/autoload.php';

use Aws\Sts\StsClient;
use Aws\S3\S3Client;
use Aws\S3\Exception\S3Exception;

$bucket = '*** Your Bucket Name ***';

$sts = StsClient::factory(array(
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));

$credentials = $sts->getSessionToken()->get('Credentials');
$s3 = S3Client::factory(array(
    'key'    => $credentials['AccessKeyId'],
    'secret' => $credentials['SecretAccessKey'],
    'token'  => $credentials['SessionToken']
));

try {
    $objects = $s3->getIterator('ListObjects', array(
        'Bucket' => $bucket
    ));

    echo "Keys retrieved!\n";
    foreach ($objects as $object) {
        echo $object['Key'] . "\n";
    }
} catch (S3Exception $e) {
    echo $e->getMessage() . "\n";
}
```

関連リソース

- 「AWS SDK for PHP for Amazon S3 の Aws\Sts\StsClient クラス」
- 「AWS SDK for PHP for Amazon S3 の Aws\Sts\StsClient::factory() メソッド」
- 「AWS SDK for PHP for Amazon S3 の Aws\Sts\StsClient::getSessionToken メソッド」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client クラス」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::factory() メソッド」
- 「AWS SDK for PHP – Amazon S3」
- 「AWS SDK for PHP」のドキュメント

IAM ユーザーの一時的な認証情報を使用したリクエストの実行 – AWS SDK for Ruby

IAM ユーザーまたは AWS アカウントは、AWS SDK for Ruby を使用して一時的なセキュリティ認証情報をリクエストし (「リクエストの実行 (p. 11)」を参照)、その認証情報を使用して、Amazon S3 にアクセスできます。これらの認証情報は、セッションの有効期間が過ぎると失効します。デフォルトでは、セッションの有効期間は 1 時間です。IAM ユーザーの認証情報を使用する場合、一時的なセキュリティ認証情報をリクエストするときに、有効期間を 1 ~ 36 時間の間で指定できます。

IAM ユーザーの一時的なセキュリティ認証情報を使用したリクエストの実行

1	認証情報を指定して、AWS Security Token Service クライアントのインスタンスである `AWS::STS::Session` を作成します。
2	前の手順で作成した STS クライアントの `new_session` メソッドを呼び出して、セッションを開始します。`GetSessionTokenRequest` オブジェクトを使用して、このメソッドにセッション情報を指定します。 このメソッドは一時的なセキュリティ認証情報を返します。
3	一時的なセキュリティ認証情報で渡して、`AWS::S3` クラスの新しいインスタンスで一時的な認証情報を使用します。 このクライアントを使用してリクエストを Amazon S3 に送信します。失効した認証情報を使用してリクエストを送信すると、Amazon S3 はエラーを返します。

以下の Ruby コード例は、前述のタスクを実装したものです。

```
# Start a session.
# In real applications, the following code is part of your trusted code. It has

# your security credentials that you use to obtain temporary security creden
tials.

sts = AWS::STS.new(
  :access_key_id => my_access_key_id,
  :secret_access_key => my_secret_key)

session = sts.new_session()
puts "Session expires at: #{session.expires_at.to_s}"

# Get an instance of the S3 interface using the session credentials.
s3 = AWS::S3.new(session.credentials)

# Get a list of all object keys in a bucket.
bucket = s3.buckets[bucket_name].objects.collect(&:key)
```

Example

Note

AWS アカウントのセキュリティ認証情報を使用して一時的なセキュリティ認証情報を取得する場合、取得した一時的なセキュリティ認証情報は 1 時間だけ有効です。セッションの有効期間を指定できるのは、IAM ユーザー認証情報を使用してセッションをリクエストする場合に限ります。

次の Ruby コード例では、指定したバケット内のオブジェクトキーをリストします。このコード例では、わかりやすいように、デフォルトの 1 時間のセッションの一時的なセキュリティ認証情報を取得し、それを使用して認証リクエストを Amazon S3 に送信します。

IAM ユーザーの認証情報を使用してサンプルをテストする場合、AWS アカウントに IAM ユーザーを作成する必要があります。IAM ユーザーを作成する方法については、『IAM 入門ガイド』の「*Set Up a Group, Grant Permissions, and Add Users*」を参照してください。

```
require 'rubygems'
require 'aws-sdk'

# In real applications, the following code is part of your trusted code. It has

# your security credentials you use to obtain temporary security credentials.

my_access_key_id = '***Provide Access Key ID***'
my_secret_key = '***Provide Secret Key***'
bucket_name = '*** Provide bucket name ***'

AWS.config({
  :access_key_id => my_access_key_id,
  :secret_access_key => my_secret_key
})

# Start a session.
sts = AWS::STS.new()
session = sts.new_session()
puts "Session expires at: #{session.expires_at.to_s}"

# get an instance of the S3 interface using the session credentials
s3 = AWS::S3.new(session.credentials)

# get a list of all object keys in a bucket
bucket = s3.buckets[bucket_name].objects.collect(&:key)
puts bucket
```

フェデレーションユーザーの一時的な認証情報を使用したリクエストの実行

Topics

一時的なセキュリティ認証情報をリクエストし、AWS リソースにアクセスする必要があるフェデレーションユーザーまたはアプリケーションにその認証情報を提供できます。このセクションでは、AWS SDK を使用して、フェデレーションユーザーまたはアプリケーションのために一時的なセキュリティ認証情報を取得し、その認証情報を使用して認証リクエストを Amazon S3 に送信する方法の例について説明します。利用可能な AWS SDK のリストについては、「Sample Code & Libraries」を参照してください。

Note

AWS アカウントと IAM ユーザーのいずれも、フェデレーションユーザーの一時的なセキュリティ認証情報をリクエストできます。ただし、より高度なセキュリティのためには、必要な権限を持つ IAM ユーザーのみが、この一時的な認証情報をリクエストできるようにすることをお勧めします。こうすることで、フェデレーションユーザーに付与される権限が、リクエストを送信した IAM ユーザーの権限以下となります。アプリケーションによっては、フェデレーションユーザーおよびアプリケーションに一時的なセキュリティ認証情報を付与することを唯一の目的として、特定の権限を持つ IAM ユーザーを作成する方が適切な場合もあります。

フェデレーションユーザーの一時的な認証情報を使用したリクエストの実行 – AWS SDK for Java

フェデレーションユーザーおよびアプリケーションから AWS リソースへのアクセスのリクエストが認証されて送信されるように、フェデレーションユーザーおよびアプリケーションに一時的なセキュリティ認証情報を提供することができます (「リクエストの実行 (p. 11)」を参照)。IAM サービスからこのような一時的な認証情報をリクエストする場合、ユーザー名と、付与するリソースアクセス許可を示す IAM ポリシーを指定する必要があります。デフォルトでは、セッションの有効期間は 1 時間です。ただし、IAM ユーザー認証情報を使用して一時的な認証情報をリクエストする場合、フェデレーションユーザーおよびアプリケーション用の一時的なセキュリティ認証情報をリクエストするときに、別の有効期間値を明示的に設定できます。

Note

フェデレーションユーザーおよびアプリケーション用の一時的なセキュリティ認証情報をリクエストするには、より高度なセキュリティのために、必要なアクセス許可のみを持つ専用の IAM ユーザーを使用することもできます。作成した一時ユーザーに、一時的なセキュリティ認証情報をリクエストした IAM ユーザーより多くの権限が付与されることはありません。詳細については、「AWS Identity and Access Management FAQs」を参照してください。

フェデレーションユーザーの一時的なセキュリティ認証情報を使用したリクエストの実行

1	自分のセキュリティ認証情報を指定して、AWS Security Token Service クライアントのインスタンスである `AWSSecurityTokenServiceClient` を作成します。
2	前の手順で作成した STS クライアントの `getFederationToken` メソッドを呼び出して、セッションを開始します。 一時的な認証情報にアタッチするユーザー名と IAM ポリシーなどのセッション情報を指定する必要があります。 このメソッドは一時的なセキュリティ認証情報を返します。
3	`BasicSessionCredentials` オブジェクトのインスタンスに一時的なセキュリティ認証情報をパッケージ化します。このオブジェクトを使用して、Amazon S3 クライアントに対して一時的なセキュリティ認証情報を提供します。
4	一時的なセキュリティ認証情報を渡して、`AmazonS3Client` クラスのインスタンスを作成します。 このクライアントを使用してリクエストを Amazon S3 に送信します。失効した認証情報を使用してリクエストを送信すると、Amazon S3 はエラーを返します。

以下の Java コード例は、前述のタスクを実装したものです。

```
// In real applications, the following code is part of your trusted code. It
has
// your security credentials you use to obtain temporary security credentials.
AWSCredentials credentials =
          new BasicAWSCredentials("*** Access Key ID ***",
                                  "*** Secret Key ***");
AWSSecurityTokenServiceClient stsClient =
                        new AWSSecurityTokenServiceClient(credentials);

GetFederationTokenRequest getFederationTokenRequest =
                        new GetFederationTokenRequest();
getFederationTokenRequest.setDurationSeconds(7200);
getFederationTokenRequest.setName("User1");

// Define the policy and add to the request.
Policy policy = new Policy();
// Define the policy here.
// Add the policy to the request.
getFederationTokenRequest.setPolicy(policy.toJson());

GetFederationTokenResult federationTokenResult =
                     stsClient.getFederationToken(getFederationTokenRequest);
Credentials sessionCredentials = federationTokenResult.getCredentials();

// Package the session credentials as a BasicSessionCredentials object
// for an S3 client object to use.
BasicSessionCredentials basicSessionCredentials = new BasicSessionCredentials(

      sessionCredentials.getAccessKeyId(),
      sessionCredentials.getSecretAccessKey(),
      sessionCredentials.getSessionToken());

// The following will be part of your less trusted code. You provide temporary
```

```
 security
// credentials so it can send authenticated requests to Amazon S3.
// Create an Amazon S3 client by passing in the basicSessionCredentials object.
AmazonS3Client s3 = new AmazonS3Client(basicSessionCredentials);

// Test. For example, send list object keys in a bucket.
ObjectListing objects = s3.listObjects(bucketName);
```

ポリシーに条件を設定するには、Condition オブジェクトを作成し、ポリシーに関連付けます。次のコード例は、指定された IP アドレスの範囲のユーザーにオブジェクトのリスト作成を許可する条件を示しています。

```
Policy policy = new Policy();

// Allow only a specified IP range.
Condition condition = new StringCondition(StringCondition.StringComparison
Type.StringLike,
    ConditionFactory.SOURCE_IP_CONDITION_KEY , "192.168.143.*");

policy.withStatements(new Statement(Effect.Allow)
    .withActions(S3Actions.ListObjects)
    .withConditions(condition)
    .withResources(new Resource("arn:aws:s3:::"+ bucketName)));

getFederationTokenRequest.setPolicy(policy.toJson());
```

Example

次の Java コード例は、指定されたバケット内のキーをリストしています。このコード例では、まずフェデレーションユーザー (User1) 用に 2 時間のセッションの一時的なセキュリティ認証情報を取得し、それを使用して認証リクエストを Amazon S3 に送信します。

他のユーザー用に一時的な認証情報をリクエストする場合、より高度なセキュリティのために、一時的なセキュリティ認証情報をリクエストする権限を持つ IAM ユーザーのセキュリティ認証情報を使用します。また、この IAM ユーザーが一時的なセキュリティ認証情報をリクエストするときに、最小限のアプリケーション固有の権限しか付与できないように、IAM ユーザーのアクセス許可を制限することができます。この例では、特定のバケットに含まれるオブジェクトのみリストされています。そのため、まず以下のポリシーをアタッチして IAM ユーザーを作成します。

```
{
  "Statement":[{
      "Action":["s3:ListBucket",
        "sts:GetFederationToken*"
      ],
      "Effect":"Allow",
      "Resource":"*"
    }
  ]
}
```

このポリシーでは、IAM ユーザーが一時的なセキュリティ認証情報と、AWS リソースをリストできるだけのアクセス許可をリクエストすることを許可しています。IAM ユーザーを作成する方法については、『*IAM 入門ガイド*』の「Set Up a Group, Grant Permissions, and Add Users」を参照してください。

これで、IAM ユーザーのセキュリティ認証情報を使用して、以下の例をテストできるようになりました。この例では、一時的なセキュリティ認証情報を使用して、認証リクエストを Amazon S3 に送信します。この例では、フェデレーションユーザー (User1) 用に、アクセスを特定のバケット (YourBucketName) 内のオブジェクトの列挙に制限する一時的なセキュリティ認証情報をリクエストするときに、以下のポリシーを指定します。ポリシーを更新し、独自の既存するバケット名を指定する必要があります。

```
{
  "Statement":[
    {
      "Sid":"1",
      "Action":["s3:ListBucket"],
      "Effect":"Allow",
      "Resource":"arn:aws:s3:::YourBucketName"
    }
  ]
}
```

以下の例を編集して、前述のフェデレーションユーザーアクセスポリシーで指定したバケット名を指定します。

```
import java.io.IOException;
import com.amazonaws.auth.BasicSessionCredentials;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.auth.policy.Policy;
import com.amazonaws.auth.policy.Resource;
import com.amazonaws.auth.policy.Statement;
```

```
import com.amazonaws.auth.policy.Statement.Effect;
import com.amazonaws.auth.policy.actions.S3Actions;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.securitytoken.AWSSecurityTokenServiceClient;
import com.amazonaws.services.securitytoken.model.Credentials;
import com.amazonaws.services.securitytoken.model.GetFederationTokenRequest;
import com.amazonaws.services.securitytoken.model.GetFederationTokenResult;
import com.amazonaws.services.s3.model.ObjectListing;

public class S3Sample {
 private static String bucketName = "*** Specify bucket name ***";
    public static void main(String[] args) throws IOException {
        PropertiesCredentials credentials = new PropertiesCredentials(
              S3Sample.class.getResourceAsStream("AwsCredentials.properties"));


        AWSSecurityTokenServiceClient stsClient =
                        new AWSSecurityTokenServiceClient(credentials);

        GetFederationTokenRequest getFederationTokenRequest =
                                            new GetFederationTokenRequest();
        getFederationTokenRequest.setDurationSeconds(7200);
        getFederationTokenRequest.setName("User1");

        // Define the policy and add to the request.
        Policy policy = new Policy();
        policy.withStatements(new Statement(Effect.Allow)
            .withActions(S3Actions.ListObjects)
            .withResources(new Resource("arn:aws:s3:::ExampleBucket")));

        getFederationTokenRequest.setPolicy(policy.toJson());

        // Get the temporary security credentials.
        GetFederationTokenResult federationTokenResult =
                     stsClient.getFederationToken(getFederationTokenRequest);

       Credentials sessionCredentials = federationTokenResult.getCredentials();


        // Package the session credentials as a BasicSessionCredentials
        // object for an S3 client object to use.
        BasicSessionCredentials basicSessionCredentials =
              new BasicSessionCredentials(sessionCredentials.getAccessKeyId(),

                                    sessionCredentials.getSecretAccessKey(),
                                    sessionCredentials.getSessionToken());
        AmazonS3Client s3 = new AmazonS3Client(basicSessionCredentials);


       // Test. For example, send ListBucket request using the temporary secur
ity credentials.
        ObjectListing objects = s3.listObjects(bucketName);
        System.out.println("No. of Objects = " + objects.getObjectSummar
ies().size());
    }
}
```

関連リソース

- AWS dynamo と Explorer の使用 (p. 475)

フェデレーションユーザーの一時的な認証情報を使用したリクエストの実行 – AWS SDK for .NET

フェデレーションユーザーおよびアプリケーションから AWS リソースへのアクセスのリクエストが認証されて送信されるように、フェデレーションユーザーおよびアプリケーションに一時的なセキュリティ認証情報を提供することができます (「リクエストの実行 (p. 11)」を参照) 。このような一時的な認証情報をリクエストする場合、ユーザー名と、付与するリソースアクセス許可を示す IAM ポリシーを指定する必要があります。デフォルトでは、セッションの有効期間は 1 時間です。フェデレーションユーザーおよびアプリケーション用に一時的なセキュリティ認証情報をリクエストするときに、明示的に別の有効期間値を設定できます。

Note

フェデレーションユーザーおよびアプリケーション用の一時的なセキュリティ認証情報をリクエストするには、より高度なセキュリティのために、必要なアクセス許可のみを持つ専用の IAM ユーザーを使用することもできます。作成した一時ユーザーに、一時的なセキュリティ認証情報をリクエストした IAM ユーザーより多くの権限が付与されることはありません。詳細については、「AWS Identity and Access Management FAQs」を参照してください。

フェデレーションユーザーの一時的な認証情報を使用したリクエストの実行

1	ご自分の認証情報を指定して、AWS Security Token Service クライアントのインスタンスである `AmazonSecurityTokenServiceClient` クラスを作成します。
2	STS クライアントの `GetFederationToken` メソッドを呼び出してセッションを開始します。 一時的な認証情報にアタッチするユーザー名と IAM ポリシーなどのセッション情報を指定する必要があります。必要に応じて、セッションの有効期間を指定できます。 このメソッドは一時的なセキュリティ認証情報を返します。
3	`SessionAWSCredentials` オブジェクトのインスタンスに一時的なセキュリティ認証情報をパッケージ化します。このオブジェクトを使用して、 Amazon S3 クライアントに対して一時的なセキュリティ認証情報を提供します。
4	一時的なセキュリティ認証情報を渡して、`AmazonS3Client` クラスのインスタンスを作成します。 このクライアントを使用してリクエストを Amazon S3 に送信します。失効した認証情報を使用してリクエストを送信すると、Amazon S3 はエラーを返します。

以下の C# コード例は、前述のタスクの例です。

```
// In real applications, the following code is part of your trusted code. It
has
// your security credentials you use to obtain temporary security credentials.
AmazonSecurityTokenServiceConfig config = new AmazonSecurityTokenServiceConfig();
AmazonSecurityTokenServiceClient stsClient =
       new AmazonSecurityTokenServiceClient(
                        accessKeyId, secretAccessKeyId, config);
```

```
GetFederationTokenRequest federationTokenRequest =
                                  new GetFederationTokenRequest();
federationTokenRequest.Name            = "User1";
federationTokenRequest.Policy          =  "*** Specify policy ***";
federationTokenRequest.DurationSeconds = 7200;

GetFederationTokenResponse federationTokenResponse =
                   stsClient.GetFederationToken(federationTokenRequest);
GetFederationTokenResult federationTokenResult =
                   federationTokenResponse.GetFederationTokenResult;
Credentials credentials = federationTokenResult.Credentials;


SessionAWSCredentials sessionCredentials =
                  new SessionAWSCredentials(credentials.AccessKeyId,
                                            credentials.SecretAccessKey,
                                            credentials.SessionToken);

// The following will be part of your less trusted code. You provide temporary
 security
// credentials so it can send authenticated requests to Amazon S3.
// Create Amazon S3 client by passing in the basicSessionCredentials object.
AmazonS3Client s3Client = new AmazonS3Client(sessionCredentials);
// Test. For example, send list object keys in a bucket.
ListObjectsRequest listObjectRequest =  new ListObjectsRequest();
listObjectRequest.BucketName = bucketName;
ListObjectsResponse response = s3Client.ListObjects(listObjectRequest);
```

Example

次の C# コード例では、指定したバケットのキーをリストします。このコード例では、まずフェデレーションユーザー (User1) 用に 2 時間のセッションの一時的なセキュリティ認証情報を取得し、それを使用して認証リクエストを Amazon S3 に送信します。

他のユーザー用に一時的な認証情報をリクエストする場合、より高度なセキュリティのために、一時的なセキュリティ認証情報をリクエストする権限を持つ IAM ユーザーのセキュリティ認証情報を使用します。また、IAM ユーザーがフェデレーションユーザーに最小限のアプリケーション固有のアクセス許可しか付与できないように、この IAM ユーザーのアクセス許可を制限することができます。この例では、特定のバケットに含まれるオブジェクトのみリストされています。そのため、まず以下のポリシーをアタッチして IAM ユーザーを作成します。

```
{
  "Statement":[{
      "Action":["s3:ListBucket",
        "sts:GetFederationToken*"
      ],
      "Effect":"Allow",
      "Resource":"*"
    }
  ]
}
```

このポリシーでは、IAM ユーザーが一時的なセキュリティ認証情報と、AWS リソースをリストできるだけのアクセス許可をリクエストすることを許可しています。IAM ユーザーを作成する方法については、『IAM 入門ガイド』の「*Set Up a Group, Grant Permissions, and Add Users*」を参照してください。

これで、IAM ユーザーのセキュリティ認証情報を使用して、以下の例をテストできるようになりました。この例では、一時的なセキュリティ認証情報を使用して、認証リクエストを Amazon S3 に送信します。この例では、フェデレーションユーザー (User1) 用に、アクセスを特定のバケット (YourBucketName) 内のオブジェクトの列挙に制限する一時的なセキュリティ認証情報をリクエストするときに、以下のポリシーを指定します。ポリシーを更新し、独自の既存するバケット名を指定する必要があります。

```
{
  "Statement":[
    {
      "Sid":"1",
      "Action":["s3:ListBucket"],
      "Effect":"Allow",
      "Resource":"arn:aws:s3:::YourBucketName"
    }
  ]
}
```

以下の例を編集して、前述のフェデレーションユーザーアクセスポリシーで指定したバケット名を指定します。

```
using System;
using System.Configuration;
using System.Collections.Specialized;
using Amazon.S3;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;
```

```
using Amazon.Runtime;
using Amazon.S3.Model;
using System.Collections.Generic;

namespace s3.amazon.com.docsamples.listingkeys
{
    class TempCred_FederatedCredentials
    {
        static string bucketName = "*** Provide bucket name ***";

        public static void Main(string[] args)
        {
            NameValueCollection appConfig = ConfigurationManager.AppSettings;
            string accessKeyID = appConfig["AWSAccessKey"];
            string secretAccessKeyID = appConfig["AWSSecretKey"];
            try
            {
                Console.WriteLine("Listing objects stored in a bucket");
                SessionAWSCredentials tempCredentials =
                     GetTemporaryFederatedCredentials(accessKeyID,
                                                      secretAccessKeyID);

                AmazonS3Client s3Client = new AmazonS3Client(tempCredentials);


                ListObjectsRequest listObjectRequest =
                                  new ListObjectsRequest();
                listObjectRequest.BucketName = bucketName;

             ListObjectsResponse response = s3Client.ListObjects(listObjectRe
quest);
                List<S3Object> objects = response.S3Objects;
                Console.WriteLine("Object count = {0}", objects.Count);

                Console.WriteLine("Press any key to continue...");
                Console.ReadKey();
            }
            catch (AmazonS3Exception s3Exception)
            {
                Console.WriteLine(s3Exception.Message,
                                  s3Exception.InnerException);
            }
            catch (AmazonSecurityTokenServiceException stsException)
            {
                Console.WriteLine(stsException.Message,
                                 stsException.InnerException);
            }
        }

     private static SessionAWSCredentials GetTemporaryFederatedCredentials(
                               string accessKeyId, string secretAccessKeyId)
        {
           AmazonSecurityTokenServiceConfig config = new AmazonSecurityTokenSer
viceConfig();
            AmazonSecurityTokenServiceClient stsClient =
                new AmazonSecurityTokenServiceClient(
                                             accessKeyId, secretAccessKeyId,
config);
```

```
            GetFederationTokenRequest federationTokenRequest =
                                    new GetFederationTokenRequest();
            federationTokenRequest.DurationSeconds = 7200;
            federationTokenRequest.Name   = "User1";
            federationTokenRequest.Policy =   @"{
  ""Statement"":
  [
    {
      ""Sid"":""Stmt1311212314284"",
      ""Action"":[""s3:ListBucket""],
      ""Effect"":""Allow"",
      ""Resource"":""arn:aws:s3:::ExampleBucket""
    }
  ]
}

";


            GetFederationTokenResponse federationTokenResponse =
                        stsClient.GetFederationToken(federationTokenRequest);

            GetFederationTokenResult federationTokenResult =
                           federationTokenResponse.GetFederationTokenResult;

            Credentials credentials = federationTokenResult.Credentials;


            SessionAWSCredentials sessionCredentials =
                new SessionAWSCredentials(credentials.AccessKeyId,
                                          credentials.SecretAccessKey,
                                          credentials.SessionToken);
            return sessionCredentials;
        }
    }
}
```

関連リソース

- AWS dynamo と Explorer の使用 (p. 475)

フェデレーションユーザーの一時的な認証情報を使用したリクエストの実行 – AWS SDK for PHP

このトピックでは、AWS SDK for PHP のクラスを使用して、フェデレーションのユーザーおよびアプリケーションの一時的セキュリティ認証情報をリクエストし、その認証情報を使用して Amazon S3 にアクセスする手順を示します。

Note

このトピックでは、既に AWS SDK for PHP の使用と PHP サンプルの実行 (p. 479) の説明が実行されていて、AWS SDK for PHP が正しくインストールされていることを前提としています。

フェデレーションユーザーおよびアプリケーションから AWS リソースへのアクセスのリクエストが認証されて送信されるように、フェデレーションユーザーおよびアプリケーションに一時的なセキュリティ認証情報を提供することができます（「リクエストの実行 (p. 11)」を参照）。このような一時的な認証情報をリクエストする場合、ユーザー名と、付与するリソースアクセス許可を示す IAM ポリシーを指定する必要があります。これらの認証情報は、セッションの有効期間が失効すると同時に失効します。デフォルトでは、セッションの有効期間は 1 時間です。フェデレーションユーザーおよびアプリケーション用に一時的なセキュリティ認証情報をリクエストするときに、明示的に別の有効期間値を設定できます。

フェデレーションユーザーおよびアプリケーション用の一時的なセキュリティ認証情報をリクエストするには、より高度なセキュリティのために、必要なアクセス許可のみを持つ専用の IAM ユーザーを使用することもできます。作成した一時ユーザーに、一時的なセキュリティ認証情報をリクエストした IAM ユーザーより多くの権限が付与されることはありません。詳細については、「AWS Identity and Access Management FAQs」を参照してください。

フェデレーションユーザーの一時的な認証情報を使用したリクエストの実行

1	Aws\Sts\StsClient クラスの factory() メソッドを使用して、AWS Security Token Service（AWS STS）クライアントのインスタンスを作成します。
2	`array` パラメータの必須キー、`Name` にフェデレーションユーザーの名前を指定して、Aws\Sts\StsClient::getFederationToken() メソッドを実行します。オプションの `array` パラメータの `Policy` および `DurationSeconds` キーを追加することもできます。 このメソッドから返される一時的なセキュリティ認証情報をフェデレーションユーザーに渡すことができます。
3	一時的なセキュリティ認証情報を渡されたフェデレーションユーザーは、その一時的なセキュリティ認証情報と共に Aws\S3\S3Client クラスの factory メソッドを使用し、Amazon S3 クライアントのインスタンスを作成することで、Amazon S3 にリクエストを送信することができます。 呼び出す `S3Client` クラスのメソッドはいずれも、一時的なセキュリティ認証情報を使用して、認証リクエストを Amazon S3 に送信します。

以下の PHP コード例では、フェデレーションユーザーの一時的なセキュリティ認証情報を取得し、その認証情報を使用して Amazon S3 にアクセスしています。

```
use Aws\Sts\StsClient;
use Aws\S3\S3Client;

// In real applications, the following code is part of your trusted code. It
has
// your security credentials that you use to obtain temporary security creden
tials.
$sts = StsClient::factory(array(
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));

// Fetch the federated credentials.
$result = $sts->getFederationToken(array(
    'Name'            => 'User1',
    'DurationSeconds' => 3600,
    'Policy'          => json_encode(array(
        'Statement' => array(
            array(
```

```
                'Sid'      => 'randomstatementid' . time(),
                'Action'   => array('s3:ListBucket'),
                'Effect'   => 'Allow',
                'Resource' => 'arn:aws:s3:::YourBucketName'
            )
        )
    ))
));

// The following will be part of your less trusted code. You provide temporary
// security credentials so it can send authenticated requests to Amazon S3.
$credentials = $result->get('Credentials');
$s3 = new S3Client::factory(array(
    'key'    => $credentials['AccessKeyId'],
    'secret' => $credentials['SecretAccessKey'],
    'token'  => $credentials['SessionToken']
));

$result = $s3->listObjects();
```

Example （一時的なセキュリティ認証情報を使用して Amazon S3 リクエストを実行するフェデレーションユーザー）

次の PHP コード例では、指定したバケットのキーをリストします。このコード例では、まずフェデレーションユーザー（User1）用に 1 時間のセッションの一時的なセキュリティ認証情報を取得し、それを使用して、認証リクエストを Amazon S3 に送信します。PHP 例の実行については、このガイド内の「PHP サンプルの実行 (p. 480)」を参照してください。

他のユーザー用に一時的な認証情報をリクエストする場合、より高度なセキュリティのために、一時的なセキュリティ認証情報をリクエストする権限を持つ IAM ユーザーのセキュリティ認証情報を使用します。また、IAM ユーザーがフェデレーションユーザーに最小限のアプリケーション固有のアクセス許可しか付与できないように、この IAM ユーザーのアクセス許可を制限することができます。この例では、特定のバケットに含まれるオブジェクトのみ表示されています。そのため、まず以下のポリシーをアタッチして IAM ユーザーを作成します。

```
{
  "Statement":[{
      "Action":["s3:ListBucket",
        "sts:GetFederationToken*"
      ],
      "Effect":"Allow",
      "Resource":"*"
    }
  ]
}
```

このポリシーでは、IAM ユーザーが一時的なセキュリティ認証情報と、AWS リソースをリストできるだけのアクセス許可をリクエストすることを許可しています。IAM ユーザーを作成する方法については、『IAM 入門ガイド』の「*Set Up a Group, Grant Permissions, and Add Users*」を参照してください。

これで、IAM ユーザーのセキュリティ認証情報を使用して、以下の例をテストできるようになりました。この例では、一時的なセキュリティ認証情報を使用して、認証リクエストを Amazon S3 に送信します。この例では、フェデレーションユーザー（User1）用に、アクセスを特定のバケット内のオブジェクトの列挙に制限する一時的なセキュリティ認証情報をリクエストするときに、以下のポリシーを指定します。ポリシーを更新し、独自の既存するバケット名を指定する必要があります。

```
{
  "Statement":[
    {
      "Sid":"1",
      "Action":["s3:ListBucket"],
      "Effect":"Allow",
      "Resource":"arn:aws:s3:::YourBucketName"
    }
  ]
}
```

以下の例を変更して、前述のフェデレーションユーザーアクセスポリシーで指定したバケット名を使用するようにします。

```
<?php

// Include the AWS SDK using the Composer autoloader.
require 'vendor/autoload.php';
```

```
$bucket = '*** Your Bucket Name ***';

use Aws\Sts\StsClient;
use Aws\S3\S3Client;
use Aws\S3\Exception\S3Exception;

// Instantiate the client.
$sts = StsClient::factory(array(
    'key'    => '*** Your IAM User Access Key ID ***',
    'secret' => '*** Your IAM User Secret Key ***'
));

$result = $sts->getFederationToken(array(
    'Name'            => 'User1',
    'DurationSeconds' => 3600,
    'Policy'          => json_encode(array(
        'Statement' => array(
            array(
                'Sid'      => 'randomstatementid' . time(),
                'Action'   => array('s3:ListBucket'),
                'Effect'   => 'Allow',
                'Resource' => 'arn:aws:s3:::YourBucketName'
            )
        )
    ))
));

$credentials = $result->get('Credentials');
$s3 = S3Client::factory(array(
    'key'    => $credentials['AccessKeyId'],
    'secret' => $credentials['SecretAccessKey'],
    'token'  => $credentials['SessionToken']
));

try {
    $objects = $s3->getIterator('ListObjects', array(
        'Bucket' => $bucket
    ));

    echo "Keys retrieved!\n";
    foreach ($objects as $object) {
        echo $object['Key'] . "\n";
    }
} catch (S3Exception $e) {
    echo $e->getMessage() . "\n";
}
```

関連リソース

- 「AWS SDK for PHP for Amazon S3 の Aws\Sts\StsClient クラス」
- 「AWS SDK for PHP for Amazon S3 の Aws\Sts\StsClient::factory() メソッド」
- 「AWS SDK for PHP for Amazon S3 の Aws\Sts\StsClient::getSessionToken メソッド」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client クラス」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::factory() メソッド」
- 「AWS SDK for PHP – Amazon S3」
- 「AWS SDK for PHP」のドキュメント

フェデレーションユーザーの一時的な認証情報を使用したリクエストの実行 – AWS SDK for Ruby

フェデレーションユーザーおよびアプリケーションから AWS リソースへのアクセスのリクエストが認証されて送信されるように、フェデレーションユーザーおよびアプリケーションの一時的なセキュリティ認証情報を提供することができます (「リクエストの実行 (p. 11)」を参照)。IAM サービスからこのような一時的な認証情報をリクエストする場合、ユーザー名と、付与するリソースアクセス許可を示す IAM ポリシーを指定する必要があります。デフォルトでは、セッションの有効期間は 1 時間です。ただし、IAM ユーザー認証情報を使用して一時的な認証情報をリクエストする場合、フェデレーションユーザーおよびアプリケーション用の一時的なセキュリティ認証情報をリクエストするときに、別の有効期間値を明示的に設定できます。

Note

フェデレーションユーザーおよびアプリケーション用の一時的なセキュリティ認証情報をリクエストするには、より高度なセキュリティのために、必要なアクセス許可のみを持つ専用の IAM ユーザーを使用することもできます。作成した一時ユーザーに、一時的なセキュリティ認証情報をリクエストした IAM ユーザーより多くの権限が付与されることはありません。詳細については、「AWS Identity and Access Management FAQs」を参照してください。

フェデレーションユーザーの一時的なセキュリティ認証情報を使用したリクエストの実行

1	自分のセキュリティ認証情報を指定して、AWS Security Token Service クライアントのインスタンスである `AWS::STS::Session` を作成します。
2	前の手順で作成した STS クライアントの `new_federated_session` メソッドを呼び出して、セッションを開始します。 一時的な認証情報にアタッチするユーザー名と IAM ポリシーなどのセッション情報を指定する必要があります。 このメソッドは一時的なセキュリティ認証情報を返します。
3	一時的なセキュリティ認証情報を渡して、`AWS::S3` クラスのインスタンスを作成します。 このクライアントを使用してリクエストを Amazon S3 に送信します。失効した認証情報を使用してリクエストを送信すると、Amazon S3 はエラーを返します。

以下の Ruby コード例は、前述のタスクの例です。

```
# Start a session with restricted permissions.
sts = AWS::STS.new()
policy = AWS::STS::Policy.new
policy.allow(
  :actions => ["s3:ListBucket"],
  :resources => "arn:aws:s3:::#{bucket_name}")

session = sts.new_federated_session(
  'User1',
  :policy => policy,
  :duration => 2*60*60)

puts "Policy: #{policy.to_json}"

# Get an instance of the S3 interface using the session credentials.
```

```
s3 = AWS::S3.new(session.credentials)

# Get a list of all object keys in a bucket.
bucket = s3.buckets[bucket_name].objects.collect(&:key)
```

Example

次の Ruby コード例では、指定したバケットのキーをリストします。このコード例では、まずフェデレーションユーザー (User1) 用に 2 時間のセッションの一時的なセキュリティ認証情報を取得し、それを使用して認証リクエストを Amazon S3 に送信します。

他のユーザー用に一時的な認証情報をリクエストする場合、より高度なセキュリティのために、一時的なセキュリティ認証情報をリクエストする権限を持つ IAM ユーザーのセキュリティ認証情報を使用します。また、この IAM ユーザーが一時的なセキュリティ認証情報をリクエストするときに、最小限のアプリケーション固有の権限しか付与できないように、IAM ユーザーのアクセス許可を制限することができます。この例では、特定のバケットに含まれるオブジェクトのみリストされています。そのため、まず以下のポリシーをアタッチして IAM ユーザーを作成します。

```
{
  "Statement":[{
      "Action":["s3:ListBucket",
        "sts:GetFederationToken*"
      ],
      "Effect":"Allow",
      "Resource":"*"
    }
  ]
}
```

このポリシーでは、IAM ユーザーが一時的なセキュリティ認証情報と、AWS リソースをリストできるだけのアクセス許可をリクエストすることを許可しています。IAM ユーザーを作成する方法については、『IAM 入門ガイド』の「*Set Up a Group, Grant Permissions, and Add Users*」を参照してください。

これで、IAM ユーザーのセキュリティ認証情報を使用して、以下の例をテストできるようになりました。次の例では、一時的なセキュリティ認証情報を使用して、認証リクエストを Amazon S3 に送信します。この例では、フェデレーションユーザー (User1) 用に一時的なセキュリティ認証情報をリクエストするときに、アクセスを特定のバケット (YourBucketName) 内のオブジェクトのリスト作成に制限する以下のようなポリシーを指定します。この例を自分のコード内で使用するには、ポリシーを更新し、自分のバケット名を指定してください。

```
{
  "Statement":[
    {
      "Sid":"1",
      "Action":["s3:ListBucket"],
      "Effect":"Allow",
      "Resource":"arn:aws:s3:::YourBucketName"
    }
  ]
}
```

この例を自分のコード内で使用するには、アクセスキー ID とシークレットキーのほか、前述のフェデレーションユーザーアクセスポリシーで指定したバケット名を指定します。

```
require 'rubygems'
require 'aws-sdk'

# In real applications, the following code is part of your trusted code. It has

# your security credentials that you use to obtain temporary security creden
```

```
tials.

my_access_key_id = '***Provide Access Key ID***'
my_secret_key = '***Provide Secret Key***'
bucket_name = '*** Provide bucket name ***'

AWS.config({
  :access_key_id => my_access_key_id,
  :secret_access_key => my_secret_key
})

# Start a session with restricted permissions.
sts = AWS::STS.new()
policy = AWS::STS::Policy.new
policy.allow(
  :actions => ["s3:ListBucket"],
  :resources => "arn:aws:s3:::#{bucket_name}")

session = sts.new_federated_session(
  'User1',
  :policy => policy,
  :duration => 2*60*60)

puts "Policy: #{policy.to_json}"

# Get an instance of the S3 interface using the session credentials.
s3 = AWS::S3.new(session.credentials)

# Get a list of all object keys in a bucket.
bucket = s3.buckets[bucket_name].objects.collect(&:key)
puts "No. of Objects = #{bucket.count.to_s}"
puts bucket
```

REST API を使用したリクエストの実行

Topics

このセクションでは、Amazon S3 REST API に固有の情報について説明します。このガイドの例では、バケットにアクセスするときに、パス形式ではなく、新しい仮想ホスト形式の方法を使用します。詳細については、「Amazon S3 バケットの使用 (p. 87)」を参照してください

mybucket バケットから puppy.jpg ファイルを削除する仮想ホスト形式のリクエストの例を次に示します。

```
DELETE /puppy.jpg HTTP/1.1
User-Agent: dotnet
Host: mybucket.s3.amazonaws.com
```

```
Date: Tue, 15 Jan 2008 21:20:27 +0000
x-amz-date: Tue, 15 Jan 2008 21:20:27 +0000
Authorization: AWS AKIAIOSFODNN7EXAMPLE:k3nL7gH3+PadhTEVn5EXAMPLE
```

同じリクエストのパス形式バージョンの例を次に示します。

```
DELETE /mybucket/puppy.jpg HTTP/1.1
User-Agent: dotnet
Host: s3.amazonaws.com
Date: Tue, 15 Jan 2008 21:20:27 +0000
x-amz-date: Tue, 15 Jan 2008 21:20:27 +0000
Authorization: AWS AKIAIOSFODNN7EXAMPLE:k3nL7gH3+PadhTEVn5EXAMPLE
```

Amazon S3 supports virtual-hosted-style and path-style access in all Regions. The path-style syntax, however, requires that you use the region-specific endpoint when attempting to access a bucket. For example, if you have a bucket called `mybucket` that resides in the EU, you want to use path-style syntax, and the object is named `puppy.jpg`, the correct URI is `http://s3-eu-west-1.amazonaws.com/mybucket/puppy.jpg`. You will receive a "PermanentRedirect" error, an HTTP response code 301, and a message indicating what the correct URI is for your resource if you try to access a non US Standard bucket with path-style syntax using:

- `http://s3.amazonaws.com`
- A different Region endpoint than where the bucket resides, for example, `http://s3-eu-west-1.amazonaws.com` and the bucket was created with the location constraint of Northern-California

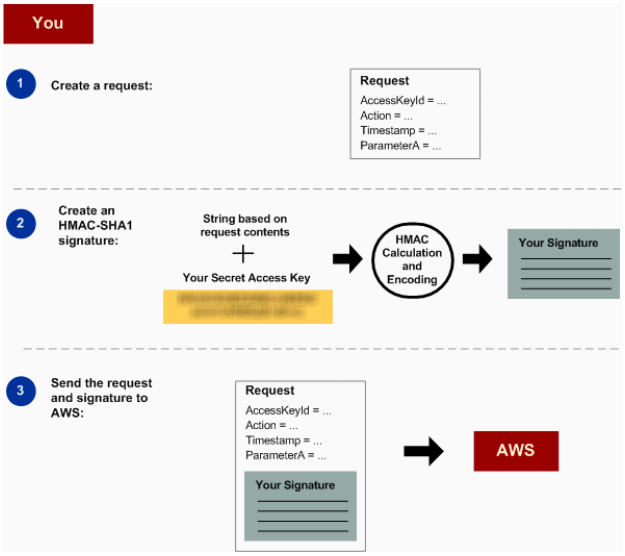
REST API を使用したリクエストの認証

REST を使用して Amazon S3 にアクセスする場合は、リクエストが認証されるように、リクエストに以下の項目を指定する必要があります。

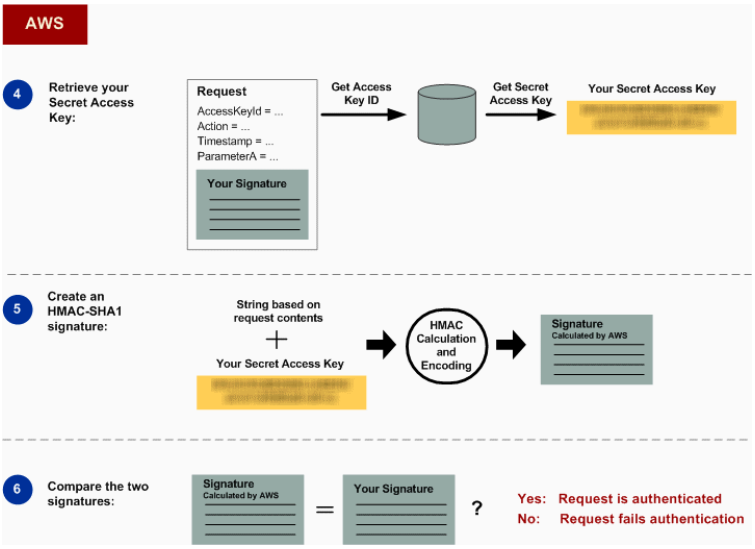
リクエストの要素

- AWS アクセスキー ID – 各リクエストには、リクエストの送信に使用する認証情報のアクセスキー ID を含める必要があります。
- 署名 – 各リクエストには、有効なリクエスト署名を含める必要があります。含まれない場合、リクエストは拒否されます。
  リクエスト署名は、シークレットアクセスキーを使用して計算されます。シークレットアクセスキーは、自分と AWS のみが把握する共有の秘密です。
- タイムスタンプ – 各リクエストには、リクエストを作成した日時を含める必要があります。日時は UTC 時間の文字列として表されます。
- 日付 – 各リクエストには、リクエストのタイムスタンプを含める必要があります。
  使用する API アクションによっては、タイムスタンプを追加するかわりに、リクエストの有効期限を示す日時を指定できます。特定のアクションに必要な項目を確認するには、そのアクションの認証のトピックを参照してください。

Amazon S3 へのリクエストを認証するための一般的なステップは以下のとおりです。この説明では、必要なセキュリティ認証情報、アクセスキー ID、およびシークレットアクセスキーをお客様が持っていることを前提としています。



1	AWS にリクエストします。
2	シークレットアクセスキーを使用して署名を計算します。
3	Amazon S3 にリクエストを送信します。リクエストにアクセスキー ID と署名を含めます。次の 3 つのステップは Amazon S3 が実行します。



4	Amazon S3 がアクセスキー ID を使用して、ユーザーのシークレットアクセスキーを調べます。
5	ユーザーがリクエストで送信した署名を生成するのに使用したのと同じアルゴリズムを使用して、Amazon S3 がリクエストデータとシークレットアクセスキーから署名を生成します。
6	Amazon S3 が生成した署名がユーザーがリクエストに含めたものと一致した場合、リクエストは正規のものと認識されます。もし署名が一致しなかった場合、リクエストの処理は拒否され、Amazon S3 はエラーレスポンスを返します。

詳細な認証情報

REST 認証の詳細については、「Signing and Authenticating REST Requests (p. 52)」を参照してください。

Signing and Authenticating REST Requests

Topics

認証とは、お客様がご本人であることをシステムに証明するプロセスをいいます。アイデンティティは、Amazon S3 のアクセスコントロールの判断を行う際に重要です。リクエストの許可または拒否は、

リクエスタのアイデンティティに部分的に基づいています。例えば、バケットを作成する権利は、登録開発者用に予約されています。また、バケットにオブジェクトを作成する権利は、対象のバケット所有者用に (デフォルトで) 予約されています。開発者としてこれらの権限を実行するリクエストを行うので、そのリクエストを認証することで、自身のアイデンティティをシステムに対して証明する必要があります。このセクションでは、その方法を説明します。

Note

このセクションの内容は HTTP POST には適用されません。詳細については、「POST を使用したブラウザベースのアップロード (p. 70)」を参照してください。

Amazon S3 REST API では、キーが設定された HMAC (ハッシュメッセージ認証コード) に基づいたカスタム HTTP スキームが認証に使用されます。リクエストを認証するには、まず、選択されているリクエストの要素を連結し、文字列を作成します。次に、AWS シークレットアクセスキーを使用して、その文字列の HMAC を計算します。正式な呼び名ではありませんが、このプロセスのことを「リクエストへの署名」と呼び、HMAC アルゴリズムの出力を署名と呼びます。これは、このプロセスが実際の署名のセキュリティプロパティを真似ているからです。最後に、このセクションで説明する構文を使用して、この署名をリクエストのパラメータとして追加します。

システムは、認証済みリクエストを受け取るとき、リクエスト送信者が所有している AWS シークレットアクセスキーを取得し、同様の方法でそのアクセスキーを使用して、受信したメッセージの署名を計算します。そして、計算した署名と、リクエスタから提示された署名を比較します。2 つの署名が一致したら、リクエスタは AWS シークレットアクセスキーにアクセスできると判断されるため、システムはそのキーの発行先となるプリンシパルの権限をサポートします。2 つの署名が一致しない場合、リクエストは中断し、エラーメッセージが返されます。

Example 認証された Amazon S3 REST リクエスト

```
GET /photos/puppy.jpg HTTP/1.1
Host: johnsmith.s3.amazonaws.com
Date: Mon, 26 Mar 2007 19:37:58 +0000

Authorization: AWS AKIAIOSFODNN7EXAMPLE:frJIUN8DYpKDtOLCwo//yllqDzg=
```

一時的なセキュリティ認証情報の使用

一時的なセキュリティ認証情報を使用してリクエストに署名する場合は (「リクエストの実行 (p. 11)」を参照)、`x-amz-security-token` ヘッダーを追加して、対応するセキュリティトークンをリクエストに含める必要があります。

AWS Security Token Service API を使用して一時的なセキュリティ認証情報を取得すると、そのレスポンスには、一時的なセキュリティ認証情報とセッショントークンが含まれます。リクエストを Amazon S3 に送信するときに、`x-amz-security-token` ヘッダーでセッショントークン値を指定します。IAM が提供する AWS Security Token Service API については、『*AWS Security Token Service API リファレンス Guide*』の「Action」を参照してください。

認証ヘッダー

Amazon S3 REST API は、標準の HTTP `Authorization` ヘッダーを使用して、認証情報を渡します。(標準ヘッダーの名前とは異なり、ヘッダーに含まれるのは承認ではなく認証情報です。) Amazon S3 認証スキームにおける認証ヘッダーの形式は次のとおりです。

```
Authorization: AWS AWSAccessKeyId:Signature
```

開発者には、登録時に AWS アクセスキー ID と AWS シークレットアクセスキーが発行されます。リクエストを認証するために、*AWSAccessKeyId* 要素は、署名の計算に使用されたアクセスキー ID を識別するほか、リクエストを行っている開発者も間接的に識別します。

Signature 要素は、リクエストから選択した要素の RFC 2104HMAC-SHA1 です。したがって、Authorization ヘッダーの *Signature* 部分はリクエストによって異なります。システムによって計算されたリクエストの署名が、リクエストに含まれる *Signature* と一致する場合は、リクエスタが AWS シークレットアクセスキーを所有していることになります。その後、リクエストは、キーの発行対象者である開発者のアイデンティティと権限に従って処理されます。

以下は擬似文法で、*Authorization* リクエストヘッダーの構文例を示しています。（例中の \n は Unicode のコードポイント U+000A を意味しています。これは通常改行と呼ばれています）。

```
Authorization = "AWS" + " " + AWSAccessKeyId + ":" + Signature;

Signature = Base64( HMAC-SHA1( YourSecretAccessKeyID, UTF-8-Encoding-Of(
StringToSign ) ) );

StringToSign = HTTP-Verb + "\n" +
 Content-MD5 + "\n" +
 Content-Type + "\n" +
 Date + "\n" +
 CanonicalizedAmzHeaders +
 CanonicalizedResource;

CanonicalizedResource = [ "/" + Bucket ] +
 <HTTP-Request-URI, from the protocol name up to the query string> +
 [ subresource, if present. For example "?acl", "?location", "?logging", or
"?torrent"];

CanonicalizedAmzHeaders = <described below>
```

HMAC-SHA1 は、RFC 2104 – Keyed-Hashing for Message Authentication で定義されているアルゴリズムです。このアルゴリズムは、キーとメッセージの 2 つのバイト文字列を入力として取ります。Amazon S3 リクエスト認証については、AWS シークレットアクセスキー（*YourSecretAccessKeyID*）をキーとして、*StringToSign* の UTF-8 エンコーディングをメッセージとして使用します。また、HMAC-SHA1 の出力もバイト文字列で、これはダイジェストと呼ばれます。*Signature* リクエストパラメータは、このダイジェストの Base64 エンコードによって作成されます。

署名のためのリクエストの標準化

既に説明したように、システムは認証済みリクエストを受け取るときに、計算されたリクエスト署名と *StringToSign* のリクエストで指定された署名を比較します。この理由から、署名は、Amazon S3 と同じ方法で計算する必要があります。署名のためにリクエストを承認済み形式にするプロセスは *正規化* と言います。

CanonicalizedResource 要素の作成

CanonicalizedResource は、リクエストの対象となる Amazon S3 リソースを表します。REST リクエストのこのリソースは次のように作成します。

プロセスを起動

1	空の文字列（""）で開始します。

2	HTTP ホストヘッダー (仮想ホスト形式) を使用してバケットが指定されているリクエストについては、バケット名の前に "/" を付けます (例: 「/bucketname」) 。パス形式のリクエスト、およびバケットを処理しないリクエストの場合は、何も行いません。仮想ホスト形式のリクエストの詳細については、「バケットの仮想ホスティング (p. 63)」を参照してください。 仮想ホスト形式のリクエスト、「https://johnsmith.s3.amazonaws.com/photos/puppy.jpg」の場合、`CanonicalizedResource` は「/johnsmith」です。 パス形式のリクエスト、「https://s3.amazonaws.com/johnsmith/photos/puppy.jpg」の場合、`CanonicalizedResource` は「""」です。
3	デコードされていない HTTP リクエスト URI のパス部分を追加します (クエリ文字列まで、ただし、その文字列は含みません) 。 仮想ホスト形式のリクエスト、「https://johnsmith.s3.amazonaws.com/photos/puppy.jpg」の場合、`CanonicalizedResource` は「/johnsmith/photos/puppy.jpg」です。 パス形式のリクエスト、「https://s3.amazonaws.com/johnsmith/photos/puppy.jpg」の場合、`CanonicalizedResource` は「/johnsmith/photos/puppy.jpg」です。この時点で、`CanonicalizedResource` は、仮想ホスト形式とパス形式の両方のリクエストで同じです。 GET Service など、バケット宛でないリクエストの場合、「/」を追加します。
4	`?versioning`、`?location`、`?acl`、`?torrent`、`?lifecycle`、`?versionid` などのサブリソースを処理するリクエストの場合は、サブリソース、そのサブリソースの値 (存在する場合) 、および疑問符を追加します。複数のサブリソースは、名前のアルファベット順に並べ替えて「&」で区切る必要があります。例えば、?acl&versionId=*`value`* のようにする必要があります。 CanonicalizedResource 要素を作成するときに含める必要があるサブリソースは、acl、lifecycle、location、logging、notification、partNumber、policy、requestPayment、torrent、uploadId、uploads、versionId、versioning、versions、および website です。 レスポンスヘッダー値を上書きするクエリ文字列パラメータを指定するリクエストの場合は (GET Object を参照) 、クエリ文字列パラメータとそれらの値を追加します。これらのパラメータ値は署名時にエンコードする必要はありませんが、リクエストの実行時にはエンコードする必要があります。GET リクエストのクエリ文字列パラメータには、`response-content-type`、`response-content-language`、`response-expires`、`response-cache-control`、`response-content-disposition`、`response-content-encoding` があります。 複数のオブジェクトの Delete リクエストに対して CanonicalizedResource を作成する際には、`delete` クエリ文字列パラメータを含める必要があります。

HTTP リクエスト URI の CanonicalizedResource の要素は、URL エンコーディングメタ文字を含め、HTTP リクエストに表示されるとおりに署名する必要があります。

`CanonicalizedResource` は、HTTP リクエスト URI とは異なる場合があります。特に、リクエストが HTTP `Host` ヘッダーを使用してバケットを指定する場合、そのバケットは HTTP リクエスト URI に表示されませんが、*`CanonicalizedResource`* にもバケットが含まれます。クエリ文字列パラメータは、リクエスト URI に表示される可能性がありますが、*`CanonicalizedResource`* には含まれません。詳細については、「バケットの仮想ホスティング (p. 63)」を参照してください。

CanonicalizedAmzHeaders 要素の作成

`StringToSign` の CanonicalizedAmzHeaders 部分を作成するには、次の手順に従って、「x-amz-」で始まるすべての HTTP リクエストヘッダーを選択します (大文字と小文字は区別されません) 。

CanonicalizedAmzHeaders プロセス

1	各 HTTP ヘッダー名を小文字に変換します。例えば、「`X-Amz-Date`」は「`x-amz-date`」に変換します。
2	ヘッダーのコレクションを辞書と同じ順序でヘッダー名ごとに並べ替えます。

3	RFC 2616、セクション 4.2 の説明に従って、同じ名前のヘッダーフィールドを、1 つの「ヘッダー名:コンマ区切り値のリスト」のペアに結合します。その際、値と値の間の空白は削除されます。例えば、2 つのメタデータヘッダー「`x-amz-meta-username: fred`」および「`x-amz-meta-username: barney`」が 1 つのヘッダーに結合されると、「`x-amz-meta-username: fred,barney`」になります。
4	折りたたまれた空白 (改行マークを含む) を 1 つのスペースに置き換えて、(RFC 2616、セクション 4.2 で許可されているように) 複数行にわたる長いヘッダーを展開します。
5	ヘッダーのコロンの前後の空白を削除します。例えば、ヘッダー「`x-amz-meta-username: fred,barney`」は「`x-amz-meta-username:fred,barney`」になります。
6	最後に、改行文字 (`U+000A`) を、結果の一覧の各標準化ヘッダーに追加します。この一覧のすべてのヘッダーを 1 つの文字列に連結することで、CanonicalizedResource 要素を作成します。

位置および指定 HTTP ヘッダーの StringToSign 要素

`StringToSign` の最初のヘッダー要素 (Content-Type、Date、および Content-MD5) は位置を示します。*`StringToSign`* には、これらのヘッダーの名前は含まれません。含まれるのはリクエストの値のみです。一方、「`x-amz-`」要素には名前が付いています。ヘッダーの名前と値は両方とも *`StringToSign`* に含まれます。

`StringToSign` の定義で呼び出される位置ヘッダーがリクエストにない場合は (例えば、`Content-Type` または `Content-MD5` は、PUT リクエストのオプションであり、GET リクエストには無意味です)、空の文字列 ("") をその位置に代入します。

タイムスタンプの要件

認証済みリクエストには有効なタイムスタンプ (HTTP `Date` ヘッダーまたは `x-amz-date` 代替) が必ず必要です。また、認証済みリクエストに含まれるクライアントタイムスタンプは、リクエスト受信時の Amazon S3 システム時間から15分以内である必要があります。そうでない場合、リクエストは失敗し、*`RequestTimeTooSkewed`* エラーコードが返されます。このように制限することで、攻撃者によって傍受されたリクエストが繰り返される可能性を限定します。傍受に対する保護をさらに強化するには、認証済みリクエストに対して HTTPS 転送を使用します。

Note

リクエスト日の検証の制約は、クエリ文字列認証を使用しない認証済みリクエストにのみ適用されます。詳細については、「クエリ文字列による代替リクエスト認証 (p. 61)」を参照してください。

HTTP クライアントライブラリによっては、リクエストの `Date` ヘッダーを設定する機能が公開されていないことがあります。標準化ヘッダーの「Date」ヘッダーの値を含めることができない場合は、代わりに「`x-amz-date`」ヘッダーを使用してリクエストのタイムスタンプを設定します。`x-amz-date` ヘッダーの値は、RFC 2616 形式 (http://www.ietf.org/rfc/rfc2616.txt) の 1 つである必要があります。`x-amz-date` ヘッダーがリクエストに存在する場合は、すべての `Date` ヘッダーがリクエスト署名の計算時に無視されます。したがって、`x-amz-date` ヘッダーを含める場合、*`StringToSign`* を作成するときは、空の文字列を `Date` に対して使用します。例については、次のセクションを参照してください。

認証の例

このセクションの例では、次の表の認証情報 (実際は機能していません) を使用します。

パラメータ	値
AWSAccessKeyId	AKIAIOSFODNN7EXAMPLE
AWSSecretAccessKey	wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

StringToSign の例では、書式設定はそれほど複雑ではありません。\n は Unicode コードポイント U+000A を意味します。これは一般的には改行と呼ばれています。また、例では「+0000」を使用してタイムゾーンを指定しています。代わりに「GMT」を使用してタイムゾーンを指定することもできますが、署名はこの例に示したものとは異なることになります。

Example オブジェクト GET

この例では、johnsmith バケットからオブジェクトを取得します。

リクエスト	StringToSign
`GET /photos/puppy.jpg HTTP/1.1` `Host: johnsmith.s3.amazonaws.com` `Date: Tue, 27 Mar 2007 19:36:42 +0000` *`Authorization: AWS AKIAIOSFODNN7EXAMPLE: bWq2s1WEIj+Ydj0vQ697zp+IXMU=`*	`GET\n` `\n` `\n` `Tue, 27 Mar 2007 19:36:42 +0000\n` `/johnsmith/photos/puppy.jpg`

バケット名は CanonicalizedResource には含まれますが、HTTP リクエスト URI には含まれないことに注意してください。（これはホストヘッダーによって指定されます。）

Example Object PUT

この例では、オブジェクトを johnsmith バケットに配置します。

リクエスト	StringToSign
`PUT /photos/puppy.jpg HTTP/1.1` `Content-Type: image/jpeg` `Content-Length: 94328` `Host: johnsmith.s3.amazonaws.com` `Date: Tue, 27 Mar 2007 21:15:45 +0000` *`Authorization: AWS AKIAIOSFODNN7EXAMPLE: MyyxeRY7whkBe+bq8fHCL/2kKUg=`*	`PUT\n` `\n` `image/jpeg\n` `Tue, 27 Mar 2007 21:15:45 +0000\n` `/johnsmith/photos/puppy.jpg`

リクエスト内および StringToSign 内の Content-Type ヘッダーに注意してください。また、Content-MD5 はリクエストにないので、StringToSign では空白のままであることに注意してください。

Example リスト

この例では、johnsmith バケットのコンテンツを表示します。

リクエスト	StringToSign
`GET /?prefix=photos&max-keys=50&marker=puppy HTTP/1.1` `User-Agent: Mozilla/5.0` `Host: johnsmith.s3.amazonaws.com` `Date: Tue, 27 Mar 2007 19:42:41 +0000` *`Authorization: AWS AKIAIOSFODNN7EXAMPLE: htDYFYduRNen8P9ZfE/s9SuKy0U=`*	`GET\n` `\n` `\n` `Tue, 27 Mar 2007 19:42:41 +0000\n` `/johnsmith/`

CanonicalizedResource の後ろにスラッシュがあること、そしてクエリ文字列パラメータがないことに注意してください。

Example 取得

この例では、「johnsmith」バケットのアクセスコントロールポリシーのサブリソースを取得します。

リクエスト	StringToSign
`GET /?acl HTTP/1.1` `Host: johnsmith.s3.amazonaws.com` `Date: Tue, 27 Mar 2007 19:44:46 +0000` *`Authorization: AWS AKIAIOSFODNN7EXAMPLE: c2WLPFtWHVgbEmeEG93a4cG37dM=`*	`GET\n` `\n` `\n` `Tue, 27 Mar 2007 19:44:46 +0000\n` `/johnsmith/?acl`

サブリソースクエリ文字列パラメータがどのように CanonicalizedResource に挿入されているかに注意してください。

Example 削除

この例では、パス形式および Date 代替を使用して、「johnsmith」バケットからオブジェクトを削除します。

リクエスト	StringToSign
`DELETE /johnsmith/photos/puppy.jpg HTTP/1.1` `User-Agent: dotnet` `Host: s3.amazonaws.com` `Date: Tue, 27 Mar 2007 21:20:27 +0000` `x-amz-date: Tue, 27 Mar 2007 21:20:26 +0000` *`Authorization: AWS AKIAIOSFODNN7EXAMPLE:lx3byBScXR6KzyMaifNkardMwNk=`*	`DELETE\n` `\n` `\n` `Tue, 27 Mar 2007 21:20:26 +0000\n` `/johnsmith/photos/puppy.jpg`

代替の「x-amz-date」方式を使用して日付を指定する方法に注意してください (クライアントライブラリにより、日付を設定できない場合など) 。このような場合、`x-amz-date` が `Date` ヘッダーより優先されます。このため、署名内の日付エントリに `x-amz-date` ヘッダーの値を含める必要があります。

Example アップロード

この例では、オブジェクトを、メタデータが含まれる CNAME スタイルの仮想ホストバケットにアップロードします。

リクエスト	StringToSign
`PUT /db-backup.dat.gz HTTP/1.1` `User-Agent: curl/7.15.5` `Host: static.johnsmith.net:8080` `Date: Tue, 27 Mar 2007 21:06:08 +0000` `x-amz-acl: public-read` `content-type: application/x-download` `Content-MD5: 4gJE4saaMU4BqNR0kLY+lw==` `X-Amz-Meta-ReviewedBy: joe@johnsmith.net` `X-Amz-Meta-ReviewedBy: jane@johnsmith.net` `X-Amz-Meta-FileChecksum: 0x02661779` `X-Amz-Meta-ChecksumAlgorithm: crc32` `Content-Disposition: attachment; file name=database.dat` `Content-Encoding: gzip` `Content-Length: 5913339` *`Authorization: AWS AKIAIOSFODNN7EXAMPLE:ilyl83RwaSoYIEdixDQcA4OnAnc=`*	`PUT\n` `4gJE4saaMU4BqNR0kLY+lw==\n` `application/x-download\n` `Tue, 27 Mar 2007 21:06:08 +0000\n` `x-amz-acl:public-read\n` `x-amz-meta-checksumalgorithm:crc32\n` `x-amz-meta-filechecksum:0x02661779\n` `x-amz-meta-reviewedby:` `joe@johnsmith.net,jane@johnsmith.net\n` `/static.johnsmith.net/db-backup.dat.gz`

「x-amz-」ヘッダーが並べ替えられ、空白を削除されて小文字に変換される様子を観察してください。また、同じ名前を持つ複数のヘッダーが値の区切りとしてカンマを使用して結合されている様子も確認してください。

`Content-Type` および `Content-MD5` HTTP エンティティヘッダーのみが *`StringToSign`* に表示されていることに注意してください。もう一方の `Content-*` エンティティヘッダーは表示されていません。

また、バケット名は *`CanonicalizedResource`* には含まれますが、HTTP リクエスト URI には含まれないないことに注意してください (バケットはホストヘッダーによって指定されます) 。

Example My Buckets をすべて表示

リクエスト	StringToSign
`GET / HTTP/1.1` `Host: s3.amazonaws.com` `Date: Wed, 28 Mar 2007 01:29:59 +0000` *`Authorization: AWS AKIAIOSFODNN7EXAMPLE:qGdzdERIC03wnaRNKh6OqZehG9s=`*	`GET\n` `\n` `\n` `Wed, 28 Mar 2007 01:29:59 +0000\n` `/`

Example ユニコードキー

リクエスト	StringToSign
GET /diction ary/fran%C3%A7ais/pr%c3%a9f%c3%a8re HT TP/1.1 Host: s3.amazonaws.com Date: Wed, 28 Mar 2007 01:49:49 +0000 *Authorization: AWS AKIAIOSFODNN7EX* *AMPLE:DNEZGsoieTZ92F3bUfSPQcbGmlM=*	GET\n \n \n Wed, 28 Mar 2007 01:49:49 +0000\n /diction ary/fran%C3%A7ais/pr%c3%a9f%c3%a8re

Note

リクエスト URI から派生した *StringToSign* の要素は、URI エンコーディングおよび大文字/小文字の設定を含め、文字どおりに取得されます。

REST リクエストの署名に関する問題

REST リクエスト認証が失敗すると、システムは、そのリクエストに対して XML エラードキュメントで応答します。このエラードキュメントに含まれる情報は、開発者が問題を診断するのに役立ちます。特に、*SignatureDoesNotMatch* エラードキュメントの *StringToSign* 要素を確認すると、システムで使用されているリクエスト標準化が正確にわかります。

ツールキットの中には、知らないヘッダーを通知なしで事前に挿入するものがあります。例えば、PUT の実行中にヘッダー `Content-Type` が追加されることがあります。この場合、挿入されたヘッダーの値は一定であることがほとんどで、Ethereal、tcpmon などのツールを使用すると、不足しているヘッダーを検出できます。

クエリ文字列による代替リクエスト認証

`Authorization` HTTP ヘッダーを使用する代わりに、必要な情報をクエリ文字列パラメータとして渡すことで、特定の種類のリクエストを認証できます。これは、サードパーティのブラウザで、リクエストをプロキシに委任せずに、プライベートの Amazon S3 データに直接アクセスできるようにするときに便利です。これを行うには、「署名付き」のリクエストを作成し、エンドユーザーのブラウザが取得できる URL としてエンコードします。さらに、署名付きのリクエストは、有効期限を指定することで制限できます。

Note

AWS SDK を使用して署名付きの URL を生成する例については、「他ユーザーとのオブジェクトの共有 (p. 163)」を参照してください。

署名の作成

クエリ文字列認証済み Amazon S3 REST リクエストの例を次に示します。

```
GET /photos/puppy.jpg
?AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE&Expires=1141889120&Signature=vjbyPxybdZaN
mGa%2ByT272YEAiv4%3D HTTP/1.1
```

```
Host: johnsmith.s3.amazonaws.com
Date: Mon, 26 Mar 2007 19:37:58 +0000
```

クエリ文字列リクエスト認証方法には、特別な HTTP ヘッダーは必要ありません。代わりに、必要な認証要素は、クエリ文字列パラメータとして指定します。

クエリ文字列パラメータ名	値の例	説明
AWSAccessKeyId	AKIAIOSFODNN7EXAMPLE	使用する AWS アクセスキー ID。リクエストへの署名に使用する AWS シークレットアクセスキーを指定します。また、リクエストを行う開発者のアイデンティティを間接的に指定します。
Expires	1141889120	署名の有効期限。エポック (1970 年 1 月 1 日 00:00:00 UTC) からの時間を秒単位で指定します。この時刻 (サーバーの時刻) より後に受信したリクエストは拒否されます。
Signature	vjbyPxybdZaNmGa%2ByT272YEAiv4%3D	StringToSign の HMAC-SHA1 の Base64 エンコーディングの URL エンコーディング。

クエリ文字列リクエスト認証方法は、通常の方法とは若干異なりますが、*Signature* リクエストパラメータおよび *StringToSign* 要素の形式だけが異なります。クエリ文字列リクエスト認証方法を示す疑似文法を次に示します。

```
Signature = URL-Encode( Base64( HMAC-SHA1( YourSecretAccessKeyID, UTF-8-Encoding-
Of( StringToSign ) ) ) );

StringToSign = HTTP-VERB + "\n" +
    Content-MD5 + "\n" +
    Content-Type + "\n" +
    Expires + "\n" +
    CanonicalizedAmzHeaders +
    CanonicalizedResource;
```

YourSecretAccessKeyID は、アマゾン ウェブ サービスの開発者としてサインアップするときに、Amazon によって割り当てられる AWS シークレットアクセスキー ID です。*Signature* が、クエリ文字列内に適切に配置されるために URL エンコードされている点に注目してください。また、HTTP では `Date` であった位置要素が、*StringToSign* では *Expires* に置き換えられていることに注意してください。*CanonicalizedAmzHeaders* および *CanonicalizedResource* は同じです。

Note

クエリ文字列認証メソッドでは、署名する文字列を計算する際に、`Date` または `x-amz-date` `request` ヘッダーを使用しません。

Example クエリ文字列リクエスト認証

リクエスト	StringToSign
`GET /photos/puppy.jpg?AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE&` `  Signature=NpgCjnDzrM%2BWFzoENXmpNDUsSn8%3D&` `  Expires=1175139620 HTTP/1.1` `Host: johnsmith.s3.amazonaws.com`	`GET\n` `\n` `\n` `1175139620\n` `/johnsmith/photos/puppy.jpg`

ブラウザが GET リクエストを行うとき、Content-MD5 または Content-Type ヘッダーを提供しないこと、また、x-amz- ヘッダーを設定しないことが前提となっています。したがって、*StringToSign* のこの部分は空白のままです。

Base64 エンコーディングの使用

HMAC リクエスト署名は、Base64 エンコードされている必要があります。Base64 エンコーディングでは、署名を、リクエストにアタッチできるシンプルな ASCII 文字列に変換します。プラス (+) 、スラッシュ (/) 、等号 (=) などの署名文字列に含めることができる文字は、URI で使用する場合には、エンコードされている必要があります。例えば、認証コードにプラス記号 (+) が含まれる場合は、リクエストで %2B としてエンコードします。スラッシュは %2F、等号は %3D でエンコードします。

Base64 エンコーディングの例については、Amazon S3 の認証の例 (p. 56)を参照してください。

バケットの仮想ホスティング

Topics

一般的に仮想ホスティングとは、単一のウェブサーバーから複数のウェブサイトにサービスを提供することです。サイトを区別する方法の 1 つとして、単なる URI のパス名部分ではなく、リクエストの明確なホスト名を使用します。通常の Amazon S3 REST リクエストは、リクエスト URI パスのスラッシュで区切られた先頭コンポーネントを使用してバケットを指定します。または、Amazon S3 仮想ホスティングにより、HTTP `Host` ヘッダーを使用して REST API 呼び出しでバケットのアドレス指定を行うことができます。実際には、Amazon S3 は `Host` を、http://*bucketname*.s3.amazonaws.com でほとんどのバケットが (限定されたタイプのリクエストに対して) 自動的にアクセス可能になるという意味に解釈します。さらに、登録されたドメイン名を使用してバケットの名前を指定し、その名前を Amazon S3 の DNS エイリアスにすることによって、Amazon S3 リソースの URL を完全にカスタマイズすることができます (例: http://*my.bucketname.com*/) 。

カスタマイズされた URL の利便性に加えて、仮想ホスティングの 2 つ目の利点として、バケットの仮想サーバーの「ルートディレクトリ」に公開する機能があります。多くの既存のアプリケーションがこの標準ロケーションでファイルを検索するため、この機能は重要です。例えば、`favicon.ico`、`robots.txt`、`crossdomain.xml` はすべてルートで見つかることが見込まれています。

Important

Amazon S3 supports virtual-hosted-style and path-style access in all Regions. The path-style syntax, however, requires that you use the region-specific endpoint when attempting to access a bucket. For example, if you have a bucket called `mybucket` that resides in the EU, you want to use path-style syntax, and the object is named `puppy.jpg`, the correct URI is `http://s3-eu-west-1.amazonaws.com/mybucket/puppy.jpg`. You will receive a "PermanentRedirect" error, an HTTP response code 301, and a message indicating what the correct URI is for your resource if you try to access a non US Standard bucket with path-style syntax using:

- `http://s3.amazonaws.com`
- A different Region endpoint than where the bucket resides, for example, `http://s3-eu-west-1.amazonaws.com` and the bucket was created with the location constraint of Northern-California

Note

Amazon S3 は、デフォルトで仮想ホスティング形式のすべてのリクエストを米国スタンダードリージョンにルーティングします。いずれのリージョンでも、バケットを作成すると、Amazon S3 は DNS を更新してリクエストを正しい口ケーションに再ルーティングします。この処理には時間がかかる場合があります。その間、デフォルトのルールが適用されて、仮想ホスティング形式のリクエストは米国スタンダードリージョンに送信され、Amazon S3 は HTTP 307 リダイレクトによりこのリクエストを正しいリージョンにリダイレクトします。詳細については、「リクエストのリダイレクトと REST API (p. 440)」を参照してください。

SSL と共に仮想ホスティング形式のバケットを使用した場合、SSL ワイルドカード証明書は、ピリオドを含まないバケットのみと一致します。この問題を回避するには、HTTP を使用するか、または独自の証明書検証口ジックを記述します。

HTTP ホストヘッダーバケット仕様

GET リクエストが SSL エンドポイントを使用しないかぎり、HTTP `Host` ヘッダーを使用してリクエストに対してバケットを指定できます。REST リクエストの `Host` ヘッダーは、次のように解釈されます。

- `Host` ヘッダーを省略するか、その値を "s3.amazonaws.com" にすると、リクエスト URI のうち、スラッシュで区切られた最初の部分がリクエストのバケットになり、リクエスト URI の残りの部分はリクエストのキーとなります。次の表の最初および 2 つ目の例に示すように、これが通常の方式です。ホストヘッダーの省略は、HTTP 1.0 リクエストでのみ有効です。
- 前記のいずれの条件も該当せず、`Host` ヘッダーの値が ".s3.amazonaws.com" の場合、バケット名は `Host` ヘッダーの値のうち、先頭から ".s3.amazonaws.com" までの部分になります。リクエストのキーはリクエスト URI になります。このように解釈される場合、バケットは s3.amazonaws.com のサブドメインとして公開されます。この解釈は、後記のテーブルの 3 番目と 4 番目の例に示されています。
- それ以外の場合、リクエストのバケットは `Host` ヘッダーの小文字の値、リクエストのキーはリクエスト URI になります。この解釈は、バケット名と同じ DNS 名を登録しており、その名前を Amazon S3 の CNAME エイリアスとして設定した場合に有用です。このドキュメントではドメイン名の登録および DNS の設定の手順については扱いませんが、次の表の最後の例にその結果を示します。

例

このセクションでは、URL およびリクエストの例を示します。

Example パス形式の方法

この例では、バケット名として johnsmith.net を、キー名として homepage.html を使用します。

以下に、URL の例を示します。

```
http://s3.amazonaws.com/johnsmith.net/homepage.html
```

以下に、リクエストの例を示します。

```
GET /johnsmith.net/homepage.html HTTP/1.1
Host: s3.amazonaws.com
```

以下に、ホストヘッダーが省略された HTTP 1.0 によるリクエストの例を示します。

```
GET /johnsmith.net/homepage.html HTTP/1.0
```

DNS 互換名については、「制約事項 (p. 67)」を参照してください。キーの詳細については、「キー (p. 4)」を参照してください。

Example 仮想ホスティング形式の方法

この例では、バケット名として johnsmith.net を、キー名として homepage.html を使用します。

以下に、URL の例を示します。

```
http://johnsmith.net.s3.amazonaws.com/homepage.html
```

以下に、リクエストの例を示します。

```
GET /homepage.html HTTP/1.1
Host: johnsmith.net.s3.amazonaws.com
```

以下に、大文字と小文字が誤って使用されたリクエストの例を示します。文の大文字小文字は関係ありません。ただし、この方式を使用して大文字のバケットにアクセスすることはできません。

```
GET /homepage.html HTTP/1.1
Host: JohnSmith.net.s3.amazonaws.com
```

Example 米国スタンダード以外のバケット用の仮想ホスト形式のメソッド

この例では、欧州 (アイルランド) リージョンのバケットの名前として johnsmith.eu を、キー名として homepage.html を使用します。

以下に、URL の例を示します。

```
http://johnsmith.eu.s3-eu-west-1.amazonaws.com/homepage.html
```

以下に、リクエストの例を示します。

```
GET /homepage.html HTTP/1.1
Host: johnsmith.eu.s3-eu-west-1.amazonaws.com
```

Example CNAME 方式

この例では、バケット名として www.johnsmith.net を、キー名として homepage.html を使用します。この方式を使用するには、DNS 名を *bucketname*.s3.amazonaws.com の CNAME エイリアスとして設定する必要があります。

以下に、URL の例を示します。

```
http://www.johnsmith.net/homepage.html
```

以下に、リクエストの例を示します。

```
GET /homepage.html HTTP/1.1
Host: www.johnsmith.net
```

CNAME による Amazon S3 URL のカスタマイズ

要件によっては、ウェブサイトやウェブサービスに "s3.amazonaws.com" を表示したくない場合もあります。例えば、Amazon S3 でウェブサイトのイメージをホストする場合に、http://johnsmith-images.s3.amazonaws.com/. ではなく http://images.johnsmith.net/ が表示されるようにするとします。

バケット名は CNAME と同じである必要があります。その場合、http://images.johnsmith.net/filename は http://images.johnsmith.net.s3.amazonaws.com/filename と同じになります (ただし、images.johnsmith.net が images.johnsmith.net.s3.amazonaws.com にマップされるように CNAME が作成されている必要があります)。

DNS 互換名を持つすべてのバケットは、http://[*bucketname*].s3.amazonaws.com/[*Filename*] (例: http://images.johnsmith.net.s3.amazonaws.com/mydog.jpg) として参照することができます。CNAME を使用して images.johnsmith.net を Amazon S3 ホスト名にマップすることができます。その場合、前述の URL は http://images.johnsmith.net/mydog.jpg になります。

CNAME DNS レコードは、ドメイン名を適切な仮想ホスティング形式のホスト名にエイリアス設定する必要があります。例えば、バケット名 (およびドメイン名) が images.johnsmith.net である場合、CNAME レコードは images.johnsmith.net.s3.amazonaws.com にエイリアス設定する必要があります。

```
images.johnsmith.net CNAME    images.johnsmith.net.s3.amazonaws.com.
```

エイリアス先を s3.amazonaws.com に設定することも有効ですが、余分な HTTP リダイレクトが発生する場合があります。

Amazon S3 はホスト名を使用してバケット名を決定します。例えば、www.example.com を www.example.com.s3.amazonaws.com の CNAME として設定したとします。http://www.example.com にアクセスすると、Amazon S3 は次のようなリクエストを受け取ります。

```
GET / HTTP/1.1
Host: www.example.com
Date: date
Authorization: signatureValue
```

Amazon S3 は元のホスト名 www.example.com のみを認識し、リクエストの解決に使用される CNAME マッピングは認識しないため、CNAME とバケット名は同じである必要があります。

CNAME では、どの Amazon S3 エンドポイントも使用できます。例えば、s3-ap-southeast-1.amazonaws.com を使用できます。エンドポイントの詳細については、「Request Endpoints (p. 13)」を参照してください。

CNAME を使用してホスト名を Amazon S3 バケットに関連付けるには

1. 管理するドメインに属するホスト名を選択します。
   この例では、`johnsmith.net` ドメインの `images` サブドメインを使用します。
2. ホスト名と一致するバケットを作成します。
   この例では、ホスト名およびバケット名は `images.johnsmith.net` です。

   Note

   バケット名はホスト名と完全に一致する必要があります。

3. ホスト名を Amazon S3 バケットのエイリアスとして定義する CNAME レコードを作成します。次に例を示します。

   ```
   images.johnsmith.net CNAME images.johnsmith.net.s3.amazonaws.com
   ```

   Important

   リクエストルーティングの理由により、CNAME レコードは、前述の例と完全に一致するように定義する必要があります。そうしない場合、正しく動作するように見えても、最終的には予期しない結果を招くことがあります。

   Note

   DNS を設定する正確な手順は、DNS サーバーまたは DNS プロバイダによって異なり、このドキュメントでは扱いません。

制約事項

DNS 名では大文字と小文字が区別されないため、仮想ホスティング方式を使用して小文字のバケットのみをアドレス指定できます。詳細については、「バケットの制約と制限 (p. 88)」を参照してください。

HTTP `Host` ヘッダーを使用してリクエストのバケットを指定する処理は、非 SSL リクエストで、REST API の使用時にサポートされます。別のエンドポイントを使用して SOAP でバケットを指定することはできません。

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

下位互換性

以前のバージョンの Amazon S3 は、HTTP `Host` ヘッダーを不正に無視していました。文書化されていないこの動作に依存するアプリケーションは、`Host` ヘッダーを正しく設定するように更新する必要

があります。Amazon S3 は Host（存在する場合）からバケット名を判断するため、この問題の最も起こりうる症状として、予期せぬ NoSuchBucket エラー結果コードを受け取ることがあります。

リクエストのリダイレクトと REST API

Topics

このセクションでは、REST を使用して HTTP リダイレクトを処理する方法について説明します。Amazon S3 のリダイレクトについては、「リクエストのリダイレクトと REST API (p. 440)」を参照してください。

リダイレクトおよび HTTP ユーザーエージェント

Amazon S3 REST API を使用するプログラムは、アプリケーションレイヤーレベルまたは HTTP レイヤーレベルのいずれかでリダイレクトを処理する必要があります。リダイレクトが適切に処理されるように、多くの HTTP クライアントライブラリおよびユーザーエージェントを自動的に設定することができます。しかし、それでもリダイレクトの実装が不適切または不完全であることはよくあります。

ライブラリでリダイレクト要件を満たすようにする前に、次のテストを行ってください。

プロセスを起動

1	すべての HTTP リクエストヘッダーが（Authorization、Date などの HTTP 標準を含む）、リダイレクトされたリクエスト（リダイレクトの受信後の 2 番目のリクエスト）に適切に含まれていることを確認します。
2	PUT、DELETE など、非 GET リダイレクトが正しく動作することを確認します。
3	大きな PUT リクエストがリダイレクトに正しく従っていることを確認します。
4	100-continue レスポンスが届くまでに時間がかかる場合は、PUT リクエストがリダイレクトに正しく従っていることを確認します。

HTTP リクエストが GET または HEAD でない場合、RFC 2616 に厳密に準拠する HTTP ユーザーエージェントは、リダイレクトにしたがう前に明示的な確認が必要になることがあります。一般的に、Amazon S3 によって自動生成されたリダイレクトに従うと安全です。システムのみが amazonaws.com ドメイン内でホストに対してリダイレクトを発行し、リダイレクトされたリクエストの効果が元のリダイレクトと同じになるからです。

リダイレクトおよび 100-continue

リダイレクトの処理を簡素化し、効率性を高め、リダイレクトされたリクエスト本文を 2 回送信するためのコストを排除するには、PUT オペレーションに対して 100-continue を使用するようにアプリケーションを設定します。100-continue を使用しているアプリケーションは、確認を受け取るまでリクエスト本文を送信しません。ヘッダーに基づいてメッセージが拒否された場合、メッセージの本文は送信されません。100-continue の詳細については、「RFC 2616 Section 8.2.3」を参照してください。

Note

RFC 2616 にしたがって、不明な HTTP サーバーで Expect: Continue を使用する場合は、リクエスト本文を送信する前の待機時間は無期限にしないでください。HTTP サーバーの中には、100-continue を認識しないものがあるからです。しかし、Amazon S3 は、リクエストに

`Expect: Continue` が含まれているかどうか、また、一時的な 100-continue ステータスを返すか、最終ステータスコードを返すかを認識します。さらに、一時的な 100 continue の承認を受け取った後、リダイレクトエラーが発生することはありません。これは、リクエスト本文を書き込んでいる間は、リダイレクトレスポンスを受け取らないようにする際に役立ちます。

リダイレクトの例

このセクションでは、HTTP リダイレクトと 100-continue を使用したクライアントとサーバーのやり取りの例を紹介します。

`quotes.s3.amazonaws.com` バケットに対する PUT の例を次に示します。

```
PUT /nelson.txt HTTP/1.1
Host: quotes.s3.amazonaws.com
Date: Mon, 15 Oct 2007 22:18:46 +0000

Content-Length: 6
Expect: 100-continue
```

Amazon S3 は以下を返します。

```
HTTP/1.1 307 Temporary Redirect
Location: http://quotes.s3-4c25d83b.amazonaws.com/nelson.txt?rk=8d47490b
Content-Type: application/xml
Transfer-Encoding: chunked
Date: Mon, 15 Oct 2007 22:18:46 GMT

Server: AmazonS3

<?xml version="1.0" encoding="UTF-8"?>
<Error>
  <Code>TemporaryRedirect</Code>
  <Message>Please re-send this request to the
  specified temporary endpoint. Continue to use the
  original request endpoint for future requests.
  </Message>
  <Endpoint>quotes.s3-4c25d83b.amazonaws.com</Endpoint>
  <Bucket>quotes</Bucket>
</Error>
```

クライアントは、リダイレクトレスポンスにしたがって、新しいリクエストを `quotes.s3-4c25d83b.amazonaws.com` 一時エンドポイントに発行します。

```
PUT /nelson.txt?rk=8d47490b HTTP/1.1
Host: quotes.s3-4c25d83b.amazonaws.com
Date: Mon, 15 Oct 2007 22:18:46 +0000

Content-Length: 6
Expect: 100-continue
```

Amazon S3 は 100-continue 返します。これは、クライアントがリクエスト本文を送信し続ける必要があることを示します。

```
HTTP/1.1 100 Continue
```

クライアントはリクエスト本文を送信します。

```
ha ha\n
```

Amazon S3 は最終レスポンスを返します。

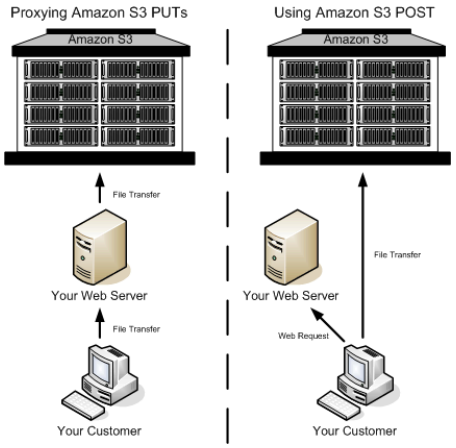
```
HTTP/1.1 200 OK
Date: Mon, 15 Oct 2007 22:18:48 GMT

ETag: "a2c8d6b872054293afd41061e93bc289"
Content-Length: 0
Server: AmazonS3
```

POST を使用したブラウザベースのアップロード

Topics

Amazon S3 は POST をサポートします。POST を使用すると、ユーザーがコンテンツを直接 Amazon S3 にアップロードできます。POST は、アップロードを簡素化し、アップロードのレイテンシーを短縮できるようにできるよう設計されています。また、ユーザーが Amazon S3 にデータをアップロードして保存するためのアプリケーションに対する費用を節約することもできます。

次の図は、Amazon S3 POST を使用したアップロードを示しています。



POST を使用したアップロード

1	ユーザーは、ウェブブラウザを開いて、開発者のウェブページにアクセスします。
2	開発者のウェブページには HTTP フォームが含まれ、このフォームには、ユーザーがコンテンツを Amazon S3 にアップロードする際に必要な情報がすべて含まれています。
3	ユーザーはコンテンツを直接 Amazon S3 にアップロードします。

Note

POST では、クエリ文字列認証がサポートされていません。

HTML フォーム

Topics

Amazon S3 と通信する場合、通常は、REST または SOAP API を使用して、put、get、delete などのオペレーションを実行します。POST を使用した場合、ユーザーは自分のブラウザでデータを直接 Amazon S3 にアップロードします。これらのブラウザでは、SOAP API または REST PUT リクエストを行う方法が認識されません。

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

ユーザーが自分のブラウザを使用してコンテンツを Amazon S3 にアップロードできるようにするには、HTML フォームを使用します。HTML フォームは、フォーム宣言とフォームフィールドで構成されます。フォーム宣言には、リクエストの概要情報が含まれています。フォームフィールドには、リクエストの詳細情報と、リクエストを認証し、指定した条件をそのリクエストが確実に満たすようにするためのポリシーが含まれています。

Note

フォームデータと境界 (ファイルのコンテンツは除く) は 20 K を超えることはできません。

このセクションでは、HTML フォームを使用する方法について説明します。

HTML フォームのエンコーディング

フォームとポリシーは UTF-8 エンコーディングされている必要があります。UTF-8 エンコードをフォームに適用するには、そのエンコードを HTML 見出しで指定するかリクエストヘッダーとして指定します。

Note

HTML フォーム宣言は、クエリ文字列認証パラメータを受け入れません。クエリ文字列認証については、「クエリ文字列認証の使用 (p. 363)」を参照してください。

HTML 見出しの UTF-8 エンコードの例を次に示します。

```
<html>
  <head>
    ...
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    ...
  </head>
  <body>
```

リクエストヘッダーの UTF-8 エンコードの例を次に示します。

```
Content-Type: text/html; charset=UTF-8
```

HTML フォーム宣言

フォーム宣言には、アクション、メソッド、エンクロージャタイプの 3 つのコンポーネントが含まれます。これらの値が適切に設定されていないと、リクエストは失敗します。

アクションは、リクエストを処理する URL を指定します。これはバケットの URL に設定されていなければなりません。例えば、バケットの名前が「johnsmith」の場合、URL は「http://johnsmith.s3.amazonaws.com/」です。

Note

キー名はフォームフィールドで指定されます。

メソッドは POST である必要があります。

エンクロージャタイプ (enctype) を指定する必要があり、ファイルアップロードとテキストエリアアップロードの両方に対して、multipart/form-data (RFC 1867 を参照) が設定されていなければなりません。

Example

バケット「johnsmith」のフォーム宣言を次に示します。

```
<form action="http://johnsmith.s3.amazonaws.com/" method="post"

enctype="multipart/form-data">
```

HTML フォームフィールド

次の表では、フォーム内で使用できる一連のフィールドについて説明します。

Note

変数 `${filename}` は、ユーザーによって指定されたファイル名に自動的に置き換えられ、すべてのフォームフィールドで認識されます。ブラウザまたはクライアントによって完全パスまたは部分パスがファイルに提供されると、最後のスラッシュ (/) またはバックスラッシュ (\ ) に続くテキストのみが使用されます (例えば、「C:\Program Files\directory1\file.txt」は、「file.txt」と解釈されます)。ファイルまたはファイル名が指定されていない場合、変数は空の文字列に置き換えられます。

要素名	説明	必須
AWSAccessKeyId	ポリシー内の一連の制約を満たすリクエストに対して匿名ユーザーアクセスを付与するバケット所有者の AWS アクセスキー ID。ポリシードキュメントがリクエストに含まれる場合に必要です。	条件付き
acl	Amazon S3 アクセスコントロールリストを指定します。無効なアクセスコントロールリストが指定されると、エラーが生成されます。ACL の詳細については、「Introduction to Amazon S3」の「Access Control Lists (p. 8)」を参照してください。 型: 文字列型 デフォルト: private Valid Values: private | public-read | public-read-write | authenticated-read | bucket-owner-read | bucket-owner-full-control	いいえ
Cache-Control, Content-Type, Content-Disposition, Content-Encoding, Expires	REST 固有ヘッダー。詳細については、「PUT Object」を参照してください。	いいえ
key	アップロードされたキーの名前。 ユーザーによって指定されたファイル名を使用するには、${filename} 変数を使用します。例えば、Betty がファイル lolcatz.jpg をアップロードし、開発者が /user/betty/${filename} を指定すると、そのファイルは /user/betty/lolcatz.jpg として保存されます。 詳細については、「オブジェクトキーとメタデータ (p. 108)」を参照してください。	あり
policy	リクエストで許可されているものについて説明するセキュリティポリシー。セキュリティポリシーがないリクエストは匿名と見なされ、誰でも書き込むことができるバケットでのみ動作します。	なし

要素名	説明	必須
success_action_redirect, redirect	アップロードが成功したときにクライアントがリダイレクトされる URL。Amazon S3 は、バケット、キー、および etag 値を、クエリ文字列パラメータとして URL に追加します。 success_action_redirect が指定されていない場合、Amazon S3 は、success_action_status フィールドで指定された空のドキュメントタイプを返します。 Amazon S3 が URL を解釈できない場合、その Amazon S3 は、フィールドが存在しないものとして動作します。 アップロードが失敗した場合、Amazon S3 にはエラーが表示され、ユーザーは URL にリダイレクトされません。 詳細については、「Redirection (p. 79)」を参照してください。  Note リダイレクトフィールド名は廃止されています。このフィールド名のサポートは削除される予定です。	いいえ
success_action_status	アップロードが正常に行われたとき、success_action_redirect が指定されていない場合にクライアントに返されるステータスコード。 200、201、または 204 (デフォルト) を使用できます。 200 または 204 が設定されている場合は、空のドキュメントとステータスコード 200 または 204 が返されます。 201 が設定されている場合は、XML ドキュメントとステータスコード 201 が返されます。XML ドキュメントのコンテンツについては、「POST Object」を参照してください。 値が設定されていないか、無効な値が設定されている場合は、空のドキュメントとステータスコード 204 が返されます。  Note Adobe Flash Player のバージョンによっては、本文が空の HTTP レスポンスが適切に処理されないことがあります。Adobe Flash でアップロードをサポートするには、*success_action_status* を 201 に設定することをお勧めします。	いいえ

要素名	説明	必須
signature	指定された AWSAccessKeyId のシークレットキーを使用して作成された HMAC 署名。ポリシードキュメントがリクエストに含まれる場合に必要です。 詳細については、「Using Auth Access」を参照してください。	条件付き
x-amz-security-token	Amazon DevPay セキュリティトークン。 Amazon DevPay を使用するリクエストごとに 2 つの x-amz-security-token フォームフィールドが必要です。1 つは製品トークン、もう 1 つはユーザートークン用のフィールドです。 詳細については、「Using Amazon DevPay with Amazon S3」を参照してください。	いいえ
プレフィックスが x-amz-meta- のその他のフィールド名	ユーザーが指定したメタデータ。 Amazon S3 ではこのデータは検証または使用されません。 詳細については、「PUT Object」を参照してください。	いいえ
ファイル	ファイルまたはテキストコンテンツ。 ファイルまたはコンテンツはフォーム内の最後のフィールドでなければなりません。その下のフィールドはすべて無視されます。 複数のファイルを一度にアップロードすることはできません。	はい

ポリシーの作成

Topics

- 有効期限 (p. 76)
- 条件 (p. 76)
- 条件一致 (p. 77)
- 文字のエスケープ (p. 78)

ポリシーは、UTF-8 および Base64 エンコードされた JSON ドキュメントで、リクエストが満たさなければならない条件を指定します。また、コンテンツを認証するときにも使用されます。ポリシードキュメントは、どのように設計するかに応じて、アップロードごと、ユーザーごと、すべてのアップロードに対して、またはニーズに応じたその他の設計にしたがって使用できます。

Note

ポリシードキュメントはオプションですが、バケットを誰でも書き込むことができるようにするよりも、ポリシードキュメントを使用することを強くお勧めします。

ポリシードキュメントの例を次に示します。

```
{ "expiration": "2007-12-01T12:00:00.000Z",

  "conditions": [

    {"acl": "public-read" },

    {"bucket": "johnsmith" },

    ["starts-with", "$key", "user/eric/"],

  ]

}
```

ポリシードキュメントには、有効期限と条件が含まれます。

有効期限

有効期限は、ポリシーの有効期限日を ISO8601 GMT 日付形式で指定します。例えば、「2007-12-01T12:00:00.000Z」は、2007 年 12 月 1 日 12:00 GMT 以降、ポリシーが無効になることを指定します。ポリシーには有効期限が必ず必要です。

条件

ポリシードキュメントの条件は、アップロードされたオブジェクトのコンテンツを検証するときに使用されます。条件の一覧には、フォームで指定した各フォームフィールドが含まれていなければなりません (AWSAccessKeyId、署名、ポリシー、および x-ignore- プレフィックスが付いているフィールド名を除く) 。

Note

同じ名前のフィールドが複数ある場合、そのフィールドの値はコンマで区切る必要があります。例えば、「x-amz-meta-tag」という名前のフィールドが 2 つあり、最初のフィールドの値が「Ninja」、2 つ目のフィールドの値が「Stallman」の場合、ポリシードキュメントは `Ninja,Stallman` に設定します。

フォーム内のすべての変数が、ポリシーの検証前に展開されます。したがって、条件との照合は必ず、展開されたフィールドに対して行われます。例えば、キーフィールドを `user/betty/${filename}` に設定した場合、ポリシーは `[ "starts-with", "$key", "user/betty/" ]` になる可能性があります。`[ "starts-with", "$key", "user/betty/${filename}" ]` は入力しないでください。詳細については、「条件一致 (p. 77)」を参照してください。

次の表では、ポリシードキュメントの条件について説明します。

要素名	説明
acl	ACL が満たす必要がある条件を指定します。 完全一致および *`starts-with`* をサポートします。
content-length-range	アップロードされたコンテンツの最小サイズと最大サイズを指定します。 範囲一致をサポートします。

要素名	説明
Cache-Control、Content-Type、Content-Disposition、Content-Encoding、Expires	REST 固有ヘッダー。 完全一致および *starts-with* をサポートします。
key	アップロードされたキーの名前。 完全一致および *starts-with* をサポートします。
success_action_redirect、redirect	アップロードが成功したときにクライアントがリダイレクトされる URL。 完全一致および *starts-with* をサポートします。
success_action_status	アップロードが正常に行われたとき、success_action_redirect が指定されていない場合にクライアントに返されるステータスコード。 完全一致をサポートします。
x-amz-security-token	Amazon DevPay セキュリティトークン。 Amazon DevPay を使用するリクエストごとに 2 つの `x-amz-security-token` フォームフィールドが必要です。1 つは製品トークン、もう 1 つはユーザートークン用のフィールドです。結果的に、値はコンマで区切られていなければなりません。例えば、ユーザートークンが *eW91dHViZQ==* で、製品トークンが *b0hnNVNKWVJIQTA=* の場合、ポリシーエントリは `{ "x-amz-security-token": "eW91dHViZQ==,b0hnNVNKWVJIQTA=" }` に設定します。 Amazon DevPay の詳細については、「Using Amazon DevPay with Amazon S3」を参照してください。
プレフィックスが x-amz-meta- のその他のフィールド名	ユーザーが指定したメタデータ。 完全一致および *starts-with* をサポートします。

Note

ツールキットにより追加のフィールドが追加されている場合は (Flash では filename が追加される)、そのフィールドをポリシードキュメントに追加する必要があります。この機能をコントロールできる場合は、この機能が無視されるようにプレフィックス `x-ignore-` をフィールドに追加します。これは、この機能の今後のバージョンには影響しません。

条件一致

次の表では、条件一致の種類について説明します。フォーム内で指定したフォームフィールドごとに 1 つの条件を指定する必要がありますが、1 つのフォームフィールドに複数の条件を指定し、より複雑な一致条件を作成することもできます。

条件	説明
完全一致	完全一致では、フィールドが特定の値と一致するかどうかを確認します。次の例は、ACL を public-read に設定する必要があることを示しています。 `{"acl": "public-read" }` ACL を public-read に設定する必要があることを示す他の例を次に示します。 `[ "eq", "$acl", "public-read" ]`
先頭が一致	先頭の値を指定する場合は、starts-with を使用します。次の例は、キーが user/betty で開始する必要があることを示しています。 `["starts-with", "$key", "user/betty/"]`
任意のコンテンツに一致	フィールド内の任意のコンテンツを許可するようにポリシーを設定するには、starts-with に空の値を使用します。次の例は、任意の success_action_redirect を許可します。 `["starts-with", "$success_action_redirect", ""]`
範囲を指定	範囲指定できるフィールドについては、範囲の上限値と下限値をコンマで区切ります。次の例は、1～10 メガバイトのサイズのファイルを許可します。 `["content-length-range", 1048579, 10485760]`

文字のエスケープ

次の表では、ポリシードキュメント内でエスケープする必要がある文字について説明します。

エスケープシーケンス	説明
\\	バックスラッシュ
\$	ドル記号
\b	バックスペース
\f	フォームフィード
\n	改行
\r	復帰文字
\t	水平タブ
\v	垂直タブ

エスケープシーケンス	説明
\u*xxxx*	すべての Unicode 文字

署名の設定

手順	説明
1	UTF-8 を使用してポリシーをエンコードします。
2	Base64 を使用して、その UTF-8 バイトをエンコードします。
3	HMAC SHA-1 を使用して、シークレットアクセスキーでポリシーに署名します。
4	Base64 を使用して、SHA-1 署名をエンコードします。

認証については、「Using Auth Access」を参照してください。

リダイレクト

このセクションでは、リダイレクトを処理する方法について説明します。

一般的なリダイレクト

POST の完了時に、ユーザーは、success_action_redirect フィールドで指定した場所にリダイレクトされます。Amazon S3 が URL を解釈できない場合、`success_action_redirect` フィールドは無視されます。

`success_action_redirect` が指定されていない場合、Amazon S3 は、`success_action_status` フィールドで指定された空のドキュメントタイプを返します。

POST が失敗した場合、Amazon S3 にはエラーが表示され、リダイレクトは行われません。

アップロード前のリダイレクト

<CreateBucketConfiguration> を使用してバケットが作成された場合、エンドユーザーをリダイレクトしなければならない可能性があります。これが発生した場合、ブラウザによっては、リダイレクトが正しく処理されないことがあります。比較的まれですが、バケットの作成直後に発生する可能性があります。

アップロードの例

Topics

ファイルのアップロード

この例は、ポリシーおよびフォームを作成し、添付ファイルをアップロードするプロセスを示しています。

ポリシーとフォームの作成

johnsmith バケットについて、Amazon S3 へのアップロードをサポートするポリシーを次に示します。

```
{ "expiration": "2007-12-01T12:00:00.000Z",
  "conditions": [
    {"bucket": "johnsmith"},
    ["starts-with", "$key", "user/eric/"],
    {"acl": "public-read"},
   {"success_action_redirect": "http://johnsmith.s3.amazonaws.com/successful_up
load.html"},
    ["starts-with", "$Content-Type", "image/"],
    {"x-amz-meta-uuid": "14365123651274"},
    ["starts-with", "$x-amz-meta-tag", ""]
  ]
}
```

このポリシーでは、以下のように規定されています。

- アップロードは 2007 年 12 月 1 日 12:00 GMT よりも前に行う必要がある
- コンテンツは johnsmith バケットにアップロードする必要がある
- キーの先頭は「user/eric/」でなければならない
- ACL が public-read に設定されている
- success_action_redirect が http://johnsmith.s3.amazonaws.com/successful_upload.html に設定されている必要がある
- オブジェクトはイメージファイルである
- x-amz-meta-uuid タグが 14365123651274 に設定されている必要がある
- x-amz-meta-tag には任意の値を含めることができる

このポリシーの Base64 エンコードされたバージョンを次に示します。

```
eyAiZXhwaXJhdGlvbiI6ICIyMDA3LTEyLTAxVDEyOjAwOjAwLjAwMFoiLAogICJjb25kaXRpb25zI
jogWwogICAgeyJidWNrZXQiOiAiam9obnNtaXRoIn0sCiAgICBbInN0YXJ0cy13aXRoIiwgIiRrZXkiL
CAidXNlci9lcmljLyJdLAogICAgeyJhY2wiOiAicHVibGljLXJlYWQifSwKICAgIH
sic3VjY2Vzc19hY3Rpb25fcmVkaXJlY3QiOiAiaHR0cDovL2pvaG5zbWl0aC5zMy5hb
WF6b25hd3MuY29tL3N1Y2Nlc3NmdWxfdXBsb2FkLmh0bWwifSwKICAgIF
sic3RhcnRzLXdpdGgiL
CAiJENvbnRlbnQtVHlwZSIsICJpbWFnZS8iXSwKICAgIHsieC1hbXotbWV0YS11dWlkIjogIjE0MzY1MT
IzNjUxMjc0In0sCiAgICBbInN0YXJ0cy13aXRoIiwgIiR4LWFtei1tZXRhLXRhZyIsICIiXQo
gIF0KfQo=
```

認証情報を使用すると署名が作成されます。例えば、`0RavWzkygo6QX9caELEqKi9kDbU=` は、前述のポリシードキュメントの署名です。

このポリシーを使用して johnsmith.net バケットへの POST をサポートするフォームを次に示します。

```
<html>
  <head>
    ...
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    ...
```

```
  </head>
  <body>
  ...
  <form action="http://johnsmith.s3.amazonaws.com/" method="post" enctype="mul
tipart/form-data">
    Key to upload: <input type="input" name="key" value="user/eric/" /><br />
    <input type="hidden" name="acl" value="public-read" />
    <input type="hidden" name="success_action_redirect" value="http://johns
mith.s3.amazonaws.com/successful_upload.html" />
    Content-Type: <input type="input" name="Content-Type" value="image/jpeg"
/><br />
    <input type="hidden" name="x-amz-meta-uuid" value="14365123651274" />
    Tags for File: <input type="input" name="x-amz-meta-tag" value="" /><br />

    <input type="hidden" name="AWSAccessKeyId" value="AKIAIOSFODNN7EXAMPLE" />

    <input type="hidden" name="Policy" value="POLICY" />
    <input type="hidden" name="Signature" value="SIGNATURE" />
    File: <input type="file" name="file" /> <br />
    <!-- The elements after this will be ignored -->
    <input type="submit" name="submit" value="Upload to Amazon S3" />
  </form>
  ...
</html>
```

リクエスト例

このリクエストでは、アップロードされたイメージが 117,108 バイト (イメージデータを除く) であることを前提としています。

```
POST / HTTP/1.1
Host: johnsmith.s3.amazonaws.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.10)
Gecko/20071115 Firefox/2.0.0.10
Accept: text/xml,application/xml,applica
tion/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Content-Type: multipart/form-data; boundary=9431149156168
Content-Length: 118698

--9431149156168
Content-Disposition: form-data; name="key"

user/eric/MyPicture.jpg
--9431149156168
Content-Disposition: form-data; name="acl"

public-read
--9431149156168
Content-Disposition: form-data; name="success_action_redirect"

http://johnsmith.s3.amazonaws.com/successful_upload.html
--9431149156168
```

```
Content-Disposition: form-data; name="Content-Type"

image/jpeg
--9431149156168
Content-Disposition: form-data; name="x-amz-meta-uuid"

14365123651274
--9431149156168
Content-Disposition: form-data; name="x-amz-meta-tag"

Some,Tag,For,Picture
--9431149156168
Content-Disposition: form-data; name="AWSAccessKeyId"

AKIAIOSFODNN7EXAMPLE
--9431149156168
Content-Disposition: form-data; name="Policy"

eyAiZXhwaXJhdGlvbiI6ICIyMDA3LTEyLTAxVDEyOjAwOjAwLjAwMFoiLAogICJjb25kaXRpb25zI
jogWwogICAgeyJidWNrZXQiOiAiam9obnNtaXRoIn0sCiAgICBbInN0YXJ0cy13aXRoIiwgIiRrZXkiL
CAidXNlci9lcmljLyJdLAogICAgeyJhY2wiOiAicHVibGljLXJlYWQifSwKICAgIH
sic3VjY2Vzc19hY3Rpb25fcmVkaXJlY3QiOiAiaHR0cDovL2pvaG5zbWl0aC5zMy5hb
WF6b25hd3MuY29tL3N1Y2Nlc3NmdWxfdXBsb2FkLmh0bWwifSwKICAgIFsic3RhcnRzLXdpdGgiL
CAiJENvbnRlbnQtVHlwZSIsICJpbWFnZS8iXSwKICAgIHsieC1hbXotbWV0YS11dWlkIjogIjE0MzY1MT
IzNjUxMjc0In0sCiAgICBbInN0YXJ0cy13aXRoIiwgIiR4LWFtei1tZXRhLXRhZyIsICIiXQo
gIF0KfQo=
--9431149156168
Content-Disposition: form-data; name="Signature"

0RavWzkygo6QX9caELEqKi9kDbU=
--9431149156168
Content-Disposition: form-data; name="file"; filename="MyFilename.jpg"
Content-Type: image/jpeg

...file content...
--9431149156168
Content-Disposition: form-data; name="submit"

Upload to Amazon S3
--9431149156168--
```

レスポンス例

```
HTTP/1.1 303 Redirect
x-amz-request-id: 1AEE782442F35865
x-amz-id-2: cxzFLJRatFHy+NGtaDFRR8YvI9BHmgLxjvJzNiGGICARZ/mVXHj7T+qQKhdpzHFh
Content-Type: application/xml
Date: Wed, 14 Nov 2007 21:21:33 GMT
Connection: close
Location: http://johnsmith.s3.amazonaws.com/successful_upload.html?bucket=john
smith&key=user/eric/MyPicture.jpg&etag=&quot;39d459df
bc0faabbb5e179358dfb94c3&quot;
Server: AmazonS3
```

テキストエリアのアップロード

Topics

この例は、ポリシーおよびフォームを作成し、テキストエリアをアップロードするプロセスを示しています。これは、ブログの投稿など、ユーザーが作成したコンテンツを送信するときに便利です。

ポリシーとフォームの作成

johnsmith バケットについて、Amazon S3 へのテキストエリアのアップロードをサポートするポリシーを次に示します。

```
{ "expiration": "2007-12-01T12:00:00.000Z",
  "conditions": [
    {"bucket": "johnsmith"},
    ["starts-with", "$key", "user/eric/"],
    {"acl": "public-read"},
    {"success_action_redirect": "http://johnsmith.s3.amazon
aws.com/new_post.html"},
    ["eq", "$Content-Type", "text/html"],
    {"x-amz-meta-uuid": "14365123651274"},
    ["starts-with", "$x-amz-meta-tag", ""]
  ]
}
```

このポリシーでは、以下のように規定されています。

- アップロードは 2007 年 12 月 1 日 12:00 GMT よりも前に行う必要がある
- コンテンツは johnsmith バケットにアップロードする必要がある
- キーの先頭は「user/eric/」でなければならない
- ACL が public-read に設定されている
- success_action_redirect が http://johnsmith.s3.amazonaws.com/new_post.html に設定されている必要がある
- オブジェクトは HTML テキストである
- x-amz-meta-uuid タグが 14365123651274 に設定されている必要がある
- x-amz-meta-tag には任意の値を含めることができる

このポリシーの Base64 エンコードされたバージョンを次に示します。

```
eyAiZXhwaXJhdGlvbiI6ICIyMDA3LTEyLTAxVDEyOjAwOjAwLjAwMFoiLAogICJjb25kaXR
pb25zIjogWwogICAgeyJidWNrZXQiOiAiam9obnNtaXRoIn0sCiAgICBbInN0YXJ0cy13aXRoIiw
gIiRrZXkiLCAidXNlci9lcmljLyJd
LAogICAgeyJhY2wiOiAicHVibGljLXJlYWQifSwKICAgIHsic3VjY2Vzc19hY3Rpb25fcm
VkaXJlY3QiOiAiaHR0cDovL2pvaG5zbWl0a
C5zMy5hbWF6b25hd3MuY29tL25ld19wb3N0Lmh0bWwifSwKICAgIFsiZXEiLCAiJENvbnRlbnQtVHl
wZSIsICJ0ZXh0L2h0bWwiXSwKI
CAgIHsieC1hbXotbWV0YS11dWlkIjogIjE0MzY1MTIzNjUxMjc0In0sCiAgICBbInN0YXJ0cy13aXRoIi
wgIiR4LWFtei1tZXRhLXRhZy
IsICIiXQogIF0KfQo=
```

認証情報を使用すると署名が作成されます。例えば、qA7FWXKq6VvU68lI9KdveT1cWgF= は、前述のポリシードキュメントの署名です。

このポリシーを使用して johnsmith.net バケットへの POST をサポートするフォームを次に示します。

```
<html>
  <head>
    ...
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    ...
  </head>
  <body>
  ...
  <form action="http://johnsmith.s3.amazonaws.com/" method="post" enctype="mul
tipart/form-data">
    Key to upload: <input type="input" name="key" value="user/eric/" /><br />
    <input type="hidden" name="acl" value="public-read" />
    <input type="hidden" name="success_action_redirect" value="http://johns
mith.s3.amazonaws.com/new_post.html" />
    <input type="hidden" name="Content-Type" value="text/html" />
    <input type="hidden" name="x-amz-meta-uuid" value="14365123651274" />
    Tags for File: <input type="input" name="x-amz-meta-tag" value="" /><br />

    <input type="hidden" name="AWSAccessKeyId" value="AKIAIOSFODNN7EXAMPLE" />

    <input type="hidden" name="Policy" value="POLICY" />
    <input type="hidden" name="Signature" value="SIGNATURE" />
    Entry: <textarea name="file" cols="60" rows="10">

Your blog post goes here.

    </textarea><br />
    <!-- The elements after this will be ignored -->
    <input type="submit" name="submit" value="Upload to Amazon S3" />
  </form>
  ...
</html>
```

リクエスト例

このリクエストでは、アップロードされたイメージが 117,108 バイト (イメージデータを除く) であることを前提としています。

```
POST / HTTP/1.1
Host: johnsmith.s3.amazonaws.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.10)
Gecko/20071115 Firefox/2.0.0.10
Accept: text/xml,application/xml,applica
tion/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Content-Type: multipart/form-data; boundary=178521717625888
Content-Length: 118635

-178521717625888
Content-Disposition: form-data; name="key"
```

```
ser/eric/NewEntry.html
--178521717625888
Content-Disposition: form-data; name="acl"

public-read
--178521717625888
Content-Disposition: form-data; name="success_action_redirect"

http://johnsmith.s3.amazonaws.com/new_post.html
--178521717625888
Content-Disposition: form-data; name="Content-Type"

text/html
--178521717625888
Content-Disposition: form-data; name="x-amz-meta-uuid"

14365123651274
--178521717625888
Content-Disposition: form-data; name="x-amz-meta-tag"

Interesting Post
--178521717625888
Content-Disposition: form-data; name="AWSAccessKeyId"

AKIAIOSFODNN7EXAMPLE
--178521717625888
Content-Disposition: form-data; name="Policy"
eyAiZXhwaXJhdGlvbiI6ICIyMDA3LTEyLTAxVDEyOjAwOjAwLjAwMFoiLAogICJjb25kaXRpb25zI
jogWwogICAgeyJidWNrZXQiOiAiam9obnNtaXRoIn0sCiAgICBbInN0YXJ0cy13aXRoIiwgIiRrZXkiL
CAidXNlci9lcmljLyJdLAogICAgeyJhY2wiOiAicHVibGljLXJlYWQifSwKICAgIH
sic3VjY2Vzc19hY3Rpb25fcmVkaXJlY3QiOiAiaHR0cDovL2pvaG5zbWl0aC5zMy5hb
WF6b25hd3MuY29tL25ld19wb3N0Lmh0bWwifSwKICAgIFsiZXEiLCAiJENvbnRlbnQtVHlwZSIs
ICJ0ZXh0L2h0bWwiXSwKICAgIHsieC1hbXotbWV0YS11dWlkIjogIjE0MzY1MTIzN
jUxMjc0In0sCiAgICBbInN0YXJ0cy13aXRoIiwgIiR4LWFtei1tZXRhLXRhZyIsICIiXQogIF0KfQo=

--178521717625888
Content-Disposition: form-data; name="Signature"

qA7FWXKq6VvU68lI9KdveT1cWgF=
--178521717625888
Content-Disposition: form-data; name="file"

...content goes here...
--178521717625888
Content-Disposition: form-data; name="submit"

Upload to Amazon S3
--178521717625888--
```

レスポンス例

```
HTTP/1.1 303 Redirect
x-amz-request-id: 1AEE782442F35865
x-amz-id-2: cxzFLJRatFHy+NGtaDFRR8YvI9BHmgLxjvJzNiGGICARZ/mVXHj7T+qQKhdpzHFh
Content-Type: application/xml
Date: Wed, 14 Nov 2007 21:21:33 GMT
Connection: close
```

```
Location: http://johnsmith.s3.amazonaws.com/new_post.html?bucket=johns
mith&key=user/eric/NewEntry.html&etag=40c3271af26b7f1672e41b8a274d28d4
Server: AmazonS3
```

Adobe Flash での POST

このセクションでは、Adobe Flash で POST を使用する方法について説明します。

Adobe Flash Player のセキュリティ

デフォルトでは、Adobe Flash Player のセキュリティモデルにより、Adobe Flash Player が、SWF ファイルを提供するドメイン外のサーバーにネットワーク接続することはできません。

このデフォルトの設定を上書きするには、公開されている crossdomain.xml ファイルを、POST アップロードを受け入れるバケットにアップロードする必要があります。crossdomain.xml ファイルの例を次に示します。

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM
"http://www.macromedia.com/xml/dtds/cross-domain-policy.dtd">
<cross-domain-policy>
<allow-access-from domain="*" secure="false" />
</cross-domain-policy>
```

Note

Adobe Flash セキュリティモデルの詳細については、Adobe のウェブサイトを参照してください。

crossdomain.xml ファイルをバケットに追加すると、Adobe Flash Player が、バケット内の crossdomain.xml ファイルに接続できるようになります。ただし、実際の Amazon S3 バケットへのアクセス権は付与されません。

Adobe Flash に関する考慮事項

Adobe Flash の FileReference API により、*Filename* フォームフィールドが POST リクエストに追加されます。FileReference API を使用して Amazon S3 にアップロードする Adobe Flash アプリケーションを構築する場合は、次の条件をポリシーに含めます。

```
['starts-with', '$Filename', '']
```

Adobe Flash Player のバージョンによっては、本文が空の HTTP レスポンスが適切に処理されないことがあります。本文が空でないレスポンスを返すように POST を設定するには、*success_action_status* を201に設定します。設定すると、XML ドキュメントとステータスコード201が返されます。XML ドキュメントのコンテンツについては、「POST Object」を参照してください。フォームフィールドについては、「HTML フォームフィールド (p. 72)」を参照してください。

Amazon S3 バケットの使用

Topics

Amazon S3 に格納されるすべてのオブジェクトは、バケットに保管されます。バケットは、Amazon S3 内に格納されるオブジェクトの名前空間を、最上位でパーティション化します。バケット内のオブジェクトにはどのような名前でも使用できますが、バケットの名前は Amazon S3 全体で一意のものにする必要があります。

バケットは、インターネットのドメイン名のようなものです。Amazon.com というドメイン名を所有するのが Amazon だけであるように、Amazon S3 内の各バケットは、1 人の個人または 1 つの組織のみが所有できます。Amazon S3 内に一意の名前を持つバケットをいったん作成した後は、バケット内のオブジェクトをどのように構成し命名してもかまいません。バケットは、Amazon S3 アカウントを保持している限り、いつまででも所有できます。

バケットとドメイン名が類似しているのは偶然ではありません。Amazon S3 のバケットと s3 のサブドメインの間は直接マッピングされています。amazonaws.com. Amazon S3 内に格納されたオブジェクトは、REST API を使用してドメイン *bucketname*.s3 の下のアドレスに解決できます。amazonaws.com. 例えば、オブジェクト `homepage.html` が Amazon S3 のバケット `mybucket` に格納されている場合、そのアドレスは `http://mybucket.s3.amazonaws.com/homepage.html` となります。詳細については、「バケットの仮想ホスティング (p. 63)」を参照してください。

REST を使用してバケット名が存在するかどうかを確認するには、`HEAD` を使用し、バケット名を指定して、`max-keys` を 0 に設定します。レスポンスが NoSuchBucket の場合は、そのバケット名は使用不可能であり、AccessDenied の場合はそのバケットが他の人に所有されており、Success の場合は自分がそのバケットを所有しているか、そのバケットへのアクセス許可を持っていることを示します。SOAP を使用してバケット名が存在するかどうかを確認するには、`ListBucket` を使用し、`MaxKeys` を 0 に設定します。

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

バケットを作成できる AWS リージョンのリストについては、『AWS General Reference』の「*Regions and Endpoints*」を参照してください。

バケットの制約と制限

各バケットは、それを作成した AWS アカウントによって所有されます。各 AWS アカウントは、同時に最大100個のバケットを所有できます。バケットの所有権は譲渡できませんが、バケットが空の場合は削除できます。バケットを削除するとそのバケット名は再度利用できるようになりますが、さまざまな理由により、削除した本人が使用できないことがあります。例えば、他のアカウントがその名前でバケットを作成する場合があります。このため、同じバケット名を使いたい場合は、バケットを削除しないでください。削除した名前が再度利用可能になるまでには、しばらく時間がかかることがあります。

1 つのバケット内に格納できるオブジェクトの数に制限はありません。また、使用するバケットの数によってパフォーマンスに差が出ることもありません。すべてのオブジェクトを 1 つのバケットに格納することができます。また、複数のバケットに分けて整理してもかまいません。

バケット内に別のバケットを作ることはできません。

Amazon S3 の高可用性技術は、GET、PUT、LIST、および DELETE オペレーションに重点を置いています。バケットオペレーションは中央集中型のグローバルリソーススペースに対して作用するため、お使いのアプリケーションの高可用性コードパスでバケットの作成や削除の呼び出しを行うのは適切ではありません。バケットの作成や削除は、個別の初期化や、頻繁に実行しないセットアップルーチンで実行する方が適しています。

Note

アプリケーションが自動的にバケットを作成する場合は、命名の衝突が起きないようなバケット命名スキームを使用してください。特定のバケット名が既に使用されている場合は、異なるバケット名をアプリケーションロジックが選択するようにしてください。

Note

Amazon DevPay を使用している場合、各顧客は、使用する Amazon DevPay 製品 1 つにつき最大 100 個のバケットを持つことができます。詳細については、「Amazon S3 での Amazon DevPay の使用 (p. 453)」を参照してください。

バケット命名規則

米国スタンダード以外のすべてのリージョンで、バケット名は以下の規則にしたがう必要があります。これにより、DNS 準拠のバケット名となります。

- バケット名は、3 ~ 63 文字以内にする必要があります。
- バケット名は、1 つのラベルか、または複数のラベルをピリオド (.) でつなげて構成します。各ラベルには次の規則が適用されます。
  - 先頭の文字には小文字の英文字または数字を使う
  - 末尾の文字には小文字の英文字または数字を使う

  - 小文字の英文字、数字、およびハイフン (-) を含めることができる
- バケット名は、IP アドレスの形式 (192.168.5.4 など) にはできない

以下は有効なバケット名の例です。

- myawsbucket
- my.aws.bucket
- myawsbucket.1

以下は無効なバケット名の例です。

無効なバケット名	コメント
.myawsbucket	バケット名の先頭にはピリオド (.) を使用できません。
myawsbucket.	バケット名の末尾にはピリオド (.) を使用できません。
my..examplebucket	ラベルの間にはピリオドを1つだけ使用できます。

仮想ホスト形式のリクエスト、たとえば SSL 経由の http://mybucket.s3.amazonaws.com を使用してバケットにアクセスする場合、バケット名にピリオド (.) を含めることはできません。

米国スタンダードリージョンでのバケット命名規則はこれらと類似していますが、制約が緩やかです。

- バケット名は最長で 255 文字です。
- バケット名には、大文字と小文字の英文字、数字、ピリオド (.)、ハイフン (-)、アンダースコア (_) を自由に組み合わせて使用できます。

米国スタンダードリージョンの命名規則にしたがった場合、DNS に準拠しないバケット名となることがあります。例えば、MyAWSBucket は大文字を含みますが、バケット名として有効です。このバケットに、仮想ホスト形式のリクエストである http://MyAWSBucket.s3.amazonaws.com/yourobject を使用してアクセスしようとすると、URL はバケット MyAWSBucket でなく、バケット myawsbucket に解決されます。このため、Amazon S3 はバケットについて Not Found エラーを返します。このような問題を防ぐため、ベストプラクティスとして、バケットを作成するリージョンに関係なく常に DNS 準拠のバケット名を使用することを推奨します。バケットへの仮想ホスト形式のアクセスの詳細については、「バケットの仮想ホスティング (p. 63)」を参照してください。

バケット設定オプション

バケットを作成する際、<CreateBucketConfiguration> XML ボディを PUT Bucket リクエストにアタッチすることで追加の Amazon S3 機能を利用できます。現時点では、ロケーションによる制約を選択できます。詳細については、「バケットとリージョン (p. 90)」を参照してください。

<CreateBucketConfiguration> で作成したバケットには、さらに制約が課せられます。

- <CreateBucketConfiguration> で作成したバケットには、パス形式リクエストを使ってリクエストを行うことはできません。仮想ホスト形式のリクエストを実行する必要があります。詳細については、「バケットの仮想ホスティング (p. 63)」を参照してください。
- 追加のバケット命名規則に従う必要があります。詳細については、「バケットの制約と制限 (p. 88)」を参照してください。

バケットとリージョン

作成したバケットを Amazon S3 でどこに格納するか、地理的なリージョンを選択できます。レイテンシーを最適化し、コストを最小限に抑えて規制用件に対応できるリージョンを選ぶとよいでしょう。例えば、ヨーロッパにお住まいの場合は、欧州 (アイルランド) リージョンにバケットを作成するとよいでしょう。

Important

SOAP API は、地理的な制約をサポートしていません。SOAP のサポートは HTTP 経由では廃止されましたが、HTTPS 経由では引き続き利用可能です。SOAP 用に Amazon S3 の新機能をサポートする予定はありません。

明示的に別のリージョンに移動する場合を除き、特定のリージョンに保管されたオブジェクトは、そのリージョンから移動されることはありません。例えば、欧州 (アイルランド) リージョンに保存されたオブジェクトは、ずっとそのリージョンに置かれたままです。

Amazon S3 は、次のリージョンをサポートしています。

- US Standard Uses Amazon S3 servers in the United States

  Provides eventual consistency for all requests. This region automatically routes requests to facilities in Northern Virginia or the Pacific Northwest using network maps.
- US West (Oregon) Region Uses Amazon S3 servers in Oregon

  Provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.
- US West (Northern California) Region Uses Amazon S3 servers in Northern California

  Provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.
- EU (Ireland) Region Uses Amazon S3 servers in Ireland

  Provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.
- Asia Pacific (Singapore) Region Uses Amazon S3 servers in Singapore

  Provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.
- Asia Pacific (Sydney) Region Uses Amazon S3 servers in Sydney

  Provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.
- Asia Pacific (Tokyo) Region Uses Amazon S3 servers in Tokyo

  Provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.
- South America (Sao Paulo) Region Uses Amazon S3 servers in Sao Paulo

  Provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.

