
# クラウドコンピューティングのための アーキテクチャ・ベストプラクティス

2011 年 5 月

ジネッシュ・バリア [著]

玉川 憲、大谷晋平 [監訳]

# はじめに

近年、ソフトウェアアーキテクトは、高度にスケーラブルな(高い拡張性を持つ)アプリケーションを構築するベストプラクティスを見いだし、実践してきました。今日の「テラの時代」では、成長の一途をたどるデータベース、予期できないトラフィックパターン、さらに早い応答時間への要求のために、こうした概念がより適用されるようになってきています。本書では、こうした従来の概念をとりあげ再び論じながらも、これらの概念がクラウドコンピューティングに照らし合わせてさらにどのように進化していくのかを解説します。また、とくにクラウドのダイナミックな特質のために出現してきた、弾力性などのこれまでは前例のない概念についても説明します。

本書は、固定化された物理的環境から仮想化されたクラウド環境へと、エンタープライズクラスのアプリケーションの移行を準備している**クラウドアーキテクト**を対象としています。本書では、新たに**クラウドアプリケーション**を構築する、または既存のアプリケーションをクラウドに移行する際の概念、原則、ベストプラクティスに焦点を当てます。

# 背景

クラウドアーキテクトとして大切なことは、クラウドコンピューティングの利点を理解することです。この章では、クラウドコンピューティングのビジネス上と技術上の利点、および今日利用可能な各種の AWS サービスについて説明します。

## クラウドコンピューティングのビジネス上の利点

クラウドでアプリケーションを構築するにあたり、明確なビジネス上の利点がいくつかあります。以下にそのいくつかを挙げます。

**ほとんどゼロのインフラストラクチャ先行投資**:大規模なシステムを構築する必要がある場合、土地、物理的セキュリティ、ハードウェア(ラック、サーバー、ルーター、バックアップ電源)、ハードウェアの管理(電源管理、冷房装置)、運用スタッフに、莫大な費用がかかります。先行投資が莫大にかかるため、通常プロジェクトが開始する前でさえ、経営陣による承認の段階をいくつも経なければなりません。ユーティリティスタイルのクラウドコンピューティングを用いれば、固定費や立ち上げ費用は必要ありません。

**ジャストインタイム インフラストラクチャ**:これまでは、もしアプリケーションが人気を博しても、インフラストラクチャをそれに従い拡張できなかった場合、ある意味成功の犠牲者になってしまいました。反対に、かなりの投資を行ったのにも関わらず、アプリケーションに人気が出なかった場合は、失敗の犠牲者になっていました。しかし、ジャストインタイム方式の自己調達を用いて、アプリケーションをクラウドで導入することにより、大規模システムのために事前に大容量を調達することについて心配する必要がなくなります。これにより、アプリケーションが人気を博して成功した場合のみ、スケールアップし、使用した分だけ支払えば良いため、アジリティが向上し、リスクが減り、運用コストが削減されます。

**さらに効率的なリソースの活用**:システム管理者は、通常ハードウェアの調達(容量を使い果たす時期)に悩み、インフラストラクチャのより高い使用率(過剰容量を抱えアイドルしている時期)に想いを馳せます。クラウドを使用すれば、アプリケーションにリソースの要求や解放をオンデマンドで実行させることにより、さらに効果的かつ効率的にリソースを管理できます。

**利用量ベースのコスト設定**:ユーティリティスタイルの価格設定により、利用したインフラストラクチャの分のみ請求されます。割り当てられたが使用していないインフラストラクチャに対して支払いは発生しません。この方式は、コスト削減の新たな一面です。クラウドアプリケーションを最適化すれば、直ちにコスト削減を認識できます(ときには翌月の請求書ですぐに認識できることもあるでしょう)。たとえば、キャッシングレイヤーへのデータリクエストを 70%減少できると、ただちに節約となり、翌月の請求書ですぐに結果がわかります。さらに、もし御社がクラウドの上でプラットフォームを構

築して顧客に提供しているのであれば、同一のフレキシブルな従量制のコスト構造を御社の顧客に得て頂くことができます。

**市場投入までの時間を短縮**：並列化は、プロセスをスピードアップする優れた方法です。並列して実行することができるジョブであれば、それがコンピュータ処理中心であれデータ処理中心であれ、1 台のマシンで処理するのに 500 時間かかっていたとします。同じジョブに対してクラウドアーキテクチャを用いれば、500 インスタンスを作成して起動することで、1 時間で処理できるのです。弾力性のあるインフラストラクチャを利用することで、並列化を利用する機能を搭載したアプリケーションが可能となり、市場投入までの時間を短縮し、対費用効果を高められるでしょう。

## クラウドコンピューティングの技術上の利点

クラウドコンピューティングの技術上の利点をいくつか挙げます。

**自動化ー「スクリプト可能なインフラストラクチャ」**：プログラマブル(API 駆動) インフラストラクチャを活用して、繰返し可能なビルドシステムやデプロイシステムを構築可能です。

**自動スケーリング**：アプリケーションは、予期しない需要に合わせて、人間が介入せずにスケールアップ、スケールダウンが可能です。自動スケーリングにより自動化が推進され、より効率運用が可能となります。

**積極的なスケーリング**：トラフィックパターンの適切な計画を理解し、予想される需要にあわせてアプリケーションをスケールアップやスケールダウンすることで、スケーリング中にコストを抑えることができます。

**より効率的な開発ライフサイクル**：本番運用環境を容易にコピーして、開発環境とテスト環境に利用できます。同様に、ステージング環境を、容易に本番運用環境に昇格します。

**テスト容易性の改善**：テストを実施するためのハードウェアが欠乏することが無くなります。開発プロセスの各ステージでテストを実施し、自動化できます。テストフェーズ期間の間だけ、事前に設定した環境を備えた「インスタントテストラボ」構築することすら可能となります。

**ディザスタリカバリーと事業継続性**：クラウドは、DR サーバーとデータストレージの複合システムの維持のために低コストの選択肢を提供します。クラウドを使用すれば、地理的分布を有効に活用し、他のロケーションの環境を数分で複製します。

**トラフィックのクラウドへの「オーバーフロー」**：数回クリックし、効果的なロードバランシング戦略を使うだけで、余分なトラフィックをクラウドに送る、オーバーフロー防止型の完全なアプリケーションを作成できます。

## Amazon Web Services クラウドについて理解するには

Amazon Web Services (AWS) クラウドは、拡張性を持ったソリューションを展開するための、極めて信頼性の高いスケーラブルなインフラを提供します。このシステムは、最低限のサポートコストと管理コストで維持でき、御社のオンプレミスや外部のデータセンターと比べると、非常にフレキシブルに構築が可能です。

AWS は、様々な種類のインフラストラクチャサービスを提供しています。以下のダイアグラムでは、AWS の用語を紹介しており、御社のアプリケーションがどのように AWS と連携することができるかを理解し、AWS の異なるサービス間の関係を理解するのに役立ちます。

Amazon Elastic Compute Cloud (Amazon EC2[1]) は、クラウド内で規模を自在に変更できるコンピュータ処理能力を提供するウェブサービスです。オペレーティングシステム、アプリケーションソフトウェアおよび関連コンフィギュレーション設定を Amazon Machine Image (AMI)として組み込むことができます。その AMI(監注:AMI はサーバーコピー、サーバーテンプレートと考えると分かりやすい)を基に、シンプルなウェブサービスコールを使用して、複数の仮想化された**インスタンス**(監注:実際に立ち上がっている仮想サーバーのこと)を立ち上げたり落としたりできます。これにより、必要とするサーバー容量の要件が変更するにつれて、サーバー容量を素早くスケールアップ/スケールダウンすることが可能です。**On-Demand Instances**(オンデマンドインスタンス、従量課金制)を購入すれば、時間単位でインスタンスの支払いを行えます。また、**Reserved Instances**(リザーブドインスタンス、購入時の予約金+従量課金制) は、最初に予約金を支払うことで、従量課金制よりも低い時間課金でインスタンスを実行できます。さらに、**Spot Instances**(スポットインスタンス)をご利用いただくと、未使用のサーバー容量に入札することができ、さらにコストの削減を図ることが可能です。インスタンスは、1 ヶ所または複数の Region(以下、リージョン)で起動できます。各リージョン内には、複数の **Availability Zones**(アベイラビリティゾーン)が存在しています。アベイラビリティゾーンは、他のゾーンからの影響を受けないように各々独立したロケーションです。同一リージョン内のアベイラビリティゾーン間においては、待ち時間の少ないネットワーク接続が提供され、安価に利用できます。