リージョンはバケットレベルで設定されます。リージョンは *LocationConstraint* バケットパラメータを使って指定します。リージョンを指定しない場合、Amazon S3 はバケットを 米国東部 (バージニア北部) リージョン リージョンのサーバーに保存します。

Important

バケットの作成にリージョン固有のエンドポイントを使用する場合は、必ず *LocationConstraint* バケットパラメータを同じリージョンに設定する必要があります。リージョン固有のエンドポイントの詳細については、「Request Endpoints (p. 13)」を参照してください。

バケットのリージョンの指定方法

次の手順にしたがって、バケットのリージョンを指定します。

バケットのリージョンの指定

1	バケット作成リクエストで、*LocationContraint* パラメータを特定のリージョンに設定します。*CreateBucketConfiguration.LocationConstraint=us-west-1*
2	北カリフォルニアリージョンでバケットを作成する場合、オプションで、バケット作成リクエスト内でリージョン固有のエンドポイントを使用できます。例えば、ステップ 1 で指定したリージョンに対応させるには、エンドポイント `https://s3-us-west-1.amazonaws.com` (または `http://s3-us-west-1.amazonaws.com`) を使用します。 リージョン固有のエンドポイントを使用することで、米国スタンダードから北カリフォルニアリージョンにリクエストをリダイレクトする際に生じるレイテンシーを排除できます。詳細については、「Redirection (p. 91)」を参照してください。  Important 北カリフォルニアリージョンのバケットを作成するためのリクエストでリージョン固有のエンドポイントを使用する場合でも、*LocationConstraint* の値は同じリージョンに設定する必要があります。

バケットアクセス

`CreateBucketConfiguration` を使用して作成された Amazon S3 のバケットとオブジェクトにアクセスする際は、すべてのリージョンで仮想ホスト形式のリクエストを使用できます。以下に例を示します。

```
http://yourbucket.s3.amazonaws.com/yourobject
```

パス形式のリクエストを使用するには、バケットが米国クラシックリージョン内にあるか、リクエスト内のエンドポイントと同じリージョンにある必要があります。次に例を示します。

```
http://s3.amazonaws.com/yourbucket/yourobject
```

リダイレクト

Amazon は一時的なリダイレクトと永続的なリダイレクトの 2 種類をサポートしています。永続的なリダイレクトの詳細については、「REST API を使用したリクエストの実行 (p. 49)」を参照してください。

一時的なリダイレクトでは、リクエストされたバケットについての DNS 情報を持たないユーザーを自動的にリダイレクトします。DNS の変更をインターネット上に伝達するには時間がかかるため、このような処理が行われます。例えば、あるユーザーが口ケーションの制約のあるバケットを作成し、その中にすぐにオブジェクトを格納した場合、そのバケットに関する情報がまだインターネット上に伝達さ

れていない場合があります。バケットは s3.amazonaws.com のサブドメインであるため、Amazon S3 により正しい Amazon S3 ロケーションにリダイレクトされます。

この (一時的な) リダイレクトのレイテンシーは、バケット作成リクエスト内でリージョン固有のエンドポイントを使用することで排除できます。*LocationConstraint* バケットパラメータは、バケットが置かれるリージョンを指定します。リージョン固有のエンドポイントの使用はオプションです。これは、米国西部でのみ実行できます。詳細については、「バケットのリージョンの指定方法 (p. 91)」を参照してください。

他のリージョンへのバケットのコンテンツの移動

次の手順にしたがって、バケットを特定のリージョンから別のリージョンに移動します。

バケットを別のリージョンに移動するには

1	データの移動先のリージョンに新しい Amazon S3 バケットを作成します。
2	copy オペレーションを使用して、各オブジェクトを移動元のリージョンから移動先のリージョンへ移動します。 この移動には、帯域幅料金がかかります。詳細については、「COPY Object」を参照してください。

バケットのウェブサイト設定の管理

Topics

- AWS Management Console によるウェブサイトの管理 (p. 92)
- AWS SDK for Java によるウェブサイトの管理 (p. 92)
- AWS SDK for .NET によるウェブサイトの管理 (p. 95)
- AWS SDK for PHP によるウェブサイトの管理 (p. 99)
- REST API によるウェブサイトの管理 (p. 101)

静的ウェブサイトを Amazon S3 でホスティングするには、バケットをウェブサイトホスティング用に設定します。詳細については、「Amazon S3 での静的ウェブサイトのホスティング (p. 411)」を参照してください。バケットのウェブサイト設定を管理する方法はいくつかあります。AWS Management Console を使用して設定を管理できるようになるので、コードを書く必要がなくなります。AWS SDK を使用して、ウェブサイト設定をプログラムによって作成、更新、削除できます。SDK には、Amazon S3 REST API のラッパークラスがあり、アプリケーションで必要な場合は、アプリケーションから直接 REST API リクエストを送信できます。

AWS Management Console によるウェブサイトの管理

詳細については、「バケットをウェブサイトホスティング用に設定 (p. 414)」を参照してください。

AWS SDK for Java によるウェブサイトの管理

以下のタスクは、Java クラスを使用して、バケットのウェブサイト設定を管理する手順を示しています。Amazon S3 ウェブサイト機能の詳細については、「Amazon S3 での静的ウェブサイトのホスティング (p. 411)」を参照してください。

ウェブサイト設定の管理

1	AWS 認証情報を指定して、AmazonS3 クラスのインスタンスを作成します。
2	ウェブサイト設定をバケットに追加するには、AmazonS3.setBucketWebsiteConfiguration メソッドを実行します。バケット名と、インデックスドキュメント名およびエラードキュメント名を含むウェブサイト設定情報を指定する必要があります。インデックスドキュメントは指定する必要がありますが、エラードキュメントは省略可能です。ウェブサイト設定情報を指定するには、BucketWebsiteConfiguration オブジェクトを作成します。 ウェブサイト設定を取得するには、バケット名を指定して、AmazonS3.getBucketWebsiteConfiguration メソッドを実行します。 バケットのウェブサイト設定を削除するには、バケット名を指定して、AmazonS3.deleteBucketWebsiteConfiguration メソッドを実行します。ウェブサイト設定を削除すると、バケットはウェブサイトエンドポイントを介して使用できなくなります。詳細については、「ウェブサイトエンドポイント (p. 412)」を参照してください。

以下の Java コード例は、前述のタスクの例です。

```
AWSCredentials myCredentials = new BasicAWSCredentials(
                                  myAccessKeyID, mySecretKey);
AmazonS3 s3client = new AmazonS3Client(myCredentials);
// Add website configuration.
s3Client.setBucketWebsiteConfiguration(bucketName,
      new BucketWebsiteConfiguration(indexDoc , errorDoc));

// Get website configuration.
BucketWebsiteConfiguration bucketWebsiteConfiguration =
         s3Client.getBucketWebsiteConfiguration(bucketName);

// Delete website configuration.
s3Client.deleteBucketWebsiteConfiguration(bucketName);
```

Example

次の Java コード例では、指定したバケットにウェブサイト設定を追加し、それを取得して、削除します。作業サンプルを作成およびテストする方法については、「Java コード例のテスト (p. 477)」を参照してください。

```
import java.io.IOException;
import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.BucketWebsiteConfiguration;


public class S3Sample {
 private static String bucketName = "mvphp1";
 private static String indexDoc = "index.html";
 private static String errorDoc = "error.html";

 public static void main(String[] args) throws IOException {
        AmazonS3 s3Client = new AmazonS3Client(new PropertiesCredentials(
           S3Sample.class.getResourceAsStream("AwsCredentials.properties")));

        try {
         // Get existing website configuration, if any.
            getWebsiteConfig(s3Client);

      // Set new website configuration.
      s3Client.setBucketWebsiteConfiguration(bucketName,
         new BucketWebsiteConfiguration(indexDoc, errorDoc));

      // Verify (Get website configuration again).
            getWebsiteConfig(s3Client);

            // Delete
            s3Client.deleteBucketWebsiteConfiguration(bucketName);

         // Verify (Get website configuration again)
              getWebsiteConfig(s3Client);



        } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException, which" +
              " means your request made it " +
                    "to Amazon S3, but was rejected with an error response" +
                    " for some reason.");
            System.out.println("Error Message:    " + ase.getMessage());
            System.out.println("HTTP Status Code: " + ase.getStatusCode());
            System.out.println("AWS Error Code:   " + ase.getErrorCode());
            System.out.println("Error Type:       " + ase.getErrorType());
            System.out.println("Request ID:       " + ase.getRequestId());
        } catch (AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException, which means"+

              " the client encountered " +
                    "a serious internal problem while trying to " +
```

```
                    "communicate with Amazon S3, " +
                    "such as not being able to access the network.");
            System.out.println("Error Message: " + ace.getMessage());
        }
    }

 private static BucketWebsiteConfiguration getWebsiteConfig(
                                                AmazonS3 s3Client) {
  System.out.println("Get website config");

  // 1. Get website config.
  BucketWebsiteConfiguration bucketWebsiteConfiguration =
     s3Client.getBucketWebsiteConfiguration(bucketName);
  if (bucketWebsiteConfiguration == null)
  {
   System.out.println("No website config.");
  }
  else
  {
       System.out.println("Index doc:" +
         bucketWebsiteConfiguration.getIndexDocumentSuffix());
       System.out.println("Error doc:" +
         bucketWebsiteConfiguration.getErrorDocument());
  }
  return bucketWebsiteConfiguration;
 }
}
```

AWS SDK for .NET によるウェブサイトの管理

以下のタスクは、.NET クラスを使用して、バケットのウェブサイト設定を管理する手順を示しています。Amazon S3 ウェブサイト機能の詳細については、「Amazon S3 での静的ウェブサイトのホスティング (p. 411)」を参照してください。

バケットのウェブサイト設定の管理

1	AWS 認証情報を指定して、AmazonS3 クラスのインスタンスを作成します。
2	ウェブサイト設定をバケットに追加するには、AmazonS3.PutBucketWebsite メソッドを実行します。バケット名と、インデックスドキュメント名およびエラードキュメント名を含むウェブサイト設定情報を指定する必要があります。インデックスドキュメントは指定する必要がありますが、エラードキュメントは省略可能です。この情報を指定するには、PutBucketWebsiteRequest オブジェクトを作成します。 ウェブサイト設定を取得するには、バケット名を指定して、AmazonS3.GetBucketWebsite メソッドを実行します。 バケットのウェブサイト設定を削除するには、バケット名を指定して、AmazonS3.DeleteBucketWebsite メソッドを実行します。ウェブサイト設定を削除すると、バケットはウェブサイトエンドポイントを介して使用できなくなります。詳細については、「ウェブサイトエンドポイント (p. 412)」を参照してください。

以下の C# コード例は、前述のタスクの例です。

```
AmazonS3 client;
client = Amazon.AWSClientFactory.CreateAmazonS3Client(
```

```
                    accessKeyID, secretAccessKeyID);
// Add website configuration.
PutBucketWebsiteRequest putRequest =
   new PutBucketWebsiteRequest()
         .WithBucketName(bucketName)
         .WithWebsiteConfiguration(new WebsiteConfiguration()
         .WithIndexDocumentSuffix(indexDocumentSuffix)
         .WithErrorDocument(errorDocument));

client.PutBucketWebsite(putRequest);

// Get bucket website configuration.
GetBucketWebsiteRequest getRequest =
   new GetBucketWebsiteRequest()
     .WithBucketName(bucketName);

GetBucketWebsiteResponse getResponse =
  client.GetBucketWebsite(getRequest);
// Print configuration data.
Console.WriteLine("Index document: {0}", getResponse.WebsiteConfiguration.Index
DocumentSuffix);
Console.WriteLine("Error document: {0}", getResponse.WebsiteConfiguration.Error
Document);

// Delete website configuration.
DeleteBucketWebsiteRequest deleteRequest =
    new DeleteBucketWebsiteRequest()
         .WithBucketName(bucketName);

client.DeleteBucketWebsite(deleteRequest);
```

Example

次の C# コード例では、指定したバケットにウェブサイト設定を追加します。この設定で、インデックスドキュメント名とエラードキュメント名の両方を指定します。作業サンプルを作成およびテストする方法については、「.NET コード例のテスト (p. 478)」を参照してください。

```
using System;
using System.Configuration;
using System.Collections.Specialized;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples.addwebsiteconfiguration
{
    class S3Sample
    {
        static string bucketName          = "*** Provide existing bucket name
***";
        static string indexDocumentSuffix = "*** Provide index document name
***";
        static string errorDocument       = "*** Provide error document name
***";
        static AmazonS3 client;

        public static void Main(string[] args)
        {
            if (checkRequiredFields())
            {
                NameValueCollection appConfig =
                    ConfigurationManager.AppSettings;

                string accessKeyID = appConfig["AWSAccessKey"];
                string secretAccessKeyID = appConfig["AWSSecretKey"];

                using (client = Amazon.AWSClientFactory.CreateAmazonS3Client(
                    accessKeyID, secretAccessKeyID))
                {
                   AddWebsiteConfig(bucketName, indexDocumentSuffix, errorDoc
ument);
                }
            }

            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }

        static void AddWebsiteConfig(string bucketName,
                                     string indexDocumentSuffix,
                                     string errorDocument)
        {
            try
            {
                PutBucketWebsiteRequest putRequest =
                    new PutBucketWebsiteRequest()
                    .WithBucketName(bucketName)
                    .WithWebsiteConfiguration(new WebsiteConfiguration()
                        .WithIndexDocumentSuffix(indexDocumentSuffix)
                        .WithErrorDocument(errorDocument));
```

```
                client.PutBucketWebsite(putRequest);
            }
            catch (AmazonS3Exception amazonS3Exception)
            {
                if (amazonS3Exception.ErrorCode != null &&
                    (amazonS3Exception.ErrorCode.Equals("InvalidAccessKeyId")
                    || amazonS3Exception.ErrorCode.Equals("InvalidSecurity")))

                {
                    Console.WriteLine("Check the provided AWS Credentials.");
                    Console.WriteLine(
                        "Sign up for service at http://aws.amazon.com/s3");
                }
                else
                {
                    Console.WriteLine(
                       "Error:{0}, occurred when adding website configuration.
 Message:'{1}",
                      amazonS3Exception.ErrorCode, amazonS3Exception.Message);

                }
            }
        }

        static bool checkRequiredFields()
        {
            NameValueCollection appConfig = ConfigurationManager.AppSettings;

            if (string.IsNullOrEmpty(appConfig["AWSAccessKey"]))
            {
                Console.WriteLine(
                    "AWSAccessKey was not set in the App.config file.");
                return false;
            }
            if (string.IsNullOrEmpty(appConfig["AWSSecretKey"]))
            {
                Console.WriteLine(
                    "AWSSecretKey was not set in the App.config file.");
                return false;
            }
            if (string.IsNullOrEmpty(bucketName))
            {
                Console.WriteLine("The variable bucketName is not set.");
                return false;
            }


            return true;
        }

    }
}
```

AWS SDK for PHP によるウェブサイトの管理

このトピックでは、AWS SDK for PHP のクラスを使用して、ウェブサイトホスティング用に Amazon S3 バケットを設定、管理する手順を示します。Amazon S3 ウェブサイト機能の詳細については、「Amazon S3 での静的ウェブサイトのホスティング (p. 411)」を参照してください。

Note

このトピックでは、既に AWS SDK for PHP の使用と PHP サンプルの実行 (p. 479) の説明が実行されていて、AWS SDK for PHP が正しくインストールされていることを前提としています。

このトピックでは、PHP SDK のクラスを使用して、ウェブサイトホスティング用に Amazon S3 バケットを設定、管理する手順を示します。

ウェブサイトホスティング用のバケットの設定

1	自分の AWS 認証情報または IAM ユーザー認証情報と共に Aws\S3\S3Client クラスの factory() メソッドを使用して、Amazon S3 クライアントのインスタンスを作成します。
2	バケットをウェブサイトとして設定するには、Aws\S3\S3Client::putBucketWebsite() メソッドを実行します。バケット名と、インデックスドキュメント名およびエラードキュメント名を含むウェブサイト設定情報を指定する必要があります。これらのドキュメント名を指定しないと、このメソッドによって、`index.html` および `error.html` のデフォルト名がウェブサイト設定に追加されます。これらのドキュメントがバケットに存在していることを確認する必要があります。
3	バケットの既存のウェブサイト設定を取得するには、Aws\S3\S3Client::getBucketWebsite() メソッドを実行します。
4	バケットからウェブサイト設定を削除するには、パラメータとしてバケット名を渡して Aws\S3\S3Client::deleteBucketWebsite() メソッドを実行します。 ウェブサイト設定を削除すると、バケットはウェブサイトエンドポイントからアクセスできなくなります。

以下の PHP コード例は、前述のタスクを実装したものです。

```
use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';

// 1. Instantiate the client.
$s3 = S3Client::factory(array(
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));

// 2. Add website configuration.
$result = $s3->putBucketWebsite(array(
    'Bucket'        => $bucket,
    'IndexDocument' => array('Suffix' => 'index.html'),
    'ErrorDocument' => array('Key' => 'error.html'),
));

// 3. Retrieve website configuration.
$result = $s3->getBucketWebsite(array(
    'Bucket' => $bucket,
```

```
));
echo $result->getPath('IndexDocument/Suffix');

// 4.) Delete website configuration.
$result = $s3->deleteBucketWebsite(array(
    'Bucket' => $bucket,
));
```

Example （ウェブサイトホスティング用のバケット Amazon S3 の設定）

次の PHP コード例では、まず指定したバケットにウェブサイト設定を追加します。create_website_config メソッドで、インデックスドキュメント名とエラードキュメント名を明示的に指定します。さらに、例では、ウェブサイト設定を取得し、レスポンスを表示します。Amazon S3 ウェブサイト機能の詳細については、「Amazon S3 での静的ウェブサイトのホスティング (p. 411)」を参照してください。

作業サンプルを作成およびテストする方法については、「AWS SDK for PHP の使用と PHP サンプルの実行 (p. 479)」を参照してください。

```
<?php

// Include the AWS SDK using the Composer autoloader.
require 'vendor/autoload.php';

use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';

// Instantiate the client.
$s3 = S3Client::factory(array(
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));

// 1.) Add website configuration.
$result = $s3->putBucketWebsite(array(
    'Bucket'        => $bucket,
    'IndexDocument' => array('Suffix' => 'index.html'),
    'ErrorDocument' => array('Key' => 'error.html'),
));

// 2.) Retrieve website configuration.
$result = $s3->getBucketWebsite(array(
    'Bucket' => $bucket,
));
echo $result->getPath('IndexDocument/Suffix');

// 3.) Delete website configuration.
$result = $s3->deleteBucketWebsite(array(
    'Bucket' => $bucket,
));
```

関連リソース

- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client クラス」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::deleteBucketWebsite() メソッド」

- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::factory() メソッド」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::getBucketWebsite() メソッド」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::putBucketWebsite() メソッド」
- 「AWS SDK for PHP – Amazon S3」
- 「AWS SDK for PHP」のドキュメント

REST API によるウェブサイトの管理

Topics

AWS Management Console または AWS SDK を使用して、バケットをウェブサイトとして設定できます。ただし、アプリケーションで必要な場合は、REST リクエストを直接送信できます。詳細については、『Amazon Simple Storage Service API リファレンス』の以下のセクションを参照してください。

- PUT Bucket website
- GET Bucket website
- DELETE Bucket website

リクエスタ支払いバケット

Topics

一般に、Amazon S3 のバケットのストレージおよびデータ転送にかかるコストはすべて、そのバケット所有者が負担します。ただしバケット所有者は、バケットをリクエスタ支払いバケットとして設定することができます。リクエスタ支払いバケットの場合、リクエストおよびバケットからのデータのダウンロードにかかるコストは、所有者でなくリクエストを実行したリクエスタが支払います。データの保管にかかるコストは常にバケット所有者が支払います。

通常は、データを共有したいが、他者がデータにアクセスする際に発生する費用を負担したくない場合に、リクエスタ支払いバケットを設定します。例えば、郵便番号リスト、参照データ、地理空間情報、ウェブクロールデータといった、大規模なデータセットを利用できるようにする際にリクエスタ支払いバケットを設定します。

Important

バケットでリクエスタ支払いを有効化すると、そのバケットには匿名アクセスができなくなります。

リクエスタ支払いバケットに関するリクエストはすべて認証する必要があります。リクエストの認証により、&S3 はリクエスタを特定し、リクエスタ支払いバケットの使用に対して課金できるようになります。

バケットをリクエスタ支払いバケットに設定した場合、リクエスタは、リクエストおよびダウンロードするデータに対して課金されることについて理解していることを示すため、POST と GET リクエストの場合はヘッダーに、REST リクエストの場合はパラメータとして、リクエストに x-amz-request-payer を含める必要があります。

リクエスタ支払いバケットは、以下をサポートしていません。

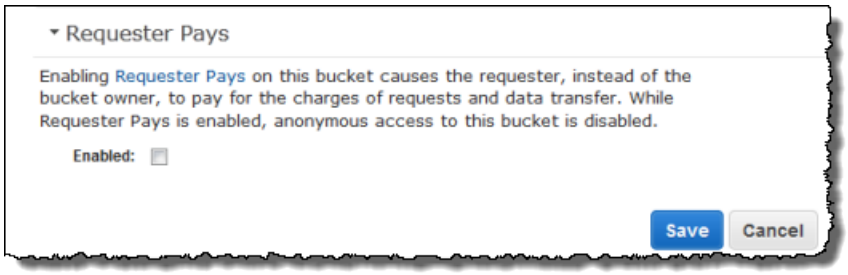
- 匿名リクエスト
- BitTorrent
- SOAP リクエスト
- リクエスタ支払いバケットをエンドユーザーのログのターゲットバケットとして使用することはできず、またその逆もできません。ただし、ターゲットバケットがリクエスタ支払いバケットでない場合は、リクエスタ支払いバケットでエンドユーザーのログを有効にできます。

&S3 コンソールを使用したリクエスタ支払いの設定

&S3 コンソールを使用して、バケットをリクエスタ支払い用に設定できます。

バケットをリクエスタ支払い用に設定するには

1. AWS マネジメントコンソールにサインインして Amazon S3 コンソール（ https://console.aws.amazon.com/s3/ ）を開きます。
2. [Buckets] リストで、バケット名の左側の詳細アイコンをクリックし、[Properties] をクリックしてバケットのプロパティを表示します。
3. [Properties] ペインの [Requester Pays] をクリックします。
4. [Enabled] チェックボックスをオンにします。



REST API を使用したリクエスタ支払いの設定

Topics

requestPayment バケットの設定

バケット所有者のみが、バケットの *RequestPaymentConfiguration.payer* 設定値を `BucketOwner`（デフォルト）または `Requester` に設定できます。*requestPayment* リソースの設定はオプションです。これを実行しない場合は、デフォルトで、非リクエスタ支払いバケットになります。

リクエスタ支払いバケットを通常のバケットに戻すには、値 `BucketOwner` を使用します。通常、Amazon S3 バケットにデータをアップロードする際に `BucketOwner` を使用し、その後、バケット内のオブジェクトを公開する前に値を `Requester` に設定します。

requestPayment を設定するには

- PUT リクエストを使用して、指定したバケットの *Payer* の値を Requester に設定します。

```
PUT ?requestPayment HTTP/1.1
Host: [BucketName].s3.amazonaws.com
Content-Length: 173
Date: Wed, 01 Mar 2009 12:00:00 GMT
Authorization: AWS [Signature]

<RequestPaymentConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
<Payer>Requester</Payer>
</RequestPaymentConfiguration>
```

リクエストが成功すると、Amazon S3 は、以下のようなレスポンスを返します。

```
HTTP/1.1 200 OK
x-amz-id-2: [id]
x-amz-request-id: [request_id]
Date: Wed, 01 Mar 2009 12:00:00 GMT
Content-Length: 0
Connection: close
Server: AmazonS3
x-amz-request-charged:requester
```

リクエスタ支払いはバケットレベルでのみ設定できます。バケット内の特定のオブジェクトに設定することはできません。

バケットは、いつでも自由に BucketOwner または Requester に設定できます。ただし、設定が有効になるまでには数分間かかることがありますので注意してください。

Note

署名付き URL を提供するバケット所有者は、特に URL の有効期限がとても長い場合は、バケットをリクエスタ支払いに設定する前によく検討してください。バケット所有者は、バケット所有者の認証情報を使用する署名付き URL をリクエスタが使用するたびに課金されます。

requestPayment 設定の取得

バケットに設定された *Payer* 値を取得するには、リソース requestPayment をリクエストします。

requestPayment リソースを返すには

- 以下のリクエストに示すように、GET リクエストを使用して *requestPayment* リソースを取得します。

```
GET ?requestPayment HTTP/1.1
Host: [BucketName].s3.amazonaws.com
Date: Wed, 01 Mar 2009 12:00:00 GMT
Authorization: AWS [Signature]
```

リクエストが成功すると、Amazon S3 は、以下のようなレスポンスを返します。

```
HTTP/1.1 200 OK
x-amz-id-2: [id]
x-amz-request-id: [request_id]
Date: Wed, 01 Mar 2009 12:00:00 GMT
Content-Type: [type]
Content-Length: [length]
Connection: close
Server: AmazonS3

<?xml version="1.0" encoding="UTF-8"?>
<RequestPaymentConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
<Payer>Requester</Payer>
</RequestPaymentConfiguration>
```

このレスポンスでは、*payer* 値が Requester に設定されていることが示されています。

リクエスタ支払いバケット内のオブジェクトのダウンロード

リクエスタ支払いバケットからデータをダウンロードするとリクエスタに対して課金されるため、リクエストには、リクエスタが課金について認識していることを示すパラメータ *x-amz-request-payer* を含める必要があります。リクエスタ支払いバケット内のオブジェクトにアクセスするには、リクエストに以下のいずれかを含める必要があります。

- GET および POST リクエストの場合は、ヘッダーに *x-amz-request-payer : requester* を含めます
- 署名付き URL の場合は、リクエストに *x-amz-request-payer=requester* を含めます

リクエストが成功してリクエスタが課金されると、レスポンスには x-amz-request-charged:requester ヘッダーが含まれます。リクエスト内に *x-amz-request-payer* がない場合、Amazon S3 は 403 エラーを返し、リクエストに対してバケット所有者に課金します。

Note

バケット所有者は、リクエストに *x-amz-request-payer* を含める必要はありません。

署名の計算に *x-amz-request-payer* とその値を含めてください。詳細については、「Constructing the CanonicalizedAmzHeaders Element (p. 55)」を参照してください。

リクエスタ支払いバケットからオブジェクトをダウンロードするには

- リクエスタ支払いバケットからオブジェクトをダウンロードするには、以下のリクエストに示すように、GET リクエストを使用します。

```
GET / [destinationObject] HTTP/1.1
Host: [BucketName].s3.amazonaws.com
x-amz-request-payer : requester
Date: Wed, 01 Mar 2009 12:00:00 GMT
Authorization: AWS [Signature]
```

GET リクエストが成功してリクエスタが課金されると、レスポンスには x-amz-request-charged:requester が含まれます。

Amazon S3 は、リクエスタ支払いバケットからオブジェクトを取得しようとするリクエストに対して、`Access Denied` エラーを返すことがあります。詳細については、「Error Responses」を参照してください。

DevPay とリクエスタ支払い

Amazon DevPay を使用して、リクエスタ支払いバケットに格納されたコンテンツを販売できます。詳細については、「Using &S3 Requester Pays with DevPay」の「Using Amazon S3 Requester Pays with DevPay」を参照してください。

課金の詳細

成功したリクエスタ支払いリクエストに対する課金は非常にシンプルです。リクエスタは、転送されたデータとリクエストに対して支払い、バケット所有者はデータのストレージに対して支払います。ただし、次の場合はリクエストに対してバケット所有者が課金されます。

- リクエスタが、パラメータ *`x-amz-request-payer`* をヘッダー (GET または POST の場合) またはパラメータとして (REST の場合) リクエストに含めなかった場合 (HTTP コード 403)
- リクエストの認証に失敗した場合 (HTTP コード 403)
- リクエストが匿名の場合 (HTTP コード 403)
- リクエストが SOAP リクエストの場合

バケットとアクセスコントロール

各バケットには、アクセスコントロールポリシーが関連付けられています。このポリシーは、バケット内のオブジェクトの作成、削除、および列挙に適用されます。詳細については、「アクセスコントロール (p. 294)」を参照してください。

バケットの請求とレポート

オブジェクトの格納とネットワークデータの転送にかかる料金は、そのオブジェクトが格納されているバケット所有者に請求されます。ただし、バケットがリクエスタ支払いバケットに設定されている場合はこの限りではありません。

AWS の開発者ポータルで提供されているレポートツールは、バケットごとに整理した Amazon S3 の使用状況レポートを作成します。詳細については、「Amazon S3 料金表」のページを参照してください。

コスト割り当てタグ

コスト割り当てタグを使用して S3 バケットにラベルを付けることができるため、プロジェクトやその他の条件に照らして、それらのコストを簡単に追跡できます。

タグを使用して、ユーザー独自のコスト構造を反映するように AWS 請求を整理します。そのためには、AWS アカウントにサインアップして、タグキー値が含まれた AWS アカウントの請求書を取得する必要があります。結合したリソースのコストを確認するには、同じタグキー値を持つリソースに従って請求情報を整理します。例えば、複数のリソースに特定のアプリケーション名のタグを付け、請求情報を整理することで、複数のサービスを利用しているアプリケーションの合計コストを確認することができます。詳細については、About AWS Account Billing の *Cost Allocation and Tagging* を参照してください。

コスト割り当てタグは名前と値のペアで、ユーザーが定義し、S3 バケットに関連付けます。S3 バケットに関連付けられているコストの追跡が簡単になるように、一貫した一連のタグキーを使用することをお勧めします。

各 S3 バケットには、そのバケットに割り当てられているすべてのタグを格納するタグセットがあります。タグセットには、最大 10 個のタグを格納でき、空にすることもできます。

特定のバケットの既存のタグと同じキーを持つタグを追加した場合、古い値は新しい値によって上書きされます。

AWS は、タグに意味を適用しません。タグは文字列として厳密に解釈されます。AWS はバケットにタグを自動的に設定しません。

Amazon S3 コンソール、CLI、または S3 API を使用して、タグを追加、リスト、編集、または削除できます。コンソールでタグを作成する詳細については、『*Amazon Simple Storage Service Console User Guide*』の「Managing Cost Allocation Tagging」を参照してください。

コスト割り当てタグの特徴は次のとおりです。

- タグキーは、必須のタグ名です。文字列値は、1 ~ 128 文字の Unicode 文字です。「aws:」をプレフィックスとして使用することはできません。文字列には、一連の Unicode 文字、数字、空白、「_」、「.」、「/」、「=」、「+」、「-」（Java 正規表現: "^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$"）のみを含めることができます。

- タグ値は、必須のタグの文字列値です。文字列値は、1 ~ 256 文字の Unicode 文字です。「aws:」をプレフィックスとして使用することはできません。文字列には、一連の Unicode 文字、数字、空白、「_」、「.」、「/」、「=」、「+」、「-」（Java 正規表現: "^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$"）のみを含めることができます。

  値はタグセット内で一意である必要はなく、null を指定できます。例えば、project/Trinity と cost-center/Trinity のタグセット内に 1 つのキーと値のペアを使用できます。

バケット設定エラー

バケット設定リクエストに対して Amazon S3 が返す可能性のあるエラーのリストを次に示します。

- MalformedXML
- MissingRequestBodyError

Amazon S3 オブジェクトの使用

Topics

Amazon S3 はできる限り多くのオブジェクトを格納するように設計された、キーと値のためのシンプルなストアです。オブジェクトは 1 つ以上のバケットに格納します。オブジェクトの構成要素を次に示します。

- キー – オブジェクトに割り当てる名前です。オブジェクトを取得するには、オブジェクトキーを使用します。
  詳細については、「オブジェクトキーとメタデータ (p. 108)」を参照してください。
- バージョン ID – バケット内ではキーとバージョン ID によってオブジェクトが一意に識別されます。バージョン ID は、バケットにオブジェクトを追加すると Amazon S3 によって生成される文字列です。 詳細については、「オブジェクトのバージョニング (p. 111)」を参照してください。
- 値 – 格納するコンテンツです。
  オブジェクトの値は任意のバイトシーケンスです。オブジェクトのサイズの範囲は 0 ~ 5 TB です。
  詳細については、「オブジェクトのアップロード (p. 168)」を参照してください。
- メタデータ – オブジェクトに関する情報と一緒に格納できる名前と値のペアのセットです。
  Amazon S3 内のオブジェクトには、ユーザー定義メタデータと呼ばれるメタデータを割り当てることができます。Amazon S3 では、オブジェクトの管理目的で使用されるシステムメタデータもオブジェクトに割り当てられます。詳細については、「オブジェクトキーとメタデータ (p. 108)」を参照してください。

- サブリソース – Amazon S3 では、オブジェクト固有の追加情報を格納するためにサブリソースメカニズムを使用します。
  サブリソースはオブジェクトの従属物なので、オブジェクトやバケットといった他のエンティティに常に関連付けられます。詳細については、「オブジェクトのサブリソース (p. 110)」を参照してください。
- アクセスコントロール情報 – Amazon S3 に格納するオブジェクトへのアクセスはコントロールできます。
  Amazon S3 は、アクセスコントロールリスト (ACL)、バケットポリシーなどのリソースベースのアクセスコントロールと、ユーザーベースのアクセスコントロールをサポートしています。 詳細については、「アクセスコントロール (p. 294)」を参照してください。

オブジェクトキーとメタデータ

Amazon S3 の各オブジェクトには、データ、キー、およびメタデータが含まれます。オブジェクトを作成するときは、キー名を指定します。このキー名によってバケット内でオブジェクトが一意に識別されます。例えば、Amazon S3 コンソール (「AWS マネジメントコンソール」を参照) では、バケットをハイライトすると、そのバケットに含まれるオブジェクトのリストが表示されます。表示される名前がオブジェクトキーです。キー名は一連の Unicode 文字で、UTF-8 にエンコードすると最大で 1,024 バイト長になります。

Note

Amazon S3 に対するワークロードが 1 秒あたり 100 個のリクエストを超えることが予想される場合、パフォーマンスを最適化するために Amazon S3 キーの命名のガイドラインに従ってください。詳細については、「リクエスト率およびリクエストパフォーマンスに関する留意事項 (p. 445)」を参照してください。

Amazon S3 の各オブジェクトには、キーに加えメタデータも含まれます。メタデータは名前と値のペアのセットです。オブジェクトメタデータは、オブジェクトをアップロードするときに設定できます。オブジェクトのアップロード後にはオブジェクトメタデータは変更できません。オブジェクトメタデータを変更する唯一の方法は、オブジェクトのコピーを作成し、メタデータを設定することです。詳細については、『*Amazon Simple Storage Service API リファレンス*』の「PUT Object - Copy」を参照してください。Amazon S3 マネジメントコンソールを使用してオブジェクトメタデータを更新することもできますが、この場合、内部的にはオブジェクトのコピーを作成し、既存のオブジェクトを置き換えてメタデータを設定しているだけです。

メタデータにはシステムメタデータとユーザー定義メタデータの 2 種類があります。

システム定義のメタデータ

Amazon S3 では、バケットに格納されるオブジェクトごとに、一連のシステムメタデータが維持されます。Amazon S3 はこのシステムメタデータを必要に応じて処理します。例えば、Amazon S3 はオブジェクトの作成日とサイズに関するメタデータを維持し、オブジェクト管理の目的でこの情報を使用します。

システムメタデータには 2 つのカテゴリがあります。

- オブジェクト作成日などのメタデータはシステムによって制御され、Amazon S3 だけがその値を変更できます。
- オブジェクトについて設定するストレージクラスなどの他のシステムメタデータや、オブジェクトでサーバー側の暗号化が有効になっているかどうかなどは、開発者が値を制御するシステムメタデータの例です。バケットをウェブサイトとして設定している場合、ページリクエストを別のページや外部 URL にリダイレクトしたいことがあります。この場合、ウェブページはバケット内のオブジェクト

です。Amazon S3 は、ページリダイレクト値をシステムメタデータとして保存し、ユーザーがその値を制御します。

オブジェクトを作成するときは、このようなシステムメタデータ項目を設定でき、必要に応じて値を更新できます。ストレージクラスおよびサーバー側の暗号化の詳細については、「データ暗号化の使用 (p. 364)」を参照してください。

次の表は、システム定義のメタデータのリストとユーザーがそれを更新できるかどうかをまとめたものです。

名前	説明	ユーザーが値を変更できるか
日付	オブジェクト作成日。	いいえ
Content-Length	オブジェクトのサイズ (バイト単位) 。	いいえ
Last-Modified	オブジェクトが最後に変更された日付。	いいえ
Content-MD5	オブジェクトの base64 エンコードされた 128 ビット MD5 ダイジェスト。	いいえ
x-amz-server-side-encryption	オブジェクトでサーバー側の暗号化が有効になっている (オブジェクトデータが保管時に暗号化されるか) どうかを示します。詳細については、「サーバー側の暗号化の使用 (p. 365)」を参照してください。	はい
x-amz-version-id	オブジェクトのバージョン。バケットのバージョニングを有効にすると、Amazon S3 はバケットに追加されたオブジェクトにバージョン番号を割り当てます。詳細については、「バージョニングの使用 (p. 388)」を参照してください。	いいえ
x-amz-delete-marker	バージョンが有効にされているバケットで、このブール値マーカーはオブジェクトが削除マーカーであるかどうかを示します。	いいえ
x-amz-storage-class	オブジェクトの保存に使われるストレージクラス。	はい
x-amz-website-redirect-location	関連付けられたオブジェクトのリクエストを同じバケット内の別のオブジェクトまたは外部 URL にリダイレクトします。詳細については、「ウェブページリダイレクトの設定 (p. 423)」を参照してください。	はい

ストレージクラス

Amazon S3 内にある各オブジェクトは、ストレージクラスと関連付けられています。Amazon S3 は、次のストレージクラスをサポートしています。

- 標準 – 標準ストレージクラスは、年間 99.999999999% のオブジェクトの耐久性と、99.99% のオブジェクトの可用性を提供するよう設計されています。2 つの施設で同時にデータ喪失が発生しないよう設計されています。
- 低冗長化ストレージ (RRS) – RRS ストレージクラスでは、それほど重要でない再生可能なデータを、標準ストレージクラスより低いレベルの冗長性で格納することで、コストを削減します。年間 99.99% のオブジェクトの耐久性と可用性を提供します。この耐久性レベルは、年間平均予測オブジェクト喪失率 0.01% に該当します。

- Glacier – Glacier ストレージクラスはデータのアーカイブに適しています。つまり、データへのアクセスが頻繁でなく、データの取得に数時間かかっても問題ない場合です。Glacier ストレージクラスでは、非常に低コストな Amazon Glacier ストレージサービスが使用されますが、このストレージクラス内にあるオブジェクトの管理は、引き続き Amazon S3 経由で行います。

Note

オブジェクトをアップロードする際に Glacier ストレージクラスと関連付けることはできません。既存の Amazon S3 オブジェクトを Glacier ストレージクラスに移行するには、ライフサイクル管理を使用します。詳細については、「オブジェクトのライフサイクル管理 (p. 114)」を参照してください。

ユーザー定義メタデータ

オブジェクトをアップロードするときに、そのオブジェクトにメタデータを割り当てることもできます。このオプション情報は、オブジェクトを作成するための PUT リクエストまたは POST リクエストを送信するときに、名前と値のペアとして指定します。REST API を使用してオブジェクトをアップロードするときは、他の HTTP ヘッダーと区別するため、オプションのユーザー定義メタデータの名前を「x-amz-meta-」で開始する必要があります。REST API を使用してオブジェクトを取得するときは、このプレフィックスが返されます。SOAP API を使用してオブジェクトをアップロードするとき、このプレフィックスは必要ありません。SOAP API を使用してオブジェクトを取得するときは、オブジェクトのアップロードに使用した API にかかわらず、プレフィックスは削除されます。

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

REST API を介してメタデータを取得するとき、同じ名前を持つヘッダー (大文字と小文字は区別されません) は Amazon S3 によってコンマ区切りリストとして結合されます。出力不可能な文字が含まれているメタデータは返されません。その代わりに、「x-amz-missing-meta」ヘッダーが、出力不可能なメタデータエントリの数を示す値と共に返されます。

Amazon S3 はユーザー定義メタデータを小文字で格納します。名前と値の各ペアは、REST 使用時には US-ASCII に、SOAP の使用時または POST を介したブラウザベースでのアップロード時には UTF-8 に、それぞれ準拠している必要があります。

Note

PUT リクエストヘッダーのサイズは 8 KB に制限されています。PUT リクエストヘッダー内のユーザー定義メタデータのサイズは 2 KB に制限されています。ユーザー定義メタデータはキーと値のペアのセットです。ユーザー定義メタデータのサイズは、キーと値の各ペアを UTF-8 にエンコードしたバイト数の合計に基づいて測定されます。

オブジェクトのサブリソース

Amazon S3 では、バケットとオブジェクトに関連付ける一連のサブリソースが定義されています。サブリソースはオブジェクトの従属物です。つまり、それ自体では存在できず、常にオブジェクトやバケットなどその他のエンティティとの関連付けで初めて存在するものです。

次の表は、Amazon S3 のオブジェクトに関連付けられるサブリソースをまとめたものです。

サブリソース	説明
acl	アクセス許可の被付与者と付与されるアクセス許可を識別するリストが含まれます。オブジェクトを作成するとき、`acl`を使用することで、オブジェクト所有者がオブジェクトに対して完全なコントロールを持つことを指定します。オブジェクト ACL は、取得したり、更新されたアクセス許可のリストで置き換えたりすることができます。ACL を更新するときは必ず既存の ACL を置き換える必要があります。ACL の詳細については、「ACL の使用 (p. 347)」を参照してください。
torrent	Amazon S3 は、BitTorrent プロトコルをサポートしています。Amazon S3 では `torrent` サブリソースを使用して、特定のオブジェクトに関連付けられた torrent ファイルが返されます。torrent ファイルを取得するには、GET リクエストの中で `torrent` サブリソースを指定します。Amazon S3 によって torrent ファイルが作成されて返されます。`torrent` サブリソースは取得のみ可能です。`torrent` サブリソースを作成、更新、削除することはできません。詳細については、「Amazon S3 での BitTorrent の使用 (p. 450)」を参照してください。

オブジェクトのバージョニング

バージョニングを使用すると、1 つのバケットで複数バージョンのオブジェクトを維持できます。バージョニングを使用する場合は、バケットに対してバージョニングを明示的に有効にする必要があります。デフォルトでバージョニングは無効になっています。バケットに対してバージョニングを有効にしたかどうかにかかわらず、各オブジェクトはバージョン ID を持ちます。バケットに対してバージョニングを有効にしていない場合、バージョン ID の値は Amazon S3 によって null に設定されます。バケットに対してバージョニングを有効にした場合は、そのオブジェクトの一意のバージョン ID が Amazon S3 によって割り当てられます。

バージョニングを使用すると、1 つのバケットで複数バージョンのオブジェクトを維持できます。例えば、`my-image.jpg` (バージョン 111111) 、`my-image.jpg` (バージョン 222222) といった具合です。バージョニングを有効にすると、誤って上書きまたは削除したオブジェクトを回復したり、以前のバージョンを取得できるようオブジェクトをアーカイブしたりできます。

Note

SOAP API は、バージョニングをサポートしていません。SOAP のサポートは HTTP 経由では廃止されましたが、HTTPS 経由では引き続き利用可能です。SOAP 用に Amazon S3 の新機能をサポートする予定はありません。

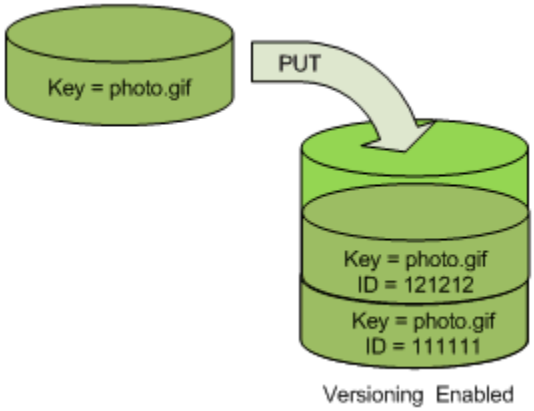
バケットに対してバージョニングを有効にしても、バケット内の既存のオブジェクトに変更はありません。バージョン ID (null) 、コンテンツ、アクセス許可は同じです。

バージョニングの有効化と停止はバケットレベルで行います。バケットに対してバージョニングを有効にすると、バケットに追加したすべてのオブジェクトは一意のバージョン ID を持ちます。一意のバージョン ID はランダムに生成されます。Unicode、UTF-8 エンコード、URL 対応、意味不定、最大1,024 バイト長といった特徴を持つ文字列です。例えば `3/L4kqtJlcpXroDTDmJ+rmSpXd3dIbrHY+MTRCxf3vjVBH40Nr8X8gdRQBpUMLUo` はバージョン ID です。バージョン ID は Amazon S3 によってのみ生成されます。編集することはできません。

Note

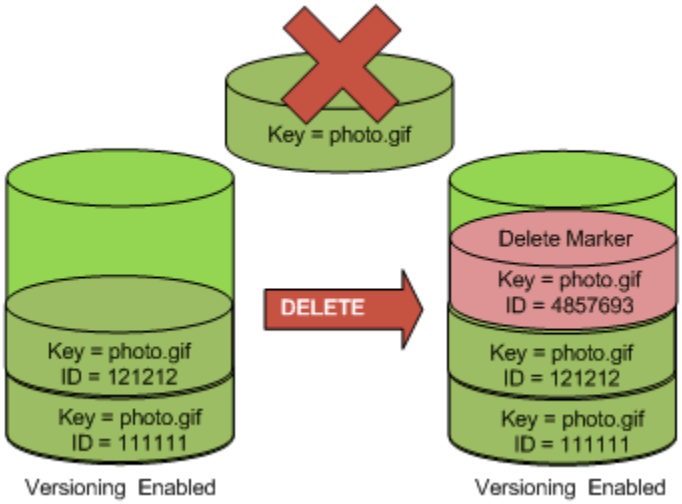
以降紹介するすべての例では、簡潔さを考慮してもっと短い ID を使用します。

バージョニング対応のバケットにオブジェクトを PUT しても、古いバージョンは上書きされません。次の図は、photo.gif という名前のオブジェクトが既に入っているバケットに、同じ名前を持つ新しいバージョンのオブジェクトを PUT する様子を示しています。バケット内の元のオブジェクト (ID = 111111) はそのまま残り、Amazon S3 によって新しいバージョン ID (121212) が生成され、その新しいバージョンがバケットに追加されます。

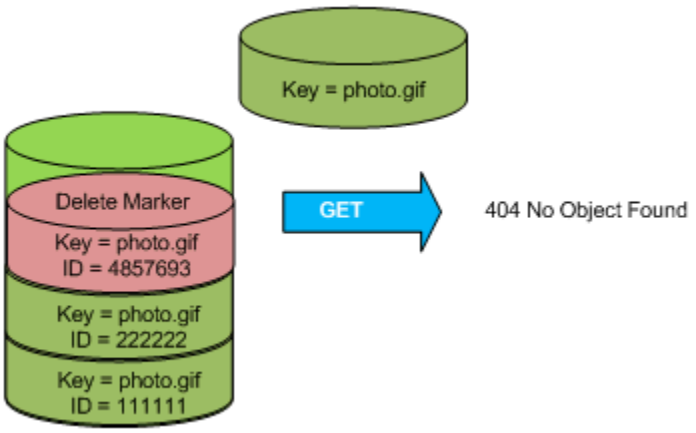


この機能によって、オブジェクトを誤って上書きまたは削除することがなくなり、オブジェクトの以前のバージョンを回復することも可能になります。

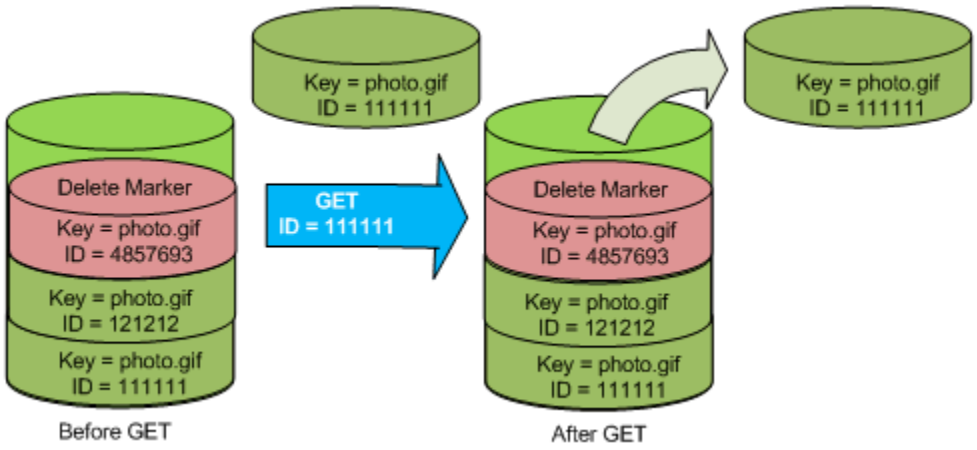
オブジェクトを DELETE するときは、次の図に示すように、バケット内のすべてのバージョンが残り、Amazon S3 によって削除マーカーが挿入されます。



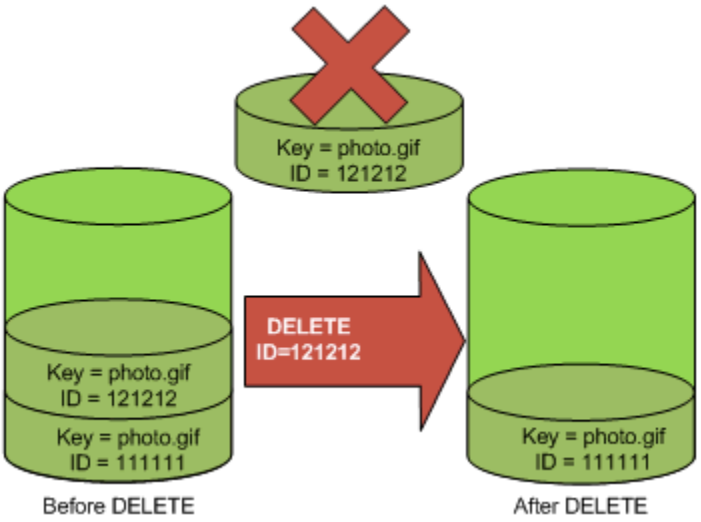
削除マーカーはオブジェクトの最新バージョンになります。デフォルトで、GET リクエストは最後に格納されたバージョンを取得します。最新バージョンが削除マーカーであるときに単純な GET Object リクエストを実行すると、次の図に示すように 404 Not Found エラーが返されます。



ただし、バージョン ID を指定すれば、オブジェクトの以前のバージョンを GET することができます。次の図では、特定のオブジェクトバージョン 111111 を GET しています。Amazon S3 は、このオブジェクトバージョンを、最新のバージョンではなくても返します。



オブジェクトを完全に削除するには、削除するバージョンを指定します。Amazon S3 バケットの所有者のみが、バージョンを完全に削除することができます。次の図は、`DELETE versionId` によってバケット内のオブジェクトが完全に削除される様子を示しています。Amazon S3 による削除マーカーの挿入も行われません。



セキュリティを強化するには、MFA (Multi-Factor Authentication) Delete に対応するようにバケットを設定します。この設定を行うと、バケット所有者は特定のバージョンを削除したりバケットのバージョニング状態を変更したりするために、すべてのリクエストに2つの認証形態を含める必要があります。詳細については、「MFA Delete (p. 389)」を参照してください。

詳細については、「バージョニングの使用 (p. 388)」を参照してください。

オブジェクトのライフサイクル管理

Topics

ライフサイクル管理では、Amazon S3 が、オブジェクトをその存続期間にわたってどのように管理するかを定義します。

Amazon S3 バケットに保存したオブジェクトの中には、明確なライフサイクルが設定されているオブジェクトもあります。

- 定期的なログをバケットにアップロードしている場合、作成した 1 週間後や 1 か月後にそのログがアプリケーションで必要になる可能性があるものの、その後は削除してかまわないこともあります。
- ドキュメントには、一定の期間中に頻繁にアクセスされるものがあります。その後は必ずしもそれらのオブジェクトにリアルタイムでアクセスする必要はないものの、所属している組織によって、それより長い期間アーカイブしておくよう要求される場合があります。その期間が過ぎれば、必要に応じて削除してかまいません。主にアーカイブを目的として Amazon S3 にアップロードするデータの種類には、デジタルメディアのアーカイブ、財務や医療の記録、生のゲノムシーケンスデータ、データベースの長期バックアップ、法規制準拠のために保管が必要なデータなどがあります。

そのようなオブジェクトに対しては、影響を受けるオブジェクト、タイムライン、および Amazon S3 でそれらのオブジェクトに実行する必要がある具体的なアクションを特定するルールを定義できます。

Amazon S3 では、ライフサイクル設定を使用してオブジェクトの存続期間を管理します。ライフサイクル管理はバケットに割り当てられ、個々のオブジェクトのルールを定義します。ライフサイクル設定に含まれる各ルールは、次のもので構成されます。

- ルールの適用先となる 1 つ以上のオブジェクトを特定する、オブジェクトキープレフィックス。
- 指定したオブジェクトに対して Amazon S3 で実行する必要がある 1 つまたは複数のアクション。
- 指定したアクションを Amazon S3 で実行するタイミングを示す日付、またはオブジェクト作成からの日数で指定する期間。

これらのルールをバケットに追加するには、 Amazon S3 コンソールを使用するか、プログラムから行います。

ライフサイクル設定ルール

1 つのルールは、単一のオブジェクトか、キー名がルールに指定されたプレフィックスで始まる複数のオブジェクトに適用できます。例えば、次のオブジェクトがあるとします。

logs/day1

logs/day2

logs/day3

ExampleObject.jpg

`ExampleObject.jpg` をプレフィックスとして指定すると、ルールは特定のオブジェクトに適用されます。`logs/` をプレフィックスとして指定すると、ルールはキー名が文字列「logs/」で始まる 3 つのオブジェクトに適用されます。

ルールでは、次のアクションを指定できます。

- 移行 – オブジェクトの存続期間に指定された日付または期間に到達すると、ストレージクラスが Glacier に設定されます。ストレージクラスの詳細については、「ストレージクラス (p. 109)」を参照してください。

  例えば、Amazon S3 のルールを設定して、「MyArchiveObject.jpg」が作成されてから 30 日後に Glacier ストレージクラスに移行させることができます。あるいは、キープレフィックス「glacierobjects/」が付いているすべてのオブジェクトを、作成の 1 年後に Glacier ストレージクラスに移行させることもできます。

- 有効期限切れ – オブジェクトの存続期間に指定された期間に到達すると、オブジェクトが削除されます。

  例えば、有効期限切れアクションのあるルールを設定して、キープレフィックス「log2012-01-01」が付いているオブジェクトを、作成の 30 日後に削除することができます。

  有効期限切れは、 Amazon Glacier にアーカイブされているものを含め、すべての Amazon S3 オブジェクトに適用されます。

指定したオブジェクトでアクションを実行するタイミングを、日付または期間 (オブジェクト作成からの日数) で指定できます。

- 日数を指定すると、Amazon S3 は、ルールに指定された日数をオブジェクトの作成時間に加算し、得られた日時を翌日の午前 00:00 UTC (協定世界時) に丸めることで、時間を算出します。例えば、

あるオブジェクトが 2012 年 1 月 15 日午前 10:30 UTC に作成されたときに、移行ルールに 3 日と指定すると、オブジェクトの移行日は 2012 年 1 月 19 日午前 00:00 UTC となります。

- ルールで日付を指定すると、指定したアクションは指定した日付に実行されます。つまり、Amazon S3 は、指定した日付にオブジェクトを移行させるか有効期限切れにします。

これらのルールは、バケットの `lifecycle` サブリソースとして利用できます。ライフサイクル設定の例を次に示します。

```
<LifecycleConfiguration>
  <Rule>
    <ID>Example Rule</ID>
    <Prefix>projectdocs/</Prefix>
    <Status>Enabled</Status>
    <Transition>
      <Days>365</Days>
      <StorageClass>GLACIER</StorageClass>
    </Transition>
    <Expiration>
      <Days>3650</Days>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

ライフサイクル設定には 1,000 個のルールを含めることができます。前述のルールでは、移行アクションと有効期限切れアクションを定義しています。このルールは、キー名がプレフィックス projectdocs/ で始まるすべてのオブジェクトに適用されます。このルールは Amazon S3 に対して、オブジェクトを作成した 365 日後に Glacier ストレージクラスに移行させ、3650 日後に削除するようにリクエストします。

Important

前述の例では、移行アクションと有効期限切れアクションに指定された期間はどちらも、*オブジェクトの作成日*からの相対値でした。

新しい、あるいは更新されたライフサイクル設定がすべての Amazon S3 システムに完全に伝達されるまでには、通常、多少のタイムラグがあります。ライフサイクル設定が完全に有効になるまで、数分間程度の遅延を想定してください。これは、ライフサイクル設定を削除する場合も同様です。

Note

バケットがバージョニング対応になっている場合、またはバージョニングが停止中の場合は、ライフサイクル設定を追加することはできません。

ライフサイクル設定をバケットに追加すると、設定ルールは既存のオブジェクトと、それ以降に追加するオブジェクトの両方に適用されます。例えば、特定のプレフィックスが付いたオブジェクトが作成から 30 日後に有効期限を迎えるようにする有効期限切れアクションを備えたライフサイクル設定ルールを本日追加すると、作成から 30 日以上が経過している既存のオブジェクトがすべて削除キューに追加されます。

オブジェクトをアーカイブすることに決定する前に

ライフサイクル設定ルールの移行アクションでは、オブジェクトを Glacier ストレージクラスに移行させることができます。つまり、Amazon Glacier にデータをアーカイブします。オブジェクトのアーカイブを決定する前に、次のことに注意してください。

- Glacier ストレージクラス内のオブジェクトはリアルタイムでは利用できません。

  アーカイブされたオブジェクトは Amazon S3 オブジェクトですが、アーカイブオブジェクトにアクセスする前に、まずその一時コピーを復元する必要があります。復元されたオブジェクトのコピーは、復元リクエストに指定された期間内のみ利用できます。その後、Amazon S3 によって一時コピーが削除されます。オブジェクトは Amazon Glacier にアーカイブされたまま残ります。

  アーカイブからオブジェクトを復元するには 3 時間から 5 時間かかる可能性があることに注意してください。
- 移行アクションでは、Glacier ストレージクラスへの一方向の移行しか許可されません。

  ライフサイクル設定ルールを使用して Glacier オブジェクトを標準オブジェクトや低冗長化ストレージ (RRS) オブジェクトに変換することはできません。既にアーカイブされたオブジェクトのストレージクラスを標準または RRS に変更する場合、まず復元操作を使用して一時コピーを作成する必要があります。その後、コピー操作を使用してオブジェクトを標準または RRS オブジェクトとして上書きします。
- Glacier オブジェクトは、 Amazon Glacier ではなく、Amazon S3 でのみ表示および利用できます。

  Amazon S3 では、アーカイブされたオブジェクトは &GL&; に保存されますが、これらは Amazon S3 オブジェクトのため、Amazon S3 のコンソールまたは API を使用することによってのみアクセスできます。アーカイブされたオブジェクトに、Amazon Glacier のコンソールまたは API 経由でアクセスすることはできません。
- 空のキープレフィックスを持つルールが、バケット内のすべてのオブジェクトに適用されます。

  空のプレフィックスを指定すると、ルールはバケット内のすべてのオブジェクトに適用されます。したがって、空のプレフィックスでの移行アクションがルールで指定されている場合、Amazon S3 に対し、バケット内にあるすべてのオブジェクトを、そのオブジェクトの存続期間における特定の期間でアーカイブするリクエストをしていることになります。

詳細については、「オブジェクトのアーカイブ (Glacier ストレージクラスへのオブジェクトの移行) (p. 121)」を参照してください。Amazon S3 のストレージクラスについては、「ストレージクラス (p. 109)」を参照してください。

オブジェクトを有効期限切れとして決定する前に

ライフサイクル設定ルールの有効期限切れアクションでは、オブジェクトを削除するように Amazon S3 にリクエストできます。ライフサイクル設定にこのアクションを設定することを決定する前に、次のことに注意してください。

- 有効期限切れアクションではオブジェクトが削除されます。

  オブジェクトは Amazon S3 にあるか、Amazon Glacier にアーカイブされている場合があります。オブジェクトはどこにあっても Amazon S3 によって削除されます。削除されたオブジェクトにはアクセスできなくなります。
- 空のキープレフィックスを持つルールが、バケット内のすべてのオブジェクトに適用されます。

  空のプレフィックスを指定すると、ルールはバケット内のすべてのオブジェクトに適用されます。したがって、空のプレフィックスでの有効期限切れアクションがルールで指定されている場合、Amazon S3 に対し、バケット内にあるすべてのオブジェクトを、そのオブジェクトの存続期間における特定の期間でアーカイブするリクエストをしていることになります。

ライフサイクル設定の指定

プログラムによって、または Amazon S3 コンソールを使用して、バケットでライフサイクル設定を指定できます。

Amazon S3 コンソールを使用したライフサイクル設定の指定

Amazon S3 コンソールを使用すると、バケットのオブジェクトに対して最大 1000 個のライフサイクルルールを設定できます。詳細については、「AWS マネジメントコンソールを使用したオブジェクトのライフサイクルの管理 (p. 124)」を参照してください。

プログラムによるライフサイクル設定の指定

Amazon S3 には、ライフサイクル設定をバケットに追加するための API アクションが用意されています。Amazon S3 では、ライフサイクル設定はバケットの `lifecycle` サブリソースに保存されます。詳細については、「REST API を使用したオブジェクトのライフサイクルの管理 (p. 133)」を参照してください。

また、AWS SDK を使用して、ライフサイクル設定をバケットに追加することもできます。

Amazon S3 API または AWS SDK のどちらをライフサイクル設定の管理に使用する場合も、ライフサイクル設定を定義するためのプロセスは共通です。ライフサイクル設定は XML で指定されます。次に示すように、この XML には一連のルールが含まれます。

```
<LifecycleConfiguration>
  <Rule>
      ...
  </Rule>
  <Rule>
      ...
  </Rule>
   ...
</LifecycleConfiguration>
```

各ルールでは、キープレフィックスによってオブジェクトまたはオブジェクトグループが特定されます。また、特定の時間に、またはオブジェクトを作成してから一定の期間にわたって、Amazon S3 が実行するアクションが指定されます。次の XML 要素によってルールが定義されます。

- <ID> 要素 – ルールの一意な識別子です。
- <Status> 要素 – Enabled (有効) 、Disabled (無効) のいずれかです。Amazon S3 は、無効にされたルールを評価しません。
- <Prefix> 要素 – ルールが適用されるオブジェクトを特定するキープレフィックスです。Prefix 値を指定しない場合、ルールはバケット内のすべてのオブジェクトに適用されます。
- Action 要素 – 1 つのルールに、次のいずれかまたは両方のアクションを定義できます。ただし、1 つのルールに同じアクションを 2 回以上は定義できません。
  - <Expiration> アクション – オブジェクトの存続期間の有効期限が切れるタイミングを、特定の日付、または作成してからの日数で定義します。子要素に <Date> または <Days> を持たせることができます。
  - <Transition> アクション – オブジェクトが Glacier ストレージクラスに移行するタイミングを、日付、または作成してからの日数で定義します。子要素として <Date> または <Days> を持つことができます。

ライフサイクル設定の例を次に示します。REST API や AWS SDK の詳細については、このセクションの冒頭にある当該のリンクをクリックしてください。

例 1: ルールを指定する

次のライフサイクル設定のルールには 2 つのアクションが含まれます。このルールは Amazon S3 に対して、キープレフィックス `projectdocs/` のオブジェクトを作成の 365 日後に Amazon Glacier ストレージクラスに移行させ、3,650 日後に削除するようにリクエストします。

```
<LifecycleConfiguration>
  <Rule>
    <ID>Transition and Expiration Rule</ID>
    <Prefix>projectdocs/</Prefix>
    <Status>Enabled</Status>
    <Transition>
      <Days>365</Days>
      <StorageClass>GLACIER</StorageClass>
    </Transition>
    <Expiration>
      <Days>3650</Days>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

期間の代わりに各アクションに日付を設定できます。ただし、同じルールに日付と期間の両方を使用することはできません。

前述の例では、2 つのアクションを持つルールが 1 つあります。1 つのオブジェクトに、次のいずれかのアクションを実行するルールを 2 つ定義することで、同じ結果を得ることができます。

```
<LifecycleConfiguration>
  <Rule>
    <ID>Transition Rule</ID>
    <Prefix>projectdocs/</Prefix>
    <Status>Enabled</Status>
    <Transition>
      <Days>365</Days>
      <StorageClass>GLACIER</StorageClass>
    </Transition>
  </Rule>
  <Rule>
    <ID>Expiration Rule</ID>
    <Prefix>projectdocs/</Prefix>
    <Status>Enabled</Status>
    <Expiration>
      <Days>3650</Days>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

例 2: バケット内のすべてのオブジェクトに適用されるルールを指定する

次のライフサイクル設定では、空のプレフィックスと移行日数 0 が指定されます。プレフィックスを指定しない場合、ルールはバケット内のすべてのオブジェクトに適用されます。Days 要素が 0 に設定されているため、すべてのオブジェクトはただちに Amazon Glacier へのアーカイブの対象になります。Amazon S3 はこれらのオブジェクトを、定期的に実行されるプロセスを使用してアーカイブします。

```
<LifecycleConfiguration>
  <Rule>
    <ID>Archive all object immediately upon creation</ID>
    <Prefix></Prefix>
    <Status>Enabled</Status>
    <Transition>
      <Days>0</Days>
      <StorageClass>GLACIER</StorageClass>
    </Transition>
  </Rule>
</LifecycleConfiguration>
```

例 3: ルールを無効にする

次のライフサイクル設定では 2 つのルールを指定していますが、そのうちの 1 つが無効になっています。Amazon S3 では、無効なルールに指定されているアクションを実行しません。

```
<LifecycleConfiguration>
  <Rule>
    <ID>30 days log objects expire rule</ID>
    <Prefix>logs/</Prefix>
    <Status>Enabled</Status>
    <Expiration>
      <Days>30</Days>
    </Expiration>
  </Rule>
  <Rule>
    <ID>1 year documents expire rule</ID>
    <Prefix>documents/</Prefix>
    <Status>Disabled</Status>
    <Expiration>
      <Days>365</Days>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

例 4: 許可されていないルールをオーバーラップする

ルールはオーバーラップできません。例えば、次のライフサイクル設定には、「documents/」というプレフィックスが付いたオブジェクトが 30 日後に有効期限切れになるルールがあります。一方、別のルールには、「documents/2011」というプレフィックスが付いたオブジェクトが 365 日後に有効期限切れになるように設定されています。このような場合はエラーメッセージが返されます。

```
<LifecycleConfiguration>
  <Rule>
    <ID>111</ID>
    <Prefix>documents/</Prefix>
    <Status>Enabled</Status>
    <Expiration>
      <Days>30</Days>
    </Expiration>
  </Rule>
  <Rule>
    <ID>222</ID>
    <Prefix>documents/2011</Prefix>
```

```
    <Status>Enabled</Status>
    <Expiration>
      <Days>365</Days>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

オブジェクトのアーカイブ（Glacier ストレージクラスへのオブジェクトの移行）

Topics

Amazon S3 内にある各オブジェクトは、ストレージクラスと関連付けられています。オブジェクトをアップロードすると、デフォルトではそのオブジェクトは標準ストレージクラスと関連付けられます。この場合、Amazon S3 は高い耐久性を確保するために冗長コピーを維持します。それほど重要でない再生可能なデータについては、低冗長化ストレージ（RRS）オプションをご利用いただけます。この場合、Amazon S3 は標準ストレージクラスより低レベルの冗長性を使用します。RRS は費用対効果および可用性の高いソリューションを提供します。標準オブジェクトと RRS オブジェクトはどちらもリアルタイムで高い可用性があります。リアルタイムでアクセスする必要がないオブジェクトには、Glacier ストレージクラスも提供されています。このストレージクラスは、主にアーカイブ目的で格納されるオブジェクトに最適です。

オブジェクトをアップロードする際に、標準ストレージクラスまたは RRS ストレージクラスを指定することはできますが、Glacier ストレージクラスを割り当てることはできません。既存のオブジェクトを Glacier ストレージクラスに移行するには、代わりに オブジェクトのライフサイクル管理 (p. 114) を使用します。Amazon S3 はオブジェクトをアーカイブし、アーカイブしたオブジェクトを定義済みルールに従って Glacier ストレージクラスのオブジェクトに関連付けます。

ライフサイクル設定を利用すると、Glacier への一方向の移行が可能になります。ストレージクラスを Glacier から標準または RRS に変更するには、後続のセクションで説明するようにオブジェクトを復元してから、復元されたオブジェクトのコピーを作成する必要があります。

料金表に関する留意事項

アクセス頻度の低いデータを数か月か数年間アーカイブする場合、ストレージコストは Glacier ストレージクラスによって通常削減されます。しかし、Glacier ストレージクラスが適切な選択となるよう、以下の点に留意してください。

- ストレージオーバーヘッド料金 – オブジェクトを Glacier ストレージクラスに移行すると、そのオブジェクトを管理するメタデータを収容するために、各オブジェクトに対して一定量のストレージが追加されます。
  - Amazon Glacier にアーカイブされたオブジェクトごとに、Amazon S3 ではオブジェクトの名前とその他のメタデータのために 8 KB のストレージを使用しています。Amazon S3 はこのメタデータを保管することで、ユーザーが Amazon S3 API（「Get Bucket (List Objects)」を参照）を使用して、アーカイブされたオブジェクトのリアルタイムのリストを取得できるようにします。この追加ストレージに対しては、Amazon S3 の標準料金で課金されます。
  - アーカイブされたオブジェクトごとに、Amazon Glacier では、インデックスおよび関連するメタデータのために 32 KB のストレージが追加されています。 この追加データは、オブジェクトを特定して復元するのに必要です。この追加ストレージに対しては、Amazon Glacier の料金で課金されます。

小さなオブジェクトをアーカイブする場合は、このストレージ料金に留意する必要があります。オーバーヘッドコストを削減するために、多数の小さなオブジェクトを少数の大きなオブジェクトに集約することを検討してください。

- オブジェクトをアーカイブしておく日数 – Amazon Glacier は長期間のアーカイブを行うソリューションです。Amazon Glacier にアーカイブされているデータの削除は、削除するオブジェクトが 3 か月以上アーカイブされている場合は、無料です。オブジェクトをアーカイブしてから 3 か月以内に削除または上書きする場合は、日割りによる早期削除料金が課金されます。
- Glacier アーカイブリクエストコスト – Glacier ストレージクラスに移行するオブジェクトごとに 1 つのアーカイブリクエストが作成されます。アーカイブリクエストごとにコストが発生します。多数のオブジェクトを移行する場合は、リクエストコストに留意する必要があります。

Amazon S3 製品詳細ページに、Amazon S3 オブジェクトをアーカイブする場合の料金表情報と計算例が掲載されています。詳細については、Amazon S3 製品詳細ページにある以下のトピックを参照してください。

- Amazon Glacier にアーカイブされた Amazon S3 オブジェクトのストレージ料金はどのように計算されますか？
- Amazon Glacier から保存後 3 か月未満のオブジェクトを削除するにはいくらかかりますか？
- 標準ストレージクラスと Glacier ストレージクラスのストレージコストを示す「Amazon S3 料金表」。このページには Glacier アーカイブリクエストの料金表も掲載されています。

アーカイブされたオブジェクトの復元

アーカイブされたオブジェクトにはリアルタイムでアクセスできません。まず復元リクエストを開始してから、リクエストで指定した期間中にオブジェクトの一時コピーが利用できるようになるまで待ちます。復元ジョブは通常、完了までに 3 時間から 5 時間かかるため、リアルタイムでアクセスする必要がないオブジェクトのみをアーカイブすることが重要です。

復元されたオブジェクトの一時コピーの取得後も、オブジェクトのストレージクラスは Glacier のままです。Amazon S3 はこの一時コピーを低冗長化ストレージ（RRS）クラスと関連付けます。なお、アーカイブを復元するときは、そのアーカイブに加えて、一時的に復元されたコピーについても料金が発生します。料金表については、「*Amazon S3 製品詳細ページ*の「料金表」」セクションを参照してください。

オブジェクトのコピーは、プログラムによって、あるいは Amazon S3 コンソールを使用して復元できます。復元リクエストは、一度に 1 つしか処理できません。コンソールと Amazon S3 API の両方を使用して、復元ステータスを確認したり、復元されたコピーが Amazon S3 によって削除されるタイミングを調べたりすることができます。

Amazon S3 コンソールを使用した Glacier オブジェクトの復元

Amazon S3 コンソールを使用した Glacier オブジェクトの復元については、「Amazon S3 コンソールを使用したオブジェクトの復元 (p. 287)」を参照してください。

プログラムを使用した Glacier オブジェクトの復元

Glacier オブジェクトをプログラムによってアプリケーションから直接復元するには、AWS SDK または Amazon S3 API のいずれかを使用します。どちらの場合でも、次の XML を送信して復元期間を指定します。

```
<?xml version="1.0" encoding="UTF-8"?>
<RestoreRequest xmlns="http://s3.amazonaws.com/doc/2006-03-01">
  <Days>NumberOfDaysToRestore</Days>
</RestoreRequest>
```

AWS SDK を使用する場合、Amazon S3 API によって、プログラミングタスクを簡単にするための適切なラッパーライブラリが提供されます。ただし、リクエストが送信されるときに、SDK によって前述の XML がリクエスト本文に組み込まれて送信されます。

プログラムを使用したオブジェクトの復元の詳細については、「オブジェクトの復元 (p. 286)」を参照してください。

オブジェクトの有効期限

Topics

Amazon S3 バケットに保存したオブジェクトに有効期限が設定されている場合があります。例えば、定期的なログをバケットにアップロードしている場合に、ログを特定の期間保持する必要がある場合などです。オブジェクトのライフサイクル管理 (p. 114) を使用して、バケット内のオブジェクトの存続期間を指定できます。オブジェクトの存続期間が終了すると、オブジェクトは削除用のキューに入れられます。

Important

有効期限が切れると、これらのオブジェクトは、Amazon S3 にあっても Amazon Glacier にアーカイブされていても削除されることに注意してください。オブジェクトがアーカイブされている場合でも、 Amazon S3 はアーカイブされたオブジェクトを削除します。Amazon S3 は、復元が完了ている、アーカイブされたオブジェクトの一時的コピーも削除します (「アーカイブされたオブジェクトの復元 (p. 122)」を参照)。

Note

オブジェクトが有効期限に達すると、Amazon S3 は存続期間が終了したオブジェクトを自動的に削除キューに追加して、非同期的に削除します。有効期限が切れてから Amazon S3 でオブジェクトが削除されるまでに遅れが生じることがありますが、有効期限が切れたオブジェクトに関連付けられた保存期間に課金されることはありません。

Note

オブジェクトを上書きすると、オブジェクトの作成日時は、更新日時で上書きされます。これに伴い、オブジェクトの有効期限も、新しい作成日時に基づいて更新されます。

オブジェクトに設定された有効期限を調べるには、GET Object API か HEAD Object API を使用します。これらの API は、オブジェクトの失効情報を提供するレスポンスヘッダーを返します。詳細については、『*Amazon Simple Storage Service API リファレンス*』の「HEAD Object」と「GET Object」を参照してください。

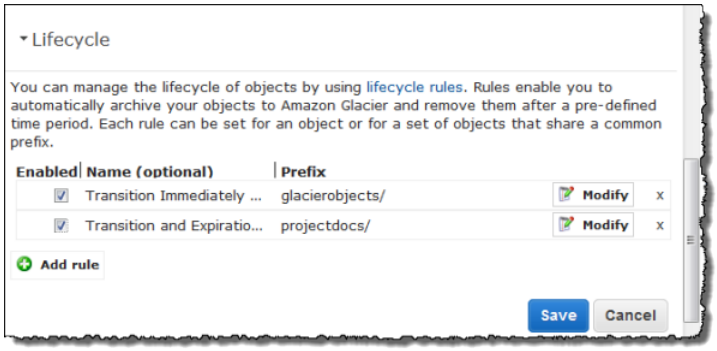
Amazon S3 がオブジェクトをいつ削除したかを調べるには、Amazon S3 サーバーアクセスログを使用します。ログレコードのオペレーションフィールドの「&S3.EXPIRE.OBJECT」という値が、このオペレーションの開始を示しています。詳細については、「サーバーアクセスのロギング (p. 461)」を参照してください。

オブジェクトを削除するには、明示的に DELETE Object アクションを呼び出すか、Amazon S3 で自動的に削除されるようにライフサイクル設定を作成します。バケットのオブジェクトをユーザーまたはアカウントが削除できないようにするには、ユーザーまたはアカウントによる `s3:DeleteObject`、`s3:DeleteObjectVersion`、および `s3:PutLifecycleConfiguration` アクションの呼び出しを拒否する必要があります。

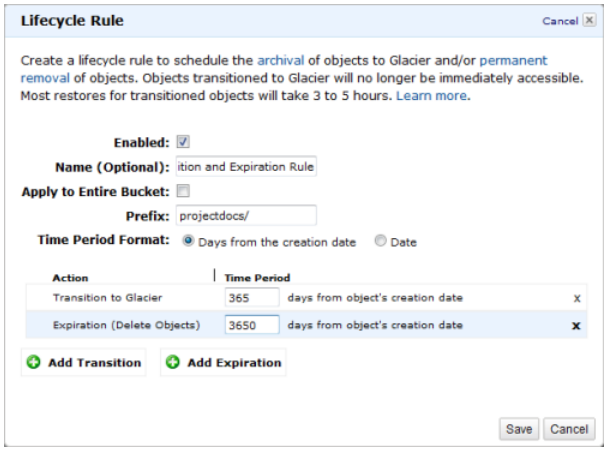
例については、「オブジェクトのライフサイクル管理 (p. 114)」を参照してください。

AWS マネジメントコンソールを使用したオブジェクトのライフサイクルの管理

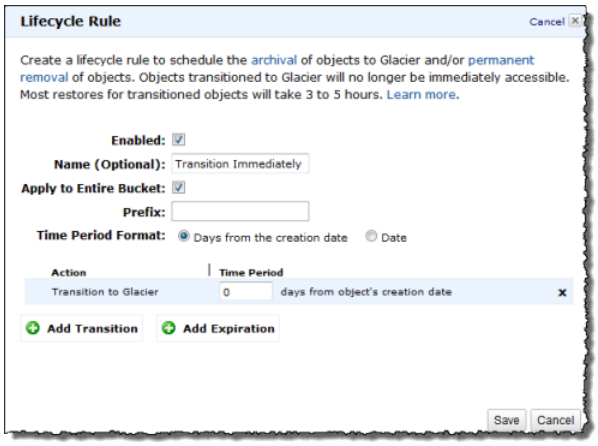
Amazon S3 コンソールを使用して、バケットにライフサイクルルール (「オブジェクトのライフサイクル管理 (p. 114)」を参照) を指定できます。コンソールで、バケット [Properties] には、次のスクリーンショットの例に示すように、[Lifecycle] タブがあります。ここには 2 つのルールを持つバケットが表示され、どちらのルールも有効になっています。[Modify] をクリックしてルールの詳細を表示するか、[Add rule] をクリックして新しいルールを追加します。



次の例に示す [Lifecycle Rule] ウィザードは、ルールの詳細を示しています。ここでは、キープレフィックス「projectdocs/」の付いたオブジェクトにルールが適用されることを示しています。このルールでは、Amazon S3 に対して 2 つのアクションを定義しています。[Transition] アクションは Amazon S3 に対して、オブジェクトを作成してから 1 年後に Glacier ストレージクラスに移行させるように指示します。また、[Expiration] アクションは Amazon S3 に対して、オブジェクトを作成の 10 年後に削除するように指示します。



次のルールは一考に値します。次のスクリーンショットに示すように、Glacier ストレージクラスオブジェクトにバケットを作成し、オブジェクトをただちに Glacier ストレージクラスに移行させるルールを Amazon S3 に作成することがあります。このルールでは、期間を 0 日に設定していることに注意してください。これは、作成後ただちに移行させることを示しています。また、このルールでは、[Prefix] を空に設定しています。これは、ルールがバケット内のすべてのオブジェクトに適用されることを示しています。



手順

詳細な手順については、『*Amazon S3 Console User Guide*』の「Managing Lifecycle Configuration」を参照してください。

AWS SDK for Java を使用したオブジェクトのライフサイクルの管理

AWS SDK for Java を使用して、バケットのライフサイクル設定を管理できます。ライフサイクル設定の管理の詳細については、「オブジェクトのライフサイクル管理 (p. 114)」を参照してください。

このセクションでは、次の表でタスクのサンプルコードスニペットの一部を示します (完全なサンプルプログラムは後述) 。

ライフサイクル設定をバケットに追加するには

1. `BucketLifecycleConfiguration.Rule` オブジェクトを作成してルールを指定します。
2. ルールを使用して `BucketLifecycleConfiguration` オブジェクトを作成します。
3. `s3Client.setBucketLifecycleConfiguration()` を呼び出して、ライフサイクル設定をバケットに追加します。

以下の Java コードスニペットは、前述のタスクの例です。ライフサイクル設定では 1 つのルール (「アーカイブおよび削除ルール」) を定義し、その中でキープレフィックス「projectdocs/」の付いたすべてのオブジェクトに対して Amazon S3 が実行する 2 つのアクションを指定しています。

```
Transition transToArchive = new Transition()
    .withDays(365)
    .withStorageClass(StorageClass.Glacier);

BucketLifecycleConfiguration.Rule ruleArchiveAndExpire = new BucketLifecycleCon
figuration.Rule()
    .withId("Archive and delete rule")
    .withPrefix("projectdocs/")
    .withTransition(transToArchive)
    .withExpirationInDays(3650)
    .withStatus(BucketLifecycleConfiguration.ENABLED.toString());

List<BucketLifecycleConfiguration.Rule> rules = new ArrayList<BucketLifecycleCon
```

```
figuration.Rule>();
rules.add(ruleArchiveAndExpire);

BucketLifecycleConfiguration configuration = new BucketLifecycleConfiguration()

    .withRules(rules);

// Save configuration.
s3Client.setBucketLifecycleConfiguration(bucketName, configuration);
```

既存のライフサイクル設定を更新するには

- ライフサイクル設定をバケットに追加すると、既存のライフサイクル設定が置き換わります。既存のライフサイクル設定を更新するには、次の Java コードス二ペットに示すように、まず既存のライフサイクル設定を取得して変更してから、修正済みのライフサイクル設定をバケットに追加する必要があります。

このス二ペットでは、既存の設定を取得して、ID が NewRule の新しいルールを追加しています。

```
// Retrieve configuration.
configuration = s3Client.getBucketLifecycleConfiguration(bucketName);

// Add a new rule.
Calendar c = Calendar.getInstance(TimeZone.getTimeZone("GMT"));
c.add(Calendar.YEAR, 10);
c.set(Calendar.HOUR_OF_DAY, 0);
c.set(Calendar.MINUTE, 0);
c.set(Calendar.SECOND, 0);
c.set(Calendar.MILLISECOND, 0);
configuration.getRules().add(
        new BucketLifecycleConfiguration.Rule()
            .withId("NewRule")
            .withPrefix("YearlyDocuments/")
            .withExpirationDate(c.getTime())
            .withStatus(BucketLifecycleConfiguration.
                ENABLED.toString())
           );
// Save configuration.
s3Client.setBucketLifecycleConfiguration(bucketName, configuration);
```

Example プログラムリスト

次の Java コード例は、バケットに対してライフサイクル設定を追加、更新、および削除する完全なコードのリストです。コードを更新し、ライフサイクル設定例の追加先となるバケット名を指定する必要があります。

作業サンプルを作成およびテストする方法については、「Java コード例のテスト (p. 477)」を参照してください。

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;
import java.util.TimeZone;

import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.AmazonS3Exception;
import com.amazonaws.services.s3.model.BucketLifecycleConfiguration;
import com.amazonaws.services.s3.model.BucketLifecycleConfiguration.Transition;
import com.amazonaws.services.s3.model.StorageClass;

public class S3LifecycleExample {
    public static String bucketName = "*** Provide bucket name ***";
    public static AmazonS3Client s3Client;

    public static void main(String[] args) throws IOException {
        s3Client = new AmazonS3Client(new PropertiesCredentials(
                S3LifecycleExample.class.getResourceAsStream(
                        "AwsCredentials.properties")));
        try {

            BucketLifecycleConfiguration.Rule rule1 =
            new BucketLifecycleConfiguration.Rule()
            .withId("Archive immediately rule")
            .withPrefix("glacierobjects/")
            .withTransition(new Transition()
                                .withDays(0)
                                .withStorageClass(StorageClass.Glacier))
            .withStatus(BucketLifecycleConfiguration.ENABLED.toString());

            BucketLifecycleConfiguration.Rule rule2 =
                new BucketLifecycleConfiguration.Rule()
                .withId("Archive and then delete rule")
                .withPrefix("projectdocs/")
                .withTransition(new Transition()
                                    .withDays(365)
                                    .withStorageClass(StorageClass.Glacier))
                .withExpirationInDays(3650)
                .withStatus(BucketLifecycleConfiguration.ENABLED.toString());

            List<BucketLifecycleConfiguration.Rule> rules =
            new ArrayList<BucketLifecycleConfiguration.Rule>();
            rules.add(rule1);
            rules.add(rule2);

            BucketLifecycleConfiguration configuration =
            new BucketLifecycleConfiguration()
```

```
            .withRules(rules);

            // Save configuration.
          s3Client.setBucketLifecycleConfiguration(bucketName, configuration);


            // Retrieve configuration.
          configuration = s3Client.getBucketLifecycleConfiguration(bucketName);


            // Add a new rule.
            Calendar c = Calendar.getInstance(TimeZone.getTimeZone("GMT"));
            c.add(Calendar.YEAR, 10);
            c.set(Calendar.HOUR_OF_DAY, 0);
            c.set(Calendar.MINUTE, 0);
            c.set(Calendar.SECOND, 0);
            c.set(Calendar.MILLISECOND, 0);
            configuration.getRules().add(
                    new BucketLifecycleConfiguration.Rule()
                        .withId("NewRule")
                        .withPrefix("YearlyDocuments/")
                        .withExpirationDate(c.getTime())
                        .withStatus(BucketLifecycleConfiguration.
                            ENABLED.toString())
                       );
            // Save configuration.
          s3Client.setBucketLifecycleConfiguration(bucketName, configuration);


            // Retrieve configuration.
          configuration = s3Client.getBucketLifecycleConfiguration(bucketName);


            // Verify there are now three rules.
          configuration = s3Client.getBucketLifecycleConfiguration(bucketName);

            System.out.format("Expected # of rules = 3; found: %s\n",
                configuration.getRules().size());

            // Delete configuration.
            s3Client.deleteBucketLifecycleConfiguration(bucketName);

            // Retrieve nonexistent configuration.
          configuration = s3Client.getBucketLifecycleConfiguration(bucketName);

            String s = (configuration == null) ? "No configuration found." :
"Configuration found.";
            System.out.println(s);


        } catch (AmazonS3Exception amazonS3Exception) {
            System.out.format("An Amazon S3 error occurred. Exception: %s",
amazonS3Exception.toString());
        } catch (Exception ex) {
            System.out.format("Exception: %s", ex.toString());
        }
    }
}
```

AWS SDK for .NET を使用したオブジェクトのライフサイクルの管理

AWS SDK for .NET を使用して、バケットのライフサイクル設定を管理できます。ライフサイクル設定の管理の詳細については、「オブジェクトのライフサイクル管理 (p. 114)」を参照してください。

ライフサイクル設定をバケットに追加

1. `LifeCycleConfiguration` クラスのインスタンスを作成し、`LifecycleRules` のリストを指定します。
2. `PutLifeCycleConfigurationRequest` オブジェクトを作成します。

   バケット名と、前述のステップで作成した `LifeCycleConfiguration` インスタンスを指定する必要があります。
3. `client.PutLifecycleConfiguration` を実行します。

以下の C# コードスニペットは、前述のタスクの例です。ライフサイクル設定では 1 つのルール (「アーカイブおよび削除ルール」) を定義し、その中でキープレフィックス「projectdocs/」の付いたすべてのオブジェクトに対して Amazon S3 が実行する 2 つのアクションを指定しています。

```
string bucketName = "examplebucket";

// Add a sample configuration
using (client = new AmazonS3Client()){
  var lifeCycleConfiguration = new LifecycleConfiguration()
  {
    Rules = new List<LifecycleRule>
    {
        new LifecycleRule
        {
             Id = "Archive and delete rule",
             Prefix = "projectdocs/",
             Status = LifecycleRuleStatus.Enabled,
              Transition = new LifecycleTransition()
              {
                   Days = 365,
                   StorageClass = S3StorageClass.Glacier
              },
              Expiration = new LifecycleRuleExpiration()
              {
                   Days = 3650
              }
        }
    }
  };

  PutLifecycleConfigurationRequest request = new PutLifecycleConfigurationRequest

  {
     BucketName = bucketName,
     Configuration = configuration
  };

  var response = client.PutLifecycleConfiguration(request);
}
```

既存のライフサイクル設定を更新するには

- ライフサイクル設定をバケットに追加すると、既存のライフサイクル設定が置き換わります。既存のライフサイクル設定を更新するには、次の C# コードスニペットに示すように、まず既存のライフサイクル設定を取得し、変更してから、修正済みのライフサイクル設定をバケットに追加する必要があります。

このスニペットでは、既存の設定を取得して、ID が NewRule の新しいルールを追加しています。

```
// Retrieve lifecycle configuration.
GetLifecycleConfigurationRequest request = new GetLifecycleConfigurationRequest
{
    BucketName = bucketName
};
var response = client.GetLifecycleConfiguration(request);
var configuration = response.Configuration;

// Add new rule.
configuration.Rules.Add(new LifecycleRule
{
    Id = "NewRule",
    Prefix = "YearlyDocuments/",
   Expiration = new LifecycleRuleExpiration { Date = DateTime.Now.AddYears(10)
 }
});

PutLifecycleConfigurationRequest request = new PutLifecycleConfigurationRequest
{
   BucketName = bucketName,
   Configuration = configuration
};

var response = client.PutLifecycleConfiguration(request);
```

Example プログラムリスト

次の C# コード例は、バケットに対してライフサイクル設定を追加、更新、および削除する完全なコードのリストです。コードを更新し、ライフサイクル設定例の追加先となるバケット名を指定する必要があります。

作業サンプルを作成およびテストする方法については、『Using the AWS SDK for .NET』の「Testing the .NET Code Examples」を参照してください。

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using Amazon.S3;
using Amazon.S3.Model;

namespace aws.amazon.com.s3.documentation
{
    class S3Sample
    {
        static string bucketName = "*** Provide bucket name ***";

        static AmazonS3 client;

        public static void Main(string[] args)
        {
            try
            {
                AmazonS3Config s3Config = new AmazonS3Config();
                s3Config.ServiceURL = "s3.aws-master.amazon.com";
                using (client = new AmazonS3Client(s3Config))
                {
                    var lifeCycleConfiguration = new LifecycleConfiguration()
                    {
                        Rules = new List<LifecycleRule>
                        {
                            new LifecycleRule
                            {
                                 Id = "Archive immediately rule",
                                 Prefix = "glacierobjects/",
                                 Status = LifecycleRuleStatus.Enabled,
                                  Transition = new LifecycleTransition()
                                  {
                                       Days = 0,
                                       StorageClass = S3StorageClass.Glacier
                                  }
                            },
                            new LifecycleRule
                            {
                                 Id = "Archive and then delete rule",
                                 Prefix = "projectdocs/",
                                 Status = LifecycleRuleStatus.Enabled,
                                  Transition = new LifecycleTransition()
                                  {
                                       Days = 365,
                                       StorageClass = S3StorageClass.Glacier
                                  },
                                  Expiration = new LifecycleRuleExpiration()
                                  {
```

```
                                    Days = 3650
                                }
                            }
                        }
                    };

                    // Add the configuration to the bucket
                    PutLifeCycleConfiguration(lifeCycleConfiguration);

                    // Retrieve an existing configuration
                    lifeCycleConfiguration = GetLifeCycleConfiguration();

                    // Add a new rule.
                    lifeCycleConfiguration.Rules.Add(new LifecycleRule
                    {
                        Id = "NewRule",
                        Prefix = "YearlyDocuments/",
                       Expiration = new LifecycleRuleExpiration { Date = Date
Time.Now.AddYears(10) }
                    });

                    // Add the configuration to the bucket
                    PutLifeCycleConfiguration(lifeCycleConfiguration);

                    // Verify that there are now three rules
                    lifeCycleConfiguration = GetLifeCycleConfiguration();
                   Console.WriteLine("Expected # of rulest=3; found:{0}", li
feCycleConfiguration.Rules.Count);

                    // Delete the configuration
                    DeleteLifecycleConfiguration();

                    // Retrieve a nonexistent configuration
                    lifeCycleConfiguration = GetLifeCycleConfiguration();
                    Debug.Assert(lifeCycleConfiguration == null);
                }

                Console.WriteLine("Example complete. To continue, click
Enter...");
                Console.ReadKey();
            }
            catch (AmazonS3Exception amazonS3Exception)
            {
                Console.WriteLine("S3 error occurred. Exception: " +
amazonS3Exception.ToString());
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception: " + e.ToString());
            }
        }

       static void PutLifeCycleConfiguration(LifecycleConfiguration configura
tion)
        {

            PutLifecycleConfigurationRequest request = new PutLifecycleConfig
urationRequest
```

```
            {
                BucketName = bucketName,
                Configuration = configuration
            };

            var response = client.PutLifecycleConfiguration(request);
        }

        static LifecycleConfiguration GetLifeCycleConfiguration()
        {
            GetLifecycleConfigurationRequest request = new GetLifecycleConfig
urationRequest
            {
                BucketName = bucketName

            };
            var response = client.GetLifecycleConfiguration(request);
            var configuration = response.Configuration;
            return configuration;
        }

        static void DeleteLifecycleConfiguration()
        {
          DeleteLifecycleConfigurationRequest request = new DeleteLifecycleCon
figurationRequest
            {
                BucketName = bucketName
            };
            client.DeleteLifecycleConfiguration(request);
        }
    }
}
```

REST API を使用したオブジェクトのライフサイクルの管理

AWS マネジメントコンソールを使用して、バケットにライフサイクルを設定できます。必要に応じて、REST リクエストを直接送信することもできます。『*Amazon Simple Storage Service API リファレンス*』の以下のセクションでは、ライフサイクル設定に関連する REST API について説明します。

- PUT Bucket lifecycle
- GET Bucket lifecycle
- DELETE Bucket lifecycle

Cross-Origin Resource Sharing の有効化

Topics

Cross-Origin Resource Sharing (CORS) は、特定のドメインにロードされたクライアントウェブアプリケーションが異なるドメイン内のリソースと通信する方法を定義します。Amazon S3 で CORS がサポートされるようになったことで、Amazon S3 で機能豊富なクライアント側ウェブアプリケーションを構築し、Amazon S3 リソースに対するクロスオリジンアクセスを選択的に許可することができるようになりました。

Cross-Origin Resource Sharing: 例

Cross-Origin Resource Sharing によって、使用事例がいくつか可能になります。例えば、「Amazon S3 での静的ウェブサイトのホスティング (p. 411)」で説明するように、website という Amazon S3 バケットでウェブサイトをホストしているとします。ユーザーはウェブサイトエンドポイント http://website.s3-website-us-east-1.amazonaws.com をロードします。ここで、このバケットに保存されているウェブページで、JavaScript を使用して、バケットの Amazon S3 の API エンドポイント website.s3.amazonaws.com を使用して、同じバケットに対して、認証済みの GET および PUT リクエストを行うことができるようにすることを考えます。ブラウザは通常、それらのリクエストを許可しないように、JavaScript をブロックしますが、CORS を使用することにより、website.s3-website-us-east-1.amazonaws.com からのクロスオリジンリクエストを明示的に有効にするようにバケットを設定できます。

別の例として、S3バケットからのウェブフォントをホストしたいとします。ここでも、ブラウザはウェブフォントをロードするために CORS チェック (プリフライトチェックとも呼ばれる) を要求するため、すべてのオリジンがこれらのリクエストを行うことができるように、ウェブフォントをホストするバケットを設定します。

バケットで CORS を有効にする方法

クロスオリジンリクエストを許可するようにバケットを設定するには、CORS 設定を作成します。これは、バケットへのアクセスを許可するオリジン、各オリジンをサポートするオペレーション (HTTP メソッド) 、およびその他のオペレーション固有の情報を識別するルールを含む XML ドキュメントです。設定には、最大 100 のルールを追加できます。その XML ドキュメントを cors サブリソースとしてバケットに追加します。

例えば、次のバケットへの cors 設定には、2 つのルールがあり、CORSRule 要素として指定されています。

- 最初のルールは、https://www.example.com オリジンからのクロスオリジン PUT、POST、および DELETE リクエストを許可します。このルールは、Access-Control-Request-Headers ヘッダーによって、プリフライト OPTIONS リクエスト内のすべてのヘッダーも許可します。プリフライト OPTIONS リクエストへのレスポンスとして、Amazon S3 はリクエストされたヘッダーを返します。
- 2 つ目のルールは、すべてのオリジンからのクロスオリジン GET リクエストを許可します。ワイルドカード文字「*」は、すべてのオリジンを表します。

```
<CORSConfiguration>
 <CORSRule>
   <AllowedOrigin>http://www.example.com</AllowedOrigin>

   <AllowedMethod>PUT</AllowedMethod>
   <AllowedMethod>POST</AllowedMethod>
   <AllowedMethod>DELETE</AllowedMethod>
```

```
    <AllowedHeader>*</AllowedHeader>
  </CORSRule>
  <CORSRule>
    <AllowedOrigin>*</AllowedOrigin>
    <AllowedMethod>GET</AllowedMethod>
  </CORSRule>
</CORSConfiguration>
```

次の CORS 設定に示すように、CORS 設定ではオプションの設定パラメータも使用できます。この例で、次の CORS 設定は、`http://www.example.com` オリジンからのクロスオリジン PUT および POST リクエストを許可します。

```
<CORSConfiguration>
  <CORSRule>
    <AllowedOrigin>http://www.example.com</AllowedOrigin>
    <AllowedMethod>PUT</AllowedMethod>
    <AllowedMethod>POST</AllowedMethod>
    <AllowedMethod>DELETE</AllowedMethod>
    <AllowedHeader>*</AllowedHeader>
   <MaxAgeSeconds>3000</MaxAgeSeconds>
   <ExposeHeader>x-amz-server-side-encryption</ExposeHeader>
   <ExposeHeader>x-amz-request-id</ExposeHeader>
   <ExposeHeader>x-amz-id-2</ExposeHeader>
  </CORSRule>
</CORSConfiguration>
```

前述の設定の `CORSRule` 要素には、次のオプションの要素が含まれます。

- `MaxAgeSeconds` – 指定したリソースのプリフライト OPTIONS リクエストへの Amazon S3 レスポンスをブラウザでキャッシュする時間を秒単位で指定します (この例では 3,000)。レスポンスのキャッシュにより、元のリクエストが繰り返された場合に、ブラウザは Amazon S3 にプリフライトリクエストを送信する必要がありません。
- `ExposeHeader` – 顧客がアプリケーション (JavaScript `XMLHttpRequest` オブジェクトなど) からアクセスできるレスポンスヘッダーを識別します (この例では `x-amz-server-side-encryption`、`x-amz-request-id` および `x-amz-id-2`)。

AllowedMethod 要素

CORS 設定では、`AllowedMethod` 要素に次の値を指定できます。

- GET
- PUT
- POST
- DELETE
- HEAD

AllowedOrigin 要素

`AllowedOrigin` 要素に、クロスドメインリクエストを許可するオリジンを指定します (`http://www.example.com` など)。オリジン文字列には、最大 1 つのワイルドカード文字 (`*`) を含めることができます。例えば、`http://*.example.com` などです。オプションで、オリジンに `*` を

指定して、すべてのオリジンでクロスオリジンリクエストを送信できるようにすることができます。さらに、https を指定して、セキュリティで保護されたオリジンのみを有効にすることもできます。

AllowedHeader 要素

AllowedHeader 要素は、Access-Control-Request-Headers ヘッダーによって、プリフライトリクエストで許可されるヘッダーを指定します。Access-Control-Request-Headers ヘッダー内の各ヘッダー名は、ルールの対応するエントリに一致する必要があります。Amazon S3 は、リクエストされたヘッダーのうち許可されたヘッダーのみをレスポンスに入れて送信します。Amazon S3 へのリクエストで使用できるヘッダーのサンプルリストについては、『*Amazon Simple Storage Service API リファレンス*』の「Common Request Headers」を参照してください。

ルール内の各 AllowedHeader 文字列には、最大 1 つのワイルドカード文字 (*) を含めることができます。例えば、<AllowedHeader>x-amz-*</AllowedHeader> は Amazon 固有のすべてのヘッダーを有効にします。

ExposeHeader 要素

各 ExposeHeader 要素は、顧客がアプリケーションから (例えば、JavasScript XMLHttpRequest オブジェクトから) アクセスできるようにするレスポンス内のヘッダーを識別します。一般的な Amazon S3 レスポンスヘッダーのリストについては、『*Amazon Simple Storage Service API リファレンス*』の「Common Response Headers」を参照してください。

MaxAgeSeconds 要素

MaxAgeSeconds 要素は、リソース、HTTP メソッド、およびオリジンによって識別されたプリフライトリクエストのレスポンスをブラウザでキャッシュできる時間を秒単位で指定します。

Amazon S3 がバケットの CORS 設定を評価する方法

Amazon S3 は、ブラウザからプリフライトリクエストを受け取ると、バケットの CORS 設定を評価し、受信ブラウザリクエストに一致する最初の CORSRule ルールを使用して、クロスオリジンリクエストを有効にします。ルールが一致するには、次の条件を満たしている必要があります。

- リクエストの Origin ヘッダーが AllowedOrigin 要素と一致する必要があります。
- リクエストメソッド (GET や PUT など) または Access-Control-Request-Method ヘッダー (プリフライト OPTIONS リクエストの場合) が AllowedMethod 要素のいずれかである必要があります。
- プリフライトリクエストのリクエストの Access-Control-Request-Headers ヘッダーにリストされているすべてのヘッダーが AllowedHeader 要素と一致する必要があります。

Note

バケットの CORS 設定を有効にすると、ACL とポリシーが引き続き適用されます。

AWS Management Console を使用した Cross-Origin Resource Sharing (CORS) の有効化

AWS Management Console を使用して、バケットで CORS 設定を実施できます。詳細な手順については、『*Amazon S3 Console User Guide*』の「Editing Bucket Permissions」を参照してください。

AWS SDK for Java を使用した Cross-Origin Resource Sharing（CORS）の有効化

AWS SDK for Java を使用して、バケットの Cross-Origin Resource Sharing（CORS）を管理できます。CORS の詳細については、「Cross-Origin Resource Sharing の有効化 (p. 133)」を参照してください。

このセクションでは、次のタスクのサンプルコードスニペットと、すべてのタスクを実演する完全なプログラム例を示します。

- Amazon S3 クライアントクラスのインスタンスの作成
- CORS 設定の作成とバケットへの追加
- 既存の CORS 設定の更新

Cross-Origin Resource Sharing のメソッド

AmazonS3Client()	AwsCredentials.properties ファイルに定義されている認証情報で `AmazonS3Client` オブジェクトを作成します。
setBucketCrossOriginConfiguration()	バケットに適用する CORS 設定を設定します。指定されたバケットに設定が既に存在する場合、既存の設定は新しい設定で置き換えられます。
getBucketCrossOriginConfiguration()	指定されたバケットの CORS 設定を取得します。バケットに設定がない場合、レスポンスの `Configuration` ヘッダーは null になります。
deleteBucketCrossOriginConfiguration()	指定されたバケットの CORS 設定を削除します。

AWS SDK for Java API の詳細については、『AWS SDK for Java API Reference 』を参照してください。

Amazon S3 クライアントクラスのインスタンスの作成

次のコードスニペットでは、`CORS_JavaSDK` というクラスの新しい `AmazonS3Client` インスタンスを作成しています。この例では、AwsCredentials.properties ファイルから `accessKey` と `secretKey` の値を取得します。

```
AmazonS3Client client;
AWSCredentials credentials = new PropertiesCredentials(
     CORS_JavaSDK.class.getResourceAsStream("AwsCredentials.properties"));

client = new AmazonS3Client(credentials);
```

CORS 設定の作成とバケットへの追加

CORS 設定をバケットに追加するには。

1. ルールを記述する `CORSRule` オブジェクトを作成します。
2. `BucketCrossOriginConfiguration` オブジェクトを作成し、ルールを設定オブジェクトに追加します。
3. `client.setBucketCrossOriginConfiguration` メソッドを呼び出して、CORS 設定をバケットに追加します。

次のコードス二ペットでは、CORSRule1 と CORSRule2 の 2 つのルールを作成し、各ルールを rules 配列に追加しています。rules 配列を使用して、ルールをバケット bucketName に追加します。

```
// Add a sample configuration
BucketCrossOriginConfiguration configuration = new BucketCrossOriginConfigura
tion();

List<CORSRule> rules = new ArrayList<CORSRule>();

CORSRule rule1 = new CORSRule()
    .withId("CORSRule1")
    .withAllowedMethods(Arrays.asList(new CORSRule.AllowedMethods[] {
            CORSRule.AllowedMethods.PUT, CORSRule.AllowedMethods.POST, COR
SRule.AllowedMethods.DELETE}))
    .withAllowedOrigins(Arrays.asList(new String[] {"http://*.example.com"}));

CORSRule rule2 = new CORSRule()
.withId("CORSRule2")
.withAllowedMethods(Arrays.asList(new CORSRule.AllowedMethods[] {
        CORSRule.AllowedMethods.GET}))
.withAllowedOrigins(Arrays.asList(new String[] {"*"}))
.withMaxAgeSeconds(3000)
.withExposedHeaders(Arrays.asList(new String[] {"x-amz-server-side-encryp
tion"}));

configuration.setRules(Arrays.asList(new CORSRule[] {rule1, rule2}));

// Save the configuration
client.setBucketCrossOriginConfiguration(bucketName, configuration);
```

既存の CORS 設定の更新

既存の CORS 設定を更新するには

1. client.getBucketCrossOriginConfiguration メソッドを呼び出して、CORS 設定を取得します。
2. ルールのリストにルールを追加または削除して、設定情報を更新します。
3. client.getBucketCrossOriginConfiguration メソッドを呼び出して、設定をバケットに追加します。

次のコードス二ペットでは、既存の設定を取得して、ID が NewRule の新しいルールを追加しています。

```
// Get configuration.
BucketCrossOriginConfiguration configuration = client.getBucketCrossOriginCon
figuration(bucketName);

// Add new rule.
CORSRule rule3 = new CORSRule()
.withId("CORSRule3")
.withAllowedMethods(Arrays.asList(new CORSRule.AllowedMethods[] {
        CORSRule.AllowedMethods.HEAD}))
.withAllowedOrigins(Arrays.asList(new String[] {"http://www.example.com"}));

List<CORSRule> rules = configuration.getRules();
rules.add(rule3);
```

```
configuration.setRules(rules);

// Save configuration.
client.setBucketCrossOriginConfiguration(bucketName, configuration);
```

Example プログラムリスト

次の Java プログラムには、前述のタスクが組み込まれています。

作業サンプルの作成およびテストについては、「Java コード例のテスト (p. 477)」を参照してください。

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.BucketCrossOriginConfiguration;
import com.amazonaws.services.s3.model.CORSRule;

public class CORS_JavaSDK {

    /**
     * @param args
     * @throws IOException
     */
    public static AmazonS3Client client;
    public static String bucketName = "***provide bucket name***";

    public static void main(String[] args) throws IOException {

        AWSCredentials credentials = new PropertiesCredentials(
                CORS_JavaSDK.class
                        .getResourceAsStream("AwsCredentials.properties"));

        client = new AmazonS3Client(credentials);

        // Create a new configuration request and add two rules
       BucketCrossOriginConfiguration configuration = new BucketCrossOriginCon
figuration();

        List<CORSRule> rules = new ArrayList<CORSRule>();

        CORSRule rule1 = new CORSRule()
            .withId("CORSRule1")
            .withAllowedMethods(Arrays.asList(new CORSRule.AllowedMethods[] {

                    CORSRule.AllowedMethods.PUT, CORSRule.AllowedMethods.POST,
 CORSRule.AllowedMethods.DELETE}))
            .withAllowedOrigins(Arrays.asList(new String[] {"http://*.ex
ample.com"}));

        CORSRule rule2 = new CORSRule()
        .withId("CORSRule2")
        .withAllowedMethods(Arrays.asList(new CORSRule.AllowedMethods[] {
                CORSRule.AllowedMethods.GET}))
        .withAllowedOrigins(Arrays.asList(new String[] {"*"}))
        .withMaxAgeSeconds(3000)
        .withExposedHeaders(Arrays.asList(new String[] {"x-amz-server-side-en
cryption"}));
```

```
        configuration.setRules(Arrays.asList(new CORSRule[] {rule1, rule2}));

         // Add the configuration to the bucket.
        client.setBucketCrossOriginConfiguration(bucketName, configuration);

        // Retrieve an existing configuration.
        configuration = client.getBucketCrossOriginConfiguration(bucketName);
        printCORSConfiguration(configuration);

        // Add a new rule.
        CORSRule rule3 = new CORSRule()
        .withId("CORSRule3")
        .withAllowedMethods(Arrays.asList(new CORSRule.AllowedMethods[] {
                CORSRule.AllowedMethods.HEAD}))
        .withAllowedOrigins(Arrays.asList(new String[] {"http://www.ex
ample.com"}));

        rules = configuration.getRules();
        rules.add(rule3);
        configuration.setRules(rules);
        client.setBucketCrossOriginConfiguration(bucketName, configuration);
        System.out.format("Added another rule: %s\n", rule3.getId());

        // Verify that the new rule was added.
        configuration = client.getBucketCrossOriginConfiguration(bucketName);
        System.out.format("Expected # of rules = 3, found %s", configura
tion.getRules().size());

        // Delete the configuration.
        client.deleteBucketCrossOriginConfiguration(bucketName);

        // Try to retrieve configuration.
        configuration = client.getBucketCrossOriginConfiguration(bucketName);
        System.out.println("\nRemoved CORS configuration.");
        printCORSConfiguration(configuration);
    }

   static void printCORSConfiguration(BucketCrossOriginConfiguration configur
ation)
    {

        if (configuration == null)
        {
            System.out.println("\nConfiguration is null.");
            return;
        }

        System.out.format("\nConfiguration has %s rules:\n", configura
tion.getRules().size());
        for (CORSRule rule : configuration.getRules())
        {
            System.out.format("Rule ID: %s\n", rule.getId());
            System.out.format("MaxAgeSeconds: %s\n", rule.getMaxAgeSeconds());

            System.out.format("AllowedMethod: %s\n", rule.getAllowedMeth
ods().toArray());
          System.out.format("AllowedOrigins: %s\n", rule.getAllowedOrigins());
```

```
            System.out.format("AllowedHeaders: %s\n", rule.getAllowedHeaders());

              System.out.format("ExposeHeader: %s\n", rule.getExposedHeaders());

          }
      }
}
```

AWS SDK for .NET を使用した Cross-Origin Resource Sharing（CORS）の有効化

AWS SDK for .NET を使用して、バケットの Cross-Origin Resource Sharing（CORS）を管理できます。CORS の詳細については、「Cross-Origin Resource Sharing の有効化 (p. 133)」を参照してください。

このセクションでは、次の表でタスクのサンプルコードスニペットの一部を示します（完全なサンプルプログラムは後述）。

Cross-Origin Resource Sharing の管理

1	`AmazonS3Client` クラスのインスタンスを作成します。
2	新しい CORS 設定を作成します。
3	既存の CORS 設定を取得して変更します。
4	設定をバケットに追加します。

Cross-Origin Resource Sharing のメソッド

AmazonS3Client()	App.config ファイルで定義されている認証情報で `AmazonS3Client` を作成します。
PutCORSConfiguration()	バケットに適用する CORS 設定を設定します。指定されたバケットに設定が既に存在する場合、既存の設定は新しい設定で置き換えられます。
GetCORSConfiguration()	指定されたバケットの CORS 設定を取得します。バケットに設定がない場合、レスポンスの `Configuration` ヘッダーは null になります。
DeleteCORSConfiguration()	指定されたバケットの CORS 設定を削除します。

AWS SDK for .NET API の詳細については、「AWS SDK for .NET API Reference」を参照してください。

Amazon S3 クラスのインスタンスの作成

次のコードスニペットでは、`CreateAmazonS3Client` メソッドを使用して、`AmazonS3Client` クラスのインスタンスを作成しています。この例では、app.config から `accessKey` と `secretKey` の値を取得します。

```
NameValueCollection appConfig = ConfigurationManager.AppSettings;

string accessKey = appConfig["AWSAccessKey"];
string secretKey = appConfig["AWSSecretKey"];
```

```
try
{
using (client = Amazon.AWSClientFactory.CreateAmazonS3Client(
    accessKey, secretKey ))
```

バケットへの CORS 設定の追加

CORS 設定をバケットに追加するには。

1. ルールを記述する CORSConfiguration オブジェクトを作成します。
2. バケット名と CORS 設定を指定する PutCORSConfigurationRequest オブジェクトを作成します。
3. client.PutCORSConfiguration を呼び出して、CORS 設定をバケットに追加します。

次のコードス二ペットでは、CORSRule1 と CORSRule2 の 2 つのルールを作成し、各ルールを rules 配列に追加しています。rules 配列を使用して、ルールをバケット bucketName に追加します。

```
// Add a sample configuration
CORSConfiguration configuration = new CORSConfiguration
{
    Rules = new System.Collections.Generic.List<CORSRule>
    {
        new CORSRule
        {
            Id = "CORSRule1",
            AllowedMethods = new List<string> {"PUT", "POST", "DELETE"},
            AllowedOrigins = new List<string> {"http://*.example.com"}
        },
        new CORSRule
        {
            Id = "CORSRule2",
            AllowedMethods = new List<string> {"GET"},
            AllowedOrigins = new List<string> {"*"},
            MaxAgeSeconds = 3000,
            ExposeHeaders = new List<string> {"x-amz-server-side-encryption"}
        }
    }
};

// Save the configuration
PutCORSConfiguration(configuration);

static void PutCORSConfiguration(CORSConfiguration configuration)
{

    PutCORSConfigurationRequest request = new PutCORSConfigurationRequest
    {
        BucketName = bucketName,
        Configuration = configuration
    };

    var response = client.PutCORSConfiguration(request);
}
```

既存の CORS 設定の更新

既存の CORS 設定を更新するには

1. `client.GetCORSConfiguration` メソッドを呼び出して、CORS 設定を取得します。
2. ルールを追加または削除して、設定情報を更新します。
3. `client.PutCORSConfiguration` メソッドを呼び出して、設定をバケットに追加します。

次のコードス二ペットでは、既存の設定を取得して、ID が `NewRule` の新しいルールを追加しています。

```
// Get configuration.
configuration = GetCORSConfiguration();
// Add new rule.
configuration.Rules.Add(new CORSRule
{
    Id = "NewRule",
    AllowedMethods = new List<string> { "HEAD" },
    AllowedOrigins = new List<string> { "http://www.example.com" }
});

// Save configuration.
PutCORSConfiguration(configuration);
```

Example プログラムリスト

次の C# プログラムには、前述のタスクが組み込まれています。

作業サンプルの作成およびテストについては、「.NET コード例のテスト (p. 478)」を参照してください。

```
using System;
using System.Configuration;
using System.Collections.Specialized;
using System.Net;
using Amazon.S3;
using Amazon.S3.Model;
using Amazon.S3.Util;
using System.Diagnostics;
using System.Collections.Generic;

namespace CORS_DotNetSDK
{
    class S3Sample
    {
        static string bucketName = "examplebucket";

        static AmazonS3 client;

        public static void Main(string[] args)
        {
           /* Create an instance of the AmazonS3 class using CreateAmazonS3Cli
ent().
            ** CreateAmazonS3Client takes the access key and secret key as
parameters.
            ** This example uses the values in app.Config for security. */

            NameValueCollection appConfig =
              ConfigurationManager.AppSettings;

            string accessKey = appConfig["AWSAccessKey"];
            string secretKey = appConfig["AWSSecretKey"];
            try
            {
                using (client = Amazon.AWSClientFactory.CreateAmazonS3Client(
                  accessKey, secretKey))
                {
                    // Create a new configuration request and add two rules

                    CORSConfiguration configuration = new CORSConfiguration
                    {
                        Rules = new System.Collections.Generic.List<CORSRule>
                        {
                          new CORSRule
                          {
                            Id = "CORSRule1",
                            AllowedMethods = new List<string> {"PUT", "POST",
"DELETE"},
                            AllowedOrigins = new List<string> {"http://*.ex
ample.com"}
                          },
                          new CORSRule
```

```
                        {
                          Id = "CORSRule2",
                          AllowedMethods = new List<string> {"GET"},
                          AllowedOrigins = new List<string> {"*"},
                          MaxAgeSeconds = 3000,
                          ExposeHeaders = new List<string> {"x-amz-server-
side-encryption"}
                        }
                    }
                };

                // Add the configuration to the bucket
                PutCORSConfiguration(configuration);

                // Retrieve an existing configuration
                configuration = GetCORSConfiguration();

                // Add a new rule.
                configuration.Rules.Add(new CORSRule
                {
                    Id = "CORSRule3",
                    AllowedMethods = new List<string> { "HEAD" },
                    AllowedOrigins = new List<string> { "http://www.ex
ample.com" }
                });

                // Add the configuration to the bucket
                PutCORSConfiguration(configuration);

                // Verify that there are now three rules
                configuration = GetCORSConfiguration();
                Console.WriteLine("Expected # of rulest=3; found:{0}",
configuration.Rules.Count);

                // Delete the configuration
                DeleteCORSConfiguration();

                // Retrieve a nonexistent configuration
                configuration = GetCORSConfiguration();
                Debug.Assert(configuration == null);
            }

            Console.WriteLine("Example complete. To continue, click
Enter...");
            Console.ReadKey();
        }
        catch (AmazonS3Exception amazonS3Exception)
        {
            Console.WriteLine("S3 error occurred. Exception: " +
amazonS3Exception.ToString());
            Console.ReadKey();
        }
        catch (Exception e)
        {
            Console.WriteLine("Exception: " + e.ToString());
            Console.ReadKey();
        }
    }
```

```
        static void PutCORSConfiguration(CORSConfiguration configuration)
        {

          PutCORSConfigurationRequest request = new PutCORSConfigurationRequest

            {
                BucketName = bucketName,
                Configuration = configuration
            };

            var response = client.PutCORSConfiguration(request);
        }

        static CORSConfiguration GetCORSConfiguration()
        {
          GetCORSConfigurationRequest request = new GetCORSConfigurationRequest

            {
                BucketName = bucketName

            };
            var response = client.GetCORSConfiguration(request);
            var configuration = response.Configuration;
            PrintCORSRules(configuration);
            return configuration;
        }

        static void DeleteCORSConfiguration()
        {
            DeleteCORSConfigurationRequest request = new DeleteCORSConfigura
tionRequest
            {
                BucketName = bucketName
            };
            client.DeleteCORSConfiguration(request);
        }

        static void PrintCORSRules(CORSConfiguration configuration)
        {
            Console.WriteLine();

            if (configuration == null)
            {
                Console.WriteLine("\nConfiguration is null");
                return;
            }

            Console.WriteLine("Configuration has {0} rules:", configura
tion.Rules.Count);
            foreach (CORSRule rule in configuration.Rules)
            {
                Console.WriteLine("Rule ID: {0}", rule.Id);
                Console.WriteLine("MaxAgeSeconds: {0}", rule.MaxAgeSeconds);
                Console.WriteLine("AllowedMethod: {0}", string.Join(", ",
rule.AllowedMethods.ToArray()));
                Console.WriteLine("AllowedOrigins: {0}", string.Join(", ",
rule.AllowedOrigins.ToArray()));
```

```
                Console.WriteLine("AllowedHeaders: {0}", string.Join(", ",
rule.AllowedHeaders.ToArray()));
                Console.WriteLine("ExposeHeader: {0}", string.Join(", ",
rule.ExposeHeaders.ToArray()));
            }
        }
    }
}
```