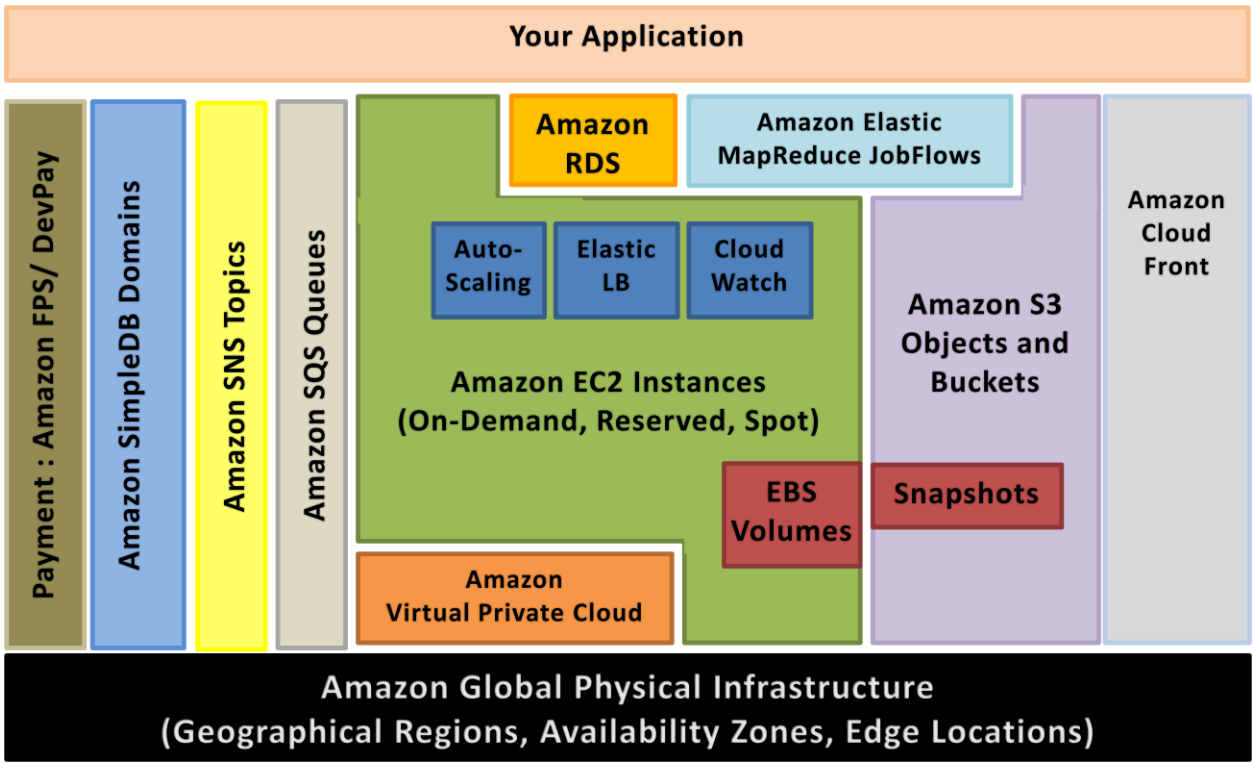


図 1: Amazon Web Services

Elastic IP は、固定 IP アドレスの割り当てを取得できるサービスであり、それをインスタンスに対してプログラマブルに割り当てることができます。Amazon CloudWatch [2]を使用して Amazon EC2 インスタンスを監視することができ、リソースの利用率、運用パフォーマンス、全体的な需要パターン(CPU の使用状況、ディスクの読み取りと書き込み、ネットワー

[1]Amazon EC2 に関する詳細は、http://aws.amazon.com/ec2 をご覧ください。
[2]Amazon CloudWatch の詳細については、http://aws.amazon.com/cloudwatch/ をご覧ください。

クトラフィックなどのメトリックスを含む）を表示できます。Auto Scaling[3]機能(自動スケーリング)を使用して **Auto Scaling Group** を作成し、Amazon CloudWatch が収集するメトリックに基づいて、特定の条件でサーバー容量を自動的にスケーリングできます。また、Elastic Load Balancing[4]サービスを使用して、弾力的なロードバランサーを構築でき、受信トラフィックを分散できます。Amazon Elastic Block Storage (EBS)[5] ボリュームは、ネットワーク接続持続性ストレージを Amazon EC2 インスタンスに提供します。EBS ボリュームの任意の時間の対応したスナップショットを作成し、Amazon Simple Storage Service (Amazon S3)に保管することが出来ます。[6]

Amazon S3 は、耐久性の高い分散型データストレージです。シンプルなウェブサービスインターフェースを用いて、大量のデータをオブジェクトとしてバケット（監注: S3 の固有名詞であり、データを入れる入れ物の名前）にいつでも保管しておき、いつでも取り出せます。標準的な HTTP を用いて、ウェブ上のどこからでもアクセスできます。Amazon CloudFront は、コンテンツ配信[7]（静的またはストリーミング）のウェブサービスで、S3 のオブジェクトのコピーを、世界中の 18 エッジロケーション(監注: 2011 年 5 月時点)で分散してキャッシュし、配信できます(監注: 2011 年より、S3 のみならず、EC2 やその他のリソースを配信できるようになっている)。Amazon SimpleDB [8]は、複雑な運用を気にすることなく、データベースの中核機能、いわば、データのリアルタイム検索と簡単なクエリを提供するクラウド・データベースのウェブサービスです。お客様はデータセットをドメイン(監注：SimpleDB の固有名詞であり、テーブルのようなもの)に組み込み、指定したドメインに保管されているすべてのデータに対してクエリを実行することができます。ドメインは、属性−値のペアで記述した項目の集まりです。Amazon Relational Database Service[9] (Amazon RDS) は、クラウド上のリレーショナルデータベースであり、セットアップ、運用、スケーリングを非常に簡単に行えます。DB インスタンスを起動するだけで、フル機能を搭載している MySQL データベース にアクセスできます。バックアップ、パッチ管理などの通常のデータベース管理に悩むことはありません。

Amazon Simple Queue Service (Amazon SQS[10]) は、コンピュータとアプリケーションの間でメッセージをやりとりする際に、それらを保管するための、信頼性の高い、拡張性のある、キューサービスを提供します。

Amazon Simple Notifications Service (Amazon SNS) [11]は、トピック(監注：SNS の固有名詞であり、通知するメッセージを入れておく入れ物の名前)を作成しておけば、購読/配信といった手順が準備されているので、クラウドからアプリケーションまたは人に通知する簡単な方法を提供します。

Amazon Elastic MapReduce[12]は、Amazon Elastic Compute Cloud (Amazon EC2) と Amazon Simple Storage Service (Amazon S3)といった Amazon が持つ大規模なインフラストラクチャ上でホストされた Hadoop フレームワークを提供します。利用者独自の JobFlows(ジョブフロー)を作成し、実行することができます。JobFlow は、一連の MapReduce ステップとなります。

---

[3]自動スケーリング機能の詳細については、 http://aws.amazon.com/auto-scaling をご覧ください。
[4]Elastic Load Balancing 機能の詳細については、 http://aws.amazon.com/jp/elasticloadbalancing をご覧ください。
[5]Elastic Block Store の詳細については、 http://aws.amazon.com/jp/ebs をご覧ください。
[6]Amazon S3 の詳細については、 http://aws.amazon.com/jp/s3 をご覧ください。
[7] Amazon CloudFront の詳細については、 http://aws.amazon.com/jp/cloudfront をご覧ください。
[8] Amazon SimpleDB の詳細については、 http://aws.amazon.com/jp/simpledb をご覧ください。
[9]Amazon RDS の詳細については、 http://aws.amazon.com/jp/rds をご覧ください。
[10] Amazon SQS の詳細については、 http://aws.amazon.com/jp/sqs をご覧ください。
[11]Amazon SNS の詳細については、 http://aws.amazon.com/jp/sns をご覧ください。
[12]Amazon ElasticMapReduce の詳細については、 http://aws.amazon.com/jp/elasticmapreduce を参照してください。

Amazon Virtual Private Cloud (Amazon VPC)[13]では、社内イントラネットワークを AWS 内にあるプライベートクラウドに拡張できます(監注:つまり、社内のIPアドレスを EC2 のインスタンスに割り当てて利用できます)。Amazon VPC は、IPSec トンネルモードを使用して、データセンターにあるゲートウェイと AWS にあるゲートウェイ間のセキュアな接続を構築します。

AWS は、Amazon の支払いインフラストラクチャを活用した各種決済、請求サービスも提供します(監注: 2011 年 5 月時点で日本のユーザーが利用するには制限があります)。[14]

すべての AWS インフラストラクチャサービスは、長期的なコミットメントや契約を必要としない、ユーティリティスタイルの価格設定を提供します。たとえば、 Amazon EC2 インスタンスの使用については時間単位で支払い、Amazon S3 の場合、 ストレージとデータ転送についてギガバイト単位で支払えます。これらのサービスの詳細と重量料金制の価格体系については、AWS のウェブサイトをご覧ください。

AWS クラウドを使用しても、ユーザーが慣れ親しんでいるフレキシビリティやコントロールを犠牲にすることはありません。

- プログラミングモデル、言語、オペレーティングシステム(Windows, OpenSolaris または Linux のいずれかのフレーバー)をお好みで自由に利用できます。
- ユーザー要件を最も満足させる AWS 製品を自由に選べます。どのサービスも個別に、または組み合わせて利用できます。
- AWS はサイズ変更が可能なリソース(ストレージ、ネットワーク帯域、コンピューティング)を提供するため、大量にも少量にも利用でき、利用した分だけ支払得ます。
- これまで使用してきたシステム管理ツールを自由に使用し、御社のデータセンターをクラウドに拡張できます。

[13] Amazon Virtual Private Cloud の詳細につきましては、 http://aws.amazon.com/jp/vpc をご覧ください。
[14]Amazon Flexible Payments Service の詳細については、 http://aws.amazon.com/fps をご覧ください。 Amazon DevPay については、 http://aws.amazon.com/devpay をご覧ください。

# クラウドの概念

クラウドは、高度にスケーラブルなインターネットアーキテクチャ[13]を構築する、という従来からある概念を強調しつつも、アプリケーションの構築と配置方法を完全に変えうる新しい概念を導入します。従って、概念段階から実践段階に移る際には、「すべて変更したのに、何も変わっていない」と感じることがあるかもしれません。クラウドは、いくつかのプロセス、パターン、プラクティス、基本哲学を変え、従来型のサービス指向 (SOA) のアーキテクチャの原理を際立たせます。この SOA の概念は、クラウドにおいて、これまでよりさらに重要となるのです。この章では、こうした新しいクラウドの概念と、さらに重要となった SOA の概念について説明します。

従来型のアプリケーションは、開発された当時には、経済的でアーキテクチャ的にも現代的であるという認識で構築されたのです。クラウドは、新しい基本哲学をもたらすので、ユーザーはこの方針を理解する必要があります。新しい基本哲学について、以下にみていきましょう。

## スケーラブルなアーキテクチャの構築

スケーラブルなインフラストラクチャを活用するには、スケーラブルなアーキテクチャの構築が不可欠です。

クラウドは、概念上、無制限のスケーラビリティを提供します。ただし、ユーザーが設計するアーキテクチャがスケーラブルではない場合、AWS のスケーラビリティをすべて活用できるわけではありません。クラウドとそれを利用するアーキテクチャは、協調して動作する必要があります。スケーラブルなインフラストラクチャを活かして、クラウドを十分に活用するには、アーキテクチャにおいて、一枚岩のコンポーネントや、そのボトルネックを識別し、オンデマンド調達機能を活用できないエリアを特定して、アプリケーションをリファクタリング(監注:ソフトウェアの外部的な振る舞いを保ちつつ、内部構造を改善していくこと)する必要があります。

真にスケーラブルなアプリケーションの特性:

- リソースの増加に比例して、パフォーマンスが増加する
- 多種多様なものを処理できる
- 効率的な運用ができる
- 回復力に富んでいる
- 成長するにつれて費用効率がさらに良くなる
  (ユニット数が増加するにつれて、ユニット単位あたりコストが下がる)。

上記のような性質をアプリケーションが備えておくべきであり、前述の特性を念頭に置きながらアーキテクチャを設計すると、アーキテクチャとインフラストラクチャの両方が一体となって、求めるスケーラビリティが達成されます。

## 弾力性に関する説明

以下のグラフは、アプリケーションが需要(監注:Web サーバーの場合、リクエストが急増するなど)に応えるためにスケーリングを行う場合に、クラウドアーキテクトがとりうる異なるアプローチを示しています。

スケールアップ アプローチ:スケーラブルなアプリケーションアーキテクチャについて頭を痛めることなく、大型のパワフルなコンピュータに重点的に投資して、需要に対処します(垂直スケーリング)。このアプローチは、通常ある程度はうまくいきますが、莫大な投資が必要です(図の「巨額の支出(Huge Capital Expenditure)」を参照)。また、新しい大型コンピュータが導入される前に需要がサーバー容量を超えてしまうことがあります(ダイアグラムの「あなたは顧客を失ったばかりです(You just lost your customers)」を参照)。

従来型のスケールアウトアプローチ:水平にスケーリングするアーキテクチャを構築し、インフラストラクチャに少額投資します。大半の企業や大型のウェブアプリケーションは、このパターンに従っており、アプリケーションコンポーネントを分配して、データベースを連結し、サービス指向のデザインを採用します。このアプローチは、多くの場合スケールアップアプローチよりもさらに効率的です。ただし、このアプローチでは、定期的に需要を予測し、需要を満たすためにある程度まとまった量のインフラストラクチャを導入する必要があります。このアプローチでは、多くの場合、過剰なサーバー容量(「キャッシュフローの焦げ付き」)や、定期的な手動モニタリングが必要となります。さらに、アプリケーションが Web 上で炎上した場合(よくスラッシュドット効果と呼ばれます[15])は、通常このアプローチはうまくいきません。
(注:上記 2 種のどちらのアプローチも、初期立ち上げ費用がかかり、本来は受動的です。)