REST API を使用した Cross-Origin Resource Sharing（CORS）の有効化

AWS Management Console を使用して、バケットで CORS 設定を実施できます。アプリケーションの必要に応じて、REST リクエストを直接送信することもできます。『*Amazon Simple Storage Service API リファレンス*』の以下のセクションでは、CORS 設定に関連する REST API アクションについて説明しています。

- PUT Bucket cors
- GET Bucket cors
- DELETE Bucket cors
- OPTIONS オブジェクト

CORS の問題のトラブルシューティング

CORS 設定を行ったバケットにアクセスした際、予期しない動作が発生する場合には、トラブルシューティングのために以下のアクションを実行することができます。

1. バケットに CORS 設定が行われていることを確認します。

   手順については、『*Amazon Simple Storage Service Console User Guide*』の「Editing Bucket Permissions」を参照してください。CORS 設定を行うと、コンソールにより、[Properties] バケットの [Permissions] セクションに [Edit CORS Configuration] リンクが表示されます。
2. 任意のツールを使用してリクエストとレスポンスの全体を取り込みます。Amazon S3 が受信するすべてのリクエストには、リクエストのデータと一致する、以下のような CORS ルールが 1 つ存在している必要があります。
   a. リクエストに Origin ヘッダーがあること。

      ヘッダーがないリクエストは、Amazon S3 でクロスオリジンリクエストとして扱われず、CORS レスポンスヘッダーをレスポンスに入れて返送されることもありません。
   b. リクエストの Origin ヘッダーが特定の `CORSRule` の `AllowedOrigin` 要素の少なくとも 1 つと一致していること。

      リクエスト の Origin ヘッダーのスキーム、ホスト、およびポートの値が `CORSRule` の `AllowedOrigin` と一致していること。例えば、オリジン `http://www.example.com` を許可するように `CORSRule` を設定した場合は、リクエスト内に `https://www.example.com` や `http://www.example.com:80` のオリジンがあると、設定で許可されているオリジンと一致しなくなります。
   c. リクエスト内の Method（または、プリフライトリクエストの場合は `Access-Control-Request-Method` に指定したメソッド）が、同じ `CORSRule` 内の `AllowedMethod` 要素の 1 つであること。

d. プリフライトリクエストの場合、それに `Access-Control-Request-Headers` ヘッダーが含まれているときに、`CORSRule` に `Access-Control-Request-Headers` ヘッダーのすべての値に対する `AllowedHeader` エントリが含まれていること。

オブジェクトに対するオペレーション

Amazon S3 では、オブジェクトを格納、取得、および削除できます。オブジェクトは、全体またはその一部の取得が可能です。バケットに対してバージョニングを有効にした場合は、特定バージョンのオブジェクトを取得できます。また、オブジェクトに関連付けられているサブリソースを取得し、必要に応じて更新することもできます。既存のオブジェクトについてはコピーを作成できます。オブジェクトのアップロードまたはコピーについては、オブジェクトのサイズに応じて次のような注意事項があります。

- オブジェクトのアップロード – 1 回のオペレーションでアップロードできるオブジェクトのサイズは最大 5 GB です。5 GB を超えるオブジェクトの場合は、マルチパートアップロード API を使用する必要があります。
  マルチパートアップロード API を使用すると、1 つで最大 5 TB のオブジェクトをアップロードできます。詳細については、「マルチパートアップロード API を使用したオブジェクトのアップロード (p. 178)」を参照してください。
- オブジェクトのコピー – コピーオペレーションは、Amazon S3 内に既に格納されているオブジェクトのコピーを作成することです。
  1 回のアトミックオペレーションでコピーできるオブジェクトのサイズは最大 5 GB です。5 GB を超えるオブジェクトをコピーする場合は、マルチパートアップロード API を使用する必要があります。詳細については、「オブジェクトのコピー (p. 222)」を参照してください。

オブジェクトを操作したり、次の AWS SDK ライブラリのいずれかを使用したりするには、REST API (「REST API を使用したリクエストの実行 (p. 49)」を参照) を使用します。

- AWS SDK for Java
- AWS SDK for .NET
- AWS SDK for PHP

これらのライブラリは、オブジェクトの操作を容易にする、高レベルの抽象化を実現します。ただし、アプリケーションの必要に応じて REST API を直接使用することもできます。

マルチパートアップロードの概要

Topics

マルチパートアップロード API を使用すると、大容量オブジェクトをいくつかに分けてアップロードできます。この API では、新しい大容量オブジェクトをアップロードしたり、既存オブジェクトのコピーを作成したりできます (「オブジェクトに対するオペレーション (p. 149)」を参照)。

マルチパートアップロードは 3 つのステップで構成されるプロセスです。まずアップロードを開始し、次にオブジェクトのパートをアップロードします。すべてのパートをアップロードしたらマルチパートアップロードを完了します。Amazon S3 の側では、マルチパートアップロードの完了リクエストを受

け取ると同時に、アップロードされたパートからオブジェクトを構築します。構築されたオブジェクトは、バケット内の他のオブジェクトと同じようにアクセスできます。

進行中のすべてのマルチパートアップロードをリストしたり、特定のマルチパートアップロードにおいてアップロードが完了したパートのリストを取得したりできます。このようなオペレーションについて、それぞれこのセクションで説明します。

Multipart Upload Initiation

リクエストを送信すると、アップロード ID を含むレスポンスが Amazon S3 から返されます。アップロード ID はマルチパートアップロードの一意の識別子です。パートのアップロード、パートのリスト、アップロードの完了、アップロードの中止を行うときは常に、このアップロード ID を指定する必要があります。アップロードするオブジェクトの説明となるメタデータを指定する場合は、マルチパートアップロードの開始リクエストの中にそれを指定する必要があります。

パートのアップロード

パートをアップロードするときは、アップロード ID に加えて、パート番号を指定する必要があります。1 ~ 10,000 の範囲で任意のパート番号を選択できます。パート番号によって、アップロードするオブジェクトに含まれるパートとその位置が一意に識別されます。以前にアップロードしたパートと同じパート番号を使って新しいパートをアップロードした場合、以前のパートは上書きされます。パートをアップロードするたびに、*ETag* ヘッダーを含むレスポンスが Amazon S3 から返されます。パートのアップロードごとに、パート番号と ETag 値を記録する必要があります。マルチパートアップロードを完了するためには、残りのリクエストにこれらの値を含める必要があるからです。

マルチパートアップロードの完了 (または中止)

マルチパートアップロードを完了すると、パート番号に基づいて昇順に連結されたオブジェクトが Amazon S3 によって作成されます。*マルチパートアップロードの開始*リクエストにオブジェクトメタデータが指定されている場合、そのメタデータは Amazon S3 によりオブジェクトに関連付けられます。*完了*リクエストが正常に処理されると、個々のパートはなくなります。*マルチパートアップロードの完了*リクエストには、アップロード ID と、パート番号およびそれに対応する ETag 値のリストが含まれている必要があります。Amazon S3 からのレスポンスには、結合されるオブジェクトデータを一意に識別する ETag が含まれています。この ETag はオブジェクトデータの MD5 ハッシュになるとは限りません。マルチパートアップロードは中止することもできます。マルチパートアップロードを中止した後は、再度同じアップロード ID を使ってパートをアップロードすることはできません。中止したマルチパートアップロードの任意のパートによって使用されていたストレージはすべて解放されます。任意のパートのアップロードが進行しているときにマルチパートアップロードを中止した場合は、中止後もそのパートのアップロードは成功または失敗する可能性があります。すべてのパートによって使用されているストレージを全部解放するには、すべてのパートのアップロードが完了した後で初めてマルチパートアップロードを中止する必要があります。

マルチパートアップロードを表示する

特定のマルチパートアップロードのパートや、進行中のすべてのマルチパートアップロードを表示できます。パートのリストオペレーションでは、特定のマルチパートアップロードについて既にアップロードしたパートの情報が返されます。パートのリストリクエストを送信するたびに、指定したマルチパートアップロードのパート情報 (最大で 1,000 個のパート) が Amazon S3 から返されます。マルチパートアップロードに 1,000 個を超えるパートが含まれる場合、すべてのパートを取得するにはパートのリストリクエストを追加で送信する必要があります。返されるパートのリストには、アップロードが完了していないパートは含まれていないことにご留意ください。

Note

返されるリストは確認の目的でのみ使用します。*マルチパートアップロードの完了*リクエストを送信するときに、リストの結果を使用しないでください。その代わりに、パートのアップロード時に指定したパート番号と、それに対応する、Amazon S3 から返される ETag 値で構成された独自のリストを保持しておいてください。

*マルチパートアップロードのリスト*オペレーションを使用すると、進行中のマルチパートアップロードのリストを取得できます。進行中のマルチパートアップロードとは、開始されているものの、まだ完了または中止されていないアップロードを意味します。リクエストごとに最大 1,000 個のマルチパートアップロードが返されます。進行中のマルチパートアップロードが 1,000 個を超えている場合、残りのマルチパートアップロードを取得するには、リクエストを追加で送信する必要があります。

マルチパートアップロードの同時オペレーション

分散開発環境においては、アプリケーションから同じオブジェクトに対して複数の更新が同時に開始されることもありえます。同じオブジェクトキーを使ってアプリケーションから複数のマルチパートアップロードが開始される可能性もあります。そのようなアップロードごとに、アプリケーションからパートのアップロードが行われ、アップロードの完了リクエストが Amazon S3 に送信されて、オブジェクトが作成されます。バケットでバージョニングが有効になっている場合は、マルチパートアップロードを完了するたびに新しいバージョンが作成されます。バージョニングが有効になっていないバケットの場合は、マルチパートアップロードの開始から完了までの間に受信された他の何らかのリクエストが優先されることもありえます。

Note

マルチパートアップロードの開始から完了までの間に受信されたその他のリクエストが優先されることもありえます。例えば、あるキーを使ってマルチパートアップロードを開始した後、アップロードが完了しないうちに別のオペレーションによってそのキーが削除されたとします。その場合、オブジェクトを確認できなくても、マルチパートアップロードの完了レスポンスによってオブジェクト作成の成功が示される可能性があります。

マルチパートアップロードと料金

マルチパートアップロードを開始すると、アップロードを完了または中止するまですべてのパートが Amazon S3 によって保持されます。マルチパートアップロードの実行期間を通して、アップロードとそれに関連するパートのために使用されるすべてのストレージ、帯域幅、リクエストに対して課金が行われます。マルチパートアップロードを中止した場合、アップロードアーティファクトおよびアップロードしたすべてのパートは Amazon S3 によって削除され、それらに対して課金されることはなくなります。料金表については、Amazon S3 料金表のページを参照してください。

基本情報

次の表は、マルチパートアップロード (「マルチパートアップロードの概要 (p. 149)」を参照) の主な仕様をまとめたものです。

項目	仕様
最大オブジェクトサイズ	5 TB
アップロードあたりの最大パート数	10,000
パート番号	1 ~ 10,000
パートサイズ	5 MB ~ 5 GB、最後のパートは < 5 MB の場合もあり
パートのリストリクエストで返されるパートの最大数	1,000
マルチパートアップロードのリストリクエストで返されるマルチパートアップロードの最大数	1,000

マルチパートアップロードの API サポート

オブジェクトをいくつかに分けてアップロードするには、AWS SDK を使用できます。次の AWS SDK ライブラリはマルチパートアップロードをサポートしています。

- AWS SDK for Java
- AWS SDK for .NET
- AWS SDK for PHP

これらのライブラリは、マルチパートオブジェクトのアップロードを容易にする、高レベルの抽象化を実現します。ただし、アプリケーションの必要に応じて REST API を直接使用することもできます。マルチパートアップロードを行うための REST API については、『Amazon Simple Storage Service API リファレンス』の以下のセクションを参照してください。

- Initiate Multipart Upload
- Upload Part
- パートのアップロード (コピー)
- Complete Multipart Upload
- マルチパートアップロードの中止
- パートのリスト
- マルチパートアップロードのリスト

マルチパートアップロード API とアクセス許可

マルチパートアップロードオペレーションを使用するには、そのためのアクセス許可が必要です。マルチパートアップロードオペレーションを実行するためのアクセス許可を付与するには、ACL、バケットポリシー、ユーザーポリシーを使用できます。次の表は、ACL、バケットポリシー、またはユーザーポリシーを使用して割り当てることのできる、さまざまなマルチパートアップロードオペレーションに必要なアクセス許可をまとめたものです。

アクション	必要なアクセス許可
マルチパートアップロードの開始	マルチパートアップロードを開始するには、オブジェクトに対して *s3:PutObject* アクションを実行するための許可が必要です。 バケット所有者は他のプリンシパルに対して *s3:PutObject* アクションの実行を許可できます。
パートのアップロード	パートをアップロードするには、オブジェクトに対して *s3:PutObject* アクションを実行するための許可が必要です。 マルチパートアップロードの開始者だけがパートをアップロードできます。マルチパートアップロードの開始者がオブジェクトのパートをアップロードできるようにするため、バケット所有者はその開始者に対しオブジェクトへの *s3:PutObject* アクションの実行を許可する必要があります。
パートのアップロード (コピー)	パートをアップロードするには、オブジェクトに対して *s3:PutObject* アクションを実行するための許可が必要です。既存のオブジェクトからパートをアップロードするので、ソースオブジェクトに対して *s3:GetObject* を実行するための許可が必要です。 マルチパートアップロードの開始者だけがパートをアップロードできます。マルチパートアップロードの開始者がオブジェクトのパートをアップロードできるようにするため、バケット所有者はその開始者に対しオブジェクトへの *s3:PutObject* アクションの実行を許可する必要があります。

アクション	必要なアクセス許可
マルチパートアップロードの完了	マルチパートアップロードを完了するには、オブジェクトに対して *s3:PutObject* アクションを実行するための許可が必要です。 マルチパートアップロードの開始者だけが、そのマルチパートアップロードを完了できます。マルチパートアップロードの開始者がオブジェクトのアップロードを完了できるようにするため、バケット所有者はその開始者に対しオブジェクトへの *s3:PutObject* アクションの実行を許可する必要があります。
マルチパートアップロードの中止	マルチパートアップロードを中止するには、*s3:AbortMultipartUpload* アクションを実行するための許可が必要です。 デフォルトでは、バケット所有者とマルチパートアップロードの開始者が、このアクションの実行を許可されます。開始者が IAM ユーザーである場合、そのユーザーの AWS アカウントもそのマルチパートアップロードを中止することを許可されます。 このようなデフォルト設定に加え、バケット所有者は他のプリンシパルに対してオブジェクトへの *s3:AbortMultipartUpload* アクションの実行を許可できます。バケット所有者は任意のプリンシパルに対し、*s3:AbortMultipartUpload* アクションを実行する権限を無効にすることができます。
パートのリスト	マルチパートアップロードに含まれるパートをリストするには、*s3:ListMultipartUploadParts* アクションを実行するための許可が必要です。 デフォルトではバケット所有者が、バケットに対する任意のマルチパートアップロードについてパートのリストを許可されています。マルチパートアップロードの開始者は、特定のマルチパートアップロードについてパートのリストを許可されます。マルチパートアップロードの開始者が IAM ユーザーである場合、その IAM ユーザーを管理している AWS アカウントもそのアップロードのパートのリストを許可されます。 このようなデフォルト設定に加え、バケット所有者は他のプリンシパルに対してオブジェクトへの *s3:ListMultipartUploadParts* アクションの実行を許可できます。バケット所有者は任意のプリンシパルに対し、*s3:ListMultipartUploadParts* アクションを実行する権限を無効にすることもできます。
マルチパートアップロードのリスト	バケットに対して進行中のマルチパートアップロードをリストするには、そのバケットに対して *s3:ListBucketMultipartUploads* アクションを実行するための許可が必要です。 デフォルト設定に加え、バケット所有者は他のプリンシパルに対してバケットへの *s3:ListBucketMultipartUploads* アクションの実行を許可できます。

アクションの詳細およびポリシー内でのアクションの使い方については、「Action (p. 497)」を参照してください。ポリシーのアクションと ACL のアクセス許可との関係については、「アクションとアクセス許可の関係 (p. 362)」を参照してください。IAM ユーザーについては、Working with Users and Groups を参照してください。

オブジェクトの取得

Topics

- 他ユーザーとのオブジェクトの共有 (p. 163)

Amazon S3 からオブジェクトを直接取得できます。オブジェクトを取得するときは、以下のオプションを利用できます。

- オブジェクトを 1 回で取得する – Amazon S3 に格納されているオブジェクトを 1 回の GET オペレーションで取得できます。
- オブジェクトをいくつかに分けて取得する – GET リクエストの Range HTTP ヘッダーを使用すると、Amazon S3 に格納されているオブジェクトの特定のバイト範囲を取得できます。アプリケーションの準備ができた時点でいつでも、オブジェクトの残り部分の取得を再開できます。この再開可能なダウンロードは、オブジェクトデータの一部だけが必要な場合に使用すると便利です。また、ネットワーク接続の状態が悪く、障害に対処する必要がある場合にも役立ちます。

オブジェクトを取得するとき、そのメタデータはレスポンスヘッダーに格納されて返されます。GET レスポンスで返されるレスポンスヘッダーの特定の値を上書きしたい場合があります。例えば、GET リクエストで `Content-Disposition` レスポンスヘッダー値を上書きする場合などです。REST GET Object API (「GET Object」を参照) を使用すると、GET リクエストでクエリ文字列パラメータを指定することで、そのような値を上書きできるようになります。

AWS SDK for Java、AWS SDK for .NET、および AWS SDK for PHP でも、GET リクエストでレスポンスヘッダーの値を指定するために必要なオブジェクトが用意されています。

関連リソース

- AWS dynamo と Explorer の使用 (p. 475)

AWS SDK for Java を使用したオブジェクトの取得

オブジェクトをダウンロードするときは、オブジェクトのメタデータと、コンテンツの読み取り元のストリームがすべて取得されます。ストリームコンテンツの読み取りはなるべく短時間で行う必要があります。データは Amazon S3 から直接ストリーミングされ、すべてのデータの読み取りが完了するか、入力ストリームを閉じるまで、ネットワーク接続は開いたままになるからです。

オブジェクトのダウンロード

1	AWS 認証情報を指定して、`AmazonS3Client` クラスのインスタンスを作成します。
2	いずれかの `AmazonS3Client.getObject()` メソッドを実行します。バケット名やキー名などのリクエスト情報を提供する必要があります。この情報は、`GetObjectRequest` クラスのインスタンスを作成することによって指定します。
3	返されるオブジェクトに対していずれかの `getObjectContent()` メソッドを実行してオブジェクトデータに関するストリームを取得し、レスポンスを処理します。

以下の Java コード例は、前述のタスクを実装したものです。

```
AWSCredentials myCredentials = new BasicAWSCredentials(
                                 myAccessKeyID, mySecretKey);
AmazonS3 s3Client = new AmazonS3Client(myCredentials);
S3Object object = s3Client.getObject(
                  new GetObjectRequest(bucketName, key));
InputStream objectData = object.getObjectContent();
```

```
// Process the objectData stream.
objectData.close();
```

GetObjectRequest オブジェクトではいくつかのオプションを利用できます。例えば、変更時間や ETag に基づいてオブジェクトを条件付きでダウンロードしたり、オブジェクトの範囲を部分的にダウンロードしたりできます。次の Java コード例は、オブジェクトから取得するデータバイトの範囲を指定する方法を示しています。

```
AWSCredentials myCredentials = new BasicAWSCredentials(
                                   myAccessKeyID, mySecretKey);
AmazonS3 s3Client = new AmazonS3Client(myCredentials);

GetObjectRequest rangeObjectRequest = new GetObjectRequest(
  bucketName, key);
rangeObjectRequest.setRange(0, 10); // retrieve 1st 10 bytes.
S3Object objectPortion = s3Client.getObject(rangeObjectRequest);

InputStream objectData = objectPortion.getObjectContent();
// Process the objectData stream.
objectData.close();
```

オブジェクトを取得するときは、オプションでレスポンスヘッダーの値を上書きすることもできます（「オブジェクトの取得 (p. 153)」を参照）。その場合は、次の Java コード例に示すとおり、ResponseHeaderOverrides オブジェクトを使用して、対応するリクエストプロパティを設定します。

```
GetObjectRequest request = new GetObjectRequest(bucketName, key);

ResponseHeaderOverrides responseHeaders = new ResponseHeaderOverrides();
responseHeaders.setCacheControl("No-cache");
responseHeaders.setContentDisposition("attachment; filename=testing.txt");

// Add the ResponseHeaderOverides to the request.
request.setResponseHeaders(responseHeaders);
```

Example

次の Java コード例では、指定した Amazon S3 バケットからオブジェクトを取得します。実際に動作するコードを作成、テストする方法については、「Java コード例のテスト (p. 477)」を参照してください。

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.GetObjectRequest;
import com.amazonaws.services.s3.model.S3Object;

public class S3Sample {
 private static String bucketName = "*** Provide bucket name ***";
 private static String key        = "*** Provide Key ***";

 public static void main(String[] args) throws IOException {
        AmazonS3 s3Client = new AmazonS3Client(new PropertiesCredentials(
                S3Sample.class.getResourceAsStream(
                  "AwsCredentials.properties")));
        try {
            System.out.println("Downloading an object");
            S3Object s3object = s3Client.getObject(new GetObjectRequest(
              bucketName, key));
            System.out.println("Content-Type: "  +
              s3object.getObjectMetadata().getContentType());
            displayTextInputStream(s3object.getObjectContent());

           // Get a range of bytes from an object.

            GetObjectRequest rangeObjectRequest = new GetObjectRequest(
              bucketName, key);
            rangeObjectRequest.setRange(0, 10);
            S3Object objectPortion = s3Client.getObject(rangeObjectRequest);

            System.out.println("Printing bytes retrieved.");
            displayTextInputStream(objectPortion.getObjectContent());

        } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException, which" +
              " means your request made it " +
                    "to Amazon S3, but was rejected with an error response" +
                    " for some reason.");
            System.out.println("Error Message:    " + ase.getMessage());
            System.out.println("HTTP Status Code: " + ase.getStatusCode());
            System.out.println("AWS Error Code:   " + ase.getErrorCode());
            System.out.println("Error Type:       " + ase.getErrorType());
            System.out.println("Request ID:       " + ase.getRequestId());
        } catch (AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException, which means"+
```

```
              " the client encountered " +
                    "an internal error while trying to " +
                    "communicate with S3, " +
                    "such as not being able to access the network.");
            System.out.println("Error Message: " + ace.getMessage());
        }
    }

    private static void displayTextInputStream(InputStream input)
    throws IOException {
     // Read one text line at a time and display.
        BufferedReader reader = new BufferedReader(new
          InputStreamReader(input));
        while (true) {
            String line = reader.readLine();
            if (line == null) break;

            System.out.println("    " + line);
        }
        System.out.println();
    }
}
```

AWS SDK for .NET を使用したオブジェクトの取得

以下のタスクは、.NET クラスを使用して、オブジェクトまたはオブジェクトの一部を取得する手順を示しています。

オブジェクトのダウンロード

1	AWS 認証情報を指定して、AmazonS3 クラスのインスタンスを作成します。
2	AmazonS3.GetObject メソッドのいずれかを実行します。バケット名、ファイルパス、ストリームなどの情報を提供する必要があります。この情報は、GetObjectRequest クラスのインスタンスを作成することによって指定します。
3	いずれかの GetObjectResponse.WriteResponseStreamToFile メソッドを実行して、ストリームをファイルに保存します。

以下の C# コード例は、前述のタスクを実装したものです。

```
static AmazonS3 client;
client = Amazon.AWSClientFactory.CreateAmazonS3Client(
                    accessKeyID, secretAccessKeyID);

GetObjectRequest request = new GetObjectRequest()
            .WithBucketName(bucketName).WithKey(keyName);
using (GetObjectResponse response = client.GetObject(request))
{
   string title = response.Metadata["x-amz-meta-title"];
   Console.WriteLine("The object's title is {0}", title);
   string dest = Path.Combine(Environment.GetFolderPath(
         Environment.SpecialFolder.Desktop), keyName);
   if (!File.Exists(dest))
```

```
    {
        response.WriteResponseStreamToFile(dest);
    }
}
```

オブジェクト全体を読み取る代わりに、オブジェクトデータの一部だけを読み取るには、次の C# コード例に示すとおり、リクエスト内でバイト範囲を指定します。

```
GetObjectRequest request = new GetObjectRequest()
        .WithBucketName(bucketName)
        .WithKey(keyName)
        .WithByteRange(0, 10);
```

オブジェクトを取得するときは、オプションでレスポンスヘッダーの値を上書きすることもできます（「オブジェクトの取得 (p. 153)」を参照）。その場合は、次の C# コード例に示すとおり、`ResponseHeaderOverrides` オブジェクトを使用して、対応するリクエストプロパティを設定します。

```
GetObjectRequest request = new GetObjectRequest()
     .WithBucketName(bucketName).WithKey(keyName);

ResponseHeaderOverrides responseHeaders = new ResponseHeaderOverrides();
responseHeaders.CacheControl = "No-cache";
responseHeaders.ContentDisposition = "attachment; filename=testing.txt";

request.ResponseHeaderOverrides = responseHeaders;
```

Example

次の C# コード例では、Amazon S3 バケットからオブジェクトを取得します。この例では、`GetObjectResponse.ResponseStream` プロパティを使用してレスポンスからオブジェクトデータを読み取っています。実際に動作するコードを作成、テストする方法については、「.NET コード例のテスト (p. 478)」を参照してください。

```
using System;
using System.Configuration;
using System.Collections.Specialized;
using System.IO;

using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples.retrieveobject
{
    class S3Sample
    {
        static string bucketName = "*** Provide bucket name ***";
        static string keyName    = "*** Provide object key ***";
        static AmazonS3 client;

        public static void Main(string[] args)
        {
            if (checkRequiredFields())
            {
                NameValueCollection appConfig =
                    ConfigurationManager.AppSettings;

                string accessKeyID = appConfig["AWSAccessKey"];
                string secretAccessKeyID = appConfig["AWSSecretKey"];
                try
                {
                    Console.WriteLine("Retrieving (getting) an object");
                    string data = ReadingAnObject(
                                                accessKeyID, secretAccessKeyID);
                }
                catch (AmazonS3Exception s3Exception)
                {
                    Console.WriteLine(s3Exception.Message,
                                        s3Exception.InnerException);
                }
            }

            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }

        static string ReadingAnObject(
                        string accessKeyID, string secretAccessKeyID)
        {
            string responseBody = "";
            using (client = Amazon.AWSClientFactory.CreateAmazonS3Client(
                                                accessKeyID, secretAccessKeyID))
            {
                GetObjectRequest request = new GetObjectRequest()
                    .WithBucketName(bucketName).WithKey(keyName);
```

```
                using (GetObjectResponse response = client.GetObject(request))

                {
                    string title = response.Metadata["x-amz-meta-title"];
                    Console.WriteLine("The object's title is {0}", title);

                    using (Stream responseStream = response.ResponseStream)
                    {
                        using (StreamReader reader =
                            new StreamReader(responseStream))
                        {
                            responseBody = reader.ReadToEnd();
                        }
                    }
                }
            }
            return responseBody;
        }

        static bool checkRequiredFields()
        {
            NameValueCollection appConfig = ConfigurationManager.AppSettings;

            if (string.IsNullOrEmpty(appConfig["AWSAccessKey"]))
            {
                Console.WriteLine(
                    "AWSAccessKey was not set in the App.config file.");
                return false;
            }
            if (string.IsNullOrEmpty(appConfig["AWSSecretKey"]))
            {
                Console.WriteLine(
                    "AWSSecretKey was not set in the App.config file.");
                return false;
            }
            if (string.IsNullOrEmpty(bucketName))
            {
                Console.WriteLine("The variable bucketName is not set.");
                return false;
            }
            if (string.IsNullOrEmpty(keyName))
            {
                Console.WriteLine("The variable keyName is not set.");
                return false;
            }

            return true;
        }
    }
}
```

AWS SDK for PHP を使用したオブジェクトの取得

このトピックでは、AWS SDK for PHP のクラスを使用して、オブジェクトを取得する手順を示します。オブジェクトの全体を取得することも、またはオブジェクトから取得する特定のバイト範囲を指定することもできます。

Note

このトピックでは、既に AWS SDK for PHP の使用と PHP サンプルの実行 (p. 479) の説明が実行されていて、AWS SDK for PHP が正しくインストールされていることを前提としています。

オブジェクトのダウンロード

1	自分の AWS 認証情報または IAM ユーザー認証情報と共に Aws\S3\S3Client クラスの factory() メソッドを使用して、Amazon S3 クライアントのインスタンスを作成します。
2	Aws\S3\S3Client::getObject() メソッドを実行します。array パラメータの必須キー、Bucket と Key に、バケット名とキー名を指定する必要があります。 オブジェクト全体を取得する代わりに、オプションでオブジェクトデータの特定のバイト範囲を取得することもできます。範囲の値を指定するには、必要なキーのほかに array パラメータの Range キーを指定します。 Amazon S3 から取得したオブジェクトはファイルに保存できます。保存するには、必要なキー、Bucket と Key のほかに、array パラメータの SaveAs キーにファイルの保存場所へのファイルパスを指定します。

以下の PHP コード例は、オブジェクトをダウンロードするための前述のタスクを実装したものです。

```
use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';
$filepath = '*** Your File Path ***';

// Instantiate the client.
$s3 = S3Client::factory(array(
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));

// Get an object.
$result = $s3->getObject(array(
    'Bucket' => $bucket,
    'Key'    => $keyname
));

// Get a range of bytes from an object.
$result = $s3->getObject(array(
    'Bucket' => $bucket,
    'Key'    => $keyname,
    'Range'  => 'bytes=0-99'
));

// Save object to a file.
$result = $s3->getObject(array(
    'Bucket' => $bucket,
    'Key'    => $keyname,
    'SaveAs' => $filepath
));
```

オブジェクトを取得するときは、オプションでレスポンスヘッダーの値を上書きすることもできます（「オブジェクトの取得 (p. 153)」を参照）。上書きするには、次の PHP コード例に示すとおり、array パラメータのレスポンスキー、ResponseContentType、ResponseContentLanguage、ResponseContentDisposition、ResponseCacheControl、および ResponseExpires を getObject() メソッドに追加します。

```
$result = $s3->getObject(array(
    'Bucket'                       => $bucket,
    'Key'                          => $keyname,
    'ResponseContentType'          => 'text/plain',
    'ResponseContentLanguage'      => 'en-US',
    'ResponseContentDisposition' => 'attachment; filename=testing.txt',
    'ResponseCacheControl'         => 'No-cache',
    'ResponseExpires'              => gmdate(DATE_RFC2822, time() + 3600),
));
```

Example（PHP を使用したオブジェクトのダウンロード）

次の PHP コード例では、オブジェクトを取得し、そのコンテンツをブラウザに表示します。この例は、getObject() メソッドの使い方を示しています。PHP 例の実行については、このガイド内の「PHP サンプルの実行 (p. 480)」を参照してください。

```
<?php

// Include the AWS SDK using the Composer autoloader.
require 'vendor/autoload.php';

use Aws\S3\S3Client;
use Aws\S3\Exception\S3Exception;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

// Instantiate the client.
$s3 = S3Client::factory(array(
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));

try {
    // Get the object
    $result = $s3->getObject(array(
        'Bucket' => $bucket,
        'Key'    => $keyname
    ));

    // Display the object in the browser
    header("Content-Type: {$result['ContentType']}");
    echo $result['Body'];
} catch (S3Exception $e) {
    echo $e->getMessage() . "\n";
}
```

関連リソース

- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client クラス」

- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::factory() メソッド」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::getObject() メソッド」
- 「AWS SDK for PHP for Amazon S3 – オブジェクトのダウンロード」
- 「AWS SDK for PHP – Amazon S3」
- 「AWS SDK for PHP」のドキュメント

REST API を使用した 1 つのオブジェクトの取得

AWS SDK を使用して、バケット内のオブジェクトキーをリストできます。ただし、アプリケーションで必要な場合は、REST リクエストを直接送信できます。GET リクエストを送信してオブジェクトキーを取得できます。リクエストとレスポンスの形式の詳細については、「GET Object」を参照してください。

他ユーザーとのオブジェクトの共有

Topics

デフォルトでは、すべてのオブジェクトがプライベートです。オブジェクトの所有者のみがプライベートのオブジェクトにアクセスできます。ただし、オブジェクトの所有者はオプションで他のユーザーとオブジェクトを共有することができます。その場合は、署名付き URL を作成し、独自のセキュリティ認証情報を使用して、オブジェクトをダウンロードするための期限付きの許可を相手に付与します。

オブジェクトの署名付き URL を作成するときは、セキュリティ認証情報を提供し、バケット名とオブジェクトキーを指定する必要があります。また、HTTP メソッド (オブジェクトをダウンロードするには GET) と有効期限の日時も指定する必要があります。署名付き URL は、指定した期間のみ有効です。

署名付き URL を受け取った相手は誰でも、そのオブジェクトにアクセスできるようになります。例えば、プライベートのバケット内にプライベートの動画を格納している場合は、署名付き URL を生成することで、その動画を他のユーザーと共有できます。

Note

有効なセキュリティ認証情報を持つすべてのユーザーが、署名付き URL を作成できます。ただし、オブジェクトを正常にアクセスするには、署名付き URL の基となるオペレーションを実行する権限を持つユーザーが、署名付き URL を作成する必要があります。

署名付き URL をプログラムによって生成するには、AWS SDK for Java と .NET を使用します。

AWS Explorer for Visual Studio を使用した署名付きオブジェクト URL の生成

Visual Studio を使用している場合は、AWS Explorer for Visual Studio を使用することで、コードを一切記述しなくてもオブジェクトの署名付き URL を生成できます。この URL を受け取った相手は誰でも、オブジェクトをダウンロードできるようになります。詳細については、「Using Amazon S3 from AWS Explorer」を参照してください。

AWS Explorer をインストールする方法については、「AWS dynamo と Explorer の使用 (p. 475)」を参照してください。

AWS SDK for Java を使用した署名付きオブジェクト URL の生成

以下のタスクは、Java クラスを使用して、署名付き URL を生成する手順を示しています。

オブジェクトのダウンロード

1	自分の AWS 証明書を指定して、AmazonS3 クラスのインスタンスを作成します。 これらの認証情報は、署名付き URL を生成するときに、認証用の署名を作成するのに使用されます。
2	AmazonS3.generatePresignedUrl メソッドを実行して署名付き URL を生成します。 バケット名、オブジェクトキー、有効期限などの情報を提供するには、GeneratePresignedUrlRequest クラスのインスタンスを作成します。

以下の Java コード例は、前述のタスクを実装したものです。

```
AWSCredentials myCredentials = new BasicAWSCredentials(
                                 myAccessKeyID, mySecretKey);
AmazonS3 s3Client = new AmazonS3Client(myCredentials);

java.util.Date expiration = new java.util.Date();
long msec = expiration.getTime();
msec += 1000 * 60 * 60; // 1 hour.
expiration.setTime(msec);

GeneratePresignedUrlRequest generatePresignedUrlRequest =
              new GeneratePresignedUrlRequest(bucketName, objectKey);
generatePresignedUrlRequest.setMethod(HttpMethod.GET); // Default.
generatePresignedUrlRequest.setExpiration(expiration);

URL s = s3client.generatePresignedUrl(generatePresignedUrlRequest);
```

Example

次の Java コード例では、署名付き URL を生成しています。この URL は、オブジェクトの取得を可能にするために他のユーザーに配布できます。作業サンプルを作成およびテストする方法については、「Java コード例のテスト (p. 477)」を参照してください。

```
import java.io.IOException;
import java.net.URL;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.HttpMethod;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.GeneratePresignedUrlRequest;
import java.net.*;
import java.io.*;

public class S3Sample {
 private static String bucketName = "*** Provide a bucket name ***";
 private static String objectKey = "*** Provide an object key ***";

 public static void main(String[] args) throws IOException {
  AmazonS3 s3client = new AmazonS3Client(new PropertiesCredentials(
   S3Sample.class.getResourceAsStream("AwsCredentials.properties")));

  try {
   System.out.println("Generating pre-signed URL.");
   java.util.Date expiration = new java.util.Date();
   long milliSeconds = expiration.getTime();
   milliSeconds += 1000 * 60 * 60; // Add 1 hour.
   expiration.setTime(milliSeconds);

   GeneratePresignedUrlRequest generatePresignedUrlRequest =
        new GeneratePresignedUrlRequest(bucketName, objectKey);
   generatePresignedUrlRequest.setMethod(HttpMethod.GET);
   generatePresignedUrlRequest.setExpiration(expiration);

   URL url = s3client.generatePresignedUrl(generatePresignedUrlRequest);

   System.out.println("Pre-Signed URL = " + url.toString());
  } catch (AmazonServiceException exception) {
   System.out.println("Caught an AmazonServiceException, " +
     "which means your request made it " +
     "to Amazon S3, but was rejected with an error response " +
   "for some reason.");
   System.out.println("Error Message: " + exception.getMessage());
   System.out.println("HTTP  Code: "    + exception.getStatusCode());
   System.out.println("AWS Error Code:" + exception.getErrorCode());
   System.out.println("Error Type:    " + exception.getErrorType());
   System.out.println("Request ID:    " + exception.getRequestId());
  } catch (AmazonClientException ace) {
   System.out.println("Caught an AmazonClientException, " +
     "which means the client encountered " +
     "an internal error while trying to communicate" +
     " with S3, " +
   "such as not being able to access the network.");
```

```
   System.out.println("Error Message: " + ace.getMessage());
  }
 }
}
```

AWS SDK for .NET を使用した署名付きオブジェクト URL の生成

以下のタスクは、.NET クラスを使用して、署名付き URL を生成する手順を示しています。

オブジェクトのダウンロード

1	AWS 認証情報を指定して、AmazonS3 クラスのインスタンスを作成します。これらの認証情報は、署名付き URL を生成するときに、認証用の署名を作成するのに使用されます。
2	AmazonS3.GetPreSignedURL メソッドを実行して署名付き URL を生成します。 バケット名、オブジェクトキー、有効期限などの情報を提供するには、GetPreSignedUrlRequest クラスのインスタンスを作成します。

以下の C# コード例は、前述のタスクを実装したものです。

```
static AmazonS3 client;
client = Amazon.AWSClientFactory.CreateAmazonS3Client(
                    accessKeyID, secretAccessKeyID);

GetPreSignedUrlRequest request = new GetPreSignedUrlRequest();
request.WithBucketName(bucketName);
request.WithKey(objectKey);
request.Verb = HttpVerb.GET; // Default.
request.WithExpires(DateTime.Now.AddMinutes(5));

string url = s3Client.GetPreSignedURL(request);
```

Example

次の C# コード例では、特定のオブジェクトの署名付き URL を生成します。実際に動作するコードを作成、テストする方法については、「.NET コード例のテスト (p. 478)」を参照してください。

```
using System;
using System.Configuration;
using System.Collections.Specialized;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples.presignedurl
{
    class S3Sample
    {
        static string bucketName = "*** Provide a bucket name ***";
        static string objectKey = "*** Provide an object name ***";
        static AmazonS3 s3Client;

        public static void Main(string[] args)
        {
            NameValueCollection appConfig = ConfigurationManager.AppSettings;

            string accessKeyID = appConfig["AWSAccessKey"];
            string secretAccessKeyID = appConfig["AWSSecretKey"];
               using (s3Client = Amazon.AWSClientFactory.CreateAmazonS3Client(

                    accessKeyID, secretAccessKeyID))
                {
                        GeneratePreSignedURL();
                }


            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }

        static void GeneratePreSignedURL()
        {
            GetPreSignedUrlRequest request = new GetPreSignedUrlRequest();
            request.WithBucketName(bucketName);
            request.WithKey(objectKey);
            request.WithExpires(DateTime.Now.AddMinutes(5));

            try
            {
                string url = s3Client.GetPreSignedURL(request);
            }
            catch (AmazonS3Exception amazonS3Exception)
            {
                if (amazonS3Exception.ErrorCode != null &&
                    (amazonS3Exception.ErrorCode.Equals("InvalidAccessKeyId")
                    ||
                    amazonS3Exception.ErrorCode.Equals("InvalidSecurity")))
                {
                    Console.WriteLine("Check the provided AWS Credentials.");
                    Console.WriteLine(
                    "To sign up for service, go to http://aws.amazon.com/s3");
```

```
                }
                else
                {
                    Console.WriteLine(
                     "Error occurred. Message:'{0}' when listing objects",
                     amazonS3Exception.Message);
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }

        }
    }
}
```

オブジェクトのアップロード

Topics

- 関連リソース (p. 168)
- 1 回のオペレーションでのオブジェクトのアップロード (p. 169)
- マルチパートアップロード API を使用したオブジェクトのアップロード (p. 178)
- 署名付き URL を使用したオブジェクトのアップロード (p. 215)

Amazon S3 には、アップロードするデータのサイズに応じた以下のオプションが用意されています。

- 1 回のオペレーションでオブジェクトをアップロードする – 1 回の PUT オペレーションでアップロードできるオブジェクトの最大サイズは 5 GB です。
  詳細については、「1 回のオペレーションでのオブジェクトのアップロード (p. 169)」を参照してください。
- オブジェクトをいくつかに分けてアップロードする – マルチパートアップロード API を使用すると、最大 5 TB の大容量オブジェクトをアップロードできます。
  マルチパートアップロード API は大容量オブジェクトのアップロードを効率よく行えるように設計されています。オブジェクトをいくつかに分けてアップロードできます。オブジェクトのパートは、単独で、任意の順序で、または並行してアップロードできます。マルチパートアップロードは 5 MB ~ 5 TB のオブジェクトで使用できます。詳細については、「Uploading Objects Using Multipart Upload」を参照してください。詳細については、「マルチパートアップロード API を使用したオブジェクトのアップロード (p. 178)」を参照してください。

Amazon S3 のお客様におかれては、100 MB を超えるオブジェクトに対してはマルチパートアップロードを使用されることをお勧めします。

関連リソース

- AWS dynamo と Explorer の使用 (p. 475)

1 回のオペレーションでのオブジェクトのアップロード

Topics

AWS SDK を使用して、オブジェクトをアップロードできます。SDK にはデータを容易にアップロードできるラッパーライブラリが用意されています。ただし、アプリケーションで必要とされる場合は、REST API をアプリケーション内で直接使用することもできます。

AWS SDK for Java を使用したオブジェクトのアップロード

以下のタスクは、Java クラスを使用して、ファイルをアップロードする手順を示しています。この API には、データのアップロードを容易にするための、複数のバリエーション (*オーバーロード*) の `putObject` メソッドが用意されています。

オブジェクトのアップロード

1	AWS 認証情報を指定して、`AmazonS3Client` クラスのインスタンスを作成します。
2	ファイルとストリームのいずれからデータをアップロードするかに応じて、いずれかの `AmazonS3Client.putObject` オーバーロードを実行します。

以下の Java コード例は、前述のタスクの例です。

```
AWSCredentials myCredentials = new BasicAWSCredentials(
                                          myAccessKeyID, mySecretKey);
AmazonS3 s3client = new AmazonS3Client(myCredentials);
s3client.putObject(new PutObjectRequest(bucketName, keyName, file));
```

Example

次の Java コード例では、ファイルを Amazon S3 バケットにアップロードします。実際に動作するコードを作成、テストする方法については、「Java コード例のテスト (p. 477)」を参照してください。

```
import java.io.File;
import java.io.IOException;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.PutObjectRequest;

public class S3Sample {
 private static String bucketName     = "*** Provide bucket name ***";
 private static String keyName        = "*** Provide key ***";
 private static String uploadFileName = "*** Provide file name ***";

 public static void main(String[] args) throws IOException {
        AmazonS3 s3client = new AmazonS3Client(new PropertiesCredentials(
                S3Sample.class.getResourceAsStream(
                  "AwsCredentials.properties")));
        try {
            System.out.println("Uploading a new object to S3 from a file\n");
            File file = new File(uploadFileName);
            s3client.putObject(new PutObjectRequest(
                             bucketName, keyName, file));

         } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException, which " +
              "means your request made it " +
                    "to Amazon S3, but was rejected with an error response" +
                    " for some reason.");
            System.out.println("Error Message:    " + ase.getMessage());
            System.out.println("HTTP Status Code: " + ase.getStatusCode());
            System.out.println("AWS Error Code:   " + ase.getErrorCode());
            System.out.println("Error Type:       " + ase.getErrorType());
            System.out.println("Request ID:       " + ase.getRequestId());
        } catch (AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException, which " +
              "means the client encountered " +
                    "an internal error while trying to " +
                    "communicate with S3, " +
                    "such as not being able to access the network.");
            System.out.println("Error Message: " + ace.getMessage());
        }
    }
}
```

AWS SDK for .NET を使用したオブジェクトのアップロード

次のプロセスの以下のタスクは、.NET のクラスを使用して、1 つのオブジェクトをアップロードする手順を示しています。この API には、データのアップロードを容易にするための、複数のバリエーション (オーバーロード) の `PutObject` メソッドが用意されています。

オブジェクトのアップロード

1	AWS 認証情報を指定して、AmazonS3 クラスのインスタンスを作成します。
2	いずれかの AmazonS3.PutObject を実行します。バケット名、ファイルパス、ストリームなどの情報を提供する必要があります。この情報は、PutObjectRequest クラスのインスタンスを作成することによって指定します。

以下の C# コード例は、前述のタスクの例です。

```
static AmazonS3 client;
client = Amazon.AWSClientFactory.CreateAmazonS3Client(
                    accessKeyID, secretAccessKeyID);

PutObjectRequest request = new PutObjectRequest();
request.WithFilePath(filePath)
       .WithBucketName(bucketName)
       .WithKey(keyName);
S3Response responseWithMetadata =
                     client.PutObject(titledRequest);
```

Note

.NET API を使用して大きなオブジェクトをアップロードする場合、データがリクエストストリームに書き込まれる間にも、タイムアウトが発生することがあります。PutObjectRequest オブジェクトを使用して、明示的なタイムアウトを設定することができます。

Example

次の C# コード例では、オブジェクトをアップロードします。オブジェクトデータはテキスト文字列としてコードに指定されています。この例は、AmazonS3.PutObject を使用してオブジェクトをアップロードする方法を示しています。ドではオブジェクトを 2 回アップロードします。1 回目のオブジェクトアップロードで、PutObjectRequest はバケット名、キー名、およびサンプルオブジェクトデータのみを指定します。2 回目のオブジェクトアップロードで、PutObjectRequest はオプションのオブジェクトメタデータやコンテンツタイプヘッダーなどの追加情報を提供します。後続の AmazonS3.PutObject 呼び出しが行われるたびに、直前のアップロードが置き換えられます。実際に動作するコードを作成、テストする方法については、「.NET コード例のテスト (p. 478)」を参照してください。

```
using System;
using System.Configuration;
using System.Collections.Specialized;
using System.Net;

using Amazon.S3;
using Amazon.S3.Model;
using Amazon.S3.Util;

namespace s3.amazon.com.docsamples.createobject
{
    class S3Sample
    {
        static string bucketName = "*** Provide bucket name ***";
        static string keyName    = "*** Provide key name ***";

        static AmazonS3 client;

        public static void Main(string[] args)
        {
            if (checkRequiredFields())
            {
                NameValueCollection appConfig =
                    ConfigurationManager.AppSettings;

                string accessKeyID = appConfig["AWSAccessKey"];
                string secretAccessKeyID = appConfig["AWSSecretKey"];

                using (client = Amazon.AWSClientFactory.CreateAmazonS3Client(
                    accessKeyID, secretAccessKeyID))
                {

                    Console.WriteLine("Uploading an object");
                    WritingAnObject();
                 }
            }

            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }

         static void WritingAnObject()
        {
            try
            {
                // 1. Simple object put.
```

```
            PutObjectRequest request = new PutObjectRequest();
            request.WithContentBody("Object data for simple put.")
                .WithBucketName(bucketName)
                .WithKey(keyName);

            S3Response response = client.PutObject(request);
            response.Dispose();

           // 2. Put a more complex object with metadata and http headers.

            PutObjectRequest request2 = new PutObjectRequest();
            request2.WithMetaData("title", "the title")
                .WithContentBody("Object data for complex put.")
                //.WithFilePath(filePath)
                .WithBucketName(bucketName)
                .WithKey(keyName);
            // Add a header to the request.
            request2.AddHeaders(AmazonS3Util.CreateHeaderEntry
                                       ("ContentType", "text/xml"));

         S3Response responseWithMetadata = client.PutObject(request2);
        }
        catch (AmazonS3Exception amazonS3Exception)
        {
            if (amazonS3Exception.ErrorCode != null &&
                (amazonS3Exception.ErrorCode.Equals("InvalidAccessKeyId")
                ||
                amazonS3Exception.ErrorCode.Equals("InvalidSecurity")))
            {
                Console.WriteLine("Check the provided AWS Credentials.");
                Console.WriteLine(
                    "For service sign up go to http://aws.amazon.com/s3");

            }
            else
            {
                Console.WriteLine(
                    "Error occurred. Message:'{0}' when writing an object"

                    , amazonS3Exception.Message);
            }
        }
    }

     static bool checkRequiredFields()
     {
         NameValueCollection appConfig = ConfigurationManager.AppSettings;


         if (string.IsNullOrEmpty(appConfig["AWSAccessKey"]))
         {
             Console.WriteLine(
                 "AWSAccessKey was not set in the App.config file.");
             return false;
         }
         if (string.IsNullOrEmpty(appConfig["AWSSecretKey"]))
         {
             Console.WriteLine(
```

```
                    "AWSSecretKey was not set in the App.config file.");
                return false;
            }
            if (string.IsNullOrEmpty(bucketName))
            {
                Console.WriteLine("The variable bucketName is not set.");
                return false;
            }
            if (string.IsNullOrEmpty(keyName))
            {
                Console.WriteLine("The variable keyName is not set.");
                return false;
            }

            return true;
        }
    }
}
```

AWS SDK for PHP を使用したオブジェクトのアップロード

このトピックでは、AWS SDK for PHP のクラスを使用して、5 GB までのサイズのオブジェクトをアップロードする手順を示します。5 GB を超える大容量ファイルの場合は、マルチパートアップロード API を使用する必要があります。詳細については、「マルチパートアップロード API を使用したオブジェクトのアップロード (p. 178)」を参照してください。

Note

このトピックでは、既に AWS SDK for PHP の使用と PHP サンプルの実行 (p. 479) の説明が実行されていて、AWS SDK for PHP が正しくインストールされていることを前提としています。

オブジェクトのアップロード

1	自分の AWS 認証情報または IAM ユーザー認証情報と共に Aws\S3\S3Client クラスの factory() メソッドを使用して、Amazon S3 クライアントのインスタンスを作成します。
2	Aws\S3\S3Client::putObject() メソッドを実行します。array パラメータの必須キー、Bucket と Key に、バケット名とキー名を指定する必要があります。 ファイルをアップロードする場合、ファイル名を指定するには、array パラメータを SourceFile キーと共に追加します。オプションのオブジェクトメタデータを array パラメータを使って指定することもできます。

以下の PHP コード例は、SourceFile メソッドの array パラメータの putObject キーに指定したファイルをアップロードすることで、オブジェクトを実際に作成する方法を示しています。

```
use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';
// $filepath should be absolute path to a file on disk
$filepath = '*** Your File Path ***';

// Instantiate the client.
```

```
$s3 = S3Client::factory(array(
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));

// Upload a file.
$result = $s3->putObject(array(
    'Bucket'       => $bucket,
    'Key'          => $keyname,
    'SourceFile'   => $filePath,
    'ContentType'  => 'text/plain',
    'ACL'          => 'public-read',
    'StorageClass' => 'REDUCED_REDUNDANCY',
    'Metadata'     => array(
        'param1' => 'value 1',
        'param2' => 'value 2'
    )
));

echo $result['ObjectURL'];
```

ファイル名を指定する代わりに、オブジェクトデータをインラインで指定できます。それには、次の PHP コード例に示すとおり、array パラメータと共に Body キーを指定します。

```
use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

// Instantiate the client.
$s3 = S3Client::factory(array(
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));

// Upload data.
$result = $s3->putObject(array(
    'Bucket' => $bucket,
    'Key'    => $keyname,
    'Body'   => 'Hello, world!'
));

echo $result['ObjectURL'];
```

Example （ データをアップロードして Amazon S3 バケットにオブジェクトを作成 ）

以下の PHP コード例では、putObject() メソッドを使用してデータをアップロードすることで、指定されたバケットにオブジェクトを作成しています。PHP 例の実行については、このガイド内の「PHP サンプルの実行 (p. 480)」を参照してください。

```
<?php

// Include the AWS SDK using the Composer autoloader.
require 'vendor/autoload.php';

use Aws\S3\S3Client;
use Aws\S3\Exception\S3Exception;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

// Instantiate the client.
$s3 = S3Client::factory(array(
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));

try {
    // Upload data.
    $result = $s3->putObject(array(
        'Bucket' => $bucket,
        'Key'    => $keyname,
        'Body'   => 'Hello, world!',
        'ACL'    => 'public-read'
    ));

    // Print the URL to the object.
    echo $result['ObjectURL'] . "\n";
} catch (S3Exception $e) {
    echo $e->getMessage() . "\n";
}
```

関連リソース

- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client クラス」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::factory() メソッド」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::putObject() メソッド」
- 「AWS SDK for PHP – Amazon S3」
- 「AWS SDK for PHP」のドキュメント

AWS SDK for Ruby を使用したオブジェクトのアップロード

以下のタスクは、Ruby クラスを使用して、ファイルをアップロードする手順を示しています。この API には、データのアップロード方法を指定するためのオプションを取ることのできる #write メソッドが用意されています。

オブジェクトのアップロード

1	AWS 認証情報を指定して、AWS::S3 クラスのインスタンスを作成します。

2	`AWS::S3::S3Object#write` メソッドを使用します。このメソッドはデータパラメータと、ファイルのデータまたはストリームのアップロードを可能にするオプションハッシュを取ります。

次の Ruby コード例は前述のタスクの例で、*`options`* ハッシュ *`:file`* を使用して、アップロードするファイルのパスを指定しています。

```
s3 = AWS::S3.new

# Upload a file.
key = File.basename(file_name)
s3.buckets[bucket_name].objects[key].write(:file => file_name)
```

Example

次の Ruby スクリプト例では、ファイルを Amazon S3 バケットにアップロードします。実際に動作するコードを作成、テストする方法については、「AWS SDK for Ruby の使用 (p. 482)」を参照してください。

```
#!/usr/bin/env ruby

require 'rubygems'
require 'aws-sdk'

AWS.config(
  :access_key_id => '*** Provide your access key ***',
  :secret_access_key => '*** Provide your secret key ***'
)

bucket_name = '*** Provide bucket name ***'
file_name = '*** Provide file name ****'

# Get an instance of the S3 interface.
s3 = AWS::S3.new

# Upload a file.
key = File.basename(file_name)
s3.buckets[bucket_name].objects[key].write(:file => file_name)
puts "Uploading file #{file_name} to bucket #{bucket_name}."
```

REST API を使用した 1 つのオブジェクトのアップロード

AWS SDK を使用して、1 つのオブジェクトをアップロードできます。ただし、アプリケーションで必要な場合は、REST リクエストを直接送信できます。PUT リクエストを送信して 1 回のオペレーションでデータをアップロードできます。詳細については、「PUT Object」を参照してください。

マルチパートアップロード API を使用したオブジェクトのアップロード

Topics

マルチパートアップロードを使用すると、単一のオブジェクトをパートのセットとしてアップロードすることができます。各パートは、オブジェクトのデータの連続する部分です。これらのオブジェクトパートは、任意の順序で個別にアップロードできます。いずれかのパートの送信が失敗すると、他のパートに影響を与えることなくそのパートを再送することができます。オブジェクトのすべてのパートがアップロードされると、Amazon S3 はパートを組み立ててオブジェクトを作成します。通常、オブジェクトサイズが 100 MB 以上の場合は、単一のオペレーションでオブジェクトをアップロードする代わりに、マルチパートアップロードを使用することを考慮してください。

マルチパートアップロードの使用には、次の利点があります。

- スループットが改善される—パートのアップロードと平行して、スループットを改善することができます。
- ネットワーク問題から速やかに回復できる—パートのサイズは比較的小さいため、ネットワークエラーにより失敗したアップロードを再開する際の影響を最小限に抑えることができます。
- オブジェクトのアップロードを一時停止/再開できる—オブジェクトパートは徐々にアップロードすることができます。マルチパートアップロードをいったん開始すると、終了期限はありません。マルチパートアップロードは明示的に完了または中止する必要があります。
- 最終的なオブジェクトサイズがわからなくてもアップロードを開始できる—オブジェクトを作成しながら、そのオブジェクトをアップロードすることができます。

詳細については、「マルチパートアップロードの概要 (p. 149)」を参照してください。

AWS SDK for Java を使用したマルチパートアップロード

Topics

AWS SDK の使用 (「AWS dynamo と Explorer の使用 (p. 475)」を参照) で説明しているように、SDK for Java はマルチパートアップロード API をサポートしています (「マルチパートアップロード API を使用したオブジェクトのアップロード (p. 178)」を参照)。このセクションでは、高レベルと低レベル両方の SDK for Java API を使用して、オブジェクトをパート単位でアップロードする例を示します。

高レベル Java API を使用したマルチパートアップロード

Topics

AWS SDK for Java では、マルチパートアップロードを簡素化する高レベル API を公開しています (「マルチパートアップロード API を使用したオブジェクトのアップロード (p. 178)」を参照)。ファイルまたはストリームからデータをアップロードできます。必要に応じて、マルチパートアップロードに使用するパートサイズ、パートを同時にアップロードする際に使用するスレッドの数など、詳細なオプション設定することができます。省略可能なオブジェクトのプロパティ、ストレージクラス、または ACL を設定することもできます。これらの詳細なオプションを設定するには、`PutObjectRequest` および `TransferManagerConfiguration` クラスを使用します。Java API の `TransferManager` クラスは、データをアップロードするための高レベル API を提供します。

可能な場合、`TransferManager` は複数のスレッドを使用して、1 回のアップロードに含まれる複数のパートを 1 度にアップロードしようとします。これにより、大きなコンテンツサイズと高帯域を処理する場合にスループットが大幅に改善されます。

ファイルアップロード機能に加えて、`TransferManager` クラスは、進行中のマルチパートアップロードを中止するメソッドを提供します。`Date` 値を指定すると、API は、指定した日付の前に開始されたすべてのマルチパートアップロードを中止します。

ファイルのアップロード

以下のタスクは、高レベル Java クラスを使用して、ファイルをアップロードする手順を示しています。この API には、データのアップロードを容易にするための、複数のバリエーション (*オーバーロード*) の `upload` メソッドが用意されています。

高レベル API のファイルアップロードプロセス

1	AWS 認証情報を指定して、`TransferManager` クラスのインスタンスを作成します。
2	ファイルとストリームのいずれからデータをアップロードするかに応じて、いずれかの `TransferManager.upload` オーバーロードを実行します。

以下の Java コード例は、前述のタスクの例です。

```
AWSCredentials myCredentials = new BasicAWSCredentials(myAccessKeyID,
mySecretKey);
```

```
TransferManager tm = new TransferManager(myCredentials);
// Asynchronous call.
Upload upload = tm.upload(existingBucketName, keyName, new File(filePath));
```

Example

次の Java コード例では、ファイルを Amazon S3 バケットにアップロードします。実際に動作するコードを作成、テストする方法については、「Java コード例のテスト (p. 477)」を参照してください。

```
import java.io.File;

import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.Upload;

public class HighLevel_Java_UploadFile {

    public static void main(String[] args) throws Exception {
        String existingBucketName = "*** Provide existing bucket name ***";
        String keyName            = "*** Provide object key ***";
        String filePath           = "*** Provide file to upload ***";

        TransferManager tm = new TransferManager(new PropertiesCredentials(
          HighLevel_Java_UploadFile.class.getResourceAsStream(
                                "AwsCredentials.properties")));

        // TransferManager processes all transfers asynchronously,
        // so this call will return immediately.
        Upload upload = tm.upload(
          existingBucketName, keyName, new File(filePath));

        try {
         // Or you can block and wait for the upload to finish
         upload.waitForCompletion();
        } catch (AmazonClientException amazonClientException) {
         System.out.println("Unable to upload file, upload was aborted.");
         amazonClientException.printStackTrace();
        }
    }
}
```

マルチパートアップロードの中止

`TransferManager` クラスは、進行中のマルチパートアップロードを中止するメソッド `abortMultipartUploads` を提供します。いったんアップロードを開始すると、そのアップロードを完了または中止するまで進行中と見なされます。`Date` 値を指定すると、この API はそのバケットで、指定した `Date` の前に開始された進行中のすべてのマルチパートアップロードを中止します。

アップロードされたパートと関連するすべてのストレージに対して課金される (「マルチパートアップロードと料金 (p. 151)」を参照) ため、マルチパートアップロードを完了してオブジェクトの作成を完了するか、またはマルチパートアップロードを中止してアップロードされたすべてのパートを削除することが重要です。

以下のタスクは、高レベル Java クラスを使用して、マルチパートアップロードを中止する手順を示しています。

高レベル API のマルチパートアップロード中止プロセス

1	AWS 認証情報を指定して、`TransferManager` クラスのインスタンスを作成します。
2	バケット名および `Date` 値を渡して、`TransferManager.abortMultipartUploads` メソッドを実行します。

以下の Java コード例は、前述のタスクを実装したものです。

```
AWSCredentials myCredentials = new BasicAWSCredentials(myAccessKeyID,
mySecretKey);

TransferManager tm = new TransferManager(myCredentials);
tm.abortMultipartUploads(existingBucketName, someDate);
```

Example

次の Java コードでは、1 週間以上前に特定のバケットで開始された進行中のすべてのマルチパートアップロードを中止します。作業サンプルを作成およびテストする方法については、「Java コード例のテスト (p. 477)」を参照してください。

```
import java.util.Date;

import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.transfer.TransferManager;

public class HighLevel_Java_AbortMultipartUploads {

    public static void main(String[] args) throws Exception {
        String existingBucketName = "*** Provide existing bucket name ***";

        TransferManager tm = new TransferManager(new PropertiesCredentials(
          HighLevel_Java_UploadFile.class.getResourceAsStream(
                          "AwsCredentials.properties")));

        int sevenDays = 1000 * 60 * 60 * 24 * 7;
  Date oneWeekAgo = new Date(System.currentTimeMillis() - sevenDays);

        try {
         tm.abortMultipartUploads(existingBucketName, oneWeekAgo);
        } catch (AmazonClientException amazonClientException) {
         System.out.println("Unable to upload file, upload was aborted.");
         amazonClientException.printStackTrace();
        }
    }
}
```

Note

特定のマルチパートアップロードを中止することもできます。詳細については、「マルチパートアップロードの中止 (p. 188)」を参照してください。

マルチパートアップロードの進行状況の追跡

高レベルマルチパートアップロード API は、`TransferManager` クラスを使用してデータをアップロードするときに、アップロードの進行状況を追跡するリッスンインターフェイス `ProgressListener` を提供します。コードでイベントを使用するには、`com.amazonaws.services.s3.model.ProgressEvent` および `com.amazonaws.services.s3.model.ProgressListener` をインポートする必要があります。

進行状況イベントが定期的に発生し、バイトが転送されたことをリスナーに通知します。

以下の Java コード例は、`ProgressEvent` イベントを登録し、ハンドラを記述する方法を示しています。

```
AWSCredentials myCredentials = new BasicAWSCredentials(myAccessKeyID,
mySecretKey);

TransferManager tm = new TransferManager(myCredentials);

PutObjectRequest request = new PutObjectRequest(
    existingBucketName, keyName, new File(filePath));

// Subscribe to the event and provide event handler.
request.setProgressListener(new ProgressListener() {
   public void progressChanged(ProgressEvent event) {
    System.out.println("Transferred bytes: " +
      event.getBytesTransfered());
            }
});
```

Example

以下の Java コードは、ファイルをアップロードし、ProgressListener を使用してアップロードの進行状況を追跡します。作業サンプルを作成およびテストする方法については、「Java コード例のテスト (p. 477)」を参照してください。

```
import java.io.File;

import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.model.ProgressEvent;
import com.amazonaws.services.s3.model.ProgressListener;
import com.amazonaws.services.s3.model.PutObjectRequest;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.Upload;

public class HighLevel_Java_UploadFile {

    public static void main(String[] args) throws Exception {
        String existingBucketName = "*** Provide bucket name ***";
        String keyName            = "*** Provide object key ***";
        String filePath           = "** Provide file to upload ***";

        TransferManager tm = new TransferManager(new PropertiesCredentials(
         HighLevel_Java_UploadFile.class.getResourceAsStream(
                                "AwsCredentials.properties")));

        // For more advanced uploads, you can create a request object
        // and supply additional request parameters (ex: progress listeners,
        // canned ACLs, etc)
        PutObjectRequest request = new PutObjectRequest(
          existingBucketName, keyName, new File(filePath));

        // You can ask the upload for its progress, or you can
        // add a ProgressListener to your request to receive notifications
        // when bytes are transfered.
        request.setProgressListener(new ProgressListener() {
   public void progressChanged(ProgressEvent event) {
    System.out.println("Transferred bytes: " +
      event.getBytesTransfered());
   }
  });

        // TransferManager processes all transfers asynchronously,
        // so this call will return immediately.
        Upload upload = tm.upload(request);

        try {
         // You can block and wait for the upload to finish
         upload.waitForCompletion();
        } catch (AmazonClientException amazonClientException) {
         System.out.println("Unable to upload file, upload aborted.");
         amazonClientException.printStackTrace();
        }
    }
}
```

低レベル Java API を使用したマルチパートアップロード

Topics

AWS SDK for Java は、Amazon S3 REST API に非常によく似たマルチパートアップロード用の低レベル API を公開しています (「マルチパートアップロード API を使用したオブジェクトのアップロード (p. 178)」を参照)。マルチパートアップロードを一時停止して再開する必要がある場合、アップロード中にパートサイズを変更する場合、またはデータのサイズが事前にわからない場合は、低レベル API を使用します。これらの要件を満たさない場合は、常に高レベル API (「高レベル Java API を使用したマルチパートアップロード (p. 179)」を参照) を使用します。

ファイルのアップロード

以下のタスクは、低レベル Java クラスを使用して、ファイルをアップロードする手順を示しています。

低レベル API のファイルアップロードプロセス

1	AWS 認証情報を指定して、`AmazonS3Client` クラスのインスタンスを作成します。
2	`AmazonS3Client.initiateMultipartUpload` メソッドを実行して、マルチパートアップロードを開始します。`InitiateMultipartUploadRequest` クラスのインスタンスを作成して、マルチパートアップロードを開始するのに必要な情報 (バケット名とキー名) を指定する必要があります。
3	`AmazonS3Client.initiateMultipartUpload` メソッドが返すアップロード ID を保存します。以降、マルチパートアップロードオペレーションのたびに、このアップロード ID を指定する必要があります。
4	パートをアップロードします。パートアップロードごとに、`AmazonS3Client.uploadPart` メソッドを実行します。アップロード ID、バケット名、パート番号などのパートアップロード情報を指定する必要があります。この情報は、`UploadPartRequest` クラスのインスタンスを作成することによって指定します。
5	`AmazonS3Client.uploadPart` メソッドのレスポンスをリストに保存します。このレスポンスには、マルチパートアップロードを完了するのに必要になる ETag 値およびパート番号が含まれています。
6	各パートについてタスク 4 と 5 を繰り返します。
7	`AmazonS3Client.completeMultipartUpload` メソッドを実行して、マルチパートアップロードを完了します。

以下の Java コード例は、前述のタスクの例です。

```
AmazonS3 s3Client = new AmazonS3Client(new PropertiesCredentials(
             LowLevel_Java_UploadFile.class.getResourceAsStream(
                                          "AwsCredentials.properties")));

// Create a list of UploadPartResponse objects. You get one of these for
// each part upload.
List<PartETag> partETags = new ArrayList<PartETag>();
```

```
// Step 1: Initialize.
InitiateMultipartUploadRequest initRequest = new InitiateMultipartUploadRequest(

                                                    existingBucketName, keyName);
InitiateMultipartUploadResult initResponse =
                               s3Client.initiateMultipartUpload(initRequest);

File file = new File(filePath);
long contentLength = file.length();
long partSize = 5 * 1024 * 1024; // Set part size to 5 MB.

try {
    // Step 2: Upload parts.
    long filePosition = 0;
    for (int i = 1; filePosition < contentLength; i++) {
        // Last part can be less than 5 MB. Adjust part size.
     partSize = Math.min(partSize, (contentLength - filePosition));

        // Create request to upload a part.
        UploadPartRequest uploadRequest = new UploadPartRequest()
            .withBucketName(existingBucketName).withKey(keyName)
            .withUploadId(initResponse.getUploadId()).withPartNumber(i)
            .withFileOffset(filePosition)
            .withFile(file)
            .withPartSize(partSize);

        // Upload part and add response to our list.
        partETags.add(s3Client.uploadPart(uploadRequest).getPartETag());

        filePosition += partSize;
    }

    // Step 3: complete.
    CompleteMultipartUploadRequest compRequest = new
                CompleteMultipartUploadRequest(existingBucketName,
                                               keyName,
                                               initResponse.getUploadId(),
                                               partETags);

    s3Client.completeMultipartUpload(compRequest);
} catch (Exception e) {
    s3Client.abortMultipartUpload(new AbortMultipartUploadRequest(
              existingBucketName, keyName, initResponse.getUploadId()));
}
```

Example

次の Java コード例では、ファイルを Amazon S3 バケットにアップロードします。作業サンプルを作成およびテストする方法については、「Java コード例のテスト (p. 477)」を参照してください。

```
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.AbortMultipartUploadRequest;
import com.amazonaws.services.s3.model.CompleteMultipartUploadRequest;
import com.amazonaws.services.s3.model.InitiateMultipartUploadRequest;
import com.amazonaws.services.s3.model.InitiateMultipartUploadResult;
import com.amazonaws.services.s3.model.ListMultipartUploadsRequest;
import com.amazonaws.services.s3.model.ListPartsRequest;
import com.amazonaws.services.s3.model.MultipartUploadListing;
import com.amazonaws.services.s3.model.PartETag;
import com.amazonaws.services.s3.model.PartListing;
import com.amazonaws.services.s3.model.UploadPartRequest;

public class LowLevel_Java_UploadFile {

    public static void main(String[] args) throws IOException {
        String existingBucketName="*** Provide-Your-Existing-BucketName ***";

        String keyName             = "*** Provide-Key-Name ***";
        String filePath            = "*** Provide-File-Path ***";

        AmazonS3 s3Client = new AmazonS3Client(new PropertiesCredentials(
          LowLevel_Java_UploadFile.class.getResourceAsStream(
                            "AwsCredentials.properties")));

        // Create a list of UploadPartResponse objects. You get one of these
        // for each part upload.
        List<PartETag> partETags = new ArrayList<PartETag>();

        // Step 1: Initialize.
        InitiateMultipartUploadRequest initRequest = new
             InitiateMultipartUploadRequest(existingBucketName, keyName);
        InitiateMultipartUploadResult initResponse =
                           s3Client.initiateMultipartUpload(initRequest);

        File file = new File(filePath);
        long contentLength = file.length();
        long partSize = 5242880; // Set part size to 5 MB.

        try {
            // Step 2: Upload parts.
            long filePosition = 0;
            for (int i = 1; filePosition < contentLength; i++) {
                // Last part can be less than 5 MB. Adjust part size.
             partSize = Math.min(partSize, (contentLength - filePosition));

                // Create request to upload a part.
```

```
                UploadPartRequest uploadRequest = new UploadPartRequest()
                    .withBucketName(existingBucketName).withKey(keyName)
                   .withUploadId(initResponse.getUploadId()).withPartNumber(i)

                    .withFileOffset(filePosition)
                    .withFile(file)
                    .withPartSize(partSize);

                // Upload part and add response to our list.
                partETags.add(
                  s3Client.uploadPart(uploadRequest).getPartETag());

                filePosition += partSize;
            }

            // Step 3: complete.
            CompleteMultipartUploadRequest compRequest = new
                        CompleteMultipartUploadRequest(
                                    existingBucketName,
                                    keyName,
                                    initResponse.getUploadId(),
                                    partETags);

            s3Client.completeMultipartUpload(compRequest);
        } catch (Exception e) {
            s3Client.abortMultipartUpload(new AbortMultipartUploadRequest(
                    existingBucketName, keyName, initResponse.getUploadId()));

        }
    }
}
```

マルチパートアップロードのリスト

以下のタスクは、低レベル Java クラスを使用して、バケットで進行中のすべてのマルチパートアップロードをリストする手順を示しています。

低レベル API のマルチパートアップロードリストプロセス

1	`ListMultipartUploadsRequest` クラスのインスタンスを作成し、バケット名を指定します。
2	`AmazonS3Client.listMultipartUploads` メソッドを実行します。このメソッドは、進行中のマルチパートアップロードに関する情報を提供する `MultipartUploadListing` クラスのインスタンスを返します。

以下の Java コード例は、前述のタスクを実装したものです。

```
ListMultipartUploadsRequest allMultpartUploadsRequest =
     new ListMultipartUploadsRequest(existingBucketName);
MultipartUploadListing multipartUploadListing =
     s3Client.listMultipartUploads(allMultpartUploadsRequest);
```

マルチパートアップロードの中止

AmazonS3.abortMultipartUpload メソッドを呼び出して、進行中のマルチパートアップロードを中止することができます。このメソッドは、Amazon S3 にアップロードされたすべてのパートを削除し、リソースを解放します。アップロード ID、バケット名、およびキー名を指定する必要があります。以下の Java コード例は、進行中のマルチパートアップロードを中止する方法を示しています。

```
AWSCredentials myCredentials = new
               BasicAWSCredentials(myAccessKeyID, mySecretKey);

InitiateMultipartUploadRequest initRequest =
    new InitiateMultipartUploadRequest(existingBucketName, keyName);
InitiateMultipartUploadResult initResponse =
               s3Client.initiateMultipartUpload(initRequest);

AmazonS3 s3Client = new AmazonS3Client(myCredentials);
s3Client.abortMultipartUpload(new AbortMultipartUploadRequest(
            existingBucketName, keyName, initResponse.getUploadId()));
```

Note

特定のマルチパートアップロードではなく、特定の時刻の前に開始された進行中のすべてのマルチパートアップロードを中止することができます。このクリーンアップオペレーションは、開始したが完了または中止していない古いマルチパートアップロードを中止する場合に有用です。詳細については、「マルチパートアップロードの中止 (p. 180)」を参照してください。

AWS SDK for .NET を使用したマルチパートアップロード

Topics

AWS SDK の使用 (「AWS dynamo と Explorer の使用 (p. 475)」を参照) で説明しているように、AWS SDK for .NET はマルチパートアップロード API をサポートしています (「マルチパートアップロード API を使用したオブジェクトのアップロード (p. 178)」を参照)。このセクションでは、高レベルおよび低レベル SDK for Java API を使用して、オブジェクトをパート単位でアップロードする例を示します。

高レベル .NET API を使用したマルチパートアップロード

Topics

AWS SDK for .NET では、マルチパートアップロードを簡素化する高レベル API を公開しています (「マルチパートアップロード API を使用したオブジェクトのアップロード (p. 178)」を参照)。ファイル、ディレクトリ、またはストリームからデータをアップロードできます。ファイルからデータをアップロードするときにオブジェクトのキー名を指定しない場合、API はオブジェクトのキー名にファイル名を使用します。ストリームからデータをアップロードする場合は、オブジェクトのキー名を指定する必要があります。必要に応じて、マルチパートアップロードに使用するパートサイズ、パートを同時にアップロードする際に使用するスレッドの数、省略可能なファイルメタデータ、ストレージクラス (STANDARD や REDUCED_REDUNDANCY)、ACL など、詳細なオプションを設定することができます。高レベル API は、これらの詳細なオプションを設定する `TransferUtilityUploadRequest` クラスを提供します。

`TransferUtility` クラスは、進行中のマルチパートアップロードを中止するメソッドを提供します。`DateTime` 値を指定すると、API は、指定した日時の前に開始されたすべてのマルチパートアップロードを中止します。

ファイルのアップロード

以下のタスクは、高レベル .NET クラスを使用して、ファイルをアップロードする手順を示しています。この API には、データのアップロードを容易にするための、複数のバリエーション (*オーバーロード*) の `Upload` メソッドが用意されています。

高レベル API のファイルアップロードプロセス

1	AWS 認証情報を指定して、`TransferUtility` クラスのインスタンスを作成します。
2	ファイル、ストリーム、またはディレクトリのいずれからデータをアップロードするかに応じて、いずれかの `TransferUtility.Upload` オーバーロードを実行します。

以下の C# コード例は、前述のタスクを実装したものです。

```
TransferUtility utility = new TransferUtility(accessKeyID, secretAccessKey);
utility.Upload(filePath, existingBucketName);
```

.NET API を使用して大きなファイルをアップロードする場合、データがリクエストストリームに書き込まれる間にも、タイムアウトが発生することがあります。以下の C# コード例に示すように、`TransferUtilityConfig.DefaultTimeout` を使用して明示的なタイムアウトを設定することができます。

```
TransferUtilityConfig config = new TransferUtilityConfig();
config.DefaultTimeout = 11111;
TransferUtility utility = new TransferUtility(accessKeyID, secretAccessKey,
config);
```

Example

以下の C# コード例では、ファイルを Amazon S3 バケットにアップロードしています。この例は、さまざまな `TransferUtility.Upload` オーバーロードを使用してファイルをアップロードする方法を示しています。後続のアップロード呼び出しが行われるたびに、前のアップロードが置き換えられます。作業サンプルを作成およびテストする方法については、「.NET コード例のテスト (p. 478)」を参照してください。

```
using System;
using System.Configuration;
using Amazon.S3;
using Amazon.S3.Model;
using System.Collections.Specialized;
using Amazon.S3.Transfer;
using System.IO;

namespace s3.amazon.com.docsamples.highlevel_uploadfile
{
    class Program
    {
        static string accessKeyID     = "";
        static string secretAccessKey = "";

        static string existingBucketName = "*** Provide bucket name ***";
        static string keyName        = "*** Provide your object key ***";
        static string filePath       = "*** Provide file name ***";

        static void Main(string[] args)
        {

            NameValueCollection appConfig = ConfigurationManager.AppSettings;
            accessKeyID     = appConfig["AWSAccessKey"];
            secretAccessKey = appConfig["AWSSecretKey"];

            try
            {
                TransferUtility fileTransferUtility = new
                    TransferUtility(accessKeyID, secretAccessKey);

                // 1. Upload a file, file name is used as the object key name.

                fileTransferUtility.Upload(filePath, existingBucketName);
                Console.WriteLine("Upload 1 completed");

                // 2. Specify object key name explicitly.
                fileTransferUtility.Upload(filePath,
                                           existingBucketName, keyName);
                Console.WriteLine("Upload 2 completed");

                // 3. Upload data from a type of System.IO.Stream.
                using (FileStream fileToUpload =
                    new FileStream(filePath, FileMode.Open, FileAccess.Read))
                {
                    fileTransferUtility.Upload(fileToUpload,
                                               existingBucketName, keyName);
                }
                Console.WriteLine("Upload 3 completed");
```

```
                // 4.// Specify advanced settings/options.
                TransferUtilityUploadRequest fileTransferUtilityRequest =
                    new TransferUtilityUploadRequest()
                    .WithBucketName(existingBucketName)
                    .WithFilePath(filePath)
                    .WithStorageClass(S3StorageClass.ReducedRedundancy)
                    .WithMetadata("param1", "Value1")
                    .WithMetadata("param2", "Value2")
                    .WithPartSize(6291456) // This is 6 MB.
                    .WithKey(keyName)
                    .WithCannedACL(S3CannedACL.PublicRead);
                fileTransferUtility.Upload(fileTransferUtilityRequest);
                Console.WriteLine("Upload 4 completed");
            }
            catch (AmazonS3Exception s3Exception)
            {
                Console.WriteLine(s3Exception.Message,
                                  s3Exception.InnerException);
            }
        }
    }
}
```

ディレクトリのアップロード

TransferUtility クラスを使用して、ディレクトリ全体をアップロードすることもできます。デフォルトでは、Amazon S3 は、指定したディレクトリのルートに存在するファイルのみをアップロードします。ただし、すべてのサブディレクトリでファイルを再帰的にアップロードするように指定することができます。

また、指定したディレクトリで、何らかのフィルタ基準に基づいてファイルを選択するフィルタ式を指定することもできます。例えば、ディレクトリから .pdf ファイルのみをアップロードするには、「*.pdf」フィルタ式を指定します。

ディレクトリからファイルをアップロードする場合は、オブジェクトのキー名を指定することはできません。この場合のキー名は、ディレクトリ内のファイルの口ケーションおよびその名前から作成されます。例えば、以下の構造の c:\myfolder ディレクトリがあるとします。

```
C:\myfolder
      \a.txt
      \b.pdf
      \media\
             An.mp3
```

このディレクトリをアップロードする際、 Amazon S3 では次のキー名が使用されます。

```
a.txt
b.pdf
media/An.mp3
```

以下のタスクは、高レベル .NET クラスを使用して、ディレクトリをアップロードする手順を示しています。

高レベル API のディレクトリアップロードプロセス

1	AWS 認証情報を指定して、TransferUtility クラスのインスタンスを作成します。

2	いずれかの `TransferUtility.UploadDirectory` オーバーロードを実行します。

以下の C# コード例は、前述のタスクを実装したものです。

```
TransferUtility utility = new TransferUtility(accessKeyID, secretAccessKey);
utility.UploadDirectory(directoryPath, existingBucketName);
```

Example

以下の C# コード例では、ディレクトリを Amazon S3 バケットにアップロードしています。この例は、さまざまな TransferUtility.UploadDirectory オーバーロードを使用してディレクトリをアップロードする方法を示しています。後続のアップロード呼び出しが行われるたびに、前のアップロードが置き換えられます。作業サンプルを作成およびテストする方法については、「.NET コード例のテスト (p. 478)」を参照してください。

```
using System;
using System.Configuration;
using Amazon.S3;
using Amazon.S3.Model;
using System.Collections.Specialized;
using Amazon.S3.Transfer;
using System.IO;

namespace s3.amazon.com.docsamples.highlevel_uploadDirectory
{
    class Program
    {
        static string accessKeyID        = "";
        static string secretAccessKey    = "";

        static string existingBucketName = "*** Provide bucket name ***";
        static string directoryPath = "*** Provide directory name ***";

        static void Main(string[] args)
        {
            NameValueCollection appConfig = ConfigurationManager.AppSettings;
            accessKeyID        = appConfig["AWSAccessKey"];
            secretAccessKey    = appConfig["AWSSecretKey"];

             try
            {
                TransferUtility directoryTransferUtility =
                    new TransferUtility(accessKeyID, secretAccessKey);

                // 1. Upload a directory.
                directoryTransferUtility.UploadDirectory(directoryPath,
                                                         existingBucketName);
                Console.WriteLine("Upload statement 1 completed");

                // 2. Upload only the .txt files from a directory.
                //    Also, search recursively.
                directoryTransferUtility.UploadDirectory(
                                             directoryPath,
                                             existingBucketName,
                                             "*.txt",
                                             SearchOption.AllDirectories);
                Console.WriteLine("Upload statement 2 completed");

                // 3. Same as 2 and some optional configuration
                //    Search recursively for .txt files to upload).
                TransferUtilityUploadDirectoryRequest trUtilDirUpReq =
                    new TransferUtilityUploadDirectoryRequest()
                .WithBucketName(existingBucketName)
                .WithDirectory(directoryPath)
                .WithSearchOption(SearchOption.AllDirectories)
```

```
                .WithSearchPattern("*.txt");

                directoryTransferUtility.UploadDirectory(trUtilDirUpReq);
                Console.WriteLine("Upload statement 3 completed");
            }

            catch (AmazonS3Exception e)
            {
                Console.WriteLine(e.Message, e.InnerException);
            }
        }
    }
}
```

マルチパートアップロードの中止

`TransferUtility` クラスは、進行中のマルチパートアップロードを中止するメソッド `AbortMultipartUploads` を提供します。いったんアップロードを開始すると、そのアップロードを完了または中止するまで進行中とみなされます。`DateTime` 値を指定すると、この API はそのバケットで、指定した `DateTime` の前に開始された進行中のすべてのマルチパートアップロードを中止します。

アップロードされたパートと関連するすべてのストレージに対して課金される (「マルチパートアップロードと料金 (p. 151)」を参照) ため、マルチパートアップロードを完了してオブジェクトの作成を完了するか、またはマルチパートアップロードを中止してアップロードされたすべてのパートを削除することが重要です。

以下のタスクは、高レベル .NET クラスを使用して、マルチパートアップロードを中止する手順を示しています。

高レベル API のマルチパートアップロード中止プロセス

1	AWS 認証情報を指定して、`TransferUtility` クラスのインスタンスを作成します。
2	バケット名および `DateTime` 値を渡して、`TransferUtility.AbortMultipartUploads` メソッドを実行します。

以下の C# コード例は、前述のタスクを実装したものです。

```
TransferUtility utility = new TransferUtility(accessKeyID, secretAccessKey);
utility.AbortMultipartUploads(existingBucketName, DateTime.Now.AddDays(-7));
```

Example

次の C# コードでは、1 週間以上前に特定のバケットで開始された進行中のすべてのマルチパートアップロードを中止します。作業サンプルを作成およびテストする方法については、「.NET コード例のテスト (p. 478)」を参照してください。

```
using System;
using System.Configuration;
using Amazon.S3;
using Amazon.S3.Model;
using System.Collections.Specialized;
using Amazon.S3.Transfer;

namespace s3.amazon.com.docsamples.highlevel_abortmultipartupload
{
    class Program
    {
        static string accessKeyID        = "";
        static string secretAccessKey    = "";

        static string existingBucketName = "***Provide bucket name***";

        static void Main(string[] args)
        {
            NameValueCollection appConfig = ConfigurationManager.AppSettings;
            accessKeyID        = appConfig["AWSAccessKey"];
            secretAccessKey    = appConfig["AWSSecretKey"];

            try
            {
                TransferUtility transferUtility =
                    new TransferUtility(accessKeyID, secretAccessKey);
                // Aborting uploads that were initiated over a week ago.
                transferUtility.AbortMultipartUploads(
                    existingBucketName, DateTime.Now.AddDays(-7));
            }

            catch (AmazonS3Exception e)
            {
                Console.WriteLine(e.Message, e.InnerException);
            }
        }
    }
}
```

Note

特定のマルチパートアップロードを中止することもできます。詳細については、「マルチパートアップロードのリスト (p. 203)」を参照してください。

マルチパートアップロードの進行状況の追跡

高レベルマルチパートアップロード API は、`TransferUtility` クラスを使用してデータをアップロードするときに、アップロードの進行状況を追跡するイベント `TransferUtilityUploadRequest.UploadProgressEvent` を提供します。

イベントが定期的に発生し、転送されるバイト総数、イベントの発生時に転送されたバイト数など、マルチパートアップロードの進行状況の情報を返します。

以下の C# コード例は、UploadProgressEvent イベントを登録し、ハンドラを記述する方法を示しています。

```
TransferUtility fileTransferUtility =
        new TransferUtility(accessKeyID, secretAccessKey);

// Use TransferUtilityUploadRequest to configure options.
// In this example we subscribe to an event.
TransferUtilityUploadRequest uploadRequest =
               new TransferUtilityUploadRequest()
               .WithBucketName(existingBucketName)
               .WithFilePath(filePath);

uploadRequest.UploadProgressEvent +=
         new EventHandler<UploadProgressArgs>(uploadRequest_UploadPartProgres
sEvent);

fileTransferUtility.Upload(uploadRequest);

static void uploadRequest_UploadPartProgressEvent(object sender, UploadProgres
sArgs e)
{
    // Process event.
    Console.WriteLine("{0}/{1}", e.TransferredBytes, e.TotalBytes);
}
```

Example

以下の C# コード例では、ファイルを Amazon S3 バケットにアップロードし、`TransferUtilityUploadRequest.UploadProgressEvent` イベントを登録することで進行状況を追跡しています。作業サンプルを作成およびテストする方法については、「.NET コード例のテスト (p. 478)」を参照してください。

```
using System;
using System.Configuration;
using Amazon.S3;
using Amazon.S3.Model;
using System.Collections.Specialized;
using Amazon.S3.Transfer;
using System.IO;


namespace s3.amazon.com.docsamples.highlevel_trackprogress
{
    class Program
    {
        static string accessKeyID        = "";
        static string secretAccessKey    = "";

        static string existingBucketName = "*** Provide bucket name ***";
        static string keyName            = "*** Provide key name ***";
        static string filePath        = "*** Provide file to upload ***";

        static void Main(string[] args)
        {

            NameValueCollection appConfig = ConfigurationManager.AppSettings;
            accessKeyID        = appConfig["AWSAccessKey"];
            secretAccessKey    = appConfig["AWSSecretKey"];

            try
            {
                TransferUtility fileTransferUtility =
                    new TransferUtility(accessKeyID, secretAccessKey);

                // Use TransferUtilityUploadRequest to configure options.
                // In this example we subscribe to an event.
                TransferUtilityUploadRequest uploadRequest =
                    new TransferUtilityUploadRequest()
                    .WithBucketName(existingBucketName)
                    .WithFilePath(filePath);

                uploadRequest.UploadProgressEvent +=
                    new EventHandler<UploadProgressArgs>
                        (uploadRequest_UploadPartProgressEvent);

                fileTransferUtility.Upload(uploadRequest);
                Console.WriteLine("Upload completed");
            }

            catch (AmazonS3Exception e)
            {
                Console.WriteLine(e.Message, e.InnerException);
            }
```

```
        }

        static void uploadRequest_UploadPartProgressEvent(
            object sender, UploadProgressArgs e)
        {
            // Process event.
            Console.WriteLine("{0}/{1}", e.TransferredBytes, e.TotalBytes);
        }
    }
}
```

低レベル .NET API を使用したマルチパートアップロード

Topics

- ファイルのアップロード (p. 199)
- マルチパートアップロードのリスト (p. 203)
- マルチパートアップロードの進行状況の追跡 (p. 204)
- マルチパートアップロードの中止 (p. 204)

AWS SDK for .NET は、Amazon S3 REST API に非常によく似たマルチパートアップロード用の低レベル API を公開しています (「REST API を使用したマルチパートアップロード (p. 214)」を参照) 。マルチパートアップロードを一時停止して再開する必要がある場合、アップロード中にパートサイズを変更する場合、またはデータのサイズが事前にわからない場合は、低レベル API を使用します。これらの要件を満たさない場合は、常に高レベル API (「高レベル .NET API を使用したマルチパートアップロード (p. 189)」を参照) を使用します。

ファイルのアップロード

以下のタスクは、低レベル .NET クラスを使用して、ファイルをアップロードする手順を示しています。

低レベル API のファイルアップロードプロセス

1	AWS 認証情報を指定して、`AmazonS3Client` クラスのインスタンスを作成します。
2	`AmazonS3Client.InitiateMultipartUpload` メソッドを実行して、マルチパートアップロードを開始します。`InitiateMultipartUploadRequest` クラスのインスタンスを作成して、マルチパートアップロードを開始するのに必要な情報を指定する必要があります。
3	`AmazonS3Client.InitiateMultipartUpload` メソッドが返すアップロード ID を保存します。以降、マルチパートアップロードオペレーションのたびに、このアップロード ID を指定する必要があります。
4	パートをアップロードします。各パートアップロードに、`AmazonS3Client.UploadPart` メソッドを実行します。アップロード ID、バケット名、パート番号などのパートアップロード情報を指定する必要があります。この情報は、`UploadPartRequest` クラスのインスタンスを作成することによって指定します。
5	`AmazonS3Client.UploadPart` メソッドのレスポンスをリストに保存します。このレスポンスには、マルチパートアップロードを完了するのに後で必要になる ETag 値およびパート番号が含まれています。
6	各パートについてタスク 4 と 5 を繰り返します。

7	AmazonS3Client.CompleteMultipartUpload メソッドを実行して、マルチパートアップロードを完了します。

以下の C# コード例は、前述のタスクの例です。

```
AmazonS3 s3Client = new AmazonS3Client(AccessKeyID, SecretAccessKey);

// List to store upload part responses.
List<UploadPartResponse> uploadResponses = new List<UploadPartResponse>();

// 1. Initialize.
InitiateMultipartUploadRequest initRequest =
    new InitiateMultipartUploadRequest()
    .WithBucketName(existingBucketName)
    .WithKey(keyName);

InitiateMultipartUploadResponse initResponse =
    s3Client.InitiateMultipartUpload(initRequest);

// 2. Upload Parts.
long contentLength = new FileInfo(filePath).Length;
long partSize = 5242880; // 5 MB

try
{
    long filePosition = 0;
    for (int i = 1; filePosition < contentLength; i++)
    {

        // Create request to upload a part.
        UploadPartRequest uploadRequest = new UploadPartRequest()
            .WithBucketName(existingBucketName)
            .WithKey(keyName)
            .WithUploadId(initResponse.UploadId)
            .WithPartNumber(i)
            .WithPartSize(partSize)
            .WithFilePosition(filePosition)
            .WithFilePath(filePath);

        // Upload part and add response to our list.
         uploadResponses.Add(s3Client.UploadPart(uploadRequest));

        filePosition += partSize;
    }

    // Step 3: complete.
    CompleteMultipartUploadRequest compRequest =
        new CompleteMultipartUploadRequest()
        .WithBucketName(existingBucketName)
        .WithKey(keyName)
        .WithUploadId(initResponse.UploadId)
        .WithPartETags(uploadResponses);

    CompleteMultipartUploadResponse completeUploadResponse =
        s3Client.CompleteMultipartUpload(compRequest);
```

```
}
catch (Exception exception)
{
    Console.WriteLine("Exception occurred: {0}", exception.Message);
    s3Client.AbortMultipartUpload(new AbortMultipartUploadRequest()
        .WithBucketName(existingBucketName)
        .WithKey(keyName)
        .WithUploadId(initResponse.UploadId));
}
```

Note

.NET API を使用して大きなオブジェクトをアップロードする場合、データがリクエストストリームに書き込まれる間にも、タイムアウトが発生することがあります。`UploadPartRequest` を使用して、明示的なタイムアウトを設定することができます。

Example

以下の C# コード例では、ファイルを Amazon S3 バケットにアップロードしています。作業サンプルを作成およびテストする方法については、「.NET コード例のテスト (p. 478)」を参照してください。

```
using System;
using System.Collections.Generic;
using System.Collections.Specialized;
using System.Configuration;
using System.IO;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples.LowLevel_UploadFromFile
{
    class Program
    {
        // Your AWS Credentials.
        static string AccessKeyID        = "";
        static string SecretAccessKey    = "";

        static string existingBucketName = "*** Provide bucket name";
        static string keyName = "*** Provide object key ***";
        static string filePath = "*** Provide file to upload ***";

         static void Main(string[] args)
        {
            NameValueCollection appConfig = ConfigurationManager.AppSettings;
            AccessKeyID        = appConfig["AWSAccessKey"];
            SecretAccessKey    = appConfig["AWSSecretKey"];

            AmazonS3 s3Client =
                new AmazonS3Client(AccessKeyID, SecretAccessKey);

            // List to store upload part responses.
            List<UploadPartResponse> uploadResponses =
                new List<UploadPartResponse>();

            // 1. Initialize.
            InitiateMultipartUploadRequest initiateRequest =
                new InitiateMultipartUploadRequest()
                .WithBucketName(existingBucketName)
                .WithKey(keyName);

            InitiateMultipartUploadResponse initResponse =
                s3Client.InitiateMultipartUpload(initiateRequest);

            // 2. Upload Parts.
            long contentLength = new FileInfo(filePath).Length;
            long partSize = 5 * (long)Math.Pow(2, 20); // 5 MB

            try
            {
                long filePosition = 0;
                for (int i = 1; filePosition < contentLength; i++)
                {

                    // Create request to upload a part.
```

```
                    UploadPartRequest uploadRequest = new UploadPartRequest()
                        .WithBucketName(existingBucketName)
                        .WithKey(keyName)
                        .WithUploadId(initResponse.UploadId)
                        .WithPartNumber(i)
                        .WithPartSize(partSize)
                        .WithFilePosition(filePosition)
                        .WithFilePath(filePath);

                    // Upload part and add response to our list.
                    uploadResponses.Add(s3Client.UploadPart(uploadRequest));

                    filePosition += partSize;
                }

                // Step 3: complete.
                CompleteMultipartUploadRequest completeRequest =
                    new CompleteMultipartUploadRequest()
                    .WithBucketName(existingBucketName)
                    .WithKey(keyName)
                    .WithUploadId(initResponse.UploadId)
                    .WithPartETags(uploadResponses);

                CompleteMultipartUploadResponse completeUploadResponse =
                    s3Client.CompleteMultipartUpload(completeRequest);

            }
            catch (Exception exception)
            {
              Console.WriteLine("Exception occurred: {0}", exception.Message);

               s3Client.AbortMultipartUpload(new AbortMultipartUploadRequest()

                    .WithBucketName(existingBucketName)
                    .WithKey(keyName)
                    .WithUploadId(initResponse.UploadId));
            }
        }
    }
}
```

マルチパートアップロードのリスト

以下のタスクは、低レベル .NET クラスを使用して、バケットで進行中のすべてのマルチパートアップロードをリストする手順を示しています。

低レベル API のマルチパートアップロードリストプロセス

1	`ListMultipartUploadsRequest` クラスのインスタンスを作成し、バケット名を指定します。
2	`AmazonS3Client.ListMultipartUploads` メソッドを実行します。このメソッドは `ListMultipartUploadsResponse` クラスのインスタンスを返し、進行中のマルチパートアップロードに関する情報を提供します。

以下の C# コード例は、前述のタスクを実装したものです。

```
ListMultipartUploadsRequest allMultipartUploadsRequest = new ListMultipartUp
loadsRequest()
   .WithBucketName(existingBucketName);
ListMultipartUploadsResponse mpUploadsResponse = s3Client.ListMultipartUp
loads(allMultipartUploadsRequest);
```

マルチパートアップロードの進行状況の追跡

低レベルマルチパートアップロード API は、アップロードの進行状況を追跡するイベント UploadPartRequest.UploadPartProgressEvent を提供します。

イベントが定期的に発生し、転送されるバイト総数、イベントの発生時に転送されたバイト数など、マルチパートアップロードの進行状況の情報を返します。

以下の C# コード例は、UploadPartProgressEvent イベントを登録し、ハンドラを記述する方法を示しています。

```
UploadPartRequest uploadRequest = new UploadPartRequest();
// Provide request data (for example, bucket name, key name and part number).
// ...
// Subscribe to the event.
uploadRequest.UploadPartProgressEvent += new
EventHandler<UploadPartProgressArgs>(uploadRequest_UploadPartProgressEvent);

// Sample event handler.
static void uploadRequest_UploadPartProgressEvent(object sender,
                                                  UploadPartProgressArgs e)
{
    // Process event.
    Console.WriteLine("{0}/{1}", e.TransferredBytes, e.TotalBytes);
}
```

マルチパートアップロードの中止

AmazonS3Client.AbortMultipartUpload メソッドを呼び出して、進行中のマルチパートアップロードを中止することができます。このメソッドは、S3 にアップロードされたすべてのパートを削除し、リソースを解放します。アップロード ID、バケット名、およびキー名を指定する必要があります。以下の C# コード例は、進行中のマルチパートアップロードを中止する方法を示しています。

```
s3Client.AbortMultipartUpload(new AbortMultipartUploadRequest()
    .WithBucketName(existingBucketName)
    .WithKey(keyName)
    .WithUploadId(uploadID));
```

Note

特定のマルチパートアップロードではなく、特定の時刻の前に開始された進行中のすべてのマルチパートアップロードを中止することができます。このクリーンアップオペレーションは、開始したが完了または中止していない古いマルチパートアップロードを中止する場合に有用です。詳細については、「マルチパートアップロードの中止 (p. 195)」を参照してください。

AWS SDK for PHP を使用したマルチパートアップロード

Topics

AWS SDK for PHP はマルチパートアップロード API (「マルチパートアップロード API を使用したオブジェクトのアップロード (p. 178)」を参照) をサポートしています。

高レベルの抽象化

AWS SDK for PHP の高レベルの抽象化により、マルチパートアップロードのフローが簡素化されます。数行のコードだけで、ファイルを Amazon S3 にアップロードすることができます。単純なファイルアップロードを行う場合には、この方法をお勧めします。

低レベル API

AWS SDK for PHP の低レベル API は、マルチパートアップロード REST オペレーションに対応しています (「REST API を使用したマルチパートアップロード (p. 214)」を参照)。マルチパートアップロードを一時停止して再開する必要がある場合、アップロード中にパートサイズを変更する場合、またはデータのサイズが事前にわからない場合は、低レベル API を使用します。これらの要件を満たさない場合は、高レベル API を使用します。

AWS SDK for PHP の高レベルの抽象化を使用したマルチパートアップロード

Amazon S3 を使用すると、大きなファイルを複数のパートに分けてアップロードできます。マルチパートアップロードは、5 GB より大きいファイルに対してのみ使用できます。AWS SDK for PHP では、マルチパートアップロードを簡素化する高レベルの Aws\S3\Model\MultipartUpload\UploadBuilder クラスを公開しています。

`Aws\S3\Model\MultipartUpload\UploadBuilder` クラスは単純なマルチパートアップロードでの使用に最も適しています。マルチパートアップロードを一時停止して再開する必要がある場合、アップロード中にパートサイズを変更する場合、またはデータのサイズが事前にわからない場合は、低レベルの PHP API を使用する必要があります。詳細については、「AWS SDK for PHP の低レベル API を使用したマルチパートアップロード (p. 208)」を参照してください。

マルチパートアップロードの詳細については、「マルチパートアップロード API を使用したオブジェクトのアップロード (p. 178)」を参照してください。サイズが 5 GB 未満のファイルのアップロードについては、「AWS SDK for PHP を使用したオブジェクトのアップロード (p. 174)」を参照してください。

高レベルのマルチパートアップロードを使用したファイルのアップロード

このトピックでは、AWS SDK for PHP の高レベルの `Aws\S3\Model\MultipartUpload\UploadBuilder` クラスを使用してマルチパートファイルアップロードを行う手順を示します。

Note

このトピックでは、既に AWS SDK for PHP の使用と PHP サンプルの実行 (p. 479) の説明が実行されていて、AWS SDK for PHP が正しくインストールされていることを前提としています。

高レベルのマルチパートファイルアップロード

1	自分の AWS 認証情報または IAM ユーザー認証情報と共に Aws\S3\S3Client クラスの factory() メソッドを使用して、Amazon S3 クライアントのインスタンスを作成します。

2	Amazon S3 `Aws\S3\Model\MultipartUpload\UploadBuilder` クラスの newInstance() メソッドを使用して、UploadBuilder のインスタンスを作成します。このメソッドは Aws\Common\Model\MultipartUpload\AbstractUploadBuilder クラスから継承されたものです。UploadBuilder オブジェクトに対して、setClient()、setBucket()、setKey() メソッドを使用して、クライアント、バケット名、キー名を設定します。setSource() メソッドを使用して、アップロードするファイルのパスと名前を設定します。
3	`UploadBuilder` オブジェクトの build() メソッドを実行することで、設定した UploadBuilder オプションに基づいて適切なアップローダー転送オブジェクトを作成します。(転送オブジェクトは Aws\S3\Model\MultipartUpload\AbstractTransfer クラスのサブクラスに属するようになります。)
4	作成された転送オブジェクトの `upload()` メソッドを実行することで、アップロードを実行します。

以下の PHP コード例は、高レベルの `UploadBuilder` オブジェクトを使用してファイルをアップロードする方法を示しています。

```
use Aws\Common\Exception\MultipartUploadException;
use Aws\S3\Model\MultipartUpload\UploadBuilder;
use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

// Instantiate the client.
$s3 = S3Client::factory(array(
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));

// Prepare the upload parameters.
$uploader = UploadBuilder::newInstance()
    ->setClient($s3)
    ->setSource('/path/to/large/file.mov')
    ->setBucket($bucket)
    ->setKey($keyname)
    ->build();

// Perform the upload. Abort the upload if something goes wrong.
try {
    $uploader->upload();
    echo "Upload complete.\n";
} catch (MultipartUploadException $e) {
    $uploader->abort();
    echo "Upload failed.\n";
    echo $e->getMessage() . "\n";
}
```

Example （高レベルの UploadBuilder を使用した、ファイルの Amazon S3 バケットへのマルチパートアップロード）

以下の PHP 例では、ファイルを Amazon S3 バケットにアップロードしています。この例は、UploadBuilder オブジェクトに詳細なオプションを設定する方法を 示しています。具体的には、setMinPartSize() メソッドを使用して、マルチパートアップロードに使用するパートのサイズを設定すると共に、setOption() メソッドを使用して、オプションのファイルメタデータやアクセスコントロールリスト（ACL）を設定しています。

このほかに、この例では、UploadBuilder オブジェクトの setConcurrency() メソッドを使用して並行処理オプションを設定することで、ファイルのパートを並行してアップロードする方法も示しています。この例で作成する転送オブジェクトは、3 つのパートを並列にアップロードすることで、最終的にファイル全体をアップロードします。PHP 例の実行については、このガイド内の「PHP サンプルの実行 (p. 480)」を参照してください。

```
<?php

// Include the AWS SDK using the Composer autoloader.
require 'vendor/autoload.php';

use Aws\Common\Exception\MultipartUploadException;
use Aws\S3\Model\MultipartUpload\UploadBuilder;
use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

// Instantiate the client.
$s3 = S3Client::factory(array(
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));

// Prepare the upload parameters.
$uploader = UploadBuilder::newInstance()
    ->setClient($s3)
    ->setSource('/path/to/large/file.mov')
    ->setBucket($bucket)
    ->setKey($keyname)
    ->setMinPartSize(25 * 1024 * 1024)
    ->setOption('Metadata', array(
        'param1' => 'value1',
        'param2' => 'value2'
    ))
    ->setOption('ACL', 'public-read')
    ->setConcurrency(3)
    ->build();

// Perform the upload. Abort the upload if something goes wrong.
try {
    $uploader->upload();
    echo "Upload complete.\n";
} catch (MultipartUploadException $e) {
    $uploader->abort();
    echo "Upload failed.\n";
    echo $e->getMessage() . "\n";
}
```

関連リソース

- 「AWS SDK for PHP Aws\Common\Model\MultipartUpload\AbstractUploadBuilder クラス」
- 「AWS SDK for PHP Aws\Common\Model\MultipartUpload\AbstractUploadBuilder::newInstance() メソッド」
- AWS SDK for PHP Aws\Common\Model\MultipartUpload\AbstractUploadBuilder::SetSource() Method
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\Model\MultipartUpload\UploadBuilder クラス」
- 「Aws\S3\Model\MultipartUpload\UploadBuilder::build() メソッド」
- 「Aws\S3\Model\MultipartUpload\UploadBuilder:setMinPartSize() メソッド」
- 「Aws\S3\Model\MultipartUpload\UploadBuilder:setOption() メソッド」
- 「Aws\S3\Model\MultipartUpload\UploadBuilder:setConcurrency() メソッド」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client クラス」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::factory() メソッド」
- 「AWS SDK for PHP for Amazon S3 – Uploading large files using multipart uploads」
- 「AWS SDK for PHP – Amazon S3」
- 「AWS SDK for PHP」のドキュメント

AWS SDK for PHP の低レベル API を使用したマルチパートアップロード

Topics

AWS SDK for PHP は、Amazon S3 REST API に非常によく似たマルチパートアップロード用の低レベル API を公開しています (「REST API を使用したマルチパートアップロード (p. 214)」を参照)。マルチパートアップロードを一時停止して再開する必要がある場合、アップロード中にパートサイズを変更する場合、またはデータのサイズが事前にわからない場合は、低レベル API を使用します。これらの要件を満たさない場合は、必ず AWS SDK for PHP の高レベル抽象化 (「AWS SDK for PHP の高レベルの抽象化を使用したマルチパートアップロード (p. 205)」を参照) を使用します。

PHP SDK の低レベル API を使用した、複数のパートに分けたファイルのアップロード

このトピックでは、AWS SDK for PHP のクラスの低レベルマルチパートアップロードを使用して、ファイルを複数のパートに分けてアップロードする手順を示します。

Note

このトピックでは、既に AWS SDK for PHP の使用と PHP サンプルの実行 (p. 479) の説明が実行されていて、AWS SDK for PHP が正しくインストールされていることを前提としています。

PHP SDK の低レベル API のマルチパートファイルアップロードプロセス

1	自分の AWS 認証情報または IAM ユーザー認証情報と共に Aws\S3\S3Client クラスの factory() メソッドを使用して、Amazon S3 クライアントのインスタンスを作成します。
2	Aws\S3\S3Client::createMultipartUpload() メソッドを実行して、マルチパートアップロードを開始します。`array` パラメータの必須キー、`Bucket` と `Key` に、バケット名とキー名を指定する必要があります。 レスポンス本文から `UploadID` を取得、保存します。`UploadID` は、以後のすべてのマルチパートアップロードオペレーションで使用されます。

3	ファイルをパートに分けてアップロードします。それには、各ファイルパートに対して Aws\S3\S3Client::uploadPart() メソッドを実行し、各ファイルパートの終わりに到達するまで待ちます。upload_part() で必須とされる array のパラメータキーは、Bucket、Key、UploadId、PartNumber です。次の各ファイルパートをアップロードするには、upload_part() への次の各呼び出しに対して、PartNumber キーの引数として渡す値を 1 だけ増やす必要があります。 各 upload_part() メソッド呼び出しのレスポンスを配列内に保存します。各レスポンスには、マルチパートアップロードを完了するのに後で必要になる ETag 値が含まれています。
4	Aws\S3\S3Client::completeMultipartUpload() メソッドを実行して、マルチパートアップロードを完了します。completeMultipartUpload() で必須とされる array のパラメータ は、Bucket、Key、UploadId です。

以下の PHP コード例は、PHP SDK の低レベル API によるファイルのマルチパートアップロードの手順を示しています。

```
use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';
$filename = '*** Path to and Name of the File to Upload ***';

// 1. Instantiate the client.
$s3 = S3Client::factory(array(
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));

// 2. Create a new multipart upload and get the upload ID.
$response = $s3->createMultipartUpload(array(
    'Bucket' => $bucket,
    'Key'    => $keyname
);
$uploadId = $result['UploadId'];

// 3. Upload the file in parts.
$file = fopen($filename, 'r');
$parts = array();
$partNumber = 1;
while (!feof($file)) {
    $result = $s3->uploadPart(array(
        'Bucket'     => $bucket,
        'Key'        => $key,
        'UploadId'   => $uploadId,
        'PartNumber' => $partNumber,
        'Body'       => fread($file, 5 * 1024 * 1024),
    ));
    $parts[] = array(
        'PartNumber' => $partNumber++,
        'ETag'       => $result['ETag'],
    );
}

// 4. Complete multipart upload.
```

```
$result = $s3->completeMultipartUpload(array(
    'Bucket'   => $bucket,
    'Key'      => $key,
    'UploadId' => $uploadId,
    'Parts'    => $parts,
));
$url = $result['Location'];

fclose($file);
```

Example（低レベルのマルチパートアップロードの PHP SDK API を使用した Amazon S3 バケットへのファイルのアップロード）

以下の PHP コード例では、低レベルの PHP API、マルチパートアップロードを使用して、Amazon S3 バケットにファイルをアップロードしています。PHP 例の実行については、このガイド内の「PHP サンプルの実行 (p. 480)」を参照してください。

```
<?php

// Include the AWS SDK using the Composer autoloader
require 'vendor/autoload.php';

use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';
$filename = '*** Path to and Name of the File to Upload ***';

// 1. Instantiate the client.
$s3 = S3Client::factory(array(
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));

// 2. Create a new multipart upload and get the upload ID.
$result = $s3->createMultipartUpload(array(
    'Bucket'            => $bucket,
    'Key'                  => $keyname,
    'StorageClass' => 'REDUCED_REDUNDANCY',
    'ACL'                 => 'public-read',
    'Metadata'    => array(
        'param1' => 'value 1',
        'param2' => 'value 2',
        'param3' => 'value 3'
    )
));
$uploadId = $result['UploadId'];

// 3. Upload the file in parts.
try {
    $file = fopen($filename, 'r');
    $parts = array();
    $partNumber = 1;
    while (!feof($file)) {
        $result = $s3->uploadPart(array(
            'Bucket'     => $bucket,
            'Key'        => $keyname,
            'UploadId'   => $uploadId,
            'PartNumber' => $partNumber,
            'Body'       => fread($file, 5 * 1024 * 1024),
        ));
        $parts[] = array(
            'PartNumber' => $partNumber++,
            'ETag'       => $result['ETag'],
        );

        echo "Uploading part {$partNumber} of {$filename}.\n";
    }
```

```
    fclose($file);
} catch (S3Exception $e) {
    $result = $s3->abortMultipartUpload(array(
        'Bucket'   => $bucket,
        'Key'      => $keyname,
        'UploadId' => $uploadId
    ));

    echo "Upload of {$filename} failed.\n";
}

// 4. Complete multipart upload.
$result = $s3->completeMultipartUpload(array(
    'Bucket'   => $bucket,
    'Key'      => $keyname,
    'UploadId' => $uploadId,
    'Parts'    => $parts,
));
$url = $result['Location'];

echo "Uploaded {$filename} to {$url}.\n";
```

関連リソース

- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client クラス」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::factory() メソッド」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::createMultipartUpload() メソッド」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::uploadPart() メソッド」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::completeMultipartUpload() メソッド」
- 「AWS SDK for PHP – Amazon S3」
- 「AWS SDK for PHP」のドキュメント

PHP SDK の低レベル API を使用したマルチパートアップロードのリスト

このトピックでは、AWS SDK for PHP の低レベル API クラスを使用して、バケットで進行中の全マルチパートアップロードをリストする手順を示します。

Note

このトピックでは、既に AWS SDK for PHP の使用と PHP サンプルの実行 (p. 479) の説明が実行されていて、AWS SDK for PHP が正しくインストールされていることを前提としています。

PHP SDK の低レベル API のマルチパートアップロードリストプロセス

1	自分の AWS 認証情報または IAM ユーザー認証情報と共に Aws\S3\S3Client クラスの factory() メソッドを使用して、Amazon S3 クライアントのインスタンスを作成します。
2	バケット名を指定して、Aws\S3\S3Client::listMultipartUploads() メソッドを実行します。このメソッドは、指定されたバケットで進行中のすべてのマルチパートアップロードを返します。

以下は、バケットで進行中の全マルチパートアップロードをリストする PHP コードの実例です。

```
use Aws\S3\S3Client;

$s3 = S3Client::factory(array(
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));

$bucket = '*** Your Bucket Name ***';

$result = $s3->listMultipartUploads(array('Bucket' => $bucket));

print_r($result->toArray());
```

関連リソース

- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client クラス」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::factory() メソッド」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::listMultipartUploads() メソッド」
- 「AWS SDK for PHP – Amazon S3」
- 「AWS SDK for PHP」のドキュメント

マルチパートアップロードの中止

このトピックでは、AWS SDK for PHP のクラスを使用して、進行中のマルチパートアップロードを中止する手順を示します。

Note

このトピックでは、既に AWS SDK for PHP の使用と PHP サンプルの実行 (p. 479) の説明が実行されていて、AWS SDK for PHP が正しくインストールされていることを前提としています。

マルチパートアップロードの中止

1	自分の AWS 認証情報または IAM ユーザー認証情報と共に Aws\S3\S3Client クラスの factory() メソッドを使用して、Amazon S3 クライアントのインスタンスを作成します。
2	Aws\S3\S3Client::abortMultipartUpload() メソッドを実行します。`array` パラメータで必要とされるキー、`Bucket` と `KeyUploadId` に、バケット名とキー名を指定する必要があります。 `abortMultipartUpload()` メソッドは、Amazon S3 にアップロードされたすべてのパートを削除し、リソースを解放します。

Example （ マルチパートアップロードの中止 ）

以下の PHP コード例は、進行中のマルチパートアップロードを中止する方法を示しています。この例は、`abortMultipartUpload()` メソッドの使い方を示しています。PHP 例の実行については、このガイド内の「PHP サンプルの実行 (p. 480)」を参照してください。

```
<?php

// Include the AWS SDK using the Composer autoloader.
require 'vendor/autoload.php';

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

// Instantiate the client.
$s3 = S3Client::factory(array(
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));

// Abort the multipart upload.
$s3->abortMultipartUpload(array(
    'Bucket'   => $bucket,
    'Key'      => $keyname,
    'UploadId' => 'VXBsb2FkIElExampleBlbHZpbmcncyBtExamplepZS5tMnRzIHVwbG9hZ',
));

?>
```

関連リソース

- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client クラス」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::factory() メソッド」
- 「AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::abortMultipartUpload() メソッド」
- 「AWS SDK for PHP – Amazon S3」
- 「AWS SDK for PHP」のドキュメント

REST API を使用したマルチパートアップロード

マルチパートアップロードを行うための REST API については、『Amazon Simple Storage Service API リファレンス』の以下のセクションを参照してください。

- Initiate Multipart Upload
- Upload Part
- Complete Multipart Upload
- マルチパートアップロードの中止
- パートのリスト
- マルチパートアップロードのリスト

これらの API を使用して独自の REST リクエストを作成するか、Amazon 提供の SDK のいずれかを使用することができます。SDK の詳細については、「マルチパートアップロードの API サポート (p. 152)」を参照してください。

署名付き URL を使用したオブジェクトのアップロード

Topics

署名付き URL の作成者がそのオブジェクトへのアクセス許可を持っている場合、署名付き URL を使用して、URL で識別されるオブジェクトにアクセスすることができます。つまり、オブジェクトをアップロードするための署名付き URL を受け取った場合、オブジェクトをアップロードすることができるのは、署名付き URL の作成者がそのオブジェクトをアップロードするのに必要な権限を持っている場合のみです。

デフォルトでは、すべてのオブジェクトおよびバケットがプライベートです。署名付き URL は、ユーザー/顧客が特定のオブジェクトをバケットにアップロードできるようにする必要があるが、AWS セキュリティ認証情報またはアクセス許可を持つことを求めない場合に有用です。署名付き URL を作成するときは、セキュリティ認証情報を提供し、バケット名、オブジェクトキー、HTTP メソッド (オブジェクトのアップロードの PUT) 、および有効期限の日時を指定する必要があります。署名付き URL は、指定した期間のみ有効です。

AWS SDK for Java または AWS SDK for .NET のプログラムを使用して、署名付き URL を生成することができます。Visual Studio を使用する場合は、AWS Explorer を使用して、コードを記述せずに署名付きオブジェクト URL を生成することもできます。有効な署名付き URL を受け取ったすべてのユーザーが、オブジェクトをプログラムでアップロードすることができます。

詳細については、「Using Amazon S3 from AWS Explorer」を参照してください。

AWS Explorer をインストールする方法については、「AWS dynamo と Explorer の使用 (p. 475)」を参照してください。

Note

有効なセキュリティ認証情報を持つすべてのユーザーが、署名付き URL を作成できます。ただし、オブジェクトを正常にアップロードするには、署名付き URL の基となるオペレーションを実行する権限を持つユーザーが、署名付き URL を作成する必要があります。

署名付き URL を使用したオブジェクトのアップロード – AWS SDK for Java

以下のタスクは、Java クラスで、署名付き URL を使用して、オブジェクトをアップロードする手順を示しています。

オブジェクトのアップロード

1	AWS 認証情報を指定して、`AmazonS3` クラスのインスタンスを作成します。これらの認証情報は、署名付き URL を生成するときに、認証用の署名を作成するのに使用されます。
2	`AmazonS3.generatePresignedUrl` メソッドを実行して、署名付き URL を生成します。 `GeneratePresignedUrlRequest` クラスのインスタンスを作成して、バケット名、オブジェクトキー、および有効期日を指定します。この URL をオブジェクトのアップロードに使用する場合は、この URL の作成時に HTTP 動詞 PUT を指定する必要があります。

3	署名付き URL を利用できるすべてのユーザーが、オブジェクトをアップロードすることができます。 アップロードにより、オブジェクトが作成されるか、または署名付き URL で指定されたものと同じキーを持つ既存のオブジェクトが置き換えられます。

以下の Java コード例は、前述のタスクの例です。

```
AWSCredentials myCredentials = new BasicAWSCredentials(
                                 myAccessKeyID, mySecretKey);
AmazonS3 s3Client = new AmazonS3Client(myCredentials);

java.util.Date expiration = new java.util.Date();
long msec = expiration.getTime();
msec += 1000 * 60 * 60; // Add 1 hour.
expiration.setTime(msec);

GeneratePresignedUrlRequest generatePresignedUrlRequest = new GeneratePresigned
UrlRequest(bucketName, objectKey);
generatePresignedUrlRequest.setMethod(HttpMethod.PUT);
generatePresignedUrlRequest.setExpiration(expiration);

URL s = s3client.generatePresignedUrl(generatePresignedUrlRequest);

// Use the pre-signed URL to upload an object.
```

Example

次の Java コード例では、署名付き URL を生成し、次に署名付き URL を使用して、サンプルデータをオブジェクトとしてアップロードします。作業サンプルを作成およびテストする方法については、「Java コード例のテスト (p. 477)」を参照してください。

```
import java.io.IOException;
import java.net.URL;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.HttpMethod;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.GeneratePresignedUrlRequest;
import java.net.*;
import java.io.*;

public class S3Sample {
 private static String bucketName = "ExampleBucket";
 private static String objectKey = "JavaObj.xml";

 public static void main(String[] args) throws IOException {
  AmazonS3 s3client = new AmazonS3Client(new PropertiesCredentials(
   S3Sample.class.getResourceAsStream("AwsCredentials.properties")));

  try {
   System.out.println("Generating pre-signed URL.");
   java.util.Date expiration = new java.util.Date();
   long milliSeconds = expiration.getTime();
   milliSeconds += 1000 * 60 * 60; // Add 1 hour.
   expiration.setTime(milliSeconds);

   GeneratePresignedUrlRequest generatePresignedUrlRequest =
        new GeneratePresignedUrlRequest(bucketName, objectKey);
   generatePresignedUrlRequest.setMethod(HttpMethod.PUT);
   generatePresignedUrlRequest.setExpiration(expiration);

   URL url = s3client.generatePresignedUrl(generatePresignedUrlRequest);

   UploadObject(url);

   System.out.println("Pre-Signed URL = " + url.toString());
  } catch (AmazonServiceException exception) {
   System.out.println("Caught an AmazonServiceException, " +
     "which means your request made it " +
     "to Amazon S3, but was rejected with an error response " +
   "for some reason.");
   System.out.println("Error Message: " + exception.getMessage());
   System.out.println("HTTP  Code: "    + exception.getStatusCode());
   System.out.println("AWS Error Code:" + exception.getErrorCode());
   System.out.println("Error Type:    " + exception.getErrorType());
   System.out.println("Request ID:    " + exception.getRequestId());
  } catch (AmazonClientException ace) {
   System.out.println("Caught an AmazonClientException, " +
     "which means the client encountered " +
     "an internal error while trying to communicate" +
```