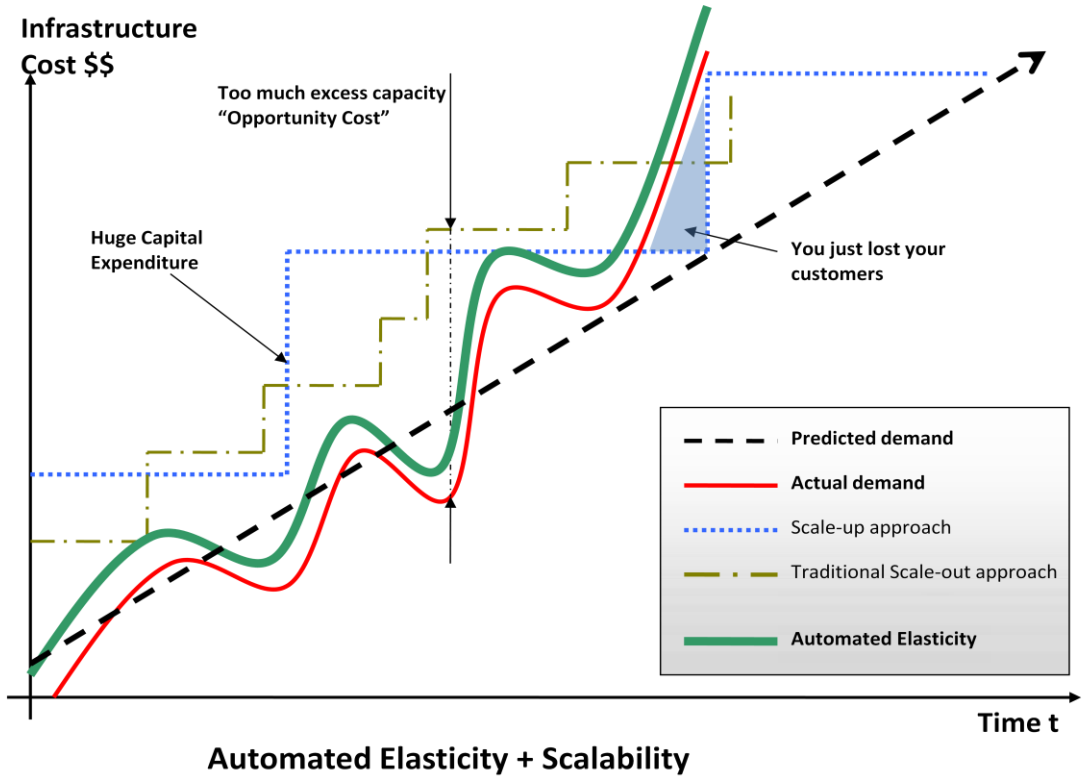


図 2:自動化された弾力性

[15] http://en.wikipedia.org/wiki/Slashdot_effect

従来型のインフラストラクチャは、一般に数年間にわたりアプリケーションが使用するコンピューティングリソース量を予測する必要があります。過小評価を行うと、アプリケーションは予想外のトラフィックを処理する馬力が足りず、顧客の不満につながる可能性があります。過剰評価を行うと、余分なリソースに資金を無駄遣いします。

しかし、オンデマンドで弾力性を持つクラウドにおいては、実際の需要に合わせてインフラストラクチャを調整できるので（拡張しようとも、収縮しようとも）、結果として、全体的な使用率が向上し、コスト削減となります。

弾力性は、クラウドの基本特性のひとつです。弾力性は、最小限の労力でコンピュータリソースを容易にスケールアップ/スケールダウンする能力です。弾力性こそが、究極的にクラウドの大半の利点の基となることを理解することは重要です。クラウドアーキテクトは、この概念を自分のものとし、クラウドの利点を最大限に活かすために、アプリケーションアーキテクチャに採り入れるのです。

従来は、アプリケーションは、固定化された、融通の利かない、事前に調達されたインフラストラクチャを用いて構築されていました。企業にとっては、毎日のようにサーバーを設定し、インストールをする必要性そのものが無かったのです。その結果、大半のソフトウェアアーキテクチャは、ハードウェアの迅速な増強や削減に対処していません。これまでは、新規リソースを取得するための調達時間や先行投資が高すぎたため、ソフトウェアアーキテクトは、ハードウェアの使用率を最適化することに対して、時間やリソースを費やさなかったのです。アプリケーションを実行するハードウェアがほとんど利用されていない場合でも、それは問題視されませんでした。新規リソースを数分で構築できるというアイデアそのものが実現不可能であったため、アーキテクチャ内の「弾力性」という概念は見過ごされていました。

クラウドでは、こうした考え方を変える必要があります。クラウドコンピューティングは、必要なリソースの取得プロセスを単純化します。前もって発注しておく、もしくは、使用しないハードウェアを保持しておく必要はありません。代わりに、クラウドアーキテクトは、必要なリソースを数分前にリクエストするか、調達プロセスを自動化し、クラウドのスケール性能と迅速な応答時間を十分に活用できるのです。リソースが必要なくなった場合、必要のないリソースや活用されていないリソースを解放する際に、同じ概念を適用できます。

アーキテクトが、こうした変化を受け入れられず、アプリケーションアーキテクチャで弾力性を実現しない場合、クラウドの利点を完全に活用することはできないでしょう。クラウドアーキテクトとして、創造的に考え、自社のアプリケーションで弾力性を実施する方法について考える必要があるのです。たとえば、毎夜ビルドを実行し、毎日午前 2 時に回帰テストとユニットテストを 2 時間実行するインフラストラクチャ（「QA/ビルドボックス」と呼ばれる）は、残りの時間はアイドル状態であったとします。そうすると、弾力性のあるインフラストラクチャを使うことで、この「活動中の」ビルドを毎夜実行し、夜間の 2 時間分の料金を支払うだけで済みます。同様に、社内の依頼申請ウェブアプリケーションについて考えます。このアプリケーションは、ピーク時の需要を満たすために、全日中ピーク容量（サーバー5 台 x24x7x365）で実行されていました。こちらも、トラフィックパターンに基づきオンデマンドで実行するように設定できます（例えば、5 台のサーバーが、午前 9 時から午後五時まで、2 台のサーバーが午後五時から午前 9 時まで）。

インテリジェントな弾力性のあるクラウドアーキテクチャの設計、すなわち、必要な時のみインフラストラクチャを実行するのは、それ自体が芸術的であるといえるでしょう。弾力性は、アーキテクチャ上の設計要件の 1 つであり、システムの属性といえます。

役に立つ質問: 自社のアプリケーションアーキテクチャで、どのコンポーネントやレイヤーが弾力的になれますか？そのコンポーネントを弾力的にするには何が必要ですか？自社のシステムアーキテクチャ全体に弾力性を実現できると、どんな影響がありますか？

次の章では、ユーザーのアプリケーションで弾力性を実現する特別なテクニックを紹介します。クラウドの利点を効率的に活用するには、この考え方で設計することが大切です。

## 制約を恐れない

ユーザーのアプリケーションをクラウドに移行することにしたとき、また、システムの仕様をクラウドで実行できるようにマップしようとするとき、クラウドには、オンプレミスにあるリソースが持つような正確な仕様がないことに気づくでしょう。たとえば、「クラウドは、あるサーバーで X 量の RAM を提供しない」、または「私のデータベースは、単一のインスタンスで取得できるよりももっと IOPS が必要です」などです。

クラウドは、抽象的なリソースを提供し、このリソースをオンデマンド調達モデルと組み合わせるとパワフルになります。クラウドリソースを使用するときは、恐れたりする必要はないし、束縛されることもありません。クラウド環境で御社のハードウェアとまったく同じ複製を得られなくても、クラウドを用いると、そのニーズを補うためにさらなるリソースを取得する能力を得られるからです。

たとえば、クラウドが、サーバーにおいて特定の値の RAM を提供しなかったり、一つのクラウドサーバーが提供する RAM よりも大量の RAM が必要であったりしても、memcached[16]などの分散キャッシュを使用するか、複数のサーバー全体でデータを分割するようにします。データベースがさらに IOPS を必要とし、クラウドの IOPS では直接マップできない場合、データのタイプやユースケースに応じて選択できるいくつかの推奨事項があります。読み取りに重点を置くアプリケーションの場合、同期されたスレーブのサーバー群全体に読み取り負荷を分散できます。また、シャーディング(sharding)[10]アルゴリズムを使用して、必要な場所へデータを送ったり、各種のデータベース クラスタリングソリューションを使用したりできます。

振り返ってみると、オンデマンド調達機能をフレキシビリティと組み合わせると、表面上の制約が実際には問題とならず、スケーラビリティやシステムの全体的なパフォーマンスを向上できます。

## 仮想管理

クラウドの出現により、システム管理者の役割が「仮想システム管理者」に変わります。これは、管理者がアプリケーションについて知識を得て、全体的に会社にとって最適なことを決断するにつれ、実施する日常的なタスクがさらに興味深いものになっています。システム管理者は、サーバーの設置やソフトウェアのインストールやネットワークデバイスの配線を行う必要はありません。こうした退屈な仕事は、数度のクリックとコマンドラインの呼び出しに取って代わられています。インフラストラクチャはプログラム可能なため、クラウドは自動化を促進します。システム管理者は、技術スタックの上の層に注力し、スクリプトを使用して抽象的なクラウドリソースの管理法を学習する必要があります。