```
    " with S3, " +
   "such as not being able to access the network.");
   System.out.println("Error Message: " + ace.getMessage());
  }
 }

 public static void UploadObject(URL url) throws IOException
 {
  HttpURLConnection connection=(HttpURLConnection) url.openConnection();
  connection.setDoOutput(true);
  connection.setRequestMethod("PUT");
  OutputStreamWriter out = new OutputStreamWriter(
    connection.getOutputStream());
  out.write("This text uploaded as object.");
  out.close();
  int responseCode = connection.getResponseCode();

 }
}
```

署名付き URL を使用したオブジェクトのアップロード – AWS SDK for .NET

以下のタスクは、.NET クラスで、署名付き URL を使用して、オブジェクトをアップロードする手順を示しています。

オブジェクトのアップロード

1	自分の AWS 証明書を指定して、AmazonS3 クラスのインスタンスを作成します。 これらの認証情報は、署名付き URL を生成するときに、認証用の署名を作成するのに使用されます。
2	AmazonS3.GetPreSignedURL メソッドを実行して、署名付き URL を生成します。 GetPreSignedUrlRequest クラスのインスタンスを作成して、バケット名、オブジェクトキー、および有効期日を指定します。この URL をオブジェクトのアップロードに使用する場合は、この URL の作成時に HTTP 動詞 PUT を指定する必要があります。
3	署名付き URL を利用できるすべてのユーザーが、オブジェクトをアップロードすることができます。署名付き URL を指定してオブジェクトをアップロードすることによって、HttpWebRequest クラスのインスタンスを作成することができます。

以下の C# コード例は、前述のタスクの例です。

```
static AmazonS3 client;
client = Amazon.AWSClientFactory.CreateAmazonS3Client(
                    accessKeyID, secretAccessKeyID);
// Generate a pre-signed URL.
GetPreSignedUrlRequest request = new GetPreSignedUrlRequest();
request.WithBucketName(bucketName);
request.WithKey(objectKey);
request.Verb = HttpVerb.PUT;
request.WithExpires(DateTime.Now.AddMinutes(5));
string url = null;
 url = s3Client.GetPreSignedURL(request);
```

```
// Upload a file using the pre-signed URL.
HttpWebRequest httpRequest = WebRequest.Create(url) as HttpWebRequest;
httpRequest.Method = "PUT";
using (Stream dataStream = httpRequest.GetRequestStream())
{
   // Upload object.
}

HttpWebResponse response = httpRequest.GetResponse() as HttpWebResponse;
```

Example

次の C# コード例では、特定のオブジェクトの署名付き URL を生成し、この URL を使用してファイルをアップロードします。作業サンプルを作成、テストする方法については、「.NET コード例のテスト (p. 478)」を参照してください。

```
using System;
using System.Configuration;
using System.Collections.Specialized;
using Amazon.S3;
using Amazon.S3.Model;
using System.Net;
using System.IO;

namespace s3.amazon.com.docsamples.putobjectusingpresignedurl
{
    class S3Sample
    {
        static AmazonS3 s3Client;
        // File to upload.
        static string filePath = @"*** Specify file to upload ***";
        // Information to generate pre-signed object URL.
        static string bucketName = "*** Provide bucket name ***";
        static string objectKey = "*** Provide object key for the new object
***";

        public static void Main(string[] args)
        {
            NameValueCollection appConfig = ConfigurationManager.AppSettings;

            string accessKeyID = appConfig["AWSAccessKey"];
            string secretAccessKeyID = appConfig["AWSSecretKey"];

            try
            {
               using (s3Client = Amazon.AWSClientFactory.CreateAmazonS3Client(

                     accessKeyID, secretAccessKeyID))
                {
                    string url = GeneratePreSignedURL();
                    UploadObject(url);

                }
            }
            catch (AmazonS3Exception amazonS3Exception)
            {
                if (amazonS3Exception.ErrorCode != null &&
                    (amazonS3Exception.ErrorCode.Equals("InvalidAccessKeyId")
                    ||
                    amazonS3Exception.ErrorCode.Equals("InvalidSecurity")))
                {
                    Console.WriteLine("Check the provided AWS Credentials.");
                    Console.WriteLine(
                    "To sign up for service, go to http://aws.amazon.com/s3");

                }
                else
                {
```

```
                    Console.WriteLine(
                     "Error occurred. Message:'{0}' when listing objects",
                     amazonS3Exception.Message);
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }


        static void UploadObject(string url)
        {
            HttpWebRequest httpRequest = WebRequest.Create(url) as HttpWebRe
quest;
            httpRequest.Method = "PUT";
            using (Stream dataStream = httpRequest.GetRequestStream())
            {
                byte[] buffer = new byte[8000];
                using (FileStream fileStream = new FileStream(filePath,
FileMode.Open, FileAccess.Read))
                {
                    int bytesRead = 0;
                    while ((bytesRead = fileStream.Read(buffer, 0, buf
fer.Length)) > 0)
                    {
                        dataStream.Write(buffer, 0, bytesRead);
                    }
                }
            }

            HttpWebResponse response = httpRequest.GetResponse() as HttpWebRe
sponse;
        }

        static string GeneratePreSignedURL()
        {
            GetPreSignedUrlRequest request = new GetPreSignedUrlRequest();
            request.WithBucketName(bucketName);
            request.WithKey(objectKey);
            request.Verb = HttpVerb.PUT;
            request.WithExpires(DateTime.Now.AddMinutes(5));
            string url = null;


                url = s3Client.GetPreSignedURL(request);

            return url;
        }
    }
}
```

署名付き URL を使用したオブジェクトのアップロード – AWS SDK for Ruby

AWS SDK for Ruby で署名付き URL を使用してオブジェクトをアップロードする方法については、「PresignedPost」を参照してください。

オブジェクトのコピー

Topics

コピーオペレーションは、Amazon S3 内に既に格納されているオブジェクトのコピーを作成することです。1 回のアトミックオペレーションでコピーできるオブジェクトのサイズは最大 5 GB です。5 GB を超えるオブジェクトをコピーする場合は、マルチパートアップロード API を使用する必要があります。copy オペレーションを使用すると、以下のことができます。

- オブジェクトの追加コピーを作成する
- オブジェクトをコピーし、元のオブジェクトを削除することで、オブジェクトの名前を変更する
- Amazon S3 のロケーション (us-west-1、EU など) 間でオブジェクトを移動する
- オブジェクトメタデータを変更する

  Amazon S3 の各オブジェクトにはメタデータがあります。メタデータは名前と値のペアのセットです。オブジェクトメタデータは、オブジェクトをアップロードするときに設定できます。オブジェクトのアップロード後にはオブジェクトメタデータは変更できません。オブジェクトメタデータを変更する唯一の方法は、オブジェクトのコピーを作成し、メタデータを設定することです。コピーオペレーションでは、コピー元と同じオブジェクトをコピー先に設定します。

各オブジェクトにメタデータがあります。システムメタデータの場合もあれば、ユーザー定義メタデータの場合もあります。ユーザーはオブジェクトに対して使用するストレージクラス設定など一部のシステムメタデータを制御し、サーバー側の暗号化を設定します。オブジェクトをコピーするときは、ユーザーが制御するシステムメタデータとユーザー定義メタデータもコピーされます。システムが制御するメタデータは Amazon S3 によってリセットされます。例えば、オブジェクトをコピーすると、コピーしたオブジェクトの作成日が Amazon S3 によってリセットされます。コピーリクエストの中でこのような値を設定する必要は一切ありません。

オブジェクトをコピーするときには、一部のメタデータ値を更新できます。例えば、コピー元のオブジェクトで標準ストレージを使用するように設定してある場合、コピー先のオブジェクトでは低冗長化ストレージを使用するように選択する場合があるかもしれません。また、コピー元のオブジェクトに存在するユーザー定義メタデータの一部を変更することもできます。オブジェクトのユーザー設定可能なメタデータ (システムまたはユーザー定義) をコピー時に更新する場合、メタデータ値の 1 つだけを変更する場合であっても、コピー元オブジェクトに存在するすべてのユーザー設定可能メタデータをリクエスト内で明示的に指定する必要があることにご注意ください。

オブジェクトメタデータの詳細については、「オブジェクトキーとメタデータ (p. 108)」を参照してください。

Note

ロケーション間でオブジェクトをコピーすると、帯域料金が発生します。

関連リソース

- AWS dynamo と Explorer の使用 (p. 475)

1 回のオペレーションでのオブジェクトのコピー

Topics

- AWS SDK for Java を使用したオブジェクトのコピー (p. 223)
- AWS SDK for .NET を使用したオブジェクトのコピー (p. 224)
- AWS SDK for PHP を使用したオブジェクトのコピー (p. 227)
- AWS SDK for Ruby を使用したオブジェクトのコピー (p. 231)
- REST API を使用した 1 つのオブジェクトのコピー (p. 232)

このセクションの例は、1 回のオペレーションで最大 5 GB のオブジェクトをコピーする方法を示しています。5 GB を超えるオブジェクトをコピーする場合は、マルチパートアップロード API を使用する必要があります。詳細については、「マルチパートアップロード API を使用したオブジェクトのコピー (p. 233)」を参照してください。

AWS SDK for Java を使用したオブジェクトのコピー

以下のタスクは、Java クラスを使用して、Amazon S3 のオブジェクトをコピーする手順を示しています。

オブジェクトのコピー

1	AWS 認証情報を指定して、`AmazonS3Client` クラスのインスタンスを作成します。
2	`AmazonS3Client.copyObject` メソッドのいずれかを実行します。コピー元のバケット名、コピー元のキー名、コピー先のバケット名、コピー先のキー名などのリクエスト情報を提供する必要があります。このような情報を提供するには、`CopyObjectRequest` クラスのインスタンスを作成するか、オプションで `AmazonS3Client.copyObject` メソッドを使用して情報を直接提供します。

以下の Java コード例は、前述のタスクを実装したものです。

```
AWSCredentials myCredentials = new BasicAWSCredentials(
                                  myAccessKeyID, mySecretKey);
AmazonS3 s3client = new AmazonS3Client(myCredentials);
s3client.copyObject(sourceBucketName, sourceKey,
                    destinationBucketName, destinationKey);
```

Example

次の Java コード例では、オブジェクトのコピーを作成します。異なるキーを使ってコピーしたオブジェクトが、同じコピー元バケットに保存されます。作業サンプルを作成およびテストする方法については、「Java コード例のテスト (p. 477)」を参照してください。

```
import java.io.IOException;
import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.CopyObjectRequest;

public class S3Sample {
 private static String bucketName     = "*** Provide bucket name ***";
 private static String key            = "*** Provide key ***  ";
 private static String destinationKey = "*** Provide dest. key ***";

    public static void main(String[] args) throws IOException {
        AmazonS3 s3client = new AmazonS3Client(new PropertiesCredentials(
                S3Sample.class.getResourceAsStream(
                  "AwsCredentials.properties")));
        try {
            // Copying object
            CopyObjectRequest copyObjRequest = new CopyObjectRequest(
              bucketName, key, bucketName, destinationKey);
            System.out.println("Copying object.");
            s3client.copyObject(copyObjRequest);

        } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException, " +
              "which means your request made it " +
              "to Amazon S3, but was rejected with an error " +
                    "response for some reason.");
            System.out.println("Error Message:    " + ase.getMessage());
            System.out.println("HTTP Status Code: " + ase.getStatusCode());
            System.out.println("AWS Error Code:   " + ase.getErrorCode());
            System.out.println("Error Type:       " + ase.getErrorType());
            System.out.println("Request ID:       " + ase.getRequestId());
        } catch (AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException, " +
              "which means the client encountered " +
                    "an internal error while trying to " +
                    " communicate with S3, " +
                    "such as not being able to access the network.");
            System.out.println("Error Message: " + ace.getMessage());
        }
    }
}
```

AWS SDK for .NET を使用したオブジェクトのコピー

以下のタスクは、高レベル .NET クラスを使用して、ファイルをアップロードする手順を示しています。この API には、データのアップロードを容易にするための、複数のバリエーション (*オーバーロード*) の `Upload` メソッドが用意されています。

オブジェクトのコピー

1	AWS 認証情報を指定して、AmazonS3 クラスのインスタンスを作成します。
2	いずれかの AmazonS3.CopyObject を実行します。コピー元のバケット、コピー元のキー名、コピー先のバケット、コピー先のキー名などの情報を提供する必要があります。この情報は、CopyObjectRequest クラスのインスタンスを作成することによって指定します。

以下の C# コード例は、前述のタスクの例です。

```
static AmazonS3 client;
client = Amazon.AWSClientFactory.CreateAmazonS3Client(
                    accessKeyID, secretAccessKeyID);

CopyObjectRequest request = new CopyObjectRequest();
request.SourceBucket = bucketName;
request.SourceKey = keyName;
request.DestinationBucket = bucketName;
request.DestinationKey = destKeyName;
S3Response response = client.CopyObject(request);
```

Example

次の C# コード例では、同じコピー元バケットにオブジェクトのコピーを作成します。この例は、AmazonS3.CopyObject メソッドの使い方を示しています。実際に動作するコードを作成、テストする方法については、「.NET コード例のテスト (p. 478)」を参照してください。

```
using System;
using System.Configuration;
using System.Collections.Specialized;

using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples.copyobject
{
    class S3Sample
    {
        static string bucketName = "*** Provide bucket name ***";
        static string keyName = "*** Provide key name ***";
        static string destKeyName = "*** Provide destination key name ***";
        static AmazonS3 client;

        public static void Main(string[] args)
        {
            if (checkRequiredFields())
            {
                NameValueCollection appConfig =
                    ConfigurationManager.AppSettings;

                string accessKeyID = appConfig["AWSAccessKey"];
                string secretAccessKeyID = appConfig["AWSSecretKey"];

                using (client = Amazon.AWSClientFactory.CreateAmazonS3Client(
                    accessKeyID, secretAccessKeyID))
                {
                    Console.WriteLine("Copying an object");
                    CopyingObject();
                }
            }

            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }

        static bool checkRequiredFields()
        {
            NameValueCollection appConfig = ConfigurationManager.AppSettings;

            if (string.IsNullOrEmpty(appConfig["AWSAccessKey"]))
            {
                Console.WriteLine(
                    "AWSAccessKey was not set in the App.config file.");
                return false;
            }
            if (string.IsNullOrEmpty(appConfig["AWSSecretKey"]))
            {
                Console.WriteLine(
                    "AWSSecretKey was not set in the App.config file.");
```

```
                return false;
            }
            if (string.IsNullOrEmpty(bucketName))
            {
                Console.WriteLine("The variable bucketName is not set.");
                return false;
            }
            if (string.IsNullOrEmpty(keyName))
            {
                Console.WriteLine("The variable keyName is not set.");
                return false;
            }

            return true;
        }

        static void CopyingObject()
        {
            try
            {
                // simple object put
                CopyObjectRequest request = new CopyObjectRequest();
                request.SourceBucket = bucketName;
                request.SourceKey = keyName;
                request.DestinationBucket = bucketName;
                request.DestinationKey = destKeyName;
                S3Response response = client.CopyObject(request);
                response.Dispose();
            }
            catch (AmazonS3Exception s3Exception)
            {
                Console.WriteLine(s3Exception.Message,
                                  s3Exception.InnerException);
            }
        }
    }
}
```

AWS SDK for PHP を使用したオブジェクトのコピー

このトピックでは、AWS SDK for PHP のクラスを使用して、Amazon S3 内の 1 つまたは複数のオブジェクトを、特定のバケットから別のバケットへ、または同じバケット内にコピーする手順を示します。

Note

このトピックでは、既に AWS SDK for PHP の使用と PHP サンプルの実行 (p. 479) の説明が実行されていて、AWS SDK for PHP が正しくインストールされていることを前提としています。

以下のタスクは、PHP SDK のクラスを使用して、Amazon S3 に既に保存されているオブジェクトをコピーする手順を示しています。

オブジェクトのコピー

1	自分の AWS 認証情報または IAM ユーザー認証情報と共に Aws\S3\S3Client クラスの factory() メソッドを使用して、Amazon S3 クライアントのインスタンスを作成します。

2	オブジェクトをコピーするため、Aws\S3\S3Client::copyObject() メソッドを実行します。コピー元のバケット、コピー元のキー名、コピー先のバケット、コピー先のキー名などの情報を指定する必要があります。

以下のタスクは、copyObject() メソッドを使用して、Amazon S3 に既に保存されているオブジェクトをコピーする手順を示しています。

```
use Aws\S3\S3Client;

$sourceBucket = '*** Your Source Bucket Name ***';
$sourceKeyname = '*** Your Source Object Key ***';
$targetBucket = '*** Your Target Bucket Name ***';
$targetKeyname = '*** Your Target Key Name ***';

// Instantiate the client.
$s3 = S3Client::factory(array(
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));

// Copy an object.
$s3->copyObject(array(
    'Bucket'     => $targetBucket,
    'Key'        => $targetKeyname,
    'CopySource' => "{$sourceBucket}/{$sourceKeyname}",
));
```

以下のタスクは、PHP のクラスを使用して、Amazon S3 内の 1 つのオブジェクトの複数のコピーを作成する手順を示しています。

オブジェクトのコピー

1	自分の AWS 認証情報または IAM ユーザー認証情報と共に Aws\S3\S3Client クラスの factory() メソッドを使用して、Amazon S3 クライアントのインスタンスを作成します。
2	オブジェクトの複数のコピーを作成するため、Guzzle\Service\Client クラスから継承された、Amazon S3 クライアントの getCommand() メソッドの呼び出しを連続的に実行します。最初の引数として CopyObject コマンドを指定し、2 番目の引数としてコピー元のバケット、コピー元のキー名、コピー先のバケット、コピー先のキー名を含む array を指定します。

以下の PHP コード例は、Amazon S3 に保存されているオブジェクトの複数のコピーを作成する手順を示しています。

```
use Aws\S3\S3Client;

$sourceBucket = '*** Your Source Bucket Name ***';
$sourceKeyname = '*** Your Source Object Key ***';
$targetBucket = '*** Your Target Bucket Name ***';
$targetKeyname = '*** Your Target Key Name ***';

// Instantiate the client.
$s3 = S3Client::factory(array(
```

```
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));

// Perform a batch of CopyObject operations.
$batch = array();
for ($i = 1; $i <= 3; $i++) {
    $batch[] = $s3->getCommand('CopyObject', array(
        'Bucket'     => $targetBucket,
        'Key'        => "{targetKeyname}-{$i}",
        'CopySource' => "{$sourceBucket}/{$sourceKeyname}",
    ));
}
try {
    $successful = $s3->execute($batch);
    $failed = array();
} catch (\Guzzle\Service\Exception\CommandTransferException $e) {
    $successful = $e->getSuccessfulCommands();
    $failed = $e->getFailedCommands();
}
```

Example （ Amazon S3 内でのオブジェクトのコピー ）

以下の PHP コードは、`copyObject()` メソッドを使用して Amazon S3 内の 1 つのオブジェクトをコピーすると共に、`getcommand()` メソッドにより `CopyObject` を連続的に呼び出してオブジェクトの複数コピーを作成する例を示しています。

```
<?php

// Include the AWS SDK using the Composer autoloader.
require 'vendor/autoload.php';

use Aws\S3\S3Client;

$sourceBucket = '*** Your Source Bucket Name ***';
$sourceKeyname = '*** Your Source Object Key ***';
$targetBucket = '*** Your Target Bucket Name ***';

// Instantiate the client.
$s3 = S3Client::factory(array(
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));

// Copy an object.
$s3->copyObject(array(
    'Bucket'     => $targetBucket,
    'Key'        => "{$sourceKeyname}-copy",
    'CopySource' => "{$sourceBucket}/{$sourceKeyname}",
));

// Perform a batch of CopyObject operations.
$batch = array();
for ($i = 1; $i <= 3; $i++) {
    $batch[] = $s3->getCommand('CopyObject', array(
        'Bucket'     => $targetBucket,
        'Key'        => "{$sourceKeyname}-copy-{$i}",
        'CopySource' => "{$sourceBucket}/{$sourceKeyname}",
    ));
}
try {
    $successful = $s3->execute($batch);
    $failed = array();
} catch (\Guzzle\Service\Exception\CommandTransferException $e) {
    $successful = $e->getSuccessfulCommands();
    $failed = $e->getFailedCommands();
}
```

関連リソース

- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client クラス」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::copyObject() メソッド」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::factory() メソッド」
- 「AWS SDK for PHP for Amazon S3 の Guzzle\Service\Client クラス」
- 「AWS SDK for PHP for Amazon S3 の Guzzle\Service\Client::getCommand() メソッド」
- 「AWS SDK for PHP – Amazon S3」
- 「AWS SDK for PHP」のドキュメント

AWS SDK for Ruby を使用したオブジェクトのコピー

以下のタスクは、Ruby のクラスを使用して、 Amazon S3 内のバケット間または同一バケット内でオブジェクトをコピーする手順を示しています。

オブジェクトのコピー

1	AWS 認証情報を指定して、`AWS::S3` クラスのインスタンスを作成します。
2	`AWS::S3::S3Object#copy_to` メソッドまたは `AWS::S3::S3Object#copy_from` メソッドを実行します。コピー元のバケット名、コピー元のキー名、コピー先のバケット名、コピー先のキー名などのリクエスト情報を提供する必要があります。

次の Ruby コード例は前述のタスクの例で、`#copy_to` メソッドを使用して、バケット間でオブジェクトをコピーしています。

```
s3 = AWS::S3.new

# Upload a file and set server-side encryption.
bucket1 = s3.buckets[source_bucket]
bucket2 = s3.buckets[target_bucket]
obj1 = bucket1.objects[source_key]
obj2 = bucket2.objects[target_key]

obj1.copy_to(obj2)
```

Example

次の Ruby スクリプト例では、`#copy_from` メソッドを使用してオブジェクトのコピーを作成します。異なるキーを使ってコピーしたオブジェクトが、同じコピー元バケットに保存されます。作業サンプルを作成およびテストする方法については、「AWS SDK for Ruby の使用 (p. 482)」を参照してください。

```
#!/usr/bin/env ruby

require 'rubygems'
require 'aws-sdk'

AWS.config(
  :access_key_id => '*** Provide access key ***',
  :secret_access_key => '*** Provide secret key ***'
)

bucket_name = '*** Provide bucket name ***'
source_key = '*** Provide source key ***'
target_key = '*** Provide target key ***'

# Get an instance of the S3 interface.
s3 = AWS::S3.new

# Copy the object.
s3.buckets[bucket_name].objects[target_key].copy_from(source_key)

puts "Copying file #{source_key} to #{target_key}."
```

REST API を使用した 1 つのオブジェクトのコピー

次の例では、REST を使用してオブジェクトをコピーする方法について説明します。REST API の詳細については、「PUT Object (Copy) 」を参照してください。

この例では、pacific バケットの flotsam オブジェクトを、atlantic バケットの jetsam オブジェクトにコピーし、そのメタデータを維持します。

```
PUT /jetsam HTTP/1.1
Host: atlantic.s3.amazonaws.com
x-amz-copy-source: /pacific/flotsam
Authorization: AWS AKIAIOSFODNN7EXAMPLE:ENoSbxYByFA0UGLZUqJN5EUnLDg=
Date: Wed, 20 Feb 2008 22:12:21 +0000
```

署名は次の情報から生成されています。

```
PUT\r\n
\r\n
\r\n
Wed, 20 Feb 2008 22:12:21 +0000\r\n

x-amz-copy-source:/pacific/flotsam\r\n
/atlantic/jetsam
```

Amazon S3 から、オブジェクトの ETag と最終変更日を示す次のレスポンスが返されます。

```
HTTP/1.1 200 OK
x-amz-id-2: Vyaxt7qEbzv34BnSu5hctyyNSlHTYZFMWK4FtzO+iX8JQNyaLdTshL0KxatbaOZt
x-amz-request-id: 6B13C3C5B34AF333
Date: Date: Wed, 20 Feb 2008 22:13:01 +0000

Content-Type: application/xml
Transfer-Encoding: chunked
Connection: close
Server: AmazonS3
<?xml version="1.0" encoding="UTF-8"?>

<CopyObjectResult>
   <LastModified>2008-02-20T22:13:01</LastModified>
   <ETag>"7e9c608af58950deeb370c98608ed097"</ETag>
</CopyObjectResult>
```

マルチパートアップロード API を使用したオブジェクトのコピー

Topics

このセクションの例は、マルチパートアップロード API を使用して 5 GB よりも大きいオブジェクトをコピーする方法を示しています。5 GB 未満のオブジェクトは 1 度の操作でコピーできます。詳細については、「1 回のオペレーションでのオブジェクトのコピー (p. 223)」を参照してください。

AWS SDK for Java を使用したオブジェクトのコピーマルチパートアップロード API

以下のタスクは、Java SDK を使用してコピー元ロケーションから別のロケーション (例えば、あるバケットから別のバケット) に Amazon &S3 オブジェクトをコピーする手順を示しています。ここに示すコードを使用すると、5 GB よりも大きいオブジェクトをコピーできます。5 GB 未満のオブジェクトについては、1 回のコピーオペレーションを使用してください (「AWS SDK for Java を使用したオブジェクトのコピー (p. 223)」を参照) 。

オブジェクトのコピー

1	AWS 認証情報を指定して、`AmazonS3Client` クラスのインスタンスを作成します。
2	`AmazonS3Client.initiateMultipartUpload` メソッドを実行してマルチパートコピーを開始します。`InitiateMultipartUploadRequest` のインスタンスを作成します。バケット名とキー名を指定する必要があります。
3	`AmazonS3Client.initiateMultipartUpload` メソッドから返されたレスポンスオブジェクトのアップロード ID を保存します。以降、マルチパートアップロードオペレーションのたびに、このアップロード ID を指定する必要があります。
4	すべてのパートをコピーします。パートのコピーごとに、`CopyPartRequest` クラスの新しいインスタンスを作成し、パート情報 (コピー元バケット、コピー先バケット、オブジェクトキー、アップロード ID、パートの先頭バイト、パートの最終バイト、パート番号) を指定します。
5	`CopyPartRequest` メソッドのレスポンスをリストに保存します。レスポンスには ETag 値とパート番号が含まれます。マルチパートアップロードを完了するにはパート番号が必要です。
6	各パートについてタスク 4 と 5 を繰り返します。
7	`AmazonS3Client.completeMultipartUpload` メソッドを実行してコピーを完了します。

以下の Java コード例は、前述のタスクの例です。

```
// Step 1: Create instance and provide credentials.
AmazonS3Client s3Client = new AmazonS3Client(new
```

```
    PropertiesCredentials(
       LowLevel_LargeObjectCopy.class.getResourceAsStream(
         "AwsCredentials.properties")));

// Create lists to hold copy responses
List<CopyPartResult> copyResponses =
        new ArrayList<CopyPartResult>();

// Step 2: Initialize
InitiateMultipartUploadRequest initiateRequest =
       new InitiateMultipartUploadRequest(targetBucketName, targetObjectKey);

InitiateMultipartUploadResult initResult =
        s3Client.initiateMultipartUpload(initiateRequest);

// Step 3: Save upload Id.
String uploadId = initResult.getUploadId();

try {

    // Get object size.
    GetObjectMetadataRequest metadataRequest =
     new GetObjectMetadataRequest(sourceBucketName, sourceObjectKey);

   ObjectMetadata metadataResult = s3Client.getObjectMetadata(metadataRequest);

    long objectSize = metadataResult.getContentLength(); // in bytes

     // Step 4. Copy parts.
    long partSize = 5 * (long)Math.pow(2.0, 20.0); // 5 MB
    long bytePosition = 0;
    for (int i = 1; bytePosition < objectSize; i++)
    {
        // Step 5. Save copy response.
     CopyPartRequest copyRequest = new CopyPartRequest()
           .withDestinationBucketName(targetBucketName)
           .withDestinationKey(targetObjectKey)
           .withSourceBucketName(sourceBucketName)
           .withSourceKey(sourceObjectKey)
           .withUploadId(initResult.getUploadId())
           .withFirstByte(bytePosition)
           .withLastByte(bytePosition + partSize -1 >= objectSize ? objectSize
 - 1 : bytePosition + partSize - 1)
           .withPartNumber(i);

        copyResponses.add(s3Client.copyPart(copyRequest));
        bytePosition += partSize;
    }
    // Step 7. Complete copy operation.
    CompleteMultipartUploadResult completeUploadResponse =
        s3Client.completeMultipartUpload(completeRequest);
} catch (Exception e) {
    System.out.println(e.getMessage());
}
```

Example

次の Java コード例では、ある Amazon S3 バケットから別のバケットにオブジェクトをコピーします。作業サンプルを作成およびテストする方法については、「Java コード例のテスト (p. 477)」を参照してください。

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.*;
import com.amazonaws.services.s3.model.*;

public class LowLevel_LargeObjectCopy {

    public static void main(String[] args) throws IOException {
        String sourceBucketName = "*** Source-Bucket-Name ***";
        String targetBucketName = "*** Target-Bucket-Name ***";
        String sourceObjectKey  = "*** Source-Object-Key ***";
        String targetObjectKey  = "*** Target-Object-Key ***";
        AmazonS3Client s3Client = new AmazonS3Client(new
          PropertiesCredentials(
            LowLevel_LargeObjectCopy.class.getResourceAsStream(
              "AwsCredentials.properties")));

        // List to store copy part responses.

        List<CopyPartResult> copyResponses =
                  new ArrayList<CopyPartResult>();

        InitiateMultipartUploadRequest initiateRequest =
        new InitiateMultipartUploadRequest(targetBucketName, targetObjectKey);


        InitiateMultipartUploadResult initResult =
         s3Client.initiateMultipartUpload(initiateRequest);

        try {
            // Get object size.
            GetObjectMetadataRequest metadataRequest =
             new GetObjectMetadataRequest(sourceBucketName, sourceObjectKey);

          ObjectMetadata metadataResult = s3Client.getObjectMetadata(metadataRe
quest);
            long objectSize = metadataResult.getContentLength(); // in bytes

            // Copy parts.
            long partSize = 5 * (long)Math.pow(2.0, 20.0); // 5 MB

            long bytePosition = 0;
            for (int i = 1; bytePosition < objectSize; i++)
            {
             CopyPartRequest copyRequest = new CopyPartRequest()
                   .withDestinationBucketName(targetBucketName)
                   .withDestinationKey(targetObjectKey)
                   .withSourceBucketName(sourceBucketName)
                   .withSourceKey(sourceObjectKey)
```

```
                    .withUploadId(initResult.getUploadId())
                    .withFirstByte(bytePosition)
                    .withLastByte(bytePosition + partSize -1 >= objectSize ?
objectSize - 1 : bytePosition + partSize - 1)
                    .withPartNumber(i);

                copyResponses.add(s3Client.copyPart(copyRequest));
                bytePosition += partSize;

            }
            CompleteMultipartUploadRequest completeRequest = new
             CompleteMultipartUploadRequest(
               targetBucketName,
               targetObjectKey,
               initResult.getUploadId(),
               GetETags(copyResponses));

            CompleteMultipartUploadResult completeUploadResponse =
                s3Client.completeMultipartUpload(completeRequest);
        } catch (Exception e) {
         System.out.println(e.getMessage());
        }
     }

    // Helper function that constructs ETags.
    static List<PartETag> GetETags(List<CopyPartResult> responses)
    {
        List<PartETag> etags = new ArrayList<PartETag>();
        for (CopyPartResult response : responses)
        {
            etags.add(new PartETag(response.getPartNumber(), re
sponse.getETag()));
        }
        return etags;
    }
}
```

AWS SDK for .NET マルチパートアップロード API を使用したオブジェクトのコピー

以下のタスクは、.NET SDK を使用してコピー元ロケーションから別のロケーション (例えば、あるバケットから別のバケット) に Amazon &S3 オブジェクトをコピーする手順を示しています。ここに示すコードを使用すると、5 GB よりも大きいオブジェクトをコピーできます。5 GB 未満のオブジェクトについては、1 回のコピーオペレーションを使用してください (「AWS SDK for .NET を使用したオブジェクトのコピー (p. 224)」を参照) 。

オブジェクトのコピー

1	AWS 認証情報を指定して、`AmazonS3Client` クラスのインスタンスを作成します。
2	`AmazonS3Client.InitiateMultipartUpload` メソッドを実行してマルチパートコピーを開始します。`InitiateMultipartUploadRequest` のインスタンスを作成します。バケット名とキー名を指定する必要があります。

3	AmazonS3Client.InitiateMultipartUpload メソッドから返されたレスポンスオブジェクトのアップロード ID を保存します。以降、マルチパートアップロードオペレーションのたびに、このアップロード ID を指定する必要があります。
4	すべてのパートをコピーします。パートのコピーごとに、CopyPartRequest クラスの新しいインスタンスを作成し、パート情報 (コピー元バケット、コピー先バケット、オブジェクトキー、アップロード ID、パートの先頭バイト、パートの最終バイト、パート番号) を指定します。
5	CopyPartRequest メソッドのレスポンスをリストに保存します。レスポンスには、マルチパートアップロードを完了するために必要な ETag 値とパート番号が含まれています。
6	各パートについてタスク 4 と 5 を繰り返します。
7	AmazonS3Client.CompleteMultipartUpload メソッドを実行してコピーを完了します。

以下の C# コード例は、前述のタスクの例です。

```
// Step 1. Create instance and provide credentials.
AmazonS3Config config = new AmazonS3Config();
AmazonS3 s3Client = new AmazonS3Client(AccessKeyID, SecretAccessKey, config);

// List to store upload part responses.
List<UploadPartResponse> uploadResponses =
    new List<UploadPartResponse>();
List<CopyPartResponse> copyResponses =
    new List<CopyPartResponse>();
InitiateMultipartUploadRequest initiateRequest =
    new InitiateMultipartUploadRequest()
        .WithBucketName(targetBucket)
        .WithKey(targetObjectKey);

// Step 2. Initialize.
InitiateMultipartUploadResponse initResponse =
s3Client.InitiateMultipartUpload(initiateRequest);

// Step 3. Save Upload Id.
String uploadId = initResponse.UploadId;

try
{
    // Get object size.
    GetObjectMetadataRequest metadataRequest = new GetObjectMetadataRequest();

    metadataRequest.BucketName = sourceBucket;
    metadataRequest.Key = sourceObjectKey;

    GetObjectMetadataResponse metadataResponse = s3Client.GetObject
Metadata(metadataRequest);
    long objectSize = metadataResponse.ContentLength; // in bytes

    // Step 4. Copy parts.
    long partSize = 5 * (long)Math.Pow(2, 20); // 5 MB
    long bytePosition = 0;
```

```
    for (int i = 1; bytePosition < objectSize; i++)
    {
        // Step 5. Save copy response.
        CopyPartRequest copyRequest = new CopyPartRequest()
            .WithDestinationBucket(targetBucket)
            .WithDestinationKey(targetObjectKey)
            .WithSourceBucket(sourceBucket)
            .WithSourceKey(sourceObjectKey)
            .WithUploadID(uploadId)
            .WithFirstByte(bytePosition)
           .WithLastByte(bytePosition + partSize -1 >= objectSize ? objectSize
 - 1 : bytePosition + partSize - 1)
            .WithPartNumber(i);

        copyResponses.Add(s3Client.CopyPart(copyRequest));
        bytePosition += partSize;

    }
     // Step 7. Complete copy operation.
     CompleteMultipartUploadRequest completeRequest =
        new CompleteMultipartUploadRequest()
            .WithBucketName(targetBucket)
            .WithKey(targetObjectKey)
            .WithUploadId(initResponse.UploadId)
            .WithPartETags(GetETags(copyResponses));

     CompleteMultipartUploadResponse completeUploadResponse =
        s3Client.CompleteMultipartUpload(completeRequest);

 }
catch (Exception e) {
    Console.WriteLine(e.Message);
}
```

Example

次の C# コード例では、ある Amazon S3 バケットから別のバケットにオブジェクトをコピーします。作業サンプルを作成およびテストする方法については、「.NET コード例のテスト (p. 478)」を参照してください。

```
using System;
using System.Collections.Generic;
using System.Collections.Specialized;
using System.Configuration;
using System.IO;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples.LowLevel_LargeObjectCopy
{
    class MPUcopyObject
    {

        static string sourceBucket = "*** Source bucket name ***";
        static string targetBucket = "*** Target bucket name ***";
        static string sourceObjectKey = "*** Source object key ***";
        static string targetObjectKey = "*** Target object key ***";

        static void Main(string[] args)
        {
            NameValueCollection appConfig = ConfigurationManager.AppSettings;
            string accessKeyID = appConfig["AWSAccessKey"];
            string secretAccessKey = appConfig["AWSSecretKey"];

            AmazonS3Config config = new AmazonS3Config();
           AmazonS3 s3Client = new AmazonS3Client(accessKeyID, secretAccessKey,
 config);

            // List to store upload part responses.
            List<UploadPartResponse> uploadResponses =
                new List<UploadPartResponse>();

            List<CopyPartResponse> copyResponses =
                      new List<CopyPartResponse>();
            InitiateMultipartUploadRequest initiateRequest =
                   new InitiateMultipartUploadRequest()
                    .WithBucketName(targetBucket)
                    .WithKey(targetObjectKey);

            InitiateMultipartUploadResponse initResponse =
                s3Client.InitiateMultipartUpload(initiateRequest);
            String uploadId = initResponse.UploadId;

            try
            {
                // Get object size.
                GetObjectMetadataRequest metadataRequest = new GetObject
MetadataRequest();
                metadataRequest.BucketName = sourceBucket;
                metadataRequest.Key = sourceObjectKey;

              GetObjectMetadataResponse metadataResponse = s3Client.GetObject
```

```
Metadata(metadataRequest);
                long objectSize = metadataResponse.ContentLength; // in bytes

                // Copy parts.
                long partSize = 5 * (long)Math.Pow(2, 20); // 5 MB

                long bytePosition = 0;
                for (int i = 1; bytePosition < objectSize; i++)
                {

                    CopyPartRequest copyRequest = new CopyPartRequest()
                       .WithDestinationBucket(targetBucket)
                       .WithDestinationKey(targetObjectKey)
                       .WithSourceBucket(sourceBucket)
                       .WithSourceKey(sourceObjectKey)
                       .WithUploadID(uploadId)
                       .WithFirstByte(bytePosition)
                       .WithLastByte(bytePosition + partSize - 1 >= objectSize
 ? objectSize - 1 : bytePosition + partSize - 1)
                       .WithPartNumber(i);

                    copyResponses.Add(s3Client.CopyPart(copyRequest));

                    bytePosition += partSize;

                }
                CompleteMultipartUploadRequest completeRequest =
                      new CompleteMultipartUploadRequest()
                         .WithBucketName(targetBucket)
                         .WithKey(targetObjectKey)
                         .WithUploadId(initResponse.UploadId)
                         .WithPartETags(GetETags(copyResponses));

                CompleteMultipartUploadResponse completeUploadResponse =
                    s3Client.CompleteMultipartUpload(completeRequest);

            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }

        // Helper function that constructs ETags.
        static List<PartETag> GetETags(List<CopyPartResponse> responses)
        {
            List<PartETag> etags = new List<PartETag>();
            foreach (CopyPartResponse response in responses)
            {
                etags.Add(new PartETag(response.PartNumber, response.ETag));
            }
            return etags;
        }
    }
}
```

REST マルチパートアップロード API を使用したオブジェクトのコピー

マルチパートアップロードを行うための REST API については、『*Amazon Simple Storage Service API リファレンス*』の以下のセクションを参照してください。既存のオブジェクトをコピーするには、パートのアップロード (コピー) API を使用し、リクエストに `x-amz-copy-source` リクエストヘッダーを追加してコピー元オブジェクトを指定します。

- Initiate Multipart Upload
- Upload Part
- パートのアップロード (コピー)
- Complete Multipart Upload
- マルチパートアップロードの中止
- パートのリスト
- マルチパートアップロードのリスト

これらの API を使用して独自の REST リクエストを作成するか、Amazon 提供の SDK のいずれかを使用することができます。SDK の詳細については、「マルチパートアップロードの API サポート (p. 152)」を参照してください。

オブジェクトキーのリスト作成

キーはプレフィックス別にリストできます。関連するキーの名前に共通するプレフィックスを選択し、階層を区切る特殊文字でそれらのキーをマーキングすることで、リストオペレーションによってキーを階層別に選択および参照できます。これは、ファイルシステムにおいてディレクトリ内にファイルを格納するしくみに似ています。

Amazon S3 では、バケット内のキーを列挙するためのリストオペレーションを実行できます。キーはバケット別またはプレフィックス別にリストされます。例えば、すべての英単語のキーを含む「dictionary」という名前のバケットがあるとします。そして、そのバケット内の文字「q」で始まるキーをすべてリストするための呼び出しを行うとします。リストの結果は常に、通常の辞書と同じ順序 (アルファベット順) で返されます。

SOAP および REST のリストオペレーションではいずれも、条件に一致するキーの名前と、各キーによって識別されるオブジェクトに関する情報を含む XML ドキュメントが返されます。

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

特殊な区切り記号で終わるプレフィックスを共有するキーのグループは、リストの目的で、その共通のプレフィックスによってロールアップできます。このような機能によりアプリケーションでは、キーを階層的に構成および参照できるようになります。ファイルシステムにおいてファイルをディレクトリに分けて整理するのに似ています。例えば、英語以外の単語も含まれるように dictionary バケットを拡張するには、各単語にその言語と区切り記号のプレフィックスを付けることでキーを形成できます (「French/logical」など)。この命名スキームと階層リスト機能を使用すれば、フランス語の単語のリストだけを取得することもできます。また、対応している言語の最上位レベルのリストを参照することで、それより下の階層間に介在するすべてのキーを反復処理する必要もなくなります。

リストにおける階層の役割の詳細については、「プレフィックスと区切り記号によるキーの階層的なリスト (p. 242)」を参照してください。

リスト実装の効率

リストのパフォーマンスは、バケット内のキーの総数によって実質的な影響を受けることはありません。また、プレフィックス、マーカー、最大キー数、または区切り記号の引数の有無によって影響を受けることもありません。

複数ページの結果内の反復処理

バケットに入れることのできるキーの数は実質的に無制限であるため、リストのクエリによっては結果が膨大な量になる可能性があります。大規模な結果セットを管理するため、Amazon S3 API ではページ分割をサポートして結果セットを複数のレスポンスに分割します。キーリストのレスポンスごとに最大 1,000 個のキーを含むページと、レスポンスが切り捨てられているかどうかを示すインジケータが返されます。すべてのキーを受信するまで、一連のキーリスト作成リクエストを送信します。このページ分割は、AWS SDK ラッパーライブラリでも行うことができます。サブセクションに示されている Java と .NET SDK の例では、ページ分割を使用してバケット内のキーをリストしています。

プレフィックスと区切り記号によるキーの階層的なリスト

プレフィックスと区切り記号のパラメータを使用すると、リストオペレーションによって返される結果の種類が制限されます。プレフィックスを指定すると、そのプレフィックスで始まるキーだけが返されます。区切り記号を使用すると、共通のプレフィックスを持つすべてのキーが 1 つの集約されたリスト結果にロールアップされます。

プレフィックスと区切り記号のパラメータの目的は、キーを階層的に構成および参照できるようにすることです。そのためには、まずバケットで使用する区切り記号を選択します。スラッシュ (/) など、キーの名前には使われないような記号を使用します。次に、階層に含まれるすべてのレベルを連結し、各レベルを区切り記号で区切ることで、キーの名前を構成します。

例えば、都市に関する情報を格納する場合は、大陸、国、州または県の順に情報を整理するのが自然でしょう。通常、都市名に句読点が含まれることはないので、区切り記号としてスラッシュ (/) を選択できます。次の例では、スラッシュ (/) 区切り記号を使用します。

- ヨーロッパ/フランス/アキテーヌ/ボルドー
- 北米/カナダ/ケベック州/モントリオール
- 北米/米国/ワシントン州/ベルビュー
- 北米/米国/ワシントン州/シアトル

などといった具合です。

世界中の全都市のデータをこの方法で格納した場合、階層のないフラットなキーネームスペースは管理しにくくなります。リストオペレーションで *Prefix* と *Delimiter* を使用することで、データをリストするために作成した階層を利用できます。例えば、米国のすべての州をリストするには、*Delimiter*='/' および *Prefix*='/North America/USA/' と設定します。データが存在するカナダのすべての州をリストするには、*Delimiter*='/' および *Prefix*='North America/Canada/' と設定します。

リストリクエストで区切り記号を使用すると、1 レベルの階層だけを参照でき、それより深いレベルで入れ子にされている (おそらく数百万もの) キーはスキップおよび集約されます。例えば、以下のキーを含むバケット (ExampleBucket) があるとします。

```
sample.jpg
```

```
photos/2006/January/sample.jpg

photos/2006/February/sample2.jpg

photos/2006/February/sample3.jpg

photos/2006/February/sample4.jpg
```

このサンプルバケットはルートレベルに sample.jpg オブジェクトだけを持ちます。バケット内のルートレベルオブジェクトだけをリストするには、バケットに対する GET リクエストを「/」の区切り文字と共に送信します。Amazon S3 からのレスポンスでは sample.jpg オブジェクトキーが返されます。このキーには「/」の区切り文字が含まれていないからです。その他すべてのキーには区切り文字が含まれています。Amazon S3 によってこれらのキーがグループ化され、特定のプレフィックス値を持つ 1 つの CommonPrefixes 要素が返されます。この場合のプレフィックス値は photos/ であり、これはキーの先頭から、指定した区切り文字の最初の出現箇所までのサブ文字列です。

```
<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Name>ExampleBucket</Name>
  <Prefix></Prefix>
  <Marker></Marker>
  <MaxKeys>1000</MaxKeys>
  <Delimiter>/</Delimiter>
  <IsTruncated>false</IsTruncated>
  <Contents>
    <Key>sample.jpg</Key>
    <LastModified>2011-07-24T19:39:30.000Z</LastModified>
    <ETag>&quot;d1a7fb5eab1c16cb4f7cf341cf188c3d&quot;</ETag>
    <Size>6</Size>
    <Owner>
      <ID>75cc57f09aa0c8caeab4f8c24e99d10f8e7faeebf76c078efc7c6caea54ba06a</ID>

      <DisplayName>displayname</DisplayName>
    </Owner>
    <StorageClass>STANDARD</StorageClass>
  </Contents>
  <CommonPrefixes>
    <Prefix>photos/</Prefix>
  </CommonPrefixes>
</ListBucketResult>
```

AWS SDK for Java を使用したキーのリスト作成

以下のタスクは、Java クラスを使用して、バケットに含まれるオブジェクトキーをリストする手順を示しています。バケット内のオブジェクトをリストするためのオプションは多数あります。大量のオブジェクトを含むバケットの場合、オブジェクトのリスト時に結果が切り捨てられる場合があることに注意してください。返されたオブジェクトのリストが切り捨てされているかどうかを確認し、切り捨てられている場合は AmazonS3.listNextBatchOfObjects オペレーションを使用して残りの結果を取得する必要があります。

オブジェクトキーのリスト

1	AWS 認証情報を指定して、AmazonS3Client クラスのインスタンスを作成します。

2	`AmazonS3Client.listObjects` メソッドのいずれかを実行します。バケット名やオプションのキープレフィックスなどのリクエスト情報を提供する必要があります。この情報は、`ListObjectsRequest` クラスのインスタンスを作成することによって指定します。オブジェクトのリスト作成の場合、Amazon S3 からのレスポンスで返されるキーは最大 1,000 個です。バケット内に 1,000 個を超えるキーが含まれる場合、レスポンスは切り捨てられます。レスポンスが切り捨てられているかどうかを常に確認する必要があります。

以下の Java コード例は、前述のタスクを実装したものです。

```
AWSCredentials myCredentials = new BasicAWSCredentials(
                                  myAccessKeyID, mySecretKey);
AmazonS3 s3client = new AmazonS3Client(myCredentials);

ListObjectsRequest listObjectsRequest = new ListObjectsRequest()
.withBucketName(bucketName)
.withPrefix("m");
ObjectListing objectListing;

do {
 objectListing = s3client.listObjects(listObjectsRequest);
 for (S3ObjectSummary objectSummary :
  objectListing.getObjectSummaries()) {
  System.out.println( " - " + objectSummary.getKey() + "  " +
                "(size = " + objectSummary.getSize() +
    ")");
 }
 listObjectsRequest.setMarker(objectListing.getNextMarker());
} while (objectListing.isTruncated());
```

Example

次の Java コード例では、Amazon S3 バケット内のオブジェクトキーをリストします。実際に動作するコードを作成、テストする方法については、「Java コード例のテスト (p. 477)」を参照してください。

```
import java.io.IOException;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.ListObjectsRequest;
import com.amazonaws.services.s3.model.ObjectListing;
import com.amazonaws.services.s3.model.S3ObjectSummary;

public class S3Sample {
 private static String bucketName = "*** Provide bucket name ***";

 public static void main(String[] args) throws IOException {
        AmazonS3 s3client = new AmazonS3Client(new PropertiesCredentials(
                S3Sample.class.getResourceAsStream(
                  "AwsCredentials.properties")));
        try {
            System.out.println("Listing objects");

            ListObjectsRequest listObjectsRequest = new ListObjectsRequest()
                .withBucketName(bucketName)
                .withPrefix("m");
            ObjectListing objectListing;
            do {
                objectListing = s3client.listObjects(listObjectsRequest);
                for (S3ObjectSummary objectSummary :
                 objectListing.getObjectSummaries()) {
                   System.out.println(" - " + objectSummary.getKey() + "  " +

                           "(size = " + objectSummary.getSize() +
                           ")");
                }
                listObjectsRequest.setMarker(objectListing.getNextMarker());
            } while (objectListing.isTruncated());
         } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException, " +
              "which means your request made it " +
                    "to Amazon S3, but was rejected with an error response " +

                    "for some reason.");
            System.out.println("Error Message:    " + ase.getMessage());
            System.out.println("HTTP Status Code: " + ase.getStatusCode());
            System.out.println("AWS Error Code:   " + ase.getErrorCode());
            System.out.println("Error Type:       " + ase.getErrorType());
            System.out.println("Request ID:       " + ase.getRequestId());
        } catch (AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException, " +
              "which means the client encountered " +
                    "an internal error while trying to communicate" +
                    " with S3, " +
                    "such as not being able to access the network.");
```

```
            System.out.println("Error Message: " + ace.getMessage());
        }
    }
}
```

AWS SDK for .NET を使用したキーのリスト作成

以下のタスクは、.NET クラスを使用して、バケットに含まれるオブジェクトキーをリストする手順を示しています。

オブジェクトキーのリスト

1	AWS 認証情報を指定して、AmazonS3 クラスのインスタンスを作成します。
2	いずれかの AmazonS3.ListObjects を実行します。キーをリストするには、バケット名を提供する必要があります。特定のプレフィックスで始まるキーを取得したり、結果セットを一定のキー数に制限したりするためのオプション情報を指定することもできます。デフォルトでは、結果セットで返されるキー数は最大で 1,000 個です。この情報は、ListObjectsRequest クラスのインスタンスを作成することによって指定します。
3	ListObjectResponse を処理するため、ListObjectResponse.S3Objects コレクションに対して反復処理を行います。結果が切り捨てられているかどうかを確認する必要があります。切り捨てられていれば、続くキーセットを取得するリクエストを送信するため、ListObjectRequest.Marker 値を、直前のレスポンスで受信した NextMarker 値に設定します。

以下の C# コード例は、前述のタスクの例です。

```
static AmazonS3 client;
client = Amazon.AWSClientFactory.CreateAmazonS3Client(
                    accessKeyID, secretAccessKeyID);

ListObjectsRequest request = new ListObjectsRequest();
request = new ListObjectsRequest();
request.BucketName = bucketName;
request.WithPrefix("m");
request.MaxKeys = 2;
do
{
   ListObjectsResponse response = client.ListObjects(request);

   // Process response.
   // ...

   // If response is truncated, set the marker to get the next
   // set of keys.
   if (response.IsTruncated)
   {
        request.Marker = response.NextMarker;
   }
        else
   {
        request = null;
```

```
    }
} while (request != null);
```

Example

次の C# コード例では、指定したバケットのキーをリストします。この例は、AmazonS3.ListObjects メソッドの使い方を示しています。また、キーをリストするためのオプションの指定方法も示しています。例えば、特定のプレフィックスで始まるキーのリスト、特定のマーカーの後に始まるキーのリスト、一定数のキーのみのリストが可能です。実際に動作するコードを作成、テストする方法については、「.NET コード例のテスト (p. 478)」を参照してください。

```
using System;
using System.Configuration;
using System.Collections.Specialized;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples.listingkeys
{
    class S3Sample
    {
        static string bucketName = "*** Provide bucket name ***";
        static AmazonS3 client;

        public static void Main(string[] args)
        {
            if (checkRequiredFields())
            {
                NameValueCollection appConfig =
                    ConfigurationManager.AppSettings;

                string accessKeyID = appConfig["AWSAccessKey"];
                string secretAccessKeyID = appConfig["AWSSecretKey"];

                using (client = Amazon.AWSClientFactory.CreateAmazonS3Client(
                    accessKeyID, secretAccessKeyID))
                {
                   Console.WriteLine("Listing objects stored in a bucket");
                    ListingObjects();
                }
            }

            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }

        static void ListingObjects()
        {
            try
            {
                ListObjectsRequest request = new ListObjectsRequest();
                request = new ListObjectsRequest();
                request.BucketName = bucketName;
                request.WithPrefix("m");
                request.MaxKeys = 2;

               do
               {
                   ListObjectsResponse response = client.ListObjects(request);
```

```
                // Process response.
                foreach (S3Object entry in response.S3Objects)
                {
                    Console.WriteLine("key = {0} size = {1}",
                        entry.Key, entry.Size);
                }

                // If response is truncated, set the marker to get the next

                // set of keys.
                if (response.IsTruncated)
                {
                    request.Marker = response.NextMarker;
                }
                else
                {
                    request = null;
                }
            } while (request != null);
        }
        catch (AmazonS3Exception amazonS3Exception)
        {
            if (amazonS3Exception.ErrorCode != null &&
                (amazonS3Exception.ErrorCode.Equals("InvalidAccessKeyId")
                ||
                amazonS3Exception.ErrorCode.Equals("InvalidSecurity")))
            {
                Console.WriteLine("Check the provided AWS Credentials.");
                Console.WriteLine(
                "To sign up for service, go to http://aws.amazon.com/s3");

            }
            else
            {
                Console.WriteLine(
                 "Error occurred. Message:'{0}' when listing objects",
                 amazonS3Exception.Message);
            }
        }
    }

    static bool checkRequiredFields()
    {
        NameValueCollection appConfig = ConfigurationManager.AppSettings;

        if (string.IsNullOrEmpty(appConfig["AWSAccessKey"]))
        {
            Console.WriteLine(
                "AWSAccessKey was not set in the App.config file.");
            return false;
        }
        if (string.IsNullOrEmpty(appConfig["AWSSecretKey"]))
        {
            Console.WriteLine(
                "AWSSecretKey was not set in the App.config file.");
            return false;
        }
        if (string.IsNullOrEmpty(bucketName))
```

```
            {
                Console.WriteLine("The variable bucketName is not set.");
                return false;
            }

            return true;
        }
    }
}
```

AWS SDK for PHP を使用したキーのリスト作成

このトピックでは、AWS SDK for PHP のクラスを使用して、Amazon S3 バケットに含まれるオブジェクトキーをリストする手順を示します。

Note

このトピックでは、既に AWS SDK for PHP の使用と PHP サンプルの実行 (p. 479) の説明が実行されていて、AWS SDK for PHP が正しくインストールされていることを前提としています。

AWS SDK for PHP を使用してバケットに含まれるオブジェクトキーをリストするには、まずバケットに含まれるオブジェクトをリストし、次にリストされた各オブジェクトからキーを抽出します。バケットのオブジェクトキーをリストする際、低レベルの Aws\S3\S3Client::listObjects() メソッドを使用するか、または高レベルの Aws\S3\Iterator\ListObjects イテレータを使用するかを選択できます。

低レベルの `listObjects()` メソッドは基礎的な Amazon S3 REST API に対応しています。1 つの `listObjects()` リクエストは、最大 1,000 オブジェクトからなるページを返します。バケットに 1,000 を超えるオブジェクトがある場合はレスポンスが切り捨てられるため、次の 1,000 個のオブジェクトのセットを取得するにはもう 1 つの `listObjects()` リクエストを送信する必要があります。

バケットに含まれるオブジェクトをリストするタスクを多少簡単にするには、高レベルの `ListObjects` イテレータを使用します。`ListObjects` イテレータを使用してオブジェクトのリストを作成するには、Guzzle\Service\Client クラスから継承された、Amazon S3 クライアントの getIterator() メソッドを、1 番目の引数として `ListObjects` コマンドを、2 番目の引数として、指定したバケットから返されるオブジェクトを格納する array を指定して、実行します。`getIterator()` メソッドを `ListObjects` イテレータとして使用すると、指定したバケットに含まれるすべてのオブジェクトが返されます。1,000 個までのオブジェクトの制限がないため、レスポンスが切り捨てられるかどうかを心配する必要はありません。

以下のタスクは、PHP Amazon S3 クライアントのメソッドを使用して、リストするオブジェクトキーが含まれるバケットにあるオブジェクトを、リストする手順を示しています。

オブジェクトキーのリスト

1	自分の AWS 認証情報または IAM ユーザー認証情報と共に Aws\S3\S3Client クラスの factory メソッドを使用して、Amazon S3 クライアントのインスタンスを作成します。
2	高レベルの Amazon S3 クライアントの `getIterator()` メソッドを、1 番目の引数として `ListObjects` コマンドを、2 番目の引数として、指定したバケットから返されるオブジェクトを格納する array を指定して、実行します。 または、低レベルの Amazon S3 クライアントの `listObjects()` メソッドを、引数として、指定したバケットから返されるオブジェクトを格納する array を指定して、実行します。
3	返されたオブジェクトのリストの各オブジェクトから、オブジェクトキーを抽出します。

以下の PHP コード例は、リストするオブジェクトキーが含まれるバケットにあるオブジェクトを、リストする方法を示しています。

```
use Aws\S3\S3Client;

// Instantiate the client.
$s3 = S3Client::factory(array(
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));

$bucket = '*** Bucket Name ***';

// Use the high-level iterators (returns ALL of your objects).
$objects = $s3->getIterator('ListObjects', array('Bucket' => $bucket));

echo "Keys retrieved!\n";
foreach ($objects as $object) {
    echo $object['Key'] . "\n";
}

// Use the plain API (returns ONLY up to 1000 of your objects).
$result = $s3->listObjects(array('Bucket' => $bucket));

echo "Keys retrieved!\n";
foreach ($result['Contents'] as $object) {
    echo $object['Key'] . "\n"
}
```

Example （ オブジェクトキーのリスト作成 ）

以下の PHP コード例は、指定されたバケット内のキーをリストする方法を示しています。これは、まず高レベルの `getIterator()` メソッドを使用してバケット内のオブジェクトをリストする方法を示し、次にリスト内の各オブジェクトからキーを抽出する方法を示します。また、バケット内のオブジェクトをリストするために低レベルの `listObjects()` メソッドを実行し、返されたリスト内の各オブジェクトからキーを抽出する方法も示します。PHP 例の実行については、このガイド内の「PHP サンプルの実行 (p. 480)」を参照してください。

```
<?php

// Include the AWS SDK using the Composer autoloader.
require 'vendor/autoload.php';

use Aws\S3\S3Client;
use Aws\S3\Exception\S3Exception;

$bucket = '*** Your Bucket Name ***';

// Instantiate the client.
$s3 = S3Client::factory(array(
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));

// Use the high-level iterators (returns ALL of your objects).
try {
    $objects = $s3->getIterator('ListObjects', array(
        'Bucket' => $bucket
    ));

    echo "Keys retrieved!\n";
    foreach ($objects as $object) {
        echo $object['Key'] . "\n";
    }
} catch (S3Exception $e) {
    echo $e->getMessage() . "\n";
}

// Use the plain API (returns ONLY up to 1000 of your objects).
try {
    $result = $s3->listObjects(array('Bucket' => $bucket));

    echo "Keys retrieved!\n";
    foreach ($result['Contents'] as $object) {
        echo $object['Key'] . "\n";
    }
} catch (S3Exception $e) {
    echo $e->getMessage() . "\n";
}
```

関連リソース

- 「AWS SDK for PHP for Amazon S3 の Aws\S3\Iterator\ListObjects」
- 「AWS SDK for PHP for Amazon S3 の Guzzle\Service\Client クラス」
- 「AWS SDK for PHP for Amazon S3 の Guzzle\Service\Client::getIterator() メソッド」

- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client クラス」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::factory() メソッド」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::listObjects() メソッド」
- 「AWS SDK for PHP – Amazon S3」
- 「AWS SDK for PHP」のドキュメント

REST API を使用したキーのリスト

AWS SDK を使用して、1 つのオブジェクトを取得できます。ただし、アプリケーションで必要な場合は、REST リクエストを直接送信できます。GET リクエストを送信して、バケット内の一部またはすべてのオブジェクトを取得できます。または選択条件を使用して、バケット内のオブジェクトのサブセットを返すことができます。詳細については、「GET Bucket (List Objects) 」を参照してください。

関連リソース

- AWS dynamo と Explorer の使用 (p. 475)

オブジェクトの削除

Topics

- バージョニング対応バケットからのオブジェクトの削除 (p. 253)
- MFA 対応バケットからのオブジェクトの削除 (p. 254)
- 関連リソース (p. 254)
- 1 件のリクエストで 1 つのオブジェクトを削除 (p. 254)
- 1 件のリクエストで複数オブジェクトを削除 (p. 264)

1 つ以上のオブジェクトを Amazon S3 から直接削除できます。オブジェクトを削除するときは、以下のオプションを利用できます。

- 単一のオブジェクトの削除 – Amazon S3 には DELETE API があり、これを使用すると、単一の HTTP リクエストで 1 つのオブジェクトを削除できます。
- 複数のオブジェクトの削除 – Amazon S3 には Multi-Object Delete API もあり、これを使用すると、単一の HTTP リクエストで最大 1,000 個のオブジェクトを削除できます。

バージョニング非対応のバケットからオブジェクトを削除する場合は、オブジェクトキー名のみを指定します。これに対し、バージョニング対応のバケットからオブジェクトを削除する場合は、オプションでオブジェクトのバージョン ID を指定すると、特定のバージョンのオブジェクトを削除できます。

バージョニング対応バケットからのオブジェクトの削除

バージョニング対応のバケットの場合、複数のバージョンのオブジェクトがバケット内に存在する可能性があります。バージョニング対応のバケットを操作する場合は、delete API で以下のオプションが可能です。

- バージョンを指定しない削除リクエストの指定 – オブジェクトのキーのみを指定し、バージョン ID は指定しません。この場合、Amazon S3 は削除マーカーを作成し、レスポンスでバージョン ID を返

します。これにより、オブジェクトはバケットから消去されます。オブジェクトのバージョニングと削除マーカーの概念については、「オブジェクトのバージョニング (p. 111)」を参照してください。

- バージョンを指定した削除リクエストの指定–キーとバージョン ID の両方を指定します。この場合、以下の 2 つの結果になる可能性があります。
  - バージョン ID が特定のオブジェクトバージョンに対応する場合、Amazon S3 はその特定のバージョンのオブジェクトを削除します。
  - バージョン ID がそのオブジェクトの削除マーカーに対応する場合、Amazon S3 は削除マーカーを削除します。これにより、オブジェクトはバケットに再表示されます。

MFA 対応バケットからのオブジェクトの削除

Multi Factor Authentication (MFA) 対応のバケットからオブジェクトを削除する場合は、以下の点に注意してください。

- 無効な MFA トークンを指定した場合、リクエストは常に失敗します。
- MFA 対応バケットを使用していて、バージョンを指定した削除リクエストを行う (オブジェクトキーとバージョン ID を指定する) 場合、有効な MFA トークンを指定しないとリクエストは失敗します。さらに、MFA 対応バケットに対して Multi-Object Delete API を使用するときに、いずれかの削除がバージョンを指定した削除リクエストである場合 (つまり、オブジェクトキーとバージョン ID を指定した場合)、MFA トークンを指定しないとリクエスト全体が失敗します。

一方、以下の場合は、リクエストは成功します。

- MFA 対応バケットを使用していて、バージョンを指定しない削除リクエストを行う (バージョンを指定せずにオブジェクトを削除する) 場合、 MFA トークンを指定しなくても削除は成功します。
- Multi-Object Delete リクエストで、MFA 対応バケットからバージョンを指定せずにオブジェクトを削除するように指定した場合、 MFA トークンを指定しなくても削除は成功します。

MFA の削除については、「MFA Delete (p. 389)」を参照してください。

関連リソース

- AWS dynamo と Explorer の使用 (p. 475)

1 件のリクエストで 1 つのオブジェクトを削除

Topics

- AWS SDK for Java を使用したオブジェクトのコピー (p. 255)
- AWS SDK for .NET を使用した 1 つのオブジェクトの削除 (p. 258)
- AWS SDK for PHP を使用したオブジェクトの削除 (p. 262)
- REST API を使用した 1 つのオブジェクトの削除 (p. 264)

Amazon S3 の DELETE API (「DELETE Object」を参照) を使用すると、1 件のリクエストで 1 つのオブジェクトを削除できます。オブジェクトの削除については、「オブジェクトの削除 (p. 253)」を参照してください。

REST API を直接使用するか、AWS SDK が提供するラッパーライブラリを使用できます。このライブラリにより、アプリケーション開発を簡素化することができます。

AWS SDK for Java を使用したオブジェクトのコピー

以下のタスクは、AWS SDK for Java のクラスを使用して、1 つのオブジェクトを削除する手順を示しています。

オブジェクトの削除 (バージョニング非対応のバケット)

1	AWS 認証情報を指定して、`AmazonS3Client` クラスのインスタンスを作成します。
2	`AmazonS3Client.deleteObject` メソッドのいずれかを実行します。 バケット名とオブジェクト名をパラメータとして指定するか、同じ情報を `DeleteObjectRequest` オブジェクトに指定し、そのオブジェクトをパラメータとして渡すことができます。 バケットに対してバージョニングを有効にしていない場合は、オペレーションでオブジェクトが削除されます。バージョニングを有効にしている場合は、オペレーションで削除マーカーが追加されます。詳細については、「1 件のリクエストで 1 つのオブジェクトを削除 (p. 254)」を参照してください。

以下の Java サンプルは、前述のステップを実装したものです。この例では、`DeleteObjectRequest` クラスを使用して、バケット名とオブジェクトキーを指定します。

```
AWSCredentials myCredentials = new BasicAWSCredentials(
                                          myAccessKeyID, mySecretKey);
AmazonS3 s3client = new AmazonS3Client(myCredentials);
s3client.deleteObject(new DeleteObjectRequest(bucketName, keyName));
```

特定のバージョンのオブジェクトの削除 (バージョニング対応のバケット)

1	AWS 認証情報を指定して、`AmazonS3Client` クラスのインスタンスを作成します。
2	`AmazonS3Client.deleteVersion` メソッドのいずれかを実行します。 バケット名とオブジェクトキーをパラメータとして直接指定するか、`DeleteVersionRequest` を使用して同じ情報を渡すことができます。

以下の Java サンプルは、前述のステップの例です。この例では、`DeleteVersionRequest` クラスを使用して、バケット名、オブジェクトキー、およびバージョン ID を指定します。

```
AWSCredentials myCredentials = new BasicAWSCredentials(
                                          myAccessKeyID, mySecretKey);
AmazonS3 s3client = new AmazonS3Client(myCredentials);
s3client.deleteObject(new DeleteVersionRequest(bucketName, keyName, versionId));
```

Example 1: オブジェクトの削除 (バージョニング非対応のバケット)

次の Java の例では、バケットからオブジェクトを削除します。バケットに対してバージョニングを有効にしていない場合、Amazon S3 はオブジェクトを削除します。バージョニングを有効にしている場合、Amazon S3 は削除マーカーを追加し、オブジェクトは削除されません。作業サンプルを作成およびテストする方法については、「Java コード例のテスト (p. 477)」を参照してください。

```
import java.io.IOException;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.DeleteObjectRequest;

public class S3Sample  {

    private static String bucketName = "*** Provide a Bucket Name ***";
    private static String keyName = "*** Provide a Key Name ****";

    public static void main(String[] args) throws IOException {
        AmazonS3 s3Client = new AmazonS3Client(new PropertiesCredentials(
          S3Sample.class.getResourceAsStream("AwsCredentials.properties")));
        try {
          s3Client.deleteObject(new DeleteObjectRequest(bucketName, keyName));

        } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException.");
            System.out.println("Error Message:    " + ase.getMessage());
            System.out.println("HTTP Status Code: " + ase.getStatusCode());
            System.out.println("AWS Error Code:   " + ase.getErrorCode());
            System.out.println("Error Type:       " + ase.getErrorType());
            System.out.println("Request ID:       " + ase.getRequestId());
        } catch (AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException.");
            System.out.println("Error Message: " + ace.getMessage());
        }
    }
}
```

Example 2: （バージョニング対応のバケットからの）オブジェクトの削除

次の Java の例では、バージョニングされたバケットから、特定のバージョンのオブジェクトを削除します。`deleteObject` リクエストは、特定のバージョンのオブジェクトをバケットから削除します。

サンプルをテストするには、バケット名を指定する必要があります。このコード例は、以下のタスクを実行します。

1. バケットに対してバージョニングを有効にします。
2. サンプルオブジェクトをバケットに追加します。レスポンスとして、Amazon S3 は新しく追加されたオブジェクトのバージョン ID を返します。
3. `deleteVersion` メソッドを使用して、サンプルオブジェクトを削除します。`DeleteVersionRequest` クラスは、オブジェクトキー名とバージョン ID の両方を指定します。

作業サンプルを作成およびテストする方法については、「Java コード例のテスト (p. 477)」を参照してください。

```
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.util.Random;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.BucketVersioningConfiguration;
import com.amazonaws.services.s3.model.CannedAccessControlList;
import com.amazonaws.services.s3.model.DeleteVersionRequest;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.PutObjectRequest;
import com.amazonaws.services.s3.model.PutObjectResult;
import com.amazonaws.services.s3.model.SetBucketVersioningConfigurationRequest;

public class S3Sample  {

    static String bucketName = "*** Provide a Bucket Name ***";
    static String keyName = "*** Provide a Key Name ****";
    static AmazonS3Client s3Client;

    public static void main(String[] args) throws IOException {
        s3Client = new AmazonS3Client(new PropertiesCredentials(
          S3Sample.class.getResourceAsStream("AwsCredentials.properties")));
        try {
            // Make the bucket version-enabled.
            enableVersioningOnBucket(s3Client, bucketName);

            // Add a sample object.
            String versionId = putAnObject(keyName);

            s3Client.deleteVersion(
                    new DeleteVersionRequest(
                            bucketName,
                            keyName,
                            versionId));

        } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException.");
```

```
            System.out.println("Error Message:    " + ase.getMessage());
            System.out.println("HTTP Status Code: " + ase.getStatusCode());
            System.out.println("AWS Error Code:   " + ase.getErrorCode());
            System.out.println("Error Type:       " + ase.getErrorType());
            System.out.println("Request ID:       " + ase.getRequestId());
        } catch (AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException.");
            System.out.println("Error Message: " + ace.getMessage());
        }

    }

    static void enableVersioningOnBucket(AmazonS3Client s3Client,
            String bucketName) {
        BucketVersioningConfiguration config = new BucketVersioningConfigura
tion()
                .withStatus(BucketVersioningConfiguration.ENABLED);
        SetBucketVersioningConfigurationRequest setBucketVersioningConfigura
tionRequest = new SetBucketVersioningConfigurationRequest(
                bucketName, config);
       s3Client.setBucketVersioningConfiguration(setBucketVersioningConfigura
tionRequest);
    }

    static String putAnObject(String keyName) {
        String content = "This is the content body!";
        String key = "ObjectToDelete-" + new Random().nextInt();
        ObjectMetadata metadata = new ObjectMetadata();
        metadata.setHeader("Subject", "Content-As-Object");
        metadata.setHeader("Content-Length", content.length());
        PutObjectRequest request = new PutObjectRequest(bucketName, key,
                new ByteArrayInputStream(content.getBytes()), metadata)
                .withCannedAcl(CannedAccessControlList.AuthenticatedRead);
        PutObjectResult response = s3Client.putObject(request);
        return response.getVersionId();
    }
}
```

AWS SDK for .NET を使用した 1 つのオブジェクトの削除

バケットから 1 つのオブジェクトを削除できます。そのバケットに対してバージョニングを有効にしている場合は、特定のバージョンのオブジェクトを削除することもできます。

以下のタスクは、.NET のクラスを使用して、1 つのオブジェクトを削除する手順を示しています。

オブジェクトの削除 (バージョニング非対応のバケット)

1	AWS 認証情報を指定して、`AmazonS3Client` クラスのインスタンスを作成します。
2	`AmazonS3.DeleteObject` メソッドを実行します。このとき、`DeleteObjectRequest` のインスタンスにバケット名とオブジェクトキーを指定します。 バケットに対してバージョニングを有効にしていない場合は、オペレーションでオブジェクトが削除されます。バージョニングを有効にしている場合は、オペレーションで削除マーカーが追加されます。詳細については、「1 件のリクエストで 1 つのオブジェクトを削除 (p. 254)」を参照してください。

以下の C# コード例は、前述のステップの例です。

```
static AmazonS3Client client;
client = new AmazonS3Client;

DeleteObjectRequest deleteObjectRequest =
    new DeleteObjectRequest()
    .WithBucketName(bucketName)
    .WithKey(keyName);

using (client = Amazon.AWSClientFactory.CreateAmazonS3Client(
     accessKeyID, secretAccessKeyID))
{
     client.DeleteObject(deleteObjectRequest);
}
```

特定のバージョンのオブジェクトの削除 (バージョニング対応のバケット)

1	AWS 認証情報を指定して、AmazonS3Client クラスのインスタンスを作成します。
2	AmazonS3.DeleteObject メソッドを実行します。このとき、DeleteObjectRequest のインスタンスにバケット名、オブジェクトキー名、オブジェクトのバージョン ID を指定します。 DeleteObject メソッドは特定のバージョンのオブジェクトを削除します。

以下の C# コード例は、前述のステップの例です。

```
static AmazonS3Client client;
client = new AmazonS3Client();

DeleteObjectRequest deleteObjectRequest =
    new DeleteObjectRequest()
    .WithBucketName(bucketName)
    .WithKey(keyName)
    .WithVersionId(versionId);

using (client = Amazon.AWSClientFactory.CreateAmazonS3Client(
     accessKeyID, secretAccessKeyID))
{
     client.DeleteObject(deleteObjectRequest);
}
```

Example 1: （バージョニング非対応のバケットからの）オブジェクトの削除

次の C# コード例では、バケットからオブジェクトを削除します。削除リクエストにバージョン ID は指定していません。バケットに対してバージョニングを有効にしていない場合、Amazon S3 はオブジェクトを削除します。バージョニングを有効にしている場合、Amazon S3 は削除マーカーを追加し、オブジェクトは削除されません。作業サンプルを作成およびテストする方法については、「.NET コード例のテスト (p. 478)」を参照してください。

```
using System;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples.deleteobject
{
  class DeleteObject
  {
    static string bucketName = "*** Provide a bucket name ***";
    static string keyName = "*** Provide a key name ****";
    static AmazonS3Client client;

    public static void Main(string[] args)
    {
      DeleteObjectRequest deleteObjectRequest =
          new DeleteObjectRequest()
          .WithBucketName(bucketName)
          .WithKey(keyName);

      using (client = new AmazonS3Client())
      {
        try
        {
          client.DeleteObject(deleteObjectRequest);
          Console.WriteLine("Deleting an object");
        }
        catch (AmazonS3Exception s3Exception)
        {
          Console.WriteLine(s3Exception.Message,
                            s3Exception.InnerException);
        }
      }
      Console.WriteLine("Press any key to continue...");
      Console.ReadKey();
    }
  }
}
```

Example 2: （バージョニング対応のバケットからの）オブジェクトの削除

次の C# コード例では、バージョニングされたバケットからオブジェクトを削除します。`DeleteObjectRequest` インスタンスは、オブジェクトキー名とバージョン ID を指定します。`DeleteObject` メソッドは、特定のバージョンのオブジェクトをバケットから削除します。

サンプルをテストするには、バケット名を指定する必要があります。このコード例は、以下のタスクを実行します。

1. バケットに対してバージョニングを有効にします。
2. サンプルオブジェクトをバケットに追加します。レスポンスとして、Amazon S3 は新しく追加されたオブジェクトのバージョン ID を返します。
3. `DeleteObject` メソッドを使用して、サンプルオブジェクトを削除します。`DeleteObjectRequest` クラスは、オブジェクトキー名とバージョン ID の両方を指定します。

作業サンプルを作成およびテストする方法については、「.NET コード例のテスト (p. 478)」を参照してください。

```
using System;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples
{
  class DeleteObject
  {
    static string bucketName = "*** Provide a Bucket Name ***";
    static string keyName = "*** Provide a Key Name ***";
    static AmazonS3Client client;

    public static void Main(string[] args)
    {
      using (client = new AmazonS3Client())
      {
        try
        {
          // Make the bucket version-enabled.
          EnableVersioningOnBucket(bucketName);

          // Add a sample object.
          string versionID = PutAnObject(keyName);

          // Delete the object by specifying an object key and a version ID.
          DeleteObjectRequest request = new DeleteObjectRequest
          {
            BucketName = bucketName,
            Key = keyName,
            VersionId = versionID
          };
          Console.WriteLine("Deleting an object");
          client.DeleteObject(request);

        }
        catch (AmazonS3Exception s3Exception)
        {
          Console.WriteLine(s3Exception.Message,
                            s3Exception.InnerException);
```

```
        }
      }
      Console.WriteLine("Press any key to continue...");
      Console.ReadKey();
    }

    static void EnableVersioningOnBucket(string bucketName)
    {
     SetBucketVersioningRequest setBucketVersioningRequest = new SetBucketVer
sioningRequest
      {
        BucketName = bucketName,
        VersioningConfig = new S3BucketVersioningConfig { Status = "Enabled" }

      };
      client.SetBucketVersioning(setBucketVersioningRequest);
    }

    static string PutAnObject(string objectKey)
    {

      PutObjectRequest request = new PutObjectRequest
      {
        BucketName = bucketName,
        Key = objectKey,
        ContentBody = "This is the content body!"
      };

      PutObjectResponse response = client.PutObject(request);
      return response.VersionId;

    }
  }
}
```

AWS SDK for PHP を使用したオブジェクトの削除

このトピックでは、AWS SDK for PHP のクラスを使用して、バージョニング非対応のバケットから 1 つのオブジェクトを削除する手順を示します。バージョニング対応のバケットから 1 つのオブジェクトを削除する方法の詳細については、「REST API を使用した 1 つのオブジェクトの削除 (p. 264)」を参照してください。

Note

このトピックでは、既に AWS SDK for PHP の使用と PHP サンプルの実行 (p. 479) の説明が実行されていて、AWS SDK for PHP が正しくインストールされていることを前提としています。

(バージョニング非対応のバケットからの) 1 つのオブジェクトの削除

1	自分の AWS 認証情報または IAM ユーザー認証情報と共に Aws\S3\S3Client クラスの factory() メソッドを使用して、Amazon S3 クライアントのインスタンスを作成します。

2	Aws\S3\S3Client::deleteObject() メソッドを実行します。array パラメータの必須キー、Bucket と Key に、バケット名とキー名を指定する必要があります。 バケットに対してバージョニングを有効にしていない場合は、このオペレーションでオブジェクトが削除されます。バージョニングを有効にしている場合は、このオペレーションで削除マーカーが追加されます。詳細については、「オブジェクトの削除 (p. 253)」を参照してください。

以下の PHP コード例は、deleteObject() メソッドを使用して、Amazon S3 のバケットから 1 つのオブジェクトを削除する方法を実際に示しています。

```
use Aws\S3\S3Client;

$s3 = S3Client::factory(array(
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

$result = $s3->deleteObject(array(
    'Bucket' => $bucket,
    'Key'    => $keyname
));
```

Example （バージョニング非対応のバケットからのオブジェクトの削除）

次の PHP コード例では、バケットからオブジェクトを削除します。削除リクエストにバージョン ID は指定していません。バケットに対してバージョニングを有効にしていない場合、Amazon S3 はオブジェクトを削除します。バージョニングを有効にしている場合、Amazon S3 は削除マーカーを追加し、オブジェクトは削除されません。PHP 例の実行については、このガイド内の「PHP サンプルの実行 (p. 480)」を参照してください。バージョニング対応のバケットから 1 つのオブジェクトを削除する方法の詳細については、「REST API を使用した 1 つのオブジェクトの削除 (p. 264)」を参照してください。

```
<?php

// Include the AWS SDK using the Composer autoloader.
require 'vendor/autoload.php';

use Aws\S3\S3Client;

$s3 = S3Client::factory(array(
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

$result = $s3->deleteObject(array(
    'Bucket' => $bucket,
    'Key'    => $keyname
));
```

関連リソース

- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client クラス」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::factory() メソッド」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::deleteObject() メソッド」
- 「AWS SDK for PHP – Amazon S3」
- 「AWS SDK for PHP」のドキュメント

REST API を使用した 1 つのオブジェクトの削除

AWS SDK を使用して、1 つのオブジェクトを削除できます。ただし、アプリケーションで必要な場合は、REST リクエストを直接送信できます。詳細については、『Amazon Simple Storage Service API リファレンス』の「*DELETE Object*」を参照してください。

1 件のリクエストで複数オブジェクトを削除

Topics

Amazon S3 の Multi-Object Delete API (「Delete Multiple Objects」を参照) を使用すると、単一のリクエストで複数のオブジェクトを削除できます。この API は、「冗長」と「簡略」という2つのレスポンスモードをサポートしています。デフォルトでは、オペレーションには冗長モードが使用されます。このモードでは、リクエストで行われたキー削除の各操作の結果がレスポンスに含まれます。簡略モードでは、レスポンスに含まれるのは削除オペレーションでエラーが発生したキーのみです。

簡略モード使用時にすべてのキーが正常に削除された場合、Amazon S3 は空のレスポンスを返します。

オブジェクトの削除については、「オブジェクトの削除 (p. 253)」を参照してください。

REST API を直接使用するか、AWS SDK を使用することができます。

AWS SDK for Java を使用した複数オブジェクトの削除

以下のタスクは、AWS SDK for Java のクラスを使用して、単一の HTTP リクエストで複数のオブジェクトを削除する手順を示しています。

複数オブジェクトの削除 (バージョニング非対応のバケット)

1	AWS 認証情報を指定して、`AmazonS3Client` クラスのインスタンスを作成します。
2	`DeleteObjectsRequest` クラスのインスタンスを作成し、削除するオブジェクトキーのリストを指定します。
3	`AmazonS3Client.deleteObjects` メソッドを実行します。

以下の Java コード例は、前述のステップの例です。

```
DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest(buck
etName);

List<KeyVersion> keys = new ArrayList<KeyVersion>();
keys.add(new KeyVersion(keyName1));
keys.add(new KeyVersion(keyName2));
keys.add(new KeyVersion(keyName3));

multiObjectDeleteRequest.setKeys(keys);

try {
    DeleteObjectsResult delObjRes = s3Client.deleteObjects(multiObjectDelete
Request);
   System.out.format("Successfully deleted all the %s items.\n", delObjRes.get
DeletedObjects().size());

} catch (MultiObjectDeleteException e) {
    // Process exception.
}
```

例外が発生した場合は、`MultiObjectDeleteException` を調べることで、削除が失敗したオブジェクトと失敗の理由を確認できます。以下の Java の例を参照してください。

```
System.out.format("%s \n", e.getMessage());
System.out.format("No. of objects successfully deleted = %s\n", e.getDeletedOb
jects().size());
System.out.format("No. of objects failed to delete = %s\n", e.ge
tErrors().size());
System.out.format("Printing error data...\n");
for (DeleteError deleteError : e.getErrors()){
    System.out.format("Object Key: %s\t%s\t%s\n",
            deleteError.getKey(), deleteError.getCode(), deleteError.getMes
sage());
}
```

以下のタスクは、バージョニング対応のバケットからオブジェクトを削除する手順を示しています。

複数オブジェクトの削除 (バージョニング対応のバケット)

1	AWS 認証情報を指定して、`AmazonS3Client` クラスのインスタンスを作成します。
2	`DeleteObjectsRequest` クラスのインスタンスを作成し、削除するオブジェクトのオブジェクトキー (およびオプションでバージョン ID) のリストを指定します。 削除するオブジェクトのバージョン ID を指定した場合、Amazon S3 はそのバージョンのオブジェクトを削除します。削除するオブジェクトのバージョン ID を指定しない場合、Amazon S3 は削除マーカーを追加します。詳細については、「1 件のリクエストで 1 つのオブジェクトを削除 (p. 254)」を参照してください。
3	`AmazonS3Client.deleteObjects` メソッドを実行します。

以下の Java コード例は、前述のステップの例です。

```
List<KeyVersion> keys = new ArrayList<KeyVersion>();
// Provide a list of object keys and versions.
```

```
DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest(buck
etName)
.withKeys(keys);

try {
    DeleteObjectsResult delObjRes = s3Client.deleteObjects(multiObjectDelete
Request);
   System.out.format("Successfully deleted all the %s items.\n", delObjRes.get
DeletedObjects().size());

} catch (MultiObjectDeleteException e) {
    // Process exception.
}
```

Example 1: （バージョニング非対応のバケットからの）複数オブジェクトの削除

次の Java コード例では、Multi-Object Delete API を使用して、バージョニング非対応のバケットからオブジェクトを削除します。この例では、最初にサンプルオブジェクトをバケットにアップロードし、次に `deleteObjects` メソッドを使用して単一リクエストでオブジェクトを削除します。

作業サンプルを作成およびテストする方法については、「Java コード例のテスト (p. 477)」を参照してください。

```
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.CannedAccessControlList;
import com.amazonaws.services.s3.model.DeleteObjectsRequest;
import com.amazonaws.services.s3.model.DeleteObjectsResult;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.PutObjectRequest;
import com.amazonaws.services.s3.model.DeleteObjectsRequest.KeyVersion;
import com.amazonaws.services.s3.model.MultiObjectDeleteException.DeleteError;
import com.amazonaws.services.s3.model.MultiObjectDeleteException;
import com.amazonaws.services.s3.model.PutObjectResult;

public class S3Sample {

    static String bucketName = "*** Provide a bucket name ***";
    static AmazonS3Client s3Client;

    public static void main(String[] args) throws IOException {

        try {
            s3Client = new AmazonS3Client(new PropertiesCredentials(
                    S3Sample.class
                    .getResourceAsStream("AwsCredentials.properties")));
           // Upload sample objects.Because the bucket is not version-enabled,

            // the KeyVersions list returned will have null values for version
 IDs.
            List<KeyVersion> keysAndVersions1 = putObjects(3);

            // Delete specific object versions.
            multiObjectNonVersionedDelete(keysAndVersions1);

        } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException.");
            System.out.println("Error Message:    " + ase.getMessage());
            System.out.println("HTTP Status Code: " + ase.getStatusCode());
            System.out.println("AWS Error Code:   " + ase.getErrorCode());
            System.out.println("Error Type:       " + ase.getErrorType());
            System.out.println("Request ID:       " + ase.getRequestId());
        } catch (AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException.");
```

```
            System.out.println("Error Message: " + ace.getMessage());
        }
    }

    static List<KeyVersion> putObjects(int number) {
        List<KeyVersion> keys = new ArrayList<KeyVersion>();
        String content = "This is the content body!";
        for (int i = 0; i < number; i++) {
            String key = "ObjectToDelete-" + new Random().nextInt();
            ObjectMetadata metadata = new ObjectMetadata();
            metadata.setHeader("Subject", "Content-As-Object");
            metadata.setHeader("Content-Length", content.length());
            PutObjectRequest request = new PutObjectRequest(bucketName, key,
                    new ByteArrayInputStream(content.getBytes()), metadata)
                    .withCannedAcl(CannedAccessControlList.AuthenticatedRead);

            PutObjectResult response = s3Client.putObject(request);
          KeyVersion keyVersion = new KeyVersion(key, response.getVersionId());

            keys.add(keyVersion);
        }
        return keys;
    }

    static void multiObjectNonVersionedDelete(List<KeyVersion> keys) {

        // Multi-object delete by specifying only keys (no version ID).
       DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest(

                bucketName).withQuiet(false);

        // Create request that include only object key names.
        List<KeyVersion> justKeys = new ArrayList<KeyVersion>();
        for (KeyVersion key : keys) {
            justKeys.add(new KeyVersion(key.getKey()));
        }
        multiObjectDeleteRequest.setKeys(justKeys);
        // Execute DeleteObjects - Amazon S3 add delete marker for each object

        // deletion. The objects no disappear from your bucket (verify).
        DeleteObjectsResult delObjRes = null;
        try {
            delObjRes = s3Client.deleteObjects(multiObjectDeleteRequest);
            System.out.format("Successfully deleted all the %s items.\n", de
lObjRes.getDeletedObjects().size());
        } catch (MultiObjectDeleteException mode) {
            printDeleteResults(mode);
        }
    }
    static void printDeleteResults(MultiObjectDeleteException mode) {
        System.out.format("%s \n", mode.getMessage());
        System.out.format("No. of objects successfully deleted = %s\n",
mode.getDeletedObjects().size());
        System.out.format("No. of objects failed to delete = %s\n", mode.ge
tErrors().size());
        System.out.format("Printing error data...\n");
        for (DeleteError deleteError : mode.getErrors()){
            System.out.format("Object Key: %s\t%s\t%s\n",
```

```
                    deleteError.getKey(), deleteError.getCode(), deleteError.get
Message());
            }
        }
    }
```

Example 2: Multi-Object Delete (バージョニング対応のバケット)

次の Java コード例では、Multi-Object Delete API を使用して、バージョニング対応のバケットからオブジェクトを削除します。

サンプルをテストする前に、サンプルバケットを作成し、例にバケット名を指定する必要があります。バケットは、AWS Management Console を使用して作成できます。

この例は以下のアクションを実行します。

1. バケットに対してバージョニングを有効にします。
2. バージョンを指定した削除を実行します。
   この例では、最初にサンプルオブジェクトをアップロードします。それに対するレスポンスとして、アップロードした各サンプルオブジェクトのバージョン ID が Amazon S3 から返されます。この例では、次に Multi-Object Delete API を使用してこれらのオブジェクトを削除します。リクエストの中で、オブジェクトキーとバージョン ID の両方を指定します (つまり、バージョンを指定した削除)。
3. バージョンを指定しない削除を実行します。

   この例では、新しいサンプルオブジェクトをアップロードします。次に、Multi-Object API を使用してオブジェクトを削除します。ただし、リクエスト内ではオブジェクトキーのみを指定しています。この場合、Amazon S3 は削除マーカーを追加し、オブジェクトはバケットに表示されなくなります。
4. 削除マーカーを削除します。

   マーカーの削除がどのように動作するかを示すために、サンプルでは削除マーカーを削除しています。Multi-Object Delete リクエストで、オブジェクトキーと、前のステップでレスポンスとして受け取った削除マーカーのバージョン ID を指定します。このアクションにより、オブジェクトがバケットに再び表示されます。

作業サンプルを作成およびテストする方法については、「Java コード例のテスト (p. 477)」を参照してください。

```
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.BucketVersioningConfiguration;
import com.amazonaws.services.s3.model.CannedAccessControlList;
import com.amazonaws.services.s3.model.DeleteObjectsRequest;
import com.amazonaws.services.s3.model.DeleteObjectsResult;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.PutObjectRequest;
import com.amazonaws.services.s3.model.DeleteObjectsRequest.KeyVersion;
import com.amazonaws.services.s3.model.DeleteObjectsResult.DeletedObject;
import com.amazonaws.services.s3.model.MultiObjectDeleteException.DeleteError;
import com.amazonaws.services.s3.model.MultiObjectDeleteException;
import com.amazonaws.services.s3.model.PutObjectResult;
import com.amazonaws.services.s3.model.SetBucketVersioningConfigurationRequest;

public class S3Sample {
```

```
    static String bucketName = "*** Provide a bucket name ***";
    static AmazonS3Client s3Client;

    public static void main(String[] args) throws IOException {

        try {
            s3Client = new AmazonS3Client(new PropertiesCredentials(
                    S3Sample.class
                   .getResourceAsStream("AwsCredentials.properties")));

            // 1. Enable versioning on the bucket.
            enableVersioningOnBucket(s3Client, bucketName);

            // 2a. Upload sample objects.
            List<KeyVersion> keysAndVersions1 = putObjects(3);
            // 2b. Delete specific object versions.
            multiObjectVersionedDelete(keysAndVersions1);

            // 3a. Upload samples objects.
            List<KeyVersion> keysAndVersions2 = putObjects(3);
            // 3b. Delete objects using only keys. Amazon S3 creates a delete
marker and
            // returns its version Id in the response.
            DeleteObjectsResult response = multiObjectNonVersionedDelete(key
sAndVersions2);
            // 3c. Additional exercise - using multi-object versioned delete,
remove the
            // delete markers received in the preceding response. This results
 in your objects
            // reappear in your bucket
            multiObjectVersionedDeleteRemoveDeleteMarkers(response);

        } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException.");
            System.out.println("Error Message:    " + ase.getMessage());
            System.out.println("HTTP Status Code: " + ase.getStatusCode());
            System.out.println("AWS Error Code:   " + ase.getErrorCode());
            System.out.println("Error Type:       " + ase.getErrorType());
            System.out.println("Request ID:       " + ase.getRequestId());
        } catch (AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException.");
            System.out.println("Error Message: " + ace.getMessage());
        }
    }

    static void enableVersioningOnBucket(AmazonS3Client s3Client,
            String bucketName) {
        BucketVersioningConfiguration config = new BucketVersioningConfigura
tion()
                .withStatus(BucketVersioningConfiguration.ENABLED);
        SetBucketVersioningConfigurationRequest setBucketVersioningConfigura
tionRequest = new SetBucketVersioningConfigurationRequest(
                bucketName, config);
       s3Client.setBucketVersioningConfiguration(setBucketVersioningConfigura
tionRequest);
    }

    static List<KeyVersion> putObjects(int number) {
```

```
        List<KeyVersion> keys = new ArrayList<KeyVersion>();
        String content = "This is the content body!";
        for (int i = 0; i < number; i++) {
            String key = "ObjectToDelete-" + new Random().nextInt();
            ObjectMetadata metadata = new ObjectMetadata();
            metadata.setHeader("Subject", "Content-As-Object");
            metadata.setHeader("Content-Length", content.length());
            PutObjectRequest request = new PutObjectRequest(bucketName, key,
                    new ByteArrayInputStream(content.getBytes()), metadata)
                    .withCannedAcl(CannedAccessControlList.AuthenticatedRead);

            PutObjectResult response = s3Client.putObject(request);
          KeyVersion keyVersion = new KeyVersion(key, response.getVersionId());

            keys.add(keyVersion);
        }
        return keys;
    }

    static void multiObjectVersionedDelete(List<KeyVersion> keys) {
       DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest(

                bucketName).withKeys(keys);

        DeleteObjectsResult delObjRes = null;
        try {
            delObjRes = s3Client.deleteObjects(multiObjectDeleteRequest);
            System.out.format("Successfully deleted all the %s items.\n", de
lObjRes.getDeletedObjects().size());
        } catch(MultiObjectDeleteException mode) {
            printDeleteResults(mode);
        }
    }

    static DeleteObjectsResult multiObjectNonVersionedDelete(List<KeyVersion>
keys) {

        // Multi-object delete by specifying only keys (no version ID).
       DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest(

                bucketName);

        // Create request that include only object key names.
        List<KeyVersion> justKeys = new ArrayList<KeyVersion>();
        for (KeyVersion key : keys) {
            justKeys.add(new KeyVersion(key.getKey()));
        }

        multiObjectDeleteRequest.setKeys(justKeys);
        // Execute DeleteObjects - Amazon S3 add delete marker for each object

        // deletion. The objects no disappear from your bucket (verify).
        DeleteObjectsResult delObjRes = null;
        try {
            delObjRes = s3Client.deleteObjects(multiObjectDeleteRequest);
            System.out.format("Successfully deleted all the %s items.\n", de
lObjRes.getDeletedObjects().size());
        } catch (MultiObjectDeleteException mode) {
```

```
            printDeleteResults(mode);
        }
        return delObjRes;
    }
    static void multiObjectVersionedDeleteRemoveDeleteMarkers(
            DeleteObjectsResult response) {

        List<KeyVersion> keyVersionList = new ArrayList<KeyVersion>();
        for (DeletedObject deletedObject : response.getDeletedObjects()) {
            keyVersionList.add(new KeyVersion(deletedObject.getKey(),
                    deletedObject.getDeleteMarkerVersionId()));
        }
        // Create a request to delete the delete markers.
        DeleteObjectsRequest multiObjectDeleteRequest2 = new DeleteObjects
Request(
                bucketName).withKeys(keyVersionList);

        // Now delete the delete marker bringing your objects back to the
bucket.
        DeleteObjectsResult delObjRes = null;
        try {
            delObjRes = s3Client.deleteObjects(multiObjectDeleteRequest2);
            System.out.format("Successfully deleted all the %s items.\n", de
lObjRes.getDeletedObjects().size());
        } catch (MultiObjectDeleteException mode) {
            printDeleteResults(mode);
        }
    }
    static void printDeleteResults(MultiObjectDeleteException mode) {
        System.out.format("%s \n", mode.getMessage());
        System.out.format("No. of objects successfully deleted = %s\n",
mode.getDeletedObjects().size());
        System.out.format("No. of objects failed to delete = %s\n", mode.ge
tErrors().size());
        System.out.format("Printing error data...\n");
        for (DeleteError deleteError : mode.getErrors()){
            System.out.format("Object Key: %s\t%s\t%s\n",
                 deleteError.getKey(), deleteError.getCode(), deleteError.get
Message());
        }
    }
}
```

AWS SDK for .NET を使用した複数オブジェクトの削除

以下のタスクは、AWS SDK for .NET のクラスを使用して、単一の HTTP リクエストで複数のオブジェクトを削除する手順を示しています。

複数オブジェクトの削除 (バージョニング非対応のバケット)

1	AWS 認証情報を指定して、`AmazonS3Client` クラスのインスタンスを作成します。
2	`DeleteObjectsRequest` クラスのインスタンスを作成し、削除するオブジェクトキーのリストを指定します。

3	AmazonS3Client.DeleteObjects メソッドを実行します。 1 つ以上のオブジェクトを削除できなかった場合、Amazon S3 は DeleteObjectsException をスローします。

以下の C# コード例は、前述のステップを実装したものです。

```
DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest();
multiObjectDeleteRequest.BucketName = bucketName;

multiObjectDeleteRequest.AddKey(*** Object1 Key ***);
multiObjectDeleteRequest.AddKey(*** Object2 Key ***);
multiObjectDeleteRequest.AddKey(*** Object3 Key ***);

try
{
  DeleteObjectsResponse response = client.DeleteObjects(multiObjectDelete
Request);
  Console.WriteLine("Successfully deleted all the {0} items", response.Delete
dObjects.Count);
}
catch (DeleteObjectsException e)
{
  // Process exception.
}
```

例外が発生した場合は、DeleteObjectsException を調べることで、削除が失敗したオブジェクトと失敗の理由を確認できます。以下の C# コードの例を参照してください。

```
var errorResponse = e.ErrorResponse;
Console.WriteLine("No. of objects successfully deleted = {0}", errorResponse.De
letedObjects.Count);
Console.WriteLine("No. of objects failed to delete = {0}", errorResponse.Delet
eErrors.Count);
Console.WriteLine("Printing error data...");
foreach (DeleteError deleteError in errorResponse.DeleteErrors)
{
   Console.WriteLine("Object Key: {0}\t{1}\t{2}", deleteError.Key, deleteEr
ror.Code, deleteError.Message);
}
```

以下のタスクは、バージョニング対応のバケットからオブジェクトを削除する手順を示しています。

複数オブジェクトの削除 (バージョニング対応のバケット)

1	AWS 認証情報を指定して、AmazonS3Client クラスのインスタンスを作成します。
2	DeleteObjectsRequest クラスのインスタンスを作成し、削除するオブジェクトのオブジェクトキー (およびオプションでバージョン ID) のリストを指定します。 削除するオブジェクトのバージョン ID を指定すると、Amazon S3 はそのバージョンのオブジェクトを削除します。削除するオブジェクトのバージョン ID を指定しない場合、Amazon S3 は削除マーカーを追加します。詳細については、「1 件のリクエストで 1 つのオブジェクトを削除 (p. 254)」を参照してください。
3	AmazonS3Client.DeleteObjects メソッドを実行します。

以下の C# コード例は、前述のステップの例です。

```
List<KeyVersion> keysAndVersions = new List<KeyVersion>();
// provide a list of object keys and versions.

DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest
  {
    BucketName = bucketName,
    Keys = keysAndVersions
  };

try
{
  DeleteObjectsResponse response = client.DeleteObjects(multiObjectDelete
Request);
  Console.WriteLine("Successfully deleted all the {0} items", response.Delete
dObjects.Count);
}
catch (DeleteObjectsException e)
{
  // Process exception.
}
```

Example 1: Multi-Object Delete (バージョニング非対応のバケット)

次の C# コード例では、Multi-Object API を使用して、バージョニング非対応のバケットからオブジェクトを削除します。この例では、最初にサンプルオブジェクトをバケットにアップロードし、次に DeleteObjects メソッドを使用して単一リクエストでオブジェクトを削除します。この例では、バージョン ID が null であるため、DeleteObjectsRequest でオブジェクトキー名のみを指定します。

作業サンプルを作成およびテストする方法については、「.NET コード例のテスト (p. 478)」を参照してください。

```
using System;
using System.Collections.Generic;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples
{
  class S3Sample
  {
    static string bucketName = "*** Provide a bucket name ***";
    static AmazonS3Client client;

    public static void Main(string[] args)
    {
      using (client = new AmazonS3Client())
      {
        var keysAndVersions = PutObjects(3);
        // Delete the objects.
        MultiObjectNonVersionedDelete(keysAndVersions);
      }

      Console.WriteLine("Click ENTER to continue.....");
      Console.ReadLine();
    }

    static void MultiObjectNonVersionedDelete(List<KeyVersion> keys)
    {
      // a. multi-object delete by specifying the key names and version IDs.
      DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest

      {
        BucketName = bucketName,
        Keys = keys // This includes the object keys and null version IDs.
      };

      try
      {
       DeleteObjectsResponse response = client.DeleteObjects(multiObjectDelete
Request);
       Console.WriteLine("Successfully deleted all the {0} items", response.De
letedObjects.Count);
      }
      catch (DeleteObjectsException e)
      {
        PrintDeletionReport(e);
      }
    }
```

```
    private static void PrintDeletionReport(DeleteObjectsException e)
    {
      var errorResponse = e.ErrorResponse;
      Console.WriteLine("No. of objects successfully deleted = {0}", errorRes
ponse.DeletedObjects.Count);
     Console.WriteLine("No. of objects failed to delete = {0}", errorResponse.De
leteErrors.Count);
      Console.WriteLine("Printing error data...");
      foreach (DeleteError deleteError in errorResponse.DeleteErrors)
      {
        Console.WriteLine("Object Key: {0}\t{1}\t{2}", deleteError.Key, delet
eError.Code, deleteError.Message);
      }
    }

    static List<KeyVersion> PutObjects(int number)
    {
      List<KeyVersion> keys =
                  new List<KeyVersion>();

      for (int i = 0; i < number; i++)
      {
        string key = "ObjectToDelete-" + new System.Random().Next();
        PutObjectRequest request = new PutObjectRequest
        {
          BucketName = bucketName,
          Key = key,
          ContentBody = "This is the content body!",

        };

        PutObjectResponse response = client.PutObject(request);
        KeyVersion keyVersion = new KeyVersion(key, response.VersionId);

        keys.Add(keyVersion);
      }
      return keys;
    }
  }
}
```

Example 2: Multi-Object Delete (バージョニング対応のバケット)

次の C# コード例では、Multi-Object API を使用して、バージョニング対応のバケットからオブジェクトを削除します。この例は、DeleteObjects Multi-Object Delete API の使用法を示すだけでなく、バージョニング対応のバケットでバージョニングがどのように機能するかを示しています。

サンプルをテストする前に、サンプル内でサンプルバケットを作成し、バケット名を付ける必要があります。バケットは、AWS Management Console を使用して作成できます。

この例は以下のアクションを実行します。

1. バケットに対してバージョニングを有効にします。
2. バージョンを指定した削除を実行します。
   この例では、最初にサンプルオブジェクトをアップロードします。それに対するレスポンスとして、アップロードした各サンプルオブジェクトのバージョン ID が Amazon S3 から返されます。この例では、次に Multi-Object Delete API を使用してこれらのオブジェクトを削除します。リクエストの中で、オブジェクトキーとバージョン ID の両方を指定します (つまり、バージョンを指定した削除)。
3. バージョンを指定しない削除を実行します。

   この例では、新しいサンプルオブジェクトをアップロードします。次に、Multi-Object API を使用してオブジェクトを削除します。ただし、リクエスト内ではオブジェクトキーのみを指定しています。この場合、Amazon S3 は削除マーカーを追加し、オブジェクトはバケットに表示されなくなります。
4. 削除マーカーを削除します。

   マーカーの削除がどのように動作するかを示すために、サンプルでは削除マーカーを削除しています。Multi-Object Delete リクエストで、オブジェクトキーと、前のステップでレスポンスとして受け取った削除マーカーのバージョン ID を指定します。このアクションにより、オブジェクトがバケットに再び表示されます。

作業サンプルを作成およびテストする方法については、「.NET コード例のテスト (p. 478)」を参照してください。

```
using System;
using System.Collections.Generic;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples
{
  class S3Sample
  {
    static string bucketName = "*** Provide a bucket name ***";
    static AmazonS3Client client;

    public static void Main(string[] args)
    {
      using (client = new AmazonS3Client())
      {

        // 1. Enable versioning on the bucket.
        EnableVersioningOnBucket(bucketName);

        // 2a. Upload the sample objects.
        var keysAndVersions1 = PutObjects(3);
        // 2b. Delete the specific object versions.
        VersionedDelete(keysAndVersions1);
```

```
        // 3a. Upload the sample objects.
        var keysAndVersions2 = PutObjects(3);

        // 3b. Delete objects using only keys. Amazon S3 creates a delete
marker and
        // returns its version Id in the response.
        List<DeletedObject> deletedObjects = NonVersionedDelete(keysAndVer
sions2);

        // 3c. Additional exercise - using a multi-object versioned delete,
remove the
        // delete markers received in the preceding response. This results in
your objects
        // reappearing in your bucket.
        RemoveMarkers(deletedObjects);
      }

      Console.WriteLine("Click ENTER to continue.....");
      Console.ReadLine();
    }

    private static void PrintDeletionReport(DeleteObjectsException e)
    {
      var errorResponse = e.ErrorResponse;
      Console.WriteLine("No. of objects successfully deleted = {0}", errorRes
ponse.DeletedObjects.Count);
     Console.WriteLine("No. of objects failed to delete = {0}", errorResponse.De
leteErrors.Count);
      Console.WriteLine("Printing error data...");
      foreach (DeleteError deleteError in errorResponse.DeleteErrors)
      {
        Console.WriteLine("Object Key: {0}\t{1}\t{2}", deleteError.Key, delet
eError.Code, deleteError.Message);
      }
    }


    static void EnableVersioningOnBucket(string bucketName)
    {
     SetBucketVersioningRequest setBucketVersioningRequest = new SetBucketVer
sioningRequest
      {
        BucketName = bucketName,
        VersioningConfig = new S3BucketVersioningConfig { Status = "Enabled" }

      };
      client.SetBucketVersioning(setBucketVersioningRequest);
    }

    static void VersionedDelete(List<KeyVersion> keys)
    {
     // a. Perform a multi-object delete by specifying the key names and version
 IDs.
      DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest

        {
          BucketName = bucketName,
```

```
          Keys = keys // This includes the object keys and specific version
IDs.
        };
      try
      {
        Console.WriteLine("Executing VersionedDelete...");
       DeleteObjectsResponse response = client.DeleteObjects(multiObjectDelete
Request);
       Console.WriteLine("Successfully deleted all the {0} items", response.De
letedObjects.Count);
      }
      catch (DeleteObjectsException e)
      {
        PrintDeletionReport(e);
      }
    }

    static List<DeletedObject> NonVersionedDelete(List<KeyVersion> keys)
    {
      // Create a request that includes only the object key names.
     DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest();

      multiObjectDeleteRequest.BucketName = bucketName;

      foreach (var key in keys)
      {
        multiObjectDeleteRequest.AddKey(key.Key);
      }
      // Execute DeleteObjects - Amazon S3 add delete marker for each object
      // deletion. The objects disappear from your bucket.
      // You can verify that using the Amazon S3 console.
      DeleteObjectsResponse response;
      try
      {
        Console.WriteLine("Executing NonVersionedDelete...");
        response = client.DeleteObjects(multiObjectDeleteRequest);
       Console.WriteLine("Successfully deleted all the {0} items", response.De
letedObjects.Count);
      }
      catch (DeleteObjectsException e)
      {
        PrintDeletionReport(e);
        throw; // Some deletes failed. Investigate before continuing.
      }

      return response.DeletedObjects; // This response contains the DeletedOb
jects list which we use to delete the delete markers.
    }

    private static void RemoveMarkers(List<DeletedObject> deletedObjects)
    {
      List<KeyVersion> keyVersionList = new List<KeyVersion>();

      foreach (var deletedObject in deletedObjects)
      {
        KeyVersion keyVersion = new KeyVersion(deletedObject.Key, deletedOb
ject.DeleteMarkerVersionId);
        keyVersionList.Add(keyVersion);
```

```
      }
      // Create another request to delete the delete markers.
      var multiObjectDeleteRequest = new DeleteObjectsRequest
      {
        BucketName = bucketName,
        Keys = keyVersionList
      };

     // Now, delete the delete marker to bring your objects back to the bucket.


      try
      {
        Console.WriteLine("Removing the delete markers .....");
        var deleteObjectResponse = client.DeleteObjects(multiObjectDelete
Request);
        Console.WriteLine("Successfully deleted all the {0} delete markers",
deleteObjectResponse.DeletedObjects.Count);
      }
      catch (DeleteObjectsException e)
      {
        PrintDeletionReport(e);
      }
    }

    static List<KeyVersion> PutObjects(int number)
    {
      List<KeyVersion> keys =
                  new List<KeyVersion>();

      for (int i = 0; i < number; i++)
      {
        string key = "ObjectToDelete-" + new System.Random().Next();
        PutObjectRequest request = new PutObjectRequest
        {
          BucketName = bucketName,
          Key = key,
          ContentBody = "This is the content body!",

        };

        PutObjectResponse response = client.PutObject(request);
        KeyVersion keyVersion = new KeyVersion(key, response.VersionId);

        keys.Add(keyVersion);
      }
      return keys;
    }
  }
}
```

AWS SDK for PHP を使用した複数オブジェクトの削除

このトピックでは、AWS SDK for PHP のクラスを使用して、バージョニング対応および非対応の Amazon S3 バケットから複数のオブジェクトを削除する手順を示します。バージョニングの詳細については、「バージョニングの使用 (p. 388)」を参照してください。

Note

このトピックでは、既に AWS SDK for PHP の使用と PHP サンプルの実行 (p. 479) の説明が実行されていて、AWS SDK for PHP が正しくインストールされていることを前提としています。

以下のタスクは、PHP SDK のクラスを使用して、バージョニング非対応のバケットから複数のオブジェクトを削除する手順を示しています。

(バージョニング非対応のバケットからの) 複数オブジェクトの削除

1	自分の AWS 認証情報または IAM ユーザー認証情報と共に Aws\S3\S3Client クラスの factory() メソッドを使用して、Amazon S3 クライアントのインスタンスを作成します。
2	Aws\S3\S3Client::deleteObjects() メソッドを実行します。バケット名とオブジェクトキーの配列をパラメータとして指定する必要があります。最大 1,000 個のキーを指定できます。

以下の PHP コードは、Amazon S3 のバージョニング非対応のバケットから複数のオブジェクトを実際に削除している例です。

```
use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname1 = '*** Your Object Key1 ***';
$keyname2 = '*** Your Object Key2 ***';
$keyname3 = '*** Your Object Key3 ***';

$s3 = S3Client::factory(array(
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));

// Delete objects from a bucket
$result = $s3->deleteObjects(array(
    'Bucket'  => $bucket,
    'Objects' => array(
        array('Key' => $keyname1),
        array('Key' => $keyname2),
        array('Key' => $keyname3),
    )
));
```

以下のタスクは、Amazon S3 のバージョニング対応のバケットから複数のオブジェクトを削除する手順を示しています。

(バージョニング対応のバケットからの) 複数オブジェクトの削除

1	自分の AWS 認証情報または IAM ユーザー認証情報と共に `Aws\S3\S3Client` クラスの `factory()` メソッドを使用して、Amazon S3 クライアントのインスタンスを作成します。

2	`Aws\S3\S3Client::deleteObjects()` メソッドを実行し、削除するオブジェクトのオブジェクトキー (およびオプションでバージョン ID) のリストを指定します。 削除するオブジェクトのバージョン ID を指定した場合、Amazon S3 はそのバージョンのオブジェクトのみを削除します。削除するオブジェクトのバージョン ID を指定しない場合、Amazon S3 は削除マーカーを追加します。詳細については、「1 件のリクエストで 1 つのオブジェクトを削除 (p. 254)」を参照してください。

以下の PHP コードは、Amazon S3 のバージョニング対応のバケットから複数のオブジェクトを実際に削除している例です。

```
use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';
$versionId1 = '*** Your Object Key Version ID1 ***';
$versionId2 = '*** Your Object Key Version ID2 ***';
$versionId3 = '*** Your Object Key Version ID3 ***';

$s3 = S3Client::factory(array(
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));

// Delete object versions from a versioning-enabled bucket.
$result = $s3->deleteObjects(array(
    'Bucket'  => $bucket,
    'Objects' => array(
        array('Key' => $keyname, 'VersionId' => $versionId1),
        array('Key' => $keyname, 'VersionId' => $versionId2),
        array('Key' => $keyname, 'VersionId' => $versionId3),
    )
));
```

Amazon S3 から返されるレスポンスには、削除されたオブジェクトと、エラー (例えば、アクセス許可エラー) が原因で削除できなかったオブジェクトが示されます。

以下の PHP コード例は、削除されたオブジェクトのオブジェクトキーを出力しています。また、削除されなかったオブジェクトキーと、関連するエラーメッセージも出力しています。

```
echo "The following objects were deleted successfully:\n";
foreach ($result['Deleted'] as $object) {
    echo "Key: {$object['Key']}, VersionId: {$object['VersionId']}\n";
}

echo "\nThe following objects could not be deleted:\n";
foreach ($result['Errors'] as $object) {
    echo "Key: {$object['Key']}, VersionId: {$object['VersionId']}\n";
}
```

Example 1: (バージョニング非対応のバケットからの) 複数オブジェクトの削除

以下の PHP コード例では、`deleteObjects()` メソッドを使用して、バージョニング非対応のバケットから複数のオブジェクトを削除しています。

この例は以下のアクションを実行します。

1. Aws\S3\S3Client::putObject() メソッドを使用して、少数のオブジェクトを作成します。
2. Aws\S3\S3Client::listObjects() メソッドを使用して、作成したオブジェクトをリストし、それらのオブジェクトのキーを取得します。
3. `Aws\S3\S3Client::deleteObjects()` メソッドを使用して、バージョニング非対応のバケットからのオブジェクト削除を実行します。

PHP 例の実行については、このガイド内の「PHP サンプルの実行 (p. 480)」を参照してください。

```
<?php

// Include the AWS SDK using the Composer autoloader.
require 'vendor/autoload.php';

use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';

// Instantiate the client.
$s3 = S3Client::factory(array(
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));

// 1. Create a few objects.
for ($i = 1; $i <= 3; $i++) {
    $s3->putObject(array(
        'Bucket' => $bucket,
        'Key'    => "key{$i}",
        'Body'   => "content {$i}",
    ));
}

// 2. List the objects and get the keys.
$keys = $s3->listObjects(array('Bucket' => $bucket))
    ->getPath('Contents/*/Key');

// 3. Delete the objects.
$result = $s3->deleteObjects(array(
    'Bucket'  => $bucket,
    'Objects' => array_map(function ($key) {
        return array('Key' => $key);
    }, $keys),
));
```

Example 2: 複数オブジェクトの削除 (バージョニング対応のバケット)

以下の PHP コード例では、`deleteObjects()` メソッドを使用して、バージョニング対応のバケットから複数のオブジェクトを削除します。

この例は以下のアクションを実行します。

1. Aws\S3\S3Client::putBucketVersioning() メソッドを使用して、バケットのバージョニングを有効化します。
2. `Aws\S3\S3Client::putObject()` メソッドを使用して、オブジェクトのバージョンをいくつか作成します。
3. Aws\S3\S3Client::listObjectVersions() メソッドを使用して、作成したバージョンのオブジェクトをリストし、それらのオブジェクトのキーとバージョン ID を取得します。
4. 取得したキーとバージョン ID と共に `Aws\S3\S3Client::deleteObjects()` を使用して、バージョニング対応のバケットからオブジェクトを削除します。
5. `Aws\S3\S3Client::putBucketVersioning()` メソッドを使用して、バケットのバージョニングを無効化します。

PHP 例の実行については、このガイド内の「PHP サンプルの実行 (p. 480)」を参照してください。

```
<?php

// Include the AWS SDK using the Composer autoloader.
require 'vendor/autoload.php';

use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

// Instantiate the client.
$s3 = S3Client::factory(array(
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));

// 1. Enable object versioning for the bucket.
$s3->putBucketVersioning(array(
    'Bucket' => $bucket,
    'Status' => 'Enabled',
));

// 2. Create a few versions of an object.
for ($i = 1; $i <= 3; $i++) {
    $s3->putObject(array(
        'Bucket' => $bucket,
        'Key'    => $keyname,
        'Body'   => "content {$i}",
    ));
}

// 3. List the objects versions and get the keys and version IDs.
$versions = $s3->listObjectVersions(array('Bucket' => $bucket))
    ->getPath('Versions');

// 4. Delete the object versions.
```

```
$result = $s3->deleteObjects(array(
    'Bucket'  => $bucket,
    'Objects' => array_map(function ($version) {
        return array(
            'Key'       => $version['Key'],
            'VersionId' => $version['VersionId']
        );
    }, $versions),
));

echo "The following objects were deleted successfully:\n";
foreach ($result['Deleted'] as $object) {
    echo "Key: {$object['Key']}, VersionId: {$object['VersionId']}\n";
}

echo "\nThe following objects could not be deleted:\n";
foreach ($result['Errors'] as $object) {
    echo "Key: {$object['Key']}, VersionId: {$object['VersionId']}\n";
}

// 5. Disable object versioning for the bucket.
$s3->putBucketVersioning(array(
    'Bucket' => $bucket,
    'Status' => 'Suspended',
));
```

関連リソース

- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client クラス」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::factory() メソッド」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::deleteObject() メソッド」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::listObjects() メソッド」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::listObjectVersions() メソッド」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::putObject() メソッド」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::putBucketVersioning() メソッド」
- 「AWS SDK for PHP – Amazon S3」
- 「AWS SDK for PHP」のドキュメント

REST API を使用した複数オブジェクトの削除

AWS SDK を使用して、Multi-Object Delete API で複数のオブジェクトを削除できます。ただし、アプリケーションで必要な場合は、REST リクエストを直接送信できます。詳細については、『*Amazon Simple Storage Service API リファレンス*』の「Delete Multiple Objects」を参照してください。

オブジェクトの復元

Topics

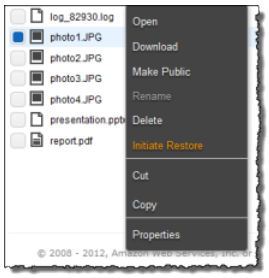
- REST API を使用した 1 つのオブジェクトの復元 (p. 293)

復元リクエストを開始しないと、アーカイブされたオブジェクトにアクセスできません。このセクションでは、プログラムを使用して復元リクエストを開始する方法を説明します。オブジェクトのアーカイブについては、「オブジェクトのライフサイクル管理 (p. 114)」を参照してください。

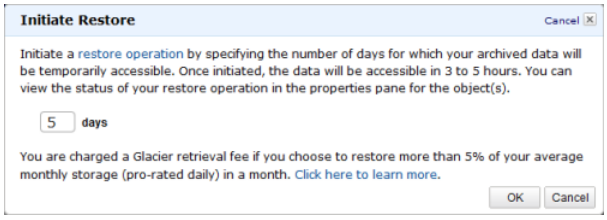
復元操作がコピーできるアーカイブにアクセスできるまで、約 4 時間かかります。復元の開始後、復元のステータスをコンソール (「Amazon S3 コンソールを使用した Glacier オブジェクトの復元 (p. 122)」参照) で確認したり、HEAD リクエストを送信して復元ステータス情報が含まれるオブジェクトのメタデータをプログラムによって取得したりできます。

Amazon S3 コンソールを使用したオブジェクトの復元

Amazon S3 コンソールを使用して、アーカイブされたオブジェクトのコピーを復元できます。コンソールでオブジェクトを右クリックして、[Initiate Restore] を選択します。

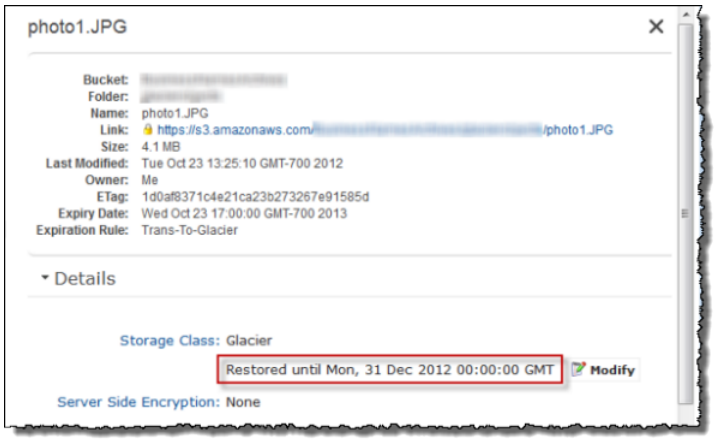


オブジェクトのコピーの復元にかける日数を指定します。



Amazon S3 が復元を完了するには、約 3 ~ 5 時間かかります。コンソールのオブジェクトプロパティに、オブジェクトの復元ステータスが表示されます。

オブジェクトのコピーが復元されると、コンソールのオブジェクトプロパティに、オブジェクトが復元されたことと、Amazon S3 が復元済みコピーを削除するタイミングが表示されます。また、復元期間を修正するためのオプションも表示されます。



なお、アーカイブを復元するときは、そのアーカイブに加えて、一時的に復元されたコピーについても料金が発生します。料金表については、「*Amazon S3 製品詳細*ページの「料金表」」セクションを参照してください。

Amazon S3 によって復元されたオブジェクトの一時コピーを利用できるのは、指定された期間が終了するまでです。その期間が終了すると、Amazon S3 は復元されたオブジェクトコピーを削除します。復元済みコピーの期限切れ日数は、復元を再び発行することで修正できます。この場合、Amazon S3 は現在の時間からの相対値を使用して有効期限を更新します。

Amazon S3 は、復元リクエストに指定された日数を現在の時間に加算し、得られた日時を翌日の午前 00:00 UTC (協定世界時) に丸めることで、復元済みオブジェクトコピーの有効期限を算出します。例えば、あるオブジェクトが 2012 年 10 月 15 日の午前 10:30 UTC に作成され、復元期間が 3 日に指定された場合、復元コピーは 2012 年 10 月 19 日の午前 00:00 UTC に有効期限切れになり、この時点で Amazon S3 によってオブジェクトコピーが削除されます。

オブジェクトコピーは任意の日数の間、復元できます。ただし、オブジェクトコピーに関連するストレージコストがかかるため、必要な期間のみオブジェクトを復元することをお勧めします。価格情報については、 Amazon S3 製品詳細ページの「Amazon S3 料金表」セクションを参照してください。

AWS SDK for Java を使用したオブジェクトの復元

以下のタスクは、AWS SDK for Java を使用して、アーカイブされたオブジェクトの復元を開始する手順を示しています。

オブジェクトのダウンロード

1	AWS 認証情報を指定して、`AmazonS3Client` クラスのインスタンスを作成します。
2	バケット名、復元するオブジェクトキー、オブジェクトコピーを復元する日数を指定して、`RestoreObjectRequest` クラスのインスタンスを作成します。
3	`AmazonS3.RestoreObject` メソッドのいずれかを実行して、アーカイブの復元を開始します。

以下の Java コード例は、前述のタスクの例です。

```
String bucketName = "examplebucket";
String objectkey = "examplekey";
AmazonS3Client s3Client = new AmazonS3Client();
```

```
RestoreObjectRequest request = new RestoreObjectRequest(bucketName, objectkey,
 2);
s3Client.restoreObject(request);
```

Amazon S3 は復元ステータスをオブジェクトのメタデータ内に維持します。次の Java コードスニペットに示すように、オブジェクトのメタデータを取得し、RestoreInProgress プロパティの値を確認できます。

```
String bucketName = "examplebucket";
String objectkey = "examplekey";
AmazonS3Client s3Client = new AmazonS3Client();

client = new AmazonS3Client();

GetObjectMetadataRequest request = new GetObjectMetadataRequest(bucketName,
objectKey);

ObjectMetadata response = s3Client.getObjectMetadata(request);

Boolean restoreFlag = response.getOngoingRestore();
System.out.format("Restoration status: %s.\n",
          (restoreFlag == true) ? "in progress" : "finished");
```

Example

次の Java コード例は、指定したアーカイブオブジェクトの復元リクエストを開始します。コードを更新し、バケット名と、アーカイブされたオブジェクトキー名を指定する必要があります。作業サンプルを作成およびテストする方法については、「Java コード例のテスト (p. 477)」を参照してください。

```
import java.io.IOException;

import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.AmazonS3Exception;
import com.amazonaws.services.s3.model.GetObjectMetadataRequest;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.RestoreObjectRequest;

public class S3LifecycleRestoreExample {

    public static String bucketName = "*** Provide bucket name ***";
    public static String objectKey = "*** Provide object key name ***";
    public static AmazonS3Client s3Client;

    public static void main(String[] args) throws IOException {
       AmazonS3Client s3Client = new AmazonS3Client(new PropertiesCredentials(

                S3LifecycleRestoreExample.class.getResourceAsStream(
                       "AwsCredentials.properties")));

        try {

          RestoreObjectRequest requestRestore = new RestoreObjectRequest(buck
etName, objectKey, 2);
          s3Client.restoreObject(requestRestore);

          GetObjectMetadataRequest requestCheck = new GetObjectMetadataRe
quest(bucketName, objectKey);
          ObjectMetadata response = s3Client.getObjectMetadata(requestCheck);

          Boolean restoreFlag = response.getOngoingRestore();
          System.out.format("Restoration status: %s.\n",
                  (restoreFlag == true) ? "in progress" : "finished");

        } catch (AmazonS3Exception amazonS3Exception) {
            System.out.format("An Amazon S3 error occurred. Exception: %s",
amazonS3Exception.toString());
        } catch (Exception ex) {
            System.out.format("Exception: %s", ex.toString());
        }
    }
}
```

AWS SDK for .NET を使用したオブジェクトの復元

以下のタスクは、AWS SDK for .NET を使用して、アーカイブされたオブジェクトの復元を開始する手順を示しています。

オブジェクトのダウンロード

1	AWS 認証情報を指定して、AmazonS3 クラスのインスタンスを作成します。
2	バケット名、復元するオブジェクトキー、オブジェクトコピーを復元する日数を指定して、RestoreObjectRequest クラスのインスタンスを作成します。
3	AmazonS3.RestoreObject メソッドのいずれかを実行して、アーカイブの復元を開始します。

以下の C# コード例は、前述のタスクの例です。

```
AmazonS3 client;
string bucketName = "examplebucket";
string objectKey = "examplekey";

client = new AmazonS3Client();

RestoreObjectRequest restoreRequest = new RestoreObjectRequest()
 {
     BucketName = bucketName,
     Key = objectKey,
     Days = 2
 };

client.RestoreObject(restoreRequest);
```

Amazon S3 は復元ステータスをオブジェクトのメタデータ内に維持します。次の C# コードスニペットに示すように、オブジェクトのメタデータを取得し、RestoreInProgress プロパティの値を確認できます。

```
AmazonS3 client;
string bucketName = "examplebucket";
string objectKey = "examplekey";

client = new AmazonS3Client();

GetObjectMetadataRequest metadataRequest = new GetObjectMetadataRequest()
{
      BucketName = bucketName,
      Key = objectKey
};
GetObjectMetadataResponse response = client.GetObjectMetadata(metadataRequest);
Console.WriteLine("Restoration status: {0}", response.RestoreInProgress);
if (response.RestoreInProgress == false)
   Console.WriteLine("Restored object copy expires on: {0}", response.Restore
Expiration);
```

Example

次の C# コード例は、指定したアーカイブオブジェクトの復元リクエストを開始します。コードを更新し、バケット名と、アーカイブされたオブジェクトキー名を指定する必要があります。作業サンプルを作成およびテストする方法については、「.NET コード例のテスト (p. 478)」を参照してください。

```
using System;
using Amazon.S3;
using Amazon.S3.Model;

namespace aws.amazon.com.s3.documentation
{
    class S3Sample
    {
        static string bucketName = "*** provide bucket name ***";
        static string objectKey = "*** archived object keyname ***";

        static AmazonS3 client;

        public static void Main(string[] args)
        {
            try
            {
                using (client = new AmazonS3Client())
                {
                    RestoreArchivedObject(client, bucketName, objectKey);
                    CheckRestorationStatus(client, bucketName, objectKey);
                }

                Console.WriteLine("Example complete. To continue, click
Enter...");
                Console.ReadKey();
            }
            catch (AmazonS3Exception amazonS3Exception)
            {
                Console.WriteLine("S3 error occurred. Exception: " +
amazonS3Exception.ToString());
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception: " + e.ToString());
            }
        }

        static void RestoreArchivedObject(AmazonS3 client, string bucketName,
string objectKey)
        {
            RestoreObjectRequest restoreRequest = new RestoreObjectRequest()
            {
                BucketName = bucketName,
                Key = objectKey,
                Days = 2
            };
            RestoreObjectResponse response = client.RestoreObject(restore
Request);
        }

        static void CheckRestorationStatus(AmazonS3 client, string bucketName,
```

```
 string objectKey)
        {
            GetObjectMetadataRequest metadataRequest = new GetObjectMetadataRe
quest()
            {
                  BucketName = bucketName,
                  Key = objectKey
            };
            GetObjectMetadataResponse response = client.GetObject
Metadata(metadataRequest);
           Console.WriteLine("Restoration status: {0}", response.RestoreInPro
gress);
            if (response.RestoreInProgress == false)
                Console.WriteLine("Restored object copy expires on: {0}", re
sponse.RestoreExpiration);
        }
    }
}
```

REST API を使用した 1 つのオブジェクトの復元

Amazon S3 には、アーカイブの復元を開始するための API が用意されています。詳細については、『*Amazon Simple Storage Service API リファレンス*』の「POST Object restore」を参照してください。

アクセスコントロール

Topics

Amazon S3 では、アクセスコントロールリスト (ACL) 、バケットポリシー、IAM ポリシーを使用してオブジェクトやバケットへのアクセスを管理できます。これらのリストやポリシーは、個別にまたは組み合わせて使用できます。このセクションでは、3 つすべてのアクセスコントロール手法について説明します。

ACL とは、許可のリストです。許可は、1 人の被付与者と、Amazon S3 リソース (バケットやオブジェクト) にアクセスするための 1 つのアクセス許可で構成されています。ACL はアクセス許可を付与するだけで、アクセス許可を拒否することはありません。ACL には、次のタイプの被付与者を含めることができます。

- 特定の AWS アカウント
- すべての AWS アカウント
- 任意の匿名のリクエスト

バケットポリシーは、バケットとバケット内のオブジェクトの両方に対するバケットレベルでのアクセスコントロール管理を提供します。バケットポリシーは、*アクセスポリシー言語*で記述された JSON ステートメントのコレクションです。このポリシーは、Amazon S3 リソースに対する非常に細分化されたアクセスコントロールを提供します。また、1 つのステートメントで多数のオブジェクトに対するアクセス許可を設定できます。

AWS Identity and Access Management (IAM) を使用して、AWS アカウント内に複数のユーザーを作成し、そのユーザーのアクセス許可を IAM ポリシー経由で管理することができます。これらのポリシーはユーザーにアタッチされるため、AWS アカウントで管理されるユーザーのアクセス許可を一元管理できます。バケットポリシーはバケットにアタッチされ、IAM ポリシーはアカウント内の個々のユーザーにアタッチされる点に注意してください。

IAM ポリシーの使用

Topics

AWS Identity and Access Management (IAM) を使用して、AWS アカウント内に複数のユーザーを作成し、そのユーザーにセキュリティ認証情報を割り当てて、アクセス許可を管理することができます。ユーザーのアクセス許可は IAM ポリシーを通じて管理します。これらのポリシーはユーザーにアタッチされるため、AWS アカウント内のユーザーのアクセス許可を一元管理できます。

Amazon S3 は、オブジェクトとバケットに対して、アクセスコントロールリスト (ACL) とバケットのポリシーを使用した、リソースベースのアクセス許可をサポートしています。ACL とバケットのポリシーはバケットとオブジェクトにアタッチされ、どの AWS アカウント (またはその他の関係者のグループ) がどのタイプのアクセス許可を持っているかを定義します。

この両方を IAM ユーザーポリシーと共に使用して、Amazon S3 リソースへのアクセスを制御できます。次の表は、ACL、バケットポリシー、および IAM ポリシーの類似点と相違点をまとめたものです。

アクセスコントロールのタイプ	AWS アカウントレベルの制御	ユーザーレベルの制御	形式
ACL	はい	なし	Amazon S3 が定義する特別な XML ベースの形式
バケットポリシー	はい	はい	アクセスポリシー言語
IAM ポリシー	なし	あり	アクセスポリシー言語

ACL では、他の AWS アカウントにのみ、Amazon S3 リソースへのアクセスを付与できます。IAM ポリシーでは、AWS アカウント内のユーザーにのみ、Amazon S3 リソースへのアクセスを付与できます。バケットポリシーでは、これらの両方を実行できます。

このセクションでは、IAM がバケットポリシーおよび ACL とどのようにして連携するかを説明します。

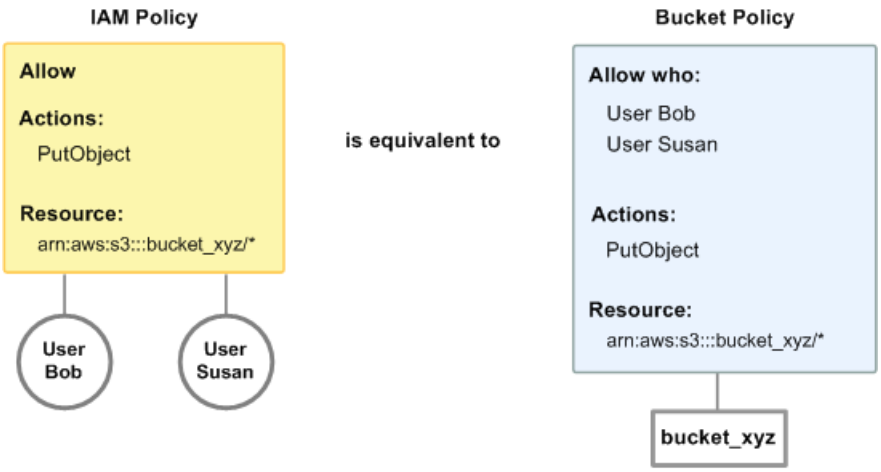
&IAM とバケットポリシーの連携

ユーザーが開発者の AWS アカウントの Amazon S3 リソースの利用をリクエストすると、Amazon は適用可能なすべての ACL、バケットポリシー、および IAM ポリシーを共に評価して、リクエスタにアクセスを付与するかどうかを判断します。

ACL を使用してアクセスを付与することもできますが、このセクションでは、IAM ポリシーとバケットポリシーを使用して、*AWS アカウントのユーザー*に、Amazon S3 リソースへのアクセスを付与する方法を説明します。バケットポリシーまたは IAM ポリシー、またはその両方を使用できます。ほとんどの場合、どちらでも同じ結果が得られます。例えば、次の図は、同じ働きを持つ IAM ポリシーとバケットポリシーを示しています。左側の IAM ポリシーは、AWS アカウントの bucket_xyz というバケットに対する Amazon S3 `PutObject` アクションを許可します。このポリシーは、ユーザーの Bob と

Susan にアタッチされています (つまり、Bob と Susan のアクセス許可がポリシー内に記述されています)。

右側のバケットポリシーは bucket_xyz にアタッチされています。IAM ポリシーの場合と同様、このバケットポリシーは、Bob と Susan に対し、bucket_xyz で `PutObject` にアクセスする許可を付与しています。



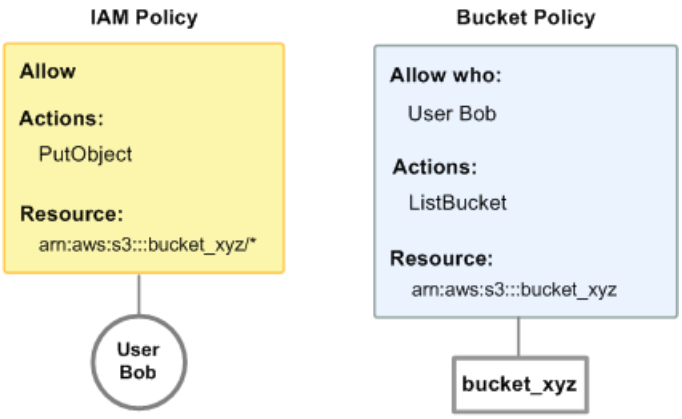
Note

先の例では、条件のない単純なポリシーを示しました。どちらのポリシーでも特定の条件を指定して、同じ結果を得ることができます。

IAM ポリシーでは、Amazon S3 リソースへのアクセスをユーザーごとに管理できますが、バケットポリシーでは、特定のリソースごとに管理できます。次の例は、2 つのポリシーシステムがどのようにして共に動作するかをさらに詳しく示しています。

Example 1

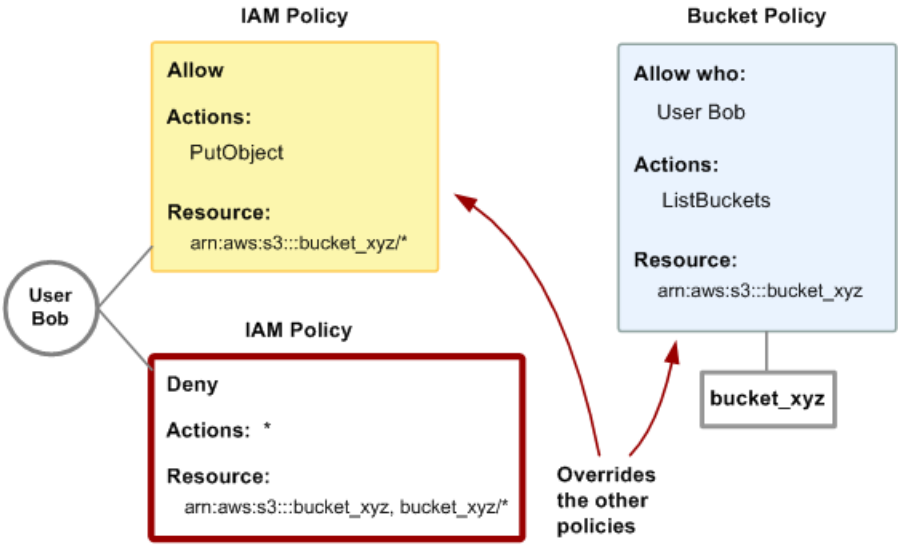
この例では、Bob に IAM ポリシーとバケットポリシーの両方が適用されています。IAM ポリシーは bucket_xyz で `PutObject` を使用するアクセス許可を付与し、バケットポリシーは同じバケットで `ListBucket` を使用するアクセス許可を付与しています。以下の図に、そのコンセプトを示します。



Bob が bucket_xyz にオブジェクトを入れるリクエストを送信すると、そのアクションは IAM ポリシーで許可されます。Bob が bucket_xyz 内のオブジェクトをリストするリクエストを送信した場合は、バケットポリシーがそのアクションを許可します。

Example 2

この例は、前の例で示した、Bob に 2 つのポリシーが適用されている状態に基づいています。Bob が bucket_xyz に対するアクセス許可を乱用したため、このバケットへの Bob のアクセス許可をすべて削除したいとします。最も簡単な方法は、そのバケットに対する Bob のアクションをすべて拒否するようなポリシーを追加することです。明示的な拒否は常に許可に優先するため、この 3 つ目のポリシーが他の 2 つより優先されます (ポリシー評価のロジックについては、「評価論理 (p. 491)」を参照してください)。次の図は、その概念を示しています。



また、そのバケットに対する Bob からのアクセスをすべて拒否するようなステートメントをバケットポリシーに追加することもできます。これは、そのバケットに対する Bob のアクセスを拒否する IAM ポリシーを追加するのと同じ効果を持ちます。

Amazon S3 のアクションとリソースを対象とするポリシーの例については、「Amazon S3 のポリシーの例 (p. 307)」を参照してください。S3 ポリシーの作成の詳細については、『Amazon Simple Storage Service 開発者ガイド』を参照してください。

リソース作成者のアクセス許可

Amazon S3 は、バケットまたはオブジェクトを作成した AWS アカウントに対し、デフォルトでそのリソースに対する完全なアクセス許可を付与します。しかし、AWS アカウントでなくユーザーがバケットまたはオブジェクトを作成した場合、デフォルトでは、そのユーザーにはそのリソースで他のアクションを実行するアクセス許可は付与されません。アクセス許可を付与するには、IAM ポリシーを使用して追加のアクセス許可を付与する必要があります。

Amazon S3 ARN

Amazon S3 では、ポリシー内で指定できるリソースはバケットとオブジェクトです。Amazon リソースネーム (ARN) は次の形式にしたがいます。

```
arn:aws:s3:::bucket_name/key_name
```

`arn:aws:s3:::bucket_name` はバケットのみを参照し、`arn:aws:s3:::bucket_name/key_name` 文字列全体はオブジェクトを参照します。

次に例を示します。

```
arn:aws:s3:::example_bucket/developers/design_info.doc
```

バケット名はグローバルであるため、ARN のリージョンおよび AWS アカウント ID 部分は空にする必要があります。

Amazon S3 ARN でポリシー変数を使用することもできます。例えば、ご使用の Amazon S3 バケットで IAM ユーザーごとに 1 つのフォルダーがあるとします。ポリシー変数を使用することで、1 つの ARN を使用してすべてのユーザーにプライベートフォルダを付与できます。

```
arn:aws:s3:::bucket_name/developers/${aws:username}/
```

ポリシーが評価されると、変数 `${aws:username}` にはリクエストを行うプリンシパルのユーザー名が代入されます。ポリシー変数の詳細については、『*AWS Identity and Access Management Using IAM*』ガイドの「IAM Policy Variables Overview」を参照してください。

Amazon S3 アクション

ポリシー内で指定できる Amazon S3 アクションは、リソースのタイプによってグループに分けられています。

オブジェクトに関連するアクション

- `s3:GetObject` (REST GET Object、REST HEAD Object、REST GET Object torrent、SOAP `GetObject`、および SOAP `GetObjectExtended` を対象とします)

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

- `s3:GetObjectVersion` (REST GET Object、REST HEAD Object、REST GET Object torrent、SOAP `GetObject`、および SOAP `GetObjectExtended` を対象とします)
- `s3:PutObject` (REST PUT Object、REST POST Object、REST Initiate Multipart Upload、REST Upload Part、REST Complete Multipart Upload、SOAP `PutObject`、および SOAP `PutObjectInline` を対象とします)
- `s3:GetObjectAcl`
- `s3:GetObjectVersionAcl`
- `s3:PutObjectAcl`
- `s3:PutObjectVersionAcl`
- `s3:DeleteObject`
- `s3:DeleteObjectVersion`
- `s3:ListMultipartUploadParts`
- `s3:AbortMultipartUpload`
- `s3:GetObjectTorrent`
- `s3:GetObjectVersionTorrent`
- `s3:RestoreObject`

バケットに関連するアクション

- `s3:CreateBucket`
- `s3:DeleteBucket`
- `s3:ListBucket`
- `s3:ListBucketVersions`
- `s3:ListAllMyBuckets` (REST GET Service と SOAP `ListAllMyBuckets` を対象とします)
- `s3:ListBucketMultipartUploads`

バケットのサブリソースに関連するアクション

- `s3:GetBucketAcl`
- `s3:PutBucketAcl`
- `s3:GetBucketCORS`
- `s3:PutBucketCORS`
- `s3:GetBucketVersioning`
- `s3:PutBucketVersioning`
- `s3:GetBucketRequestPayment`
- `s3:PutBucketRequestPayment`
- `s3:GetBucketLocation`
- `s3:GetBucketPolicy`
- `s3:DeleteBucketPolicy`
- `s3:PutBucketPolicy`
- `s3:GetBucketNotification`
- `s3:PutBucketNotification`

- s3:GetBucketLogging
- s3:PutBucketLogging
- s3:GetBucketWebsite
- s3:PutBucketWebsite
- s3:DeleteBucketWebsite
- s3:GetLifecycleConfiguration
- s3:PutLifecycleConfiguration

オブジェクトを削除するには、明示的に DELETE Object API を呼び出すか、Amazon S3 で自動的に削除できるようにライフサイクルを設定します (「オブジェクトの有効期限 (p. 123)」を参照)。バケットのオブジェクトをユーザーまたはアカウントが削除できないようにするには、ユーザーやアカウントによる s3:DeleteObject、s3:DeleteObjectVersion、および s3:PutLifecycleConfiguration アクションを拒否する必要があります。

Amazon S3 ポリシーキー

ポリシーキーとポリシー条件を使用すると、リクエストされている API アクションだけでなく、それ以外の情報に基づいて、リソースへのアクセスを制限することが可能です。リクエスタの IP アドレス、リクエストの日時といった、リクエストに関するコンテキスト情報に基づいてアクセスを制限できます。条件はキーと値のペアで作成されます。条件はキーに指定します。条件およびよく使用される AWS 条件の詳細については、『AWS Identity and Access Management Using IAM』ガイドの「*Condition*」を参照してください。

Amazon S3 には、よく使用される AWS 条件のほかにも、一連の独自の条件が用意されています。例えば、リクエストにある、一般的な HTTP ヘッダーや Amazon S3 固有のヘッダーの値に基づいてアクセスを制限することができます。

すべての条件に、任意のキーと値のペアを入れるプレースホルダを指定できます。ポリシーが評価されると、プレースホルダはリクエストからの情報に置換されます。例えば、特定のパスに対するアクセス許可を指定するために、リソース名や条件キーの一部として ${aws:username} などの変数を指定できます。詳細については、『*AWS Identity and Access Management Using IAM*』ガイドの Policy Variables に関するセクションを参照してください。

このセクションでは、AWS アカウントの Amazon S3 リソースへのアクセスの制限に使用できるポリシーキーをリストします。

利用可能なキー

AWS では、アクセスポリシー言語を使用するすべての AWS 製品でサポートされる、一般的な一連のキーを用意しています。これらのキーのリストについては、『*AWS Identity and Access Management Using IAM*』ガイドの利用可能なキーに関するセクションを参照してください。

Amazon S3 にはアクション固有のポリシーキーもあります。これらは、リソースの種類と、以下の表に示す適用可能なアクションごとにグループに分けられています。ポリシーキーの中には、複数の種類のリソースやアクションに適用できるものもあります。

Important

IAM は、Amazon S3 内でポリシーの有効性を評価することはできません。無効なキーとアクションの組み合わせを指定しても、ポリシーを IAM にアップロードしたときに IAM が例外を返すことはありません。また、Amazon S3 からもエラーメッセージは返されません。Amazon S3 が判断できるのは、ポリシーが条件を満たさないために適用不可能である場合のみです。しかし、ポリシー条件の指定方法が適切でない場合 (数値比較に文字列フィールドを使用するなど) は、Amazon S3 はリクエストに例外を返し、アクセスが拒否されます。

特に明記されていない限り、各キーは、アクセスポリシー言語の文字列条件で使用するために提供されています。詳細については、『AWS Identity and Access Management Using IAM』ガイドの「*Condition*」を参照してください。

Amazon S3 ポリシー内のオブジェクトキー

次のリストは、Amazon S3 ポリシーに含めることができるオブジェクトに関連するキーを示しています。

Action	Applicable Keys	Description
`s3:PutObject`	`s3:x-amz-acl`	The Amazon S3 canned ACL that is applied to the object. A canned ACL represents a predefined permission that can be applied to the object being PUT to Amazon S3. Valid values: `private` | `public-read` | `public-read-write` | `authenticated-read` | `bucket-owner-read` | `bucket-owner-full-control` | `log-delivery-write`. Example value: `public-read`
	`s3:x-amz-copy-source`	The header that specifies the name of the source bucket and key name of the source object, separated by a slash (/). Used when copying an object. Example value: `/bucketname/keyname`
	`s3:x-amz-grant-read`, `s3:x-amz-grant-write`, `s3:x-amz-grant-read-acp`, `s3:x-amz-grant-write-acp`, `s3:x-amz-grant-full-control`	These conditions relate to the ACL-specific request headers the action supports. You can add a condition in your bucket policy to deny or grant the action based on the presence of these headers with specific values. For more information about the API and the request headers, see PUT Object. For an example policy that uses these condition keys, see バケット所有者はフルコントロール権限を持ちながら、オブジェクトをアップロードするクロスアカウントアクセス許可を付与する (p. 337).
	`s3:x-amz-server-side-encryption`	Allow the specific action only if `x-amz-server-side-encryption` header is present in the request and its value matches the specified condition. Valid values: AES256 Example value: `AES256`
	`s3:x-amz-metadata-directive`	

Action	Applicable Keys	Description
		The header that specifies whether the metadata is copied from the source object or replaced with metadata provided in the request. If copied, the metadata, except for the version ID, remains unchanged. Otherwise, all original metadata is replaced by the metadata you specify. Used when copying an object. Valid values: `COPY` | `REPLACE`. The default is `COPY`. Example value: `REPLACE`
`s3:PutObjectAcl`	`s3:x-amz-acl`	The Amazon S3 canned ACL that is applied to the object. A canned ACL represents a predefined permission that can be applied to the object being PUT to Amazon S3. Valid values: `private` | `public-read` | `public-read-write` | `authenticated-read` | `bucket-owner-read` | `bucket-owner-full-control` | `log-delivery-write`. Example value: `public-read`
	`s3:x-amz-grant-read`, `s3:x-amz-grant-write`, `s3:x-amz-grant-read-acp`, `s3:x-amz-grant-write-acp`, `s3:x-amz-grant-full-control`	These conditions relate to the ACL-specific request headers the action supports. You can add a condition in your bucket policy to deny or grant the action based on the presence of these headers with specific values. For more information about the API and the request headers, see PUT Object acl. For an example policy that uses these condition keys, see バケット所有者はフルコントロール権限を持ちながら、オブジェクトをアップロードするクロスアカウントアクセス許可を付与する (p. 337).
`s3:GetObjectVersion`	`s3:VersionId`	The version ID of the object being retrieved. Example value: `Upfdndhfd8438MNFDN93jdnJFkdmqnh893`

Action	Applicable Keys	Description
`s3:GetObjectVersionAcl`	`s3:VersionId`	The version ID of the object ACL being retrieved. Example value: `Upfdndhfd8438MNFDN93jdnJFkdmqnh893`
`s3:PutObjectVersionAcl`	`s3:VersionId`	The version ID of the object ACL being PUT. Example value: `Upfdndhfd8438MNFDN93jdnJFkdmqnh893`
	`s3:x-amz-acl`	The Amazon S3 canned ACL that is applied to the object. A canned ACL represents a predefined permission that can be applied to the object being PUT to Amazon S3. Valid values: `private` | `public-read` | `public-read-write` | `authenticated-read` | `bucket-owner-read` | `bucket-owner-full-control` | `log-delivery-write`. Example value: `public-read`
	`s3:x-amz-grant-read`, `s3:x-amz-grant-write`, `s3:x-amz-grant-read-acp`, `s3:x-amz-grant-write-acp`, `s3:x-amz-grant-full-control`	These conditions relate to the ACL-specific request headers the action supports. You can add a condition in your bucket policy to deny or grant the action based on the presence of these headers with specific values. For more information about the API and the request headers, see PUT Object acl. For an example policy that uses these condition keys, see バケット所有者はフルコントロール権限を持ちながら、オブジェクトをアップロードするクロスアカウントアクセス許可を付与する (p. 337).
`s3:DeleteObjectVersion`	`s3:VersionId`	The version ID of the object being deleted. Example value: `Upfdndhfd8438MNFDN93jdnJFkdmqnh893`

Amazon S3 ポリシー内のバケットキー

次の表は、Amazon S3 ポリシーに含めることができるバケットに関連するキーを示しています。

Action	Applicable Keys	Description
`s3:CreateBucket`	`s3:x-amz-acl`	The Amazon S3 canned ACL that is applied to the bucket. A canned ACL represents a predefined permission that can be applied to the bucket being created. Valid values: `private` | `public-read` | `public-read-write` | `authenticated-read` | `bucket-owner-read` | `bucket-owner-full-control` | `log-delivery-write`. Example value: `public-read`
	`s3:LocationConstraint`	Specifies the region where the bucket will be created. Valid values are `us-west-1` (for Northern California) or `EU` (for Ireland). Do not specify a value for US Standard. Example value: `us-west-1`
	`s3:x-amz-grant-read`, `s3:x-amz-grant-write`, `s3:x-amz-grant-read-acp`, `s3:x-amz-grant-write-acp`, `s3:x-amz-grant-full-control`	These conditions relate to the ACL-specific request headers the action supports. You can add a condition in your bucket policy to deny or grant the action based on the presence of these headers with specific values. For more information about the API and the request headers, see PUT Bucket. For an example policy that uses these condition keys, see バケット所有者はフルコントロール権限を持ちながら、オブジェクトをアップロードするクロスアカウントアクセス許可を付与する (p. 337).

Action	Applicable Keys	Description
s3:ListBucket	s3:prefix	Limits the response to objects that begin with the specified prefix. Use this to allow or deny access to objects that begin with the prefix. Example value: home
	s3:delimiter	The character you use to group objects. Example value: /
	s3:max-keys	The number of objects to return from the call. The maximum allowed value (and default) is 1000. For use with access policy language numeric conditions. For more information, see 数値の条件 (p. 501). Example value: 100
s3:ListBucketVersions	s3:prefix	Header that lets you limit the response to include only keys that begin with the specified prefix. Example value: home
	s3:delimiter	The character you use to group objects. Example value: /
	s3:max-keys	The number of objects to return from the call. The maximum allowed value (and default) is 1000. For use with access policy language numeric conditions. For more information, see 数値の条件 (p. 501). Example value: 100

Action	Applicable Keys	Description
s3:PutBucketAcl	s3:x-amz-acl	The Amazon S3 canned ACL that is applied to the bucket. A canned ACL represents a predefined permission that can be applied to the bucket being created. Valid values: private | public-read | public-read-write | authenticated-read | bucket-owner-read | bucket-owner-full-control | log-delivery-write. Example value: public-read
	s3:x-amz-grant-read, s3:x-amz-grant-write, s3:x-amz-grant-read-acp, s3:x-amz-grant-write-acp, s3:x-amz-grant-full-control	These conditions relate to the ACL-specific request headers the action supports. You can add a condition in your bucket policy to deny or grant the action based on the presence of these headers with specific values. For more information about the API and the request headers, see PUT Bucket acl. For an example policy that uses these condition keys, see バケット所有者はフルコントロール権限を持ちながら、オブジェクトをアップロードするクロスアカウントアクセス許可を付与する (p. 337).

Amazon S3 のポリシーの例

このセクションでは、Amazon S3 へのユーザーアクセスを制御するための簡単な IAM ポリシーをいくつか紹介します。

Note

以下のポリシー例は、プログラミング上のテストでは動作しますが、 これらを Amazon S3 コンソールで使用するには、コンソールで必要とされる追加のアクセス許可を付与する必要があります。これらをはじめとするポリシーを Amazon S3 コンソールで使用するための詳細については、「例: IAM ポリシーを使用したバケットへのアクセスの制御 (p. 313)」を参照してください。

例: バケットの 1 つへのアクセスを IAM ユーザーに許可する

次の例では、 自分の AWS アカウント内の IAM ユーザーにバケットの 1 つ、examplebucket へのアクセス権を付与して、ユーザーがオブジェクトを追加、更新、および削除できるようにします。

このポリシーでは、ユーザーにs3:PutObject、s3:GetObject、s3:DeleteObject アクションへのアクセス許可を付与するだけでなく、s3:ListAllMyBuckets、s3:GetBucketLocation、および s3:ListBucket アクションへのアクセス許可も付与します。これらが、コンソールで必要とされる追

加のアクセス許可です。コンソールの動作の詳細については、「例: IAM ポリシーを使用したバケットへのアクセスの制御 (p. 313)」を参照してください。

```
{
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "s3:ListAllMyBuckets"
         ],
         "Resource":"arn:aws:s3:::*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "s3:ListBucket",
            "s3:GetBucketLocation"
         ],
         "Resource":"arn:aws:s3:::examplebucket"
      },
      {
         "Effect":"Allow",
         "Action":[
            "s3:PutObject",
            "s3:GetObject",
            "s3:DeleteObject"
         ],
         "Resource":"arn:aws:s3:::examplebucket/*"
      }
   ]
}
```

例: バケット内のフォルダーへのアクセスをすべての IAM ユーザーに許可する

次の例では、IAM ユーザー、Alice と Bob にバケット examplebucket にアクセスできるようにして、この 2 人がオブジェクトを追加、更新、および削除できるようにします。ただし、バケット内の単一のフォルダに全ユーザーのアクセスを制限したいとします。フォルダを作成する際、ユーザー名と同じフォルダ名にすることもできます。

```
examplebucket
   Alice/
   Bob/
```

各ユーザーに本人のフォルダのみへのアクセス権を付与するには、各ユーザー用のポリシーを作成し、個別にアタッチします。例えば、ユーザー、アリスに次のポリシーをアタッチして、examplebucket/Alice フォルダへの特定の Amazon S3 アクションを許可することができます。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "s3:PutObject",
```

```
            "s3:GetObject",
            "s3:GetObjectVersion",
            "s3:DeleteObject",
            "s3:DeleteObjectVersion"
         ],
         "Resource":"arn:aws:s3:::examplebucket/Alice/*"
      }
   ]
}
```

次に、ユーザー Bob にも同様のポリシーをアタッチし、Resource の値にフォルダ 名として Bob を指定します。

各ユーザーにポリシーをアタッチする代わりに、ポリシー変数を使用する単一のポリシーを作成し、グループにアタッチすることもできます。まずグループを作成し、それに Alice と Bob を追加する必要があります。次のポリシー例では、examplebucket/${aws:username} フォルダに対して一連の Amazon S3 アクションを許可しています。ポリシーが評価されると、ポリシー変数 ${aws:username} はリクエスタのユーザー名で置き換えられます。たとえば、Alice がオブジェクトの PUT リクエストを送信した場合は、Alice が examplebucket/Alice フォルダにオブジェクトをアップロードする PUT アクションのみが許可されます。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "s3:PutObject",
            "s3:GetObject",
            "s3:GetObjectVersion",
            "s3:DeleteObject",
            "s3:DeleteObjectVersion"
         ],
         "Resource":"arn:aws:s3:::examplebucket/${aws:username}/*"
      }
   ]
}
```

Note

ポリシー変数を使用する場合は、バージョン 2012-10-17 をポリシー内で明示的に指定する必要があります。アクセスポリシー言語のデフォルトバージョンは 2008-10-17 です。このバージョンでは、ポリシー変数をサポートしていません。

Amazon S3 コンソールで前述のポリシーをテストする場合、次のポリシーに示すように、追加の Amazon S3 アクションに対するアクセス許可が要求されます。これらのアクセス許可をコンソールで使用する方法については、「例: IAM ポリシーを使用したバケットへのアクセスの制御 (p. 313)」を参照してください。

```
{
 "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowGroupToSeeBucketListInTheConsole",
      "Action": [ "s3:ListAllMyBuckets", "s3:GetBucketLocation" ],
```

```
      "Effect": "Allow",
      "Resource": [ "arn:aws:s3:::*"  ]
    },
    {
      "Sid": "AllowRootLevelListingOfTheBucket",
      "Action": ["s3:ListBucket"],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::examplebucket"],
      "Condition":{
            "StringEquals":{
                    "s3:prefix":[""], "s3:delimiter":["/"]
                           }
                }
    },
    {
      "Sid": "AllowListBucketOfASpecificUserPrefix",
      "Action": ["s3:ListBucket"],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::examplebucket"],
      "Condition":{  "StringLike":{"s3:prefix":["${aws:username}/*"] }
       }
    },
      {
     "Sid": "AllowUserSpecificActionsOnlyInTheSpecificUserPrefix",
         "Effect":"Allow",
         "Action":[
            "s3:PutObject",
            "s3:GetObject",
            "s3:GetObjectVersion",
            "s3:DeleteObject",
            "s3:DeleteObjectVersion"
         ],
         "Resource":"arn:aws:s3:::examplebucket/${aws:username}/*"
      }
  ]
}
```

Note

2012-10-17 バージョンのポリシーでは、ポリシー変数の先頭には $ が付きます。使用するオブジェクトキーに $ が含まれている場合、この構文の変化により衝突が発生します。例えば、ポリシーにオブジェクトキー my$file を含めるには、my${$}file のように ${$} を使用して $ を指定します。

IAM ユーザー名は人間が読んで理解できるわかりやすい識別子ですが、グローバルで一意である必要はありません。例えば、Bob が退職して別の Bob が入社した場合、この別の Bob が前の Bob の情報にアクセスできます。フォルダを作成する際に、ユーザー名の代わりにユーザー ID を使用することもできます。ユーザー ID はそれぞれ一意であるためです。この場合、${aws:userid} ポリシー変数を使用するように前述のポリシーを修正する必要があります。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
```

```
            "s3:PutObject",
            "s3:GetObject",
            "s3:GetObjectVersion",
            "s3:DeleteObject",
            "s3:DeleteObjectVersion"
         ],
         "Resource":"arn:aws:s3:::my_corporate_bucket/home/${aws:userid}/*"
      }
   ]
}
```

非 IAM ユーザー (モバイルアプリユーザー) に対するバケット内のフォルダへのアクセス許可

ユーザーのデータを S3 バケットに格納するモバイルゲームアプリを開発するとします。バケットに各アプリユーザーのフォルダを作成します。また、各ユーザーには各自のフォルダのみにアクセスを制限します。しかし、ユーザーがアプリをダウンロードしてゲームをプレイし始める前にフォルダを作成することはできません。お客様にはユーザーのユーザー ID がないためです。

この場合、ユーザーには、Login with Amazon、 Facebook、または Google などのパブリックアイデンティプロバイダを使用してアプリにサインインするよう要求できます。ユーザーがこれらのプロバイダの 1 つを使用してアプリにサインインすると、ユーザー ID が設定されるため、お客様はこれを使用して実行時にユーザー固有のフォルダを作成することができます。

これにより、AWS Security Token Service のウェブ認証フェデレーションを使用して、アイデンティプロバイダからの情報をアプリに組み入れ、各ユーザーの一時的なセキュリティ認証情報を取得することができます。続いて、IAM ポリシーを作成して、アプリがバケットにアクセスできるようにしたり、ユーザー固有のフォルダの作成、データのアップロードなどのオペレーションを実行できるようにすることができます。ウェブ認証フェデレーションの詳細については、『*一時的なセキュリティ認証情報の使用*』*ガイド*の「Creating Temporary Security Credentials for Mobile Apps Using Identity Providers」を参照してください。

例: 特定のグループに対し、Amazon S3 で共有フォルダを持つことを許可する

次のポリシーをグループにアタッチすることで、グループ内の全メンバーに Amazon S3 のフォルダ `my_corporate_bucket/share/marketing` へのアクセス権が付与されます。 グループメンバーは、指定されたフォルダのオブジェクトに対してだけ、ポリシーに示された特定の Amazon S3 アクションのみへのアクセスを許可されます。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "s3:PutObject",
            "s3:GetObject",
            "s3:GetObjectVersion",
            "s3:DeleteObject",
            "s3:DeleteObjectVersion"
         ],
         "Resource":"arn:aws:s3:::my_corporate_bucket/share/marketing/*"
      }
```

```
   ]
}
```

例: すべてのユーザーに対し、企業バケットの特定部分のオブジェクトの読み取りを許可する

次の例では、AWS アカウントが所有者であるすべての IAM ユーザーが含まれる、`AllUsers` というグループを作成します。その後、`my_corporate_bucket/readonly` フォルダ内のオブジェクトに対してのみ `GetObject` と `GetObjectVersion` アクセス権をグループに付与するポリシーをアタッチします。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "s3:GetObject",
            "s3:GetObjectVersion"
         ],
         "Resource":"arn:aws:s3:::my_corporate_bucket/readonly/*"
      }
   ]
}
```

例: パートナーに対し、企業バケットの特定部分へのファイルのドロップを許可する

次の例では、パートナー会社を表す `WidgetCo` というグループを作成します。アクセス権を必要としているパートナー会社の特定の個人またはアプリケーションに対して IAM ユーザーを作成し、作成したユーザーをグループに入れます。

その後、企業バケット `my_corporate_bucket/uploads/widgetco` 内のフォルダへの `PutObject` アクセス権をグループに付与するポリシーをアタッチします。

`WidgetCo` グループがそのバケットに対して他の操作ができないようにするため、AWS アカウント内でのすべての Amazon S3 リソースに対する `PutObject` 以外のあらゆる Amazon S3 アクションへのアクセス権を明示的に拒否するステートメントを追加します。このステップは、AWS アカウント内のどこかで、ユーザーに Amazon S3 リソースへの幅広いアクセス権を付与するポリシーが使われている場合にのみ必要です。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":"s3:PutObject",
         "Resource":"arn:aws:s3:::my_corporate_bucket/uploads/widgetco/*"
      },
      {
         "Effect":"Deny",
         "NotAction":"s3:PutObject",
         "Resource":"arn:aws:s3:::my_corporate_bucket/uploads/widgetco/*"
```

```
      },
      {
         "Effect":"Deny",
         "Action":"s3:*",
         "NotResource":"arn:aws:s3:::my_corporate_bucket/uploads/widgetco/*"
      }
   ]
}
```

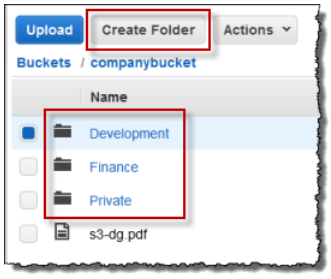
例: IAM ポリシーを使用したバケットへのアクセスの制御

Topics

このチュートリアルでは、Amazon S3 でのユーザーアクセス許可について説明します。フォルダを含むバケットを作成した後、AWS アカウントに AWS Identity and Access Management ユーザーを作成し、作成したユーザーに、Amazon S3 バケットおよびバケット内のフォルダに対する段階的なアクセス許可を付与します。

背景: バケットとフォルダの基本

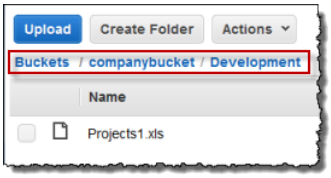
Amazon S3 のデータモデルはフラットな構造です。バケットを作成し、そのバケットにオブジェクトが格納されます。サブバケットやサブフォルダの階層はありませんが、フォルダ階層をエミュレートすることができます。Amazon S3 コンソールなどのツールで、バケット内の論理フォルダおよび論理サブフォルダを次のように表示できます。



コンソールには、[companybucket] というバケットがあり、その中に 3 つのフォルダ ([Private]、[Development]、および [Finance]) と [s3-dg.pdf] というオブジェクトが表示されています。コンソールではオブジェクト名 (キー) を使用して、フォルダおよびサブフォルダからなる論理的な階層を作成します。次の例を考えます。

- [Development] フォルダを作成すると、コンソールは Development/ というキーを持つオブジェクトを作成します。後ろに区切り記号「/」が付いていることに注意してください。
- [project1.pdf] というオブジェクトを [Development] フォルダにアップロードすると、コンソールはオブジェクトをアップロードし、[Development/project1.pdf] というキーを設定します。

  このキーの Development はプレフィックスで、'/' は区切り記号です。Amazon S3 API のオペレーションではプレフィックスと区切り記号がサポートされます。例えば、特定のプレフィックスと区切り記号を持つすべてのオブジェクトのリストを取得できます。コンソールで [Development] フォルダをダブルクリックすると、そのフォルダ内のオブジェクトのリストが表示されます。次の例では、1 つのオブジェクトが [Development] フォルダに含まれています。



  コンソールに companybucket バケットの Development フォルダを表示すると、Amazon S3 にリクエストが送信されます。そのリクエストに、プレフィックス Development と区切り記号 '/' が指定されています。コンソールのレスポンスは、コンピュータのファイルシステムのフォルダリストと似ています。前述の例は、バケット companybucket に Development/project1.pdf というキーを持つ 1 つのオブジェクトがあることを示しています。

コンソールはオブジェクトキーを使用して論理的な階層を表します。Amazon S3 に物理的な階層はなく、フラットなファイル構造にオブジェクトを含むバケットが存在するだけです。Amazon S3 API を使用してオブジェクトを作成するときに、論理的な階層を示すオブジェクトキーを使用できます。

オブジェクトの論理階層を作成するときに、個別のフォルダへのアクセスを管理できます。このチュートリアルでは、その方法について説明します。

チュートリアルに入る前に、理解しておくべき概念がもう 1 つあります。「ルートレベル」のバケットの内容です。companybucket バケットに次のオブジェクトがあるとします。

Private/privDoc1.txt

Private/privDoc2.zip

Development/project1.xls

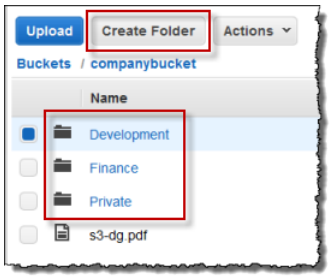
Development/project2.xls

Finance/Tax2011/document1.pdf

Finance/Tax2011/document2.pdf

s3-dg.pdf

これらのオブジェクトキーにより、ルートレベルフォルダ Private、Development、および Finance と、ルートレベルオブジェクト s3-dg.pdf を持つ論理階層が作成されます。Amazon S3 コンソールでバケット名をクリックすると、次のようにルートレベルの項目が表示されます。コンソールにはルートレベルフォルダとしてトップレベルのプレフィックス (Private/、Development/ および Finance/) が表示されます。オブジェクトキー s3-dg.pdf にはプレフィックスがないため、ルートレベルアイテムとして表示されます。



チュートリアルの例

このチュートリアルの例は次のとおりです。

- バケットを 1 つ作成し、3 つのフォルダ (Private、Development、および Finance) を追加します。
- ユーザーは Alice と Bob の 2 名です。Alice には Development フォルダのみ、Bob には Finance フォルダのみへのアクセスを許可し、Private フォルダの内容は非公開にします。このチュートリアルでは、AWS Identity and Access Management (IAM) ユーザー (Alice と Bob という同じユーザー名を使用) を作成してアクセスを管理し、必要なアクセス許可を付与します。

  IAM では、ユーザーグループを作成し、そのグループのすべてのユーザーに適用するグループレベルのアクセス許可を付与することもできます。これにより、アクセス許可を効果的に管理できます。この演習では、Alice と Bob に共通のアクセス許可が必要になります。そのため、Consultants というグループを作成し、Alice と Bob をグループに追加します。最初に、グループポリシーをグループにアタッチしてアクセス許可を付与します。次に、特定のユーザーにポリシーをアタッチしてユーザー固有のアクセス許可を追加します。

Note

このチュートリアルでは、`companybucket` をバケット名として、Alice および Bob を IAM ユーザーとして、Consultants をグループ名として使用しています。Amazon S3 ではバケット名はグローバルに一意である必要があるため、実際に作成する名前に置き換えてください。

ステップ 0: チュートリアルに関する準備を行う

この例では、AWS アカウントの認証情報を使用して IAM ユーザーを作成します。当初、これらのユーザーにはアクセス許可はありません。これらのユーザーに、特定の Amazon S3 アクションを実行するためのアクセス許可を段階的に付与します。アクセス許可をテストするには、各ユーザーの認証情報を使用してコンソールにサインインします。AWS アカウントの所有者としてアクセス許可を段階的に付与する場合と、IAM ユーザーとしてアクセス許可をテストする場合とでは、異なる認証情報を使用してサインイン/サインアウトする必要があります。このテストは 1 つのブラウザで行うこともできますが、2 つの異なるブラウザを使用するとより効率的です。1 つのブラウザで AWS アカウントの認証情報を使用して AWS Management Console に接続し、もう 1 つのブラウザで IAM ユーザー認証情報を使用して接続します。

AWS アカウントの認証情報を使用して AWS Management Console にサインインするには、http://aws.amazon.com/console にアクセスします。 IAM ユーザーは、この同じリンクからはサインインできません。IAM ユーザーは IAM 対応のサインインページを使用する必要があります。アカウント所有者からユーザーにこのリンクを通知します。

IAM ユーザー用のサインインリンクを通知するには

1. AWS マネジメントコンソールにサインインし、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. [Navigation] ペインで、[IAM Dashboard] をクリックします。

3. [AWS Account Alias] で、[&IAM users sign in link] の URL を確認します。このリンクを IAM ユーザーに通知し、IAMユーザー名とパスワードを使用してコンソールにサインインしてもらいます。

IAM の詳細については、「*AWS Identity and Access Management Using IAM*」の「The AWS Management Console Sign-in Page」を参照してください。

ステップ 1: バケットを作成する

このステップでは、AWS アカウントの認証情報を使用して Amazon S3 コンソールにサインインし、バケットを作成します。作成したバケットにフォルダ (Development、Finance、Private) を追加し、各フォルダにサンプルドキュメントを 1 つか 2 つアップロードします。

1. AWS マネジメントコンソールにサインインして Amazon S3 コンソール (https://console.aws.amazon.com/s3/) を開きます。
2. バケットを作成します。
   詳細な手順については、「*Amazon Simple Storage Service Console User Guide*」の「Creating a Bucket」を参照してください。
3. バケットにドキュメントを 1 つアップロードします。
   この演習では、このバケットのルートレベルに `s3-dg.pdf` というドキュメントがあると想定しています。別のドキュメントをアップロードする場合は、`s3-dg.pdf` をそのドキュメントのファイル名に置き換えてください。
4. Private、Finance、および Development という 3 つのフォルダをバケットに追加します。

   詳細な手順については、「*Amazon Simple Storage Service Console User Guide*」の「Creating a Folder」を参照してください。
5. 各フォルダにドキュメントを 1 つか 2 つアップロードします。

   この演習では、各フォルダに数個のドキュメントをアップロードしてあり、バケットに次のキーを持つオブジェクトが存在すると想定しています。

   Private/privDoc1.txt

   Private/privDoc2.zip

   Development/project1.xls

   Development/project2.xls

   Finance/Tax2011/document1.pdf

   Finance/Tax2011/document2.pdf

   s3-dg.pdf

   詳細な手順については、『*Amazon Simple Storage Service Console User Guide*』の「Uploading Objects into Amazon S3」を参照してください。

ステップ 2: IAM ユーザーとグループを作成する

IAM コンソールを使用して 2 人の IAM ユーザー (Alice と Bob) を AWS アカウントに追加します。また、Consultants という管理グループを作成し、両ユーザーをこのグループに追加します。

Caution

ユーザーとグループを追加するときに、これらのユーザーにアクセス許可を付与するポリシーをアタッチしないでください。最初の時点では、これらのユーザーにアクセス許可は付与せず、以降のセクションで段階的にアクセス許可を付与します。最初に、これらの IAM ユーザーにパスワードを割り当てたことを確認します。これらのユーザー認証情報を使用して Amazon S3 アクションをテストし、アクセス許可が指定したとおりに機能することを確認します。

新しい IAM ユーザーを作成する詳細な手順については、「*IAM を使用する*」の「Adding a New User to Your AWS Account」を参照してください。

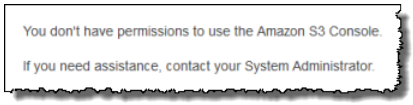
管理グループを作成する詳細な手順については、「*IAM を使用する*」の「Creating an Admins Group」を参照してください。

ステップ 3: IAM ユーザーにアクセス許可が付与されていないことを確認する

ブラウザを 2 つ使用する場合は、この時点で 2 つ目のブラウザからどちらかの IAM ユーザー認証情報を使用して、コンソールにサインインできます。

1. IAM ユーザーのサインインリンク (「IAM ユーザー用のサインインリンクを通知するには (p. 315)」を参照) から、どちらかの IAM ユーザー認証情報を使用して AWS コンソールにサインインします。
2. Amazon S3 コンソール (https://console.aws.amazon.com/s3/) を開きます。

   アクセス許可が付与されていないことを示す次のコンソールメッセージを確認します。



次に、ユーザーに段階的にアクセス許可を付与します。最初に、両方のユーザーに必要なアクセス許可を付与するグループポリシーをアタッチします。

ステップ 4: グループレベルのアクセス許可を付与する

すべてのユーザーが次のことを実行できるようにします。

- 親アカウントが所有するすべてのバケットを表示する

  これを行うには、Bob と Alice に `s3:ListAllMyBuckets` アクションのためのアクセス許可が必要です。
- `companybucket` バケット内のルートレベルのアイテム、フォルダ、およびオブジェクトのリストを表示する

  これを行うには、Bob と Alice に `companybucket` バケットに対する `s3:ListBucket` アクションのためのアクセス許可が必要です。

次に、これらのアクセス許可を付与するポリシーを作成し、グループにアタッチします。

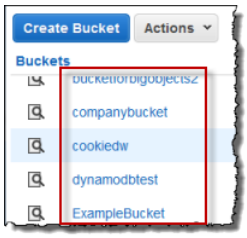
ステップ 4.1: すべてのバケットのリストを表示するアクセス許可を付与する

次のポリシーを使用して、親アカウントが所有するすべてのバケットを表示するために必要な最小限のアクセス許可をユーザーに付与します。

```
{
  "Statement": [
    {
      "Sid": "AllowGroupToSeeBucketListInTheConsole",
      "Action": ["s3:ListAllMyBuckets"],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::*"]
    }
  ]
}
```

ポリシーは JSON ドキュメントです。そのドキュメントの `Statement` はオブジェクトの配列であり、各オブジェクトが名前と値のペアを使用してアクセス許可を定義しています。前述のポリシーは、1 つの特定のアクセス許可を定義しています。`Action` はアクセスの種類を指定します。ポリシーの `s3:ListAllMyBuckets` は、定義済みの &S3 アクションです。このアクションは Amazon S3 GET サービスオペレーションを対象とし、認証された送信者が所有するすべてのバケットのリストを返します。`Effect` 要素の値は、特定のアクセスを許可するかどうかを決定します。

前述のポリシーをユーザーまたはグループにアタッチすると、そのユーザーまたはグループに、親 AWS アカウントが所有するバケットのリストを取得することを許可します。



IAM コンソールで、ポリシードキュメントを IAM ユーザーおよびグループにアタッチします。両方のユーザーがバケットのリストを取得できるようにするため、次の手順に従ってポリシーをグループにアタッチします。

Note

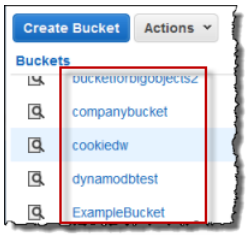
ユーザーのアクセス許可を付与するには、IAM ユーザーとしてではなく、AWS アカウントの認証情報を使用してサインインする必要があります。

1. AWS マネジメントコンソールにサインインし、IAM コンソール（https://console.aws.amazon.com/iam/ ）を開きます。
2. 作成したカスタムポリシーをグループにアタッチします。

   詳細な手順については、「IAM を使用する」の「*Managing &IAM Policies*」を参照してください。
3. アクセス許可をテストします。

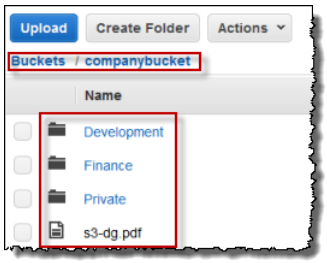
   1. IAM ユーザーのサインインリンク（「IAM ユーザー用のサインインリンクを通知するには (p. 315)」を参照）から、どちらかの IAM ユーザー認証情報を使用して AWS コンソールにサインインします。
   2. Amazon S3 コンソール（https://console.aws.amazon.com/s3/ ）を開きます。

      コンソールにすべてのバケットが表示されますが、バケット内のオブジェクトは表示されません。



ステップ 4.2: バケットのルートレベルの内容をユーザーが表示できるようにする

次に、すべてのユーザーに、ルートレベルの companybucket バケットアイテムを表示することを許可します。ユーザーは、Amazon S3 コンソールで会社のバケットをクリックすると、そのバケット内のルートレベルの項目を表示できるようになります。



ここでは、説明のために companybucket を使用しています。実際には、この演習で作成したバケットの名前を使用してください。

バケット名をクリックしたときにコンソールから Amazon S3 にどのようなリクエストが送信され、Amazon S3 からどのようなレスポンスが返されるか、さらに、レスポンスがコンソールでどのように解釈されるかを理解するには、少し詳細な知識が必要です。

バケット名をクリックすると、コンソールから Amazon S3 に GET Bucket (List Objects) リクエストが送信されます。このリクエストには次のパラメータが含まれます。

- 空の文字列を値に持つ prefix パラメータ。
- / を値に持つ delimiter パラメータ。

リクエストの例を次に示します。

```
GET ?prefix=&delimiter=/ HTTP/1.1
Host: companybucket.s3.amazonaws.com
Date: Wed, 01 Aug  2012 12:00:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

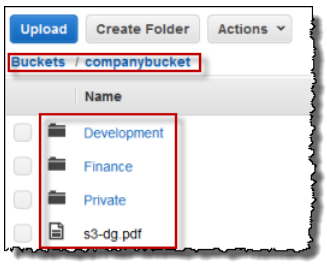
Amazon S3 から返される応答には次の <ListBucketResult/> 要素が含まれます。

```
<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Name>companybucket</Name>
  <Prefix></Prefix>
  <Delimiter>/</Delimiter>
```

```
   ...
  <Contents>
    <Key>s3-dg.pdf</Key>
    ...
  </Contents>
  <CommonPrefixes>
    <Prefix>Development/</Prefix>
  </CommonPrefixes>
  <CommonPrefixes>
    <Prefix>Finance/</Prefix>
  </CommonPrefixes>
  <CommonPrefixes>
    <Prefix>Private/</Prefix>
  </CommonPrefixes>
</ListBucketResult>
```

キー s3-dg.pdf には区切り記号 '/' が含まれておらず、Amazon S3 から <Contents/> 要素にキーが返されます。一方、このバケット例のその他すべてのキーには区切り記号 '/' が含まれています。Amazon S3 は、キーをグループ化し、Development/、Finance/、および /Private のそれぞれのプレフィックス値の <CommonPrefixes/> 要素を返します。この要素は、これらのキーの先頭から、指定した '/' 区切り記号の最初の出現箇所までのサブ文字列です。

コンソールでこの結果が解釈され、ルートレベルのアイテムが 3 つのフォルダと 1 つのオブジェクトキーとして表示されます。



次に、Bob か Alice が [Development] フォルダをダブルクリックすると、コンソールから Amazon S3 に GET Bucket (List Objects) リクエストが送信されます。このリクエストの prefix および delimiter パラメータは次の値に設定されています。

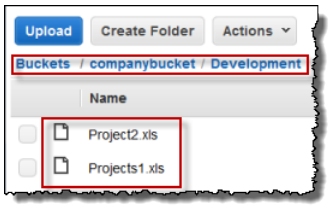
- prefix パラメータ: 値 Development/。
- delimiter パラメータ: '/' 値。

レスポンスとして、指定したプレフィックスで始まるオブジェクトキーが Amazon S3 から返されます。

```
<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Name>companybucket</Name>
  <Prefix>Development</Prefix>
  <Delimiter>/</Delimiter>
   ...
  <Contents>
    <Key>Project1.xls</Key>
    ...
  </Contents>
  <Contents>
```

```
    <Key>Project2.xls</Key>
    ...
  </Contents>
</ListBucketResult>
```

コンソールにオブジェクトキーが表示されます。



ここで、ルートレベルのバケットアイテムを表示するためのアクセス許可をユーザーに付与する作業に戻ります。バケットの内容を表示するには、ユーザーは `s3:ListBucket` アクションを呼び出すためのアクセス許可を持っている必要があります。次のポリシーステートメントを参照してください。ルートレベルの内容のみを表示できるようにするため、条件を追加して、ユーザーがリクエストに空の `prefix` を指定しなければならないようにします。つまり、ユーザーはどのルートレベルフォルダもダブルクリックすることはできません。最後に、ユーザーリクエストに値が `'/'` の `delimiter` パラメータを組み込むようユーザーに要求することにより、フォルダ形式のアクセスを要求する条件を追加します。

```
{
  "Sid": "AllowRootLevelListingOfCompanyBucket",
  "Action": ["s3:ListBucket"],
  "Effect": "Allow",
  "Resource": ["arn:aws:s3:::companybucket"],
  "Condition":{
         "StringEquals":{
             "s3:prefix":[""], "s3:delimiter":["/"]
                       }
          }
}
```

Amazon S3 コンソールを使用する場合、バケットをクリックすると、コンソールは最初に GET Bucket location リクエストを送信して、バケットがデプロイされている AWS リージョンを確認します。その後、バケットのリージョン固有エンドポイントを使用して GET Bucket (List Objects) リクエストを送信します。結果として、ユーザーがコンソールを使用する場合は、次のポリシーステートメントに示すように、`s3:GetBucketLocation` アクションのためのアクセス許可を付与する必要があります。

```
{
   "Sid": "RequiredByS3Console",
   "Action": ["s3:GetBucketLocation"],
   "Effect": "Allow",
   "Resource": ["arn:aws:s3:::*"]
}
```

ユーザーがルートレベルのバケットの内容を表示できるようにするには

1. AWS マネジメントコンソールにサインインして Amazon S3 コンソール (https://console.aws.amazon.com/s3/) を開きます。

IAM ユーザーの認証情報ではなく、AWS アカウントの認証情報を使用してコンソールにサインインします。

2. 既存のグループポリシーを以下のカスタムポリシーに置き換えます。このカスタムポリシーでは `s3:ListBucket` アクションも許可されます。

   詳細な手順については、「Managing IAM Policies」を参照してください。

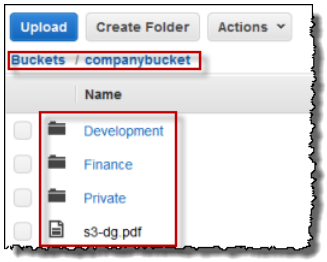
```
{
  "Statement": [
    {
     "Sid": "AllowGroupToSeeBucketListAndAlsoAllowGetBucketLocationRequired
ForListBucket",
      "Action": [ "s3:ListAllMyBuckets", "s3:GetBucketLocation" ],
      "Effect": "Allow",
      "Resource": [ "arn:aws:s3:::*"  ]
    },
    {
      "Sid": "AllowRootLevelListingOfCompanyBucket",
      "Action": ["s3:ListBucket"],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::companybucket"],
      "Condition":{
            "StringEquals":{
                    "s3:prefix":[""], "s3:delimiter":["/"]
                          }
                }
    }
  ]
}
```

3. 更新されたアクセス許可をテストします。

   1. IAM ユーザーのサインインリンク (「IAM ユーザー用のサインインリンクを通知するには (p. 315)」を参照) を使用して AWS Management Console にサインインします。

      Amazon S3 コンソール (https://console.aws.amazon.com/s3/) を開きます。

   2. この演習で作成したバケットをクリックすると、今度はコンソールにルートレベルのバケットアイテムが表示されます。バケット内のフォルダをクリックしても、フォルダの内容は表示されません。そのためのアクセス許可が付与されていないからです。



ユーザーが Amazon S3 コンソールを使用している場合、このテストは成功します。コンソールでバケットをクリックすると、コンソールの実装により、空の文字列を値に持つ `prefix` パラメータと、`'/'` を値に持つ `delimiter` パラメータが送信されるためです。

ステップ 4.3: グループポリシーの概要

追加したグループポリシーによって、IAM ユーザーである Alice と Bob に次のような最小限のアクセス許可が付与されます。

- 親アカウントが所有するすべてのバケットを表示する。
- `companybucket` バケット内のルートレベルのアイテムを表示する。

しかし、これではユーザーが実行できる操作としてはまだ不十分です。そこで、次のようにユーザー固有のアクセス許可を付与します。

- Alice に Development フォルダのオブジェクトの読み書きを許可します。
- Bob に Finance フォルダのオブジェクトの読み書きを許可します。

ユーザー固有のアクセスを許可するには、グループではなく特定のユーザーにポリシーをアタッチします。次のセクションでは、Development フォルダを操作するためのアクセス許可を Alice に付与します。この手順を繰り返すと、Finance フォルダを操作するための同様のアクセス許可を Bob に付与することができます。

ステップ 5: IAM ユーザー Alice に特定のアクセス許可を付与する

次に、追加のアクセス許可を Alice に付与し、Alice が Development フォルダの内容を表示し、そのフォルダ内のオブジェクトを読み書きできるようにします。

ステップ 5.1: IAM ユーザー Alice に Development フォルダの内容を表示するためのアクセス許可を付与する

Alice が Development フォルダの内容を表示できるようにするには、`companybucket` バケットに対する `s3:ListBucket` アクションのためのアクセス許可を付与するポリシーをユーザーにアタッチする必要があります (プレフィックス `Development/` がリクエストに含まれることを前提とします)。

```
{
  "Statement": [
    {
      "Sid": "AllowListBucketIfSpecificPrefixIsIncludedInRequest",
      "Action": ["s3:ListBucket"],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::companybucket"],
      "Condition":{  "StringLike":{"s3:prefix":["Development/*"] }
       }
    }
  ]
}
```

1. AWS マネジメントコンソールにサインインして Amazon S3 コンソール (https://console.aws.amazon.com/s3/) を開きます。

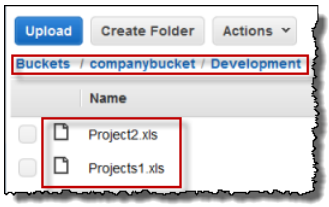
   IAM ユーザーの認証情報ではなく、AWS アカウントの認証情報を使用してコンソールにサインインします。
2. 前のステップで作成したポリシーをユーザー Alice にアタッチします。

   手順については、「*IAM を使用する*」の「Managing Policies (AWS Management Console) 」を参照してください。

3. Alice のアクセス許可の変更をテストします。

   a. IAM ユーザーのサインインリンク (「IAM ユーザー用のサインインリンクを通知するには (p. 315)」を参照) を使用して AWS Management Console にサインインします。
   b. Amazon S3 コンソール (https://console.aws.amazon.com/s3/) を開きます。
   c. Amazon S3 コンソールで、Alice がバケットの `Development/` フォルダ内のオブジェクトのリストを表示できることを確認します。

      ユーザーが `/Development` フォルダをクリックしてそのフォルダ内のオブジェクトのリストを表示すると、Amazon S3 コンソールから Amazon S3 に `ListObjects` リクエストが送信されます。このリクエストでは、プレフィックス `/Development` が指定されています。ユーザーにはプレフィックス `Development` と区切り記号 `'/'` を持つオブジェクトリストを表示するためのアクセス許可が付与されているため、Amazon S3 からは、キープレフィックス `Development/` を持つオブジェクトのリストが返され、コンソールにそのリストが表示されます。



ステップ 5.2: IAM ユーザー Alice に Development フォルダ内のオブジェクトを読み書きするためのアクセス許可を付与する

Alice が Development フォルダ内のオブジェクトを読み書きできるようにするには、`s3:GetObject` および `s3:PutObject` アクションを呼び出すためのアクセス許可が Alice に必要です。次のポリシーステートメントは、そのアクセス許可を付与します (値 `Development/` が指定された `prefix` パラメータがリクエストに含まれることを前提としています) 。

```
{
    "Sid":"AllowUserToReadWriteObjectData",
    "Action":["s3:GetObject", "s3:PutObject"],
    "Effect":"Allow",
    "Resource":["arn:aws:s3:::companybucket/Development/*"]
 }
```

1. AWS マネジメントコンソールにサインインして Amazon S3 コンソール (https://console.aws.amazon.com/s3/) を開きます。

   IAM ユーザーの認証情報ではなく、AWS アカウントの認証情報を使用してコンソールにサインインします。
2. 次のポリシーをユーザー Alice にアタッチします。

```
{
   "Statement":[
      {
         "Sid":"AllowListBucketIfSpecificPrefixIsIncludedInRequest",
         "Action":["s3:ListBucket"],
         "Effect":"Allow",
```

```
         "Resource":["arn:aws:s3:::companybucket"],
         "Condition":{
            "StringLike":{"s3:prefix":["Development/*"]
            }
         }
      },
      {
        "Sid":"AllowUserToReadWriteObjectDataInDevelopmentFolder",
        "Action":["s3:GetObject", "s3:PutObject"],
        "Effect":"Allow",
        "Resource":["arn:aws:s3:::companybucket/Development/*"]
      }
   ]
}
```

詳細な手順については、「*IAM を使用する*」の「Managing Policies（AWS Management Console）」を参照してください。

3. 更新されたポリシーをテストします。

   1. IAM ユーザーのサインインリンク（「IAM ユーザー用のサインインリンクを通知するには (p. 315)」を参照）を使用して AWS Management Console にサインインします。
   2. Amazon S3 コンソール（https://console.aws.amazon.com/s3/）を開きます。
   3. Amazon S3 コンソールで、Alice がオブジェクトを追加し、`Development` フォルダにダウンロードできるようになったことを確認します。

ステップ 5.3: バケットの他のフォルダへの IAM ユーザー Alice のアクセス許可を明示的に拒否する

ユーザー Alice が `companybucket` バケットのルートレベルの内容を表示できるようになったことを確認します。Alice は `Development` フォルダのオブジェクトを読み書きすることもできます。厳密にアクセス許可を制限したい場合は、バケットの他のフォルダへの Alice のアクセスを明示的に拒否することができます。バケットの他のフォルダへのアクセス許可を Alice に付与する他のポリシー（バケットポリシーまたは ACL）が存在する場合は、この明示的な拒否がそれらのアクセス許可よりも優先されます。

次のステートメントをユーザー Alice のポリシーに追加できます。これにより、Alice が Amazon S3 に送信するすべてのリクエストに必ず `prefix` パラメータが含まれるようになります。このパラメータの値は `Development/*` または空の文字列でなければなりません。

```
{
   "Sid": "ExplicitlyDenyAnyRequestsForAllOtherFoldersExceptDevelopment",
   "Action": ["s3:ListBucket"],
   "Effect": "Deny",
   "Resource": ["arn:aws:s3:::companybucket"],
   "Condition":{  "StringNotLike": {"s3:prefix":["Development/*"] },
                  "Null"         : {"s3:prefix":false }
    }
}
```

`Condition` ブロックに 2 つの条件式があることに注意してください。これらの条件式の結果は、論理 OR を使用して結合されます。いずれかの条件が true の場合、結合された条件の結果は true です。

- `Null` 条件式により、Alice からのリクエストには必ず `prefix` パラメータが含まれるようになります。

  `prefix` パラメータはフォルダのようなアクセスを必要とします。`prefix` パラメータのないリクエストを送信すると、Amazon S3 によりすべてのオブジェクトキーが返されます。

  リクエストに null 値の `prefix` パラメータが含まれている場合、式の評価結果は true になり、`Condition` 全体の評価結果が true になります。空の文字列を `prefix` パラメータの値として許可する必要があります。null 文字列を許可すると、この前の説明に示したコンソールでの操作と同様に、Alice はルートレベルのバケット項目を取得できるようになります。詳細については、「ステップ 4.2: バケットのルートレベルの内容をユーザーが表示できるようにする (p. 319)」を参照してください。
- 条件式 `StringNotLike` を使用すると、指定されている `prefix` パラメータの値が `Developement/*` でない場合はリクエストは失敗します。

更新されたユーザーポリシーは次のようになります。前のセクションの手順にしたがって、ユーザー Alice にアタッチしたポリシーを、この更新されたポリシーに置き換えます。

```
{
   "Statement":[
      {
         "Sid":"AllowListBucketIfSpecificPrefixIsIncludedInRequest",
         "Action":["s3:ListBucket"],
         "Effect":"Allow",
         "Resource":["arn:aws:s3:::companybucket"],
         "Condition":{
            "StringLike":{"s3:prefix":["Development/*"]
            }
         }
      },
      {
        "Sid":"AllowUserToReadWriteObjectDataInDevelopmentFolder",
        "Action":["s3:GetObject", "s3:PutObject"],
        "Effect":"Allow",
        "Resource":["arn:aws:s3:::companybucket/Development/*"]
      },
      {
        "Sid": "ExplicitlyDenyAnyRequestsForAllOtherFoldersExceptDevelopment",

         "Action": ["s3:ListBucket"],
         "Effect": "Deny",
         "Resource": ["arn:aws:s3:::companybucket"],
         "Condition":{  "StringNotLike": {"s3:prefix":["Development/*"] },
                        "Null"         : {"s3:prefix":false }
          }
      }
   ]
}
```

ステップ 6: IAM ユーザー Bob に特定のアクセス許可を付与する

次に、Finance フォルダへのアクセス許可を Bob に付与します。Alice にアクセス許可を付与するときに使用した手順にしたがいます。ただし、Development フォルダは Finance フォルダに置き換えます。手順については、「ステップ 5: IAM ユーザー Alice に特定のアクセス許可を付与する (p. 323)」を参照してください。

ステップ 7: Private フォルダをセキュリティで保護する

この例では、ユーザーは 2 名だけです。グループレベルで最小限必要なすべてのアクセス許可を付与し、ユーザーレベルのアクセス許可は、個々のユーザーレベルでアクセスを許可することが必要な場合にのみ付与しました。このようにすると、アクセス許可を管理する手間を最小限に抑えることができます。ユーザー数が増えるに従って、アクセス許可の管理は煩雑になります。例えば、この例のどのユーザーも Private フォルダの内容にアクセスできないようにするとします。このフォルダへのアクセス許可を誤って付与することがないようにするには、どのような方法があるでしょうか。このフォルダへのアクセスを明示的に拒否するポリシーを追加します。明示的な拒否は他のあらゆるアクセス許可よりも優先されます。Private フォルダを非公開に保つには、次の 2 つの拒否ステートメントをグループポリシーに追加します。

- 次のステートメントを追加して、Private フォルダ (companybucket/Private/*) のリソースに対するあらゆるアクションを明示的に拒否します。

```
{
  "Sid": "ExplictDenyAccessToPrivateFolderToEveryoneInTheGroup",
  "Action": ["s3:*"],
  "Effect": "Deny",
  "Resource":["arn:aws:s3:::companybucket/Private/*"]
}
```

- また、リクエストに Private/ プレフィックスが指定されている場合に、オブジェクトを表示するアクションに必要なアクセス許可を拒否します。コンソールで、Bob または Alice が Private フォルダをダブルクリックすると、このポリシーにより Amazon S3 がエラーレスポンスを返します。

```
{
  "Sid": "DenyListBucketOnPrivateFolder",
  "Action": ["s3:ListBucket"],
  "Effect": "Deny",
  "Resource": ["arn:aws:s3:::*"],
  "Condition":{
      "StringLike":{"s3:prefix":["Private/"]}
   }
}
```

グループポリシーを、更新された次のポリシー (前述の拒否ステートメントを含む) に置き換えます。こうすると、グループ内のどのユーザーも、バケット内の Private フォルダにアクセスできなくなります。

```
{
  "Statement": [
    {
      "Sid": "AllowGroupToSeeBucketListAndAlsoAllowGetBucketLocationRequired
ForListBucket",
      "Action": ["s3:ListAllMyBuckets", "s3:GetBucketLocation"],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::*"]
    },
    {
      "Sid": "AllowRootLevelListingOfCompanyBucket",
      "Action": ["s3:ListBucket"],
      "Effect": "Allow",
```

```
      "Resource": ["arn:aws:s3:::companybucket"],
      "Condition":{
          "StringEquals":{"s3:prefix":[""]}
       }
    },
    {
      "Sid": "RequireFolderStyleList",
      "Action": ["s3:ListBucket"],
      "Effect": "Deny",
      "Resource": ["arn:aws:s3:::*"],
      "Condition":{
          "StringNotEquals":{"s3:delimiter":"/"}
       }
     },
    {
      "Sid": "ExplictDenyAccessToPrivateFolderToEveryoneInTheGroup",
      "Action": ["s3:*"],
      "Effect": "Deny",
      "Resource":["arn:aws:s3:::companybucket/Private/*"]
    },
    {
      "Sid": "DenyListBucketOnPrivateFolder",
      "Action": ["s3:ListBucket"],
      "Effect": "Deny",
      "Resource": ["arn:aws:s3:::*"],
      "Condition":{
          "StringLike":{"s3:prefix":["Private/"]}
       }
    }
  ]
}
```

クリーンアップ

クリーンアップするには、IAM コンソールに移動し、ユーザー Alice と Bob を削除します。手順については、「*IAM を使用する*」の「Deleting a User from Your AWS Account」を参照してください。

ストレージに課金されないようにするには、この演習で作成したオブジェクトとバケットも削除してください。

バケットポリシーの使用

Topics

以下のセクションでは、Amazon S3 リソースでバケットポリシーを設定および管理する方法について説明します。

バケットポリシーの記述

バケットポリシーは、 Amazon S3 リソースのアクセス権を定義します。バケットポリシーを記述できるのは、バケット所有者のみです。バケット所有者は、次の目的でバケットポリシーを記述できます。

- バケットレベルのアクセス許可を許可/拒否する。
- バケット内の任意のオブジェクトに対するアクセス許可を拒否する。バケット所有者はバケットの財政的な責任者であるため、バケット内の任意のオブジェクトに対するアクセス許可を拒否するバケットポリシーを記述できます。
- バケット所有者がオブジェクト所有者である場合に限り、バケット内のオブジェクトに対するアクセス許可を付与する。他のアカウントによって所有されているオブジェクトの場合は、そのオブジェクトの所有者が ACL を使用してアクセス許可を管理する必要があります。

ポリシーはアクセスポリシー言語を使用して JSON 内に記述されます。アクセスポリシー言語とバケットポリシーを記述する方法の詳細については、「The access policy language (p. 485)」を参照してください。

AWS Policy Generator ツール

AWS Policy Generator ツールを使用し、Amazon S3 バケットのバケットポリシーを作成できます。その後、バケットポリシーを設定するには、Amazon S3 コンソール、サードパーティ製ツール、またはアプリケーションで、生成されたドキュメントを使用します。ポリシー生成ツールを使用するには、AWS Policy Generator にアクセスしてください。

バケットでのバケットポリシーの設定

バケットにポリシーを設定するには、`PUT Bucket` オペレーションを *`policy`* サブリソースで使用し、バケットポリシーをリクエストの本文に含めます。例えば、次のリクエストを使用すると、2 人のユーザー (`1111-2222-3333`、`4444-5555-6666`)) がアクセスして、`mybucket` (`arn:aws:s3:::mybucket/*"`,) 内のオブジェクトに対し `GET` リクエスト (`s3:GetObject*`) を実行できます。

```
PUT /?policy HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Tue, 04 Apr 2010 20:34:56 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:VGhpcyBSAMPLEBieSBlbHZpbmc=

{
"Version":"2012-10-17",
"Id":"aaaa-bbbb-cccc-dddd",
"Statement" : [
    {
        "Effect":"Allow",
        "Sid":"1",
        "Principal" : {
            "AWS":["1111-2222-3333","4444-5555-6666"]
        },
        "Action":["s3:GetObject*"],
        "Resource":"arn:aws:s3:::mybucket/*"
    }
 ]
}
```

Note

Resource 値には、バケット名を含める必要があります。

ポリシーをバケットにアタッチするには、バケット所有者である必要があります。バケット所有者には、デフォルトで PUT Bucket policy を使用して各自のバケットにバケットポリシーをアタッチする権限があります。バケットのポリシーが既に存在する場合は、リクエスト内のポリシーによって完全に置き換えられます。

詳細については、『Amazon S3 API Reference』の「*PUT Bucket policy*」を参照してください。

バケットでバケットポリシーの取得

指定したバケットでバケットポリシーを取得するには、GET Bucket オペレーションを *policy* サブリソースで使用します。次のリクエストは、バケット mybucket.s3.amazonaws.com のバケットポリシーを返します。

```
GET ?policy HTTP/1.1
Host: mybucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:VGhpcyBSAMPLEBieSBlbHZpbmc=
```

バケット所有者には、デフォルトで GET Bucket policy を使用してバケットポリシーを取得する権限があります。

詳細については、『Amazon S3 API Reference』の「*GET Bucket policy*」を参照してください。

バケットでのバケットポリシーの削除

バケットに関連付けられたポリシーを削除するには、DELETE Bucket オペレーションを *policy* サブリソースで使用します。次のリクエストは、mybucket と関連付けられたバケットポリシーを削除します。

```
DELETE /?policy HTTP/1.1
Host: mybucket.s3.amazonaws.com
Date: Tue, 04 Apr 2010 20:34:56 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:VGhpcyBSAMPLEeSBlbHZpbmc=
```

削除オペレーションを使用するには、指定したバケットに対する *DeletePolicy* アクセス許可を持っているバケット所有者である必要があります。バケット所有者には、デフォルトでバケットポリシーを削除する権限があります。

詳細については、『Amazon S3 API Reference』の「*DELETE Bucket policy*」を参照してください。

Amazon S3 バケットポリシーの参考例

このセクションでは、バケットポリシーの一般的なユースケース例をいくつか紹介します。以下のポリシーでは、リソース値の中で「bucket」および「examplebucket」の文字列が使用されています。これらのポリシーをテストするには、これらの文字列を使用するバケット名に置き換える必要があります。

Note

AWS Policy Generator ツールを使用し、Amazon S3 バケットのバケットポリシーを作成できます。その後、バケットポリシーを設定するには、Amazon S3 コンソール、サードパーティ製

ツール、またはアプリケーションで、生成されたドキュメントを使用します。ポリシー生成ツールを使用するには、AWS Policy Generator にアクセスしてください。

追加制限付きでの複数のアカウントへのアクセス許可の付与

次のポリシー例は、複数のアカウントに PutObject、PutObjectAcl アクセス許可を付与し、public-read 既定 ACL が含められることを要求します。

```
{
  "Version":"2012-10-17",
  "Statement":[{
 "Sid":"AddCannedAcl",
       "Effect":"Allow",
   "Principal": {
          "AWS":
["arn:aws:iam::111122223333:root","arn:aws:iam::444455556666:root"]
        },
   "Action":["s3:PutObject","s3:PutObjectAcl"
      ],
      "Resource":["arn:aws:s3:::bucket/*"
      ],
      "Condition":{
        "StringEquals":{
          "s3:x-amz-acl":["public-read"]
        }
      }
    }
  ]
}
```

匿名ユーザーへのアクセス許可の付与

次のポリシー例では、匿名ユーザーにアクセス許可を付与します。

```
{
  "Version":"2012-10-17",
  "Statement":[{
 "Sid":"AddPerm",
       "Effect":"Allow",
   "Principal": {
          "AWS": "*"
        },
      "Action":["s3:GetObject"],
      "Resource":["arn:aws:s3:::bucket/*"
      ]
    }
  ]
}
```

特定の IP アドレスへのアクセスの制限

このステートメントは、指定したバケット内のオブジェクトに対して任意の S3 アクションを実行するアクセス許可をすべてのユーザーに付与します。ただし、リクエストは条件で指定された IP アドレス範囲からのリクエストである必要があります。このステートメントの条件では、192.168.143.* の範囲の IP アドレスが許可されています。ただし、192.168.143.188 を除きます。

条件で指定されている IPAddress と NotIpAddress の値は、RFC 2632 の CIDR 表記を使用していることに注意してください。詳細については、「http://www.rfc-editor.org/rfc/rfc4632.txt」を参照してください。

```
{
    "Version": "2012-10-17",
    "Id": "S3PolicyId1",
    "Statement": [
        {
            "Sid": "IPAllow",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::bucket/*",
            "Condition" : {
                "IpAddress" : {
                    "aws:SourceIp": "192.168.143.0/24"
                },
                "NotIpAddress" : {
                    "aws:SourceIp": "192.168.143.188/32"
                }
            }
        }
    ]
}
```

特定の HTTP Referrer へのアクセスの制限

ドメイン名が www.example.com または example.com のウェブサイトがあり、そのドメインの Amazon S3 バケット、examplebucket には、写真や動画へのリンクが格納されているとします。デフォルトでは、すべての Amazon S3 リソースはプライベートです。リソースを作成した AWS アカウントのみが、そのリソースにアクセスできます。ウェブサイト内のオブジェクトへの読み取りアクセスを許可するには、aws:referer キーを使用して特定のウェブサイトからのみ取得リクエストを発信できるという条件で s3:GetObject アクションを許可する、バケットポリシーを追加します。

次のバケットポリシーは、特定の参照子からリクエストが発信された場合に、examplebucket バケット内のすべてのオブジェクトに対する s3:GetObject アクションを許可します。

```
{
  "Version":"2012-10-17",
  "Id":"http referer policy example",
  "Statement":[
    {
     "Sid":"Allow get requests originated from www.example.com and example.com",

      "Effect":"Allow",
      "Principal":"*",
      "Action":"s3:GetObject",
      "Resource":"arn:aws:s3:::examplebucket/*",
      "Condition":{
        "StringLike":{
          "aws:Referer":[
            "http://www.example.com/*",
            "http://example.com/*"
```

```
          ]
        }
      }
    }
  ]
}
```

examplebucket バケット内のオブジェクトへのアクセスのセキュリティを強化するには、次のバケットポリシーに示すように、バケットポリシーに明示的な拒否を追加します。明示的な拒否は、ACL やユーザーポリシーなどの他の方法を使用して examplebucket バケット内のオブジェクトに付与したすべてのアクセス許可よりも優先されます。

```
{
 "Version": "2012-10-17",
 "Id": "http referer policy example",
 "Statement": [
  {
   "Sid": "Allow get requests referred by www.mysite.com and mysite.com",
   "Effect": "Allow",
   "Principal": "*",
   "Action": "s3:GetObject",
   "Resource": "arn:aws:s3:::examplebucket/*",
   "Condition": {
    "StringLike": {
     "aws:Referer": [
      "http://www.example.com/*",
      "http://example.com/*"
     ]
    }
   }
  },
  {
   "Sid": "Explicit deny to ensure requests are allowed only from specific
referer.",
   "Effect": "Deny",
   "Principal": "*",
   "Action": "s3:*",
   "Resource": "arn:aws:s3:::examplebucket/*",
   "Condition": {
    "StringNotLike": {
     "aws:Referer": [
      "http://www.example.com/*",
      "http://example.com/*"
     ]
    }
   }
  }
 ]
}
```

使用するブラウザのリクエストに http 参照子ヘッダーが含まれることを確認する必要があります。

S3 バケットに対するログ配信を有効にするためのアクセス許可の付与

次のポリシー例は、Amazon S3 バケットに対するログ配信を有効にします。次のポリシーで指定されているアカウントは、Log Delivery グループです。Log Delivery グループを識別するために、このポリシーで指定された ARN を使用する必要があります。詳細については、「サーバーアクセスログギングのセットアップ (p. 470)」を参照してください。

```
{
 "Version":"2012-10-17",
 "Id":"LogPolicy",
 "Statement":[{
   "Sid":"Enables the log delivery group to publish logs to your bucket ",
   "Effect":"Allow",
   "Principal":{
    "AWS":"arn:aws:iam::111122223333:root"
   },
   "Action":["s3:GetBucketAcl",
    "s3:GetObjectAcl",
    "s3:PutObject"
   ],
   "Resource":["arn:aws:s3:::example-bucket",
    "arn:aws:s3:::example-bucket/*"
   ]
  }
 ]
}
```

正規 ID を使用した CloudFront オリジンアクセスアイデンティティへのアクセス許可の付与

次のバケットポリシー例は、Amazon S3 バケット内のすべてのオブジェクトを取得するために CloudFront オリジンアイデンティティへのアクセス許可を付与します。CloudFront Origin Identity は、CloudFront のプライベートコンテンツ機能を有効にするために使用します。このポリシーでは AWS ではなく、CanonicalUser プレフィックスを使用して正規ユーザー ID を指定しています。プライベートコンテンツの提供に関する CloudFront のサポートの詳細については、『Amazon CloudFront 開発者ガイド』の Serving Private Content に関するトピックを参照してください。CloudFront ディストリビューションのオリジンアクセスアイデンティティには、正規ユーザー ID を指定する必要があります。

```
{
 "Version":"2012-10-17",
 "Id":"PolicyForCloudFrontPrivateContent",
 "Statement":[{
   "Sid":" Grant a CloudFront Origin Identity access to support private content",

   "Effect":"Allow",
   "Principal":{
   "CanonicalUser":"79a59df900b949e55d96a1e698fbaced
fd6e09d98eacf8f8d5218e7cd47ef2be"
   },
   "Action":"s3:GetObject",
   "Resource":"arn:aws:s3:::example-bucket/*"
  }
 ]
```

```
}
```

MFA 認証を要求するバケットポリシーの追加

Amazon S3 は、MFA で保護された API アクセスをサポートしています。つまり、S3 リソースへのアクセス時に AWS 多要素認証を適用することができます。AWS Multi-Factor Authentication により、AWS 環境に適用できるセキュリティのレベルが高まります。これは、ユーザーが、有効な MFA コードを提供することによって、MFA デバイスの物理的所有を証明する必要があるセキュリティ機能です。詳細については、「AWS Multi-Factor Authentication」を参照してください。Amazon S3 リソースにアクセスするすべてのリクエストに、MFA 認証を要求することができます。

バケットポリシーで、`aws:MultiFactorAuthAge` キーを使用して、MFA 認証要件を適用できます。IAM ユーザーは、AWS Security Token Service (STS) によって発行された一時的な認証情報を使用して、S3 リソースにアクセスできます。STS リクエスト時に、MFA コードを指定します。

Amazon S3 が MFA 認証付きのリクエストを受信すると、`aws:MultiFactorAuthAge` キーから、一時的な認証情報が作成されてからの時間 (秒単位) を示す数値が渡されます。リクエストで提供された一時的な認証情報が MFA デバイスを使用して作成されていない場合、このキー値は null (不在) になります。次のバケットポリシー例に示すように、バケットポリシーに、この値を確認する条件を追加できます。このポリシーは、リクエストが MFA 認証されていない場合に、`examplebucket` バケット内の `/taxdocuments` フォルダへの Amazon S3 アクションを拒否します。MFA 認証の詳細については、「Configuring MFA-Protected API Access」を参照してください。

```
{
   "Version": "2012-10-17",
   "Id": "123",
   "Statement": [
      {
         "Sid": "",
         "Effect": "Deny",
         "Principal": { "AWS": "*" },
         "Action": "s3:**",
         "Resource": "arn:aws:s3:::examplebucket/taxdocuments/*",
         "Condition": { "Null": { "aws:MultiFactorAuthAge": true }}
      }
   ]
}
```

`aws:MultiFactorAuthAge` キー値が null で、リクエスト内の一時的なセキュリティ認証情報が MFA キーを使用せずに作成されたことを示している場合、`Condition` ブロック内の `Null` 条件の評価は true になります。

次のバケットポリシーは、前述のバケットポリシーの拡張です。2 つのポリシーステートメントが含まれています。1 つのステートメントは、全員にバケット (`examplebucket`) への `s3:GetObject` アクションを許可し、もう 1 つのステートメントは、MFA 認証を要求することによって、バケット内の `examplebucket/taxdocuments` フォルダへのアクセスの制限を強化します。

```
{
   "Version": "2012-10-17",
   "Id": "123",
   "Statement": [
      {
         "Sid": "",
         "Effect": "Deny",
```

```
         "Principal": { "AWS": "*"},
         "Action": "s3:**",
         "Resource": "arn:aws:s3:::examplebucket/taxdocuments/*",
         "Condition": { "Null": { "aws:MultiFactorAuthAge": true } }
      },
      {
         "Sid": "",
         "Effect": "Allow",
         "Principal": {"AWS": "*" },
         "Action": ["s3:GetObject"],
         "Resource": "arn:aws:s3:::examplebucket/*"
      }
   ]
}
```

オプションで、リクエストの認証に使われる一時的なセキュリティ認証情報の寿命に関係なく、aws:MultiFactorAuthAge キーの有効期間を制限する数値条件を使用することができます。例えば、次のバケットポリシーでは、MFA 認証を要求するほかに、一時セッションが作成されてからの時間も確認します。このポリシーは、aws:MultiFactorAuthAge キー値が、一時セッションが 1 時間 (3,600 秒) 以上前に作成されたことを示す場合に、すべてのアクションを拒否します。

```
{
   "Version": "2012-10-17",
   "Id": "123",
   "Statement": [
      {
         "Sid": "",
         "Effect": "Deny",
         "Principal": { "AWS": "*" },
         "Action": "s3:**",
         "Resource": "arn:aws:s3:::examplebucket/taxdocuments/*",
         "Condition": {"Null": {"aws:MultiFactorAuthAge": true }
         }
      },
      {
         "Sid": "",
         "Effect": "Deny",
         "Principal": { "AWS": "*"},
         "Action": "s3:**",
         "Resource": "arn:aws:s3:::examplebucket/taxdocuments/*",
         "Condition": {"NumericGreaterThan": {"aws:MultiFactorAuthAge": 3600 }
 }
      },
      {
         "Sid": "",
         "Effect": "Allow",
         "Principal": {"AWS": "*"},
         "Action": ["s3:GetObject"],
         "Resource": "arn:aws:s3:::examplebucket/*"
      }
   ]
}
```

バケット所有者はフルコントロール権限を持ちながら、オブジェクトをアップロードするクロスアカウントアクセス許可を付与する

他の AWS アカウントがご自分のバケットにオブジェクトをアップロードすることを許可できます。ただし、バケット所有者として、バケットにアップロードされたオブジェクトに対してフルコントロール権限を持っている必要がある場合があります。次のポリシーは、特定の AWS アカウント (111111111111) が、メールアドレス (xyz@amazon.com) で識別されるバケット所有者がフルコントロールアクセス許可を付与しない限り、そのアカウントによるオブジェクトのアップロードを拒否する例です。

ポリシーは、`PutObject` リクエストに `x-amz-grant-full-control` リクエストヘッダーを含めるように要求します。詳細については、「PUT Object」を参照してください。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"111",
         "Effect":"Allow",
         "Principal":{
            "AWS":"1111111111"
         },
         "Action":"s3:PutObject",
         "Resource":"arn:aws:s3:::examplebucket/*"
      },
      {
         "Sid":"112",
         "Effect":"Deny",
         "Principal":{
            "AWS":"1111111111"
         },
         "Action":"s3:PutObject",
         "Resource":"arn:aws:s3:::examplebucket/*",
         "Condition":{
            "StringNotEquals":{
               "s3:x-amz-grant-full-control":[
                  "emailAddress=xyz@amazon.com"
               ]
            }
         }
      }
   ]
}
```

バケットポリシーでのリソース、プリンシパル、オペレーション、条件の使用方法

Topics

バケットポリシーでの Amazon S3 リソースの指定

バケットポリシーでは、バケットおよびオブジェクトを参照できます。Amazon S3 ポリシーでバケットやオブジェクトを指定するには、次のように Amazon リソースネーム (ARN) 形式を使用します。

```
arn:aws:s3:::[resourcename]
```

リソース名とは、ユーザーがアクセスをリクエストするバケットまたはオブジェクトの完全修飾名です。バケットの場合、リソース名は bucketname です。ここで、*bucketname* はバケットの名前です。オブジェクトの場合、リソース名の形式は、bucketname/keyname です。ここで、*bucketname* はバケット名、*keyname* はオブジェクトの完全名です。例えば、「Ooyala」という名前のバケットと shared/developer/settings.conf という名前のオブジェクトがあるとします。この場合、バケットのリソース名は Ooyala、オブジェクトのリソース名は Ooyala/shared/developer/settings.conf になります。

バケットポリシーでのプリンシパルの指定

Principal とは、ポリシーにしたがってアクセス許可を付与または拒否される 1 人以上のユーザーです。プリンシパルは、そのプリンシパルの AWS アカウント ID (例えば、1234-5678-9012、ハイフンの有無は任意) を使用して指定する必要があります。AWS アカウント ID は、AWS アカウントまたは IAM ユーザーのどちらかのものとすることができます。複数のプリンシパルを指定したり、指定可能なすべてのユーザーを示すワイルドカード (*) を指定することができます。ご自分のアカウント ID を確認するには、http://aws.amazon.com で AWS アカウントにログインし、[Accounts] タブの [Account Activity] をクリックします。

AWS アカウントにアクセス許可を付与するときに、AWS アカウント ID を指定する代わりに正規ユーザー ID を指定することも可能です。自分の正規ユーザー ID を確認するには、http://aws.amazon.com で AWS アカウントにログインし、[Accounts] タブで [Security Credentials] をクリックします。また、CloudFront オリジンアクセスアイデンティティと関連付けられた正規ユーザー ID を使用して、このアイデンティティを付与できます。プライベートコンテンツの提供に関する CloudFront のサポートの詳細については、『Amazon CloudFront 開発者ガイド.』の Serving Private Content に関するトピックを参照してください。CloudFront ディストリビューションのオリジンアイデンティティには、AWS アカウントではなく、正規ユーザー ID を指定する必要があります。

JSON では、"AWS": をプリンシパルの AWS アカウント ID のプレフィックスとして使用し、"CanonicalUser": をプリンシパルの AWS 正規ユーザー ID のプレフィックスとして使用します。

Note

他の AWS アカウントにお客様の AWS リソースへのアクセスを許可した場合、その AWS アカウントはアカウント内のユーザーにアクセス許可を譲渡できることに注意してください。これはクロスアカウントアクセスと呼ばれます。クロスアカウントアクセスの使用については、「AWS Identity and Access Management」の「Cross-Account Access Using Resource-Based Policies」を参照してください。

Amazon S3 アクション

次のリストは、ポリシーで参照できる Amazon S3 アクションの形式を示しています。

オブジェクトに関連するアクション

- `s3:GetObject` (REST GET Object、REST HEAD Object、REST GET Object torrent、SOAP `GetObject`、および SOAP `GetObjectExtended` を対象とします)

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

- `s3:GetObjectVersion` (REST GET Object、REST HEAD Object、REST GET Object torrent、SOAP `GetObject`、および SOAP `GetObjectExtended` を対象とします)
- `s3:PutObject` (REST PUT Object、REST POST Object、REST Initiate Multipart Upload、REST Upload Part、REST Complete Multipart Upload、SOAP `PutObject`、および SOAP `PutObjectInline` を対象とします)
- `s3:GetObjectAcl`
- `s3:GetObjectVersionAcl`
- `s3:PutObjectAcl`
- `s3:PutObjectVersionAcl`
- `s3:DeleteObject`
- `s3:DeleteObjectVersion`
- `s3:ListMultipartUploadParts`
- `s3:AbortMultipartUpload`
- `s3:GetObjectTorrent`
- `s3:GetObjectVersionTorrent`
- `s3:RestoreObject`

バケットに関連するアクション

- `s3:CreateBucket`
- `s3:DeleteBucket`
- `s3:ListBucket`
- `s3:ListBucketVersions`
- `s3:ListAllMyBuckets` (REST GET Service と SOAP `ListAllMyBuckets` を対象とします)
- `s3:ListBucketMultipartUploads`

バケットのサブリソースに関連するアクション

- `s3:GetBucketAcl`
- `s3:PutBucketAcl`
- `s3:GetBucketCORS`
- `s3:PutBucketCORS`
- `s3:GetBucketVersioning`
- `s3:PutBucketVersioning`
- `s3:GetBucketRequesterPays`
- `s3:PutBucketRequesterPays`
- `s3:GetBucketLocation`
- `s3:PutBucketPolicy`

- s3:GetBucketPolicy
- s3:PutBucketNotification
- s3:GetBucketNotification
- s3:GetBucketLogging
- s3:PutBucketLogging
- s3:GetLifecycleConfiguration
- s3:PutLifecycleConfiguration

オブジェクトを削除するには、明示的に DELETE Object API を呼び出すか、Amazon S3 で自動的に削除できるようにライフサイクルを設定します (「オブジェクトの有効期限 (p. 123)」を参照)。バケットのオブジェクトをユーザーまたはアカウントが削除できないようにするには、ユーザーやアカウントによる s3:DeleteObject、s3:DeleteObjectVersion、および s3:PutLifecycleConfiguration アクションを拒否する必要があります。

Amazon S3 ポリシーで使用可能な AWS キー

AWS にはよく使用される一連のキーが用意されています。これらのキーは、アクセスポリシー言語を採用しているすべての AWS 製品でサポートされます。これらのキーのリストについては、『*AWS Identity and Access Management Using IAM*』ガイドの Available Keys に関するセクションを参照してください。

Amazon S3 にはアクション固有のキーもあります。これらのキーは、以下の表ではリソースの種類および適用可能なアクションごとにグループ化されています。キーの中には、複数の種類のリソースやアクションに適用できるものもあります。

Important

IAM は、Amazon S3 内でポリシーの有効性を評価することはできません。無効なキーとアクションの組み合わせを指定しても、ポリシーを IAM にアップロードしたときに IAM が例外を返すことはありません。また、Amazon S3 からもエラーメッセージは返されません。Amazon S3 が判断できるのは、ポリシーが条件を満たさないために適用不可能である場合のみです。しかし、ポリシー条件の指定方法が適切でない場合 (数値比較に文字列フィールドを使用するなど) は、Amazon S3 はリクエストに例外を返し、アクセスが拒否されます。

特に明記されていない限り、各キーは、アクセスポリシー言語の文字列条件で使用するために提供されています。詳細については、『*AWS Identity and Access Management Using IAM*』ガイドの「Condition」を参照してください。

Amazon S3 ポリシー内のバケットキー

次の表は、Amazon S3 ポリシーに含めることができるバケットに関連するキーを示しています。

Action	Applicable Keys	Description
`s3:CreateBucket`	`s3:x-amz-acl`	The Amazon S3 canned ACL that is applied to the bucket. A canned ACL represents a predefined permission that can be applied to the bucket being created. Valid values: `private` | `public-read` | `public-read-write` | `authenticated-read` | `bucket-owner-read` | `bucket-owner-full-control` | `log-delivery-write`. Example value: `public-read`
	`s3:LocationConstraint`	Specifies the region where the bucket will be created. Valid values are `us-west-1` (for Northern California) or `EU` (for Ireland). Do not specify a value for US Standard. Example value: `us-west-1`
	`s3:x-amz-grant-read`, `s3:x-amz-grant-write`, `s3:x-amz-grant-read-acp`, `s3:x-amz-grant-write-acp`, `s3:x-amz-grant-full-control`	These conditions relate to the ACL-specific request headers the action supports. You can add a condition in your bucket policy to deny or grant the action based on the presence of these headers with specific values. For more information about the API and the request headers, see PUT Bucket. For an example policy that uses these condition keys, see バケット所有者はフルコントロール権限を持ちながら、オブジェクトをアップロードするクロスアカウントアクセス許可を付与する (p. 337).

Action	Applicable Keys	Description
s3:ListBucket	s3:prefix	Limits the response to objects that begin with the specified prefix. Use this to allow or deny access to objects that begin with the prefix. Example value: home
	s3:delimiter	The character you use to group objects. Example value: /
	s3:max-keys	The number of objects to return from the call. The maximum allowed value (and default) is 1000. For use with access policy language numeric conditions. For more information, see 数値の条件 (p. 501). Example value: 100
s3:ListBucketVersions	s3:prefix	Header that lets you limit the response to include only keys that begin with the specified prefix. Example value: home
	s3:delimiter	The character you use to group objects. Example value: /
	s3:max-keys	The number of objects to return from the call. The maximum allowed value (and default) is 1000. For use with access policy language numeric conditions. For more information, see 数値の条件 (p. 501). Example value: 100

Action	Applicable Keys	Description
`s3:PutBucketAcl`	`s3:x-amz-acl`	The Amazon S3 canned ACL that is applied to the bucket. A canned ACL represents a predefined permission that can be applied to the bucket being created. Valid values: `private` | `public-read` | `public-read-write` | `authenticated-read` | `bucket-owner-read` | `bucket-owner-full-control` | `log-delivery-write`. Example value: `public-read`
	`s3:x-amz-grant-read`, `s3:x-amz-grant-write`, `s3:x-amz-grant-read-acp`, `s3:x-amz-grant-write-acp`, `s3:x-amz-grant-full-control`	These conditions relate to the ACL-specific request headers the action supports. You can add a condition in your bucket policy to deny or grant the action based on the presence of these headers with specific values. For more information about the API and the request headers, see PUT Bucket acl. For an example policy that uses these condition keys, see バケット所有者はフルコントロール権限を持ちながら、オブジェクトをアップロードするクロスアカウントアクセス許可を付与する (p. 337).