同様に、データベース管理者の役割も「仮想データベース管理者」に変わります。管理者は、ウェブベースのコンソールを通じてリソースを管理し、データベースのハードウェアの容量が欠乏したり、日常のプロセスを自動化する場合に、容量を新たに追加するスクリプトを実行します。仮想データベース管理者は、新しい導入手法(仮想マシンイメージ)を学び、新モデル(クエリの並行化、地理的な冗長性、非同期の複製)を採り入れ[11]、データアーキテクチャにおけるアプローチを再考し(シャーディング[9]、水平分割[13]、フェデレーション[14])、様々なタイプのデータセットに対してクラウドで利用可能なストレージオプションを上手に活用します。

従来型の企業では、アプリケーション開発者はネットワーク管理者と密接に協力せず、ネットワーク管理者はアプリケーションについて全く知識がないことも多々ありました。その結果として、ネットワークレイヤーとアプリケーションアーキテクチャ レイヤーでの最適化の可能性が見過ごされてきました。クラウドを使用すれば、2 つの役割をある程度のとこ

[16] http://www.danga.com/memcached/

ろまで 1 つにまとめることができます。未来のアプリケーションを設計する際に、企業は 2 つの役割の間で知識の相互交流を奨励すべきですし、これらの役割を統合しつつあることを認識する必要があります。

# クラウドのベストプラクティス

この章では、クラウドにアプリケーションを構築するのに役立つベストプラクティスを紹介します。

## 故障に備えた設計(Design for failure)をすれば、何も故障しない

経験則:クラウドでアーキテクチャを設計するときは悲観論者になるべきです。物事は失敗すると仮定する。言い換えると、故障から自動的に回復するための設計、実践、導入を実行するのです。

まず、ハードウェアは故障すると仮定します。停電が生じると仮定します。災害がアプリケーションに襲いかかると仮定します。1 秒あたりの予想リクエスト数をいつか超えてしまうと仮定します。時間が経つにつれて、アプリケーションソフトウェアは故障すると仮定します。悲観論者になることで、設計時に回復戦略について考えるようになり、全体的なシステムをさらにうまく設計するのに役立ちます。

時間がたてば故障する、ということを認識し、この考えをアーキテクチャに採り入れ、災害が襲ってからスケーラブルなインフラストラクチャに対処するということが起きないように、故障を処理するメカニズムをあらかじめ構築すればいいのです。そうすれば、クラウドに対して最適化された耐故障性の高いアーキテクチャを構築できます。

役に立つ質問: システムのあるノードが故障したらどうなりますか？この故障をどのように認識しますか？そのノードを交換するにはどうすればよいですか？どのようなシナリオを計画すべきですか？単一障害点はどこにありますか？アプリケーションサーバーの前にロードバランサーがある場合、そのロードバランサーが故障したらどうなりますか？アーキテクチャにマスターとスレーブがある場合、マスターノードが故障したらどうなりますか？フェールオーバーはどのように行われますか？また、新しいスレーブはどのように起動され、どのようにマスターに同期化されますか？

ハードウェアの故障に備えて設計するのと同じように、ソフトウェアの故障に備えても設計する必要があります。

役に立つ質問: 依存しているサービスがインターフェイスを変えると、当社のアプリケーションに何が生じますか？ダウンストリームサービスがタイムアウトになるか、例外を返したらどうなりますか？キャッシュキーがインスタンスのメモリー限度値を超えて成長したらどうなりますか？

この障害を処理するメカニズムを構築する必要があります。たとえば、障害が発生した場合は、次のような戦略が役立ちます。

1. 常にバックアップを作成しておき、データの復元を手順化し、それを自動化する
2. リブートの際に、再開すべきプロセススレッドを整理する
3. クエリからメッセージを再ロードし、システムの状態を再び戻せるようにする
4. 起動時/リブート時で (2) と(3)を可能にするため、事前設定、事前最適化されたバーチャルイメージを保持する
5. インメモリーセッションやステートフルなユーザーコンテキストを避け、それらをデータストアに移動させる

優れたクラウドアーキテクチャは、リブートや再始動(re-launches)に影響されません。GrepTheWeb(『クラウドアーキテクチャ白書』[6]で説明)では Amazon SQS と Amazon SimpleDB の組み合わせを使用することにより、コントローラーアーキテクチャ全体(監注:参考文献で挙げられている中で取り上げられているアーキテクチャ)が、様々な故障のタイプに対して回復力があります。たとえば、コントローラスレッドが稼働していたインスタンスが落ちた場合、何事もなかっ

たかのように以前の状態に戻します。このコントローラスレッドは、事前設定された Amazon Machine Image を構築しておくことで実現しています。このスレッドはリブート時に、 Amazon SQS キューからすべてのメッセージのキューを解除し、Amazon SimpleDB ドメインからの状態を読み取ります。

基盤となるハードウェアが故障する、という前提で設計するということは、実際に故障したときの準備をしっかりしておくことにつながります。

この設計方針は、Hamilton [11]の論文でも強調されているように、運用が楽なアプリケーションの設計に役立ちます。もし、この設計方針に加え、積極的な監視と、動的なロードバランスを行えば、複数テナントという性質上クラウドが持たざるを得ないネットワークとディスクのパフォーマンスの変化に対しても処理することが可能となります。

このベストプラクティスを実行するための AWS を用いた戦術を以下に挙げます。

1. Elastic IP を用いてフェールオーバーさせる: Elastic IP は、動的に再マップが可能な固定 IP です。迅速に別のサーバーに付け替え、フェールオーバーすることで、トラフィックを新サーバーに送れるようにします。旧バージョンから新バージョンにアップグレードする際や、ハードウェアが故障した場合に役に立つでしょう。
2. 複数の Availability Zones を活用する: Availability Zones は、概念としては、論理データセンターのようなものです。アーキテクチャを複数の Availability Zones に配置することにより、高可用性を確保できます。Amazon RDS Multi-AZ [21] 配備機能を活用し、複数の Availability Zones 全体でデータベースを自動的に複製します。
3. Amazon Machine Image を保持し、異なる Availability Zones でクローンを極めて容易に復元できるようにしておきます。Availability Zones 全体で複数のデータベーススレーブを維持し、ホットレプリカを準備します。
4. Amazon CloudWatch (または各種のリアルタイムのオープンソース モニタリングツール)を活用して可視性を高め、ハードウェアの故障やパフォーマンスの劣化が生じた場合に適切な手段を講じます。自動スケーリンググループを設定して、問題のある Amazon EC2 インスタンスを新しいものに取り替えられるようにして、サーバー数を維持します。
5. Amazon EBS を利用して cron ジョブを設定し、インクリメンタルスナップショットが Amazon S3 に自動的にアップロードされ、データがユーザーのインスタンスの寿命から独立して永続するようにします。
6. Amazon RDS を利用して、バックアップの保存期間を設定し、自動バックアップが行えるようにします。

## コンポーネントの分離

クラウドは SOA の設計原則を補強します。システムのコンポーネントを疎結合にすればするほど、スケーリングが大規模にうまく行えます。

鍵となるのは、互いに過度に依存し合わないコンポーネントを構築し、あるコンポーネントが何らかの理由で故障、スリープ(反応しない)またはビジー(反応が遅い)状態になっても、システムの他のコンポーネントが構築され、故障が生じてないかのように作業を継続することです。基本的には、疎結合は各種のアプリケーションのレイヤーやコンポーネントを隔離することであり、つまり、各コンポーネントは他のコンポーネントと非同期に相互作用し、お互いに「ブラックボックス」として扱います。たとえば、ウェブアプリケーションアーキテクチャの場合、アプリケーションサーバーをウェブサーバーとデータベースから隔離できます。アプリケーションサーバーは、ユーザーのウェブサーバーについて知らず、またウ

ェブサーバーは、アプリケーションサーバーについて知りません。従って、コード的な観点、機能的な観点からの依存関係はなくなります。バッチ処理のアーキテクチャにおいては、互いに独立している非同期コンポーネントを作成できます。

役に立つ質問: どのビジネスコンポーネントや機能が、現在のモノリシックアプリケーションから隔離できますか？それだけスタンドアロンで実行できますか？現在のシステムを壊さずに、このコンポーネントのインスタンスをさらに追加し、もっと多くのユーザーにサービスを提供するにはどうすればよいですか？コンポーネントが他のコンポーネントと非同期に相互作用するように、コンポーネントをカプセル化するにはどのぐらいの手間がかかりますか？

コンポーネントの分離、非同期システムの構築、水平にスケーリングを行うことは、クラウドの文脈では非常に重要です。同じコンポーネントのインスタンスをさらに追加してスケールアウトできるようになるだけでなく、革新的なハイブリッドモデルを設計する事が出来ます。幾つかのコンポーネントはオンプレミスで動き続け、その他のコンポーネントはクラウドのスケールを活用し、追加のコンピュートパワーや帯域を得るということが可能になるでしょう。最小限のコストで、スマートなロードバランシング戦略を実装することにより、余剰トラフィックをクラウドに「オーバーフロー」できます。

疎結合なシステムは、メッセージングキューを使用して構築する事が出来ます。2 つのどのようなコンポーネントの連携においても、キュー/バッファを使用するのであれば、並行性、高可用性、ロードスパイクをサポートすることができます。その結果、コンポーネントの一部が一時的に使用できなくても、システム全体が処理を継続する事が出来ます。あるコンポーネントが故障したか、一時的に使用できなくても、システムはメッセージをバッファーに保留し、コンポーネントが復旧したときに処理を再開することができます。

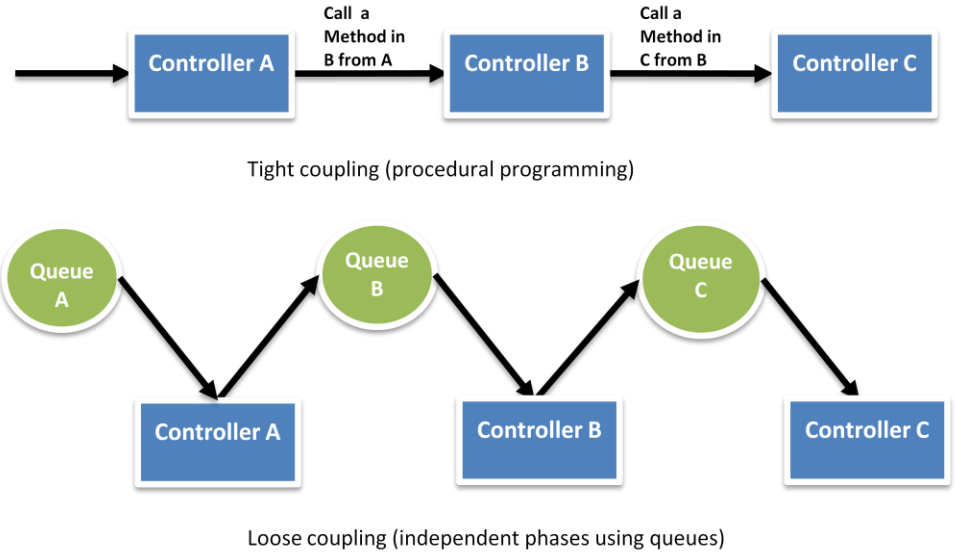


図 3: キューを使用してコンポーネントを分離

Cloud Architectures ペーパー[6]で説明した GrepTheWeb Architecture でも、キューを多用しているのがわかります。GrepTheWeb では、多数のリクエストが突然サーバーに到達したり(インターネットのせいでオーバーロードが起こっている状況)、または正規表現の処理に時間がかかると(コンポーネントの遅い応答)、Amazon SQS のキューは、遅延が他のコンポーネントに影響を与えないような方法でリクエストをバッファに保留します。

このベストプラクティスを実践するための AWS 特有の戦術:

1. Amazon SQS を使用してコンポーネントを独立させる[18]
2. Amazon SQS をコンポーネント間のバッファとして使用する[18]

3. このように各コンポーネントを設計すると、サービスインターフェースのみが公開され、適切な粒度で自らのスケーラビリティの責任を持ち、他のコンポーネントとは非同期でインタラクションを行います。
4. コンポーネントの論理的な構造を Amazon Machine Image にバンドルし、さらに頻繁にデプロイできるようにします。
5. アプリケーションを極力ステートレスにします。セッションの状態はコンポーネントの外に保存します（もし適切である場合は Amazon SimpleDB を使う事も検討）

## 弾力性の実装

クラウドは、アプリケーションに弾力性という新しい概念をもたらします。弾力性は、以下の3つの方法で実践できます。

1. 巡回スケーリング：一定間隔（毎日、週ごと、月ごと、四半期ごと）に発生する定期的なスケーリング
2. イベントベーススケーリング：予定されているビジネスイベントのためにトラフィックリクエストが急激に増加されると予想されるときに実施するスケーリング（新製品の立ち上げや、マーケティングキャンペーン等）
3. オンデマンドの自動スケーリング。モニタリングサービスを使用することにより、システムがトリガーを送信し、メトリックスに基づいてスケールアップ、ダウンする適切なアクションを実行します（例えばサーバーの利用量やネットワーク I/O 量などを活用）。

「弾力性」を実践するには、まずデプロイメントのプロセスを自動化して、コンフィギュレーションとビルドプロセスを合理化します。これにより、システムが人の介入を得ずにスケーリング可能となります。

その結果として、サーバーがほとんど稼働していない状況を改善し、リソースが需要に応じて調整されることを実現すれば、全体の使用率が向上するので、直接的な費用対効果が得られるでしょう。

### インフラストラクチャの自動化

クラウド環境を使用する最も重要な利点の 1 つに、クラウドの API を使用してデプロイメント・プロセスを自動化できる機能です。クラウドへの移行プロセスの終わりの時期ではなく、その初期の段階から、自動デプロイメント・プロセスに時間をかけることを推奨します。自動化され、繰り返し可能なデプロイメント・プロセスを構築すると、エラーの削減につながり、効果的でスケーラブルなアップデートプロセスを促進します。

デプロイメント・プロセスを自動化するには：

- 「レシピ」という小型の頻繁に使用されるスクリプト（インストールとコンフィギュレーション用）を作成する
- AMI にバンドルされているエージェントを使用して、コンフィギュレーションとデプロイメント・プロセスを管理する
- インスタンスをブートストラップする

### インスタンスをブートストラップ

インスタンスの起動時に「私は誰？私の役割は？」という質問をさせます。各インスタンスには、とりまく環境の中で演じる役割があります（ウェブアプリケーションの場合は「DB サーバー」、「アプリケーションサーバー」、「スレーブサーバー」）。この役割は、起動時に引数として渡され、AMI がブートした直後にどのようなステップをとるべきか示唆します。起動時に、インスタンスは、役割に基づいて必要なリソース（コード、スクリプト、コンフィギュレーション）を取得し、役割を果たすために自分自身を何らかのクラスタに紐付けます。インスタンスをブートストラッピングする利点としては：

1. 数回のクリックと最低限の手間で（開発、ステージング、本番）環境を再現

2. 抽象度の高いクラウドベースのリソースをよりコントロールしやすくなる
3. 手作業が原因のデプロイメント・エラーを削減する
4. ハードウェアの故障に対してより回復力がある自己修復、自己発見する環境を構築できる