Amazon S3 ポリシー内のオブジェクトキー

次のリストは、Amazon S3 ポリシーに含めることができるオブジェクトに関連するキーを示しています。

Action	Applicable Keys	Description
`s3:PutObject`	`s3:x-amz-acl`	The Amazon S3 canned ACL that is applied to the object. A canned ACL represents a predefined permission that can be applied to the object being PUT to Amazon S3. Valid values: `private` | `public-read` | `public-read-write` | `authenticated-read` | `bucket-owner-read` | `bucket-owner-full-control` | `log-delivery-write`. Example value: `public-read`
	`s3:x-amz-copy-source`	The header that specifies the name of the source bucket and key name of the source object, separated by a slash (/). Used when copying an object. Example value: `/bucketname/keyname`
	`s3:x-amz-grant-read`, `s3:x-amz-grant-write`, `s3:x-amz-grant-read-acp`, `s3:x-amz-grant-write-acp`, `s3:x-amz-grant-full-control`	These conditions relate to the ACL-specific request headers the action supports. You can add a condition in your bucket policy to deny or grant the action based on the presence of these headers with specific values. For more information about the API and the request headers, see PUT Object. For an example policy that uses these condition keys, see バケット所有者はフルコントロール権限を持ちながら、オブジェクトをアップロードするクロスアカウントアクセス許可を付与する (p. 337).
	`s3:x-amz-server-side-encryption`	Allow the specific action only if `x-amz-server-side-encryption` header is present in the request and its value matches the specified condition. Valid values: AES256 Example value: `AES256`
	`s3:x-amz-metadata-directive`	

Action	Applicable Keys	Description
		The header that specifies whether the metadata is copied from the source object or replaced with metadata provided in the request. If copied, the metadata, except for the version ID, remains unchanged. Otherwise, all original metadata is replaced by the metadata you specify. Used when copying an object. Valid values: `COPY` | `REPLACE`. The default is `COPY`. Example value: `REPLACE`
`s3:PutObjectAcl`	`s3:x-amz-acl`	The Amazon S3 canned ACL that is applied to the object. A canned ACL represents a predefined permission that can be applied to the object being PUT to Amazon S3. Valid values: `private` | `public-read` | `public-read-write` | `authenticated-read` | `bucket-owner-read` | `bucket-owner-full-control` | `log-delivery-write`. Example value: `public-read`
	`s3:x-amz-grant-read`, `s3:x-amz-grant-write`, `s3:x-amz-grant-read-acp`, `s3:x-amz-grant-write-acp`, `s3:x-amz-grant-full-control`	These conditions relate to the ACL-specific request headers the action supports. You can add a condition in your bucket policy to deny or grant the action based on the presence of these headers with specific values. For more information about the API and the request headers, see PUT Object acl. For an example policy that uses these condition keys, see バケット所有者はフルコントロール権限を持ちながら、オブジェクトをアップロードするクロスアカウントアクセス許可を付与する (p. 337).
`s3:GetObjectVersion`	`s3:VersionId`	The version ID of the object being retrieved. Example value: `Upfdndhfd8438MNFDN93jdnJFkdmqnh893`

Action	Applicable Keys	Description
`s3:GetObjectVersionAcl`	`s3:VersionId`	The version ID of the object ACL being retrieved. Example value: `Upfdndhfd8438MNFDN93jdnJFkdmqnh893`
`s3:PutObjectVersionAcl`	`s3:VersionId`	The version ID of the object ACL being PUT. Example value: `Upfdndhfd8438MNFDN93jdnJFkdmqnh893`
	`s3:x-amz-acl`	The Amazon S3 canned ACL that is applied to the object. A canned ACL represents a predefined permission that can be applied to the object being PUT to Amazon S3. Valid values: `private` | `public-read` | `public-read-write` | `authenticated-read` | `bucket-owner-read` | `bucket-owner-full-control` | `log-delivery-write`. Example value: `public-read`
	`s3:x-amz-grant-read`, `s3:x-amz-grant-write`, `s3:x-amz-grant-read-acp`, `s3:x-amz-grant-write-acp`, `s3:x-amz-grant-full-control`	These conditions relate to the ACL-specific request headers the action supports. You can add a condition in your bucket policy to deny or grant the action based on the presence of these headers with specific values. For more information about the API and the request headers, see PUT Object acl. For an example policy that uses these condition keys, see バケット所有者はフルコントロール権限を持ちながら、オブジェクトをアップロードするクロスアカウントアクセス許可を付与する (p. 337).
`s3:DeleteObjectVersion`	`s3:VersionId`	The version ID of the object being deleted. Example value: `Upfdndhfd8438MNFDN93jdnJFkdmqnh893`

ACL の使用

Topics

アクセスコントロールリスト（ACL）の概要

Topics

Amazon S3 アクセスコントロールリスト（ACL）では、バケットとオブジェクトへのアクセスを管理できます。各バケットとオブジェクトには、サブリソースとして ACL がアタッチされています。これにより、アクセスが許可される AWS アカウントまたはグループと、アクセスの種類が定義されます。リソースに対するリクエストを受信すると、Amazon S3 は該当する ACL を調べて、必要なアクセス許可がリクエスタにあることを確認します。

バケットまたはオブジェクトを作成すると、以下のバケット ACL のサンプルに示すように、Amazon S3 はリソースに対する完全なコントロールをリソース所有者に付与するデフォルト ACL を作成します（デフォルトのオブジェクト ACL は同じ構造です）。

```
<?xml version="1.0" encoding="UTF-8"?>
<AccessControlPolicy xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Owner>
    <ID>*** Owner-Canonical-User-ID ***</ID>
    <DisplayName>owner-display-name</DisplayName>
  </Owner>
  <AccessControlList>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:type="Canonical User">
        <ID>*** Owner-Canonical-User-ID ***</ID>
        <DisplayName>display-name</DisplayName>
      </Grantee>
      <Permission>FULL_CONTROL</Permission>
    </Grant>
  </AccessControlList>
</AccessControlPolicy>
```

サンプル ACL には、AWS アカウントの正規ユーザー ID を通じて所有者を識別する `Owner` 要素が含まれています。`Grant` 要素は、被付与者（AWS アカウントまたはあらかじめ定義されたグループ）と付与されたアクセス許可を識別します。このデフォルトの ACL には、所有者に対する1つの `Grant` 要素があります。`Grant` 要素を追加してアクセス許可を付与します。各許可は被付与者とアクセス許可を識別します。

Note

1 つの ACL には最大 100 個の許可を指定することができます。

被付与者とは

被付与者は、AWS アカウントまたはあらかじめ定義されたいずれかの Amazon S3 グループとすることができます。メールアドレスまたは正規ユーザー ID による AWS アカウントにアクセス許可を付与します。ただし、許可リクエストにメールを指定した場合でも、Amazon S3 はそのアカウントの正規ユーザー ID を検出し、ACL に追加します。その結果、ACL には AWS アカウントのメールアドレスではなく、常に AWS アカウントの正規ユーザー ID が格納されます。

Amazon S3 にはあらかじめ定義された一連のグループがあります。グループにアカウントアクセスを許可するときは、正規ユーザー ID の代わりに当社のいずれかの URI を指定します。あらかじめ定義された以下のグループが用意されています。

- Authenticated Users グループ – `http://acs.amazonaws.com/groups/global/AuthenticatedUsers` で表されます。
  このグループはすべての Amazon AWS アカウントを表します。このグループへのアクセス許可により、任意の Amazon AWS アカウントがリソースにアクセスすることが許可されます。ただし、すべてのリクエストは署名（認証）されている必要があります。
- All Users グループ – `http://acs.amazonaws.com/groups/global/AllUsers` によって表されます。
  このグループへのアクセス許可により、誰でもリソースにアクセスすることが許可されます。リクエストは署名（認証）済み、または署名なし（匿名）とすることができます。署名なしのリクエストでは、リクエストの Authentication ヘッダーが省略されます。
- Log Delivery グループ – `http://acs.amazonaws.com/groups/s3/LogDelivery` によって表されます。
  バケットの WRITE 許可により、このグループはサーバーアクセスログ（「サーバーアクセスのロギング (p. 461)」を参照）をバケットに書き込むことができます。

Note

ACL を使用するときの被付与者は、AWS アカウントまたはあらかじめ定義されたいずれかの Amazon S3 グループとすることができます。被付与者を IAM ユーザーとすることはできません。IAM 内の AWS ユーザーとアクセス許可の詳細については、「Using AWS Identity and Access Management」を参照してください。

Note

他の AWS アカウントに自分のリソースへのアクセスを許可した場合、その AWS アカウントはアカウント内のユーザーにアクセス許可を譲渡できることに注意してください。これはクロスアカウントアクセスと呼ばれます。クロスアカウントアクセスの使用については、「AWS Identity and Access Management」の「Cross-Account Access Using Resource-Based Policies」を参照してください。

正規ユーザー ID の検索

正規ユーザー ID は、AWS アカウントに関連付けられています。この ID は次の方法で検索できます。

正規ユーザー ID を検索するには

1. 「Security Credentials」に移動します。

2. まだログインしていない場合は、[アマゾン ウェブ サービス] にサインインするよう求められます。
3. AWS アカウントに関連付けられた正規ユーザー ID の [Account Identifier] セクションに移動します。

また、AWS アカウントがアクセス許可を持つバケットまたはオブジェクトの ACL を読み取って、AWS アカウントの正規ユーザー ID を検索することもできます。個別の AWS アカウントにアクセス許可を付与すると、ACL には AWS アカウントの正規ユーザー ID が含まれた許可エントリが作成されます。

付与できるアクセス許可

次の表は、ACL で Amazon S3 がサポートする一連のアクセス許可をまとめたものです。この表では、アクセス許可のリストと、オブジェクトとバケットのアクセス許可におけるその意味について説明しています。

アクセス許可	バケットで付与された場合	オブジェクトで付与された場合
READ	被付与者がバケット内のオブジェクトをリストすることを許可します	被付与者がオブジェクトデータとそのメタデータを読み取ることを許可します
WRITE	被付与者がバケット内のオブジェクトを作成、上書き、削除することを許可します	該当しません
READ_ACP	被付与者がバケット ACL を読み取ることを許可します	被付与者がオブジェクト ACL を読み取ることを許可します
WRITE_ACP	被付与者が該当するバケットの ACL を書き込むことを許可します	被付与者が該当するオブジェクトの ACL を書き込むことを許可します
FULL_CONTROL	バケットの READ、WRITE、READ_ACP、WRITE_ACP アクセス許可を被付与者に許可します	オブジェクトの READ、READ_ACP、WRITE_ACP アクセス許可を被付与者に許可します

サンプル ACL

バケットの以下のサンプル ACL は、リソース所有者と一連の許可を識別します。形式は Amazon S3 REST API の ACL の XML 表現です。バケット所有者にはリソースに対する FULL_CONTROL が許可されます。さらに、この ACL では、正規ユーザー ID で識別される 2 つの AWS アカウントと、前のセクションで説明した、あらかじめ定義された 2 つの Amazon S3 グループに対して、リソースへのアクセス許可が付与される方法が示されています。

```
<?xml version="1.0" encoding="UTF-8"?>
<AccessControlPolicy xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Owner>
    <ID>Owner-canonical-user-ID</ID>
    <DisplayName>display-name</DisplayName>
  </Owner>
  <AccessControlList>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="CanonicalUser">
        <ID>Owner-canonical-user-ID</ID>
        <DisplayName>display-name</DisplayName>
      </Grantee>
      <Permission>FULL_CONTROL</Permission>
```

```
    </Grant>

    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="CanonicalUser">
        <ID>user1-canonical-user-ID</ID>
        <DisplayName>display-name</DisplayName>
      </Grantee>
      <Permission>WRITE</Permission>
    </Grant>

    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="CanonicalUser">
        <ID>user2-canonical-user-ID</ID>
        <DisplayName>display-name</DisplayName>
      </Grantee>
      <Permission>READ</Permission>
    </Grant>

    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="Group">
        <URI>http://acs.amazonaws.com/groups/global/AllUsers</URI>
      </Grantee>
      <Permission>READ</Permission>
    </Grant>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="Group">
        <URI>http://acs.amazonaws.com/groups/s3/LogDelivery</URI>
      </Grantee>
      <Permission>WRITE</Permission>
    </Grant>

  </AccessControlList>
</AccessControlPolicy>
```

既定 ACL

Amazon S3 は、既定 ACL と呼ばれる、あらかじめ定義された一連の許可をサポートしています。各既定 ACL には、あらかじめ定義された一連の被付与者とアクセス許可が含まれています。次の表は、一連の既定 ACL と、関連するあらかじめ定義された許可をまとめたものです。

既定 ACL	対象	ACL に追加されるアクセス許可
`private`	バケットとオブジェクト	所有者は `FULL_CONTROL` を取得します。他のユーザーにはアクセス許可は付与されません（デフォルト）。
`public-read`	バケットとオブジェクト	所有者は `FULL_CONTROL` を取得します。`AllUsers` グループ（「被付与者とは (p. 348)」を参照）は `READ` アクセス許可を取得します。
`public-read-write`	バケットとオブジェクト	所有者は `FULL_CONTROL` を取得します。`AllUsers` グループは `READ` および `WRITE` アクセス許可を取得します。通常、これをバケットで付与することはお勧めしません。

既定 ACL	対象	ACL に追加されるアクセス許可
authenticated-read	バケットとオブジェクト	所有者は FULL_CONTROL を取得します。AuthenticatedUsers グループは READ アクセス許可を取得します。
bucket-owner-read	オブジェクト	オブジェクト所有者は FULL_CONTROL を取得します。バケット所有者は READ を取得します。バケットの作成時にこの既定 ACL を指定しても、Amazon S3 には無視されます。
bucket-owner-full-control	オブジェクト	オブジェクト所有者とバケット所有者はオブジェクトに対する FULL_CONTROL を取得します。バケットの作成時にこの既定 ACL を指定しても、Amazon S3 には無視されます。
log-delivery-write	バケット	LogDelivery グループはバケットに対する WRITE および READ_ACP アクセス許可を取得します。ログの詳細については、「サーバーアクセスのロギング (p. 461)」を参照してください。

Note

リクエストではこれらの既定 ACL を 1 つのみ指定できます。

x-amz-acl リクエストヘッダーを使用して、既定 ACL をリクエストで指定できます。Amazon S3 が既定 ACL を含むリクエストを受信すると、あらかじめ定義された許可がリソースの ACL に追加されます。

ACL の指定方法

Amazon S3 API を使用して、バケットまたはオブジェクトの作成時に、ACL を設定できます。Amazon S3 には、既存のバケットまたはオブジェクトに ACL を設定する API も用意されています。これらの API は ACL を設定する次のメソッドを提供します。

- リクエストヘッダーを使用した ACL の設定 – リソース (バケットまたはオブジェクト) を作成するためにリクエストを送る際に、リクエストヘッダーを使用して ACL を設定します。これらのヘッダーを使用して、既定 ACL を指定するか、許可を明示的に指定 (被付与者とアクセス許可を明示的に識別) します。
- リクエスト本文を使用した ACL の設定 – 既存のリソースに ACL を設定するリクエストを送信する際に、リクエストヘッダーまたは本文で ACL を設定できます。

詳細については、「ACL の管理 (p. 351)」を参照してください。

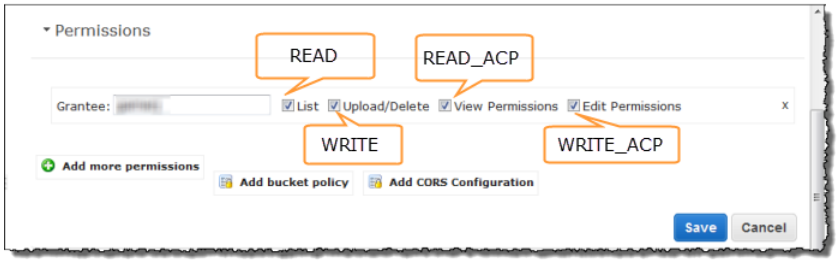
ACL の管理

Topics

リソース ACL に許可を追加する方法は複数あります。AWS マネジメントコンソールを使用できます。コンソールの UI を使用して、コードを記述することなくアクセス許可を管理できます。REST API またはいずれかの AWS SDK を使用できます。これらのライブラリにより、さらにプログラミングタスクが簡略化されます。

AWS Management Consoleでの ACL の管理

AWS Management Consoleには、バケットとオブジェクトに対する ACL ベースのアクセス許可を付与できる UI が用意されています。[Properties] ペインの [Permissions] タブでは、ACL に基づくアクセス許可を付与できます。次のスクリーンショットには、バケットの場合の [Permissions] が表示されています。



つまり、バケット ACL にある許可のリストが表示されています。許可ごとに、被付与者と、付与されたアクセス許可を示す一連のチェックボックスが表示されます。コンソール内のアクセス許可名は、ACL のアクセス許可名とは異なります (「付与できるアクセス許可 (p. 349)」を参照) 。上記の図には、これらの対応が示されています。

上記の図では、FULL_CONTROL アクセス許可が付与された被付与者が示されています。すべてのチェックボックスが選択されていることに注意してください。[Add bucket policy] リンクを除く、表示されているすべての UI コンポーネントは、ACL ベースのアクセス許可に関連します。UI により、アクセス許可を追加または削除できます。アクセス許可を追加するには、[Add more permissions] をクリックします。アクセス許可を削除するには、行をハイライトし、その右側にある [X] をクリックします。アクセス許可の更新を終了したら、[Save] をクリックして ACL を更新します。コンソールは必要なリクエストを Amazon S3 に送信して、特定のリソースの ACL を更新します。

詳細な手順については、『*Amazon Simple Storage Service Console User Guide*』の「Editing Object Permissions」と「Editing Bucket Permissions」を参照してください。

AWS SDK for Java を使用した ACL の管理

リソースの作成時の ACL の設定

リソース (バケットとオブジェクト) の作成時に、リクエストに AccessControlList を追加して、アクセス許可を付与できます (「アクセスコントロールリスト (ACL) の概要 (p. 347)」を参照) 。アクセス許可ごとに、被付与者とアクセス許可を明示的に指定します。

例えば、次の Java コードスニペットでは、オブジェクトをアップロードする PutObject リクエストを送信します。このリクエストでは、コードスニペットが 2 つの AWS アカウントと Amazon S3 AllUsers グループにアクセス許可を指定しています。PutObject 呼び出しには、リクエスト本文にオブジェクトデータが含まれ、リクエストヘッダーに ACL 許可が含まれます (「PUT Object」を参照) 。

```
String bucketName     = "bucket-name";
String keyName        = "object-key";
String uploadFileName = "file-name";
```

```
AmazonS3 s3client = new AmazonS3Client(new PropertiesCredentials(
        S3Sample.class.getResourceAsStream("AwsCredentials.properties")));

AccessControlList acl = new AccessControlList();
acl.grantPermission(new Canonic
alGrantee("d25639fbe9c19cd30a4c0f43fbf00e2d3f96400a9aa8dabfbbebe1906Example"),
 Permission.ReadAcp);
acl.grantPermission(GroupGrantee.AllUsers, Permission.Read);
acl.grantPermission(new EmailAddressGrantee("user@email.com"), Permis
sion.WriteAcp);

File file = new File(uploadFileName);
s3client.putObject(new PutObjectRequest(bucketName, keyName, file).withAccessCon
trolList(acl));
```

オブジェクトのアップロードについては、「Amazon S3 オブジェクトの使用 (p. 107)」を参照してください。

前述のコードスニペットでは、各アクセス許可の付与で、被付与者とアクセス許可を明示的に識別しました。または、リソースの作成時に、リクエストに既定 (定義済み) の ACL を指定できます (「既定 ACL (p. 350)」を参照) 。次の Java コードスニペットでは、バケットを作成し、Amazon S3 `LogDelivery` グループに書き込みアクセス許可を付与する既定 ACL `LogDeliveryWrite` をリクエストに指定します。

```
String bucketName     = "bucket-name";
AmazonS3 s3client = new AmazonS3Client(new PropertiesCredentials(
        S3Sample.class.getResourceAsStream("AwsCredentials.properties")));

s3client.createBucket(new CreateBucketRequest (bucketName).withCannedAcl(CannedAc
cessControlList.LogDeliveryWrite));
```

基本の REST API については、「PUT Bucket」を参照してください。

既存のリソースでの ACL の更新

既存のオブジェクトまたはバケットで ACL を設定できます。`AccessControlList` クラスのインスタンスを作成し、アクセス許可を付与し、適切な ACL 設定メソッドを呼び出します。次の Java コードスニペットでは、既存のオブジェクトで ACL を設定する `setObjectAcl` メソッドを呼び出します。

```
String bucketName     = "bucket-name";
String keyName        = "object-key";

AmazonS3 s3client = new AmazonS3Client(new PropertiesCredentials(
        S3Sample.class.getResourceAsStream("AwsCredentials.properties")));

AccessControlList acl = new AccessControlList();
acl.grantPermission(new Canonic
alGrantee("d25639fbe9c19cd30a4c0f43fbf00e2d3f96400a9aa8dabfbbebe1906Example"),
 Permission.ReadAcp);
acl.grantPermission(GroupGrantee.AuthenticatedUsers, Permission.Read);
acl.grantPermission(new EmailAddressGrantee("user@email.com"), Permis
sion.WriteAcp);
Owner owner = new Owner();
owner.setId("852b113e7a2f25102679df27bb0ae12b3f85be6f290b936c4393484beExample");
owner.setDisplayName("display-name");
acl.setOwner(owner);
```

```
s3client.setObjectAcl(bucketName, keyName, acl);
```

Note

前述のコードスニペットでは、getObjectAcl メソッドを呼び出すことによって、オプションでまず既存の ACL を読み取り、それに新しい許可を追加し、変更した ACL をリソースに設定できます。

被付与者とアクセス許可を明示的に指定することによってアクセス許可を付与する代わりに、リクエストに既定 ACL を指定することもできます。次の Java コードスニペットでは、既存のオブジェクトで ACL を設定します。このリクエストでは、スニペットは規定 ACL AuthenticatedRead を指定し、読み取り許可を Amazon S3 Authenticated Users グループに付与します。

```
String bucketName     = "bucket-name";
String keyName        = "object-key";

AmazonS3 s3client = new AmazonS3Client(new PropertiesCredentials(
        S3Sample.class.getResourceAsStream("AwsCredentials.properties")));

s3client.setObjectAcl(bucketName, keyName, CannedAccessControlList.Authentic
atedRead);
```

例

次の Java コード例では、最初にバケットを作成します。作成リクエストで public-read 既定 ACL を指定します。次に、AccessControlList インスタンスで ACL を取得し、許可をクリアし、新しい許可を AccessControlList に追加します。最後に、更新された AccessControlList を保存します。つまり、バケット ACL サブリソースを置き換えます。

次の Java コード例は、次のタスクを実行します。

- バケットを作成します。このリクエストでは、log-delivery-write 既定 ACL を指定し、書き込み許可を LogDelivery Amazon S3 グループに付与します。
- バケットの ACL を読み取ります。
- 既存のアクセス許可をクリアし、ACL に新しいアクセス許可を追加します。
- setBucketAcl を呼び出し、新しい ACL をバケットに追加します。

Note

次のコード例をテストするには、コードを更新して、認証情報を提供し、さらにアクセス許可を付与したいアカウントの正規ユーザー ID とメールアドレスも指定する必要があります。

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collection;
import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.AccessControlList;
```

```
import com.amazonaws.services.s3.model.Bucket;
import com.amazonaws.services.s3.model.CannedAccessControlList;
import com.amazonaws.services.s3.model.CanonicalGrantee;
import com.amazonaws.services.s3.model.CreateBucketRequest;
import com.amazonaws.services.s3.model.EmailAddressGrantee;
import com.amazonaws.services.s3.model.Grant;
import com.amazonaws.services.s3.model.GroupGrantee;
import com.amazonaws.services.s3.model.Permission;
import com.amazonaws.services.s3.model.Region;


public class S3Sample {
 private static String bucketName = "*** Provide bucket name ***";

 public static void main(String[] args) throws IOException {
        AmazonS3 s3Client = new AmazonS3Client(new PropertiesCredentials(
                S3Sample.class.getResourceAsStream(
                  "AwsCredentials.properties")));

        Collection<Grant> grantCollection = new ArrayList<Grant>();
        try {
            // 1. Create bucket with Canned ACL.
            CreateBucketRequest createBucketRequest =
             new CreateBucketRequest(bucketName, Region.US_Standard).with
CannedAcl(CannedAccessControlList.LogDeliveryWrite);

            Bucket resp = s3Client.createBucket(createBucketRequest);

            // 2. Update ACL on the existing bucket.
            AccessControlList bucketAcl = s3Client.getBucketAcl(bucketName);


            // (Optional) delete all grants.
            bucketAcl.getGrants().clear();

            // Add grant - owner.
            Grant grant0 = new Grant(
              new Canonic
alGrantee("852b113e7a2f25102679df27bb0ae12b3f85be6f290b936c4393484beExample"),

              Permission.FullControl);
            grantCollection.add(grant0);

            // Add grant using canonical user id.
            Grant grant1 = new Grant(
              new Canonic
alGrantee("d25639fbe9c19cd30a4c0f43fbf00e2d3f96400a9aa8dabfbbebe1906Example"),

              Permission.Write);
            grantCollection.add(grant1);

            // Grant LogDelivery group permission to write to the bucket.
            Grant grant3 = new Grant(GroupGrantee.LogDelivery,
                                Permission.Write);
            grantCollection.add(grant3);

           bucketAcl.getGrants().addAll(grantCollection);
```

```
            // Save (replace) ACL.
            s3Client.setBucketAcl(bucketName, bucketAcl);

        } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException, which" +
              " means your request made it " +
                    "to Amazon S3, but was rejected with an error response" +
                    " for some reason.");
            System.out.println("Error Message:    " + ase.getMessage());
            System.out.println("HTTP Status Code: " + ase.getStatusCode());
            System.out.println("AWS Error Code:   " + ase.getErrorCode());
            System.out.println("Error Type:       " + ase.getErrorType());
            System.out.println("Request ID:       " + ase.getRequestId());
        } catch (AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException, which means"+

              " the client encountered " +
                    "a serious internal problem while trying to " +
                    "communicate with S3, " +
                    "such as not being able to access the network.");
            System.out.println("Error Message: " + ace.getMessage());
        }
    }
}
```

AWS SDK for .NET を使用した ACL の管理

リソースの作成時の ACL の設定

リソース (バケットおよびオブジェクト) の作成時に、リクエストに許可のコレクションを指定して (「アクセスコントロールリスト (ACL) の概要 (p. 347)」を参照) 、アクセス許可を付与できます。各許可に、明示的に被付与者とアクセス許可を指定する S3Grant オブジェクトを作成します。

例えば、次の C# コードスニペットでは、オブジェクトをアップロードする PutObject リクエストを送信します。このリクエストでは、コードスニペットが 2 つの AWS アカウントと Amazon S3 AllUsers グループにアクセス許可を指定しています。PutObject 呼び出しには、リクエスト本文にオブジェクトデータが含まれ、リクエストヘッダーに ACL 許可が含まれます (「PUT Object」を参照) 。

```
AmazonS3 client;
string bucketName;
string keyName;

var grantee1 = new S3Grantee().WithCanonic
alUser("d25639fbe9c19cd30a4c0f43fbf00e2d3f96400a9aa8dabfbbebe1906Example",
"display-name");
var grantee2 = new S3Grantee().WithEmailAddress("user@email.com");
var grantee3 = new S3Grantee().WithURI("http://acs.amazonaws.com/groups/glob
al/AllUsers");

S3Grant grant1 = new S3Grant().WithGrantee(grantee1).WithPermission(S3Permis
sion.READ_ACP);
S3Grant grant2 = new S3Grant().WithGrantee(grantee2).WithPermission(S3Permis
sion.WRITE_ACP);
S3Grant grant3 = new S3Grant().WithGrantee(grantee3).WithPermission(S3Permis
sion.READ);
```

```
// 1. Simple object put.
PutObjectRequest request = new PutObjectRequest();
request.WithContentBody("Object data for simple put.")
    .WithBucketName(bucketName)
    .WithKey(keyName)
    .WithGrants(grant1, grant2, grant3);

S3Response response = client.PutObject(request);
```

オブジェクトのアップロードについては、「Amazon S3 オブジェクトの使用 (p. 107)」を参照してください。

前述のコードスニペットでは、S3Grant ごとに、被付与者とアクセス許可を明示的に識別します。または、リソースの作成時に、リクエストに既定 (定義済み) の ACL を指定できます (「既定 ACL (p. 350)」を参照) 。次の C# コードスニペットでは、オブジェクトを作成し、読み取りアクセス許可を Amazon S3 AuthenticatedUsers グループに付与する既定 ACL AuthenticatedRead をリクエストに指定します。

```
AmazonS3 client;
string bucketName;

PutObjectRequest request = new PutObjectRequest();
request.WithBucketName(bucketName)
       .WithCannedACL(S3CannedACL.AuthenticatedRead);

client.PutObject(request).Dispose();
```

基本の REST API については、「PUT Bucket」を参照してください。

既存のリソースでの ACL の更新

既存のオブジェクトまたはバケットで ACL を設定するには、AmazonS3.SetAcl を呼び出します。ACL 許可のリストを含む S3AccessControlList クラスのインスタンスを作成し、リストを SetAcl リクエストに含めます。次の C# コードスニペットでは、既存のオブジェクトに ACL を設定する SetACL リクエストを送信します。

```
var grantee1 = new S3Grantee().WithCanonic
alUser("d25639fbe9c19cd30a4c0f43fbf00e2d3f96400a9aa8dabfbbebe1906Example",
"display-name");
var grantee2 = new S3Grantee().WithEmailAddress("user@email.com");
var grantee3 = new S3Grantee().WithURI("http://acs.amazonaws.com/groups/glob
al/AllUsers");

S3Grant grant1 = new S3Grant().WithGrantee(grantee1).WithPermission(S3Permis
sion.READ_ACP);
S3Grant grant2 = new S3Grant().WithGrantee(grantee2).WithPermission(S3Permis
sion.WRITE_ACP);
S3Grant grant3 = new S3Grant().WithGrantee(grantee3).WithPermission(S3Permis
sion.READ);

S3AccessControlList aclList = new S3AccessControlList();
aclList.Grants.Add(grant1);
aclList.Grants.Add(grant2);
aclList.Grants.Add(grant3);
Owner aclOwner = new Owner()
  .WithId("852b113e7a2f25102679df27bb0ae12b3f85be6f290b936c4393484beexample")
```

```
  .WithDisplayName("owner-display-name");
aclList.Owner = aclOwner;

SetACLRequest req = new SetACLRequest();
req.WithBucketName(bucketName)
   .WithKey(keyName)
   .WithCannedACL(S3CannedACL.AuthenticatedRead);

SetACLResponse setACLResponse = client.SetACL(req);
```

Note

前述のコードスニペットでは、AmazonS3.GetAcl メソッドを使用して、オプションでまず既存の ACL を読み取り、それに新しい許可を追加し、変更した ACL をリソースに設定できます。

S3Grant オブジェクトを作成し、被付与者とアクセス許可を明示的に指定する代わりに、リクエストに既定 ACL を指定することもできます。次の C# コードスニペットでは、既存のオブジェクトに ACL を設定します。このリクエストでは、スニペットは規定 ACL AuthenticatedRead を指定し、読み取り許可を Amazon S3 Authenticated Users グループに付与します。

```
AmazonS3 client;
string bucketName;
string keyName;

SetACLRequest req = new SetACLRequest();
req.WithBucketName(bucketName)
   .WithKey(keyName)
   .WithCannedACL(S3CannedACL.AuthenticatedRead);

SetACLResponse setACLResponse = client.SetACL(req);
```

例

次の C# コード例は、次のタスクを実行します。

- バケットを作成します。このリクエストでは、log-delivery-write 既定 ACL を指定し、書き込み許可を LogDelivery Amazon S3 グループに付与します。
- バケットの ACL を読み取ります。
- 既存のアクセス許可をクリアし、新しいアクセス許可を ACL に追加します。
- SetACL リクエストを呼び出して、バケットに新しい ACL を追加します。

```
using System;
using System.Collections.Specialized;
using System.Configuration;
using Amazon.S3;
using Amazon.S3.Model;
using Amazon.S3.Util;

namespace s3.amazon.com.docsamples
{
  class S3Sample
  {
```

```
static string bucketName = "*** Provide bucket name ***";
static AmazonS3 client;

public static void Main(string[] args)
{
  PutBucketRequest request = new PutBucketRequest();

  NameValueCollection appConfig = ConfigurationManager.AppSettings;

  string accessKeyID = appConfig["AWSAccessKey"];
  string secretAccessKeyID = appConfig["AWSSecretKey"];
  try
  {
    using (client = Amazon.AWSClientFactory.CreateAmazonS3Client(
        accessKeyID, secretAccessKeyID))
    {
      // Add bucket (specify canned ACL).
      AddBucketWithCannedACL();

      // Get ACL on a bucket.
      GetBucketACL(bucketName);

      // Add (replace) ACL on a bucket.
      AddACLToExistingBucket();
    }
  }
  catch (AmazonS3Exception amazonS3Exception)
  {
    if (amazonS3Exception.ErrorCode != null &&
        (amazonS3Exception.ErrorCode.Equals("InvalidAccessKeyId")
        ||
        amazonS3Exception.ErrorCode.Equals("InvalidSecurity")))
    {
      Console.WriteLine("Check the provided AWS Credentials.");
      Console.WriteLine(
          "For service sign up go to http://aws.amazon.com/s3");
    }
    else
    {
      Console.WriteLine(
          "Error occurred. Message:'{0}' when writing an object"
          , amazonS3Exception.Message);
    }
  }
  catch (Exception e)
  {
    Console.WriteLine(e.Message);
  }
  Console.WriteLine("Press any key to continue...");
  Console.ReadKey();
}

static void AddBucketWithCannedACL()
{
   PutBucketRequest request = new PutBucketRequest();
  request.WithBucketName(bucketName)
      .WithBucketRegion(S3Region.US);
  // Add canned acl.
```

```
      request.AddHeaders(AmazonS3Util.CreateHeaderEntry(
                                    "x-amz-acl", "log-delivery-write"));
      client.PutBucket(request).Dispose();
    }

    static void GetBucketACL(string bucketName)
    {
      GetACLRequest request = new GetACLRequest();
      request.WithBucketName(bucketName);

      GetACLResponse response = client.GetACL(request);
      S3AccessControlList accessControlList =
                               response.AccessControlList;
      response.Dispose();
    }

    static void AddACLToExistingBucket()
    {
       GetACLRequest getRequest = new GetACLRequest();
      getRequest.BucketName = bucketName;

      GetACLResponse getResponse = client.GetACL(getRequest);
      S3AccessControlList acl = getResponse.AccessControlList;
      getResponse.Dispose();

      // Clear existing grants.
      acl.Grants.Clear();

      // Add grants. First, reset owner's full permission
      // (previous clear statement removed all permissions).
      S3Grantee grantee0 = new S3Grantee();
      grantee0.WithCanonicalUser(acl.Owner.Id, acl.Owner.DisplayName);
      acl.AddGrant(grantee0, S3Permission.FULL_CONTROL);

      // Grant permission using email.
      S3Grantee grantee1 = new S3Grantee();
      grantee1.EmailAddress = "user@email.com";
      acl.AddGrant(grantee1, S3Permission.WRITE_ACP);

      // Grant permission using Canonical ID.
      S3Grantee grantee2 = new S3Grantee();
      Amazon.S3.Model.Tuple<string, string> t =
          new Amazon.S3.Model.Tuple<string, string>
              ("d25639fbe9c19cd30a4c0f43fbf00e2d3f96400a9aa8dabfbbebe1906Ex
ample", "display-name");
      grantee2.CanonicalUser = t;
      acl.AddGrant(grantee2, S3Permission.WRITE);

      // Grant permission to the LogDelivery group.
      S3Grantee grantee3 = new S3Grantee();
      grantee3.URI = "http://acs.amazonaws.com/groups/s3/LogDelivery";
      acl.AddGrant(grantee3, S3Permission.WRITE);

      // Now update the ACL.
      SetACLRequest request = new SetACLRequest();
      request.BucketName = bucketName;
      request.ACL = acl;
```

```
      SetACLResponse response = client.SetACL(request);
      response.Dispose();

      // Get and print the updated ACL XML.
      Console.WriteLine(client.GetACL(new GetACLRequest()
          .WithBucketName(bucketName)).ResponseXml);
    }
  }
}
```

REST API を使用した ACL の管理

ACL の管理に関する REST API のサポートについては、「」*API リファレンスの以下のセクションを参照してください。*

- GET Bucket acl
- PUT Bucket acl
- GET Object acl
- PUT Object acl
- PUT Object
- PUT Bucket
- PUT Object – Copy
- Initiate Multipart Upload

ACL とバケットポリシーを使用するタイミング

ACL は大まかなアクセス許可モデルを提供し、単純にバケットまたはオブジェクトへのアクセス許可を付与します。一方、バケットポリシーは、付与するアクセス許可に対して詳細な制御を提供します。例えば、ユーザーが特定の IP アドレスからリクエストを送信した場合や、特定の日時以降にリクエストが到着した場合に、バケットまたはオブジェクトへのアクセスをユーザーに許可するポリシーを記述できます。ニーズに応じて、これらのいずれかまたは両方のアクセス許可モデルを使用できます。ただし、以下のように、ACL が最も適切となる特定の使用事例があります。

- バケットポリシーのみがある (オブジェクトポリシーがない) —バケット内の各オブジェクトにさまざまなアクセス許可を付与することが必要な場合があります。例えば、バケットへの書き込み許可を付与する場合、他のユーザーは、開発者がアクセス許可を持っていないオブジェクトをバケットに追加することができます。これらの新しいオブジェクトの所有者は、他のユーザーによるアクセスを有効化する前に、これらのオブジェクトに明示的にアクセス許可を付与する必要があります。
- バケットポリシーのサイズが 20 KB に制限されている—多数のオブジェクトとユーザーが存在する場合、バケットポリシーは 20 KB のサイズ制限に達する可能性があります。この場合は、追加の許可のために ACL の使用を検討する必要があります。

Amazon S3 は、ACL ポリシーとバケットポリシーの両方をサポートしています。既に ACL を使用している場合は、変更する必要はありません。より単純なシナリオでは、ACL で使用事例に合った適切なレベルのアクセス許可が提供される可能性もあります。例えば、少数の被付与者にアクセス許可を付与するときは、ACL の使用が適切となることがあります。

ACL とバケットポリシーを共に使用

バケットに ACL とバケットポリシーを割り当てると、Amazon S3 リソースに対するアカウントのアクセス許可を決定する際に既存の Amazon S3 ACL に加えて、バケットポリシーが評価されます。あるアカウントに ACL またはポリシーで指定されたリソースに対するアクセスが許可されている場合、そのアカウントはリクエストされたリソースにアクセスできます。

既存の Amazon S3 ACL では、許可により必ずバケットまたはオブジェクトにアクセスできるようになります。ポリシーを使用する場合、拒否は必ず許可よりも優先されます。

Note

バケットポリシーには、許可および拒否を優先付ける独自のルールセットがあります。詳細については、「Evaluation Logic (p. 491)」を参照してください。

既存の Amazon S3 ACL はポリシーに移行することができます。

ACL をバケットポリシーに移行するには

1	ポリシーを特定のユーザー、グループ、またはバケットと関連付けます。
2	ACL でユーザーまたはグループにアクセスが許可された Amazon S3 のリソースごとに、ポリシーに許可を追加します。

完了後は、ACL の代わりにポリシーでアカウントのアクセス許可を管理できるようになります。

アクションとアクセス許可の関係

ポリシーは特定のアクションを許可または拒否できます。ACL は特定のアクセス許可を付与できます。ポリシーで許可または拒否されるアクションは、ACL で付与されるアクセス許可の上位セットです。以下のセクションでは、アクションとアクセス許可の関係についてまとめています。

オブジェクトの ACL アクセス許可

- READ – オブジェクトの ACL で *READ* アクセス許可を付与すると、そのオブジェクトに対して *s3:GetObject*、*s3:GetObjectVersion*、および *s3:GetObjectTorrent* アクションを実行できるようになります。
- READ_ACP – オブジェクトの ACL で *READ_ACP* アクセス許可を付与すると、そのオブジェクトに対して *s3:GetObjectAcl* と *s3:GetObjectVersionAcl* アクションを実行できるようになります。
- WRITE_ACP – オブジェクトの ACL で *WRITE_ACP* アクセス許可を付与すると、そのオブジェクトに対して *s3:PutObjectAcl* と *s3:PutObjectVersionAcl* アクションを実行できるようになります。
- FULL_CONTROL – オブジェクトの ACL で *FULL_CONTROL* アクセス許可を付与することは、*READ*、*READ_ACP*、および *WRITE_ACP* アクセス許可を付与することと同じことです。

バケットの ACL アクセス許可

- READ – バケットの ACL で *READ* アクセス許可を付与すると、そのバケットに対して *s3:ListBucket*、*s3:ListBucketVersions*、および *s3:ListBucketMultipartUploads* アクションを実行できるようになります。
- WRITE – バケットの ACL で *WRITE* アクセス許可を付与すると、そのバケット内の任意のオブジェクトに対して *s3:PutObject* と *s3:DeleteObject* アクションを実行できるようになります。さらに、被付与者がバケット所有者であるときは、バケットの ACL で *WRITE* アクセス許可を付与する

と、そのバケット内の任意のバージョンに対して *s3:DeleteObjectVersion* アクションを実行できるようになります。

- READ_ACP – バケットの ACL で *READ_ACP* アクセス許可を付与すると、そのバケットに対して *s3:GetBucketAcl* アクションを実行できるようになります。
- WRITE_ACP – バケットの ACL で *WRITE_ACP permission* を付与すると、そのバケットに対して *s3:PutBucketAcl* アクションを実行できるようになります。
- FULL_CONTROL – バケットの ACL で *FULL_CONTROL* アクセス許可を付与することは、*READ*、*WRITE*、*READ_ACP*、および *WRITE_ACP* アクセス許可を付与することと同じです。

ポリシーで許可または拒否できるアクションの詳細については、「バケットポリシーの記述 (p. 329)」を参照してください。

クエリ文字列認証の使用

クエリ文字列認証は、通常は認証を必要とするリソースに対する HTTP またはブラウザアクセスを提供するのに役立ちます。クエリ文字列の署名は、リクエストをセキュリティ保護します。クエリ文字列認証のリクエストでは、有効期限が必要となります。任意の有効期限を エポックまたは UNIX 時刻 (1970 年 1 月 1 日以降の秒数) で指定できます。

クエリ文字列認証を使用する

1	クエリを作成します。
2	クエリの有効期限を指定します。
3	自分の署名で署名します。
4	データを HTTP リクエストに配置します。
5	リクエストをユーザーに配信するか、またはウェブページに埋め込みます。

例えば、クエリ URL とは次の例のようなものです。

```
http://quotes.s3.amazonaws.com/nelson?AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE&Ex
pires=1177363698&Signature=vjSAMPLENmGa%2ByT272YEAiv4%3D
```

リクエストに署名する方法の詳細については、「クエリ文字列による代替リクエスト認証 (p. 61)」を参照してください。. AWS SDK を使用して署名付きの URL を生成する例については、「他ユーザーとのオブジェクトの共有 (p. 163)」を参照してください。

データ保護

Topics

Amazon S3 は、ミッションクリティカルで重要なデータストレージのために設計された、極めて堅牢なストレージインフラストラクチャです。オブジェクトは冗長化のため、同一の Amazon S3 リージョン内の複数施設に分散した複数のデバイスに保存されます。データの耐久性を高めるため、Amazon S3 の PUTオペレーションおよび `PUT Object copy` オペレーションは、SUCCESS を返す前に、複数の施設で同期をとりながらデータを保存します。保存したオブジェクトからオブジェクトの冗長性が失われると、Amazon S3 によりすばやく検出、修復され、オブジェクトの耐久性が維持されます。

Amazon S3 は、チェックサムを用いて、保存されているデータの完全性を定期的に検証しています。Amazon S3 で検出されたデータの破損は、冗長データを使用して修復されます。さらに、Amazon S3 は、ネットワークの全トラフィックに対してチェックサムを計算し、データの保存または取得時のデータパケットの損傷を検出しています。

Amazon S3 の標準ストレージは以下のようになっています。

- 「Amazon S3 サービスレベル利用規約」で保証されています。
- 年間 99.999999999% のオブジェクトの耐久性と、99.99% のオブジェクトの可用性を提供するよう設計されています。
- 2 つの施設で同時にデータ喪失が発生しないよう設計されています。

Amazon S3 は、バージョニングを使用してデータ保護を強化しています。バージョニングを使用して、Amazon S3 バケットに保存されたあらゆるオブジェクトのあらゆるバージョンを保存、取得、復元することができます。バージョニングを使用すれば、意図せぬユーザーアクションからもアプリケーション障害方からも、簡単に回復することができます。デフォルトでは、リクエストは最も新しく書き込まれたバージョンを取得します。リクエストでオブジェクトのバージョンを指定することにより、古いバージョンのオブジェクトを取得できます。

データ暗号化の使用

Topics

暗号化によって、Amazon S3 のバケットに保存するオブジェクトデータのセキュリティを強化できます。クライアント側でデータを暗号化し、暗号化したデータを Amazon S3 にアップロードすることができます。この場合、暗号化プロセス、暗号化キー、関連ツールはお客様が管理してください。必要に応じ、オプションでサーバー側の暗号化機能を使用できます。この場合、Amazon S3 がオブジェクトデータを暗号化してからデータセンター内のディスクに保存し、オブジェクトをダウンロードするときに暗号化を解除します。そのため、お客様による暗号化、暗号化キー、関連ツールの管理作業は不要となります。

サーバー側の暗号化の使用

Topics

サーバー側の暗号化は、保管時のデータ暗号化に関するものです。つまり、Amazon S3 は、データセンターに書き込まれたデータを暗号化し、お客様がデータにアクセスするときに復号化します。リクエストが認証され、お客様がアクセス許可を持っていれば、オブジェクトが暗号化されているかどうかに関係なく同じ方法でアクセスできます。暗号化と復号化の管理は Amazon S3 が行います。例えば、署名付き URL を使用してオブジェクトを共有する場合、その署名付き URL は、オブジェクトが暗号化されているかどうかに関係なく同じように動作します。

クライアント側の暗号化の場合、データの暗号化/復号化、暗号化キー、および関連ツールを管理するのはお客様です。サーバー側の暗号化は、クライアント側の暗号化に代わる手段です。Amazon S3 がデータの暗号化を管理するので、お客様は暗号化や暗号化キーの管理作業をする必要がありません。

Amazon S3 のサーバー側の暗号化では、強力な多要素暗号化を採用しています。各オブジェクトは一意のキーで暗号化されます。さらにセキュリティを強化するために、キー自体が、定期的に更新されるマスターキーで暗号化されます。Amazon S3 のサーバー側の暗号化では、最強のブロック暗号の一つである、256ビットの高度暗号化規格 (AES-256) を使用してデータを暗号化します。

データの暗号化はオブジェクトレベルで指定できます。オブジェクトをアップロードするときに、オブジェクトデータを暗号化して保存するかどうかをリクエストで明示的に指定できます。サーバー側の暗号化はオプションです。バケットには、暗号化したオブジェクトと暗号化していないオブジェクトを混在して格納することができます。バケットに保存するすべてのオブジェクトに対してサーバー側の暗号化を必要とする場合は、 Amazon S3 のバケットポリシーを使用できます。例えば、次のバケットポリシーは、サーバー側の暗号化を要求する `x-amz-server-side-encryption` ヘッダーがリクエストに含まれない場合に、すべてのユーザーに対してオブジェクト (`s3:PutObject`) のアップロード許可を拒否します。

```
{
   "Version":"2012-10-17",
   "Id":"PutObjPolicy",
   "Statement":[{
         "Sid":"DenyUnEncryptedObjectUploads",
         "Effect":"Deny",
```

```
         "Principal":{
            "AWS":"*"
         },
         "Action":"s3:PutObject",
         "Resource":"arn:aws:s3:::YourBucket/*",
         "Condition":{
            "StringNotEquals":{
               "s3:x-amz-server-side-encryption":"AES256"
            }
         }
      }
   ]
}
```

サーバー側の暗号化では、オブジェクトデータのみが暗号化されます。オブジェクトメタデータは暗号化されません。

サーバー側の暗号化での API サポート

オブジェクト作成の REST API (「REST API を使用したサーバー側の暗号化の指定 (p. 373)」を参照) は、`x-amz-server-side-encryption` というリクエストヘッダーを提供します。これを使用してサーバー側の暗号化を要求できます。

AWS SDK にも、サーバー側の暗号化を要求するためのラッパー API があります。また、AWS マネジメントコンソールを使用して、オブジェクトをアップロードしてサーバー側の暗号化を要求することもできます。

AWS SDK for Java を使用したサーバー側の暗号化の指定

AWS SDK for Java を使用してオブジェクトをアップロードするときに、`PutObjectRequest` の `ObjectMetadata` プロパティを使用してリクエストの `x-amz-server-side-encryption` ヘッダーを設定できます (「REST API を使用したサーバー側の暗号化の指定 (p. 373)」を参照)。次の Java コード例に示すように、Amazon S3 クライアントの `PutObject` メソッドを呼び出すと、Amazon S3 がデータを暗号化して保存します。

```
File file = new File(uploadFileName);
PutObjectRequest putRequest = new PutObjectRequest(
                                      bucketName, keyName, file);

// Request server-side encryption.
ObjectMetadata objectMetadata = new ObjectMetadata();
objectMetadata.setServerSideEncryption(
                     ObjectMetadata.AES_256_SERVER_SIDE_ENCRYPTION);
putRequest.setMetadata(objectMetadata);

PutObjectResult response = s3client.putObject(putRequest);
System.out.println("Uploaded object encryption status is " +
                   response.getServerSideEncryption());
```

レスポンスでは、オブジェクトデータの暗号化に使用した暗号化アルゴリズムが返されます。これは `getServerSideEncryption` メソッドで確認できます。

オブジェクトのアップロード方法を示す作業サンプルについては、「AWS SDK for Java を使用したオブジェクトのアップロード (p. 169)」を参照してください。サーバー側の暗号化の場合、リクエストに `ObjectMetadata` プロパティを追加します。

マルチパートアップロード API を使用して大容量のオブジェクトをアップロードするときに、アップロード中のオブジェクトに対してサーバー側の暗号化を要求できます。

- 低レベルマルチパートアップロード API を使用して大容量オブジェクトをアップロードする場合（「ファイルのアップロード (p. 184)」を参照）は、マルチパートアップロードの開始時にサーバー側の暗号化を指定できます。つまり、`InitiateMultipartUploadRequest.setObjectMetadata` メソッドを呼び出して `ObjectMetadata` プロパティを追加します。
- 高レベルマルチパートアップロード API を使用する場合（「高レベル Java API を使用したマルチパートアップロード (p. 179)」を参照）は、`TransferManager` クラスのメソッドでオブジェクトをアップロードします。パラメータとして `ObjectMetadata` を受け取るどのアップロードメソッドを呼び出してもかまいません。

使用された暗号化アルゴリズムの確認

次の Java コード例に示すように、オブジェクトメタデータを取得して、既存のオブジェクトの暗号化状態を確認できます。

```
GetObjectMetadataRequest request2 =
                new GetObjectMetadataRequest(bucketName, keyName);

ObjectMetadata metadata = s3client.getObjectMetadata(request2);

System.out.println("Encryption algorithm used: " +
            metadata.getServerSideEncryption());
```

Amazon S3 に保存されたオブジェクトにサーバー側の暗号化が使用されていない場合、メソッドは null を返します。

既存のオブジェクトのサーバー側暗号化の変更（コピーオペレーション）

既存のオブジェクトの暗号化状態を変更するには、オブジェクトのコピーを作成し、コピー元オブジェクトを削除します。デフォルトでは、明示的にサーバー側の暗号化を要求しない限り、コピー API は対象を暗号化しません。次の Java コード例に示すように、`CopyObjectRequest` で `ObjectMetadata` プロパティを使用してサーバー側の暗号化を指定することで、対象オブジェクトの暗号化を要求できます。

```
CopyObjectRequest copyObjRequest = new CopyObjectRequest(
sourceBucket, sourceKey, targetBucket, targetKey);

// Request server-side encryption.
ObjectMetadata objectMetadata = new ObjectMetadata();
objectMetadata.setServerSideEncryption(
          ObjectMetadata.AES_256_SERVER_SIDE_ENCRYPTION);

copyObjRequest.setNewObjectMetadata(objectMetadata);

CopyObjectResult response =  s3client.copyObject(copyObjRequest);
System.out.println("Copied object encryption status is " +
                  response.getServerSideEncryption());
```

オブジェクトのコピー方法を示す作業サンプルについては、「AWS SDK for Java を使用したオブジェクトのコピー (p. 223)」を参照してください。前述のコード例に示すように、`CopyObjectRequest` オブジェクトでサーバー側の暗号化を指定できます。

AWS SDK for .NET を使用したサーバー側の暗号化の指定

AWS SDK for .NET を使用してオブジェクトをアップロードするときに、PutObjectRequest の WithServerSideEncryptionMethod プロパティを使用してリクエストの x-amz-server-side-encryption ヘッダーを設定できます (「REST API を使用したサーバー側の暗号化の指定 (p. 373)」を参照) 。次の C# コード例に示すように、Amazon S3 クライアントの PutObject メソッドを呼び出すと、Amazon S3 はデータを暗号化して保存します。

```
static AmazonS3 client;
client = new AmazonS3Client(accessKeyID, secretAccessKeyID);

PutObjectRequest request = new PutObjectRequest();
request.WithContentBody("Object data for simple put.")
    .WithBucketName(bucketName)
    .WithKey(keyName)
    .WithServerSideEncryptionMethod(ServerSideEncryptionMethod.AES256);

S3Response response = client.PutObject(request);

// Check the response header to determine if the object is encrypted.
ServerSideEncryptionMethod destinationObjectEncryptionStatus = response.Server
SideEncryptionMethod;
```

レスポンスでは、オブジェクトデータの暗号化に使用した暗号化アルゴリズムが返されます。これは ServerSideEncryptionMethod プロパティで確認できます。

オブジェクトのアップロード方法を示す作業サンプルについては、「AWS SDK for .NET を使用したオブジェクトのアップロード (p. 170)」を参照してください。サーバー側の暗号化の場合、WithServerSideEncryptionMethod メソッドを呼び出して ServerSideEncryptionMethod プロパティを設定します。

マルチパートアップロード API を使用して大容量のオブジェクトをアップロードするときに、アップロード中のオブジェクトに対してサーバー側の暗号化を指定できます。

- 低レベルマルチパートアップロード API を使用して大容量オブジェクトをアップロードする場合 (「低レベル .NET API を使用したマルチパートアップロード (p. 199)」を参照) は、InitiateMultipartUpload リクエストでサーバー側の暗号化を指定できます。つまり、WithServerSideEncryptionMethod メソッドを呼び出して ServerSideEncryptionMethod プロパティを InitiateMultipartUploadRequest に設定します。
- 高レベルマルチパートアップロード API を使用する場合 (「高レベル .NET API を使用したマルチパートアップロード (p. 189)」を参照) は、TransferUtility クラスのメソッド (Upload および UploadDirectory) でオブジェクトをアップロードします。この場合、TransferUtilityUploadRequest オブジェクトと TransferUtilityUploadDirectoryRequest オブジェクトを使用してサーバー側の暗号化を要求できます。

使用された暗号化アルゴリズムの確認

次の C# コード例に示すように、オブジェクトメタデータを取得して、既存のオブジェクトの暗号化状態を確認できます。

```
AmazonS3 client;
client = new AmazonS3Client(accessKeyID, secretAccessKeyID);

ServerSideEncryptionMethod objectEncryption;
```

```
GetObjectMetadataRequest metadataRequest = new GetObjectMetadataRequest()
                                                .WithBucketName(bucketName)
                                                .WithKey(keyName);

objectEncryption = client.GetObjectMetadata(metadataRequest)
                                    .ServerSideEncryptionMethod;
```

暗号化アルゴリズムは列挙型で指定します。格納されたオブジェクトが暗号化されていない場合 (デフォルトの動作) 、オブジェクトの ServerSideEncryptionMethod プロパティはデフォルトで None に設定されます。

既存のオブジェクトのサーバー側暗号化の変更 (コピーオペレーション)

既存のオブジェクトの暗号化状態を変更するには、オブジェクトのコピーを作成し、コピー元オブジェクトを削除します。デフォルトでは、明示的にサーバー側の暗号化を要求しない限り、コピー API は対象オブジェクトを暗号化しません。次の C# コード例では、オブジェクトのコピーを作成します。リクエストで明示的に対象オブジェクトのサーバー側の暗号化を指定します。

```
AmazonS3 client;
client = new AmazonS3Client(accessKeyID, secretAccessKeyID);

CopyObjectResponse response = client.CopyObject(new CopyObjectRequest()
            .WithSourceBucket(sourceBucketName)
            .WithSourceKey(sourceObjetKey)
            .WithDestinationBucket(targetBucketName)
            .WithDestinationKey(targetObjectKey)
            .WithServerSideEncryptionMethod(ServerSideEncryptionMethod.AES256)
);
// Check the response header to determine if the object is encrypted.
ServerSideEncryptionMethod destinationObjectEncryptionStatus = response.Server
SideEncryptionMethod;
```

オブジェクトのコピー方法を示す作業サンプルについては、「AWS SDK for .NET を使用したオブジェクトのコピー (p. 224)」を参照してください。前述のコード例に示すように、CopyObjectRequest オブジェクトでサーバー側の暗号化を指定できます。

AWS SDK for PHP を使用したサーバー側の暗号化の指定

このトピックでは、AWS SDK for PHP のクラスを使用して、Amazon S3 にアップロードするオブジェクトにサーバー側の暗号化を追加する手順を示します。

Note

このトピックでは、既に AWS SDK for PHP の使用と PHP サンプルの実行 (p. 479) の説明が実行されていて、AWS SDK for PHP が正しくインストールされていることを前提としています。

オブジェクトを Amazon S3 にアップロードする際には、 Aws\S3\S3Client::putObject() メソッドを使用できます。 オブジェクトのアップロード方法を示す作業サンプルについては、「AWS SDK for PHP を使用したオブジェクトのアップロード (p. 174)」を参照してください。

アップロードリクエストにそのリクエストの x-amz-server-side-encryption ヘッダー (「REST API を使用したサーバー側の暗号化の指定 (p. 373)」を参照) を追加する場合は、次の PHP コード例に示すように、値 AES256 を使用して array パラメータの ServerSideEncryption キーを指定します。

```
use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';
// $filepath should be absolute path to a file on disk
$filepath = '*** Your File Path ***';

// Instantiate the client.
$s3 = S3Client::factory(array(
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));

// Upload a file with server-side encryption.
$result = $s3->putObject(array(
    'Bucket'               => $bucket,
    'Key'                  => $keyname,
    'SourceFile'           => $filepath,
    'ServerSideEncryption' => 'AES256',
));
```

レスポンスでは、x-amz-server-side-encryption ヘッダーが返されます。このヘッダーには、オブジェクトデータの暗号化に使用された暗号化アルゴリズムの値が設定されています。

マルチパートアップロード API を使用して大容量のオブジェクトをアップロードするときに、アップロード中のオブジェクトに対してサーバー側の暗号化を指定できます。

- 低レベルマルチパートアップロード API を使用する場合 (「AWS SDK for PHP の低レベル API を使用したマルチパートアップロード (p. 208)」を参照) は、Aws\S3\S3Client::createMultipartUpload() メソッドを呼び出すときにサーバー側の暗号化を指定できます。リクエストにそのリクエストの x-amz-server-side-encryption ヘッダーを追加する場合は、値 AES256 を使用して array パラメータの ServerSideEncryption キーを指定します。
- 高レベルマルチパートアップロードを使用する場合は、setOption('ServerSideEncryption','AES256') などの Aws\S3\Model\MultipartUpload\UploadBuilder:setOption() メソッドを使用してサーバー側の暗号化を指定できます。高レベルの UploadBuilder と共に setOption() メソッドを使用する例については、「AWS SDK for PHP の高レベルの抽象化を使用したマルチパートアップロード (p. 205)」を参照してください。

使用された暗号化アルゴリズムの確認

次の PHP コード例に示すように、Aws\S3\S3Client::headObject() メソッドを呼び出し、オブジェクトメタデータを取得して、既存のオブジェクトの暗号化状態を確認できます。

```
use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

// Instantiate the client.
$s3 = S3Client::factory(array(
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));
```

```
// Check which server-side encryption algorithm is used.
$result = $s3->headObject(array(
    'Bucket' => $bucket,
    'Key'    => $keyname,
));
echo $result['ServerSideEncryption'];
```

既存のオブジェクトのサーバー側暗号化の変更 (コピーオペレーション)

Aws\S3\S3Client::copyObject() メソッドを使用してオブジェクトのコピーを作成し、コピー元のオブジェクトを削除することで、既存のオブジェクトの暗号化状態を変更できます。デフォルトでは、コピー先は `copyObject()` によって暗号化されません。ただし、値 AES256 を指定して `array` パラメータの `ServerSideEncryption` キーを使用し、コピー先オブジェクトのサーバー側の暗号化を明示的にリクエストする場合は異なります。以下の PHP コード例では、オブジェクトのコピーを作成し、コピー先のオブジェクトにサーバー側の暗号化を追加しています。

```
use Aws\S3\S3Client;

$sourceBucket = '*** Your Source Bucket Name ***';
$sourceKeyname = '*** Your Source Object Key ***';

$targetBucket = '*** Your Target Bucket Name ***';
$targetKeyname = '*** Your Target Object Key ***';

// Instantiate the client.
$s3 = S3Client::factory(array(
    'key'    => '*** Your AWS Access Key ID ***',
    'secret' => '*** Your AWS Secret Key ***'
));

// Copy an object and add server-side encryption.
$result = $s3->copyObject(array(
    'Bucket'               => $targetBucket,
    'Key'                  => $targetKeyname,
    'CopySource'           => "{$sourceBucket}/{$sourceKeyname}",
    'ServerSideEncryption' => 'AES256',
));
```

オブジェクトのコピー方法を示す作業サンプルについては、「AWS SDK for PHP を使用したオブジェクトのコピー (p. 227)」を参照してください。

関連リソース

- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client クラス」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::factory() メソッド」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::copyObject() メソッド」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::createMultipartUpload() メソッド」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::headObject() メソッド」
- 「AWS SDK for PHP for Amazon S3 の Aws\S3\S3Client::putObject() メソッド」
- 「Aws\S3\Model\MultipartUpload\UploadBuilder:setOption() メソッド」
- 「AWS SDK for PHP – Amazon S3」
- 「AWS SDK for PHP」のドキュメント

AWS SDK for Ruby を使用したサーバー側の暗号化の指定

AWS SDK for Ruby を使用してオブジェクトをアップロードするときに、`#write` インスタンスメソッドのオプションハッシュ `server_side_encryption` を指定することで、保管するオブジェクトを暗号化するよう指定できます。オブジェクトは、読み戻すときに自動的に復号化されます。

次の Ruby スクリプト例は、Amazon S3 にアップロードするファイルを保管時に暗号化するよう指定する方法を示しています。

```
# Upload a file and set server-side encryption.
key_name = File.basename(file_name)
s3.buckets[bucket_name].objects[key_name].write(:file => file_name, :serv
er_side_encryption => :aes256)
```

オブジェクトのアップロード方法を示す作業サンプルについては、「AWS SDK for Ruby を使用したオブジェクトのアップロード (p. 176)」を参照してください。

使用された暗号化アルゴリズムの確認

保管中のオブジェクトデータに使用された暗号化アルゴリズムを確認するには、`S3Object` インスタンスの `#server_side_encryption` メソッドを使用します。次のコード例は、既存のオブジェクトの暗号化状態を確認する方法を示しています。

```
# Determine server-side encryption of an object.
enc = s3.buckets[bucket_name].objects[key_name].server_side_encryption
enc_state = (enc != nil) ? enc : "not set"
puts "Encryption of #{key_name} is #{enc_state}."
```

Amazon S3 に保存されたオブジェクトにサーバー側の暗号化が使用されていない場合、メソッドは null を返します。

既存のオブジェクトのサーバー側暗号化の変更 (コピーオペレーション)

既存のオブジェクトの暗号化状態を変更するには、オブジェクトのコピーを作成し、コピー元オブジェクトを削除します。Ruby API の `S3Object` クラスの `#copy_from` メソッドと `#copy_to` メソッドを使用して、オブジェクトをコピーできます。デフォルトでは、明示的にサーバー側の暗号化を要求しない限り、コピーメソッドは対象を暗号化しません。次の Ruby コード例に示すように、オプションハッシュ引数に `server_side_encryption` を指定することで、対象オブジェクトの暗号化を要求できます。コード例は、`#copy_to` メソッドの使い方を示しています。

```
s3 = AWS::S3.new

# Upload a file and set server-side encryption.
bucket1 = s3.buckets[source_bucket]
bucket2 = s3.buckets[target_bucket]
obj1 = bucket1.objects[source_key]
obj2 = bucket2.objects[target_key]

obj1.copy_to(obj2, :server_side_encryption => :aes256)
```

オブジェクトのコピー方法を示す作業サンプルについては、「AWS SDK for Ruby を使用したオブジェクトのコピー (p. 231)」を参照してください。

REST API を使用したサーバー側の暗号化の指定

オブジェクト作成時、つまり、新しいオブジェクトをアップロードするときや既存オブジェクトのコピーを作成するときに、リクエストに `x-amz-server-side-encryption` ヘッダーを追加することで、データを暗号化するよう Amazon S3 に指定することができます。ヘッダーの値を、Amazon S3 がサポートする暗号化アルゴリズム `AES256` に設定します。Amazon S3 によりレスポンスヘッダー `x-amz-server-side-encryption` が返されるため、サーバー側の暗号化を使用してオブジェクトが保存されたことを確認できます。

次の REST アップロード API は、`x-amz-server-side-encryption` リクエストヘッダーを受け付けます。

- PUT Object
- PUT Object – Copy
- POST Object
- Initiate Multipart Upload

マルチパートアップロード API を使用して大容量のオブジェクトをアップロードするときに、`x-amz-server-side-encryption` ヘッダーを Initiate Multipart Upload リクエストに追加することでサーバー側の暗号化を指定することができます。既存のオブジェクトをコピーするときは、コピー元オブジェクトが暗号化されているかどうかに関係なく、明示的にサーバー側の暗号化を要求しない限り、コピー先オブジェクトは暗号化されません。

次の REST API のレスポンスヘッダーは、オブジェクトがサーバー側の暗号化を使用して保存されているときに `x-amz-server-side-encryption` ヘッダーを返します。

- PUT Object
- PUT Object – Copy
- POST Object
- Initiate Multipart Upload
- Upload Part
- Upload Part – Copy
- Complete Multipart Upload
- Get Object
- Head Object

AWS マネジメントコンソールを使用したサーバー側の暗号化の指定

AWS マネジメントコンソールを使用してオブジェクトをアップロードするときに、サーバー側の暗号化を指定できます。オブジェクトのアップロード方法を示す作業サンプルについては、「Uploading Objects into Amazon S3」を参照してください。

AWS マネジメントコンソールでオブジェクトをコピーする場合、オブジェクトは現状のままコピーされます。つまり、コピー元が暗号化されていると、コピー先オブジェクトも暗号化されます。コンソールを使用してオブジェクトをコピーする方法の例については、「Copying an Object」を参照してください。コンソールで、1 つ以上のオブジェクトのプロパティを更新することもできます。例えば、1 つ以上のオブジェクトを選択して、サーバー側の暗号化を選択することができます。

クライアント側の暗号の使用

概要

Amazon S3 のサーバー側の暗号化を使用するのではなく、Amazon S3 に送信する前にデータを暗号化することもできます。オブジェクトデータを Amazon S3 にアップロードする前にクライアント側で暗号化するライブラリを独自に構築できます。オプションで、AWS SDK for Java を使用して、データを Amazon S3 にアップロードする前に自動的に暗号化することもできます。

現時点では、AWS SDK for Java のみがクライアント側での暗号化をサポートしています。

Important

プライベート暗号化キーと暗号化されていないデータは AWS には送信されません。したがって、暗号化キーは安全に管理することが重要です。暗号化キーを紛失すると、データを復号化できなくなります。

AWS Java SDK エンベロープ暗号化

AWS SDK for Java では、*エンベロープ暗号化*というプロセスを使用します。エンベロープ暗号化では、暗号化キーを Amazon S3 暗号化クライアントに提供すると、それ以降のプロセスはクライアントで実行されます。このプロセスの動作は次のようになります。

1. Amazon S3 暗号化クライアントは 1 回限定使用の対称キー（*エンベロープ対称キー*）を生成し、これを使用してデータを暗号化します。
2. クライアントは、プライベート暗号化キーを使用してエンベロープ対称キーを暗号化します。
3. その後、クライアントは暗号化したエンベロープキーと暗号化したデータを Amazon S3 にアップロードします。デフォルトで、暗号化したエンベロープキーはオブジェクトメタデータ（`x-amz-meta-x-amz-key`）として Amazon S3 に格納されます。

クライアント側で暗号化したデータを Amazon S3 から取得して復号化する場合は、上記の暗号化の逆の流れになります。

1. クライアントは、暗号化されたデータと暗号化されたエンベロープキーを Amazon S3 から取得します。
2. その後、プライベート暗号化キーを使用して、暗号化されたエンベロープキーを復号化します。
3. クライアントは、エンベロープキーを使用してデータを復号化します。

エンベロープ暗号化プロセスでクライアントが使用するプライベート暗号化キーには、非対称キーペア（パブリックキーとプライベートキーのペア）を使用することも、対称キーを使用することもできます。エンベロープ暗号化プロセスでのセキュリティレベルを強化するために、非対称キーの使用をお勧めします。

エンベロープ暗号化プロセスの詳細については、「Client-Side Data Encryption with the AWS SDK for Java and Amazon S3」の記事を参照してください。

AWS SDK for Java を使用したクライアント側の暗号化の指定

Amazon S3 にアップロードするオブジェクトをクライアント側で暗号化/復号化するには、AWS SDK for Java の `AmazonS3EncryptionClient` クラスを使用します。このクラスは、暗号化されていない `AmazonS3Client` クラスと同様の機能を提供するため、既存のコードをアップグレードして暗号化クライアントを使用するのも簡単です。

AmazonS3EncryptionClient クラスは、EncryptionMaterials インスタンスで初期化する必要があります。このインスタンスは、使用する暗号化キー（非対称キーと対称キーのどちらも可）を記述します。

Note

初めて暗号化 API を使用するときに暗号の暗号化エラーメッセージが表示される場合、使用する JDK のバージョンに、暗号化/復号化変換に使用する最大キー長を 128 ビットに制限する Java Cryptography Extension（JCE）管轄ポリシーファイルが含まれている可能性があります。AWS SDK では最大 256 ビットのキー長を必要とします。最大キー長を確認するには、javax.crypto.Cipher クラスの getMaxAllowedKeyLength メソッドを使用します。キー長の制限を削除するには、Java SE ダウンロードページから Java Cryptography Extension（JCE）無制限強度の管轄ポリシーファイルをインストールします。

クライアント側で暗号化したオブジェクトのアップロードと取得

1	セキュリティ認証情報を指定して、新しい AWSCredentials オブジェクトを作成します。
2	非対称キーペアを指定して、新しい EncryptionMaterials オブジェクトを作成します。
3	AWSCredentials オブジェクトと EncryptionMaterials オブジェクトを使用して、新しい AmazonS3EncryptionClient インスタンスを作成します。
4	putObject メソッドを使用してオブジェクトをアップロードし、getObject メソッドを使用してオブジェクトを取得します。必要な暗号化と復号化はクライアントが実行します。

以下の Java コード例は、前述のタスクの例です。

```
// Specify the asymmetric key pair to use.
KeyPairGenerator keyGenerator = KeyPairGenerator.getInstance("RSA");
keyGenerator.initialize(1024, new SecureRandom());
KeyPair myKeyPair = keyGenerator.generateKeyPair();

// Construct an instance of AmazonS3EncryptionClient.
AWSCredentials credentials = new BasicAWSCredentials(myAccessKeyId, mySecretKey);
EncryptionMaterials encryptionMaterials = new EncryptionMaterials(myKeyPair);
AmazonS3EncryptionClient encryptedS3client =
    new AmazonS3EncryptionClient(credentials, encryptionMaterials);

// Upload the object.
encryptedS3client.putObject(new PutObjectRequest(bucketName, key, createSample
File()));

// Retrieve the object.
S3Object downloadedObject = encryptedS3client.getObject(bucketName, key);
```

AWS SDK for Java を使用したクライアント側の暗号化の例

このセクションでは、AWS SDK を使用してクライアント側の暗号化を実装する 3 つの例を示します。最初の例では、256 ビットの Advanced Encryption Standard（AES）シークレットキーを生成します。このキーはローカルに保存され、ファイルから読み込まれます。2 番めの例では、クライアント側の暗号化で 256 ビットの AES キーを対称キーとして使用して、ディレクトリを Amazon S3 にアップロードする方法を示します。（このキーは最初の例から使用できます。）3 番めの例では、キーペア（パブ

リックキーとプライベートキー) を作成し、それを使用して、クライアント側で暗号化されたアップロードを Amazon S3 に行う方法を示します。このキーペアは、プライベート暗号化キーとして使用される対称キーペアの例です。

例: 256 ビットの AES シークレットキーを作成する

この例では、暗号化されたアップロードを Amazon S3 に行うためのプライベート対称キーとして、シークレットキーを作成する方法を示します。この例では、ローカルファイルとの間でキーを読み書きします。典型的なユースケースでは、キーを生成するためにこのプログラムを 1 度だけ実行します。続いて、キーを安全な場所に保存します。

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.X509EncodedKeySpec;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import org.apache.commons.codec.binary.Base64;

public class SymAES256KeyGenerate {

    private static String keyDir = "***specify local directory ***";
    private static String keyName = "symmetric.key";

    public static void main(String[] args) throws IOException, NoSuchAlgorith
mException, InvalidKeyException, InvalidKeySpecException {

        //Generate symmetric 256 AES key.
        KeyGenerator symKeyGenerator = KeyGenerator.getInstance("AES");
        symKeyGenerator.init(256);
        SecretKey symKey = symKeyGenerator.generateKey();
        System.out.println("Symmetric key saved  (base 64): " + new
String(Base64.encodeBase64(symKey.getEncoded())));

        //Save key.
        saveSymmetricKey(keyDir, symKey);

        //Load key.
        SecretKey symKeyLoaded = loadSymmetricAESKey(keyDir, "AES");

        //Compare with what we saved.
        System.out.println("Symmetric key loaded (base 64): " + new
String(Base64.encodeBase64(symKeyLoaded.getEncoded())));
    }

    public static void saveSymmetricKey(String path, SecretKey secretKey)
        throws IOException {
        X509EncodedKeySpec x509EncodedKeySpec = new X509EncodedKeySpec(
                secretKey.getEncoded());
        FileOutputStream keyfos = new FileOutputStream(path + "/" + keyName);
        keyfos.write(x509EncodedKeySpec.getEncoded());
        keyfos.close();
    }
```

```
    public static SecretKey loadSymmetricAESKey(String path, String algorithm)

        throws IOException, NoSuchAlgorithmException, InvalidKeySpecException,
 InvalidKeyException{
        //Read private key from file.
        FileInputStream keyfis = new FileInputStream(path + "/" + keyName);
        byte[] encodedPrivateKey = new byte[keyfis.available()];
        keyfis.read(encodedPrivateKey);
        keyfis.close();

        //Generate secret key.
        return new SecretKeySpec(encodedPrivateKey, "AES");
    }
    }
```

例: 対称キーとして使用してディレクトリを Amazon S3 にアップロードする

この例では、クライアント側の暗号化で 256 ビットの AES 対称キーをプライベート暗号化キーとして使用して、ディレクトリおよびその中の全ファイルを Amazon S3 にアップロードする方法を示します。例中のコードでは、プロパティファイル (.properties) を使用して、暗号化およびアップロードの設定情報を指定しています。この例をユーザー自身のソースコードに組み込むには、使用環境を反映するようプロパティファイル内の値を変更する必要があります。

プロパティファイルは、次に示すように 5 つの設定パラメータを取ります。このプロパティファイルの master_symmetric_key パラメータとして使用できるキーを生成する例については、「例: 256 ビットの AES シークレットキーを作成する (p. 376)」を参照してください。

```
# Base64 encoded AES 256 bit symmetric master key.
master_symmetric_key=***specify your key***

# Endpoint. Use s3-external-1.amazonaws.com for buckets in us-east
s3_endpoint=s3.amazonaws.com

# Bucket to upload data.
s3_bucket=***specify your bucket name***

# S3 prefix to add to uploaded data files. All files loaded will have this
prefix.
# Leave blank if no prefix is desired.
s3_prefix=

# Source files
src_dir = ***specify local directory***
```

プロパティファイルを使用してクライアント側の暗号化を実行するコードを次に示します。

```
import java.io.File;
import java.util.Properties;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import org.apache.commons.codec.binary.Base64;
import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3;
```

```
import com.amazonaws.services.s3.AmazonS3EncryptionClient;
import com.amazonaws.services.s3.model.EncryptionMaterials;
import com.amazonaws.services.s3.model.PutObjectRequest;

public class SymKeyEncryptAndUploadDirectoryToS3 {

    private static final int MAX_RETRY_COUNT = 10;
    private static AmazonS3EncryptionClient encryptedS3Client;

    public static void main(String[] args) throws Exception {

        AWSCredentials credentials = new PropertiesCredentials(
                SymKeyEncryptAndUploadDirectoryToS3.class
                        .getResourceAsStream("AwsCredentials.properties"));

        // Specify values in SymKeyEncryptAndUploadDirectoryToS3.properties
file.
        Properties config = new Properties();
        config.load(SymKeyEncryptAndUploadDirectoryToS3.class
               .getResourceAsStream("SymKeyEncryptAndUploadDirectoryToS3.prop
erties"));

        // Get property values.
        String masterSymmetricKeyBase64 = getProperty(config,
                "master_symmetric_key");
        String bucketName = getProperty(config, "s3_bucket");
        String s3Prefix = getProperty(config, "s3_prefix");
        String s3Endpoint = getProperty(config, "s3_endpoint");
        String sourceDir = getProperty(config, "src_dir");

        // Get secret key in correct format and create encryption materials.
        SecretKey mySymmetricKey = new SecretKeySpec(
              Base64.decodeBase64(masterSymmetricKeyBase64.getBytes()), "AES");

       EncryptionMaterials materials = new EncryptionMaterials(mySymmetricKey);


       encryptedS3Client = new AmazonS3EncryptionClient(credentials, materials);

        encryptedS3Client.setEndpoint(s3Endpoint);

        // Upload all files.
        uploadAllFilesToS3(encryptedS3Client, bucketName, s3Prefix, new File(
                sourceDir));

    }

    private static void uploadAllFilesToS3(AmazonS3 s3, String bucketName,
            String s3Prefix, final File folder) {

        System.out.println("Reading files from directory " + folder);

        for (final File fileEntry : folder.listFiles()) {
            if (!fileEntry.isDirectory()) { // Skip sub directories.

                int retryCount = 0;
                boolean done = false;
                while (!done) {
```

```
                    try {
                        uploadToS3(s3, bucketName, s3Prefix, fileEntry);
                        done = true;
                    } catch (Exception e) {
                        retryCount++;
                        if (retryCount > MAX_RETRY_COUNT) {
                            System.out
                                    .println("Retry count exceeded max retry
count "
                                            + MAX_RETRY_COUNT + ". Giving up");

                            throw new RuntimeException(e);
                        }

                        // Do retry after 10 seconds.
                       System.out.println("Failed to upload file " + fileEntry

                                + ". Retrying...");
                        try {
                            Thread.sleep(10 * 1000);
                        } catch (Exception te) {
                        }

                    }
                }// while

            }// for
        }

    }

    private static void uploadToS3(AmazonS3 s3, String bucketName,
            String s3Prefix, File file) {

        try {
            System.out.println("Uploading a new object to S3 object '"
                    + s3Prefix + "' from file " + file);
            String key = s3Prefix + "/" + file.getName();
            s3.putObject(new PutObjectRequest(bucketName, key, file));

        } catch (AmazonServiceException ase) {
            System.out
                    .println("Caught an AmazonServiceException, which means
your request made it "
                            + "to Amazon S3, but was rejected with an error
response for some reason.");
            System.out.println("Error Message:    " + ase.getMessage());
            System.out.println("HTTP Status Code: " + ase.getStatusCode());
            System.out.println("AWS Error Code:   " + ase.getErrorCode());
            System.out.println("Error Type:       " + ase.getErrorType());
            System.out.println("Request ID:       " + ase.getRequestId());

            throw ase;
        } catch (AmazonClientException ace) {
            System.out
                    .println("Caught an AmazonClientException, which means the
 client encountered "
                            + "a serious internal problem while trying to com
```

```
municate with S3, "
                            + "such as not being able to access the network.");

            System.out.println("Error Message: " + ace.getMessage());
            throw ace;
        }

    }

    private static String getProperty(Properties config, String name) {

        if (config.containsKey(name)) {
            return config.getProperty(name);
        }

        throw new RuntimeException(name + " property not configured");
    }
}
```

例: キーペアを使用した Amazon S3 への暗号化されたアップロード

この例では、キーペア (パブリックキーとプライベートキー) を作成し、それを対称キーとして使用して、クライアント側で暗号化されたアップロードを Amazon S3 に行う方法を示します。このコードは、キーペアを生成、ローカル保存、ロードおよび使用してテストファイルをアップロードします。典型的なユースケースでは、キーペアを 1 度だけ生成して安全な場所に保存します。

この例では、RSA アルゴリズムと 1024 ビットのキーサイズを使用するキーペアを作成します。別のアルゴリズムやキーサイズを使用するようコードを変更することもできます。

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStreamWriter;
import java.io.Writer;
import java.security.KeyFactory;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.SecureRandom;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;
import java.util.Properties;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3EncryptionClient;
import com.amazonaws.services.s3.model.EncryptionMaterials;
import com.amazonaws.services.s3.model.PutObjectRequest;
import com.amazonaws.services.s3.model.S3Object;

public class KeyPairSampleEncryptAndUploadDataToS3 {
```

```
    private static AmazonS3EncryptionClient encryptedS3Client;
    private static String encryptionAlgorithm = "RSA";
    private static String bucketName = "***provide bucket name***";
    private static String downloadDir = "***provide local directory***";
    private static String key = "keypairtest.txt";

    public static void main(String[] args) throws Exception {

        // Specify values in KeyPairSampleEncryptAndUploadDataToS3.properties
file.

        AWSCredentials credentials = new PropertiesCredentials(
                KeyPairSampleEncryptAndUploadDataToS3.class
                        .getResourceAsStream("AwsCredentials.properties"));

        // Generate a key pair. In a typical scenario, you would just generate
 this once.
        KeyPairGenerator keyGenerator = KeyPairGenerator
                .getInstance(encryptionAlgorithm);
        keyGenerator.initialize(1024, new SecureRandom());
        KeyPair myKeyPair = keyGenerator.generateKeyPair();

        // Save key pair so we can use key later.
        saveKeyPair(downloadDir, myKeyPair);

        // Create encryption materials and instantiate client.
        EncryptionMaterials encryptionMaterials = new EncryptionMaterials(
                myKeyPair);
        encryptedS3Client = new AmazonS3EncryptionClient(credentials,
                encryptionMaterials);

        // Upload a test file.
        encryptedS3Client.putObject(new PutObjectRequest(bucketName, key,
                createSampleFile()));

        // Load the test key from file to show that key was saved correctly.
        KeyPair myKeyPairRetrieved = loadKeyPair(downloadDir,
                encryptionAlgorithm);

        // Create new encryption materials using the retrieved key pair and a
        // new client based on the new materials.
        encryptionMaterials = new EncryptionMaterials(myKeyPairRetrieved);
        encryptedS3Client = new AmazonS3EncryptionClient(credentials,
                encryptionMaterials);

        // Download the object.
        // When you use the getObject method with the encrypted client,
        // the data retrieved from Amazon S3 is automatically decrypted on the
 fly.
        S3Object downloadedObject = encryptedS3Client
                .getObject(bucketName, key);
        SaveS3ObjectContent(downloadedObject, downloadDir, key);

    }

    private static File createSampleFile() throws IOException {
        File file = File.createTempFile("encryptiontest", ".txt");
        file.deleteOnExit();
```

```
        Writer writer = new OutputStreamWriter(new FileOutputStream(file));
        writer.write(new java.util.Date().toString() + "\n");
        writer.write("abcdefghijklmnopqrstuvwxyz\n");
        writer.write("01234567890112345678901234\n");
        writer.write("!@#$%^&*()-=[]{};':',.<>/?\n");
        writer.write("01234567890112345678901234\n");
        writer.write("abcdefghijklmnopqrstuvwxyz\n");
        writer.close();

        return file;
    }

    public static void saveKeyPair(String path, KeyPair keyPair)
            throws IOException {
        PublicKey publicKey = keyPair.getPublic();
        PrivateKey privateKey = keyPair.getPrivate();

        // Save public key to file.
        X509EncodedKeySpec x509EncodedKeySpec = new X509EncodedKeySpec(
                publicKey.getEncoded());
        FileOutputStream keyfos = new FileOutputStream(path + "/public.key");
        keyfos.write(x509EncodedKeySpec.getEncoded());
        keyfos.close();

        // Save private key to file.
        PKCS8EncodedKeySpec pkcs8EncodedKeySpec = new PKCS8EncodedKeySpec(
                privateKey.getEncoded());
        keyfos = new FileOutputStream(path + "/private.key");
        keyfos.write(pkcs8EncodedKeySpec.getEncoded());
        keyfos.close();

    }

    public static KeyPair loadKeyPair(String path, String algorithm)
            throws IOException, NoSuchAlgorithmException,
            InvalidKeySpecException {
        // Read public key from file.
        FileInputStream keyfis = new FileInputStream(path + "/public.key");
        byte[] encodedPublicKey = new byte[keyfis.available()];
        keyfis.read(encodedPublicKey);
        keyfis.close();

        // Read private key from file.
        keyfis = new FileInputStream(path + "/private.key");
        byte[] encodedPrivateKey = new byte[keyfis.available()];
        keyfis.read(encodedPrivateKey);
        keyfis.close();

        // Generate KeyPair from public and private keys.
        KeyFactory keyFactory = KeyFactory.getInstance(algorithm);
        X509EncodedKeySpec publicKeySpec = new X509EncodedKeySpec(
                encodedPublicKey);
        PublicKey publicKey = keyFactory.generatePublic(publicKeySpec);

        PKCS8EncodedKeySpec privateKeySpec = new PKCS8EncodedKeySpec(
                encodedPrivateKey);
        PrivateKey privateKey = keyFactory.generatePrivate(privateKeySpec);
```

```
        return new KeyPair(publicKey, privateKey);

    }

    public static void SaveS3ObjectContent(S3Object obj, String path,
            String filename) throws IOException {
        InputStream stream = obj.getObjectContent();
        FileOutputStream fos = new FileOutputStream(path + "/" + filename);
        byte[] buffer = new byte[1024];
        int len;
        while ((len = stream.read(buffer)) != -1) {
            fos.write(buffer, 0, len);
        }
        fos.close();
    }
}
```

クライアント側暗号化環境をカスタマイズする

AWS SDK for Java を使用したクライアント側の暗号化では、暗号化/復号化プロセスをカスタマイズするいくつかのオプションがあります。このトピックでは、よく使用するオプションの設定方法とその意味について説明します。

`AmazonS3EncryptionClient` クラスの適切なコンストラクタを選択することで、クライアント側の暗号化環境を設定できます。このコンストラクタには複数の署名があり、`AWSCredentials`、`ClientConfiguration`、`EncryptionMaterials`、および `CryptoConfiguration` という 4 つの引数を組み合わせて取ることができます。`AWSCredentials` 引数と `ClientConfiguration` 引数は、暗号化されない Amazon S3 クライアント `AmazonS3Client` で使用するのと同じ引数です。そのため、ここでは説明しません。残り 2 つの `EncryptionMaterials` 引数と `CryptoConfiguration` 引数を使用して、クライアント側の暗号化のさまざまな側面をカスタマイズできます。これらについて以下で説明します。暗号化クライアントコンストラクタの詳細については、AWS Java SDK のドキュメントを参照してください。

暗号化マテリアルの指定

Amazon S3 のクライアント側の暗号化を使用するには、暗号化プロセスに使用するキーを指定する必要があります。キーは、暗号化したデータのアップロードと取得に使用します。このキーは安全な場所に保管する必要があります。

Important

プライベート暗号化キーと暗号化されていないデータは AWS には送信されません。したがって、暗号化キーを紛失すると、データを復号化できなくなります。

`EncryptionMaterials` クラスを使用すると、Amazon S3 の暗号化クライアントに対して、暗号化に使用するキーを指定できます。非対称キーと対称キーのどちらも指定できますが、可能な限り非対称キーを使用することをお勧めします。非対称キーは、パブリックキーとプライベートキーで構成されるキーペアです。

次のコードスニペットは、新しいキーペア (対称キー) を作成する方法を示しています。キーペアをファイルとの間で読み書きする方法の例については、「例: キーペアを使用した Amazon S3 への暗号化されたアップロード (p. 380)」を参照してください。

```
// Create a new asymmetric key.
KeyPairGenerator keyGenerator = KeyPairGenerator.getInstance(encryptionAl
```

```
gorithm);
keyGenerator.initialize(4096, new SecureRandom());
KeyPair keyPair = keyGenerator.generateKeyPair();

// Construct an instance of AmazonS3EncryptionClient.
AWSCredentials credentials = new BasicAWSCredentials(myAccessKeyId, mySecretKey);

// Create an EncryptionMaterials object using the symmetric key.
EncryptionMaterials encryptionMaterials = new EncryptionMaterials(keyPair);

// Create a new encryption client.
AmazonS3EncryptionClient encryptedS3Client =
 new AmazonS3EncryptionClient(credentials, encryptionMaterials);
```

次のコードス二ペットは、新しい対称キーを作成する方法を示しています。対称キーをファイルとの間で読み書きする方法の例については、「例: 256 ビットの AES シークレットキーを作成する (p. 376)」を参照してください。

```
// Create a new symmetric key.
KeyGenerator symKeyGenerator = KeyGenerator.getInstance("DES");
SecretKey symKey = symKeyGenerator.generateKey();

// Construct an instance of AmazonS3EncryptionClient.
AWSCredentials credentials = new BasicAWSCredentials(myAccessKeyId, mySecretKey);

// Create an EncryptionMaterials object using the symmetric key.
EncryptionMaterials encryptionMaterials = new EncryptionMaterials(symKey);

// Create a new encryption client.
AmazonS3EncryptionClient encryptedS3Client =
 new AmazonS3EncryptionClient(credentials, encryptionMaterials);
```

暗号化メタデータのストレージロケーションの指定

Amazon S3 クライアントが (`AmazonS3EncryptionClient` クラスを使用して) データを暗号化して Amazon S3 にアップロードするときに、暗号化されたエンベロープ対称キーも &S3 に格納されます。デフォルトでは、暗号化されたキーは、ユーザー定義のオブジェクトメタデータとして格納されます。暗号化したオブジェクトをアップロードした後で、オブジェクトのプロパティを表示し、暗号化に関連する追加のメタデータの名前/値のペアを確認できます。例えば、クライアント側で暗号化され、Amazon S3 にアップロードされたオブジェクトには、x-amz-meta-x-amz-key というキー名と、エンベロープキーと等しいキー値が設定されます。

オプションで、暗号化メタデータを*インストラクションファイル*として、暗号化したファイルと同じ場所に格納することもできます。インストラクションファイルは、暗号化したデータファイルと同じキー名で、「.instruction」という拡張子が末尾に付きます。オブジェクトメタデータとして大きすぎるために暗号化キーの強度として対称キーを使用する場合は、インストラクションファイルを使用するとよいでしょう。メタデータは 2 KB 未満にしてください。暗号化メタデータは、オブジェクトメタデータとインストラクションファイルのどちらかの形式で格納されます。併用はできません。

次のコード例は、`CryptoConfiguration` クラスと、`AmazonS3EncryptionClient` クラスの適切なコンストラクタを使用して、暗号化メタデータをインストラクションファイルに格納する方法を示しています。

```
// Load the asymmetric key pair.
KeyPair myKeyPair = S3ClientSideEncryptionTest.LoadKeyPair(pathToKey, encryp
tionAlgorithm);
```

```
// Construct an instance of AmazonS3EncryptionClient.
AWSCredentials credentials = new BasicAWSCredentials(myAccessKeyId, mySecretKey);
EncryptionMaterials encryptionMaterials = new EncryptionMaterials(myKeyPair);

// Create an instance of the CryptoConfiguration class to specify using the
instruction file.
CryptoConfiguration cryptoConfig =
    new CryptoConfiguration().withStorageMode(CryptoStorageMode.InstructionFile);

// Use the CryptoConfiguration instance in the encryption client constructor.
AmazonS3EncryptionClient encryptedS3 =
 new AmazonS3EncryptionClient(credentials, encryptionMaterials, cryptoConfig);

// Upload the object.
encryptedS3.putObject(new PutObjectRequest(bucketName, key, createSampleFile()));


// Retrieve the object.
S3Object downloadedObject = encryptedS3.getObject(bucketName, key);
```

暗号化プロバイダの指定

Amazon S3 のクライアント側暗号化クライアントで使用されるデフォルトの暗号化プロバイダは、Java Cryptography Extension (JCE) です。オプションで、別の暗号化プロバイダ実装を指定して、データの暗号化と復号化に使用できます。

次のサンプルコードでは、Bouncy Castle プロバイダを使用します。

```
// Create a new instance of the provider.
Provider bcProvider = new BouncyCastleProvider();

// Load the asymmetric key pair.
KeyPair myKeyPair = S3ClientSideEncryptionTest.LoadKeyPair(pathToKey, encryp
tionAlgorithm);

// Construct an instance of AmazonS3EncryptionClient.
AWSCredentials credentials = new BasicAWSCredentials(myAccessKeyId, mySecretKey);
EncryptionMaterials encryptionMaterials = new EncryptionMaterials(myKeyPair);

// Create an instance of the CryptoConfiguration class to specify a crypto
provider.
CryptoConfiguration cryptoConfig =
    new CryptoConfiguration().withCryptoProvider(bcProvider);

// Use the CryptoConfiguration instance in the encryption client constructor.
AmazonS3EncryptionClient encryptedS3 =
 new AmazonS3EncryptionClient(credentials, encryptionMaterials, cryptoConfig);

// Upload the object.
encryptedS3.putObject(new PutObjectRequest(bucketName, key, createSampleFile()));


// Retrieve the object.
S3Object downloadedObject = encryptedS3.getObject(bucketName, key);
```

低冗長化ストレージの使用

Topics

Amazon S3 低冗長化ストレージ (RRS) オプションは、年間 99.99% のオブジェクトの耐久性を提供するように設計されています。耐久性は多少下がるものの可用性の高いストレージオプションです。

Amazon S3 はストレージクラスに従ってオブジェクトを保存します。ストレージクラスは、オブジェクトが Amazon S3 に書き込まれるときに、Amazon S3 によってオブジェクトに割り当てられます。デフォルトのストレージクラスは STANDARD です。お客様がオブジェクトに特定のストレージクラス (STANDARD または REDUCED_REDUNDANCY) を割り当てることができるのは、Amazon S3 にオブジェクトを書き込むとき、または Amazon &S3 に保存されているオブジェクトをコピーするときのみです。RRS は、オブジェクトを書き込むときにオブジェクト単位で指定します。

この機能の全般的な概要については、「低冗長化ストレージ (p. 7)」を参照してください。

アップロードするオブジェクトのストレージクラスの設定

RRS にアップロードするオブジェクトのストレージクラスを設定するには、PUT リクエストで *x-amz-storage-class* を REDUCED_REDUNDANCY に設定します。

RRS にアップロードするオブジェクトのストレージクラスを設定する方法

- PUT Object リクエストを作成します。このとき、*x-amz-storage-class* リクエストヘッダーを REDUCED_REDUNDANCY に設定します。
  PUT オペレーションを実行するには、バケットに対する適切なアクセス許可が必要です。ストレージクラスのデフォルト値は、STANDARDです (通常の Amazon S3 ストレージの場合) 。

  次の例では、my-image.jpg のストレージクラスを RRS に設定します。

```
PUT /my-image.jpg HTTP/1.1
Host: myBucket.s3.amazonaws.com
Date: Wed, 12 Oct 2009 17:50:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:xQE0diMbLRepdf3YB+FIEXAMPLE=
Content-Type: image/jpeg
Content-Length: 11434
Expect: 100-continue
x-amz-storage-class: REDUCED_REDUNDANCY
```

Amazon S3 のオブジェクトのストレージクラスの変更

Topics

Amazon S3 に既に保存されているオブジェクトを、同じバケットの同じキー名にコピーして、そのオブジェクトのストレージクラスを変更することもできます。これを行うには、PUT Object copy リクエストで以下のリクエストヘッダーを使用します。

- *x-amz-metadata-directive* を COPY に設定します。
- *x-amz-storage-class* を STANDARD または REDUCED_REDUNDANCY に設定します。

Important

コピーリクエストの実行を最適化するには、PUT Object copy リクエストで他のメタデータを変更しないでください。ストレージクラス以外のメタデータを変更する必要がある場合は、*x-amz-metadata-directive* を REPLACE に設定するとパフォーマンスが向上します。

Amazon S3 でオブジェクトのストレージクラスを書き換える方法

- PUT Object copy リクエストを作成し、リクエストの *x-amz-storage-class* ヘッダーを REDUCED_REDUNDANCY (RRS の場合) または STANDARD (通常の &S3 ストレージの場合) に設定し、コピー先の名前をコピー元の名前と同じにします。

  コピーオペレーションを実行するには、バケットに対する適切なアクセス許可が必要です。

  次の例では、my-image.jpg のストレージクラスを RRS に設定します。

```
PUT /my-image.jpg HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
x-amz-copy-source: /bucket/my-image.jpg
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
x-amz-storage-class: REDUCED_REDUNDANCY
x-amz-metadata-directive: COPY
```

  次の例では、my-image.jpg のストレージクラスを標準に設定します。

```
PUT /my-image.jpg HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
x-amz-copy-source: /bucket/my-image.jpg
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
x-amz-storage-class: STANDARD
x-amz-metadata-directive: COPY
```

Note

RRS オブジェクトをコピーするときに *x-amz-storage-class* リクエストヘッダーを含めなかった場合、コピー先オブジェクトのストレージクラスはデフォルトで STANDARD になります。

特定バージョンのオブジェクトのストレージクラスは変更できません。このようなオブジェクトをコピーすると、Amazon S3 によってオブジェクトに新しいバージョン ID が設定されます。

Note

コピーリクエストでオブジェクトを書き込む場合は、新しいストレージクラスを適用するために、オブジェクト全体が再度書き込まれます。

バージョニングの詳細については、「バージョニングの使用 (p. 388)」を参照してください。

データ消失のリターンコード

Amazon S3 でオブジェクトの消失が検出された場合、後続の `GET` または `HEAD` オペレーション、あるいは消失したオブジェクトをコピー元として使用する `PUT Object copy` オペレーションは、`405 Method Not Allowed` エラーになります。オブジェクトが消失したとマークされると、Amazon S3 ではオブジェクトを復元できません。この場合、キーを削除するか、オブジェクトのコピーをアップロードします。

バージョニングの使用

Topics

バージョニングとは、同じバケット内でオブジェクトの複数のバリアントを保持する手段です。例えば、1 つのバケット内に、`photo.gif` (バージョン 111111) と `photo.gif` (バージョン 121212) のように、キーは同じだがバージョン ID が異なる 2 つのオブジェクトを保持することができます。

バージョニングを有効にすると、オブジェクトが誤って削除または上書きされるのを防いだり、以前のバージョンを取得できるようオブジェクトをアーカイブしたりできます。

Note

SOAP API は、バージョニングをサポートしていません。

バケットのバージョニング状態の有効化

Topics

バケットは、バージョニング無効 (デフォルト)、バージョニング有効、バージョニング停止の 3 つの状態のいずれかに設定できます。一度バケットのバージョニングを有効にすると、バージョニング無効の状態に戻すことはできません。ただし、そのバケットのバージョニングを停止することは可能です。

バケットのバージョニング状態を設定できるのは、バケット所有者のみです。バージョニング状態は、そバケット内の一部ではなくすべてのオブジェクトに適用されます。最初にバケットのバージョニングを有効にした時点で、バケット内のオブジェクトは常にバージョニングされ、固有のバージョン ID が与えられます。

MFA Delete

セキュリティ層をさらに強化するには、MFA (Multi-Factor Authentication) Delete に対応するようにバケットを設定します。Amazon S3 バケットで MFA Delete を有効にした場合、次の 2 つの形態の認証を一緒に指定すると、バケットのバージョニング状態を変更するか、またはオブジェクトバージョンを永久に削除するかのいずれかのみを行うことができます。

- AWS アカウント認証情報
- 有効なシリアル番号、スペース、および承認済みの認証デバイスに表示される 6 桁のコードを連結した文字

MFA Delete を使用するには、ハードウェアデバイスまたは仮想 MFA デバイスを使用して認証コードを生成します。次の例は、生成された認証コードがハードウェアデバイスに表示されている様子を示しています。

Note

MFA Delete および MFA で保護された API アクセスは、異なるシナリオで保護を提供することを目的とした機能です。バケットに MFA Delete を設定して、バケット内のデータが誤って削除されないようにします。MFA で保護された API アクセスは、 Amazon S3 の機密リソースにアクセスする場合に、別の認証要素 (MFA コード) を適用するために使用します。これらの Amazon S3 リソースに対するすべてのオペレーションを実行するのに、MFA を使用して作成された一時的な認証情報を必ず使用するように要求できます。例については、「MFA 認証を要求するバケットポリシーの追加 (p. 335)」を参照してください。

認証デバイスを購入およびアクティベートする方法の詳細については、http://aws.amazon.com/mfa/ を参照してください。

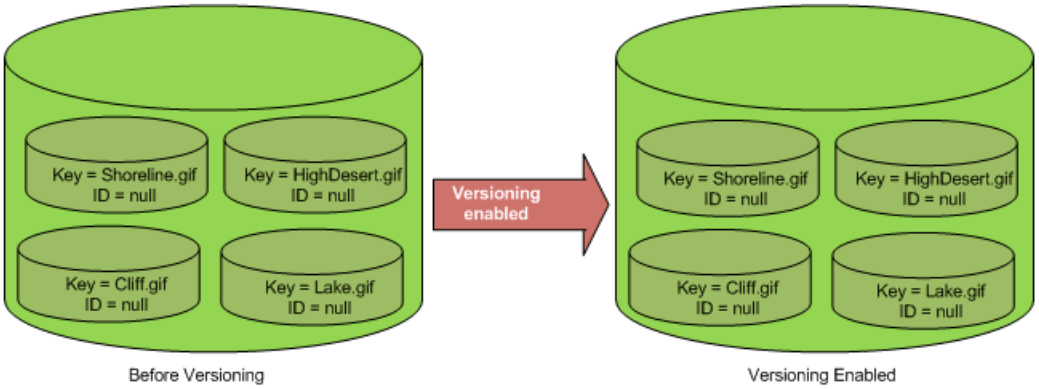
MFA Delete を有効にするようにバケットを設定する方法の詳細については、「MFA Delete でのバケットの設定 (p. 390)」を参照してください。

バージョニングの有効化

Topics

- MFA Delete でのバケットの設定 (p. 390)

バージョニング状態を設定する前にバケットに格納されているオブジェクトには、バージョン ID `null` が付けられています。次の図に示すように、バージョニングを有効にした場合、バケット内のオブジェクトが変更されることはありません。



変更されるのは、今後のリクエストでそのバケット内のオブジェクトが Amazon S3 で処理される方法です。

オブジェクトのバージョニングを有効にするには

1. `PUT Bucket` リクエストに、*`versioning`* サブリソースを含めます。
2. リクエストの本文で、*`Status`* リクエスト要素の値に `Enabled` を使用します。

Example バージョニングの有効化

次のリクエストは、バケット *`bucketName`* でバージョニングを有効にします。

```
PUT /?versioning HTTP/1.1
Host: bucketName.s3.amazonaws.com
Date: Wed, 12 Oct 2009 17:50:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:xQE0diMbLRepdf3YB+FIEXAMPLE=
Content-Type: text/plain
Content-Length: 124

<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Status>Enabled</Status>
</VersioningConfiguration>
```

MFA Delete でのバケットの設定

バケットのバージョニング状態を指定するのと同時に、MFA Delete を有効にするかどうかを選択できます。MFA Delete が有効になるようにバケットを設定した場合、バージョニング状態の変更またはバージョンの削除を行う今後のすべてのリクエストで、リクエストヘッダー `x-amz-mfa: [SerialNumber] [AuthenticationCode]` が必要になります。*`[SerialNumber]`* と *`[AuthenticationCode]`* の間にスペースがあることに注意してください。*`x-amz-mfa`* を含むリクエストでは、HTTPS を使用する必要があります。

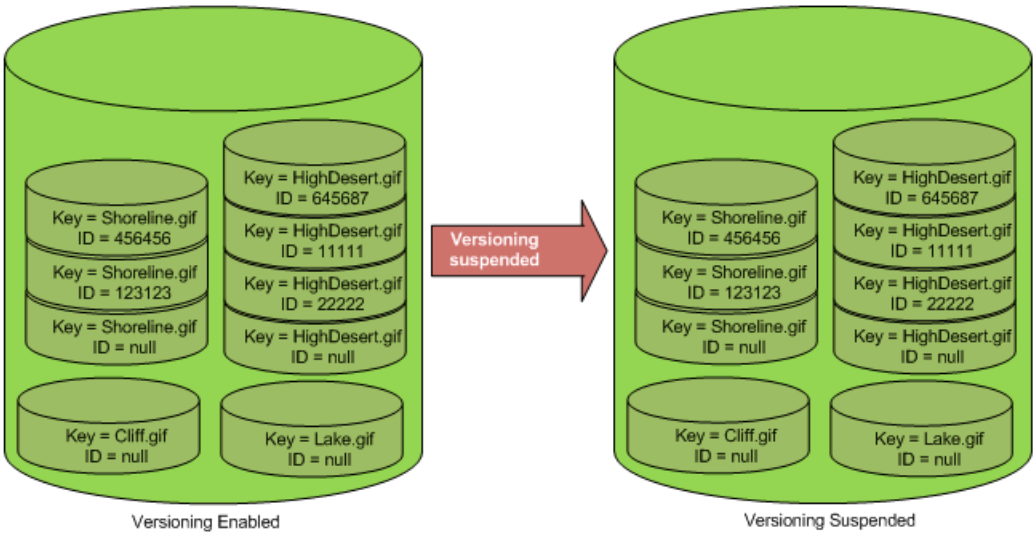
Example バージョニングおよび MFA Delete の有効化

次のリクエストは、*bucketName* でバージョニングおよび MFA Delete を有効にします。

```
PUT /?versioning HTTPS/1.1
Host: bucketName.s3.amazonaws.com
Date: Wed, 12 Oct 2009 17:50:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:xQE0diMbLRepdf3YB+FIEXAMPLE=
Content-Type: text/plain
Content-Length: 124
x-amz-mfa: 20899872 301749

<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Status>Enabled</Status>
  <MfaDelete>Enabled</MfaDelete>
</VersioningConfiguration>
```

バージョニングの停止

バケット所有者はバージョニングを停止して、オブジェクトバージョンが生成されないようにすることができます。次の図に示すように、バージョニングを停止した場合、バケット内のオブジェクトが変更されることはありません。



変更されるのは、今後のリクエストでオブジェクトが Amazon S3 で処理される方法です。バージョニングを停止した場合の効果の詳細については、「バージョニングが停止されたバケットの使用 (p. 405)」を参照してください。

オブジェクトのバージョニングを停止するには

1. `PUT Bucket` リクエストのヘッダーに、*versioning* サブリソースを含めます。
2. リクエストの本文で、*Status* リクエスト要素の値に `Suspended` を使用します。

Example バージョニングの停止

次のリクエストは、*bucketName* でバージョニングを停止します。

```
PUT /?versioning HTTP/1.1
Host: bucketName.s3.amazonaws.com
Date: Wed, 12 Oct 2009 17:50:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:xQE0diMbLRepdf3YB+FIEXAMPLE=
Content-Type: text/plain
Content-Length: 124

<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Status>Suspended</Status>
</VersioningConfiguration>
```

MFA Delete が有効になるようにバケットを設定した場合、バケットのバージョニング状態を変更するリクエストに、*x-amz-mfa* リクエストヘッダーおよび *MfaDelete* リクエスト要素を含める必要があります。

Example MFA Delete を使用したバージョニングの停止

次のリクエストは、MFA Delete で設定された *bucketName* でバージョニングを停止します。

```
PUT /?versioning HTTPS/1.1
Host: bucketName.s3.amazonaws.com
Date: Wed, 12 Oct 2009 17:50:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:xQE0diMbLRepdf3YB+FIEXAMPLE=
x-amz-mfa: 20899872 301749
Content-Type: text/plain
Content-Length: 124

<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Status>Suspended</Status>
  <MfaDelete>Enabled</MfaDelete>
</VersioningConfiguration>
```

x-amz-mfa を含むリクエストでは、HTTPS を使用する必要があります。

バージョニング状態の確認

versioning サブリソースを使用すると、バケットのバージョニング状態を取得できます。

バケットのバージョニング状態を取得するには

1. GET Bucket リクエストのヘッダーに、*versioning* サブリソースを含めます。

   ```
   GET /?versioning HTTP/1.1
   Host: bucketName.s3.amazonaws.com
   Date: Wed, 12 Oct 2009 17:50:00 GMT
   Authorization: AWS AKIAIOSFODNN7EXAMPLE:xQE0diMbLRepdf3YB+FIEXAMPLE=
   Content-Type: text/plain
   ```

2. レスポンスで、レスポンス要素 *Status* の値を確認します。次の 3 とおりの結果が考えられます。
   - バケットでバージョニングが有効になっている場合、レスポンスは次のようになります。

```
<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Status>Enabled</Status>
</VersioningConfiguration>
```

- バケットでバージョニングが停止されている場合、レスポンスは次のようになります。

```
<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Status>Suspended</Status>
</VersioningConfiguration>
```

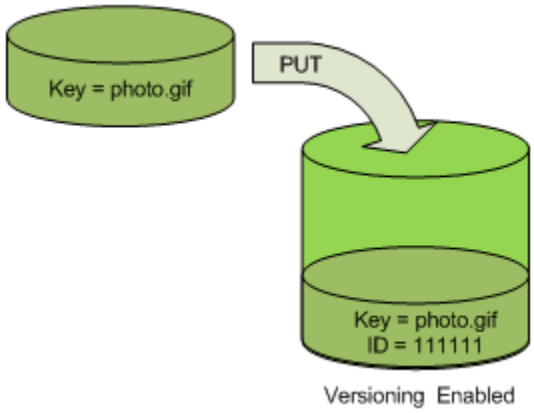
- バケットのバージョニングが一度も有効になっていない場合、レスポンスは *status* 要素を取得しません。

```
<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/"/>
```

バージョニングが有効なバケットへのオブジェクトの追加

バケットでバージョニングを有効にすると、PUT、POST、または COPY を使用してバケットに保存されているすべてのオブジェクトに、一意のバージョン ID が自動的に追加されます。

次の図は、バージョニングが有効なバケットにオブジェクトが追加されたときに、Amazon S3 がそのオブジェクトに一意のバージョン ID を追加する方法を示しています。



バージョニングが有効なバケットへのオブジェクトの追加

1	PUT Bucket versioning リクエストを使用して、バケットでバージョニングを有効にします。詳細については、「PUT Bucket versioning」を参照してください。
2	PUT、POST、または COPY リクエストを送信して、バケットにオブジェクトを格納します。

バージョニングが有効なバケットにオブジェクトを追加すると、オブジェクトのバージョン ID が *x-amz-versionid* レスポンスヘッダーに入れて返されます。次に例を示します。

```
x-amz-version-id: 3/L4kqtJlcpXroDTDmJ+rmSpXd3dIbrHY
```

Note

通常の Amazon S3 料金は、保存または移行されたオブジェクトのバージョンごとに適用されます。オブジェクトの各バージョンはオブジェクト全体であり、以前のバージョンの単なる差分ではありません。したがって、オブジェクトの 3 つのバージョンを格納している場合、3 つのオブジェクトに対して課金されます。

バージョニングが有効なバケットでのオブジェクトのリスト

Topics

- バケット内のオブジェクトのサブセットの取得 (p. 394)

バケット内のすべてのオブジェクトのすべてのバージョンをリストするには、GET Bucket リクエストの *versions* サブリソースを使用します。Amazon S3 が取得できるオブジェクトは最大 1,000 個のみで、各バージョンのオブジェクトは完全な 1 個のオブジェクトとしてカウントされます。したがって、バケットに 2 つのキー (例えば、photo.gif と picture.jpg) が含まれており、最初のキーに 990 個のバージョン、2 つ目のキーに 400 個のバージョンがある場合、最初のリクエストで、photo.gif の 990 個のすべてのバージョンと picture.jpg の最新の 10 個のバージョンのみが取得されます。

Amazon S3 は、保存時刻の新しい順にオブジェクトバージョンをリストします。

バケット内のすべてのオブジェクトバージョンを表示するには

- GET Bucket リクエストに、*versions* サブリソースを含めます。

```
GET /?versions HTTP/1.1
Host: bucketName.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 +0000
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

バケット内のオブジェクトのサブセットの取得

Topics

- キーのすべてのバージョンの取得 (p. 395)
- 最大キー超過後の追加のオブジェクトバージョンの取得 (p. 395)

バケット内のすべてのオブジェクトバージョンのサブセットを取得する場合や、オブジェクトバージョンの数が *max-key* の値 (デフォルトでは 1000) を超えたために、2 つ目のリクエストを送信して残りのオブジェクトバージョンを取得しなければならない場合があります。オブジェクトバージョンのサブセットを取得するには、GET Bucket のリクエストパラメータを使用する必要があります。詳細については、「GET Bucket」を参照してください。

キーのすべてのバージョンの取得

以下の手順にしたがって *versions* サブリソースおよび *prefix* リクエストパラメータを使用すると、オブジェクトのすべてのバージョンを取得できます。*prefix* の詳細については、「GET Bucket」を参照してください。

キーのすべてのバージョンの取得

1	*prefix* パラメータを、取得するオブジェクトのキーに設定します。
2	*versions* サブリソースおよび *prefix* を使用して、GET Bucket リクエストを送信します。 `GET /?versions&prefix=objectName HTTP/1.1`

Example プレフィックスを使用したオブジェクトの取得

次の例では、キーが myObject であるか、またはそれで始まるオブジェクトを取得します。

```
GET /?versions&prefix=myObject HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

オブジェクトのすべてのバージョンのサブセットを取得するには、他のリクエストパラメータを使用します。詳細については、「GET Bucket」を参照してください。

最大キー超過後の追加のオブジェクトバージョンの取得

GET リクエストで返すことができるオブジェクトの数が *max-keys* の値を超えた場合、レスポンスには <isTruncated>true</isTruncated> と、リクエストを満たすが返されなかった最初のキー（*NextKeyMarker* 内）および最初のバージョン ID（*NextVersionIdMarker* 内）が含まれます。GET リクエストを満たす追加のオブジェクトを取得する後続のリクエストで、これらの戻り値を開始位置として使用します。

次の手順を行って、バケットの元の GET Bucket versions リクエストを満たす追加のオブジェクトを取得します。*key-marker*、*version-id-marker*、*NextKeyMarker*、および *NextVersionIdMarker* の詳細については、「GET Bucket」を参照してください。

元の GET リクエストを満たす追加のレスポンスの取得

1	*key-marker* の値を、前のレスポンスの *NextKeyMarker* で返されたキーに設定します。
2	*version-id-marker* の値を、前のレスポンスの *NextVersionIdMarker* で返されたバージョン ID に設定します。
3	*key-marker* および *version-id-marker* を使用して GET Bucket versions リクエストを送信します。

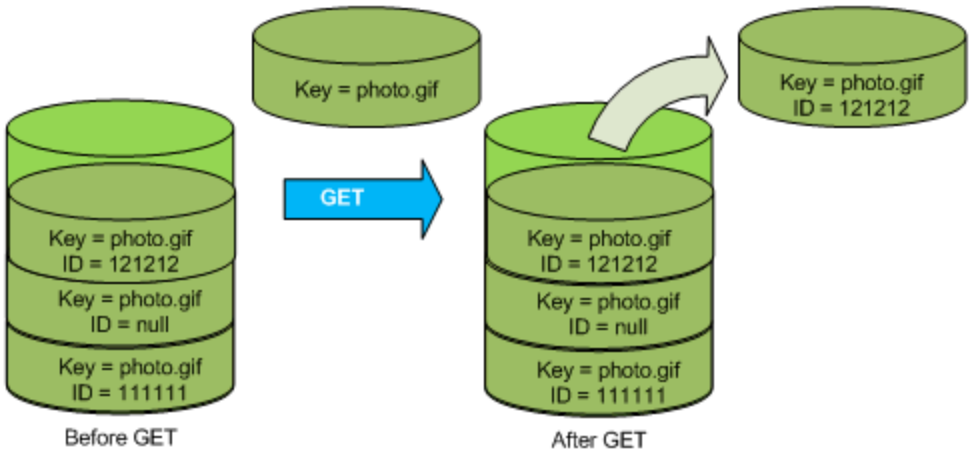
Example 指定したキーおよびバージョン ID を始点としたオブジェクトの取得

```
GET /?versions&key-marker=myObject&version-id-marker=298459348571 HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

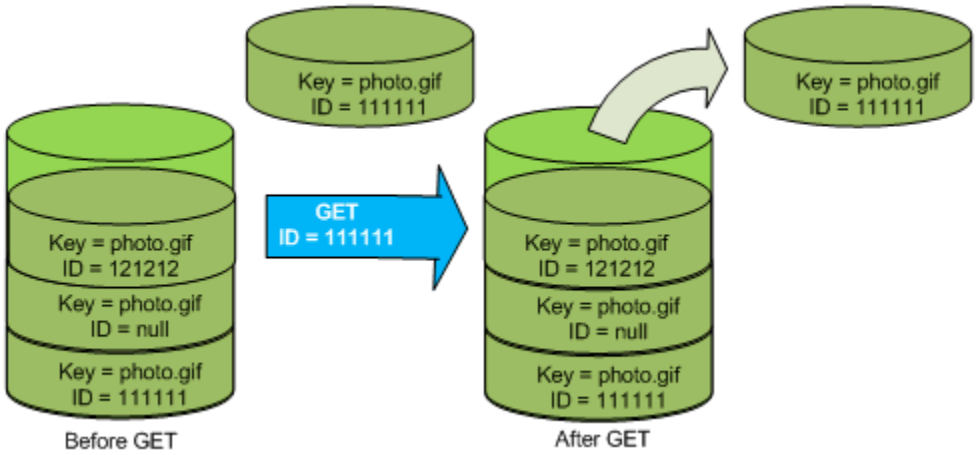
オブジェクトバージョンの取得

Topics

シンプルな GET リクエストが、オブジェクトの最新バージョンを取得します。次の図は、GET がオブジェクト (photo.gif) の最新バージョンを返す方法を示しています。



特定のバージョンを取得するには、そのバージョン ID を指定する必要があります。次の図は、GET versionId リクエストが、オブジェクトの指定したバージョン (最新とは限らない) を取得する方法を示しています。



特定のオブジェクトバージョンを取得するには

1. *versionId* を、取得するオブジェクトのバージョン ID に設定します。
2. GET Object versionId リクエストを送信します。

Example バージョニングされたオブジェクトの取得

次のリクエストは、my-image.jpg のバージョン L4kqtJlcpXroDTDmpUMLUo を取得します。

```
GET /my-image.jpg?versionId=L4kqtJlcpXroDTDmpUMLUo HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

オブジェクトバージョンのメタデータの取得

オブジェクトのメタデータのみを取得するには (コンテンツを除く) 、HEAD オペレーションを使用します。デフォルトでは、最新バージョンのメタデータが取得されます。特定のオブジェクトバージョンのメタデータを取得するには、そのバージョン ID を指定します。

オブジェクトバージョンのメタデータを取得するには

1. *versionId* を、メタデータを取得するオブジェクトのバージョン ID に設定します。
2. HEAD Object versionId リクエストを送信します。

Example バージョニングされたオブジェクトのメタデータの取得

次のリクエストは、my-image.jpg のバージョン 3HL4kqCxf3vjVBH40Nrjfkd のメタデータを取得します。

```
HEAD /my-image.jpg?versionId=3HL4kqCxf3vjVBH40Nrjfkd HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

以下に、サンプルレスポンスを示します。

```
HTTP/1.1 200 OK
x-amz-id-2: ef8yU9AS1ed4OpIszj7UDNEHGran
x-amz-request-id: 318BC8BC143432E5
x-amz-version-id: 3HL4kqtJlcpXroDTDmjVBH40Nrjfkd
Date: Wed, 28 Oct 2009 22:32:00 GMT
Last-Modified: Sun, 1 Jan 2006 12:00:00 GMT
ETag: "fba9dede5f27731c9771645a39863328"
Content-Length: 434234
Content-Type: text/plain
Connection: close
Server: AmazonS3
```

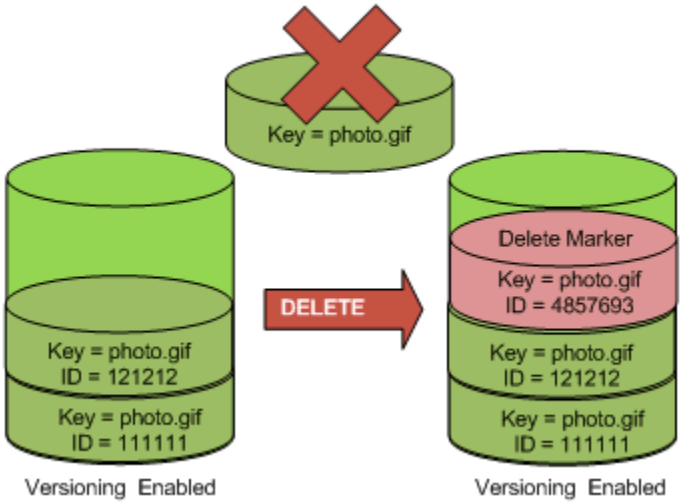
オブジェクトバージョンの削除

Topics

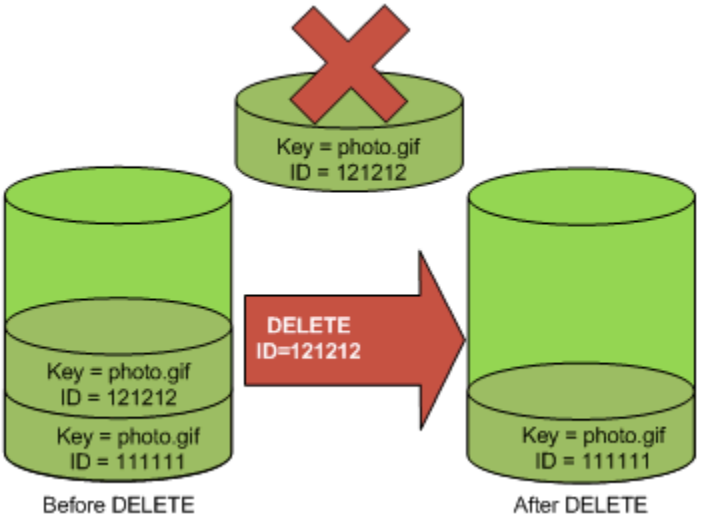
DELETE リクエストには次のユースケースがあります。

- バージョニングが有効になっている場合、シンプルな DELETE ではオブジェクトを完全に削除することはできません。
  代わりに、Amazon S3 はバケットに削除マーカーを挿入します。このマーカーが新しい ID を持つオブジェクトの最新バージョンになります。最新バージョンが削除マーカーであるオブジェクトを GET しようとすると、Amazon S3 は、オブジェクトが (消去されていなくても) 削除されたものとして動作し、エラー 404 を返します。
- バージョニングされたオブジェクトを完全に削除するには、`DELETE Object versionId` を使用する必要があります。

次の図は、シンプルな DELETE が、指定したオブジェクトを実際には削除しないことを示しています。代わりに、Amazon S3 は削除マーカーを挿入します。



次の図は、指定したオブジェクトバージョンを削除することによって、そのオブジェクトを完全に削除する方法を示しています。



オブジェクトの特定のバージョンを削除するには

- DELETE リクエストでバージョン ID を指定します。

Example 特定のバージョンの削除

次の例は、photo.gif のバージョン UIORUnfnd89493jJFJ を削除する方法を示しています。

```
DELETE /photo.gif?versionId=UIORUnfnd89493jJFJ HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 12 Oct 2009 17:50:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:xQE0diMbLRepdf3YB+FIEXAMPLE=
Content-Type: text/plain
Content-Length: 0
```

MFA Delete の使用

バケットのバージョニング設定で MFA Delete が有効になっている場合に、オブジェクトバージョンを完全に削除するか、バケットのバージョニング状態を変更するには、バケット所有者はリクエストに *x-amz-mfa* リクエストヘッダーを含める必要があります。ヘッダーの値は、認証デバイスのシリアル番号、スペース、および認証デバイスに表示される認証コードの連結文字です。このリクエストヘッダーを含めない場合、リクエストは失敗します。

x-amz-mfa を含むリクエストでは、HTTPS を使用する必要があります。

認証デバイスの詳細については、http://aws.amazon.com/mfa/ を参照してください。

Example MFA Delete が有効なバケットからのオブジェクトの削除

次の例は、MFA Delete が有効に設定されたバケット内の my-image.jpg を (バージョンを指定して) 削除する方法を示しています。

```
DELETE /my-image.jpg?versionId=3HL4kqCxf3vjVBH40Nrjfkd HTTPS/1.1
Host: bucketName.s3.amazonaws.com
x-amz-mfa: 20899872 301749
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

[SerialNumber] と *[AuthenticationCode]* の間にスペースがあることに注意してください。詳細については、「DELETE Object」を参照してください。

削除マーカーの使用

削除マーカーは、シンプルな DELETE リクエストで指定された、バージョニングされたオブジェクトのプレースホルダー (マーカー) です。オブジェクトがバージョニング有効なバケット内にあったので、そのオブジェクトは削除されませんでした。ただし、Amazon S3 は、削除マーカーを挿入したため、オブジェクトを削除したものとして動作します。

削除マーカーは、(キーとバージョン ID を持つ) 他のオブジェクトと同じように動作しますが、以下の点を除きます。

- 関連付けられたデータがない
- アクセスコントロールリスト (ACL) 値に関連付けられていない
- データがないため、GET リクエストから何も取得しない。そのため、エラー 404 を受け取ります。
- 削除マーカーに対して使用できる唯一のオペレーションは DELETE で、バケット所有者のみがこのリクエストを発行できる

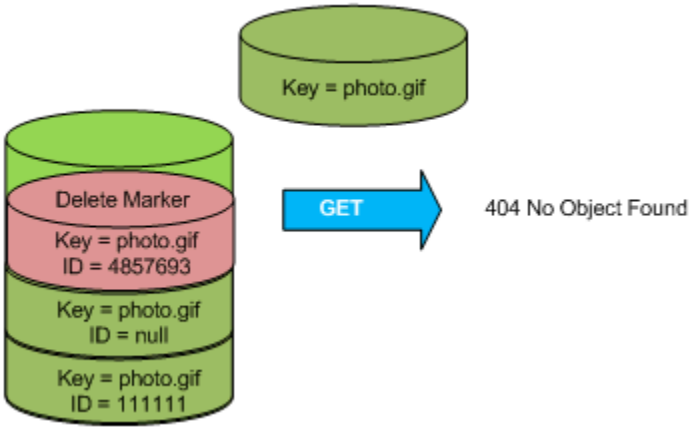
Amazon S3 のみが削除マーカーを作成できます。この削除マーカーは、バージョニングが有効なバケットまたは停止されたバケット内のオブジェクトに対して DELETE Object リクエストが送信された

びに作成されます。DELETE リクエストで指定されたオブジェクトは実際には削除されず、代わりに削除マーカーがオブジェクトの最新バージョンになります (オブジェクトのキーが削除マーカーのキーになります) 。オブジェクトを取得しようとしたときに、その最新バージョンが削除マーカーである場合、Amazon S3 は次のように応答します。

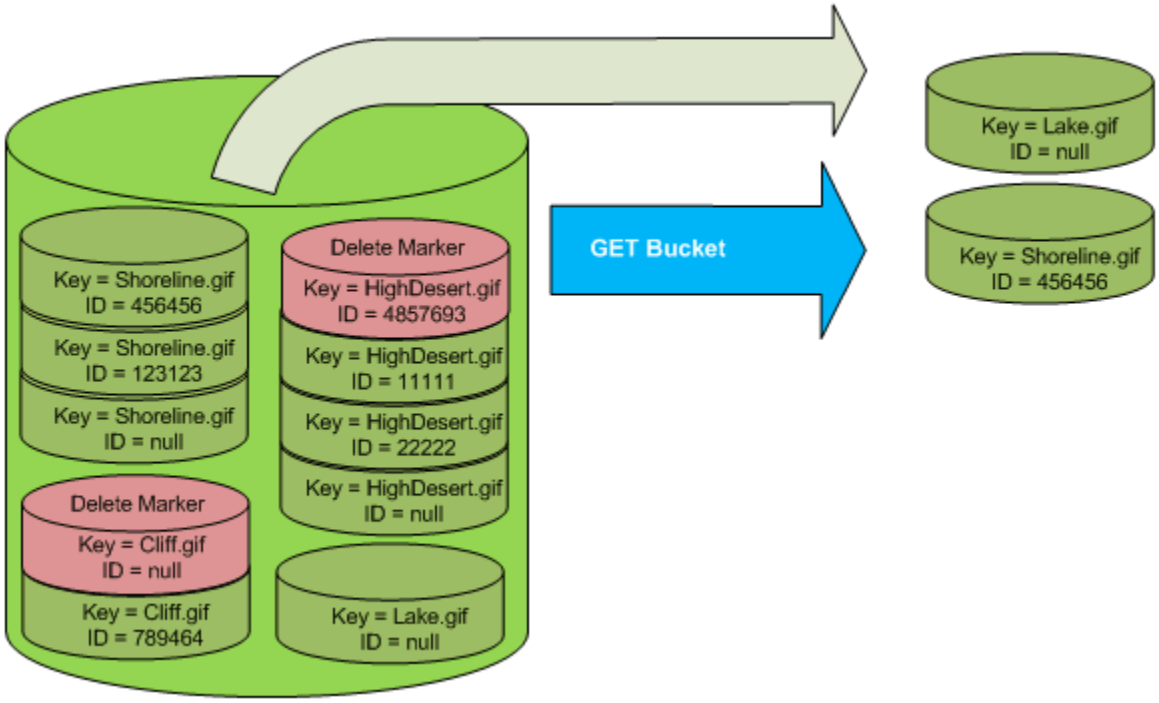
- 404 エラー (オブジェクトが見つからない)
- レスポンスヘッダー、x-amz-delete-marker: true

レスポンスヘッダーによって、アクセスされたオブジェクトが削除マーカーであったことがわかります。このレスポンスヘッダーが `false` を返すことはありません。値が `false` の場合は、Amazon S3 がレスポンスにこのレスポンスヘッダーを含めないためです。

次の図は、最新バージョンが削除マーカーであるオブジェクトに対するシンプルな GET が、エラー 404 「No Object Found」を返す方法を示しています。



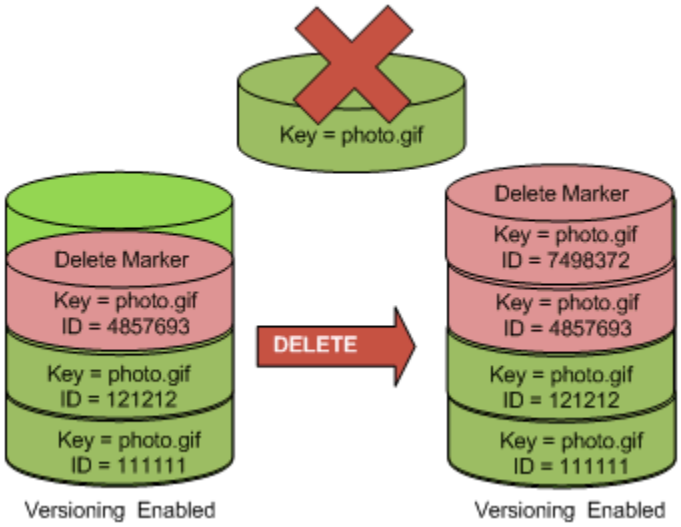
削除マーカー (およびオブジェクトのその他のバージョン) をリストするには、`GET Bucket versions` リクエストで *`versions`* サブリソースを使用するしかありません。シンプルな GET は、削除マーカーオブジェクトを取得しません。次の図は、GET Bucket リクエストが、最新バージョンが削除マーカーであるオブジェクトを返さないことを示しています。



削除マーカーの削除

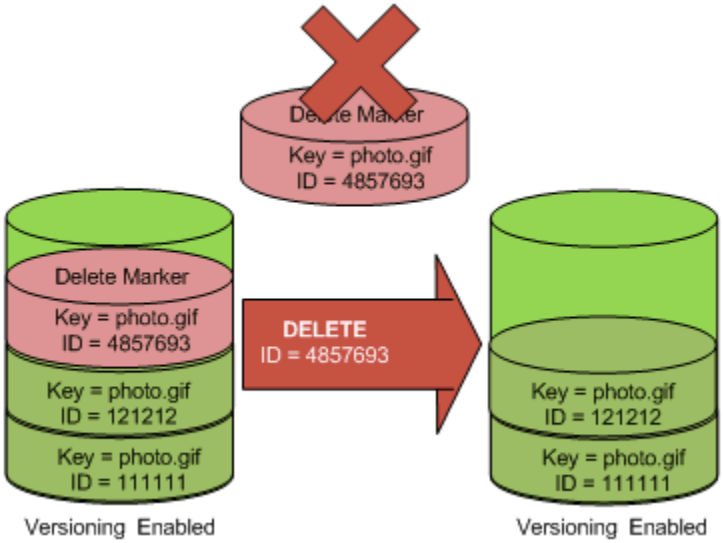
削除マーカーを削除するには、`DELETE Object versionId` リクエストにそのバージョン ID を含める必要があります。`DELETE` を使用して (削除マーカーのバージョン ID を指定せずに) 削除マーカーを削除する場合、Amazon S3 は、削除マーカーを削除せず、代わりに別の削除マーカーを挿入します。

次の図は、削除マーカーに対するシンプルな `DELETE` が何も削除せず、バケットに新しい削除マーカーを追加する方法を示しています。



バージョニングが有効なバケットでは、この新しい削除マーカーに固有のバージョン ID が付けられます。したがって、1 つのバケット内に同じオブジェクトの複数の削除マーカーが存在する場合があります。

削除マーカーを完全に削除するには、DELETE Object versionId リクエストにそのバージョン ID を含める必要があります。次の図は、DELETE Object versionId リクエストが削除マーカーを完全に削除する方法を示しています。バケット所有者のみが削除マーカーを完全に削除することができます。



削除マーカーを削除すると、シンプルな GET リクエストで、オブジェクトの最新バージョン (121212) が取得されます。

削除マーカーを完全に削除するには

1. *versionId* を、削除する削除マーカーのバージョン ID に設定します。
2. DELETE Object versionId リクエストを送信します。

Example 削除マーカーの削除

次の例では、photo.gif のバージョン 4857693 の削除マーカーを削除します。

```
DELETE /photo.gif?versionId=4857693 HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

削除マーカーを削除すると、レスポンスには以下が含められます。

```
204 NoContent
x-amz-version-id: versionID
x-amz-delete-marker: true
```

Note

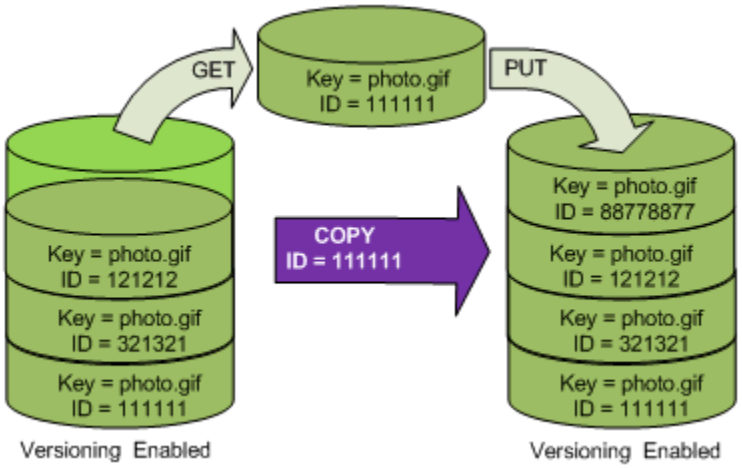
削除マーカーにより、Amazon S3 内のストレージに対して小額の課金が発生します。削除マーカーはキー名のサイズと等しくなります。例えば、キー「photo.gif」の削除マーカーは、バケットに 9 バイトのストレージを追加します。

以前のバージョンの復元

バージョニングの価値ある提案の 1 つに、オブジェクトの以前のバージョンを取得する機能があります。この機能を実行する方法は 2 つあります。

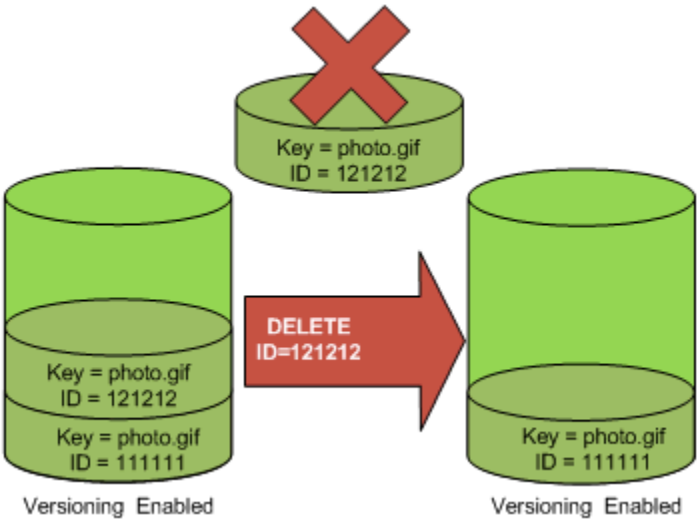
- オブジェクトの以前のバージョンを同じバケットにコピーする
  コピーされたオブジェクトはそのオブジェクトの最新バージョンになり、すべてのオブジェクトバージョンが維持されます。
- オブジェクトの最新バージョンを完全に削除する
  最新のオブジェクトバージョンを削除すると、事実上、以前のバージョンがそのオブジェクトの最新バージョンになります。

すべてのオブジェクトバージョンが維持されるため、オブジェクトの特定のバージョンをバケットにコピーすることにより、以前の任意のバージョンを最新バージョンにすることができます。次の図では、ソースオブジェクト（ID = 111111）が同じバケットにコピーされます。Amazon S3 が新しい ID（88778877）を指定し、それがオブジェクトの最新バージョンになります。したがって、バケットには元のオブジェクトバージョン（111111）とそのコピー（88778877）の両方が存在します。



以降の GET は、バージョン 88778877 を取得します。

次の図は、オブジェクトの最新バージョン（121212）を削除して、以前のバージョン（111111）を最新のオブジェクトとして残す方法を示しています。



以降の GET は、バージョン 111111 を取得します。

バージョニングされたオブジェクトのアクセス許可および ACL

アクセス許可はバージョンレベルで設定されます。各バージョンには個々のオブジェクト所有者 (バージョンを PUT したユーザー) がいます。したがって、同じオブジェクトの異なるバージョンに対して、異なるアクセス許可を設定することができます。これを行うには、`PUT Object versionId acl` リクエストでアクセス許可を設定するオブジェクトのバージョン ID を指定する必要があります。ACL の使用の詳細と手順については、「アクセスコントロール (p. 294)」を参照してください。