AWS を用いてインフラストラクチャを自動化する戦術

1. Amazon EC2 の Amazon Auto-scaling の機能を利用して、異なるクラスタに自動スケーリンググループを定義します。
2. Amazon CloudWatch を利用して、システムメトリクス（CPU、メモリー、ディスク、I/O、ネットワーク I/O）をモニターし、適切な措置を講じるか（自動スケーリングサービスを使用して、新 AMI をダイナミックに起動）、または通知を送信します。
3. マシンのコンフィギュレーション情報をダイナミックに保管、取り込みます。 Amazon SimpleDB を利用し、インスタンスの起動時間中にデータを取り込みます（例、データベース接続文字列）。SimpleDB は、IP アドレス、マシン名、役割などのインスタンス情報の保管にも使用できます。
4. 最新のビルドを Amazon S3 のバケットにダンプし、システムの起動中に最新版のアプリケーションをダウンロードするように、ビルドプロセスを設計しておきます。
5. リソースマネジメントツール（自動化スクリプト、事前設定イメージ）の構築に自己投資するか、または Chef[17]、 Puppet[18]、CFEngine [19]または Genome[20]などの便利なオープンソースのコンフィギュレーションマネジメントツールを使用します。
6. Just Enough Operating System (JeOS[21]) とソフトウェアの依存情報を Amazon Machine Image にバンドルしておき、管理と維持をしやすくします。起動時にコンフィギュレーションファイルまたはパラメータをパスし、起動後はユーザーデータ[22]とインスタンスメタデータを検索します。
7. Amazon EBS ボリューム[23]から起動し、複数の Amazon EBS ボリュームをインスタンスに接続することで、バンドリングと起動時間を削減します。共通ボリュームのスナップショットを作成し、スナップショットを[24]適切なアカウント間で共有します。
8. アプリケーションコンポーネントは、自身が動いているハードウェアが(いつも)正常に稼働していることを仮定すべきではないですし、そのロケーションを前提にすべきでありません。たとえば、新ノードの IP アドレスを、クラスタに動的にアタッチします。もし、故障した場合は、自動的にフェールオーバーして、新たなクローンを立ち上げます。

[17] Chef の詳細については、 http://wiki.opscode.com/display/chef/Home をご覧ください。
[18] Puppet の詳細については、 http://reductivelabs.com/trac/puppet/ をご覧ください。
[19] CFEngine の詳細については、 http://www.cfengine.org/ をご覧ください。
[20] Genome の詳細については、 http://genome.et.redhat.com/ をご覧ください。
[21] http://en.wikipedia.org/wiki/Just_enough_operating_system
[22] インスタンスメタデータとユーザーデータの情報は、 http://docs.amazonwebservices.com/AWSEC2/latest/DeveloperGuide/index.html?AESDG-chapter-instancedata.html をご覧ください。
[23] Amazon EBS からの起動の機能の詳細については、 http://developer.amazonwebservices.com/connect/entry.jspa?externalID=3121 をご覧ください。
[24]スナップシェアの共有方法については、 http://aws.amazon.com/ebs/ をご覧ください。

## 並列化を考慮する

クラウドは、並列化を簡単に行えます。クラウドアーキテクトとして、ユーザーがクラウドからデータを要求するか、クラウドへデータを保存するか、クラウドでデータ処理（またはジョブの実行）をするかどうかにかかわらず、クラウドでアーキテクチャを設計する場合は、並列化の概念を採り入れる必要があります。クラウドは、極めて容易に繰り返し可能なプロセスを構築できるため、できるだけ並列化を実践し、さらに可能であれば自動化することを推奨します。

データにアクセス（検索して保管）するという観点では、クラウドでは大量の並列オペレーションを処理できるように設計されています。最大のパフォーマンスとスループットを達成するには、リクエストの並列化を活用する必要があります。複数の並列なスレッドを使用してリクエストをマルチスレッドにすると、リクエストを逐次的に処理するよりも速く、データを保管または取り出す事が出来ます。従って、可能な限り、クラウドアプリケーションのプロセスは、シェアードナッシング型を志向してスレッドセーフにしておく必要があり、マルチスレッドを最大限活用すべきです

クラウドで、リクエストを処理、実行するという観点では、並列化の活用は、さらに重要になります。ウェブアプリケーションにおける一般的なベストプラクティスでは、ロードバランサーを使用して、複数の非同期のウェブサーバー全体で受信リクエストを分散させます。バッチ処理アプリケーションの場合は、マスターのノードを用いて、並列でタスクを処理する複数のスレーブワーカーノードを作成することができます（Hadoop[25]などの分散処理フレームワークのように）。

弾力性と並列化を組み合わせると、クラウドの美しさが引き立ちます。クラウドアプリケーションは、API を数回呼び出すだけで、数分で設定済みのコンピュートインスタンスのクラスタを構築し、並列でタスクを実行してジョブを処理し、結果を保管して、全てのインスタンスを終了させる事が出来ます。[6]で説明している GrepTheWeb アプリケーションは、その一例です。

AWS を用いた並列化戦術：

1. ベストプラクティスペーパーで説明しているように、 Amazon S3 リクエストをマルチスレッド化する[2]
2. Amazon SimpleDB GET と BATCHPUT のリクエストをマルチスレッド化する[3][4][5]
3. 日次のバッチ処理など各バッチに対し、Amazon Elastic MapReduce Service を使って JobFlow を作成する（インデックスか、ログ分析など）事で、ジョブを並列に処理し、時間を節約する
4. Elastic Load Balancing サービスを利用し、複数のウェブアプリケーションサーバー全体で、ダイナミックに負荷を拡散する

## 動的データをコンピュータの近くに、静的データをエンドユーザーの近くに保管する

一般に、レイテンシを減らすには、データを計算または処理要素のできるだけ近くに保管すると良いでしょう。クラウドでは、この場合のベストプラクティスは、インターネットのレイテンシに対処する必要があるため、さらに関連性が深く、重要になります。とりわけ、クラウドでは、ギガバイト単位でクラウドの内外からのデータ移転に対して料金がかかるので、コストがかさみがちです。

クラウド外にある大量のデータを処理する必要がある場合は、データをまずクラウドに「出荷」してデータを移し、次に計算を実行します。たとえば、データウェアハウスアプリケーションの場合、データセットをクラウドに移行し、次にデータセットにたいして並列クエリを実行することを推奨します。リレーショナルデータベースを用いてデータを保管、検索するウェブアプリケーションの場合は、データベースとアプリケーションサーバーをクラウドに一度にまとめて移行することを推奨します。

[25] http://hadoop.apache.org/

クラウドでデータが生成されると、データを消費するアプリケーションもクラウドに展開されるので、クラウド内における無料転送料金と低レイテンシを最大限に活用できます。たとえば、ログとクリックストリームデータを生成する E コマースウェブアプリケーションの場合、クラウドでログアナライザとレポートエンジンを実行することを推奨します。

逆に、データが静的でそれほど頻繁に変更されない場合（たとえば、画像、ビデオ、オーディオ、PDF、JS CSS ファイル）、コンテンツ配信サービスを利用して、静的データをエンドユーザー（依頼者）に近いエッジロケーションにキャッシュし、アクセスのレイテンシを短縮します。キャッシングのおかげで、コンテンツ配信サービスは、頻繁に利用されるオブジェクトに素早くアクセスできます。

このベストプラクティスを実践するための AWS を用いた戦術：

1. Import Export Services を利用して、データドライブを Amazon に移転します[26]。大量のデータを移行するには、インターネットでアップロードするより、スニーカーネット(データを媒体で送付する事)[27]を使用する方が速く、コストもかかりません。
2. 同一の Availability Zone を利用してマシンのクラスタを起動します。
3. Amazon S3 バケットで Amazon CloudFront のディストリビューションを作成し、Amazon CloudFront に、全世界の 18 のエッジロケーション全体にバケットのコンテンツをキャッシュさせます。

## セキュリティのベストプラクティス

マルチテナント環境においては、クラウドアーキテクトは、セキュリティの懸念を挙げます。セキュリティは、クラウドアプリケーションアーキテクチャの全てのレイヤーで実践すべきです。物理的なセキュリティは、通常サービスプロバイダが処理しており（セキュリティホワイトペーパー[7]）、これは、クラウドを使用する上での大きなメリットです。ネットワークとアプリケーションレベルのセキュリティは、ユーザー側の責任であり、普段から企業で利用しているベストプラクティスを実践していただけます。この章では、AWS 環境でクラウドアプリケーションの安全を確保するための特殊なツール、機能、ガイドラインについて説明します。基本セキュリティを実施するためのツールや機能を利用し、さらに、適切と思われる標準的な方法を使用して、セキュリティのベストプラクティスを実践することを推奨します。