Example オブジェクトバージョンに対するアクセス許可の設定

次のリクエストは、キー my-image.jpg、バージョン ID 3HL4kqtJvjVBH40Nrjfkd に対する、被付与者 BucketOwner@amazon.com のアクセス許可を FULL_CONTROL に設定します。

```
PUT /my-image.jpg?acl&versionId=3HL4kqtJvjVBH40Nrjfkd HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
Content-Length: 124

<AccessControlPolicy>
  <Owner>
    <ID>75cc57f09aa0c8caeab4f8c24e99d10f8e7faeebf76c078efc7c6caea54ba06a</ID>
    <DisplayName>mtd@amazon.com</DisplayName>
  </Owner>
  <AccessControlList>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="CanonicalUser">
       <ID>a9a7b886d6fd24a52fe8ca5bef65f89a64e0193f23000e241bf9b1c61be666e9</ID>

        <DisplayName>BucketOwner@amazon.com</DisplayName>
      </Grantee>
      <Permission>FULL_CONTROL</Permission>
    </Grant>
  </AccessControlList>
  </AccessControlPolicy>
```

同様に、特定のオブジェクトバージョンに対するアクセス許可を取得するには、GET Object versionId acl リクエストでそのバージョン ID を指定する必要があります。デフォルトでは GET Object acl はオブジェクトの最新バージョンに対するアクセス許可を返すため、バージョン ID を含める必要があります。

Example 指定したオブジェクトバージョンに対するアクセス許可の取得

次の例では、Amazon S3 はキー my-image.jpg、バージョン ID DVBH40Nr8X8gUMLUo に対するアクセス許可を返します。

```
GET /my-image.jpg?versionId=DVBH40Nr8X8gUMLUo&acl HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU
```

詳細については、「GET Object ACL」を参照してください。

バージョニングが停止されたバケットの使用

Topics

バケット内で同じオブジェクトの新しいバージョンが生成されないようにするには、バージョニングを停止します。この処理は、バケット内で単一のバージョンのオブジェクトのみが必要な場合や、複数のバージョンに対して課金されないようにする場合に行います。

Note

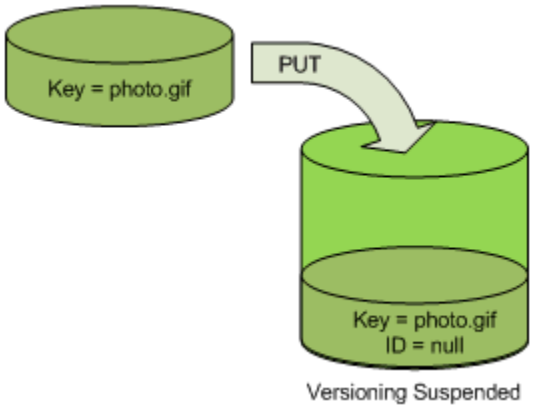
バケットでバージョニングが有効になっているときにオブジェクトが追加された場合、バージョニングが停止されたバケット内にバージョニングされたオブジェクトが存在することがあります。

次のセクションでは、バージョニングが停止された場合のバケットおよびオブジェクトの動作について説明します。

バージョニングが停止されたバケットへのオブジェクトの追加

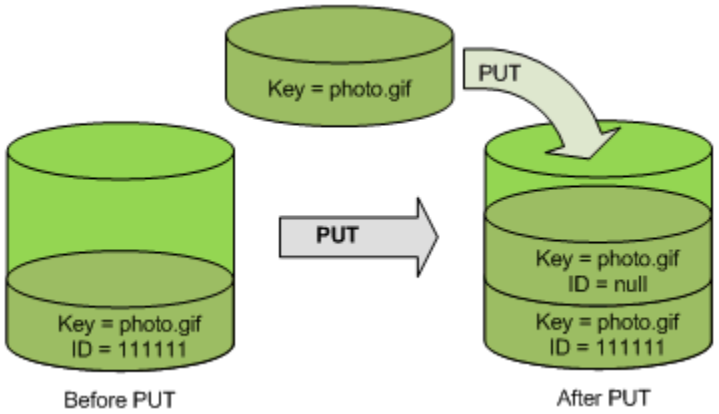
バケットでバージョニングを停止すると、それ以降、PUT、POST、または COPY を使用して、そのバケットに保存された後続のすべてのオブジェクトに、null バージョン ID が自動的に追加されます。

次の図は、バージョニングが停止されたバケットにオブジェクトが追加されたときに、Amazon S3 がそのオブジェクトにバージョン ID null を追加する方法を示しています。

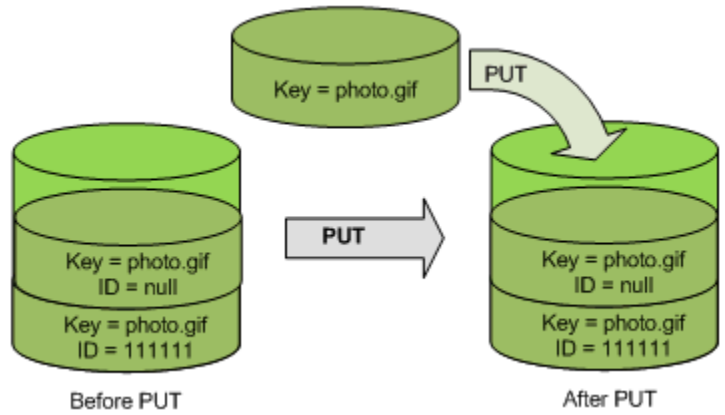


バケット内に null バージョンが既に存在しており、同じキーを持つ別のオブジェクトを追加した場合、元の null バージョンは、追加したオブジェクトによって上書きされます。

バケット内にバージョニングされたオブジェクトがある場合、PUT したバージョンはオブジェクトの最新バージョンになります。次の図は、バージョニングされたオブジェクトを含むバケットにオブジェクトを追加しても、バケット内に既に存在するオブジェクトを上書きしないことを示しています。この場合、バケット内にバージョン 111111 が既に存在しています。Amazon S3 は、追加されるオブジェクトに null のバージョン ID をアタッチし、バケット内に保存します。バージョン 111111 は上書きされません。



次の図に示すように、バケット内に null バージョンが既に存在する場合、null バージョンは上書きされます。



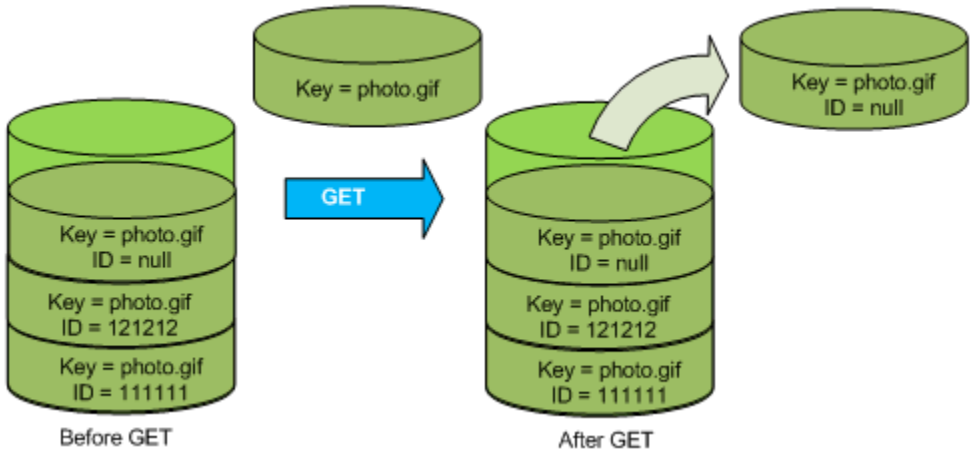
null バージョンのキーおよびバージョン ID (`null`) は PUT の前後で同じですが、バケット内に元々格納されていた null バージョンの内容は、バケット内に PUT されたオブジェクトの内容に置き換えられます。

バージョニングが停止されたバケットへのオブジェクトの追加

1	`PUT Bucket versioning` リクエストを使用して、バケットでバージョニングを停止します。詳細については、「PUT Bucket versioning」を参照してください。
2	`PUT`、`POST`、または `COPY` リクエストを送信して、バケットにオブジェクトを格納します。

バージョニングが停止されたバケットからのオブジェクトの取得

`GET Object` リクエストは、バケットでバージョニングを有効にしているかどうかにかかわらず、オブジェクトの最新バージョンを返します。次の図は、シンプルな `GET` がオブジェクトの最新バージョンを返す方法を示しています。



Example 最新バージョンの取得

次のリクエストは、photo.gif の最新バージョンを取得します。

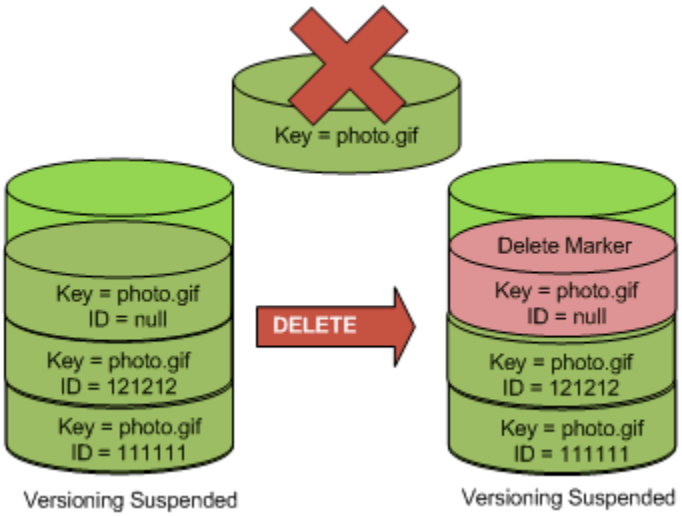
```
GET /photo.gif HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

バージョニングが停止されたバケットからのオブジェクトの削除

バージョニングが停止されている場合、DELETE リクエストは次のように動作します。

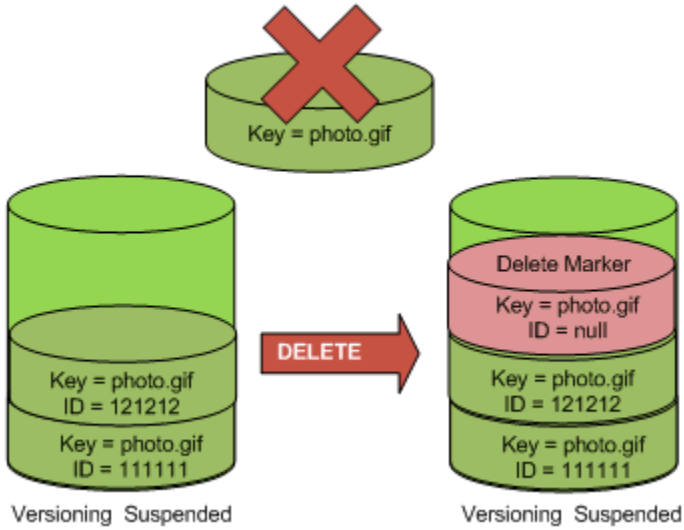
- バージョン ID が null のオブジェクトのみを削除できる
  バケット内にオブジェクトの null バージョンが存在しない場合は、何も削除しません。
- バケットに削除マーカーを挿入する

次の図は、シンプルな DELETE が null バージョンを削除し、Amazon S3 がその位置に、null のバージョン ID を持つ削除マーカーを挿入する方法を示しています。

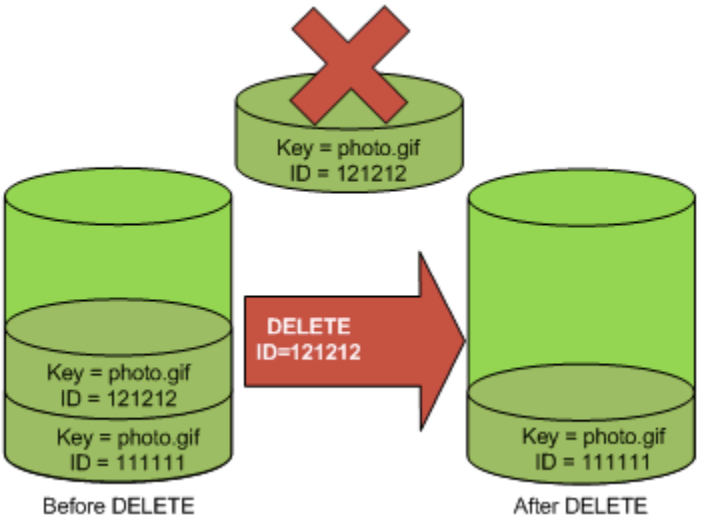


削除マーカーには内容がないため、削除マーカーに置き換えられるときに null バージョンの内容が失われることに注意してください。

次の図は、null バージョンが存在しないバケットを示しています。この場合、DELETE は何も削除せず、Amazon S3 が削除マーカーを挿入するだけです。



バージョニングが停止されたバケットでも、バケット所有者は指定したバージョンを完全に削除できます。次の図は、指定したオブジェクトバージョンを削除することによって、そのオブジェクトを完全に削除する方法を示しています。バケット所有者のみが、指定したオブジェクトバージョンを削除できます。


Key = photo.gif
ID = 121212
DELETE
ID=121212
Key = photo.gif
ID = 121212
Key = photo.gif
ID = 111111
Before DELETE
Key = photo.gif
ID = 111111
After DELETE

Amazon S3 での静的ウェブサイトのホスティング

Topics

静的ウェブサイトを Amazon S3 でホスティングできます。静的ウェブサイトでは、個々のウェブページの内容は静的コンテンツです。ほかに、クライアント側スクリプトが含まれていることもあります。対照的に、動的ウェブサイトはサーバー側処理に依存しており、例えばサーバー側スクリプト (PHP、JSP、ASP.NET など) が使用されます。サーバー側スクリプトは Amazon S3 ではサポートされていません。

静的ウェブサイトをホスティングするには、Amazon S3 バケットをウェブサイトホスティング用に設定してから、ウェブサイトのコンテンツをそのバケットにアップロードします。バケットのリージョン固有のウェブサイトエンドポイントで、そのウェブサイトにアクセスできるようになります。

```
<bucket-name>.s3-website-<AWS-region>.amazonaws.com
```

Amazon S3 のリージョン固有のウェブサイトエンドポイントのリストについては、「ウェブサイトエンドポイント (p. 412)」を参照してください。例えば、`examplebucket` という名前のバケットを米国東部リージョンで作成し、このバケットをウェブサイトとして設定したとします。次の例に示す URL を使用すると、このウェブサイトのコンテンツにアクセスできます。

- この URL にアクセスすると、ウェブサイトに対して設定されたデフォルトのインデックスドキュメントが返されます。

  ```
  http://examplebucket.s3-website-us-east-1.amazonaws.com/
  ```

- この URL は、photo.jpg というオブジェクトをリクエストするものです。このオブジェクトは、バケットのルートレベルに格納されています。

  ```
  http://examplebucket.s3-website-us-east-1.amazonaws.com/photo.jpg
  ```

- この URL は、バケット内の `docs/doc1.html` オブジェクトをリクエストするものです。

```
http://examplebucket.s3-website-us-east-1.amazonaws.com/docs/doc1.html
```

自分が所有するドメインの使用

ウェブサイトへのアクセスに Amazon S3 ウェブサイトエンドポイントを使用する代わりに、`example.com` などの独自のドメインを使用してコンテンツを配信できます。Amazon S3 を Amazon Route 53 と組み合わせると、ウェブサイトをルートドメインでホスティングできるようになります。例えば、ルートドメインが `example.com` で、ウェブサイトを Amazon S3 でホスティングする場合、ウェブサイト閲覧者は「`http://www.example.com`」と入力せずに「`http://example.com`」と入力するだけで、ブラウザからそのウェブサイトにアクセスできます。チュートリアル例については、「例: カスタムドメインを使用して静的ウェブサイトをセットアップする (p. 428)」を参照してください。

バケットをウェブサイトホスティング用に設定するには、ウェブサイト設定をバケットに追加します。詳細については、「バケットをウェブサイトホスティング用に設定 (p. 414)」を参照してください。

ウェブサイトエンドポイント

Topics

バケットをウェブサイトホスティング用に設定すると、ウェブサイトはリージョン固有のウェブサイトエンドポイントを介してアクセスできるようになります。ウェブサイトエンドポイントは、REST API リクエストを送信するエンドポイントとは別のものです。エンドポイントの詳細については、「エンドポイントのリクエスト (p. 13)」を参照してください。

Amazon S3 ウェブサイトエンドポイントの一般的な形式は次のとおりです。

```
bucket-name.s3-website-region.amazonaws.com
```

例えば、`example-bucket` という名前のバケットが米国スタンダードリージョンに存在する場合、ウェブサイトには次の Amazon S3 ウェブサイトエンドポイントでアクセスできます。

```
http://example-bucket.s3-website-us-east-1.amazonaws.com/
```

次の表は、Amazon S3 リージョンおよび対応するウェブサイトエンドポイントをまとめたものです。

リージョン	ウェブサイトエンドポイント
米国スタンダード	.s3-website-us-east-1amazonaws.com
米国西部 (オレゴン) リージョン	.s3-website-us-west-2amazonaws.com
米国西部 (北カリフォルニア) リージョン	.s3-website-us-west-1amazonaws.com

リージョン	ウェブサイトエンドポイント
欧州 (アイルランド) リージョン	.s3-website-eu-west-1amazonaws.com
アジアパシフィック (シンガポール) リージョン	.s3-website-ap-southeast-1amazonaws.com
アジアパシフィック (シドニー) リージョン	.s3-website-ap-southeast-2amazonaws.com
アジアパシフィック (東京) リージョン	.s3-website-ap-northeast-1amazonaws.com
南米 (サンパウロ) リージョン	.s3-website-sa-east-1amazonaws.com

ウェブサイトエンドポイントにあるコンテンツにアクセスできるようにするには、すべてのコンテンツがパブリックに読み取り可能となっている必要があります。このようにするには、バケットポリシーを使用するか、オブジェクトに対する ACL を使用して、必要なアクセス許可を付与します。

Note

「リクエスタ支払い」バケットや DevPay バケットに、ウェブサイトエンドポイントを介してアクセスすることはできません。このようなバケットに対してリクエストを行うと、レスポンスは 403 Access Denied となります。詳細については、「リクエスタ支払いバケット (p. 101)」を参照してください。

登録済みのドメインがある場合は、Amazon S3 ウェブサイトエンドポイントを指し示す DNS CNAME エントリを追加できます。例えば、ドメイン www.example-bucket.com が登録済みの場合は、バケット www.example-bucket.com を作成し、www.example-bucket.com.s3-website-<region>.amazonaws.com を指し示す DNS CNAME エントリを追加できます。http://www.example-bucket.com へのリクエストはすべて、www.example-bucket.com.s3-website-<region>.amazonaws.com にルーティングされます。詳細については、「バケットの仮想ホスティング (p. 63)」を参照してください。

Amazon ウェブサイトと REST API エンドポイントの主な違い

ウェブサイトエンドポイントは、ウェブブラウザを介してアクセスするように最適化されています。次の表は、Amazon REST API エンドポイントとウェブサイトエンドポイントの主な違いについてまとめたものです。

主な違い	REST API エンドポイント	ウェブサイトエンドポイント
サポートされるリクエスト	バケットおよびオブジェクトのすべてのオペレーションをサポートします。	オブジェクトに対しては GET リクエストと HEAD リクエストのみサポートします。

主な違い	REST API エンドポイント	ウェブサイトエンドポイント
バケットのルートでの GET リクエストと HEAD リクエストにレスポンスします。	バケット内のオブジェクトキーのリストを返します。	ウェブサイト設定の中で指定されているインデックスドキュメントを返します。
エラーメッセージの処理	XML 形式のエラーレスポンスを返します。	HTML ドキュメントを返します。
アクセスコントロール	パブリックコンテンツとプライベートコンテンツの両方をサポートします。	公開で読み取り可能なコンテンツのみをサポートします。
リダイレクトのサポート	該当しません	オブジェクトレベルとバケットレベルの両方のリダイレクトをサポートします。

バケットをウェブサイトホスティング用に設定

Topics

概要

バケットを静的ウェブサイトのホスティング用に設定するには、ウェブサイト設定をバケットに追加します。この設定には、次に示す情報が含まれます。

- インデックスドキュメント

  ユーザーが http://example.com のような URL を入力した場合、そのユーザーは特定のページを要求してはいません。この場合は、要求されたウェブサイトコンテンツが格納されているディレクトリの、デフォルトページがウェブサーバーから配信されます。このデフォルトページは「インデックスドキュメント」と呼ばれ、ほとんどの場合は index.html という名前が付けられます。バケットをウェブサイトホスティング用に設定するときは、インデックスドキュメントを指定する必要があります。Amazon S3 からこのインデックスドキュメントが返されるのは、ルートドメインまたはサブフォルダに対するリクエストが行われたときです。詳細については、「インデックスドキュメントとフォルダ (p. 420)」を参照してください。
- エラードキュメント

  エラーが発生した場合は、Amazon S3 から HTML エラードキュメントが返されます。4XX クラスのエラーの場合は、ウェブサイト独自のエラードキュメントを返すこともでき、このドキュメントの中で、ウェブサイトのユーザー向けに追加のガイダンスを記述することができます。詳細については、「カスタムエラードキュメントのサポート (p. 421)」を参照してください。
- すべてのリクエストをリダイレクト

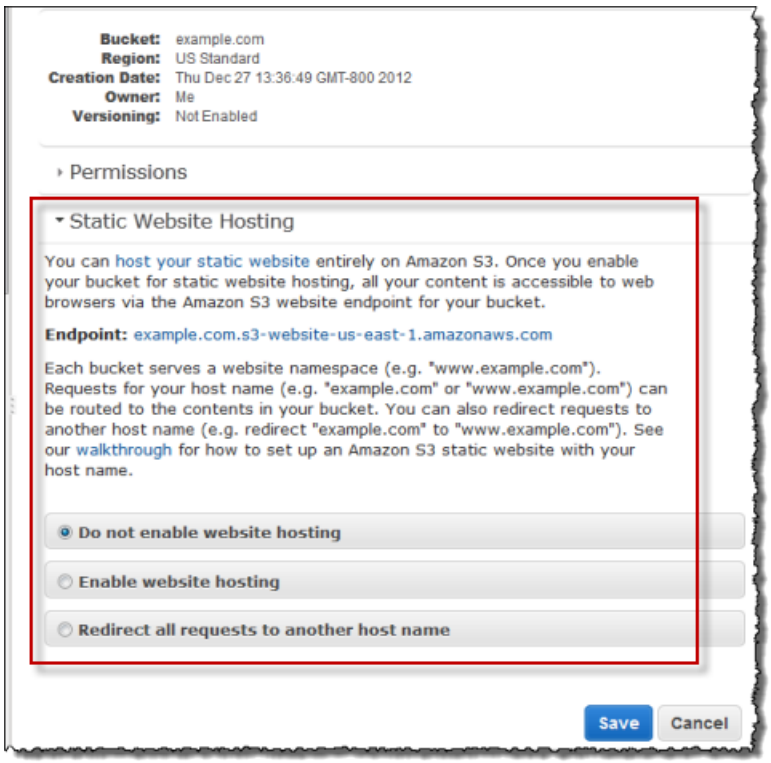
  ルートドメインが `example.com` の場合に、`http://example.com` と `http://www.example.com` のどちらに対するリクエストにも応答できるようにするには、2 つのバケット `example.com` と `www.example.com` を作成し、ウェブサイトのコンテンツを一方のバケット、例えば `example.com`

内のみで維持し、他方のバケットはリクエストをすべて example.com バケットにリダイレクトするように設定します。

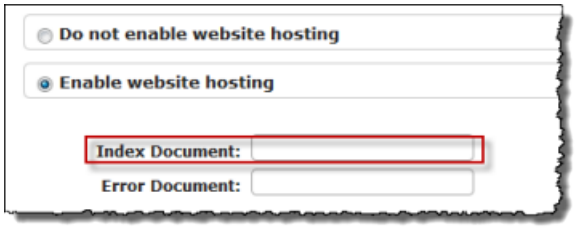
- 高度な条件付きリダイレクト

  条件に応じてリクエストのルーティング先を変えることができます。この条件として使用できるのは、リクエストの中の特定のオブジェクトキー名またはプレフィックス、またはレスポンスコードです。例えば、バケット内のあるオブジェクトを削除する、または名前を変更する場合は、リクエストを別のオブジェクトにリダイレクトするルーティングルールを追加します。例えば、あるフォルダを使用不可能にする場合は、リクエストを別のページにリダイレクトするルーティングルールを作成し、リダイレクト先のページで、フォルダが使用できなくなった理由を説明します。ルーティングルールは、エラー状態を処理するために追加することもできます。その場合は、エラーを発生させたリクエストを別のドメインにリダイレクトし、そのドメインでエラーを処理します。

バケットウェブサイトの設定を管理するには、Amazon S3 コンソールを使用します。コンソールの、バケットの [Properties] パネルで、ウェブサイト設定を指定します。

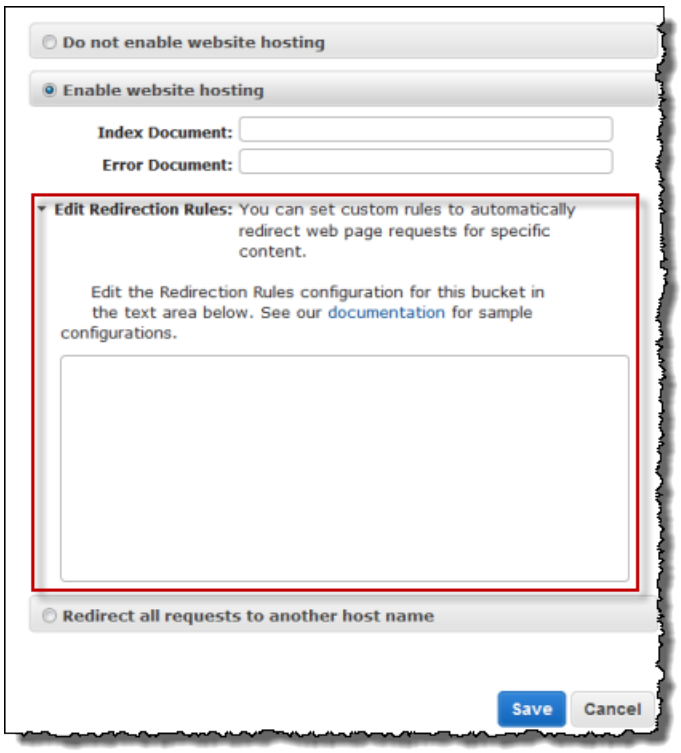


静的ウェブサイトを Amazon S3 でホスティングするには、インデックスドキュメント名の指定だけが必要です。



バケットのウェブサイトエンドポイントへのリクエストをすべて別のホストにリダイレクトするには、ホスト名の指定だけが必要です。

ただし、バケットをウェブサイトホスティング用に設定するときは、高度なリダイレクトルールを指定することもできます。



ルールは XML を使用して記述します。次のセクションでは、リダイレクトルールの一般的な構文と、指定の例を示します。

ルーティングルールを指定するための構文

ウェブサイト設定でルーティングルールを定義するための一般的な構文を次に示します。

```
<RoutingRules> =
    <RoutingRules>
         <RoutingRule>...</RoutingRule>
         [<RoutingRule>...</RoutingRule>
```

```
        ...]
    </RoutingRules>

<RoutingRule> =
   <RoutingRule>
      [ <Condition>...</Condition> ]
      <Redirect>...</Redirect>
   </RoutingRule>

<Condition> =
   <Condition>
      [ <KeyPrefixEquals>...</KeyPrefixEquals> ]
      [ <HttpErrorCodeReturnedEquals>...</HttpErrorCodeReturnedEquals> ]
   </Condition>
    Note: <Condition> must has at least one child element.

<Redirect> =
   <Redirect>
      [ <HostName>...</HostName> ]
      [ <Protocol>...</Protocol> ]
      [ <ReplaceKeyPrefixWith>...</ReplaceKeyPrefixWith>  ]
      [ <ReplaceKeyWith>...</ReplaceKeyWith> ]
      [ <HttpRedirectCode>...</HttpRedirectCode> ]
   </Redirect>
    Note: <Redirect> must has at least one child element.
          Also, you can have either ReplaceKeyPrefix with or ReplaceKeyWith,

          but not both.
```

次の表は、ルーティングルールの要素をまとめたものです。

名前	説明
RoutingRules	RoutingRule 要素のコレクションを入れるコンテナ。
RoutingRule	条件とリダイレクトを明示したルール。この条件が満たされたときにこのリダイレクトが適用されます。 条件: コンテナである *RoutingRules* の中には、少なくとも 1 つのルールが存在する必要があります。
Condition	条件を記述するためのコンテナ。この条件が満たされているときに、指定されたリダイレクトが適用されます。ルーティングルールの中に条件がない場合は、そのルールはすべてのリクエストに適用されます。
KeyPrefixEquals	このオブジェクトキー名プレフィックスに対するリクエストがリダイレクトされます。 HttpErrorCodeReturnedEquals が指定されていない場合、*KeyPrefixEquals* は必須です。*KeyPrefixEquals* と HttpErrorCodeReturnedEquals の両方が指定されている場合は、両方が真である場合に、条件が満たされていると見なされます。

名前	説明
HttpErrorCodeReturnedEquals	この HTTP エラーコードに一致している場合にのみ、リダイレクトが適用されます。エラーが発生した場合に、エラーコードがこの値に等しい場合は、指定されたリダイレクトが適用されます。 KeyPrefixEquals が指定されていない場合、*HttpErrorCodeReturnedEquals* は必須です。*KeyPrefixEquals* と HttpErrorCodeReturnedEquals の両方が指定されている場合は、両方が真である場合に、条件が満たされていると見なされます。
Redirect	コンテナ要素。リクエストのリダイレクトに関する指示を表すのに使用されます。リクエストを別のホストや別のページにリダイレクトすることも、別のプロトコルの使用を指定することもできます。1 つの RoutingRule に対して、1 つの Redirect 要素が存在する必要があります。1 つの Redirect 要素の中に、Protocol、HostName、ReplaceKeyPrefixWith、ReplaceKeywith、HttpRedirectCode の要素のうち 1 つ以上が存在する必要があります。
Protocol	レスポンスとして返される Location ヘッダーで使用されるプロトコル (http または https) 。 *Protocol* は、同レベルの要素の 1 つが指定されている場合は省略可能です。
HostName	レスポンスとして返される Location ヘッダーで使用されるホスト名。 *HostName* は、同レベルの要素の 1 つが指定されている場合は省略可能です。
ReplaceKeyPrefixWith	リダイレクトリクエストの中の KeyPrefixEquals の値を置き換えるオブジェクトキー名プレフィックス。 *ReplaceKeyPrefixWith* は、同レベルの要素の 1 つが指定されている場合は省略可能です。これを指定できるのは、ReplaceKeyWith が指定されていない場合のみです。
ReplaceKeyWith	レスポンスでとして返される Location ヘッダーで使用されるオブジェクトキー。 *ReplaceKeyWith* は、同レベルの要素の 1 つが指定されている場合は省略可能です。これを指定できるのは、ReplaceKeyPrefixWith が指定されていない場合のみです。
HttpRedirectCode	レスポンスとして返される Location ヘッダーで使用される HTTP リダイレクトコード。 *HttpRedirectCode* は、同レベルの要素の 1 つが指定されている場合は省略可能です。

例をいくつか次に示します。

Example 1: キープレフィックスの名前を変更した後にリダイレクトする

バケットに次のオブジェクトが含まれているとします。

index.html

docs/article1.html

docs/article2.html

ここでフォルダ名を docs/ から documents/ に変更することにしました。この変更を行った後は、プレフィックス /docs に対するリクエストを documents/ にリダイレクトする必要があります。例えば、docs/article1.html に対するリクエストを documents/article1.html にリダイレクトする必要があります。

この場合、次のルーティングルールをウェブサイト設定に追加します。

```
<RoutingRules>
  <RoutingRule>
  <Condition>
    <KeyPrefixEquals>docs/</KeyPrefixEquals>
  </Condition>
  <Redirect>
    <ReplaceKeyPrefixWith>documents/</ReplaceKeyPrefixWith>
  </Redirect>
  </RoutingRule>
</RoutingRules>
```

Example 2: 削除されたフォルダに対するリクエストをページにリダイレクトする

images/ フォルダ (つまり、キープレフィックスが images/ のすべてのオブジェクト) を削除するとします。ルーティングルールを追加し、その中で、キープレフィックス images/ を持つオブジェクトに対するリクエストをすべて folderdeleted.html というページにリダイレクトするように設定します。

```
<RoutingRules>
  <RoutingRule>
  <Condition>
     <KeyPrefixEquals>images/</KeyPrefixEquals>
  </Condition>
  <Redirect>
    <ReplaceKeyWith>folderdeleted.html</ReplaceKeyWith>
  </Redirect>
  </RoutingRule>
</RoutingRules>
```

Example 3: HTTP エラーの場合にリダイレクトする

リクエストされたオブジェクトが見つからないときに、リクエストを特定の Amazon EC2 インスタンスにリダイレクトするとします。リダイレクトルールを追加し、その中で、HTTP ステータスコード 404 (Not Found) が返されたときに特定の EC2 インスタンス (このインスタンスでリクエストを処理する) にリダイレクトするように設定します。次に示す例では、オブジェクトキープレフィックス `report-404/` もリダイレクトに挿入します。例えば、ExamplePage.html というページをリクエストした結果が HTTP 404 エラーとなったときは、指定の EC2 インスタンスにある `report-404/ExamplePage.html` というページにリクエストをリダイレクトします。ルーティングルールが何も設定されていない場合に HTTP エラー 404 が発生したときは、設定内で指定されているエラードキュメントが返されます。

```
<RoutingRules>
  <RoutingRule>
  <Condition>
    <HttpErrorCodeReturnedEquals>404</HttpErrorCodeReturnedEquals >
  </Condition>
  <Redirect>
    <HostName>ec2-11-22-333-44.compute-1.amazonaws.com</HostName>
    <ReplaceKeyPrefixWith>report-404/</ReplaceKeyPrefixWith>
  </Redirect>
  </RoutingRule>
</RoutingRules>
```

インデックスドキュメントのサポート

インデックスドキュメントは、リクエストがウェブサイトのルートまたはサブフォルダに対して行われた場合に返されるウェブページです。例えば、ユーザーがブラウザに `http://www.example.com` と入力した場合は、このユーザーは特定のページをリクエストしてはいません。この場合、Amazon S3 は、インデックスドキュメントを提供します。これがデフォルトページとして指定されていることもあります。

バケットをウェブサイトとして設定するときは、インデックスドキュメントの名前を指定する必要があります。この名前のオブジェクトをアップロードして、パブリックに読み取り可能となるように設定する必要があります。

ルートレベル URL の末尾のスラッシュは省略可能です。例えば、`index.html` をウェブサイトのインデックスドキュメントとして設定した場合は、次の URL のどちらからも `index.html` が返されます。

```
http://example-bucket.s3-website-region.amazonaws.com/
http://example-bucket.s3-website-region.amazonaws.com
```

インデックスドキュメントとフォルダ

Amazon S3 では、バケットとはオブジェクトが格納されるフラットなコンテナです。つまり、コンピュータのファイルシステムとは異なり、階層構造ではありません。論理的な階層を作成するには、フォルダ構造を意味する名前をオブジェクトキーに付けます。例えば、3 つのオブジェクトと以下のキー名を含むバケットがあるとします。

```
sample1.jpg

photos/2006/Jan/sample2.jpg

photos/2006/Feb/sample3.jpg
```

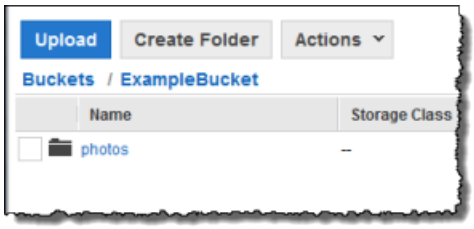
これらは、物理的な階層構造で格納されているわけではありませんが、キー名から次のような論理構造を推測できます。

`sample1.jpg` オブジェクトはバケットのルートにあります。

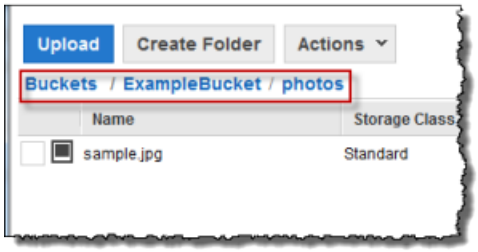
`sample2.jpg` オブジェクトは `photos/2006/Jan` サブフォルダにあります。

`sample3.jpg` オブジェクトは `photos/2006/Feb` サブフォルダにあります。

Amazon S3 コンソールがサポートするフォルダ概念は、オブジェクトキー名に基づいています。前の例からの続きとして、コンソールには `ExampleBucket` が表示されており、この中に `photos` フォルダがあります。



バケットまたはバケット内の `photos` フォルダにオブジェクトをアップロードできます。オブジェクト `sample.jpg` をバケットに追加する場合、キー名は `sample.jpg` です。オブジェクトを `photos` フォルダにアップロードする場合、オブジェクトキー名は `photos/sample.jpg` です。



このようなフォルダ構造をバケット内に作成する場合は、各レベルにインデックスドキュメントが存在している必要があります。ユーザーが指定した URL がフォルダルックアップに似ている場合は、末尾のスラッシュの有無によって、ウェブサイトエンドポイントの動作が決まります。例えば、末尾にスラッシュのある次の URL は `photos/index.html` インデックスドキュメントを返します。

```
http://example-bucket.s3-website-region.amazonaws.com/photos/
```

ただし、前の URL から末尾のスラッシュを除外した場合、Amazon S3 はまずバケット内のオブジェクト `photos` を検索します。`photos` オブジェクトが見つからない場合、インデックスドキュメント `photos/index.html` を検索します。見つかった場合、Amazon S3 は `302 Found` メッセージを返し、`photos/` キーを指し示します。それ以降の `photos/` に対するリクエストについては、Amazon S3 は `photos/index.html` を返します。インデックスドキュメントが見つからない場合は、エラーを返します。

カスタムエラードキュメントのサポート

次の表は、エラーの発生時に Amazon S3 が返す HTTP レスポンスコードのサブセットをまとめたものです。

HTTP エラーコード	説明
301 Moved Permanently	ユーザーが Amazon S3 ウェブサイトエンドポイント（`http://s3-website-<region>.amazonaws.com/`）に直接リクエストを送信した場合、Amazon S3 は [301 Moved Permanently] レスポンスを返し、それらのリクエストを `http://aws.amazon.com/s3` にリダイレクトします。
[302 Found]	Amazon S3 が末尾にスラッシュのないキー x に対するリクエスト、`http://<bucket>.s3-website-<region>.amazonaws.com/x` を受け取った場合、まずキー名 x のオブジェクトを検索します。オブジェクトが見つからない場合、Amazon S3 はリクエストがサブフォルダ x に対するものであると判断し、末尾にスラッシュを追加して、リクエストをリダイレクトし、[302 Found] を返します。
[304 Not Modified]	Amazon S3 は、リクエストヘッダー `If-Modified-Since`、`If-Unmodified-Since`、`If-Match`、`If-None-Match` を使用して、リクエストされたオブジェクトがクライアントで保持されているキャッシュされたコピーと同じであるかどうかを判断します。オブジェクトが同じである場合、ウェブサイトエンドポイントは [304 Not Modified] レスポンスを返します。
[400 Malformed Request]	ユーザーが誤ったリージョンエンドポイント経由でバケットにアクセスを試みると、ウェブサイトエンドポイントは [400 Malformed Request] で応答します。
[403 Forbidden]	ユーザーのリクエストが公開で読み取ることのできないオブジェクトに変換する場合、ウェブサイトエンドポイントは [403 Forbidden] で応答します。オブジェクト所有者は、バケットポリシーまたは ACL を使用して、オブジェクトを公開する必要があります。
[404 Not Found]	ウェブサイトエンドポイントは、以下の理由により [404 Not Found] で応答します。 • ウェブサイト URL が存在しないオブジェクトキーを参照しているものと、Amazon S3 が判断する • リクエストが存在しないインデックスドキュメントに対するものであると、Amazon が推論する • URL に指定されたバケットが存在しない • URL に指定されたバケットは存在するが、ウェブサイトとして設定されていない [404 Not Found] で返すドキュメントをカスタマイズすることができます（カスタムドキュメントの作成）。必ず、このドキュメントをウェブサイトとして設定されたバケットにアップロードすると共に、このドキュメントを使用するようにウェブサイトホスティング設定を行ってください。 Amazon S3 がオブジェクトまたはインデックスドキュメントのリクエストとして URL をどのように解釈するかについては、「インデックスドキュメントのサポート (p. 420)」を参照してください。
[500 Service Error]	内部サーバーエラーが発生した場合、ウェブサイトエンドポイントは [500 Service Error] で応答します。
[503 Service Unavailable]	Amazon S3 がユーザーのリクエスト頻度を減らす必要があると判断した場合に、ウェブサイトエンドポイントは [503 Service Unavailable] で応答します。

これらの各エラーに対し、Amazon S3 は、[403 Forbidden] レスポンスで返される以下のサンプル HTML に示すような定義済みの HTML を返します。

オプションで、ユーザーにわかりやすいエラーメッセージとヘルプの追加情報を記載したカスタムエラードキュメントを提供できます。このカスタムエラードキュメントは、バケットにウェブサイト設定を追加する作業の一環として提供します。Amazon S3 は HTTP 4XX クラスのエラーコードに対してのみ、カスタムエラードキュメントを返します。

エラードキュメントとブラウザの動作

エラーが発生した場合は、Amazon S3 から HTML エラードキュメントが返されます。ウェブサイトにカスタムエラードキュメントを設定している場合、Amazon S3 からそのエラードキュメントが返されます。ただし、エラーが発生した場合、一部のブラウザは独自のエラーメッセージを表示し、Amazon S3 が返すエラードキュメントを無視することがあります。例えば、HTTP 404 Not Found エラーが発生した場合、Chrome は独自のエラーを表示し、Amazon S3 が返すエラードキュメントを無視することがあります。

ウェブページリダイレクトの設定

Amazon S3 バケットがウェブサイトホスティング用に設定されている場合は、特定のオブジェクトに対するリクエストを、同じバケット内の別のオブジェクトか外部 URL にリダイレクトできます。リダイレクトを設定するには、オブジェクトメタデータに `x-amz-website-redirect-location` プロパティを追加します。これで、そのオブジェクトは 301 リダイレクトとして解釈されるようになります。リクエストを別のオブジェクトにリダイレクトするには、リダイレクト場所をターゲットオブジェクトのキーに設定します。リクエストを外部 URL にリダイレクトするには、リダイレクト場所を目的の URL に設定します。オブジェクトメタデータの詳細については、「システム定義のメタデータ (p. 108)」を参照してください。

ウェブサイトホスティング用に設定されたバケットには、ウェブサイトエンドポイントと REST エンドポイントの両方があります。リクエストされたページが 301 リダイレクトとして設定されている場合の結果は、リクエストのエンドポイントに応じて、次のようになると考えられます。

- リージョン固有のウェブサイトエンドポイント – Amazon S3 は、`x-amz-website-redirect-location` プロパティの値に従ってページリクエストをリダイレクトします。
- REST エンドポイント – Amazon S3 はページリクエストをリダイレクトしません。リクエストされたオブジェクトを返します。

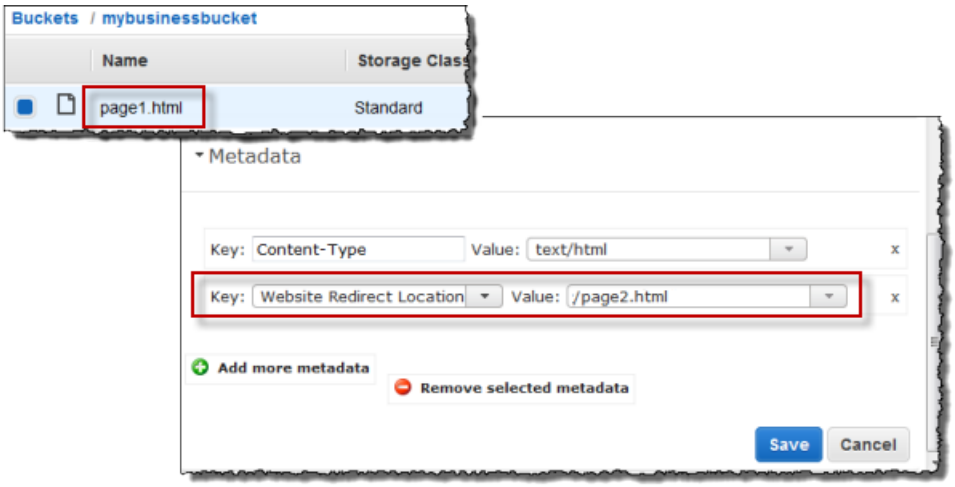
エンドポイントの詳細については、「Amazon ウェブサイトと REST API エンドポイントの主な違い (p. 413)」を参照してください。

ページリダイレクトの設定は、Amazon S3 コンソールから行うことも、Amazon S3 REST API を使用して行うこともできます。

Amazon S3 コンソールでのページリダイレクトのサポート

Amazon S3 コンソールを使用して、オブジェクトのメタデータでウェブサイトのリダイレクト場所を設定できます。ページリダイレクトを設定するときに、ソースオブジェクトコンテンツを維持することも、削除することもできます。例えば、バケットに page1.html オブジェクトがあるとします。このページに対するリクエストをすべて別のオブジェクト page2.html にリダイレクトするには、次のいずれかを行います。

- `page1.html` オブジェクトのコンテンツを維持してページリクエストのリダイレクトのみを行う場合は、`page1.html` の [Properties] の [Metadata] タブをクリックします。次の例に示すように、[Website Redirect Location] をメタデータに追加し、その値を `/page2.html` に設定します。値の `/` というプレフィックスは必須です.



  値を外部 URL に設定することもできます (例: `http://www.example.com`) 。

- page1.html オブジェクトのコンテンツを削除し、リクエストをリダイレクトするには、同じキー page1.html で、0 バイトの新しいオブジェクトをアップロードして、既存のオブジェクトを置き換え、アップロードプロセスで、page1.html の `Website Redirect Location` を指定できます。オブジェクトのアップロードについては、『Amazon Simple Storage Service Console User Guide』の「*Uploading Objects into Amazon S3*」を参照してください。

REST API からのページリダイレクトの設定

次の Amazon S3 API アクションでは、リクエストで `x-amz-website-redirect-location` ヘッダーを使用できます。Amazon S3 により、オブジェクトメタデータのヘッダー値として `x-amz-website-redirect-location` が格納されます。

- PUT Object
- Initiate Multipart Upload
- POST Object
- PUT Object – Copy

ページリダイレクトを設定するときに、オブジェクトのコンテンツを維持することも、削除することもできます。例えば、バケットに `page1.html` というオブジェクトがあるとします。

- `page1.html` のコンテンツを維持し、ページリクエストのリダイレクトのみを行う場合は、PUT Object – Copy リクエストを送信して新しい `page1.html` オブジェクトを作成し、ソースとして既

存の page1.html オブジェクトを使用するように設定します。リクエストに、x-amz-website-redirect-location ヘッダーを設定します。リクエストが完了すると、元のページのコンテンツは変更されないままで、ページへのすべてのリクエストは指定されたリダイレクト場所に Amazon S3 によりリダイレクトされます。

- page1.html オブジェクトのコンテンツを削除してこのページに対するリクエストをリダイレクトするには、PUT Object リクエストを送信して 0 バイトのオブジェクトをアップロードします。このオブジェクトには、同じオブジェクトキー page1.html を付けます。この PUT リクエストの中で、page1.html の x-amz-website-redirect-location を新しいオブジェクトに設定します。このリクエストが完了すると、page1.html にはコンテンツがない状態になり、リクエストはすべて、x-amz-website-redirect-location で指定された場所にリダイレクトされます。

GET Object アクションを使用して、他のオブジェクトメタデータと一緒にオブジェクトを取得すると、Amazon S3 はレスポンスで x-amz-website-redirect-location ヘッダーを返します。

ウェブサイトアクセスに必要なアクセス許可

バケットをウェブサイトとして設定するときは、そのサイトで配信するオブジェクトを、パブリックに読み取り可能となるように設定する必要があります。このようにするには、全員に s3:GetObject の実行アクセス許可を付与するバケットポリシーを作成します。ウェブサイトエンドポイント上に、ユーザーがリクエストしたオブジェクトが存在しない場合は、Amazon S3 から HTTP レスポンスコード 404 (Not Found) が返されます。オブジェクトが存在するが、そのオブジェクトの読み取りアクセス許可が付与されていない場合は、ウェブサイトエンドポイントから HTTP レスポンスコード 403 (Access Denied) が返されます。ユーザーはこのレスポンスコードを見て、特定のオブジェクトが存在するかどうかを推測できます。この動作を希望しない場合は、バケットのウェブサイトのサポートを有効にしないでください。

次のサンプルバケットポリシーは、指定したフォルダ内のオブジェクトへのすべてのユーザーのアクセスを付与します。バケットポリシーの詳細については、「バケットポリシーの使用 (p. 328)」を参照してください。

```
{
  "Version":"2012-10-17",
  "Statement":[{
 "Sid":"PublicReadGetObject",
        "Effect":"Allow",
    "Principal": {
            "AWS": "*"
         },
      "Action":["s3:GetObject"],
      "Resource":["arn:aws:s3:::example-bucket/*"
      ]
    }
  ]
}
```

Note

バケットポリシーは、バケット所有者が所有するオブジェクトにのみ適用されます。バケットに、バケット所有者として所有していないオブジェクトが含まれる場合は、オブジェクトの ACL を使用して、それらのオブジェクトへのパブリック読み取りアクセス許可を付与する必要があります。

オブジェクトにパブリック読み取りアクセス許可を付与するには、バケットポリシーまたはオブジェクト ACL を使用します。ACL を使用してオブジェクトを公開で読み取り可能にするには、次の許可要素

に示すように、AllUsers グループに READ アクセス許可を付与します。この許可要素をオブジェクト ACL に追加します。ACL の管理については、「ACL の使用 (p. 347)」を参照してください。

```
<Grant>
  <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:type="Group">
    <URI>http://acs.amazonaws.com/groups/global/AllUsers</URI>
  </Grantee>
  <Permission>READ</Permission>
</Grant>
```

チュートリアル例 – Amazon S3 でのウェブサイトのホスティング

Topics

このセクションでは、2 つの例を紹介します。最初の例では、バケットをウェブサイトホスティング用に設定して、サンプルのインデックスドキュメントをアップロードし、ウェブサイトのテストを、このバケット用の Amazon S3 ウェブサイトエンドポイントを使用して行います。2 番目の例では、独自のドメイン (例: example.com) を Amazon S3 バケットウェブサイトエンドポイントの代わりに使い、ウェブサイトとして設定された Amazon S3 バケットからコンテンツを配信する方法を示します。この例では、Amazon S3 でルートドメインがどのようにサポートされているかについても説明します。

例: 静的ウェブサイトをセットアップする

Amazon S3 バケットを、ウェブサイトのように機能させるよう設定できます。この例では、Amazon S3 でウェブサイトをホスティングする手順を説明します。この手順を次に示します。必要なタスクの実行には AWS Management Console を使用します。

1. Amazon S3 バケットを作成し、ウェブサイトとして設定します。
2. バケットのコンテンツをパブリックにアクセス可能にするためのバケットポリシーを追加します。

   ウェブサイトエンドポイントから配信するコンテンツは、パブリックに読み取り可能であることが必要です。必要なアクセス許可を付与するには、バケットポリシーを追加するか、アクセスコントロールリスト (ACL) を使用します。次に、バケットポリシーの追加について説明します。
3. インデックスドキュメントをアップロードします。
4. Amazon S3 バケットウェブサイトエンドポイントを使用して、ウェブサイトをテストします。

バケットを作成し、ウェブサイトとして設定するには

1. AWS Management Console にサインインして Amazon S3 コンソールを開きます (https://console.aws.amazon.com/s3)。
2. バケットを作成します。

   詳細な手順については、『*Amazon Simple Storage Service Console User Guide*』の「Creating a Bucket」を参照してください。

バケットの命名のガイドラインについては、「バケットの制約と制限 (p. 88)」を参照してください。登録済みのドメイン名をお持ちの場合、バケットの命名に関する追加情報については、「CNAME による Amazon S3 URL のカスタマイズ (p. 66)」を参照してください。

3. バケットの [Properties] パネルで、[Static Website Hosting] をクリックして次の操作を行います。

   1. [Enable website hosting] を選択します。
   2. [Index Document] ボックスに、使用するインデックスドキュメント名を追加します。この名前は通常、index.html です。
   3. [Save] をクリックしてウェブサイト設定を保存します。
   4. [Endpoint] の内容を書き留めます。

      これが、Amazon S3 がこのバケットのために用意したウェブサイトエンドポイントです。このエンドポイントは、後のステップでウェブサイトをテストするときに使用します。

バケットのコンテンツを公開するバケットポリシーを追加するには

1. バケットの [Properties] ペインで、[Permissions] をクリックします。
2. [Add Bucket Policy] をクリックします。
3. 次のバケットポリシーをコピーし、[Bucket Policy Editor] に貼り付けます。

```
{
  "Version":"2012-10-17",
  "Statement":[{
 "Sid":"PublicReadForGetBucketObjects",
        "Effect":"Allow",
   "Principal": {
            "AWS": "*"
         },
      "Action":["s3:GetObject"],
      "Resource":["arn:aws:s3:::example-bucket/*"
      ]
    }
  ]
}
```

4. ポリシーで、*example-bucket* をバケット名に置き換えます。
5. [Save] をクリックします。

インデックスドキュメントをアップロードするには

1. ドキュメントを作成します。ファイル名は、先程インデックスドキュメントとして指定した名前と同じにする必要があります。
2. コンソールを使用して、インデックスドキュメントをバケットにアップロードします。詳細な手順については、『*Amazon Simple Storage Service Console User Guide*』の「Uploading Objects into Amazon S3」を参照してください。

ウェブサイトをテストします。

- ブラウザに次の URL を入力します。example-bucket はバケット名に置き換えます。また、*website-region* は、バケットをデプロイしたリージョン名に置き換えます。リージョン名については、「ウェブサイトエンドポイント (p. 412)」を参照してください。

```
http://example-bucket.s3-website-region.amazonaws.com
```

ブラウザに index.html ページが表示されれば、ウェブサイトのデプロイが成功しています。

Note

ウェブサイトへの HTTPS アクセスはサポートされていません。

これで、Amazon S3 でウェブサイトがホスティングされるようになりました。このウェブサイトには、Amazon S3 ウェブサイトエンドポイントの URL を指定してアクセスできます。ただし、作成したウェブサイトのコンテンツを配信するのに、example.com などのドメインを使用することもできます。また、Amazon S3 のルートドメインサポートを利用すると、http://www.example.com と http://example.com のどちらのリクエストでも処理できるようになります。処理できるようになるには、追加のステップが必要です。例については、「例: カスタムドメインを使用して静的ウェブサイトをセットアップする (p. 428)」を参照してください。

例: カスタムドメインを使用して静的ウェブサイトをセットアップする

Topics

Amazon S3 で静的ウェブサイトをホスティングするとします。ドメインは登録済みであり (例: example.com) 、http://www.example.com と http://example.com へのリクエストに対しては Amazon S3 からコンテンツを配信するようにします。

既に存在する静的ウェブサイトを Amazon S3 でホスティングすることにした場合も、一から始める場合も、ウェブサイトを Amazon S3 でホスティングするときは、この例を参考にしてください。

開始する前に

この例の手順では、次のサービスを操作する方法について説明します。

選択したドメインレジストラ – example.com などの登録済みドメイン名がない場合は、選択したレジストラでドメイン名を作成する必要があります。一般的には、ドメインを登録するには毎年少額の手数料が必要です。ドメイン名を登録する手順については、レジストラのウェブサイトを参照してください。

Amazon S3 – Amazon S3 を使用して、バケットを作成したり、サンプルウェブサイトページをアップロードしたり、全員がコンテンツを表示できるようアクセス許可を設定したりします。また、ウェブサイトホスティング用にバケットを設定することもできます。この例では、http://www.example.com と http://example.com のどちらに対するリクエストも許可するため 2 つのバケットを作成しますが、ホスティング対象は一方のバケットのみのコンテンツです。他方の Amazon S3 バケットは、そのコンテンツをホスティングしているバケットにリクエストをリダイレクトするように設定します。

Amazon Route 53 – Amazon Route 53 を DNS プロバイダとして設定します。ドメインのホストゾーンを Amazon Route 53 で作成し、適切な DNS レコードを設定します。既存の DNS プロバイダから切り替える場合は、ドメインの DNS レコードがすべて転送されたことを確認する必要があります。

この例で説明する手順を実行するときは、ドメイン、DNS (Domain Name System)、CNAME レコード、および A レコードについて基本的な知識があるとよいでしょう。これらの概念に関する詳しい説明はこのガイドでは取り上げませんが、ドメイン登録時に、ドメインレジストラから必要な基本的情報が提供されるのが一般的です。

Note

この例の手順では、`example.com` がドメイン名として使用されています。このドメイン名は、実際に登録したドメイン名で置き換える必要があります。

ステップ 1: ドメインを作成する

登録済みドメインが既にある場合、このステップは省略できます。ウェブサイトをホスティングするのが初めての場合は、最初のステップはドメイン (例: `example.com`) を任意のレジストラに登録することです。

レジストラを選択したら、そのレジストラのウェブサイトに記載されている手順にしたがってドメイン名を登録します。ドメイン名の登録に使用できるレジストラのウェブサイトのリストについては、ICANN.org にアクセスしてください。

ドメイン名の登録が完了したら、次の作業は、Amazon S3 バケットを作成してウェブサイトホスティング用に設定することと、ウェブサイトのコンテンツをアップロードすることです。

ステップ 2: バケットを作成および設定し、データをアップロードする

この例では、ルートドメイン (例: `example.com`) およびサブドメイン (例: `www.example.com`) の両方からのリクエストをサポートするために、2 つのバケットを作成します。一方のバケットにはコンテンツを格納し、他方のバケットはリクエストをリダイレクトするように設定します。次に示す作業を Amazon S3 コンソールで実行してウェブサイトを作成し、設定します。

1. バケットを 2 つ作成します。
2. これらのバケットをウェブサイトホスティング用に設定します。
3. Amazon S3 で用意されたバケットウェブサイトエンドポイントをテストします。

ステップ 2.1: バケットを 2 つ作成する

バケット名は、ホスティングするウェブサイトの名前と同じである必要があります。例えば、`example.com` というウェブサイトを Amazon S3 でホスティングするには、作成するバケットの名前は `example.com` となります。ウェブサイトを `www.example.com` の下でホスティングするには、バケットの名前を `www.example.com` とします。この例のウェブサイトは、`example.com` と `www.example.com` のどちらのリクエストもサポートします。

このステップでは、Amazon S3 コンソールに AWS アカウント認証情報でログインして、次の 2 つのバケットを作成します。

- *`example.com`*
- `www.`*`example.com`*

Note

この例のためのバケットを作成するには、次の手順を行います。この例のとおりに実行するときに、*example.com* の代わりに実際に登録したドメイン名を使用してください。

1. AWS マネジメントコンソールにサインインして Amazon S3 コンソール（ https://console.aws.amazon.com/s3/ ）を開きます。
2. バケットを 2 つ作成し、ドメイン名と同じ名前を付けます。バケット名の 1 つにはサブドメイン www が含まれます。例えば、*example.com* と www.*example.com* です。
詳細な手順については、『*Amazon Simple Storage Service Console User Guide*』の「Creating a Bucket」を参照してください。
3. ウェブサイトのデータを *example.com* バケットにアップロードします。

   ウェブサイトのコンテンツは、ルートドメインのバケット (*example.com*) からホスティングし、www.*example.com* に対するリクエストはルートドメインのバケットにリダイレクトします。コンテンツは、どちらのバケットにも格納できます。この例では、コンテンツを *example.com* バケットでホスティングします。コンテンツの種類は、テキストファイルや、家族の写真、ビデオなど、何でもかまいません。まだウェブサイトを作成したことがない場合は、この例のためにファイルを 1 つだけ用意してください。どのようなファイルでもアップロードできます。例えば、次に示す HTML を使用してファイルを作成し、バケットにアップロードします。ウェブサイトのホームページのファイル名は、一般的には index.html ですが、任意の名前を付けることができます。後のステップで、このファイル名をウェブサイトのインデックスドキュメント名として指定します。

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
    <title>My Website Home Page</title>
</head>
<body>
  <h1>Welcome to my website</h1>
  <p>Now hosted on Amazon S3!</p>
</body>
</html>
```

   詳細な手順については、『*Amazon Simple Storage Service Console User Guide*』の「Uploading Objects into Amazon S3」を参照してください。
4. オブジェクトに誰でもアクセスできるように、アクセス許可を設定します。

   次に示すバケットポリシーを *example.com* バケットにアタッチします。*example.com* を実際のバケットの名前に置き換えてください。バケットポリシーをアタッチする詳細な手順については、『*Amazon Simple Storage Service Console User Guide*』の「Editing Bucket Permissions」を参照してください。

```
{
  "Version":"2012-10-17",
  "Statement":[{
 "Sid":"AddPerm",
        "Effect":"Allow",
   "Principal": {
            "AWS": "*"
         },
      "Action":["s3:GetObject"],
      "Resource":["arn:aws:s3:::example.com/*"
      ]
    }
```

```
  ]
}
```

これで、2 つのバケット *example.com* と *www.example.com* が作成され、ウェブサイトのコンテンツが *example.com* バケットにアップロードされました。次のステップでは、リクエストを *example.com* バケットにリダイレクトするように *www.example.com* を設定します。リクエストをリダイレクトすると、同じウェブサイトコンテンツをコピーして 2 か所で以上維持する必要がなくなり、訪問者がブラウザで「www」を指定した場合もルートドメインだけを指定した場合も、同じウェブサイトコンテンツが「example.com」バケットから表示されるようになります。

ステップ 2.2: バケットをウェブホスティング用に設定する

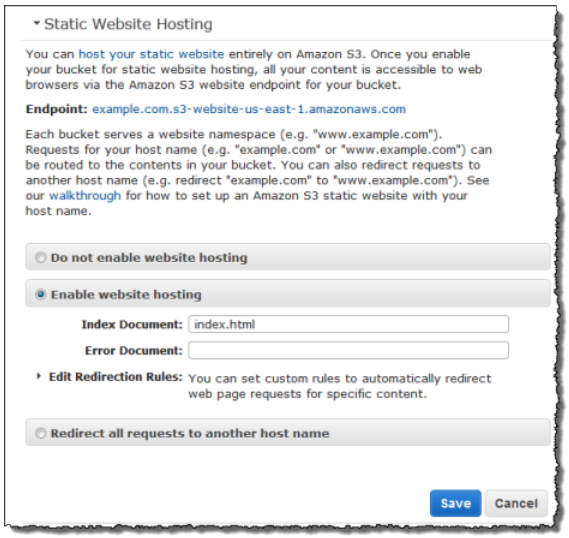
バケットをウェブサイトホスティング用に設定すると、そのウェブサイトには、Amazon S3 によって割り当てられたバケットウェブサイトエンドポイントを使用してアクセスできるようになります。

このステップでは、両方のバケットをウェブサイトホスティング用に設定します。最初に、*example.com* をウェブサイトとして設定してから、すべてのリクエストを *example.com* バケットにリダイレクトするように www.*example.com* を設定します。

example.com バケットをウェブサイトホスティング用に設定するには

1. *example.com* バケットをウェブサイトホスティング用に設定します。[Index Document] ボックスに、インデックスページの名前を入力します。

   詳細な手順については、『*Amazon Simple Storage Service Console User Guide*』の「To manage a bucket's website configuration」を参照してください。ウェブサイトエンドポイントの URL を書き留めます。これは、後で必要になります。

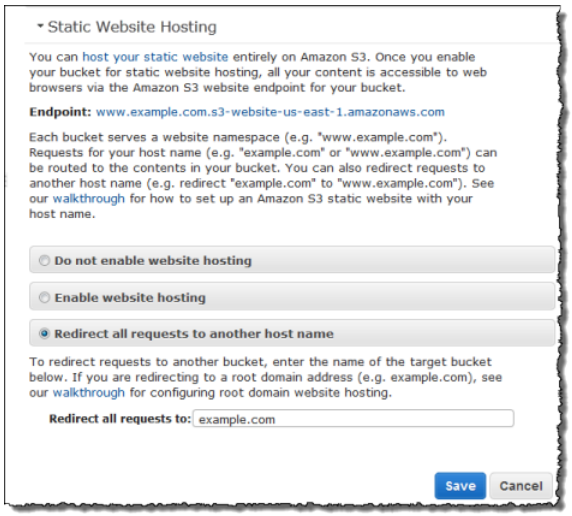


2. このウェブサイトをテストするには、[Endpoint] の URL をブラウザに入力します。

   ブラウザにインデックスドキュメントページが表示されます。次に、www.*example.com* へのリクエストをすべて *example.com* にリダイレクトするように www.*example.com* バケットを設定します。

リクエストを **www.*example.com*** から ***example.com*** にリダイレクトするには

1. Amazon S3 コンソールの [Buckets] リストで、[www.*example.com*] を右クリックし、[Properties] をクリックします。
2. [Static Website Hosting] で、[Redirect all requests to another host name] をクリックします。[Redirect all requests] ボックスに、「*example.com*」と入力します。



3. このウェブサイトをテストするには、[Endpoint] の URL をブラウザに入力します。

   リクエストがリダイレクトされて、ブラウザには *example.com* のインデックスドキュメントが表示されます。

次の Amazon S3 バケットウェブサイトエンドポイントには、すべてのインターネットユーザーがアクセスできます。

example.com.s3-website-us-east-1.amazonaws.com

http://www.*example.com*.s3-website-us-east-1.amazonaws.com

ここで、前述のステップで登録したドメインからのリクエストを提供するように、追加の設定を行います。例えば、ドメイン example.com を登録した場合は、次の URL からリクエストを提供します。

http://*example.com*

http://www.*example.com*

次のステップでは、カスタマーが上記の URL を使用してサイトにアクセスできるように、Amazon Route 53 を使用します。

ステップ 3: Amazon Route 53 ホストゾーンを作成、設定する

ここでは、Amazon Route 533 をドメインネームシステム (DNS) プロバイダとして設定します。コンテンツをルートドメイン (例: *example.com*) から配信できるようにしたい場合、Amazon Route 53 を使用する必要があります。ホストゾーンを作成し、そのホストゾーンで、ドメインに関連付けた DNS レコードを保持します。

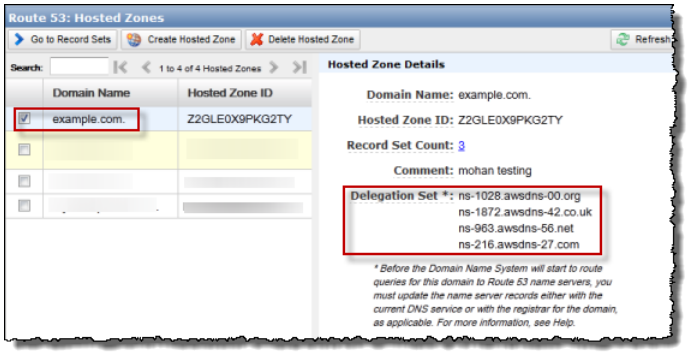
- エイリアスレコード – ドメイン *example.com* を Amazon S3 ウェブサイトエンドポイント (例: s3-website-us-east-1.amazonaws.com) にマップします。

- CNAME レコード – ドメイン www.*example.com* を、対応する Amazon S3 バケットウェブサイトエンドポイントにマップします。`http://www.`*example.com* に対するリクエストは、Amazon S3 バケットウェブサイトエンドポイントにルーティングされます。

ステップ 3.1: ドメインのホストゾーンを作成する

Amazon Route 53 コンソール (https://console.aws.amazon.com/route53) に移動し、ドメインのホストゾーンを作成します。詳細な手順については、『*http://docs.aws.amazon.com/Route53/latest/DeveloperGuide/*』の「Creating a Hosted Zone」を参照してください。

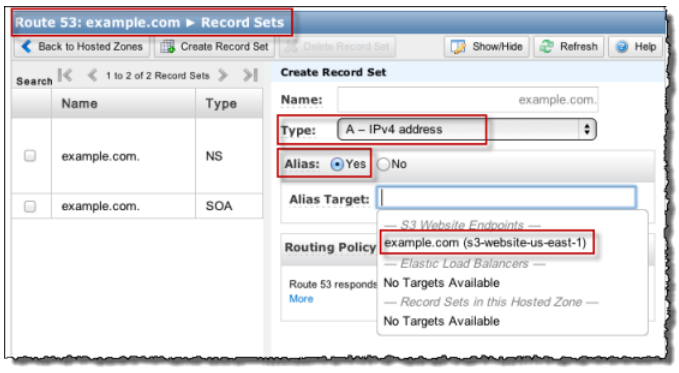
次の例は、example.com ドメイン用に作成されたホストゾーンを示しています。このドメインの Route 53 ネームサーバー (NS) を書き留めます。これは、後で必要になります。



ステップ 3.2: エイリアスレコードをホストゾーンに追加する

ドメイン用にホストゾーンに追加したエイリアスゾーンによって、*example.com* が対応する Amazon S3バケットにマップされます。このエイリアスレコードでは、IPアドレスを使用する代わりに、Amazon S3 ウェブサイトエンドポイントが使用されます。Amazon Route 53 によって、エイリアスレコードと、Amazon S3 バケットが存在する IP アドレスとのマッピングが維持されます。

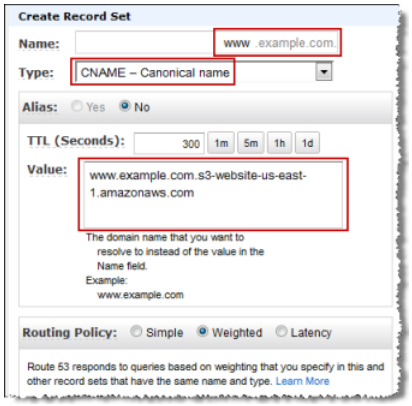
詳細な手順については、「*Amazon Route 53 開発者ガイド*」の「Creating Alias Resource Record Sets Using the Route 53 Console」を参照してください。



このホストゾーンを有効にするには、Amazon Route 53 をドメイン *example.com* の DNS サーバーとして使用する必要があります。既存のウェブサイトを Amazon S3 に移行する予定の場合は、切り替える前に、ドメイン *example.com* に関連付けられている DNS レコードを、このドメイン用に Amazon Route 53 で作成したホストゾーンに転送する必要があります。新しいウェブサイトを作成する場合は、ステップ 4 に直接進むことができます。

ステップ 3.3: CNAME レコードをホストゾーンに追加する

追加した CNAME レコードによって www.*example.com* に対するリクエストが、対応する Amazon S3 バケットのエンドポイントにルーティングされます。詳細な手順については、『*Amazon Route 53 開発者ガイド*』の「Creating, Changing, and Deleting Resource Record Sets Using the Route 53 API」を参照してください。次の例に示されている CNAME の値は、Amazon S3 コンソールで指定したホスト名です。



ステップ 3.4: 他の DNS レコードを現在の DNS プロバイダから Route 53 に転送する

DNS プロバイダを Amazon Route 53 に切り替える前に、MX レコード、CNAME レコード、A レコードなど、残りのすべての DNS レコードを Amazon Route 53 に転送する必要があります。以下のレコードは、転送する必要はありません。

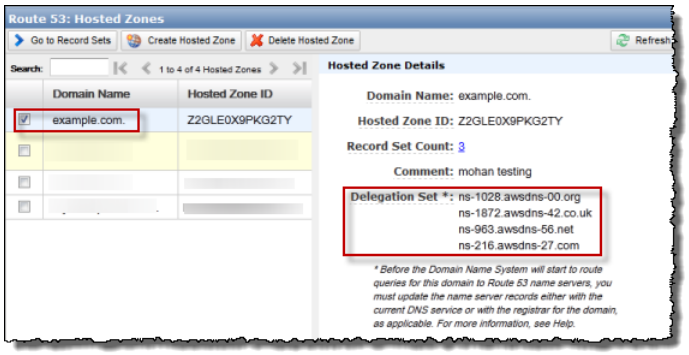
- NS レコード – 転送するのではなく、このレコードの値を Amazon Route 53 で用意されたネームサーバーの値で置き換えます。
- SOA レコード – Amazon Route 53 によってこのレコードがホストゾーン内に作成され、デフォルト値が設定されます。

必須の DNS レコードを移行するステップは、既存のすべてのサービスが引き続きこのドメイン名で利用できるようにするために重要です。

ステップ 4: DNS プロバイダを Amazon Route 53 に切り替える

DNS プロバイダを Amazon Route 53 に切り替えるには、現在利用している DNS プロバイダ側のネームサーバー (NS) レコードを、Amazon Route 53 の委託セット内のネームサーバーを使用するように更新する必要があります。

DNS プロバイダのサイトにアクセスし、次の Amazon Route 53 コンソールのスクリーンショットに示すホストゾーンの委託セット値を反映するように NS レコードを更新します。詳細については、『*Amazon Route 53 開発者ガイド*』の「Updating Your Registrar's Name Servers」を参照してください。



Amazon Route 53 への転送が完了したら、ドメインのネームサーバーが確かに変更されたことを、ツールを使って確認します。Linux コンピュータでは、`dig` という DNS ルックアップユーティリティを使用します。例えば、次の `dig` コマンドがあるとします。

```
dig +recurse +trace www.example.com any
```

このコマンドは次の出力を返します (ここでは、一部のみを示します)。この出力に表示されている 4 つのネームサーバーは、`example.com` ドメインのために Amazon Route 53 ホストゾーンで作成したネームサーバーと同じものです。

```
...
example.com.       172800  IN      NS      ns-9999.awsdns-99.com.
example.com.       172800  IN      NS      ns-9999.awsdns-99.org.
example.com.       172800  IN      NS      ns-9999.awsdns-99.co.uk.
example.com.       172800  IN      NS      ns-9999.awsdns-99.net.

www.example.com.   300     IN      CNAME   www.example.com.s3-website-us-east-
1.amazonaws.com.
...
```

ステップ 5: テストする

ウェブサイトが正しく動作することを確認するために、ブラウザで次の URL を入力します。

- `http://example.com` – `example.com` バケット内のインデックスドキュメントが表示されます。
- `http://www.example.com` – リクエストが `http://example.com` にリダイレクトされます。

バケットイベントの通知の設定

Amazon S3 通知機能により、Amazon S3 がバケットに対するキーイベントを検出した場合に、Amazon Simple Notification Service（Amazon SNS）トピックに対するメッセージを発行するように、バケットを設定できます。このトピックを登録すると、ウェブサーバー、メールアドレス、Amazon Simple Queue Service キューなどのエンドポイントに、バケットイベントのメッセージを配信することができます。

通知料金は、通常の Amazon SNS 料金で課金されることに注意してください。http://aws.amazon.com/sns/#pricing にある料金表を参照してください。この機能を有効および無効にするリクエストの料金以外に、メッセージを Amazon SNS トピックに発行するための Amazon S3 料金が追加で発生することはありません。バケットイベントの通知を受け取るには、そのバケットに設定されたトピックに登録する必要があります。

現在のところ、Amazon S3 でサポートされているイベントは *`s3:ReducedRedundancyLostObject`* イベントだけです。*`s3:ReducedRedundancyLostObject`* イベントは、低冗長化ストレージ（RRS）オブジェクトのすべてのレプリカが消失し、そのオブジェクトのリクエストを処理できなくなったことを Amazon S3 が検出した場合にトリガーされます。このイベントの通知に基づいて、顧客に影響を与えることがないように、消失した RRS オブジェクトを積極的に置き換えることができます。

通知は、指定した Amazon SNS トピックに発行されます。現在のところ、通知に対して設定できるトピックは 1 つだけです。バケット所有者と Amazon SNS トピック所有者が同じ場合、バケット所有者はデフォルトで通知をトピックに発行するアクセス許可を持ちます。それ以外の場合、トピック所有者は、バケット所有者にトピックへの通知発行を許可するポリシーを作成する必要があります。このポリシーの作成の詳細については、「Example Cases for Amazon SNS Access Control」を参照してください。コンソールを使用して通知を許可する方法の詳細については、『*Amazon Simple Storage Service Console User Guide*』の「Enabling RRS Lost Object Notifications」を参照してください。

Important

バケットの通知設定に指定する Amazon SNS トピックは、バケットと同じリージョンに存在する必要があります。

次の表は、バケットの可能な *`LocationContraint`* 値のトピックリージョンをまとめたものです。

LocationConstraint	トピックリージョン
空の文字列	us-east-1
EU	eu-west-1

LocationConstraint	トピックリージョン
us-west-1	us-west-1
us-west-2	us-west-2
ap-northeast-1	ap-northeast-1
ap-southeast-1	ap-southeast-1
sa-east-1	sa-east-1

デフォルトでは、バケット所有者のみがバケットの通知を設定および表示する権限を持っています。ただし、バケット所有者はバケットポリシーを使用して他のユーザーに許可を与えることができます。設定を許可するには *s3:PutBucketNotification* 許可を使用し、この設定の表示を許可するには *s3:GetBucketNotification* 許可を使用します。バケットポリシーの詳細については、「Using Bucket Policies (p. 328)」を参照してください。

他のバケット設定と同様に、各バケットには、そのバケットの通知設定を実行または表示するためのサブリソースが存在します。バケットで通知を設定するには、Amazon S3 REST API でバケットの *notification* サブリソースを使用します。*notification* サブリソースの詳細については、「Put Bucket notification」および「Get Bucket notification」を参照してください。

バケットの通知を設定するには

- Amazon S3 トピックの Amazon リソースネーム（ARN）、通知が発行される場所、および報告されるイベントを指定する *NotificationConfiguration* 要素を含んだリクエスト本文を指定して、PUT オペレーションを実行します。

```
PUT ?notification HTTP/1.1
Host: quotes.s3.amazonaws.com
Date: Wed, 02 June 2010 12:00:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:xQE0diMbLRepdf3YB+FIEXAMPLE=
<NotificationConfiguration>
   <TopicConfiguration>
       <Topic>arn:aws:sns:us-east-1:111122223333:myTopic</Topic>
           <Event>s3:ReducedRedundancyLostObject</Event>
   </TopicConfiguration>
</NotificationConfiguration>
```

PUT オペレーションを呼び出してバケットで通知を設定すると、Amazon S3 はテスト通知を発行して、トピックが存在すること、および指定したトピックに発行するアクセス許可をバケット所有者が持っていることを確認します。通知が Amazon SNS トピックに正常に発行された場合、PUT オペレーションはバケット設定を更新し、*x-amz-sns-test-message-id* ヘッダーにトピックに送信されたテスト通知のメッセージ ID を含む 200 OK レスポンスを返します。PUT オペレーションエラーについては、「Put Bucket notification」を参照してください。

イベントに関してトピックに発行されるメッセージには、以下の表に示すフィールドが含まれています。

名前	説明
Service	Amazon S3。

名前	説明
Event	現在のところ、サポートされているイベントは *s3:ReducedRedundancyLostObject* だけです。
Time	イベントがトリガーされた時刻。
バケット	バケットの名前。
キー	オブジェクト名。
VersionId	バージョニングが有効になっている場合は、バージョン ID。バージョニングが有効になっていない場合は、空の文字列。
RequestID	通知を発行した Amazon S3 オペレーションを識別する一意の ID。
HostID	メッセージを送信したホストを識別する一意の ID。

次のメッセージは、*s3:ReducedRedundancyLostObject* イベントを発行するメッセージの例です。

```
{
    "Service":"Amazon S3",
    "Event": "s3:ReducedRedundancyLostObject",
    "Time":"2010-02-11T23:48:22.000Z",
    "Bucket": "myphotos",
    "Key": "Home/2009/10/carvingpumpkins.jpg",
    "VersionId":"qfXHy5689N7n9mWVwanN_hIroMn_rzXl",
    "RequestId":"4442587FB7D0A2F9",
   "HostId":"fHZOHz+oQlKAFQ7RgVSklvcDNYI0v05gsOWWeJcyTTXWgGzI6CC5FZVICjD9bUzT"
}
```

次のメッセージは、通知に対してバケットが有効の場合に Amazon S3 が送信するテストメッセージの例です。

```
{
    "Service":"Amazon S3",
    "Event": "s3:TestEvent",
    "Time":"2010-02-11T23:48:22.000Z",
    "Bucket": "myphotos",
    "RequestId":"4442587FB7D0A2F9",
   "HostId":"fHZOHz+oQlKAFQ7RgVSklvcDNYI0v05gsOWWeJcyTTXWgGzI6CC5FZVICjD9bUzT"
}
```

バケットの通知設定を読み取るには

- GET オペレーションを使用します。このオペレーションは、バケットの通知に設定された Amazon SNS トピックおよびイベントを含む *NotificationConfiguration* 要素を返します。

```
GET ?notification HTTP/1.1
Host: quotes.s3.amazonaws.com
Date: Wed, 09 June 2010 12:00:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

バケットで通知を無効にするには

- リクエスト本文に空の PUT 要素を含む *NotificationConfiguration* オペレーションを使用します。

リクエストルーティング

Topics
- リクエストのリダイレクトと REST API (p. 440)
- DNS に関する考慮事項 (p. 443)

<CreateBucketConfiguration> API を使用して作成されたバケットに対してリクエストを行うプログラムは、リダイレクトをサポートしている必要があります。また、DNS TTL にしたがわない一部のクライアントで問題が発生する場合があります。

このセクションでは、Amazon S3 で使用するサービスまたはアプリケーションを設計するときに考慮すべきルーティングおよび DNS の問題について説明します。

リクエストのリダイレクトと REST API

概要

Amazon S3 はドメインネームシステム (DNS) を使用して、リクエストを処理できる施設にルーティングします。このシステムは非常に効果的に機能します。ただし、一時的なルーティングエラーが発生する場合があります。

リクエストが誤った Amazon S3 ロケーションに到達した場合、Amazon S3 は、リクエストを新しいエンドポイントに再送するようリクエスタに指示する一時的なリダイレクトを返します。

リクエストの形式が正しくない場合、Amazon S3 は永続的なリダイレクトを使用して、リクエストを正しく実行する方法を指示します。

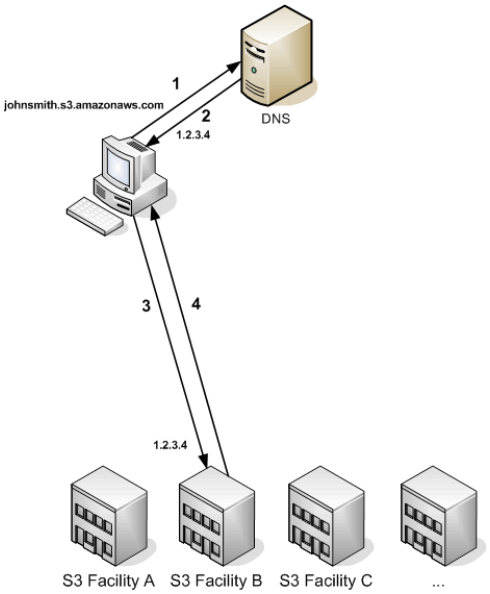
Important

各 Amazon S3 プログラムは、リダイレクトレスポンスを処理するよう設計する必要があります。唯一の例外は、`<CreateBucketConfiguration>` なしで作成されたバケットのみを使用するプログラムの場合です。ロケーションの制約の詳細については、「バケットとリージョン (p. 90)」を参照してください。

DNS ルーティング

DNS ルーティングは、リクエストを適切な Amazon S3 施設にルーティングします。

次の図は、DNS ルーティングの例を示しています。



1	クライアントが、Amazon S3 に格納されているオブジェクトを取得するために DNS リクエストを行います。
2	クライアントが、リクエストを処理できる施設の 1 つまたは複数の IP アドレスを受け取ります。
3	クライアントが、Amazon S3 施設 B へのリクエストを行います。
4	施設 B がオブジェクトのコピーを返します。

一時的なリクエストのリダイレクト

一時的なリダイレクトとは、リクエストを異なるエンドポイントに再送する必要があることをリクエスタに知らせるタイプのエラーレスポンスです。

Amazon S3 が持つ分散型の特質により、リクエストが誤った施設に一時的にルーティングされる場合があります。このエラーは、大半の場合、バケットが作成または削除された直後に発生します。例えば、新しいバケットを作成し、ただちにバケットへのリクエストを実行する場合、バケットのロケーションの制約によっては一時的にリダイレクトされる可能性もあります。米国スタンダードリージョン（s3.amazonaws.com エンドポイント）にバケットを作成した場合は、 リダイレクトされません。このリージョンがデフォルトのエンドポイントであるためです。これに対し、その他のリージョンにバケットを作成した場合、バケットに対するリクエストはこのデフォルトのエンドポイントに転送されますが、バケットの DNS エントリは伝播されます。デフォルトのエンドポイントは、HTTP 302 レスポンスを表示すると共に、リクエストを正しいエンドポイントにリダイレクトします。

一時的なリダイレクトには正しい施設への URI も含まれており、これを使用して直ちにリクエストを再送することができます。

Important

以前のリダイレクトレスポンスで指定されたエンドポイントを再度使用しないでください。こうしたエンドポイントは長い期間を経ても機能しているように見えることがありますが、予期せぬ結果を招く場合があり、最終的には通知なしに失敗します。

次の図は、一時的なリダイレクトの例を示しています。

1	クライアントが、Amazon S3 に格納されているオブジェクトを取得するために DNS リクエストを行います。
2	クライアントが、リクエストを処理できる施設の 1 つまたは複数の IP アドレスを受け取ります。
3	クライアントが、Amazon S3 施設 B へのリクエストを行います。
4	施設 B がリダイレクトを返して、オブジェクトがロケーション C から利用可能であることを示します。
5	クライアントがリクエストを施設 C に再送します。
6	施設 C がオブジェクトのコピーを返します。

永続的なリクエストのリダイレクト

永続的なリダイレクトは、リクエストがリソースのアドレス指定を不適切に行ったことを示します。例えば、パス形式のリクエストを使用して、`<CreateBucketConfiguration>` によって作成されたバケットにアクセスした場合、永続的なリダイレクトが発生します。詳細については、「Using CreateBucketConfiguration (p. 89)」を参照してください。

開発中にこれらのエラーを見つけやすいように、このタイプのリダイレクトには、リクエストを正しいロケーションに自動的に誘導するロケーション HTTP ヘッダーが含まれていません。正しい Amazon S3 エンドポイントを使用するために、生成される XML エラードキュメントを参照してください。

Example REST API のリダイレクト

```
HTTP/1.1 307 Temporary Redirect
Location: http://johnsmith.s3-gztb4pa9sq.amazonaws.com/pho
tos/puppy.jpg?rk=e2c69a31
Content-Type: application/xml
Transfer-Encoding: chunked
Date: Fri, 12 Oct 2007 01:12:56 GMT
Server: AmazonS3

<?xml version="1.0" encoding="UTF-8"?>
<Error>
  <Code>TemporaryRedirect</Code>
  <Message>Please re-send this request to the specified temporary endpoint.
  Continue to use the original request endpoint for future requests.</Message>

  <Endpoint>johnsmith.s3-gztb4pa9sq.amazonaws.com</Endpoint>
</Error>
```

Example SOAP API のリダイレクト

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

```
<soapenv:Body>
  <soapenv:Fault>
    <Faultcode>soapenv:Client.TemporaryRedirect</Faultcode>
   <Faultstring>Please re-send this request to the specified temporary endpoint.

    Continue to use the original request endpoint for future requests.</Fault
string>
    <Detail>
      <Bucket>images</Bucket>
      <Endpoint>s3-gztb4pa9sq.amazonaws.com</Endpoint>
    </Detail>
  </soapenv:Fault>
</soapenv:Body>
```

DNS に関する考慮事項

Amazon S3 の設計要件の 1 つは、非常に高い可用性です。この要件を満たす方法の1つとして、DNS 内の Amazon S3 エンドポイントに関連付られた IP アドレスを必要に応じて更新します。これらの変更は、存続時間の短いクライアントには自動的に反映されますが、存続時間の長いクライアントには反映されない場合があります。存続時間の長いクライアントの場合、これらの変更を活用するには、Amazon S3 エンドポイントを定期的に再解決するための特殊なアクションを実行する必要があります。仮想マシン (VM) の詳細については、以下を参照してください:

- Java の場合、Sun の JVM はデフォルトで DNS ルックアップを永続的にキャッシュに保持します。この動作を変更する方法については、InetAddress のドキュメントの「InetAddress キャッシング」セクションを参照してください。

- PHP の場合、ほとんどの一般的な導入構成で実行される永続的な PHP VM は、VM が再起動されるまで DNS ルックアップをキャッシュに保持します。getHostByName に関する PHP ドキュメントを参照してください。

パフォーマンスの最適化

Topics
- リクエスト率およびリクエストパフォーマンスに関する留意事項 (p. 445)
- TCP ウィンドウスケーリング (p. 449)
- TCP 選択的送達確認 (p. 449)

Amazon S3 には、高パフォーマンスネットワーキングをサポートする新機能が用意されています。これらの機能には、TCP ウィンドウスケーリングや選択的送達確認などがあります。

Note

高パフォーマンスの調整の詳細については、「http://www.psc.edu/networking/projects/tcptune/」を参照してください。

リクエスト率およびリクエストパフォーマンスに関する留意事項

Topics
- 複数のリクエストタイプの組み合わせが含まれるワークロード (p. 446)
- 大量の GET を使用するワークロード (p. 448)

Amazon S3 は数百万台のクライアントからのアクセスをサポートするように拡張することができます。Amazon S3 に対するお客様のワークロードが *1 秒あたり 100 個のリクエスト*を定常的に超えている場合は、このトピック内のガイドラインに従って、最高のパフォーマンスと拡張性を得られるようにしてください。

このトピックでは、2 つのタイプのワークロードについて説明します。

- 複数のリクエストタイプの組み合わせが含まれるワークロード – お客様のリクエストが GET、PUT、DELETE、または GET Bucket (List Objects) の組み合わせであることがほとんどの場合は、 オブジェクトに対して適切なキー名を選択することで、Amazon S3 のインデックス (次のセクションで説明) へのアクセスでのレイテンシーが短くなるため、確実にパフォーマンスが向上します。また、この選択により、1 秒あたりに送信されるリクエスト数の影響も受けないレベルでの拡張性が保証されます。

- 大量の GET を使用するワークロード – ワークロードの大部分が GET リクエストで構成される場合、Amazon CloudFront コンテンツ配信サービスを使用することをお勧めします。

Note

1 秒あたり 100 個以上のリクエストを定常的に処理している場合は、このセクションのガイドラインが該当します。通常のワークロードで 1 秒あたり 100 個以上のリクエストが発生するのが極めてまれの場合は、このセクションのガイドラインに従う必要はありません。

複数のリクエストタイプの組み合わせが含まれるワークロード

大量のオブジェクトをアップロードする場合、キー名の一部として連番や日付と時刻の値を使用することがあります。例えば、日付と時刻の組み合わせを使用するキー名を選択できます。このような例を次に示します。この例では、タイムスタンプがプレフィックスに含まれています。

```
examplebucket/2013-26-05-15-00-00/cust1234234/photo1.jpg
examplebucket/2013-26-05-15-00-00/cust3857422/photo2.jpg
examplebucket/2013-26-05-15-00-00/cust1248473/photo2.jpg
examplebucket/2013-26-05-15-00-00/cust8474937/photo2.jpg
examplebucket/2013-26-05-15-00-00/cust1248473/photo3.jpg
...
examplebucket/2013-26-05-15-00-01/cust1248473/photo4.jpg
examplebucket/2013-26-05-15-00-01/cust1248473/photo5.jpg
examplebucket/2013-26-05-15-00-01/cust1248473/photo6.jpg
examplebucket/2013-26-05-15-00-01/cust1248473/photo7.jpg
...
```

キー名を連続するパターンにすると、パフォーマンス上の問題が発生します。この問題を理解するため、キー名が Amazon S3 でどのように格納されるかを説明します。

Amazon S3 では、各 AWS リージョンにオブジェクトキー名のインデックスを保持しています。オブジェクトキーは、インデックス内の複数のパーティションに分散して辞書におけるのと同じ順序で格納されています。つまり、Amazon S3 はキー名をアルファベット順に格納しています。キー名により、自動的にそのキーが格納されるパーティションが決まります。タイムスタンプやアルファベット順などの連続するプレフィックスを使用すると、Amazon S3 が大量のキーに対して特定のパーティションを対象にするため、そのパーティションの I/O 能力がひっ迫する可能性が増大します。キー名のプレフィックスをランダムにすると、キー名、したがって I/O ロードは複数のパーティションに分散されます。

ワークロードが 1 秒あたり 100 個のリクエストを常に超えることが予想される場合、連続するキー名は避けてください。キー名で連番や日付と時刻のパターンを使用する場合は、キー名にランダムなプレフィックスを追加します。プレフィックスがランダムであることで、キー名が複数のインデックスパーティションに均等に分散されます。ランダムにする例は、このトピック内で後述します。

Note

次のセクションのキー名プレフィックスについてのガイドラインは、バケット名にも適用できます。Amazon S3 がキー名をインデックスに格納する場合、格納するキー名の一部はバケット名になります (例、`examplebucket/object.jpg`) 。

例 1: 16 進のハッシュプレフィックスをキー名に追加する

キー名をランダムにする 1 つの方法は、キー名のプレフィックスとしてハッシュ文字列を追加することです。例えば、キー名として付ける文字列の MD5 ハッシュを計算することができます。ハッシュから特定の数の文字を選択し、キー名のプレフィックスとして追加します。以下の例では、4 文字のハッシュが追加されたキー名を示します。

Note

3 ~ 4 文字のハッシュプレフィックスを追加すれば十分です。 プレフィックスとしては、16 進のハッシュを使用することを強くお勧めします。

```
examplebucket/232a-2013-26-05-15-00-00/cust1234234/photo1.jpg
examplebucket/7b54-2013-26-05-15-00-00/cust3857422/photo2.jpg
examplebucket/921c-2013-26-05-15-00-00/cust1248473/photo2.jpg
examplebucket/ba65-2013-26-05-15-00-00/cust8474937/photo2.jpg
examplebucket/8761-2013-26-05-15-00-00/cust1248473/photo3.jpg
examplebucket/2e4f-2013-26-05-15-00-01/cust1248473/photo4.jpg
examplebucket/9810-2013-26-05-15-00-01/cust1248473/photo5.jpg
examplebucket/7e34-2013-26-05-15-00-01/cust1248473/photo6.jpg
examplebucket/c34a-2013-26-05-15-00-01/cust1248473/photo7.jpg
...
```

キー名をランダムにすることと GET Bucket (List Objects) オペレーション

Amazon S3 では、GET Bucket (List Objects) オペレーションが可能です。これは、アルファベットのキー名のリストを返すものです。ただし、ハッシュプレフィックスのために、アルファベットのリストはランダムに並んでいます。オブジェクトをグループ化する必要がある場合は、キー名のハッシュ文字列の前にさらにプレフィックスを追加します。次の例では、キー名に `animations/` と `videos/` のプレフィックスを追加しています。

```
animations/232a-2013-26-05-15-00-00/cust1234234/animation1.obj
animations/7b54-2013-26-05-15-00-00/cust3857422/animation2.obj
animations/921c-2013-26-05-15-00-00/cust1248473/animation3.obj
videos/ba65-2013-26-05-15-00-00/cust8474937/video2.mpg
videos/8761-2013-26-05-15-00-00/cust1248473/video3.mpg
videos/2e4f-2013-26-05-15-00-01/cust1248473/video4.mpg
videos/9810-2013-26-05-15-00-01/cust1248473/video5.mpg
videos/7e34-2013-26-05-15-00-01/cust1248473/video6.mpg
videos/c34a-2013-26-05-15-00-01/cust1248473/video7.mpg
...
```

この場合、GET Bucket (List Objects) オペレーションで返される整列済みのリストは、`animations` と `videos` のプレフィックスでグループ化されています。

Note

ここでもまた、オブジェクトをグループ化するために追加したプレフィックスは連続していてはなりません。連続していると、単一のインデックスパーティションが前述のようにひっ迫するためです。

例 2: キー名の文字列を左右反転する

プレフィックスに昇順のアプリケーション ID が含まれるキー名のオブジェクトをアプリケーションでアップロードする場合を考えます。

```
2134857/data/start.png
2134857/data/resource.rsrc
2134857/data/results.txt
2134858/data/start.png
2134858/data/resource.rsrc
2134858/data/results.txt
2134859/data/start.png
2134859/data/resource.rsrc
2134859/data/results.txt
```

このようなキー名のスキームで書き込みオペレーションを行うと、 単一のインデックスパーティションがひっ迫します。これに対して、アプリケーション ID 文字列を左右に反転すると、ランダムなプレフィックスの付いたキー名が得られます。

```
7584312/data/start.png
7584312/data/resource.rsrc
7584312/data/results.txt
8584312/data/start.png
8584312/data/resource.rsrc
8584312/data/results.txt
9584312/data/start.png
9584312/data/resource.rsrc
9584312/data/results.txt
```

キー名の文字列を左右反転することで、1 文字めが異なるキー名ごとに Amazon S3 が固有のパーティションを使用するようになるための準備が整います。`examplebucket` はアプリケーションデータのアップロード先となるバケットの名前を指しています。

```
examplebucket/7
examplebucket/8
examplebucket/9
```

キーのカウントとリクエスト率は時間と共に増大するため、Amazon S3 が自動的にこれらのパーティションをさらに分割する場合があります。

大量の GET を使用するワークロード

お客様のワークロードが主に GET リクエストを送信することである場合は、パフォーマンスを最適化するために、前述のガイドラインに加えて、Amazon CloudFront の使用を検討してください。

Amazon CloudFront を Amazon S3 と合わせて使用することで、短いレイテンシーと高いデータ転送速度でユーザーにコンテンツを配信することができます。また、Amazon S3 に送信される直接リクエストが減少するため、負荷が削減されます。

例えば、使用頻度の高い少数のオブジェクトがあるとします。Amazon CloudFront はこのようなオブジェクトを Amazon S3 から取得し、キャッシュします。これにより、Amazon CloudFront はキャッシュを使用してオブジェクトに対する後続のリクエストを処理できるようになるため、Amazon S3 に送信する GET の数が削減されます。詳細については、Amazon CloudFront の製品詳細ページを参照してください。

TCP ウィンドウスケーリング

TCP ウィンドウスケーリングを使用すると、64 KB を超えるウィンドウサイズをサポートすることにより、オペレーティングシステムおよびアプリケーションレイヤーと Amazon S3 との間でネットワークスループットのパフォーマンスを向上させることができます。TCP セッションの開始時に、クライアントは、そのサポートされているウィンドウ WSCALE 係数をアドバタイズし、Amazon S3 は、アップストリーム方向でサポートされている受信ウィンドウ WSCALE 係数を返します。

TCP ウィンドウスケーリングによってパフォーマンスを向上させることができるものの、正しく設定することが困難な場合があります。アプリケーションとカーネルの両方のレベルで設定を調整してください。TCP ウィンドウスケーリングについては、オペレーティングシステムのドキュメントおよび RFC 1323 を参照してください。

TCP 選択的送達確認

TCP 選択的送達確認では、多数のパケット消失が発生すると回復時間が増大します。TCP 選択的送達確認は、新しいオペレーティングシステムのほとんどでサポートされていますが、有効にする必要がある場合があります。TCP 選択的送達確認については、オペレーティングシステムに付属のドキュメントおよび RFC 2018 を参照してください。

Amazon S3 での BitTorrent の使用

Topics

BitTorrent™ は、ファイルを配信するためのオープンなピアツーピアプロトコルです。BitTorrent プロトコルを使用することで、Amazon S3 で公にアクセス可能なあらゆるオブジェクトを取得することができます。ここでは、BitTorrent を使った Amazon S3 でのデータ配信について、その有用性と配信方法について説明します。

Amazon S3 は BitTorrent プロトコルをサポートしています。これにより開発者は、大規模なコンテンツの配信時にコストを削減できます。Amazon S3 は、あらゆるデータをシンプルかつ確実に保管するのに便利です。Amazon S3 データのデフォルトの配信メカニズムは、クライアント/サーバー方式のダウンロードを使用します。クライアント/サーバー方式による配信では、オブジェクト全体は Amazon S3 からオブジェクトをリクエストしたすべての認証済みユーザーへポイントツーポイントで転送されます。クライアント/サーバー方式の配信はさまざまな使用事例に適していますが、すべてについて最適ではありません。特に、クライアント/サーバー方式の配信におけるコストは、オブジェクトをダウンロードするユーザー数が増えるにつれて直線的に増加します。それでは人気のオブジェクトの配信には費用がかかります。

BitTorrent では、オブジェクトのダウンロードを行うクライアントそのものをディストリビューターとして活用することでこの問題に対処しています。各クライアントは、Amazon S3 からオブジェクトの一部分をダウンロードし、他のクライアントからも別の一部分をダウンロードします。同時に、対象となる他の「ピア」にもそのオブジェクトの一部分をアップロードします。大容量の人気ファイルの発行者にとっては、Amazon S3 によって提供されるデータ量が、同じクライアントにクライアント/サーバー方式でダウンロードする場合よりも大幅に少なくなるという利点があります。データ転送量がより少なければ、オブジェクトの発行者にとってのコストはより低くなります。

Note

torrent は、容量が 5 GB 未満のオブジェクトについてのみ取得できます。

BitTorrent 配信への課金方法

Amazon S3 で BitTorrent を使用することにより追加料金は発生しません。BitTorrent プロトコルによるデータ転送は、クライアント/サーバー方式による配信の場合と同一のレートで課金されます。正確に

は、ダウンロードを行う BitTorrent クライアントが Amazon S3 「シーダー」にオブジェクトの「一部」をリクエストした場合、REST または SOAP プロトコルを使って匿名リクエストがその「一部」に対して行われたのと同じように課金されます。この料金は Amazon S3 の請求書と使用状況レポートに通常と同じように表示されます。異なっている点は、多数のクライアントが同一のオブジェクトを BitTorrent によって同時にリクエストした場合、Amazon S3 がクライアントに対応するために扱うデータ量は、クライアント/サーバー方式による配信の場合よりも少なくなるということです。これは BitTorrent クライアントが相互間で同時にアップロードとダウンロードを行うためです。

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

BitTorrent を使用したデータ転送量の節約は、オブジェクトの人気度に依存しています。より人気のないオブジェクトはクライアントに対処するためにより多く「シーダー」を使わなければならず、このような場合には BitTorrent による配信コストとクライアント/サーバー方式による配信でのコストでは、差が小さいかも知れません。特に、もしある特定のオブジェクトを一度に 1 つのクライアントのみがダウンロードする場合、BitTorrent による配信のコストは直接ダウンロードする場合と同じになります。

BitTorrent による Amazon S3に格納されたオブジェクトの取得

Amazon S3 にある匿名で読み取り可能なオブジェクトはすべて、BitTorrent 経由でダウンロードすることもできます。それには BitTorrent クライアントアプリケーションが必要です。Amazon は BitTorrent クライアントアプリケーションを配布していませんが、多くの無料クライアントが入手可能です。Amazon S3BitTorrent の実装は、公式の BitTorrent クライアント (http://www.bittorrent.com/ を参照) との動作をテスト済みです。

BitTorrent によるダウンロードは、.torrent ファイルから始めます。この小さなファイルは、BitTorrent クライアントに対してダウンロードすべきデータと、どこからデータを探し始めるかを規定します。.torrent ファイルは、ダウンロードされる実際のオブジェクトの容量のほんの一部を占めます。BitTorrent クライアントアプリケーションに Amazon S3 が生成した.torrent ファイルが入力されると、Amazon S3 *and*、いずれかの「ピア」BitTorrent クライアントからただちにダウンロードを開始します。

公開されている任意のオブジェクトの .torrent ファイルを取得するのは簡単です。単に「?torrent」クエリ文字列パラメータを、オブジェクトへの REST GET リクエストの末尾に加えるだけです。認証は必要ありません。BitTorrent クライアントをインストールすれば、BitTorrent を使ってのオブジェクトのダウンロードはウェブブラウザでこの URL を開くのと同じくらい簡単です。

SOAP API を使用して Amazon S3 オブジェクトの .torrent をフェッチするメカニズムは存在していません。

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

Example

この例では、「quotes」バケットにあるオブジェクト「Nelson」の .torrent ファイルを取得します。

Sample Request

```
GET /quotes/Nelson?torrent HTTP/1.0
Date: Wed, 25 Nov 2009 12:00:00 GMT
```

Sample Response

```
HTTP/1.1 200 OK
x-amz-request-id: 7CD745EBB7AB5ED9
Date: Wed, 25 Nov 2009 12:00:00 GMT
Content-Disposition: attachment; filename=Nelson.torrent;
Content-Type: application/x-bittorrent
Content-Length: 537
Server: AmazonS3

<body: a Bencoded dictionary as defined by the BitTorrent specification>
```

Amazon S3 と BitTorrent を使用したコンテンツの発行

Amazon S3 に格納された匿名で読み取り可能なオブジェクトはすべて、自動的に BitTorrent を使ってダウンロード可能になっています。匿名での READ オペレーションを許可するためにオブジェクトの ACL を変更する方法については、「アクセスコントロール (p. 294)」で説明しています。

.torrent ファイルを直接配布するか、またはオブジェクトの ?torrent URL へのリンクを発行することで、クライアントを BitTorrent でアクセス可能なオブジェクトへ誘導することができます。ここで重要なことは、Amazon S3 オブジェクトを記述する .torrent ファイルは、初めてリクエストされたときに、オンデマンドで生成されるということです (REST ?torrent リソース経由) 。オブジェクトの .torrent の生成には、オブジェクトの容量に比例して時間がかかります。大規模オブジェクトの場合には、かなりの時間を要します。したがって、?torrent リンクを発行する前に、初回のリクエストを行うことをお勧めします。.torrent ファイルを生成する際、Amazon S3 が初回のリクエストに応答するまでに数分かかる場合があります。対象のオブジェクトを更新しない限り、その後の .torrent に対するリクエストは迅速に行えます。?torrent リンクの配布前にこの手順を行うことで、カスタマーの BitTorrent によるダウンロード操作はよりスムーズなものになります。

BitTorrent を使用したファイルの配布を中止するには、単純にファイルへの匿名アクセスを削除します。それには Amazon S3 からファイルを削除するか、匿名での読み取りを禁止するようアクセスコントロールポリシーを変更します。すると、BitTorrent ネットワーク内で Amazon S3 はファイルの「シーダー」として機能しなくなり、?torrent REST API による .torrent ファイルの提供を行わなくなります。しかし、ファイル用の .torrent の発行後には、このアクションでは BitTorrent のピアツーピアネットワークのみを使ったオブジェクトのパブリックダウンロードは停止できない場合があります。

Amazon S3 での Amazon DevPay の使用

Topics

Amazon DevPay を使用すると、Amazon の認証および請求インフラストラクチャを通じて、お客様の Amazon S3 製品を使用するカスタマーに課金することができます。使用量 (ストレージ、トランザクション、帯域幅) 、月ごとの定額、一括払いを含め、製品に対してどのような金額でも課金できます。

Amazonは、月に一度、お客様に代わり、カスタマーに請求を行います。その後、AWSは定額のAmazon DevPay手数料を差し引き、残額がお客様に支払われます。その後、AWSは、カスタマーの操作によって生じた Amazon S3 使用料と、料率に基づく Amazon DevPay 料金をお客様に個別に請求します。

カスタマーが請求に対して支払いを行わない場合、AWSはAmazonS3 (および製品) へのアクセスを停止します。すべての支払い処理は AWS が行います。

Amazon S3 カスタマーデータの分離

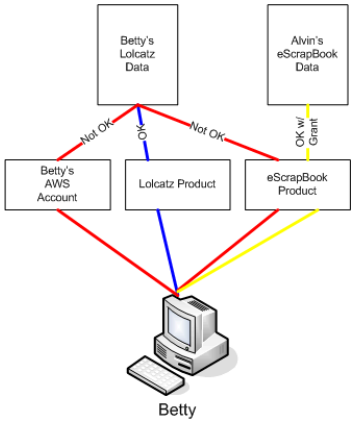
Amazon DevPay リクエストは、お客様の製品を利用するユーザーに代わってデータの保管とアクセスを行います。お客様のアプリケーションによって作成されたリソースは、ユーザーが所有権を持ちます。ACL を変更しない限り、お客様がユーザーデータを読み取ったり変更したりすることはできません。

お客様の製品によって保存されたデータは、他の Amazon DevPay 製品および一般的な Amazon S3 アクセスから隔離されます。製品を通じて Amazon S3 にデータを*保存する*カスタマーは、製品を通じてのみそのデータに*アクセス*できます。そのデータに他の Amazon DevPay 製品や個人の AWS アカウントからアクセスすることはできません。

製品のユーザー間では、アプリケーションが ACL を通じて明示的にアクセス権を与えた場合のみ、お互いのデータにアクセスできます。

例

次の図は、データアクセスの許可、不許可、条件付き（任意）の状態を示しています。



Betty のアクセスは以下のように制限されています。

- Lolcatz のデータには Lolcatz の製品を通じてアクセスできます。他の製品または個人の AWS アカウントを通じて自分の Lolcatz データにアクセスしようとすると、リクエストが拒否されます。
- Alvin の eScrapBook データには、明示的にアクセス権が与えられていれば eScrapBook 製品を通じてアクセスできます。

Amazon DevPay トークンのメカニズム

お客様がカスタマーに代わってリクエストを実行し、カスタマーがお客様のアプリケーションの使用に対して課金されるようにするには、リクエストごとにアプリケーションから、製品トークンとユーザートークンの 2 つのトークンが送信される必要があります。

製品トークンは開発者の製品を特定します。提供する Amazon DevPay 製品ごとに 1 つの製品トークンが必要です。ユーザートークンは、製品ごとにユーザーを特定します。ユーザーと製品の組み合わせそれぞれに 1 つずつユーザートークンが必要です。例えば、2 つの製品を提供していて、あるユーザーがどちらもサブスクライブしている場合は、それぞれの製品について個別のユーザートークンを取得する必要があります。

製品トークンとユーザートークンの取得については、『*Amazon DevPay Amazon DevPay Getting Started Guide*』を参照してください。

Amazon S3 および Amazon DevPay の認証

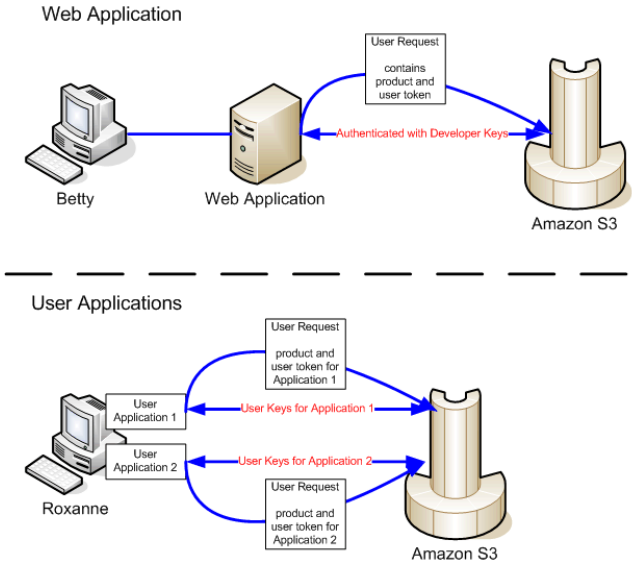
トークンのメカニズムはカスタマーと製品を一意に特定しますが、認証は行いません。

通常、アプリケーションは、開発者のアクセスキー ID とシークレットアクセスキーを使って Amazon S3 と直接通信します。Amazon DevPay については、Amazon S3 認証の動作はやや異なります。

開発者の Amazon DevPay 製品がウェブアプリケーションの場合は、シークレットアクセスキーをサーバー上に安全に保管し、リクエスト実行対象のカスタマーをユーザートークンを使って指定することができます。

ただし、開発者の Amazon S3 アプリケーションがカスタマーのコンピュータにインストールされている場合は、各インストールについてアプリケーションがアクセスキー ID とシークレットアクセスキーを取得し、それらを使用して Amazon S3 と通信する必要があります。

次の図は、ウェブアプリケーションとユーザーアプリケーションの認証の違いを示しています。



Amazon S3 バケットの制限

各カスタマーは、開発者が販売する各 Amazon DevPay 製品ごとに、最大 100 個のバケットを持つことができます。例えば、あるカスタマーが開発者の製品のうち 3 つを使用している場合、そのカスタマーは、最大 300 個のバケット（100 * 3）に加え、開発者の Amazon DevPay 製品以外のバケット（他の開発者の Amazon DevPay 製品やカスタマー個人の AWS アカウントのバケット）を持つことができます。

Amazon S3 および Amazon DevPay のプロセス

Amazon DevPay のプロセスの概要を次に示します。

プロセスを起動

1	カスタマーが、Amazon を通じて開発者の製品にサインアップします。
2	カスタマーがアクティベーションキーを受け取ります。
3	カスタマーがアクティベーションキーを開発者のアプリケーションに入力します。
4	開発者のアプリケーションが Amazon と通信し、ユーザーのトークンを取得します。アプリケーションがユーザーのコンピューターにインストールされている場合は、カスタマーに代わってアクセスキー ID とシークレットアクセスキーも取得します。
5	アプリケーションがカスタマーのために Amazon S3 リクエストを実行する際、ユーザートークンとアプリケーションの製品トークンを提供します。アプリケーションがカスタマーのコンピューターにインストールされている場合は、カスタマーの認証情報を使用して認証を行います。

6	Amazon はカスタマーのトークンと製品トークンを使用して、Amazon S3 の使用に対して請求する相手を判断します。
7	Amazon は月に 1 度使用データを処理し、開発者が定義した条件にしたがってカスタマーに請求を行います。
8	AWS は定額の Amazon DevPay 手数料を差し引き、残額を開発者に支払います。その後、AWS は、カスタマーの操作によって生じた Amazon S3 使用料と、料率に基づく Amazon DevPay 料金をお客様に個別に請求します。

追加情報

Amazon DevPay の使用、セットアップ、および統合については、『*Amazon DevPay*』を参照してください。

エラー処理

Topics

このセクションでは、REST エラーと SOAP エラー、およびその処理方法について説明します。

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

REST エラーレスポンス

Topics

REST リクエストがエラーになった場合、HTTP の応答には次のものが含まれます。

- レスポンス本文としての XML エラードキュメント
- コンテンツタイプ: application/xml
- 該当する HTTP ステータスコード (3xx、4xx、または 5xx)

REST エラーレスポンスの例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<Error>
  <Code>NoSuchKey</Code>
  <Message>The resource you requested does not exist</Message>
  <Resource>/mybucket/myfoto.jpg</Resource>
```

```
  <RequestId>4442587FB7D0A2F9</RequestId>
</Error>
```

Amazon S3 エラーの詳細については、「ErrorCodeList」を参照してください。

レスポンスヘッダー

すべてのオペレーションで返されるレスポンスヘッダーを次に示します。

- *x-amz-request-id:* システムによって各リクエストに割り当てられた一意の ID。万一 Amazon S3 の使用時に問題が発生した場合でも、Amazon はこの ID を使用して問題のトラブルシューティングを行うことができます。
- *x-amz-id-2:* トラブルシューティングに役立つ特殊なトークン。

エラーレスポンス

Topics

Amazon S3 リクエストがエラーになると、クライアントはエラーレスポンスを受け取ります。エラーレスポンスの正確な形式は API 固有です。例えば、REST エラーレスポンスは SOAP エラーレスポンスとは異なります。ただし、すべてのエラーレスポンスには共通の要素があります。

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

エラーコード

エラーコードは、エラー状態を個別に識別する文字列です。エラーを検出してタイプ別に処理するプログラムによって読み取りおよび解釈されるためのものです。多くのエラーコードは SOAP と REST API で共通ですが、API 固有のものもあります。例えば、NoSuchKey は共通ですが、UnexpectedContent は、無効な REST リクエストへのレスポンスでのみ発生します。SOAP フォルトコードには、どの場合でも、エラーコードの表に示すプレフィックスが付いているので、NoSuchKey エラーは SOAP では実際には Client.NoSuchKey として返されます。

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

エラーメッセージ

エラーメッセージには、エラー状態の一般的な説明が英語で含まれます。これは人が理解できるようにするためのものです。シンプルなプログラムの場合、処理できない、または処理されないエラー状態が

発生すると、メッセージがエンドユーザーに直接表示されます。より徹底したエラー処理を備え、適切に国際化されている洗練されたプログラムでは、エラーメッセージが無視される傾向にあります。

詳細情報

多くのエラーレスポンスには、プログラミングエラーを診断する開発者が読んで理解できるように追加の構造化データが含まれています。例えば、REST PUT リクエストと共に Content-MD5 ヘッダーを送信し、これがサーバーで計算されたダイジェストと一致しない場合、BadDigest エラーが発生します。エラーレスポンスには、計算されたダイジェスト、指定されたダイジェストなど、詳細要素も含まれます。開発中にこの情報を使用してエラーを診断できます。実稼働では、正常に動作するプログラムによりこの情報がエラーログに記録されることがあります。

SOAP エラーレスポンス

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

SOAP の場合、エラー結果は SOAP フォルトとして HTTP レスポンスコード 500 と共にクライアントに返されます。SOAP フォルトを受け取っていない場合、リクエストは成功しています。Amazon S3 SOAP フォルトコードは、標準の SOAP 1.1 フォルトコード (「Server」または「Client」) に Amazon S3 固有のエラーコードが連結した構成になっています。例えば、「Server.InternalError」や「Client.NoSuchBucket」などです。SOAP フォルト文字列要素には、人が読むことができる英語の汎用エラーメッセージが含まれます。また、SOAP フォルト詳細要素には、エラーに関連するさまざまな情報が含まれます。

例えば、「Fred」というオブジェクトを削除しようとして、このオブジェクトが存在しなかった場合、SOAP レスポンスの本文には「NoSuchKey」という SOAP エラーが含まれます。

Example

```
<soapenv:Body>
  <soapenv:Fault>
    <Faultcode>soapenv:Client.NoSuchKey</Faultcode>
    <Faultstring>The specified key does not exist.</Faultstring>
    <Detail>
      <Key>Fred</Key>
    </Detail>
  </soapenv:Fault>
</soapenv:Body>
```

Amazon S3 エラーの詳細については、「ErrorCodeList」を参照してください。

&S3 のエラーに関するベストプラクティス

Amazon S3 と連係するアプリケーションを設計するときは、Amazon S3 のエラーを適切に処理することが重要です。このセクションでは、アプリケーションの設計時に考慮すべき問題について説明します。

InternalError の場合は再試行する

内部エラーは Amazon S3 内で発生するエラーです。

InternalError レスポンスを受け取った場合、リクエストが処理されていない可能性があります。例えば、PUT リクエストが InternalError を返す場合、それ以降の GET は古い値を取得することもあれば、更新された値を取得することもあります。

Amazon S3 が InternalError レスポンスを返す場合は、リクエストを再試行してください。

SlowDown エラーを繰り返すアプリケーションの調整

S3 は分散システムの常として、意図的であるかどうかに関係なくリソースの過剰な消費を検出して対応する保護メカニズムを備えています。SlowDown エラーは、リクエストレートが高いためにこのようなメカニズムのいずれかが引き起こされた場合に発生することがあります。リクエストレートを低くすると、この種のエラーは減るかなくなります。一般に、ほとんどのユーザーはこのようなエラーを日常的に経験することはありません。ただし、詳細を知りたい場合や、SlowDown エラーが頻繁にまたは予期せず発生する場合は、Amazon S3 開発者フォーラム (http://aws.amazon.com/premiumsupport/) に投稿するか、または AWS プレミアムサポート (http://aws.amazon.com/premiumsupport/) にサインアップしてください。

エラーを分離する

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

Amazon S3 には、SOAP と REST API の両方で使用されるエラーコードが用意されています。SOAP API は標準の Amazon S3 エラーコードを返します。REST API は標準の HTTP サーバーのように見え、既存の HTTP クライアント (例: ブラウザ、HTTP クライアントライブラリ、プロキシ、キャッシュなど) を操作するように設計されています。HTTP クライアントがエラーを適切に処理できるように、各 Amazon S3 エラーは HTTP ステータスコードにマッピングされています。

HTTP ステータスコードは Amazon S3 エラーコードよりも内容がおおまかであるため、エラーに関する情報は少なくなります。例えば、`NoSuchKey` と `NoSuchBucket` という Amazon S3 エラーはどちらも `HTTP 404 Not Found` ステータスコードにマッピングされます。

HTTP ステータスコードに含まれるエラー情報は少なくなりますが、クライアントが HTTP を認識できて Amazon S3 API を認識できない場合も、通常はエラーが適切に処理されます。

このため、エラーを処理するときや Amazon S3 エラーをエンドユーザーに報告するときは、HTTP ステータスコードではなく、エラーに関する情報が多く含まれる Amazon S3 エラーコードを使用します。さらに、アプリケーションをデバッグするときは、人が読み取り可能な XML エラーレスポンスの <Details> 要素も参照してください。

サーバーアクセスのロギング

Topics

Important

このセクションでは、今後のリリースで変更予定のベータ機能について説明します。この機能に関するフィードバックを Amazon S3開発者フォーラムにお寄せください。

Amazon S3 バケットが自らに対するリクエストのアクセスログレコードを生成するように設定できます。アクセスログレコードには、リクエストタイプや、リクエストが動作したリソース、リクエストが処理された日時などのリクエストに関する詳細が含まれます。サーバーアクセスログからは、バケット所有者の制御下にないクライアントからのリクエストの特性についてわかるため、多くのアプリケーションにとって有用です。

デフォルトでは、バケットはサーバーアクセスログを収集しません。サーバーアクセスのロギングを有効にする方法については、「サーバーアクセスのロギング設定 API (p. 462)」を参照してください。

バケットのロギングが有効になると、利用可能なログレコードはログファイルに収集され、1 時間ごとに指定した Amazon S3 バケット経由で配信されます。このプロセスの詳細については、「サーバーアクセスログの配信 (p. 464)」を参照してください。

プログラミングまたは Amazon S3 コンソールで、バケットのロギングを有効または無効にすることができます。コンソールの使用の詳細については、『*Amazon Simple Storage Service Console User Guide*』の「Managing Bucket Logging」を参照してください。

ログファイルの内容を解釈する方法については、「サーバーアクセスログの形式 (p. 466)」を参照してください。

バケットのロギングを有効にする手順については、「サーバーアクセスロギングのセットアップ (p. 470)」を参照してください。

Note

Amazon S3 バケットでサーバーアクセスのロギング機能を有効にするには追加料金はありません。ただし、システムから配信されるすべてのログファイルについては、通常のストレージ料金が発生します (ログファイルはいつでも削除できます)。ログファイルの配信によるデータ転送料金は課金されませんが、配信されたログファイルへのアクセスには通常どおりのデータ転送料金が適用されます。

サーバーアクセスのロギング設定 API

Important

このセクションでは、今後のリリースで変更予定のベータ機能について説明します。この機能に関するフィードバックを Amazon S3開発者フォーラムにお寄せください。

各 Amazon S3 バケットには関連する XML サブリソースがあり、そのバケットのログステータスを調査または変更するために読み書きすることができます。バケットログステータスリソースの XML スキーマは、SOAP と REST で共通です。

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

`BucketLoggingStatus` 要素の構成は次のとおりです。

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<BucketLoggingStatus xmlns="http://doc.s3.amazonaws.com/2006-03-01">
    <LoggingEnabled>
        <TargetBucket>mylogs</TargetBucket>
        <TargetPrefix>access_log-</TargetPrefix>
        <TargetGrants>
     <Grant>
         <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="AmazonCustomerByEmail">
             <EmailAddress>email_address</EmailAddress>
         </Grantee>
         <Permission>permission</Permission>
     </Grant>
        </TargetGrants>
    </LoggingEnabled>
</BucketLoggingStatus>
```

`BucketLoggingStatus` 要素に属する要素のリストは次のとおりです。

- *`LoggingEnabled`*

この要素が存在するということは、サーバーアクセスのロギングがそのバケットで有効になっていることを示します。この要素 (およびすべての入れ子要素) が存在しないということは、ログがそのバケットで無効になっていることを示します。

- *TargetBucket*

  この要素は、サーバーアクセスログの配信先を指定します。ログを記録しているバケット自体も含めて、所有するどのバケットにもログを配信できます。また、複数のバケットのログを1つのバケットに配信するよう設定することもできます。この場合、配信されたログファイルをキーで区別できるように、配信元バケットごとに異なる TargetPrefix を選択する必要があります。

  Note

  配信元と配信先のバケットは同じ場所に存在する必要があります。バケットの場所の制限の詳細については、「バケットとリージョン (p. 90)」を参照してください。

- *TargetPrefix*

  この要素を使用すると、キーのプレフィックスを指定できます。配信されたログファイルはこのキーに基づいて格納されます。ログファイルのキー名の構成については、「サーバーアクセスログの配信 (p. 464)」を参照してください。

- *TargetGrants*

  バケット所有者には、そのバケットに配信されるすべてのログに対する FULL_CONTROL が自動的に与えられます。このオプションの要素を使用すると、他のユーザーにアクセス許可を与えることができます。指定された TargetGrants は、デフォルトの ACL に追加されます。ACL の詳細については、「アクセスコントロールリスト (p. 8)」を参照してください。

サーバーアクセスのロギングを有効にするには、入れ子になった LoggingEnabled 要素で BucketLoggingStatus を設定または PUT します。サーバーアクセスのロギングを無効にするには、空の BucketLoggingStatus 要素を設定または PUT します。

REST の場合、バケット「mybucket」の BucketLoggingStatus リソースのアドレスは http://s3.amazonaws.com/*mybucket*?logging です。このリソースには PUT メソッドと GET メソッドが有効です。例えば、次のリクエストは mybucket の BucketLoggingStatus リソースを取得します。

```
GET ?logging HTTP/1.1
Host: mybucket.s3.amazonaws.com
Date: Wed, 25 Nov 2009 12:00:00 GMT
Authorization: AWS YOUR_AWS_ACCESS_KEY_ID:YOUR_SIGNATURE_HERE

HTTP/1.1 200 OK
Date: Wed, 25 Nov 2009 12:00:00 GMT
Connection: close
Server: AmazonS3

<?xml version="1.0" encoding="UTF-8"?>
<BucketLoggingStatus xmlns="http://doc.s3.amazonaws.com/2006-03-01">
  <LoggingEnabled>
    <TargetBucket>mybucketlogs</TargetBucket>
    <TargetPrefix>mybucket-access_log-/</TargetPrefix>
        <TargetGrants>
     <Grant>
         <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="AmazonCustomerByEmail">
```

```
            <EmailAddress>user@company.com</EmailAddress>
         </Grantee>
         <Permission>READ</Permission>
      </Grant>
         </TargetGrants>

  </LoggingEnabled>
</BucketLoggingStatus>
```

SOAP の場合、BucketLoggingStatus リソースを操作するには、SOAPSetBucketLoggingStatus オペレーションと SOAPGetBucketLoggingStatus オペレーションを使用します。

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

Amazon S3 では、指定された `BucketLoggingStatus` に対して設定または `PUT` するときに、その有効性を確認します。`TargetBucket` が存在しない場合、所有者が異なる場合、または適切なアクセス許可がない場合は、`InvalidTargetBucketForLogging` エラーが発生します。指定された `BucketLoggingStatus` ドキュメントの XML の書式に誤りがあるか、Amazon の公開済みスキーマに一致しない場合、`MalformedXMLError` が発生します。

BucketLoggingStatus の変更が有効になるまでの期間

バケットのログステータスの変更は、設定 API ではすぐに確認できますが、実際にログファイルの配信に反映されるには時間がかかります。例えば、バケットのログを有効にする場合、その後数時間に行われるリクエストは記録されることもあれば、されないこともあります。また、ログの配信先バケットをバケット A からバケット B に変更する場合、その後数時間は引き続きバケット A に配信されるログもあれば、新しい配信先バケット B に配信されるログもあります。どの場合も、最終的には新しい設定が有効になるので、他の操作は不要です。

サーバーアクセスログの配信

Important

このセクションでは、今後のリリースで変更予定のベータ機能について説明します。この機能に関するフィードバックを Amazon S3開発者フォーラムにお寄せください。

サーバーアクセスログは、選択したバケットに書き込まれます。ログの配信元のバケットにすることも、別のバケットにすることもできます。別のバケットを選択する場合、その所有者は配信元バケットと同じである必要があります。異なる場合、ログは配信されません。

Note

配信元と配信先のバケットは同じ場所に存在する必要があります。バケットの場所の制限の詳細については、「バケットとリージョン (p. 90)」を参照してください。

バケットに配信されたログファイルは、キーに基づいて次の形式で格納されます。

```
TargetPrefixYYYY-mm-DD-HH-MM-SS-UniqueString
```

キーの YYYY、mm、DD、HH、MM、および SS はそれぞれ、ログファイル配信日時の年、月、日、時、分、および秒を表します。

ある時点で配信されたログファイルには、その時点より前に書き込まれたレコードが含まれます。特定の期間のすべてのログレコードが配信されたかどうかを知る方法はありません。

キーの TargetPrefix コンポーネントは、バケット所有者によってログ設定 API を使用して提供される文字列です。詳細については、「サーバーアクセスのロギング設定 API (p. 462)」を参照してください。

キーの UniqueString コンポーネントに意味はありません。ログ処理ソフトウェアでは無視してください。

古いログファイルは自動削除されません。サーバーログを蓄積したくない場合は手動で削除する必要があります。これを行うには、List オペレーションに `prefix` パラメータを指定して、削除する古いログを指定します。詳細については、「オブジェクトキーのリスト作成 (p. 241)」を参照してください。

アクセスコントロール操作

ログファイルは、`http://acs.amazonaws.com/groups/s3/LogDelivery` グループのメンバーの ID に基づいて配信先バケットに書き込まれます。このような書き込みは、通常のアクセスコントロールの制約に従います。このため、配信先バケットのアクセスコントロールポリシーがログ配信グループに `WRITE` アクセスを許可していない場合、ログは配信されません。ログファイルが正しく配信されるようにするには、ログ配信グループにも配信先バケットの `READ_ACP` 許可を与える必要があります。アクセスコントロールリストとグループの詳細については、「アクセスコントロール (p. 294)」を参照してください。配信先バケットのアクセスコントロールポリシーの正しい設定の詳細については、「サーバーアクセスロギングのセットアップ (p. 470)」を参照してください。

配信先バケットに作成されたログファイルには、アクセスコントロールリストのエントリがあります。このエントリは、バケット所有者への FULL_CONTROL の付与、および *`TargetGrants`* 要素で指定された任意のユーザーに対する付与で構成されています。

ベストエフォート型のサーバーログ配信

サーバーアクセスのロギングはベストエフォート型です。バケットのログ配信が適切に設定されている場合、ほとんどのリクエストでログレコードが配信され、ほとんどのログレコードは記録後数時間以内に配信されることを期待できます。

ただし、サーバーログ機能はベストエフォート型で提供されます。サーバーログの完全性や適時性は保証されません。リクエストのログレコードが、リクエストが実際に処理されてからかなり後に配信されたり、配信すらされないこともあり得ます。サーバーログの目的は、バケットのトラフィックの特性をその所有者に知らせることです。すべてのリクエストを完全に報告するためのものではありません。

使用状況レポートの整合性

ベストエフォート型というサーバーログ機能の特性にしたがい、AWS ポータルで利用できる使用状況レポートに含まれる使用状況は、配信されたサーバーログ内のすべてのリクエストには対応していない可能性があります。

サーバーアクセスログの形式

Important

このセクションでは、今後のリリースで変更予定のベータ機能について説明します。この機能に関するフィードバックを Amazon S3開発者フォーラムにお寄せください。

ログファイルは、改行で区切られた一連のログレコードで構成されます。ログレコードの表示順は不定です。各ログレコードは 1 つのリクエストを表し、次の表に示すフィールドで構成されます。各フィールドはスペースで区切られます。

フィールド名	エントリ例	注意事項
Bucket Owner	`314159b66967d86f031c7249d1d9a8024 9109428335cd0ef1cdc487b4566cb1b`	配信元バケット所有者の正規ユーザー ID。
バケット	`mybucket`	リクエストの処理対象のバケットの名前。システムで受け取ったリクエストの形式に誤りがあり、バケットを特定できない場合、そのリクエストはサーバーアクセスログに表示されません。
時間	`[04/Aug/2006:22:34:02 +0000]`	リクエストを受け取った時刻。形式は `strftime()` の用語を使用し、`[%d/%b/%Y:%H:%M:%S %z]` になります。
Remote IP	`72.21.206.5`	リクエスタの表面上のインターネットアドレス。中間プロキシやファイアウォールにより、リクエストを作成したマシンの実際のアドレスが不明確になる場合があります。
Requester	`314159b66967d86f031c7249d1d9a80 249109428335cd0ef1cdc487b4566cb1b`	リクエスタの正規ユーザー ID。未認証リクエストの場合は「Anonymous」という文字列。この識別子は、アクセスコントロールに使用されるものと同じです。
Request ID	`3E57427F33A59F07`	リクエスト ID は、各リクエストを一意に識別するために Amazon S3 によって生成される文字列です。
オペレーション	`SOAP.CreateBucket` または `REST.PUT.OBJECT`	SOAP.*operation*、REST.*HTTP_method*.*resource_type*、または *WEBSITE.HTTP_method.resource_type*

フィールド名	エントリ例	注意事項
キー	`/photos/2006/08/puppy.jpg`	リクエストの URL エンコードされた「key」部分、オペレーションがキーパラメータを取らない場合は「-」。
Request-URI	`"GET /mybucket/photos/2006/08/puppy.jpg?x-foo=bar"`	HTTP リクエストメッセージの Request-URI の部分。
HTTP status	`200`	レスポンスの HTTP ステータスの数値。
エラーコード	`NoSuchBucket`	Amazon S3 エラーコード (p. 458)。エラーがない場合は「-」。
Bytes Sent	`2662992`	送信されたレスポンスのバイト数（HTTP プロトコルオーバーヘッドを除きます）。ゼロの場合は「-」。
Object Size	`3462992`	該当するオブジェクトの合計サイズ。
Total Time	`70`	サーバーから見た、リクエストの転送中の時間数（ミリ秒単位）。これは、リクエストが受信されてから、レスポンスの最終バイトが送信されるまでの時間を計測した値です。クライアント側での計測値は、ネットワークレイテンシーにより長くなる場合があります。
Turn-Around Time	`10`	Amazon S3 でリクエストの処理に要した時間数（ミリ秒単位）。これは、リクエストの最終バイトが受信されてから、レスポンスの先頭バイトが送信されるまでの時間を計測した値です。
Referrer	`"http://www.amazon.com/webservices"`	HTTP Referrer ヘッダーの値（存在する場合）。一般に、HTTP ユーザーエージェント（例: ブラウザ）はこのヘッダーをリクエスト作成時のリンクまたは埋め込みページの URL に設定します。
User-Agent	`"curl/7.15.1"`	HTTP User-Agent ヘッダーの値。
Version Id	3HL4kqtJvjVBH40Nrjfkd	リクエストのバージョン ID。オペレーションが *versionId* パラメータを取らない場合は「-」。

フィールドが「-」に設定されている場合、データが不明であるか利用できないこと、またはこのリクエストに該当しないフィールドであることを示します。

カスタムアクセスログ情報

カスタムの query-string パラメータをリクエストの URL に追加することで、カスタム情報をリクエストのアクセスログレコードに含めることができます。Amazon S3 では、「x-」で始まる query-string パラメータは無視されますが、これらのパラメータは、ログレコードの `Request-URI` フィールドの一部として、リクエストのアクセスログレコードに追加されます。例えば、「s3.amazonaws.com/mybucket/photos/2006/08/puppy.jpg?x-user=johndoe」に関する GET リクエストは、「s3.amazonaws.com/mybucket/photos/2006/08/puppy.jpg」に関するリクエストと同じように機能します。ただし、関連するログレコードの `Request-URI` フィールドに、「x-user=johndoe」という文字列が追加される点が異なります。この機能は REST インターフェイスでのみ利用できます。

拡張可能なサーバーアクセスログの形式

新しいフィールドを各行末に追加することで、アクセスログレコードの形式をいつでも拡張することができます。サーバーアクセスログを解析するコードは、後に続くフィールドを理解できなくても処理するよう作成する必要があります。

コピーオペレーションの追加ロギング

コピーオペレーションには GET と PUT が含まれます。このため、コピーオペレーションの実行時には 2 つのログレコードが記録されます。前述の表では、コピーオペレーションの PUT 部分に関連するフィールドを説明しています。次の表では、コピーオペレーションの GET 部分に関連するフィールドを説明します。

フィールド名	エントリ例	注意事項
Bucket Owner	`314159b66967d86f031c7249d1d9a8024 9109428335cd0ef1cdc487b4566cb1b`	コピー対象のオブジェクトを格納するバケット所有者。
バケット	`mybucket`	コピー対象のオブジェクトを格納するバケットの名前。
時間	`[04/Aug/2006:22:34:02 +0000]`	リクエストを受け取った時刻。形式は `strftime()` の用語を使用し、`[%d/%B/%Y:%H:%M:%S %z]` になります。
Remote IP	`72.21.206.5`	リクエスタの表面上のインターネットアドレス。中間プロキシやファイアウォールにより、リクエストを作成したマシンの実際のアドレスが不明確になる場合があります。
Requester	`314159b66967d86f031c7249d1d9a80 249109428335cd0ef1cdc487b4566cb1b`	リクエスタの正規ユーザー ID。未認証リクエストの場合は「Anonymous」という文字列。この識別子は、アクセスコントロールに使用されるものと同じです。

フィールド名	エントリ例	注意事項
Request ID	`3E57427F33A59F07`	リクエスト ID は、各リクエストを一意に識別するために Amazon S3 によって生成される文字列です。
オペレーション	`REST.COPY.OBJECT_GET`	SOAP.*operation*、REST.*HTTP_method*.*resource_type*、または *WEBSITE.HTTP_method.resource_type*
キー	`/photos/2006/08/puppy.jpg`	コピー対象のオブジェクトの「key」部分。オペレーションがキーパラメータを取らない場合は「-」。
Request-URI	`"GET /mybucket/photos/2006/08/puppy.jpg?x-foo=bar"`	HTTP リクエストメッセージの Request-URI の部分。
HTTP status	`200`	コピーオペレーションの GET 部分の HTTP ステータスの数値。
エラーコード	`NoSuchBucket`	コピーオペレーションの GET 部分の Amazon S3 エラーコード (p. 458)。エラーがない場合は「-」。
Bytes Sent	`2662992`	送信されたレスポンスのバイト数（HTTP プロトコルオーバーヘッドを除きます）。ゼロの場合は「-」。
Object Size	`3462992`	該当するオブジェクトの合計サイズ。
Total Time	`70`	サーバーから見た、リクエストの転送中の時間数（ミリ秒単位）。これは、リクエストが受信されてから、レスポンスの最終バイトが送信されるまでの時間を計測した値です。クライアント側での計測値は、ネットワークレイテンシーにより長くなる場合があります。
Turn-Around Time	`10`	Amazon S3 でリクエストの処理に要した時間数（ミリ秒単位）。これは、リクエストの最終バイトが受信されてから、レスポンスの先頭バイトが送信されるまでの時間を計測した値です。
Referrer	`"http://www.amazon.com/webservices"`	HTTP Referrer ヘッダーの値（存在する場合）。一般に、HTTP ユーザーエージェント（例: ブラウザ）はこのヘッダーをリクエスト作成時のリンクまたは埋め込みページの URL に設定します。

フィールド名	エントリ例	注意事項
User-Agent	`"curl/7.15.1"`	HTTP User-Agent ヘッダーの値。
Version Id	3HL4kqtJvjVBH40Nrjfkd	コピー対象のオブジェクトのバージョン ID。x-amz-copy-source ヘッダーでコピー元の一部として *versionId* パラメータを指定しなかった場合は「-」。

サーバーアクセスロギングのセットアップ

Important

このセクションでは、今後のリリースで変更予定のベータ機能について説明します。この機能に関するフィードバックを Amazon S3開発者フォーラムにお寄せください。

Amazon S3 のサーバーアクセスのロギング機能を使用して、所有するバケットのアクセスログファイルを生成できます。このようなログファイルは、所有するバケット (別のバケットも選択可) に書き込むことで配信されます。配信されたアクセスログは、通常のオブジェクトとして自由に読み取ったり、リストを表示したり、削除したりすることができます。

以下の手順は、既存のバケットのサーバーアクセスのロギングを有効にし、それらのログを log 専用に作成する新しいバケットに配信する手順です。ログを記録するバケットは「mybucket」という名前で、アクセスログ保存用に新しく作成するバケットは「mylogs」という名前であるとします。したがって、ログの配信元バケットが「mybucket」、ログの配信先バケットが「mylogs」となります。例に表示される「mybucket」や「mylogs」は、ご自分のログを作成するバケットやアクセスログを保存するバケットの名前に置き換えてください。

このチュートリアルでは、s3curl (s3curl.pl サンプルプログラムを参照) を使用して Amazon S3 REST API を操作します。使用する s3curl が最新版であることを確認してください。このチュートリアルに対応するために更新されています。s3curl を起動した後は必ず `200 OK` HTTP レスポンスを確認してください。別のレスポンスコードが返された場合は、XML エラーレスポンスに不具合に関する情報が含まれている可能性が高いため、XML エラーレスポンスを参照してください。

配信先バケットの準備

配信先バケットを準備するには

1. 最初に、ログを既存のバケットに配信するか、アクセスログファイル専用のバケットを新規作成するかを決定します。ログ用の配信先バケットを新規作成するコマンドは次のとおりです。既定の ACL 引数は、このバケットにログファイルを書き込むシステム許可を付与することに注意してください。

Note

配信元と配信先のバケットは同じ場所に存在する必要があります。バケットの場所の制限の詳細については、「バケットとリージョン (p. 90)」を参照してください。

```
$ ./s3curl.pl --id YOUR_AWS_ACCESS_KEY_ID --key YOUR_AWS_SECRET_ACCESS_KEY -
-acl log-delivery-write --put /dev/null -- -s -v http://s3.amazonaws.com/mylogs
```

2. ログ用のバケットを新規作成した場合は、次のセクションにスキップします。それ以外の場合は、アクセスログファイルを既存のバケットに配信するために、そのバケットのアクセスコントロールポリシーを手動で変更する必要があります。配信先バケットの ?acl サブリソースを取得してローカルファイルに保存します。

```
$ ./s3curl.pl --id YOUR_AWS_ACCESS_KEY_ID --key YOUR_AWS_SECRET_ACCESS_KEY -
- -s -v 'http://s3.amazonaws.com/mylogs?acl' > mylogs.acl
```

3. ログリソースのローカルコピーをテキストエディタで開き、新しい <Grant> 要素を <AccessControlList> セクションに挿入します。これにより、ログ配信グループに、バケットへの WRITE および READ_ACP アクセス許可が与えられます。

```
<Grant>

    <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="Group">

        <URI>http://acs.amazonaws.com/groups/s3/LogDelivery</URI>

    </Grantee>

    <Permission>WRITE</Permission>

</Grant>

<Grant>

    <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="Group">

        <URI>http://acs.amazonaws.com/groups/s3/LogDelivery</URI>

    </Grantee>

    <Permission>READ_ACP</Permission>

</Grant>
```

4. 最後に、変更したアクセスコントロールポリシーをAmazon S3 に書き戻すことによって、適用します。

```
$ ./s3curl.pl --id YOUR_AWS_ACCESS_KEY_ID --key YOUR_AWS_SECRET_ACCESS_KEY -
-put mylogs.acl -- -s -v 'http://s3.amazonaws.com/mylogs?acl'
```

配信元バケットでのサーバーアクセスのロギングの有効化

配信先バケットをログファイルを受け取れるようになったので、配信元バケットの ?logging サブリソースを更新して、サーバーアクセスのロギングを有効にします。このリソースの読み取りや書き込みを行うには、バケット所有者である必要があります。

次の例に示すコマンドを使用して、?logging サブリソースを取得して変更します。

Example

```
$ ./s3curl.pl --id YOUR_AWS_ACCESS_KEY_ID --key YOUR_AWS_SECRET_ACCESS_KEY --
-s -v 'http://s3.amazonaws.com/mybucket?logging' > mybucket.logging
```

テキストエディタで mybucket.logging を開き、<LoggingSettings> セクションのコメントを解除します。<TargetBucket> と <TargetPrefix> の内容をそれぞれ「mylogs」と「mybucket-access_log-」に置き換えます。

また、バケット内のログファイルへのアクセス許可をユーザーに付与するには、<TargetGrants> セクションでユーザーを指定します。ユーザーの指定にはメールアドレス (EmailAddress) か正規のユーザー ID (CanonicalUser) を使用できます。アクセス許可には READ、WRITE、および FULL_CONTROL があります。次のような結果が返るのを確認してください。

Example

```
<?xml version="1.0" encoding="UTF-8"?>

<BucketLoggingStatus xmlns="http://doc.s3.amazonaws.com/2006-03-01">

    <LoggingEnabled>

        <TargetBucket>mylogs</TargetBucket>

        <TargetPrefix>mybucket-access_log-/</TargetPrefix>

            <TargetGrants>

         <Grant>

         <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="AmazonCustomerByEmail">

             <EmailAddress>user@company.com</EmailAddress>

         </Grantee>

         <Permission>READ</Permission>

     </Grant>

        </TargetGrants>

    </LoggingEnabled>

</BucketLoggingStatus>
```

Note

認証については、「アクセスコントロール (p. 294)」を参照してください。

ドキュメントを Amazon S3 の ?logging サブリソースに書き戻すことで、変更を適用します。

Example

```
$ ./s3curl.pl --id YOUR_AWS_ACCESS_KEY_ID --key YOUR_AWS_SECRET_ACCESS_KEY -
-put mybucket.logging -- -s -v 'http://s3.amazonaws.com/mybucket?logging'
```

変更を確認するには、?logging サブリソースを取得して、書き込んだ内容と比較します。

これでサーバーアクセスのロギングが有効になりました。配信元バケットにいくつかのリクエストを行うと、数時間以内に配信先バケットへのアクセスログ配信が開始されます。

バケットのサーバーロギングの無効化

前述の手順と同じ方法で `?logging` サブリソースを取得、変更、および適用します。異なるのは、テキストエディタで `<LoggingEnabled>` 要素を削除することです。

Note

ログの変更は直ちに有効になるわけではありません。ログを無効にしても、しばらくの間はログが配信されます。

AWS dynamo と Explorer の使用

Topics

Amazon S3 でのアプリケーション開発に AWS SDK を使用できます。AWS SDK for Java、AWS SDK for PHP、および AWS SDK for .NET は、基盤となる REST API をラップして、プログラミング作業を簡素化します。AWS を使用して接続されるモバイルアプリケーションの構築用に、モバイル SDK も提供されています。このセクションでは、Amazon S3 アプリケーションの開発に AWS SDK を使用する方法を概説します。また、このガイドで提供されている AWS SDK コードサンプルのテスト方法も説明します。

AWS SDK に加え、Visual Studio および Eclipse for Java IDE で使用できる AWS Explorer も提供されています。この場合、SDK と Explorer が、AWS Toolkit としてバンドルされて提供されます。

AWS Toolkit for Eclipse

AWS Toolkit for Eclipse には、AWS SDK for Java と AWS Explorer for Eclipse が両方とも同梱されています。AWS Explorer for Eclipse は、AWS を使用して開発者が Java アプリケーションを容易に開発、デバッグ、およびデプロイできるようにする Eclipse for Java IDE のオープンソースプラグインです。使いやすい GUI インターフェイスで、Amazon S3 を含む AWS インフラストラクチャにアクセスし管理することができます。アプリケーションの開発中に、バケットとオブジェクトの管理や IAM ポリシーの設定といった一般的な作業をすべて Eclipse for Java IDE のコンテキストから実行できます。設定手順については、「Setting Up the AWS Toolkit for Eclipse」を参照してください。Explorer を使った Amazon S3 の使用例については、「Viewing and Editing Amazon S3 Buckets」を参照してください。

AWS Toolkit for Visual Studio

AWS Explorer for Visual Studio は、開発者がアマゾン ウェブ サービスを使用して .NET アプリケーションを容易に開発、デバッグ、およびデプロイできるようにする Microsoft Visual Studio の拡張機能です。使いやすい GUI インターフェイスで、Amazon S3 を含む AWS インフラストラクチャにアクセスし管理することができます。アプリケーションの開発中に、バケットとオブジェクトの管理や IAM ポリシーの設定といった一般的な作業をすべて Visual Studio のコンテキストから実行できます。設定手順については、「Setting Up the AWS Toolkit for Visual Studio」を参照してください。 Explorer を使った Amazon S3 の使用例については、「Using Amazon S3 from AWS Explorer」を参照してください。

AWS SDK

SDK のみをダウンロードできます。SDK ライブラリのダウンロードについては、「Sample Code Libraries」を参照してください。

AWS SDK for Java の使用

AWS SDK for Java は、Amazon S3 バケットおよびオブジェクトのオペレーションを行うための API を提供しています。オブジェクト操作用として、単一操作で複数のオブジェクトをアップロードするための API に加え、大きなオブジェクトをパートごとにアップロードするための API も用意されています (「マルチパートアップロード API を使用したオブジェクトのアップロード (p. 178)」を参照)。API では、高レベル API を使うか低レベル API を使うかを選択できます。

低レベル API

低レベル API は、バケットやオブジェクトに適用される作成、更新、削除などの基礎的な Amazon S3 REST オペレーションに対応します。大きなオブジェクトをアップロードする際に低レベルのマルチパートアップロード API を使用すると、マルチパートアップロードを一度停止して再開したり、アップロード中にパートサイズを変更したり、事前にデータサイズがわからない状態でアップロードを開始するなど、さまざまな制御が可能となります。このような制御の必要がない場合は、高レベル API を使ってオブジェクトをアップロードしてください。

高レベル API

SDK では、オブジェクトのアップロード用に、`TransferManager` クラスによる高レベルの抽象化を提供しています。高レベル API はより単純な API で、ほんの数行のコードを使って、ファイルやストリームを Amazon S3 にアップロードすることができます。データをアップロードする際、前述の低レベル API のセクションで説明したような各種の制御が必要ない場合は、高レベル API を使用してください。

小さなサイズのデータの場合は、`TransferManager` API により、単一の操作でデータをアップロードできます。ただしデータのサイズが特定のしきい値を超えると、`TransferManager` はマルチパートアップロード API の使用に切り替わります。可能であれば、`TransferManager` は複数のスレッドを使って複数のパートを同時にアップロードします。あるパートのアップロードに失敗すると、API は、失敗したパートのアップロードを 3 回まで再試行します。ただし、このオプションは、`TransferManagerConfiguration` クラスを使って設定変更できます。

Note

データのソースにストリームを使っている場合、`TransferManager` クラスは同時アップロードを実行しません。

Java API の構成

この API は、AWS SDK for Java 内の以下のパッケージに同梱されています。

- com.amazonaws.services.s3 – Amazon S3 バケットおよびオブジェクトのオペレーションを行うための実装 API を同梱しています。
  例えば、バケットの作成、オブジェクトのアップロード、オブジェクトの取得、オブジェクトの削除、キーのリストなどを行うためのメソッドが用意されています。
- com.amazonaws.services.s3.transfer – 高レベル API データアップロードを提供します。
  この高レベル API は、Amazon S3 へのオブジェクトのアップロードをさらに簡素化するためのものです。`TransferManager` クラスが含まれています。大きなオブジェクトをパートに分けてアップロードする際に特に有用です。さらに、アップロードするパートの最小パートサイズと、マルチパー

トアップロードを使用する場合のバイト単位のしきい値を設定するために使用できる `TransferManagerConfiguration` クラスも含まれます。

- com.amazonaws.services.s3.model – リクエストを作成しレスポンスを処理するための低レベル API クラスを提供します。
  例えば、オブジェクト取得リクエストを記述するための `GetObjectRequest` クラス、キーのリストリクエストを記述するための `ListObjectRequest` クラス、マルチパートアップロードを開始する際の `InitiateMultipartUploadRequest` クラスと `InitiateMultipartUploadResult` クラスなどが含まれます。

AWS SDK for Java API の詳細については、『AWS SDK for Java API Reference 』を参照してください。

Java コード例のテスト

Java コード例の使用を最も手早く開始する方法は、最新の AWS Toolkit for Eclipse をインストールすることです。最新バージョンのインストールと更新については、http://aws.amazon.com/eclipse を参照してください。以下のタスクは、このセクションで提供されている Java コードサンプルを作成およびテストする手順を示しています。

Java コード例作成の一般的な手順

1	Eclipse で新しい AWS Java プロジェクトを作成します。プロジェクトは AWS SDK for Java を使用できるように事前に設定されているほか、AWS セキュリティ認証情報用の AwsCredentials.properties ファイルが含まれています。
2	任意のセクションからプロジェクトにコードをコピーします。
3	必要なデータを指定してコードを修正します。例えばファイルをアップロードする場合は、ファイルのパスとバケットの名前を指定します。
4	コードを実行します。AWS Management Console を使用して、オブジェクトが作成されることを確認します。AWS Management Console の詳細については、http://aws.amazon.com/console/ を参照してください。

AWS SDK for .NET の使用

Topics

AWS SDK for .NET は、Amazon S3 のバケットとオブジェクトのオペレーションを行うための API を提供しています。オブジェクト操作用として、単一操作で複数のオブジェクトをアップロードするための API に加え、大きなオブジェクトをパートごとにアップロードするための API も用意されています（「マルチパートアップロード API を使用したオブジェクトのアップロード (p. 178)」を参照）。API では、高レベル API を使うか低レベル API を使うかを選択できます。

低レベル API

低レベル API は、バケットやオブジェクトに適用される作成、更新、削除などの基礎的な Amazon S3 REST オペレーションに対応します。大きなオブジェクトをアップロードする際に低レベルのマルチパートアップロード API を使用すると（「マルチパートアップロード API を使用したオブジェクトのアップロード (p. 178)」を参照）、マルチパートアップロードを一度停止して再開したり、アップロー

ド中にパートサイズを変更したり、事前にデータサイズがわからない状態でアップロードを開始するなど、さまざまな制御が可能となります。このような制御の必要がない場合は、高レベル API を使ってオブジェクトをアップロードしてください。

高レベル API

SDK では、オブジェクトのアップロード用に、`TransferUtility` クラスによる高レベルの抽象化を提供しています。高レベル API はより単純な API で、ほんの数行のコードを使って、ファイルやストリームを Amazon S3 にアップロードすることができます。データをアップロードする際、前述の低レベル API のセクションで説明したような各種の制御が必要ない場合は、高レベル API を使用してください。

小さなサイズのデータの場合は、`TransferUtility` API により、単一の操作でデータをアップロードできます。ただしデータのサイズが特定のしきい値を超えると、`TransferUtility` はマルチパートアップロード API の使用に切り替わります。デフォルトでは、複数のスレッドを使用して複数のパートを同時にアップロードします。あるパートのアップロードに失敗すると、API は、失敗したパートのアップロードを 3 回まで再試行します。ただし、このオプションは設定変更できます。

Note

データのソースにストリームを使っている場合、`TransferUtility` クラスは同時アップロードを実行しません。

.NET API の構成

AWS SDK for .NET を使用して Amazon S3 アプリケーションを作成する場合、`AWSSDK.dll` を使用します。このアセンブリの以下の名前空間に、マルチパートアップロード API が用意されています。

- Amazon.S3.Transfer – データをパートごとにアップロードするための高レベル API を提供します。データをアップロードするための、ファイル、ディレクトリ、またはストリームを指定する `TransferUtility` クラスが含まれます。また `TransferUtilityUploadRequest` および `TransferUtilityUploadDirectoryRequest` クラスが含まれており、同時スレッドの数、パートのサイズ、オブジェクトメタデータ、ストレージクラス (STANDARD、REDUCED_REDUNDANCY) 、オブジェクト ACL などの高度な設定を指定できます。
- Amazon.S3 – 低レベル API の実装を提供します。
  これには、Amazon S3 REST マルチパートアップロード API に対応するメソッドが含まれています (「REST API を使用したマルチパートアップロード (p. 214)」を参照) 。
- Amazon.S3.Model – リクエストを作成しレスポンスを処理するための低レベル API クラスを提供します。
  例えば、マルチパートアップロードを開始する際に使用できる `InitiateMultipartUploadRequest` クラスや `InitiateMultipartUploadResponse` クラス、パートをアップロードする際に使用できる `UploadPartRequest` クラスや `UploadPartResponse` クラスなどを提供します。

AWS SDK for .NET API の詳細については、『AWS SDK for .NET Reference』を参照してください。

.NET コード例のテスト

.NET コード例の使用を最も手早く開始する方法は、AWS SDK for .NET をインストールすることです。詳細については、http://aws.amazon.com/sdkfornet を参照してください。以下のタスクは、このセクションで提供されている C# コードサンプルを作成およびテストする手順を示しています。

.NET コード例作成の一般的な手順

1	*AWS Empty Project* テンプレートを使用して、新しい Visual Studio プロジェクトを作成します。
2	[AWS Access Credentials] ダイアログボックス内で、AWS 認証情報を指定します。
3	*AWS Empty Project* テンプレートは、ユーザーの AWS 認証情報および以下の必須参照を使用するよう、あらかじめ `App.config` ファイルで設定されています。 `AWSSDK` `System.Configuration` `App.config` ファイルに、ご自分の認証情報の以下のキーがあることを確認してください。 `<configuration>` `  <appSettings>` `  <add key="AWSAccessKey" value="*** Your access key ID ***"/>` `  <add key="AWSSecretKey" value="*** Your secret key ***"/>` `  </appSettings>` `</configuration>`
4	プロジェクトファイル `Program.cs` 内のコードを、任意のセクションのコードで置き換えます。
5	コードを実行します。AWS Management Console を使用して、オブジェクトが作成されることを確認します。AWS Management Console の詳細については、http://aws.amazon.com/console/ を参照してください。

AWS SDK for PHP の使用と PHP サンプルの実行

AWS SDK for PHP では、Amazon S3 のバケットとオブジェクトのオペレーションを行うための API を使用できます。サービスの低レベル API を使用するか、高レベルの抽象化を使用するかは、選択できます。

この SDK は、AWS SDK for PHP で入手できます。ここには、SDK のインストールおよび使用開始を行うためのドキュメントも掲載されています。

Note

AWS SDK for PHP を使用するための設定は、環境によって、また、どのようにアプリケーションを実行するかによって異なります。このドキュメントに掲載されている例を実行できるようお使いの環境をセットアップする方法については、『AWS SDK for PHP Getting Started Guide』を参照してください。

AWS SDK for PHP のレベル

低レベル API

低レベル API は、バケットやオブジェクトで実行される作成、更新、削除などの基礎的な Amazon S3 REST オペレーションに対応します。この API を使用すると、これらのオペレーションを詳細に制御することができます。例えば、リクエストをバッチ処理して並列に実行したり、マルチパートアップロー

ド API を使用するときに（「マルチパートアップロード API を使用したオブジェクトのアップロード (p. 178)」を参照）、オブジェクトのパートを個別に管理できます。これらの低レベル API 呼び出しは、すべての Amazon S3 レスポンスの詳細を含む結果を返します。

高レベルの抽象化

高レベルの抽象化は、一般的ユースケースを簡素化するためのものです。例えば、低レベル API を使用して大きなオブジェクトをアップロードするには、まず `Aws\S3\S3Client::createMultipartUpload()` を呼び出し、次に `Aws\S3\S3Client::uploadPart()` メソッドを呼び出してオブジェクトのパートをアップロードし、最後に `Aws\S3\S3Client::completeMultipartUpload()` メソッドを呼び出してアップロードを完了します。代わりに、マルチパートアップロードの作成を簡素化する、高レベルの `Aws\S3\Model\MultipartUpload\UploadBuilder` オブジェクトを使用することもできます。

高レベルの抽象化を使用するもう 1 つの例を挙げると、バケットに保管されているオブジェクト（オブジェクトの数は問わない）を列挙する際、AWS SDK for PHP のイテレータ機能を使用してすべてのオブジェクトキーを返すことができます。これを低レベル API で行った場合、レスポンスによって返されるキーは最大 1,000 であるため、バケット内に 1,000 以上のオブジェクトがあると、結果が切り捨てられてしまいます。このため、レスポンスを管理して切り捨てを確認する必要があります。

PHP サンプルの実行

以下の手順では、このガイドに記載した PHP コード例の実行方法を説明します。

PHP コード例を実行するには

1	AWS SDK for PHP をダウンロードし、インストールします。次に、環境が最低要件（『AWS SDK for PHP Getting Started Guide』を参照）を満たしていることを確認します。
2	『AWS SDK for PHP Getting Started Guide』の指示に沿って AWS SDK for PHP をインストールします。使用するインストール方法によっては、PHP 拡張モジュール間の依存性を解決するために、コードを変更しなければならない場合があります。 このドキュメント内の PHP コードサンプルはすべて、Composer dependency manager を使用します。詳しくは AWS SDK for PHP Getting Started Guide に説明があります。各コード例には、依存関係を取り込むための以下の行が含まれています。 `require 'vendor/autoload.php';`
3	ドキュメントからプロジェクトにサンプルコードをコピーします。ご利用の環境によっては、設定ファイルおよび SDK ファイルを参照するコード例に対して、行の追加が必要な場合があります。 たとえば、PHP のサンプルをブラウザにロードするには、以下を PHP コードの先頭に追加し、ウェブアプリケーションディレクトリ（`www` や `htdocs` など）内に PHP ファイル（拡張子は `.php`）として保存します。 `<?php` `header('Content-Type: text/plain; charset=utf-8');` `// Include the AWS SDK using the Composer autoloader` `require 'vendor/autoload.php';`

4	ご利用の設定に従い、コード例をテストします。

AWS アクセスキーの設定

安全のために、AWSアクセスキーIDとシークレットキーを含む設定ファイルを作成することをお勧めします。この手法を使用することで、PHP コード自体へのアクセスキーのハードコーディングを避けることができます。実行時に、新しい Amazon S3 クライアントオブジェクトを作成する場合、クライアントはそのオブジェクトのキーを設定ファイルから取得できます。

以下に、このような設定ファイルの例 (`config.php` という名前) を示します。

```
<?php

return array(
  'includes' => array('_aws'),
  'services' => array(
      'default_settings' => array(
          'params' => array(
              'key'    => '*** Your AWS Access Key ID ***',
              'secret' => '*** Your AWS Secret Key ***'
          )
      )
  )
);
?>
```

以下のコード例は、`config.php` を使用して新しいクライアントをインスタンス化する方法を示しています。

```
use Aws\Common\Aws;

// Instantiate the client with your AWS access keys
$aws = Aws::factory('./config.php');
$client = $aws->get('s3');
```

設定ファイルの設定に関する詳細については、AWS SDK for PHP Migration Guide の「Configuration / Service Builder」セクションを参照してください。

リージョンの設定

設定ファイルを使用する場合は、クライアント用のデフォルトのリージョンを変更できます (以下の例を参照) 。

```
<?php

return array(
  'includes' => array('_aws'),
  'services' => array(
      'default_settings' => array(
          'params' => array(
              'key'    => '*** Your AWS Access Key ID ***',
              'secret' => '*** Your AWS Secret Key ***'
```

```
            'region' => 'US West (Oregon) Region'
        )
    )
  )
);

?>
```

Amazon S3 クライアントを変更すれば、実行時にもリージョンを設定できます。

```
$client->setRegion('eu-west-1'); // EU (Ireland)
```

現在サポートされているリージョンとエンドポイントのリストについては、「リージョンとエンドポイント」を参照してください。

関連リソース

- 「AWS SDK for PHP – Amazon S3」
- 「AWS SDK for PHP」のドキュメント

AWS SDK for Ruby の使用

AWS SDK for Ruby は、Amazon S3 バケットおよびオブジェクトのオペレーションを行うための API を提供しています。オブジェクト操作用として、単一操作で複数のオブジェクトをアップロードしたり、大きなオブジェクトをパートごとにアップロードしたりできます (「Uploading Objects Using Multipart Upload」を参照) 。しかし、単一操作でのアップロード用の API は、大きなオブジェクトを受け付けて背後でパートに分割してアップロードすることもできるため、スクリプト作成の作業を簡素化できます。

Ruby API の構成

AWS SDK for Ruby を使用して Amazon S3 アプリケーションを作成する場合、SDK for Ruby gem を使用する必要があります。詳細については、『AWS SDK for Ruby Getting Started Guide』を参照してください。インストールが完了すると、以下のキークラスを含む API を使用できるようになります。

- AWS::S3 – Ruby SDK の Amazon S3 へのインターフェイスを表します。

  `S3` クラスは、既存のバケットにアクセスしたり新しいバケットを作成したりするための `#buckets` インスタンスメソッドを提供します。
- AWS::S3::Bucket – Amazon S3 バケットを表します。

  `Bucket` クラスは、バケット内のオブジェクトにアクセスするための `#objects` インスタンスメソッドに加え、バケットを削除したり、バケットのポリシーといったバケットに関する情報を返したりするメソッドを提供します。
- AWS::S3::S3Object – キーによって識別される Amazon S3 オブジェクトを表します。
  `S3Object` クラスは、オブジェクトのプロパティを取得および設定するメソッド、オブジェクト保管のためにストレージクラスを指定するメソッド、およびアクセスコントロールリストを使用してオブジェクトのアクセス許可を設定するメソッドを提供します。`S3Object` クラスには、オブジェクトを削除、アップロード、およびコピーするメソッドもあります。オブジェクトをパートごとにアップロードする場合、このクラスが提供するオプションを使用して、アップロードするパートの順序やパートのサイズを指定できます。

AWS SDK for Ruby API の詳細については、『AWS SDK for Ruby API Reference』を参照してください。

Ruby スクリプト例のテスト

Ruby スクリプト例を使う最も簡単な方法は、最新の AWS SDK for Ruby gem をインストールすることです。最新の gem のインストールと更新については、http://aws.amazon.com/sdkforruby/ を参照してください。以下のタスクは、AWS SDK for Ruby が既にインストールされているものとして、Ruby スクリプト例を作成およびテストする手順を示しています。

Ruby スクリプト例を作成およびテストする一般的な手順

1	新しい SDK for Ruby スクリプトを作成し、スクリプトの先頭に以下の行を追加します。 `#!/usr/bin/env ruby` `require 'rubygems'` `require 'aws-sdk'` 最初の行はインタプリタ指示文で、2つの `require` 文によって必要な2つの gem がスクリプトにインポートされます。
2	`AWS::Core::Configuration` クラスを使用して、認証情報を指定します。 `AWS.config(` `  :access_key_id => '*** Provide your access key ***,` `  :secret_access_key => '*** Provide your secret key ***'` `)`
3	任意のセクションからスクリプトにコードをコピーします。
4	必要なデータを指定してコードを修正します。例えばファイルをアップロードする場合は、ファイルのパスとバケットの名前を指定します。
5	スクリプトを実行します。AWS Management Console を使用して、バケットとオブジェクトに加えられた変更を確認します。AWS Management Console の詳細については、http://aws.amazon.com/console/ を参照してください。

AWS SDK for Python (boto) の使用

Boto は Python のパッケージで、 Amazon S3 を含む AWS へのインターフェイスです。Boto の詳細については、『AWS SDK for Python (boto)』を参照してください。SDK for Python の使用を開始するには、『Getting Started with Boto』を参照してください。

付録

この『Amazon Simple Storage Service 開発者ガイド』の付録には以下のセクションがあります。

Topics

付録 A: アクセスポリシー言語

Topics

この付録は、独自のアクセスコントロールポリシーを作成する Amazon S3 ユーザーを対象としています。AWS アカウント ID と基本的な許可のみに基づいてアクセスを許可する場合、独自のポリシーを作成する必要はありません。アクセスを明示的に拒否する場合や、より細かい条件 (リクエストの受信時刻やリクエスタの IP アドレスなど) に基づいてアクセスを許可する場合は、独自のポリシーを作成して AWS にアップロードする必要があります。

Note

独自のポリシーを作成するには、JSON の知識が必要です。詳細については、http://json.org を参照してください。

この付録では主に、理解する必要がある基本概念、ポリシーを記述する方法、AWS がポリシーを評価し、リクエスタにリソースへのアクセスを許可するかどうかを決定するために AWS で使用される口ジックについて説明します。この付録に示す情報の大部分は、特定のサービスに限ったものではありませんが、知っておく必要がある Amazon S3 固有の詳細情報もあります。詳細については、「Amazon S3 ポリシーの特別な情報 (p. 506)」を参照してください。

概要

Topics

このセクションでは、アクセスポリシー言語を使用してポリシーを作成するにあたって理解しておくべき基本的なコンセプトを説明します。また、アクセスポリシー言語と連携したアクセスコントロール法やポリシーの評価方法の一般的なプロセスも合わせて説明します。

主要なコンセプト

以下のセクションでは、access policy language を使用するにあたって理解しておくべきコンセプトを説明します。基本的なものから順に分かりやすく説明していきます。

アクセス権限

*アクセス権限*とは、特定のリソースへのある種のアクセスに対し、許可または拒否をするというコンセプトです。アクセス許可は、基本的に「A は、条件 D に該当する C を対象とするアクション B の実行を許可または禁止されている」という形態をとります。例えば、*Jane* (A) は、IP アドレスが特定の範囲 (D) にある*オブジェクト* (B) を *AcmeProductsBucket* に配置するアクセス許可を付与されています。Jane がオブジェクトをバケットに配置するリクエストを Amazon S3 に送信すると、Jane にアクセス許可があるかどうか、またポリシーでバケット所有者が設定した条件をリクエストが満たしているかどうかがチェックされます。

ステートメント

*ステートメント*とは、access policy language で使用するアクセス権限を定義する書式です。1 つのステートメントで 1 つのアクセス権限を定義します。このステートメントの集まり (コンテナドキュメント) のことを、*ポリシー* (次のコンセプト参照) といいます。

ポリシー

*ポリシー*とは、1 つ以上のステートメントを入れるコンテナの役目を果たすドキュメント (JSON で記述) です。例えば、1 つのポリシーに「Jane は指定されたバケットにオブジェクトを配置することができる」というステートメントと、「Bob は同じバケットにオブジェクトを配置することができない」というステートメント、計 2 つのステートメントを含めることができます。次の図に示すように、これは 2 つのポリシーを設定することと同じシナリオです。

Amazon S3 では、リソースへのアクセスを要求しているユーザーに対してアクセスを許可すべきかどうかを、単一または複数のポリシーのステートメントに記述されている情報に基づいて決定します。

発行者

*発行者*とは、リソース用のアクセス権限についてのポリシーを記述する人物です。発行者は当然のこととして、リソースの所有者でなければなりません。AWS のサービスでは、リソースを所有していない AWS のユーザーに対し、ポリシー作成の許可を与えていません。例えば、ジョンが自らの所有するリソースへのアクセス権限を定義するポリシーを書いて提出した場合、AWS ではジョンの本人確認が行われます。

プリンシパル

*プリンシパル*とは、ポリシーのアクセス権限を適用される個人またはグループを指します。「A は、条件 D に該当する C を対象とするアクション B の実行を許可または禁止されている」というポリシーにおいては、A がプリンシパルに相当します。ポリシーにおいて「誰でも」プリンシパルに設定することができます (例えば、ワイルドカードと特定することによりすべての人々が設定可能であることなど)。このような決定は、例えば、Amazon S3 リソースのサブセットへのアクセスを全ユーザー (認証されているか匿名かは問わない) に付与する場合に行うことができます。

アクション

*アクション*とは、プリンシパルに対し、実行が許可されているアクティビティです。「A は、条件 D に該当する C を対象とするアクション B の実行を許可または禁止されている」というポリシーにおいては、文字通り B がアクションに相当します。通常、アクションとは、リクエストに埋め込まれて AWS に渡されるオペレーションのことです。 1 つのポリシーに 1 つまたは複数のアクションを指定することができます。

ポリシーに指定可能な Amazon S3 アクションの名前については、「Amazon S3 Actions (p. 338)」を参照してください。

リソース

*リソース*とは、プリンシパルがアクセスを要求するバケットまたはオブジェクトのことです。「A は、条件 D に該当する C を対象とするアクション B の実行を許可または禁止されている」というポリシーにおいては、C がリソースに相当します。1 つのポリシーに 1 つまたは複数のリソースを特定することができます。ポリシーにリソースを指定する方法の詳細については、「Specifying Amazon S3 Resources in Bucket Policies (p. 338)」を参照してください。

条件とキー

*条件*とは、アクセス権限についての制限や詳細のことです。「A は、条件 D に該当する C を対象とするアクション B の実行を許可または禁止されている」というポリシーにおいては、文字通り D が条件に相当します。ポリシーの中でも、記述が最も詳細かつ複雑になるのが、この条件部分です。よく使用される条件の設定項目は以下のとおりです。

- 日時 (特定の日付以前に到着したリクエストのみ処理するなど)
- IP アドレス (特定の CIDR 範囲内の IP アドレスからのリクエストのみ処理するなど)

*キー*は、アクセス制限に使用される基本項目です。例えば、リクエストの日時がこれに相当します。

制限は、*条件*と*キー*の両方を使用して定義します。具体例を挙げて説明します。2010 年 5 月 30 日以前のアクセスを制限するには、`DateLessThan` 条件を使用します。キーは を使用し、値を `2010-05-30T00:00:00Z` に設定します。使用する条件やキーは AWS により定義されています。その他に、により定義されているサービス固有のキーもあります。条件の詳細については、「Condition (p. 498)」を参照してください。利用可能なキーの詳細については、「Bucket Keys in Amazon S3 Policies (p. 341)」と「Object Keys in Amazon S3 Policies (p. 344)」を参照してください。

リクエスタ

*リクエスタ*とは、AWS サービスにリクエストを送信する人物、または特定のリソースへのアクセスを要求する人物です。リクエスタが AWS に送信するリクエストの内容は、基本的には次のようなものです。「条件 D に該当する C に対してアクション B を実行することを許可してください」

評価

*評価*とは、AWS サービスが受信したリクエストを拒否または許可するかを、該当するポリシーに基づいて判断するプロセスのことです。評価論理の詳細については、評価論理 (p. 491)を参照してください。

実行

*エフェクト*とは、評価時にポリシーのステートメントによって返される結果のことです。この値はポリシーのステートメントに特定します。使用可能な値は *deny* と *allow* です。

例えば、南極大陸からのすべてのリクエストを*拒否*するステートメントを記述した場合、リクエストの送信元 IP アドレスが南極大陸に割り当てられているものであれば、エフェクトの値は deny となります。また前述の代案として、南極大陸から*ではない*すべてのリクエストを*許可する*というステートメントも考えられます。この場合、リクエストの送信元が南極大陸でなければ、エフェクトの値は allow となります。2 つのステートメントは同じことを行うように見えますが、access policy language の論理上では異なるものです。詳細については、「評価論理 (p. 491)」を参照してください。

エフェクトに特定できる値は allow と deny の 2 つだけですが、ポリシーの評価結果には、*デフォルトで拒否*、*許可*および*明示的な拒否*の 3 種類があります。詳細については、以下のコンセプトおよび評価論理 (p. 491)を参照してください。

デフォルトで拒否

*デフォルトで拒否*とは、ポリシーに許可または明示的な拒否が指定されていない場合に、デフォルトで適用される拒否のことです。

許可

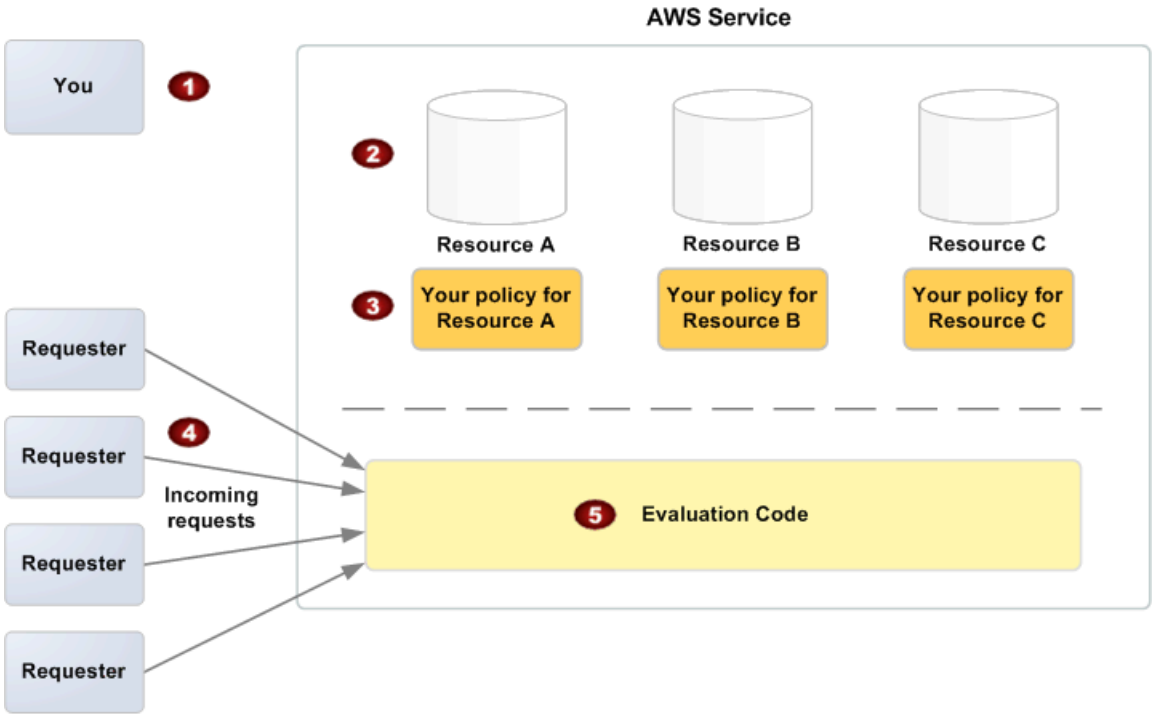
*許可*とは、ステートメントに effect=allow が指定されていて、許可条件がすべて満たされている場合に返される結果です。例えば、2010 年 4 月 30 日午後 1 時までに受信されたリクエストが許可されます。許可は、すべてのデフォルトで拒否に優先して適用されますが、明示的な拒否が 1 つでもあれば適用されません。

明示的な拒否

*明示的な拒否*とは、ステートメントに effect=deny が指定されていて、拒否条件がすべて満たされている場合に返される結果です。例えば、送信元が南極大陸であるすべてのリクエストが拒否されます。その他のポリシーによって許可されている場合においても、南極から来たリクエストに対しては常に拒否します。

アーキテクチャの概要

以下の図と表に、リソースのアクセスコントロールに関与する主要コンポーネントとそのインタラクティブな関わり合いを表します。



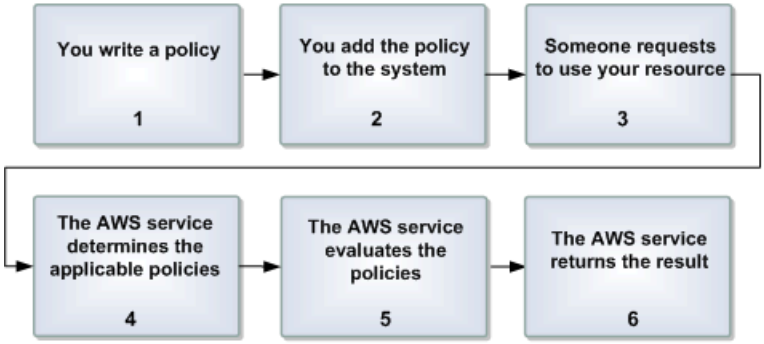
1	リソース所有者。
2	AWS サービスに含まれるリソース (Amazon S3 のバケットまたはオブジェクトなど) 。

3	ポリシー。 Amazon S3 では、バケットごとに 1 つのポリシーのみあります。Amazon S3 には、バケットポリシーのアップロードおよび管理を可能にする API が搭載されています。ポリシーの内容の詳細については、「ポリシーの書き方 (p. 494)」を参照してください。
4	リクエスタ、およびリクエスタから送信される AWS サービスへのリクエスト。
5	access policy language 評価コード。 AWS サービス内のコードセットです。受信したリクエストと該当するポリシーを照合して、リクエスタにリソースへのアクセスを許可するかどうかを判定します。このサービスによる判定の詳細については、評価論理 (p. 491)を参照してください。

各コンポーネントの代表的な関係については、Access Policy Language の使用 (p. 490)を参照してください。

Access Policy Language の使用

以下の図と表は、アクセスコントロールと access policy language の連携方法の通常プロセスを表しています。



access policy language でアクセスコントロールを使用するプロセス

1	リソース用ポリシーを記述します。 例えば、 Amazon S3 オブジェクト用の特定のアクセス許可についてのポリシーを作成します。詳細については、ポリシーの書き方 (p. 494)を参照してください。
2	AWS にポリシーをアップロードします。 AWS サービスでは、ポリシーのアップロードに使用できる API を提供しています。例えば、バケットにポリシーを設定するために Amazon S3 `PUT Bucket policy` アクションを使用します。
3	ある人物から、リソースの使用許可を求めるリクエストが送信されます。 例えば、オブジェクトをバケットにアップロードするために、Amazon S3 にリクエストを送信します。
4	どのポリシーがリクエストに適用可能であるか、AWS サービスによって決定されます。 例えば、Amazon S3 がすべての利用可能な Amazon S3 ポリシーを調べ、どのポリシーが適用可能であるかを決定します (リソースの内容、リクエスタがどのユーザーであるかなどを基準とする) 。
5	AWS サービスがポリシーを評価します。 例えば、Amazon S3 がポリシーを評価し、オブジェクトをバケットにアップロードする許可をリクエスタに付与するかどうかを決定します。決定論理の詳細については、評価論理 (p. 491)を参照してください。
6	AWS サービスにおいては、リクエスタを拒否するか、またはプロセスを継続するかのどちらかが行われます。 例えば、ポリシーの評価結果に基づいて、サービスによって「アクセス拒否」エラーがリクエスタに返されるか、リクエストのプロセスを継続するかのどちらかが行われます。

関連トピック

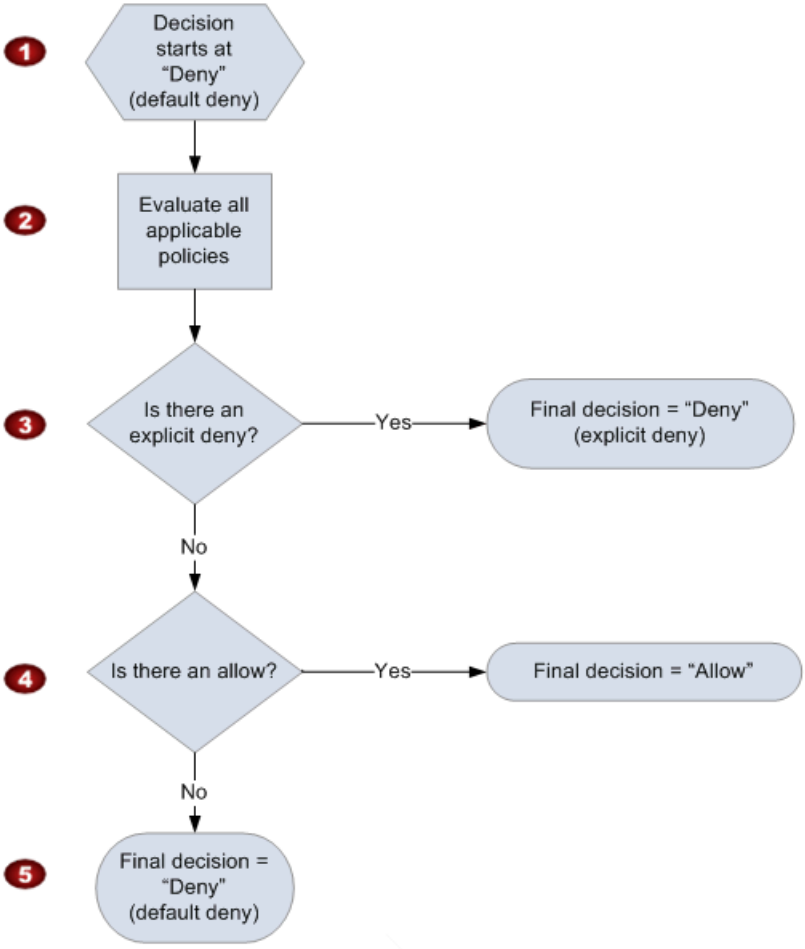
- アーキテクチャの概要 (p. 488)

評価論理

評価時の目標は、与えられたリクエストに対し、許可するか拒否するかを決定することです。評価論理は、以下の複数の基本ルールに従っています。

- デフォルトでは、リソースの使用許可を求めるリクエストについては、リクエスタが自分自身である場合を除いて、拒否を適用する
- 許可はすべてのデフォルトで拒否に優先する
- 明示的な拒否はすべての許可に優先する
- ポリシー評価の順序は重要ではない

以下のフローチャートと考察では、決定方法についての詳細説明を紹介します。



1	決定は [デフォルトで拒否] から始まります。
2	次にエンフォースメントコードによって、リクエストに適用される (リソース、プリンシパル、アクション、条件を基準に決定) が評価されます。 エンフォースメントコードによるポリシー評価の順序は重要ではありません。

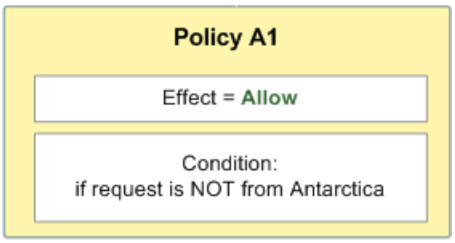
3	前述のすべてのポリシーにおいて、リクエストに適応する明示的な拒否のインストラクションがエンフォースメントコードによって検索されます。 仮に 1 つでも見つかった場合、エンフォースメントコードは拒否の決定を返し、プロセスを終了します (これは明示的な拒否となります。詳細については、明示的な拒否 (p. 488)を参照してください)。
4	明示的な拒否が見つからなかった場合、リクエストに適応する許可のインストラクションがエンフォースメントコードによって検索されます。 仮に 1 つでも見つかった場合、エンフォースメントコードは許可の決定を返し、プロセスは完了します (サービスはリクエストのプロセスを継続します)。
5	許可が見つからなかった場合、最終決定は「拒否」となります (明示的な拒否または許可が見つからない場合、[デフォルトで拒否] としてみなされるためです。詳細については、「デフォルトで拒否 (p. 488)」を参照してください)。

明示的な拒否とデフォルトで拒否の相互作用

ポリシーがリクエストに直接適用されない場合の結果は、[デフォルトで拒否] となります。例えば、あるアカウントが Amazon S3 の使用をリクエストしたが、そのユーザーの状態で適用される唯一のポリシーでは Amazon S3 バケットの内容をリストできるだけの場合、このポリシーの適用結果は [デフォルトで拒否] となります。

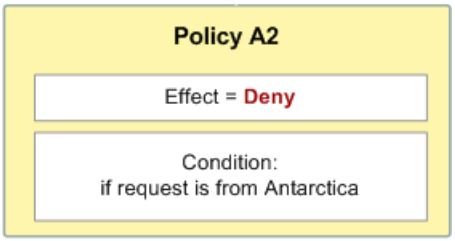
ステートメントの条件が満たされていない場合も、ポリシーの適用結果は [デフォルトで拒否] となります。ステートメントのすべての条件が満たされている場合、ポリシーの *エフェクト* 要素の値に基づいて、ポリシーの適用結果は許可または明示的な拒否のどちらかとなります。

例えば、南極大陸から来るリクエストを防ぐとします。その場合、南極大陸から来ていないリクエストにのみ許可を与えるポリシー (ポリシー A1 とする) を記述します。以下の図はポリシーについて解説しています。



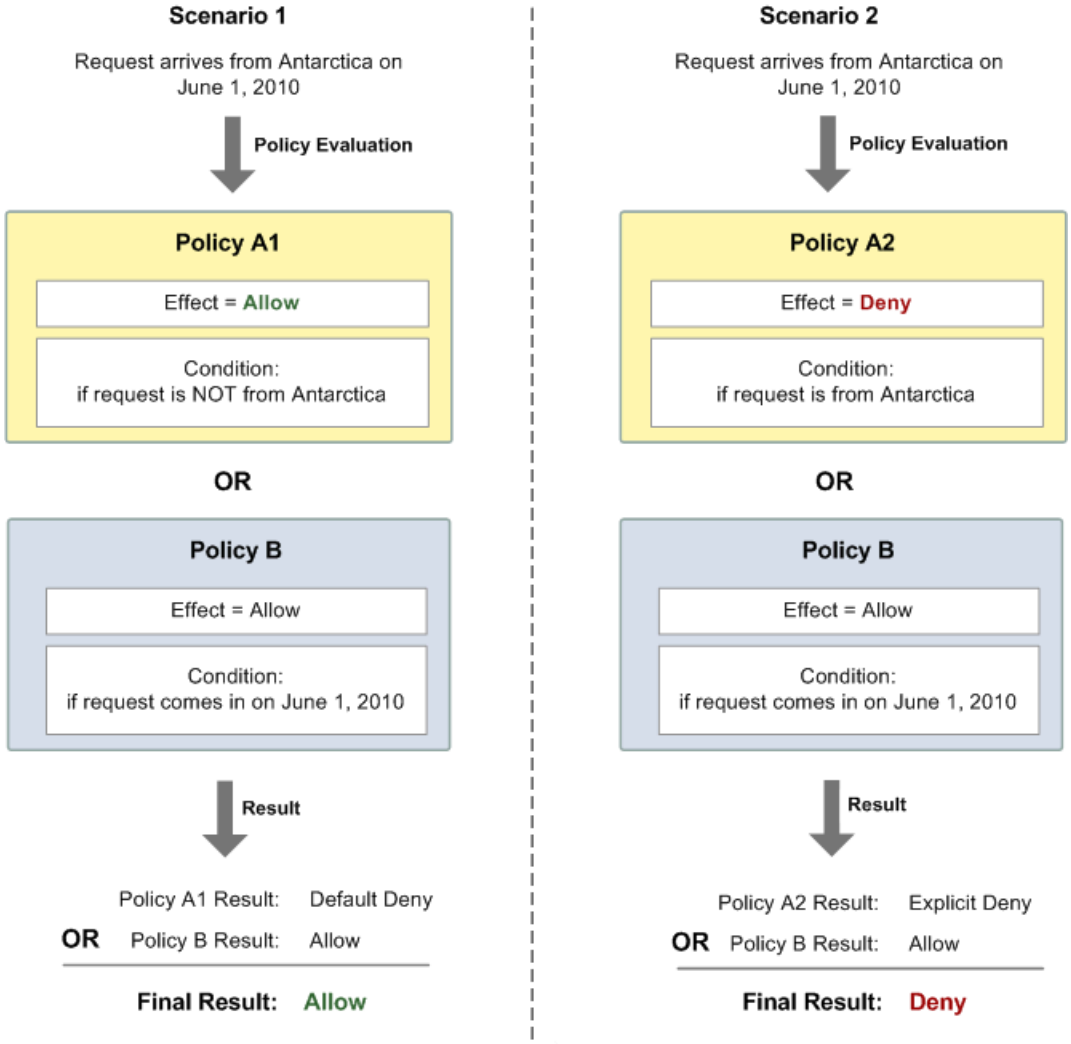
リクエストがアメリカから送られてきた場合、条件を満たしています (リクエストが南極大陸からのものでないため)。従って、そのリクエストは許可されます。リクエストが南極大陸から送られてきた場合、条件を満たしていないため、ポリシーの結果としてデフォルトで拒否となります。

以下の図のとおり、ポリシー (ポリシー A2 とする) を書き換えることにより、結果を明示的な拒否に変えることができます。南極大陸から送られてきた場合、ポリシーによってリクエストが明示的に拒否されます。



リクエストが南極大陸から送られてきた場合、条件を満たしているため、ポリシーの結果として明示的な拒否となります。

デフォルトで拒否は、許可によって優先されますが、明示的な拒否は優先されないため、デフォルトで拒否と明示的な拒否との差異は重要となります。例えば、リクエストが 2010 年 6 月 1 日に届いた場合、そのリクエストは許可されるという別のポリシーがあるとします。このポリシーが、南極大陸からのアクセスを制限しているポリシーと併用されている場合、全体の結果にどのような影響を及ぼすでしょうか?日付ベースのポリシー (ポリシー B とする) が前述のポリシー A1 および A2 と併用されている場合、全体の結果が比較されます。シナリオ 1 は、ポリシー A1 とポリシー B が併用されている場合、シナリオ 2 は、ポリシー A2 とポリシー B が併用されている場合です。以下の図と考察は、2010 年 6 月 1 日に南極大陸からリクエストが来た場合についての結果を示しています。



このセクションの最初に説明したとおり、シナリオ 1 においては、ポリシー A1 はデフォルトで拒否を返します。2010 年 6 月 1 日に到着したリクエストは、当然のことながら許可されるため、ポリシー B は許可を返します。ポリシー B による許可は、ポリシー A1 のデフォルトで拒否に優先するため、結果としてリクエストは許可されます。

このセクションの最初に説明したとおり、シナリオ 2 においては、ポリシー B2 は明示的な拒否を返します。再度、ポリシー B は許可を返します。ポリシー A2 による明示的な拒否は、ポリシー B の許可に優先するため、結果としてリクエストは拒否されます。

ポリシーの書き方

Topics

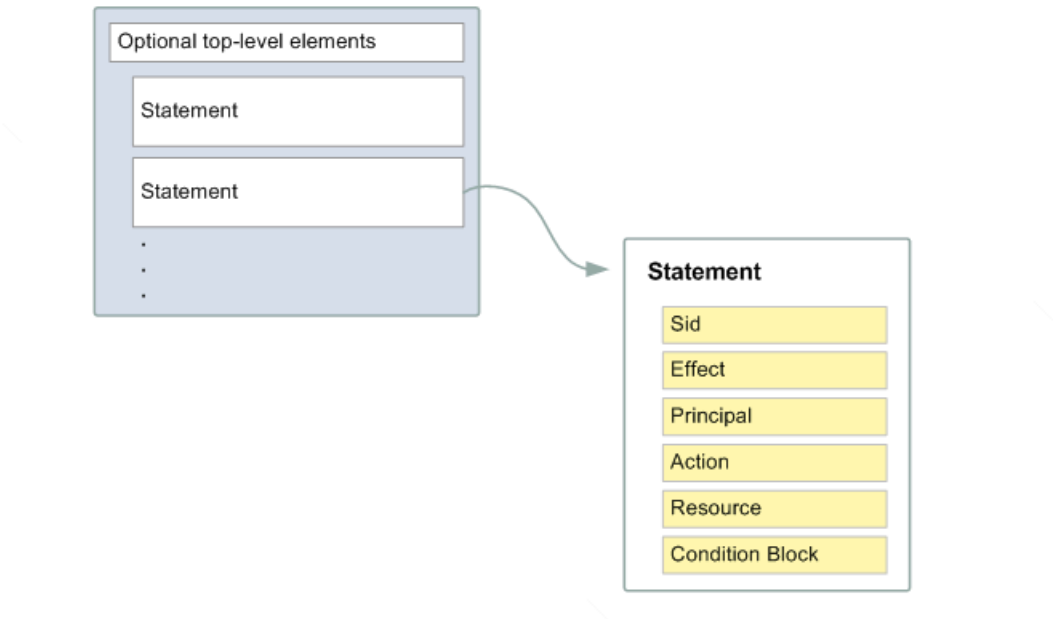
このセクションではポリシーの書き方、およびポリシーのエレメントについてのリファレンス情報が記載されてます。

基本的なポリシーの構造

各ポリシーは JSON ドキュメントです。以下の図が示しているように、ポリシーには以下の事項を含みます。

- 広範な情報を含む任意のポリシー (ドキュメントの最上部に記載)
- 1 つまたはそれ以上の個別のステートメント

各ステートメントには、1 回限りのアクセス権限についての核になる情報が含まれています。ポリシーが複数のステートメントを含んでいる場合、評価時のステートメント全体で論理的な OR を適用します。複数のポリシーがリクエストに適合する場合、評価時のポリシー全体で論理的な OR を適用します。



ステートメントの情報は、一連の*エレメント*の中に含まれています。これらのエレメントの詳細については、要素の説明 (p. 495)を参照してください。

要素の説明

Topics

- リソース (p. 498)
- Condition (p. 498)

このセクションでは、ポリシーとそのステートメントで使用できる要素について説明します。要素は、ポリシーで使用される一般的な順序で記載されています。Id、Version、および Statement がポリシーの最上位レベルの要素で、その他はステートメントレベルの要素です。JSON の例を示します。

どの要素もポリシードキュメントの解析では必須ではありません。要素の順序は重要ではありません（Resource 要素を Action 要素の前にもってくることなどが可能です）。ポリシーへの条件の指定は必須ではありません。

Version

Version は access policy language のバージョンのことです。これはオプションの要素で、現時点で使用可能な値は 2008-10-17 のみです。

```
"Version":"2008-10-17"
```

Id

Id は、ポリシーを示す識別子で、オプションです。値には、UUID を使用するか、または一意性を保証するため UUID を ID の一部として組み込むことをお勧めします。

Important

Amazon S3 では、access policy languageを実装するため、*Id* 要素が必須であるだけでなく一意であることが必要です。ポリシーの作成に関するサービス固有情報については、「Amazon S3 ポリシーの特別な情報 (p. 506)」を参照してください。

アクセス許可

アクセス許可とは、特定のリソースへのある種のアクセスを許可または拒否するというコンセプトです。アクセス許可は、基本的に「A は、条件 D に該当する C を対象とするアクション B の実行を許可または禁止されている」という形態をとります。例えば、Jane（A）は、John の Amazon SQS キュー（C）から送信されたメッセージを受信（B）することを許可されている。ただし、Jane が、2009 年 5 月 30 日の午前 12 時以前までに受信を要求したものに限る（D）、という具合です。Jane が John のキューを使用するために、Amazon SQS にリクエストを送った時点で、Jane にアクセス許可があるかどうか、またそのリクエストが、John によってアクセス許可に定められた条件を満たしているかどうかが、サービスによってチェックされます。

Statement

Statement は、ステートメントの主要要素です。これは、複数の要素を含むことができます（このガイドの以降のセクションを参照）。

Statement 要素は、各ステートメントからなる配列を含みます。各ステートメントは、波かっこ {} で囲まれた、JSON 形式の固有のブロックです。

```
"Statement":[{...},{...},{...}]
```

ポリシー

ポリシーとは、1 つ以上のステートメントを入れるコンテナの役目を果たすドキュメント（アクセスポリシー言語で作成）です。例えば、1 つのポリシーに「Jane は John のオブジェクトを読み込むことが

できる」というステートメントと、「Bob は John のオブジェクトを読み込むことができない」というステートメント、計 2 つのステートメントを含むことができます。これは、「Jane は John のオブジェクトを読み込むことができる」というステートメントを含むポリシーと、「Bob は John のオブジェクトを読み込むことができない」というステートメントを含むポリシー、計 2 つのポリシーを設定することと同じシナリオです。

Issuer

Issuer、つまり発行者とは、リソースに対するアクセス許可についてのポリシーを作成するユーザーです。発行者は当然のこととして、リソースの所有者になります。AWS の製品では、リソースを所有していない AWS のユーザーに対し、ポリシー作成の許可を与えていません。例えば、ジョンが自らの所有するリソースへのアクセス許可を定義するポリシーを作成して提出した場合、AWS ではジョンが本人であることが認証されます。

Sid

`Sid` (ステートメント ID) は、お客様がポリシードキュメントに指定するオプションの識別子です。基本的に、これはポリシードキュメント ID の副 ID に過ぎません。

Important

access policy languageの Amazon S3 実装では、この要素が必須であるだけでなく一意であることが必要です。ポリシーの作成に関するサービス固有情報については、「Amazon S3 ポリシーの特別な情報 (p. 506)」を参照してください。

```
"Sid" : "1"
```

Effect

`Effect` は、必須の要素であり、ステートメントを許可または明示的な拒否のいずれにするかを指定します (詳細については「明示的な拒否 (p. 488)」を参照) 。

`Effect` の有効値は、`Allow` と `Deny` です。

```
"Effect":"Allow"
```

Principal

`Principal` とは、ポリシーに基づいてアクセス許可を付与または拒否される 1 人以上のユーザーです。プリンシパルは、そのプリンシパルの AWS アカウント ID (例えば、1234-5678-9012、ハイフンの有無は任意) を使用して指定する必要があります。複数のプリンシパルを指定したり、指定可能なすべてのユーザーを示すワイルドカード (*) を指定することができます。アカウント ID を確認するには、http://aws.amazon.com で AWS アカウントにログインし、[Account Activity] をクリックします。

JSON の場合は、プリンシパルの AWS アカウント ID のプレフィックスとして `"AWS":` を使用します。

Action

`Action` は、許可または拒否される 1 つ以上の特定のタイプのアクセスです (読み込みや書き込みなど) 。この要素には複数の値を指定することができます。値はフリーフォームで指定できますが、AWS サービスで想定されている値と一致する必要があります (詳細については「Amazon S3 ポリシーの特別な情報 (p. 506)」を参照) 。その AWS サービスで他の開発者と共有できるすべてのアクションへのアクセスをプリンシパルに付与するには、ワイルドカード (*) を使用することができます。

NotAction

NotAction 要素は、アクションのリストに例外を設ける場合に使用できます。これは、例えば、ユーザーに GET Object のみの使用を許可する場合に使用できます。

以下の例は、GET Object *以外*のすべてのアクションを参照しています。ポリシー内でこれを "Effect":"Deny" と共に使用することで、ユーザーが他のアクションにアクセスすることを防ぐことができます。

```
"NotAction":"s3:GetObject"
```

リソース

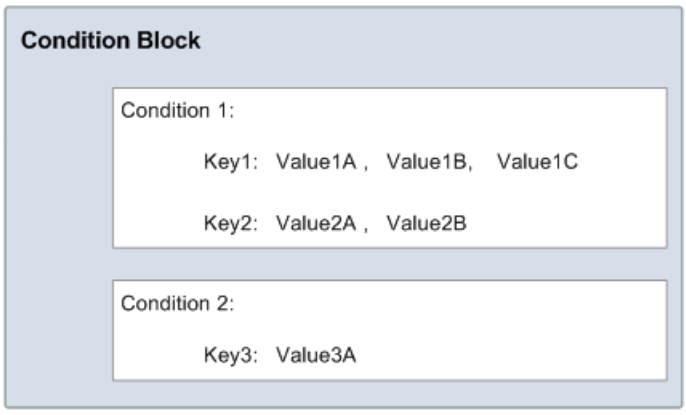
Resourceはポリシーの対象となる 1 つまたは複数のオブジェクトです。値には、複数文字一致のワイルドカード (*) または 1 文字一致のワイルドカード (?) を文字列のどこにでも含めることができます。値はフリーフォームで指定できますが、AWS サービスで想定されている形式と一致する必要があります。

Condition

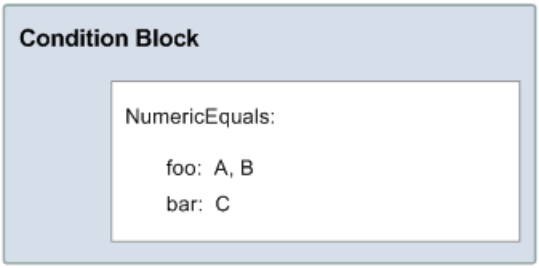
このセクションでは、Condition 要素と、この要素内で使用できる情報について説明します。

Condition ブロック

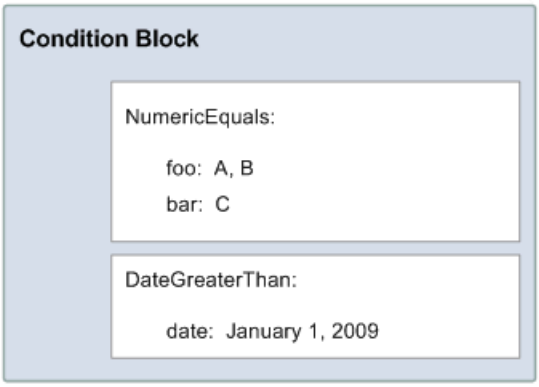
Condition 要素は、ポリシーのステートメントの中で最も複雑なパートです。この要素を *Condition* ブロックと呼ぶ理由は、その中に 1 つの Condition 要素しかないのに、複数の条件を含むことができ、また各条件に複数のキーと値のペアを含むことができるためです。この内容を以下に図で示します。特定のキーに単一の値を指定する場合を除き、すべてのキーに複数の値を設定できます。



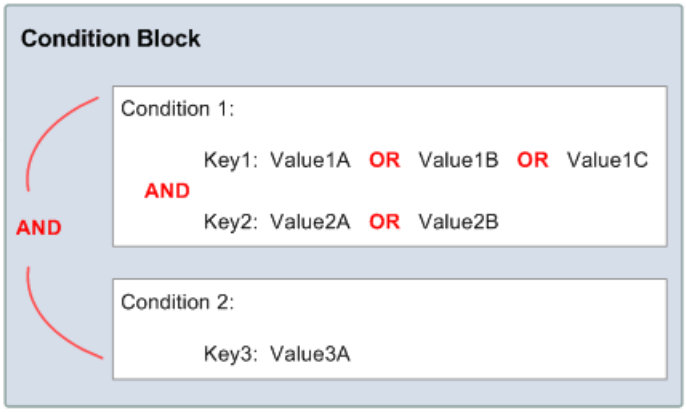
Condition ブロックを作成するときは、各条件の名前と最低 1 つのキーと値のペアを指定します。AWS では、使用可能な条件およびキー (以降のセクションに記載) を定義しています。NumericEquals は条件の例です。架空のリソースがあり、特定の数値 *foo* が A または B と等しく、かつもう 1 つの数値 *bar* が C と等しいときに限り、そのリソースをジョンに使用させるとします。このような場合、次の図のような Condition ブロックを作成します。



加えて、ジョンのアクセスを 2009 年 1 月 1 日以降に制限する場合を考えます。このような場合は、2009 年 1 月 1 日の日付を指定したもう 1 つの条件、`DateGreaterThan` を追加します。これにより、Condition ブロックは以下の図のようになります。



以下の図に示すように、Condition ブロック内の条件と条件内のキーには常に論理 AND を適用します。単一キーの値には常に論理 OR を適用します。許可または明示的な拒否の決定を返すためには、すべての条件に一致しなければいけません。条件が一致しない場合、結果はデフォルトで拒否となります。



前述のように、AWS では、使用可能な条件およびキーを定義しています (キーの 1 つの例は `aws:CurrentTime` で、日付と時刻に基づいてアクセスを制限するのに使用されます) 。その他に、AWS サービス自体により定義されているサービス固有のキーもあります。使用可能なキーのリストは、利用可能なキー (p. 500)を参照してください。

実際のキーを使用する具体的な例として、以下の 3 つの条件のもとでジョンにオブジェクトをアップロードさせる場合を考えます。

- 日時は 2010/8/16 の正午を過ぎたところです。
- 日時は 2010/8/16 の午後 3 時前です。
- リクエストは、192.168.176.0/24 の範囲または 192.168.143.0/24 の範囲に含まれる IP アドレスから来る。

お客様の Condition ブロックには 3 つの異なる条件があり、ジョンがお客様のバケットにアクセスするには、それら 3 つすべての条件を満たす必要があります。

お客様のポリシーの Condition ブロックは以下のようになります。

```
"Condition" :  {
      "DateGreaterThan" : {
         "aws:CurrentTime" : "2009-04-16T12:00:00Z"
       }
      "DateLessThan": {
         "aws:CurrentTime" : "2009-04-16T15:00:00Z"
       }
       "IpAddress" : {
          "aws:SourceIp" : ["192.168.176.0/24","192.168.143.0/24"]
      }
}
```

利用可能なキー

AWS は、アクセスコントロールに access policy language を使用するすべての AWS 製品でサポートされる、一般的な一連のキーを提供しています。提供されているキーは以下のとおりです。

- `aws:CurrentTime`– 日時条件を表示 (「日付の条件 (p. 502)」を参照してください) 。
- `aws:EpochTime`– エポック以後の秒数。
- `aws:MultiFactorAuthAge`– リクエストを行った、 ブリックサブネット (MFA) で認証されたセキュリティ認証情報が、MFA を使用して今から何秒前に発行されたかを示す数値を提供するキー。他のキーと異なり、MFA が正常に使用されなかった場合には、このキーは作成されません (「条件キーの有無 (p. 503)」と「数値の条件 (p. 501)」を参照) 。 詳細については、「Using Multi-Factor Authentication (MFA) Devices with AWS」を参照してください。
- `aws:SecureTransport`– リクエストが SSL を使用して送信されたかどうかを示すブール値 (「ブール条件 (p. 503)」を参照してください) 。
- `aws:SourceIp`– IP アドレス条件で使用するための、リクエスタの IP アドレス (「IP アドレス (p. 503)」を参照してください) 。
- `aws:UserAgent`– 文字列条件で使用するための、リクエスタのクライアントアプリケーションに関する情報 (「文字列の条件 (p. 501)」を参照してください) 。
- `aws:Referer`– HTTP *`referer`* フィールドと同じ。

キー名では大文字と小文字が区別されません。例えば、`aws:CurrentTime` と `AWS:currenttime` は同じです。

Note

`aws:SourceIp` を使用する場合、リクエストが Amazon EC2 インスタンスからのものであるときは、アクセスを許可するかどうかを判断するために、インスタンスのパブリック IP アドレスが評価されます。

また、access policy language を使用する各 AWS サービスが、サービス固有のキーを提供する場合もあります。 使用可能なサービス固有のキーのリストについては、「Amazon S3 ポリシーの特別な情報 (p. 506)」を参照してください。

条件タイプ

指定できる条件タイプは以下のとおりです。

- スプリング
- 数値
- 日付および時間
- ブール
- IP アドレス
- Amazon リソースネーム (ARN)
- 条件キーの有無

文字列の条件

文字列の条件では、文字列の一致ルールを使用して制約を加えることができます。実際に使用するデータ型は文字列です。

条件	説明
`StringEquals`	完全一致 短縮版: `streq`
`StringNotEquals`	符号反転の完全一致 短縮版: `strneq`
`StringEqualsIgnoreCase`	大文字と小文字の区別がない完全一致 短縮版: `streqi`
`StringNotEqualsIgnoreCase`	大文字と小文字の区別がない符号反転の完全一致 短縮版: `strneqi`
`StringLike`	大文字と小文字の区別がないあいまい一致。値には、複数文字一致のワイルドカード (*) または 1 文字一致のワイルドカード (?) を文字列のどこにでも含むことができます。 短縮版 : `strl`
`StringNotLike`	大文字と小文字の区別がない符号反転のあいまい一致。値には、複数文字一致のワイルドカード (*) または 1 文字一致のワイルドカード (?) を文字列のどこにでも含むことができます。 短縮版 : `strnl`

数値の条件

数値の条件では、数値の一致ルールを使用して制約を加えることができます。整数と 10 進数の両方とも使用できます。分数や無理数の構文はサポートされていません。

条件	説明
NumericEquals	完全一致 短縮版: numeq
NumericNotEquals	符号反転の完全一致 短縮版: numneq
NumericLessThan	「未満」の一致 短縮版 : numlt
NumericLessThanEquals	「以下」の一致 短縮版 : numlteq
NumericGreaterThan	「より大きい」の一致 短縮版 : numgt
NumericGreaterThanEquals	「以上」の一致 短縮版 : numgteq

日付の条件

日付の条件では、日付と時刻の一致ルールを使用して制約を加えることができます。 すべての日付/時刻の値を指定する際は、ISO 8601 日付形式の W3C 実装の 1 つ (詳細については、http://www.w3.org/TR/NOTE-datetime を参照) を使用します。 これらの条件を aws:CurrentTime キーと合わせて使用することで、 リクエストの時刻に基づいてアクセスを制限できます。

Note

ワイルドカードは、日付条件では認められていません。

条件	説明
DateEquals	完全一致 短縮版: dateeq
DateNotEquals	符号反転の完全一致 短縮版: dateneq
DateLessThan	キーの実行が停止する時間 短縮版 : datelt
DateLessThanEquals	キーの有効期間の終了時刻 短縮版: datelteq
DateGreaterThan	キーの実行が開始する時間 短縮版 : dategt
DateGreaterThanEquals	キーの有効期間の開始時刻 短縮版: dategteq

ブール条件

条件	説明
Bool	ブールの完全一致

IP アドレス

IP アドレスの条件では、IP アドレスの一致ルールに基づいて制約を加えることができます。これらを aws:SourceIp キーと合わせて使用します。値は、標準的な CIDR 形式でなければいけません (例 : 10.52.176.0/24) 。詳細については、「Request for Comments: 4632」を参照してください。

条件	説明
IpAddress	IP アドレスまたは IP アドレス範囲に基づくホワイトリスト
NotIpAddress	IP アドレスまたは IP アドレス範囲に基づくブラックリスト

条件キーの有無

条件キーが認証時に存在するかどうかを確認するために Null 条件を使用します。ポリシーステートメントでは、true (キーは存在しない) または false (キーは存在し、その値は not null である) のどちらかを使用します。この条件を使用して、ユーザーが MFA で認証されているかどうかを判断することができます。例えば、以下の条件では、ユーザーが Amazon EC2 API を使用するために MFA 認証が存在していなければならない (not null である必要がある) ことを指定しています。

```
{
"Statement":[{
       "Action":["ec2:*"],
       "Effect":"Allow",
       "Resource":["*"],
        "Condition":{"Null":{"aws:MultiFactorAuthAge":"false"}
       }
     }
  ]
}
```

Amazon リソースネーム (ARN)

Amazon リソースネーム (ARN)条件では、ARN 一致ルールに基づいて制約を加えることができます。実際に使用するデータ型は文字列です。

条件	説明
ArnEquals	ARN の完全一致
ArnNotEquals	ARN の符号反転の完全一致
ArnLike	ARN の大文字と小文字の区別がないあいまい一致。ARN のコロンで分割された 6 個の各構成要素は、個別に確認され、それぞれ複数文字一致のワイルドカード (*) または 1 文字一致のワイルドカード (?) を含むことができます。

条件	説明
ArnNotLike	ARN の大文字と小文字の区別がない、符号反転のあいまい一致。値には、複数文字一致のワイルドカード (*) または 1 文字一致のワイルドカード (?) を文字列のどこにでも含めることができます。

```
{
   "Statement":[{
      "Effect": "Allow",
      "Principal": {
         "AWS": "210987654321"
      },
      "Action": "sqs:SendMessage",
      "Resource": "arn:aws:sqs:us-east-1:01234567891:your_queue_xyz",
      "Condition" : {
         "ArnEquals" : {
          "aws:SourceArn":"arn:aws:sns:us-east-1:123456789012:your_special_top
ic_1"}
            }
      }
   ]
}
```

サポートされているデータの種類

このセクションでは、access policy language でサポートされているデータの種類について説明します。各ポリシー工レメントのすべての種類が当該言語でサポートされているわけではありません (各工レメントでサポートされているデータの種類については、要素の説明 (p. 495)を参照してください) 。

access policy language は以下のデータの種類をサポートしています。

- 文字列
- 数字 (整数および浮動小数点数)
- ブール
- Null
- リスト
- マップ
- 構造体 (入れ子形式のマップ)

以下の表は、各データの種類をシリアル化してまとめたものです。すべてのポリシーは UTF-8 形式でなければいけないことに注意してください。JSON データに関する情報については、RFC 4627 を参照してください。

種類	JSON
文字列	文字列
整数	数字
浮動小数点	数字
ブール	true または false

種類	JSON
Null	null
日付	ISO 8601 の W3C プロファイルに準拠する文字列
IP アドレス	RFC 4632 に準拠する文字列
リスト	配列
オブジェクト	オブジェクト

Amazon S3 ポリシーの特別な情報

ここでは、Amazon S3 ポリシーの制約について説明します。

- ポリシーの最大サイズは 20 KB です。
- *Resource* の値には、バケット名またはバケット名とそれに続くパス (bucket/*) を、プレフィックスとして付ける必要があります。後に /* を付けずにバケット名のみを指定すると、ポリシーはそのバケットに適用されます。
- 各ポリシーには固有のポリシー ID (*Id*) が必要です。
- ポリシーを構成する各ステートメントには固有のステートメント ID (*sid*) が必要です。
- 各ポリシーの適用範囲は、単一のバケットと、そのバケット内のリソースに限定する必要があります (ポリシーを記述する際、他のバケットや他のバケット内のリソースを参照するステートメントを含めないでください) 。

付録 B: SOAP API の使用

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

このセクションでは、Amazon S3 SOAP API に固有の情報について説明します。

Note

SOAP リクエストは、認証済みの場合も匿名の場合も、SSL を使用して Amazon S3 に送信する必要があります。HTTP を介して SOAP リクエストを送信すると、Amazon S3 によりエラーが返されます。

一般的な SOAP API 要素

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

HTTP 経由で SOAP 1.1 を使用して、Amazon S3 と通信できます。機械による読み取りが可能な方法で Amazon S3 API を記述する Amazon S3 WSDL は、http://doc.s3.amazonaws.com/2006-03-01/AmazonS3.wsdl で入手可能です。Amazon S3 スキーマは、http://doc.s3.amazonaws.com/2006-03-01/AmazonS3.xsd で入手可能です。

ユーザーのほとんどが、言語や開発環境に合わせてカスタマイズされた SOAP ツールキットを使用して、Amazon S3 と通信します。ツールキットごとに Amazon S3 API は異なる方法で公開されます。使用方法については、特定のツールキットのドキュメントを参照してください。このセクションでは、「オンライン」での XML リクエストやレスポンスを示すことにより、ツールキットから独立して Amazon S3 SOAP オペレーションについて説明します。

一般的な要素

いずれの SOAP リクエストにも、認証に関連する次の要素を含めることができます。

- *AWSAccessKeyId:* リクエスタの AWS アクセスキー ID
- *Timestamp:* システムの現在の時刻
- *Signature:* リクエストの署名

エンドポイントについては、「Request Endpoints (p. 13)」を参照してください。

SOAP リクエストの認証方法

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

リクエスト元となるプリンシパルのアイデンティティを確立するには、すべての非匿名リクエストに認証情報が含まれている必要があります。SOAP では、認証情報は SOAP リクエストの以下の要素に置かれています。

- *AWSAccessKeyId:* AWS アクセスキー ID

Note

認証済みの SOAP リクエストを行う場合、一時的なセキュリティ認証情報はサポートされません。認証情報の種類の詳細については、「リクエストの実行 (p. 11)」を参照してください。

- *Timestamp:* これは、`2009-01-01T12:00:00.000Z` など、協定世界時 (グリニッジ標準時) タイムゾーンの dateTime (http://www.w3.org/TR/xmlschema-2/#dateTime を参照) である必要があります。このタイムスタンプが Amazon S3 サーバーの時刻と 16 分以上の誤差がある場合、認証は失敗します。
- *Signature:* AWS シークレットアクセスキーをキーとして使用し、「AmazonS3」+ OPERATION + タイムスタンプの連結文字で生成される RFC 2104 MAC-SHA1 ダイジェスト (「http://www.ietf.org/rfc/rfc2104.txt」を参照) 。例えば、以下の CreateBucket サンプルリクエストでは、署名要素に「AmazonS3CreateBucket2009-01-01T12:00:00.000Z」という値の HMAC-SHA1 ダイジェストが含まれています。

例えば、以下の CreateBucket サンプルリクエストでは、署名要素に「AmazonS3CreateBucket2009-01-01T12:00:00.000Z」という値の HMAC-SHA1 ダイジェストが含まれています。

Example

```
<CreateBucket xmlns="http://doc.s3.amazonaws.com/2006-03-01">
  <Bucket>quotes</Bucket>
  <Acl>private</Acl>
  <AWSAccessKeyId>AKIAIOSFODNN7EXAMPLE</AWSAccessKeyId>
  <Timestamp>2009-01-01T12:00:00.000Z</Timestamp>
  <Signature>Iuyz3d3P0aTou39dzbqaEXAMPLE=</Signature>
</CreateBucket>
```

Note

SOAP リクエストは、認証済みの場合も匿名の場合も、SSL を使用して Amazon S3 に送信する必要があります。HTTP を介して SOAP リクエストを送信すると、Amazon S3 によりエラーが返されます。

Important

特別な時間精度をどの程度割愛するかについてはさまざまな解釈が存在するために、.NET ユーザーは過度に詳細なタイムスタンプを Amazon S3 へ送信しないよう気をつける必要があります。ミリ秒での精度で `DateTime` オブジェクトを手動で構築することによって、これを達成することができます。

SOAP のアクセスポリシーの設定

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

アクセスコントロールは、バケットまたはオブジェクトの記述時に、「AccessControlList」要素と一緒に `CreateBucket`、`PutObjectInline`、または `PutObject` へのリクエストを含めることによって設定できます。AccessControlList 要素についてはアクセスコントロール (p. 294) で説明しています。これらのオペレーションでアクセスコントロールリストが指定されていない場合、リクエスタに FULL_CONTROL アクセスを付与するデフォルトのアクセスポリシーでリソースを生成します (これは該当のリクエストが、既に存在するオブジェクトへの PutObjectInline または PutObject リクエストである場合にも当てはまります)。

オブジェクトにデータを書き込み、オブジェクトを匿名プリンシパルによって読み取り可能にして、特定のユーザーにバケットへの FULL_CONTROL の権限を付与するリクエストを次に示します (ほとんどの開発者は、自分で自分のバケットに FULL_CONTROL アクセスを付与したいと考えるでしょう)。

Example

オブジェクトにデータを書き込み、オブジェクトを匿名プリンシパルによって読み取り可能にするリクエストを次に示します。

Sample Request

```
<PutObjectInline xmlns="http://doc.s3.amazonaws.com/2006-03-01">
  <Bucket>quotes</Bucket>
  <Key>Nelson</Key>
  <Metadata>
    <Name>Content-Type</Name>
    <Value>text/plain</Value>
  </Metadata>
  <Data>aGEtaGE=</Data>
  <ContentLength>5</ContentLength>
  <AccessControlList>
    <Grant>
      <Grantee xsi:type="CanonicalUser">
       <ID>75cc57f09aa0c8caeab4f8c24e99d10f8e7faeebf76c078efc7c6caea54ba06a</ID>

        <DisplayName>chriscustomer</DisplayName>
      </Grantee>
      <Permission>FULL_CONTROL</Permission>
    </Grant>
    <Grant>
      <Grantee xsi:type="Group">
        <URI>http://acs.amazonaws.com/groups/global/AllUsers<URI>
      </Grantee>
      <Permission>READ</Permission>
    </Grant>
  </AccessControlList>
  <AWSAccessKeyId>AKIAIOSFODNN7EXAMPLE</AWSAccessKeyId>
  <Timestamp>2009-03-01T12:00:00.183Z</Timestamp>
  <Signature>Iuyz3d3P0aTou39dzbqaEXAMPLE=</Signature>
</PutObjectInline>
```

Sample Response

```
<PutObjectInlineResponse xmlns="http://s3.amazonaws.com/doc/2006-03-01">
  <PutObjectInlineResponse>
    <ETag>&quot828ef3fdfa96f00ad9f27c383fc9ac7f&quot</ETag>
    <LastModified>2009-01-01T12:00:00.000Z</LastModified>
  </PutObjectInlineResponse>
</PutObjectInlineResponse>
```

既存のバケットまたはオブジェクトのアクセスコントロールポリシーは、GetBucketAccessControlPolicy、GetObjectAccessControlPolicy、SetBucketAccessControlPolicy、および SetObjectAccessControlPolicy メソッドを使用して、読み取ったり、設定したりできます。詳細については、これらのメソッドの詳しい説明を参照してください。

Amazon S3 リソース

次の表は、このサービスを利用する際に役立つ関連リソースのリストです。

リソース	説明
『Amazon Simple Storage Service 入門ガイド』	この入門ガイドは、簡単なユースケースを使用した、サービスのクイックチュートリアルです。
『Amazon Simple Storage Service API リファレンス』	API リファレンスでは、Amazon S3 のオペレーションについて詳しく説明しています。
Amazon S3 テクニカル FAQ	この製品について開発者から最もよく寄せられる質問です。
Amazon S3 リリースノート	リリースノートには、最新リリースの概要が記載されています。新機能、解決された問題、既知の問題が具体的に記載されています。
AWS 開発者リソースセンター	資料、コード例、リリースノートをはじめとする、AWS ベースの革新的なアプリケーション開発に役立つさまざまな情報が収められた、中心的起点となるリソースセンターです。
AWS マネジメントコンソール	コンソールを使用して、Amazon S3 のほとんどの機能をプログラミング不要で実行できます。
フォーラム	開発者が AWS に関連する技術的な質問についてディスカッションできる、コミュニティベースのフォーラムです。
AWS サポートセンター	当社の開発者フォーラム、技術上のよくある質問、サービスステータスページ、AWS Premium Support へのアクセスを含む、AWS テクニカルサポートのホームページです。
AWS 料金見積もりツール	AWS 料金見積もりツールを使用すると、AWS サービス使用の月額料金を見積もることができます。
AWS プレミアムサポート	AWS Premium Support に関する情報のメインウェブページです。1対1のサポートチャネルで、迅速に対応します。AWS インフラストラクチャサービスでのアプリケーションの構築および実行を支援します。
Amazon S3 製品情報	Amazon S3 に関する情報のメインウェブページです。

リソース	説明
お問い合わせ	AWS の支払い、アカウント設定その他に関する問い合わせ先です。
利用規約	Amazon.com およびその関連会社における著作権、商標の利用に関する規約について詳しく説明しています。

文書の履歴

このドキュメント履歴は、『*Amazon Simple Storage Service 開発者ガイド*』の前回リリース以降の重要な変更点をまとめたものです。

この履歴に関連する日付:

- 現行製品バージョン: 2006-03-01
- 前回のドキュメントの更新: 2013 年 9 月 20 日

次の表は、『*Amazon Simple Storage Service 開発者ガイド*』の前回リリース以降の重要な変更点をまとめたものです。

変更点	説明	日付
HTTP 経由での SOAP のサポートが廃止されました。	SOAP のサポートは HTTP 経由では廃止されましたが、HTTPS 経由では引き続き利用可能です。SOAP では、Amazon S3 の新機能がサポートされなくなります。REST API か AWS SDK を使用することをお勧めします。	このリリース内に記載されています。
IAM ポリシー変数のサポート	IAM アクセスポリシー言語では、変数がサポートされるようになりました。ポリシーが評価されると、認証ユーザーのセッションの文脈ベースの情報から得られた値によって、ポリシー変数が置換されます。ポリシー変数を使用すれば、ポリシーのすべての構成要素を明示的にリストしなくても、汎用的なポリシーを定義できます。ポリシー変数の詳細については、*AWS Identity and Access Management Using IAM* ガイドの Policy Variables を参照してください。 Amazon S3 のポリシー変数の例については、「Amazon S3 のポリシーの例 (p. 307)」を参照してください。	2013 年 4 月 03 日
コンソールでのリクエスタ支払いのサポート	Amazon S3 コンソールで、バケットをリクエスタ支払い用に設定できるようになりました。詳細については、「&S3 コンソールを使用したリクエスタ支払いの設定 (p. 102)」を参照してください。	2012 年 12 月 31 日

変更点	説明	日付
ウェブサイトホスティングでのルートドメインのサポート	Amazon S3 が、ルートドメインでの静的ウェブサイトのホスティングをサポートするようになりました。ウェブサイトにアクセスする訪問者がブラウザにウェブアドレスを入力するときに、「www」を省略できます (例:「example.com」) 。既に Amazon S3 でホスティングされている静的ウェブサイトの多くが、「www」サブドメイン (例:「www.example.com」) からアクセスできるようになっています。これまでは、ルートドメインアクセスをサポートするには、ルートドメインへのリクエストをブラウザから Amazon S3 のウェブサイトに中継するためのウェブサーバーを独自に運用することが必要でした。リクエストを中継するためのウェブサーバー運用では、コストや運用の負担の増加が伴うだけではなく、障害点が出てくるおそれがあります。今後は、「www」とルートドメインの両方のアドレスで、Amazon S3 の可用性と耐久性の高さを活用できるようになりました。詳細については、「Amazon S3 での静的ウェブサイトのホスティング (p. 411)」を参照してください。	2012 年 12 月 27 日
コンソールの修正	Amazon S3 コンソールが更新されました。ドキュメントのトピックのうち、コンソールに言及している部分も、これに従って修正されています。	2012 年 12 月 14 日
Amazon Glacier へのデータのアーカイブのサポート	Amazon S3 で、Amazon Glacier の低コストなストレージサービスを使用してデータをアーカイブできるようにする新しいストレージオプションがサポートされるようになりました。オブジェクトをアーカイブするには、オブジェクトを特定するアーカイブルールと、そのオブジェクトを Amazon Glacier にアーカイブするスケジュールを定義します。Amazon S3 コンソールを使用して、あるいは Amazon S3 API または AWS SDK を使用したプログラミングによって、バケット上にルールを簡単に設定できます。 詳細については、「オブジェクトのライフサイクル管理 (p. 114)」を参照してください。	2012 年 11 月 13 日
ウェブサイトページのリダイレクトのサポート	バケットがウェブサイトとして設定されている場合、オブジェクトに対するリクエストを同じバケット内の別のオブジェクトまたは外部 URL に Amazon S3 がリダクレクトできるようになりました。詳細については、「ウェブページリダイレクトの設定 (p. 423)」を参照してください。 ウェブサイトのホスティングについては、「Amazon S3 での静的ウェブサイトのホスティング (p. 411)」を参照してください。	2012 年 10 月 4 日
Cross-Origin Resource Sharing (CORS) のサポート	Amazon S3 で Cross Origin Resource Sharing (CORS) がサポートされるようになりました。CORS は、あるドメインにロードされたクライアントウェブアプリケーションが、異なるドメイン内のリソースと通信またはアクセスする方法を定義します。Amazon S3 で CORS がサポートされたことにより、 S3 を基盤とする機能豊富なクライアント側ウェブアプリケーションを構築し、Amazon S3 リソースに対するクロスドメインアクセスを選択的に許可することができます。詳細については、「Cross-Origin Resource Sharing の有効化 (p. 133)」を参照してください。	2012 年 8 月 31 日

変更点	説明	日付
コスト割り当てタグのサポート	Amazon S3 でコスト割り当てタグがサポートされました。S3 バケットにラベルを付けることができるため、プロジェクトやその他の条件に照らして、それらのコストを簡単に追跡できます。バケットのタグの使用に関する詳細については、「コスト割り当てタグ (p. 105)」を参照してください。	2012 年 8 月 21 日
バケットポリシーでの MFA で保護された API アクセスのサポート	Amazon S3 で、MFA で保護された API アクセスがサポートされるようになりました。これにより、 AWS Multi-Factor Authentication を適用して、Amazon S3 リソースへのアクセス時のセキュリティを強化できます。このセキュリティ機能では、有効な MFA コードを入力して MFA デバイスを物理的に所有していることを証明することがユーザーに要求されます。詳細については、「AWS Multi-Factor Authentication」を参照してください。Amazon S3 リソースへのすべてのアクセスリクエストに対して、MFA 認証を要求できるようになりました。 MFA 認証を適用するために、バケットポリシーで `aws:MultiFactorAuthAge` キーがサポートされるようになりました。バケットポリシーの例については、「MFA 認証を要求するバケットポリシーの追加 (p. 335)」を参照してください。	2012 年 7 月 10 日
オブジェクトの有効期限のサポート	オブジェクトの有効期限を使用して、設定した期間を経過後にデータを自動削除するよう設定できます。オブジェクトの有効期限を設定するには、バケットにライフサイクル設定を追加します。詳細については、「オブジェクトの有効期限 (p. 123)」を参照してください。	2011 年 12 月 27 日
新しいリージョンのサポート	Amazon S3 で、南米 (サンパウロ) リージョンがサポートされるようになりました。詳細については、「Buckets and Regions (p. 90)」を参照してください。	2011 年 12 月 14 日
Multi-Object Delete	Amazon S3 で、単一のリクエストで複数のオブジェクトを削除できる Multi-Object Delete API がサポートされるようになりました。この機能を使用して、複数の個別の DELETE リクエストを使用するよりもすばやく多数のオブジェクトを Amazon S3 から削除できます。詳細については、「オブジェクトの削除 (p. 253)」を参照してください。	2011 年 12 月 7 日
新しくサポートされるリージョン	Amazon S3 で、米国西部 (オレゴン) リージョンがサポートされるようになりました。詳細については、「Buckets and Regions (p. 90)」を参照してください。	2011年11月8日
ドキュメントの更新	ドキュメントのバグの修正。	2011年11月8日
ドキュメントの更新	ドキュメントのバグの修正に加えて、このリリースには、次の機能強化が含まれています。 • AWS SDK for PHP (「AWS SDK for PHP を使用したサーバー側の暗号化の指定 (p. 369)」を参照) と AWS SDK for Ruby (「AWS SDK for Ruby を使用したサーバー側の暗号化の指定 (p. 372)」を参照) を使用したサーバー側の新しい暗号化セクション。 • Ruby サンプルの作成およびテストに関する新セクション (「AWS SDK for Ruby の使用 (p. 482)」を参照)。	2011 年 10 月 17 日

変更点	説明	日付
サーバー側暗号化のサポート	Amazon S3 で、サーバー側暗号化がサポートされるようになりました。これにより、保存時にデータを暗号化すること、つまり、Amazon &S3 がそのデータセンター内のディスクにオブジェクトデータを書き込むときにデータを暗号化することを、Amazon S3 にリクエストできます。AWS SDK for Java および AWS SDK for .NET には、REST API の更新に加えて、サーバー側暗号化をリクエストするために必要な機能もあります。さらに、AWS マネジメントコンソールを使用して、オブジェクトのアップロード時にサーバー側暗号化をリクエストすることもできます。データ暗号化については、「Using Data Encryption」を参照してください。	2011 年 10 月 4 日
ドキュメントの更新	ドキュメントのバグの修正に加えて、このリリースには、次の機能強化が含まれています。 • 「リクエストの実行 (p. 11)」セクションに Ruby および PHP のサンプルが追加されました。 • 署名付き URL の生成および使用方法を説明するセクションが追加されました。詳細については、「他ユーザーとのオブジェクトの共有 (p. 163)」および「署名付き URL を使用したオブジェクトのアップロード (p. 215)」を参照してください。 • AWS Explorer for Eclipse および AWS Explorer for Visual Studio を紹介する既存のセクションが更新されました。詳細については、「AWS dynamo と Explorer の使用 (p. 475)」を参照してください。	2011 年 9 月 22 日
一時的なセキュリティ認証情報を使用したリクエスト送信のサポート	AWS アカウントおよび IAM ユーザーセキュリティ認証情報を使用して認証済みリクエストを Amazon S3 に送信できるだけでなく、AWS Identity and Access Management (IAM) から取得する一時的なセキュリティ認証情報を使用してリクエストを送信できるようにもなりました。AWS Security Token Service API または AWS SDK ラッパーライブラリを使用して、これらの一時的な認証情報を IAM からリクエストできます。これらの一時的なセキュリティ認証情報は、ご自分で使用するためにリクエストするか、フェデレーションユーザーおよびアプリケーションに渡すことができます。この機能により、AWS の外部でユーザーを管理し、それらのユーザーに AWS リソースにアクセスするための一時的なセキュリティ認証情報を提供できます。 詳細については、「リクエストの実行 (p. 11)」を参照してください。 一時的なセキュリティ認証情報の IAM サポートについては、「Granting Temporary Access to Your AWS Resources」を参照してください。	2011 年 8 月 3 日

変更点	説明	日付
マルチパートアップロード API 拡張による最大 5 TB のオブジェクトのコピー	このリリース以前、Amazon S3 API では 5 GB までのサイズのオブジェクトのコピーをサポートしていました。5 GB 以上のオブジェクトをコピーできるようにするため、新しいオペレーション `Upload Part (Copy)` を加えて、マルチパートアップロード API を拡張しました。このマルチパートアップロード操作を使用して、最大 5 TB のサイズのオブジェクトをコピーできます。詳細については、「オブジェクトのコピー (p. 222)」を参照してください。 マルチパートアップロード API の概念については、「マルチパートアップロード API を使用したオブジェクトのアップロード (p. 178)」を参照してください。	2011 年 6 月 21 日
HTTP を介した SOAP API 呼び出しの無効化	セキュリティの向上のため、HTTP を介した SOAP API 呼び出しが無効になりました。認証済みおよび匿名の SOAP リクエストは SSL を使用して、Amazon S3 に送信する必要があります。	2011 年 6 月 6 日
IAM によるクロスアカウント委任の有効化	以前、Amazon S3 リソースにアクセスするには、IAM ユーザーは親 AWS アカウントと Amazon S3 リソース所有者の両方からのアクセス許可が必要でした。クロスアカウントアクセスによって、IAM ユーザーは所有者アカウントからのアクセス許可のみが必要になりました。つまり、リソース所有者が AWS アカウントへのアクセス許可を付与した場合、AWS アカウントはその IAM ユーザーのこれらのリソースへのアクセス許可を付与できるようになりました。 詳細については、「*Using Identity and Access Management*」の「Enabling Cross-Account Access」を参照してください。 バケットポリシーでのプリンシパルの指定については、「バケットポリシーでのプリンシパルの指定 (p. 338)」を参照してください。	2011 年 6 月 6 日
新しいリンク	このサービスのエンドポイント情報が、AWS General Reference に配置されるようになりました。詳細については、『AWS General Reference』の「Regions and Endpoints」を参照してください。	2011 年 3 月 1 日
Amazon S3 での静的ウェブサイトのホストのサポート	Amazon S3 に、静的ウェブサイトをホストするための拡張サポートが導入されました。これには、インデックスドキュメントとカスタムエラードキュメントのサポートが含まれます。これらの機能を使用すると、バケットのルートまたはサブフォルダへのリクエスト (例えば `http://mywebsite.com/subfolder`) は、バケット内のオブジェクトのリストの代わりに、インデックスドキュメントを返します。エラーが検出されると、Amazon S3 は Amazon S3 エラーメッセージの代わりにカスタムエラーメッセージを返します。詳細については、「Amazon S3 での静的ウェブサイトのホスティング (p. 411)」を参照してください。	2011 年 2 月 17 日
レスポンスヘッダー API のサポート	GET Object REST API により、各リクエストの REST GET Object リクエストのレスポンスヘッダーを変更できるようになりました。つまり、オブジェクト自体を変更せずに、レスポンスのオブジェクトメタデータを変更できます。詳細については、「オブジェクトの取得 (p. 153)」を参照してください。	2011 年 1 月 14 日

変更点	説明	日付
大容量オブジェクトのサポート	Amazon S3 で S3 バケットに保存できるオブジェクトの最大サイズが 5 GB から 5 TB に増加しました。REST API を使用する場合、単一の PUT 操作で、最大 5 GB のオブジェクトをアップロードできます。これよりも大きなオブジェクトは、マルチパートアップロード REST API を使用して、オブジェクトを分割してアップロードする必要があります。詳細については、「マルチパートアップロード API を使用したオブジェクトのアップロード (p. 178)」を参照してください。	2010 年 12 月 9 日
マルチパートアップロード	マルチパートアップロードにより、Amazon S3 に高速で柔軟にアップロードできます。一連のパーツとして、単一のオブジェクトをアップロードできます。詳細については、「マルチパートアップロード API を使用したオブジェクトのアップロード (p. 178)」を参照してください。	2010 年 11 月 10 日
バケットポリシーでの正規化 ID のサポート	バケットポリシーで正規化 ID を指定できるようになりました。詳細については、「バケットポリシーでのプリンシパルの指定 (p. 338)」を参照してください。また、サンプルについては、「正規 ID を使用した CloudFront オリジンアクセスアイデンティティへのアクセス許可の付与 (p. 334)」を参照してください。	2010 年 9 月 17 日
Amazon S3 と IAM の連携	このサービスに、AWS Identity and Access Management（IAM）が組み込まれました。詳細については、「Using AWS Identity and Access Management」の「Integrating with Other AWS Products」を参照してください。	2010 年 9 月 2 日
通知	Amazon S3 通知機能により、Amazon S3 がバケットに対するキーイベントを検出した場合に、Amazon Simple Notification Service（Amazon SNS）トピックに対するメッセージを発行するように、バケットを設定できます。詳細については、「Setting Up Notification of Bucket Events (p. 436)」を参照してください。	2010 年 7 月 14 日
バケットポリシー	バケットポリシーは、バケット、オブジェクト、一連のオブジェクト間でアクセス許可を設定するために使用するアクセス管理システムです。この機能は、アクセスコントロールリストを補完し、多くの場合に置き換わるものです。詳細については、「Bucket Policies (p. 7)」を参照してください。	2010 年 7 月 6 日
すべてのリージョンで使用可能なパススタイル構文	Amazon S3 では、米国クラシックリージョンの、またはバケットがリクエストのエンドポイントと同じリージョンにある場合、すべてのバケットでパススタイル構文がサポートされるようになりました。詳細については、「Virtual Hosting (p. 63)」を参照してください。	2010 年 6 月 9 日
欧州 – アイルランドの新しいエンドポイント	Amazon S3 で、欧州（アイルランド）のエンドポイント `http://s3-eu-west-1.amazonaws.com` が提供されるようになりました。	2010 年 6 月 9 日
コンソール	AWS Management Console 経由で、Amazon S3 を使用できるようになりました。コンソールの任意の Amazon S3 機能については、『*Amazon Simple Storage Service Console User Guide*』を参照してください。	2010 年 6 月 9 日

変更点	説明	日付
低冗長化	Amazon S3 で、低冗長化ストレージに Amazon S3 のオブジェクトを保存することにより、ストレージコストを削減できるようになりました。詳細については、「低冗長化ストレージ (p. 7)」を参照してください。	2010 年 5 月 12 日
新しいリージョンのサポート	Amazon S3 でアジアパシフィック (シンガポール) リージョンがサポートされるようになりました。詳細については、「Buckets and Regions (p. 90)」を参照してください。	2010 年 4 月 28 日
オブジェクトのバージョニング	本リリースでオブジェクトのバージョニングが導入されました。すべてのオブジェクトはキーとバージョンを持つことができます。バケットのバージョニングを有効にすると、Amazon S3 はバケットに追加されたすべてのオブジェクトに一意のバージョン ID を付与します。この機能により、意図しない上書きと削除からの復元が可能です。詳細については、「Versioning (p. 8)」と「Using Versioning (p. 388)」を参照してください。	2010 年 2 月 8 日
新しくサポートされるリージョン	Amazon S3 で米国西部 (北カリフォルニア) リージョンがサポートされるようになりました。このリージョンへのリクエストのための新しいエンドポイントは、s3-us-west-1.amazonaws.com です。詳細については、「Buckets and Regions (p. 90)」を参照してください。	2009 年 12 月 2 日
AWS SDK for .NET	AWS では、REST や SOAP の代わりに .NET 言語固有の API を使用してアプリケーションを構築するソフトウェア開発者向けに、ライブラリ、サンプルコード、チュートリアルなどのリソースを提供するようになりました。これらのライブラリはリクエスト認証、リクエストの再試行、エラー処理などの基本機能 (REST または SOAP API に含まれない) を提供しているため、簡単に開始できます。言語固有のライブラリおよびリソースの詳細については、「AWS dynamo と Explorer の使用 (p. 475)」を参照してください。	2009 年 11 月 11 日

用語集

100-continue	クライアントが実際にリクエストを送信する前に、サーバーがリクエストを受け付けることができるかどうかを確認するメソッド。大容量の PUTs を行う場合、これを行うことで時間と帯域幅の料金を節約できます。
アカウント	特定の開発者に関連する AWS アカウント
認証	お客様の ID をシステムに証明するプロセス。
バケット	Amazon S3 に格納されるオブジェクトのコンテナ。すべてのオブジェクトはバケット内に格納されます。例えば、`photos/puppy.jpg` という名前のオブジェクトが `johnsmith` バケットに格納される場合、URL `http://johnsmith.s3.amazonaws.com/photos/puppy.jpg` を使ってアドレスを解決できます。
既定アクセスポリシー	バケットやオブジェクトに適用できる標準のアクセスコントロールポリシー。private、public-read、public-read-write、authenticated-read のオプションがあります。
正規化	データを Amazon S3 などのサービスが認識できるように標準形式に変換するプロセス。
整合性モデル	Amazon S3 が高可用性を実現するための手法。Amazon のデータセンター内の複数のサーバーにデータが複製されます。「success」が返された後、データは安全に格納されます。ただし、変更に関する情報が直ちに Amazon S3 全体で複製されるわけではありません。
削除マーカー	削除マーカーは、キーとバージョン ID を持ち、コンテンツがないオブジェクトです。バージョニングが有効になっているバケット内のオブジェクトに対して DELETE リクエストを実行すると、削除マーカーが自動的にバケットに挿入されます。
デフォルトで拒否	デフォルト設定が DENY のアクセス許可。
キー	バケット内オブジェクトの個別の識別子。バケット内のすべてのオブジェクトは、厳密に 1 個のキーを持ちます。バケットとキーの組み合わせによって各オブジェクトが一意に識別されるため、Amazon S3 は「バケット + キー」とオブジェクト自体の基本的なデータマップであると考えることができます。Amazon S3 内の各オブジェクトは、ウェブサービスエンドポイント、バケット名、およびキーの組み合わせによって一意にアドレスを指定できます。http://doc.s3.amazonaws.com/2006-03-01/AmazonS3.wsdl では、「doc」がバケット名で「2006-03-01/AmazonS3.wsdl」がキーです。

メタデータ	メタデータは、オブジェクトについて説明する、名前と値のペアのセットです。これには最終更新日などのデフォルトのメタデータや、Content-Type などの標準 HTTP メタデータが含まれます。開発者が、オブジェクトの格納時にカスタムメタデータを指定することもできます。
null オブジェクト	バージョン ID が `null` であるオブジェクト。Amazon S3 では、バケットのバージョンが停止されると null オブジェクトがバケットに追加されます。バケット内の各キーには null オブジェクトを 1 つだけ設定できます。
オブジェクト	Amazon S3 に格納される基本的なエンティティ。オブジェクトは、オブジェクトデータとメタデータで構成されます。データ部分を Amazon S3 から見ることはできません。
パート	オブジェクトデータの隣接部分。Amazon S3 マルチパートアップロード機能では、大容量のオブジェクトをパート単位でアップロードできます。オブジェクトパートを個別に順不同でアップロードできます。いずれかのパートの送信に失敗しても、他のパートに影響を与えることなく、そのパートを再送信できます。すべてのオブジェクトパートがアップロードされると、Amazon S3 がこれらのパートを組み合わせてオブジェクトを作成します。
サービスエンドポイント	宛先 URL 内で通信を試みる際に使用するホストとポート。仮想ホスト形式のリクエストの場合、これは `mybucket.s3.amazonaws.com` です。パス形式のリクエストの場合、これは `s3.amazonaws.com` です。
バージョニング	Amazon S3 内の各オブジェクトにはキーとバージョン ID があります。オブジェクトのバージョン ID が異なっていてもキーが同じであれば、同じバケットに格納できます。`PUT Bucket versioning` を使用すると、バケット層でバージョニングが有効になります。

Index

Symbols

A

B

C

D

F

H

J

P

R

S

T