### 転送中のデータを保護

ブラウザとウェブサーバー間で秘密情報を交換する場合は、サーバーインスタンスに SSL を設定します。VeriSign[28]または Entrust[29]などの外部認証機関からの認証を必要とします。認証に含まれるパブリックキーは、サーバーをブラウザに対して認証し、両方向のデータの暗号化に使用される共有セッションキーを作成する基盤として働きます。

いくつかのコマンドライン呼び出しによって Virtual Private Cloud を構築（Amazon VPC を使用できる）し、これを使うと、AWS クラウド内で独自に隔離されたリソースを使用し、そのリソースを業界標準の暗号化された IPsec VPN 接続を使用して既存の企業システムに直接接続することができます。

また VPC を使わなくても、Amazon EC2 インスタンスで OpenVPN サーバーをセットアップし[15]、すべての全ユーザー PC で OpenVPN クライアントをインストールする事も可能です

[26] Import Export Services の詳細については、 http://aws.amazon.com/jp/importexport をご覧ください。
[27] http://en.wikipedia.org/wiki/Sneakernet
[28] http://www.verisign.com/ssl/
[29] http://www.entrust.net/ssl-products.htm

### 保存データを保護

クラウドに機密データを保管することに懸念を抱いている場合は、クラウドにアップロードする前にデータ（個別ファイル）を暗号化すべきです。たとえば、データを Amazon S3 のオブジェクトとして保管する前にオープンソース[30]または商業用の[31]PGP ベースツールを使用してデータを暗号化し、ダウンロード後に暗号を解除します。この方法は、 Protected Health Information (PHI)を保管する必要がある HIPPA 対応のアプリケーション[8]を構築する際には、優れたプラクティスとなります。

Amazon EC2 では、ファイルの暗号化はオペレーティングシステムにより異なります。Windows を実行している Amazon EC2 インスタンスは、ビルトイン方式の Encrypting File System (EFS)機能 [16]を使用できます。この機能は、ファイルとフォルダの暗号化と暗号解除を自動的に処理し、ユーザーに対してトランスペアレントにプロセスを処理します [19]。ただし、この名称にもかかわらず、EFS はファイルシステム全体を暗号化するわけではなく、個別のファイルを暗号化します。[32]ファイルシステム全体を暗号化するには、オープンソースの TrueCrypt 製品の使用を検討してください。この製品は、NTFS 形式の EBS ボリュームを非常にうまく統合します。Linux を実行する Amazon EC2 インスタンスは、各種アプローチ (EncFS[33]、Loop-AES[34]、 dm-crypt[35]、 TrueCrypt[36])を備えた暗号化されたファイルシステムを使用しながら、EBS ボリュームを装着できます。同様に、OpenSolaris を実行する EC2 インスタンスは、ZFS[37]暗号化サポート [20]を利用できます。どの方法を選んでも、Amazon EC2 の暗号化するファイルとボリュームは、ファイルとログデータを保護し、ユーザーとサーバー上のプロセスのみがクリアテキスト形式でデータを参照できますが、サーバーの外からは、暗号化されたデータしか参照できません。

どのオペレーティングシステムやテクノロジーを選んでも、保存中のデータの暗号化は、データの暗号化に使用するキーの管理という課題を生じさせます。キーをなくしたら、データも永遠に失われ、キーが破損したら、データも危険な状態になります。従って、選んだ製品のキー管理機能について研究し、キーをなくすリスクを最小限に抑える手順を確立します。

データを盗聴から守る他に、災害から守る方法も考慮します。 Amazon EBS ボリュームのスナップショットを定期的に取り、高い耐久性と可用性を確保します。スナップショットはインクリメンタル(差分保存)で設計されており、Amazon S3 に保管され（別の地理上のロケーション）、数回クリックするかコマンドライン呼び出しを行うだけで、復元されます。

### AWS 資格情報の保護

AWS は、AWS アクセスキーと X.509 認証の 2 種類のセキュリティ資格情報を提供します。AWS アクセスキーは、キー ID とシークレットアクセスキーの 2 つの部分に分かれます。REST または Query API を使用する場合、シークレットアクセスキーを使用して、認証リクエストに含める署名を計算します。転送中の改ざんを回避するには、すべてのリクエストを HTTPS 上で送信します。

Amazon Machine Image (AMI) が他の AWS ウェブサービスとコミュニケーションが必要な場合に（たとえば、Amazon SQS キューをポーリングするため、 Amazon S3 からオブジェクトを読み取るため）、よく犯してしまう設計ミスの一つが

---

[30] http://www.gnupg.org
[31] http://www.pgp.com/
[32] http://www.truecrypt.org/
[33] http://www.arg0.net/encfs
[34] http://loop-aes.sourceforge.net/loop-AES.README
[35] http://www.saout.de/misc/dm-crypt/
[36] http://www.truecrypt.org/
[37] http://www.opensolaris.org/os/community/zfs/

AMI に AWS セキュリティ証明書を同梱してしまう事です。セキュリティ証明書を埋め込んでしまうのではなく、起動時に引数として渡し、送信前に暗号化するべきです。[17]。

シークレットアクセスキーが信用できなくなった場合、新しいアクセスキーID にローテーションし、[38]新しいキーを取得します。優れたプラクティスとしては、キーローテーションメカニズムをアプリケーションアーキテクチャに統合することで、ローテーションを定期的に、または必要時に（不満を持った従業員が退職するとき等に）行えるようにし、信用できなくなったキーが続けて使用されないようにします。

また、X.509 認証を、幾つかの AWS サービスに使用して認証を行えます。認証ファイルには、 base64-encoded DER 認証機関のパブリックキーが含まれています。別のファイルには、対応する base64-encoded PKCS#8 プライベートキーが含まれます。

AWS は、マルチファクター認証[39]をサポートし、aws.amazon.com と AWS Management Console[40]上のアカウント情報を追加で保護します。

### アプリケーションの安全性を確保

Amazon EC2 は、1つまたは複数のセキュリティグループ[41]で保護され、どの受信ネットワークトラフィックをインスタンスに配信するかを指定するルールがあります。 TCP と UDP ポート、ICMP タイプとコード、ソースアドレスを指定できます。セキュリティグループは、実行中のインスタンスに対して、基本的なファイアウォールのような保護を提供します。たとえば、ウェブアプリケーションに属するインスタンスは、図 4 のセキュリティグループ設定を備えることが可能です。

受信トラフィックを制限する別の方法は、インスタンスにソフトウェアベースのファイアウォールを設定することです。[42]Windows のインスタンスは、ビルトイン方式のファイアウォールを使用できます。Linux のインスタンスは、netfilter[43] も iptables も使用できます。

ソフトウェアのエラーは幾度となく発見されるものであり、その度に修正するためのパッチをあてることが必要となっています。アプリケーションのセキュリティを最大化するには、以下の基本ガイドラインを確実に順守してください。

- ベンダーのウェブサイトからパッチを定期的にダウンロードし、AMI をアップデートします。
- 新 AMI からインスタンスを再度配置し、アプリケーションをテストしてパッチにより何も破壊されていないことを確認します。最新の AMI がすべてのインスタンスに配置されていることを確認します。
- テストスクリプトに投資して、セキュリティチェックを定期的に実行し、プロセスを自動化します。
- サードパーティソフトウェアが最もセキュアな設定に構成されていることを確認します。
- 絶対に必要な場合以外は、プロセスを root または管理者ログインとして実行しないでください。

クラウド時代以前のすべての標準セキュリティプラクティス、たとえば、優れたコーディングプラクティスの採用や、機密データの隔離は変わることなく適用可能であり、むしろ実践すべきです。

---

[38] http://aws.amazon.com/about-aws/whats-new/2009/08/31/seamlessly-rotate-your-access-credentials/
[39] マルチファクター認証の詳細については、 http://aws.amazon.com/mfa/ をご覧ください。
[40] AWS Management Console http://aws.amazon.com/console/
[41] セキュリティグループの詳細については、 http://docs.amazonwebservices.com/AWSEC2/2009-07-15/UserGuide/index.html?using-network-security.html をご覧ください。
[42] http://technet.microsoft.com/en-us/library/cc779199(WS.10).aspx, March 2003
[43] http://www.netfilter.org/

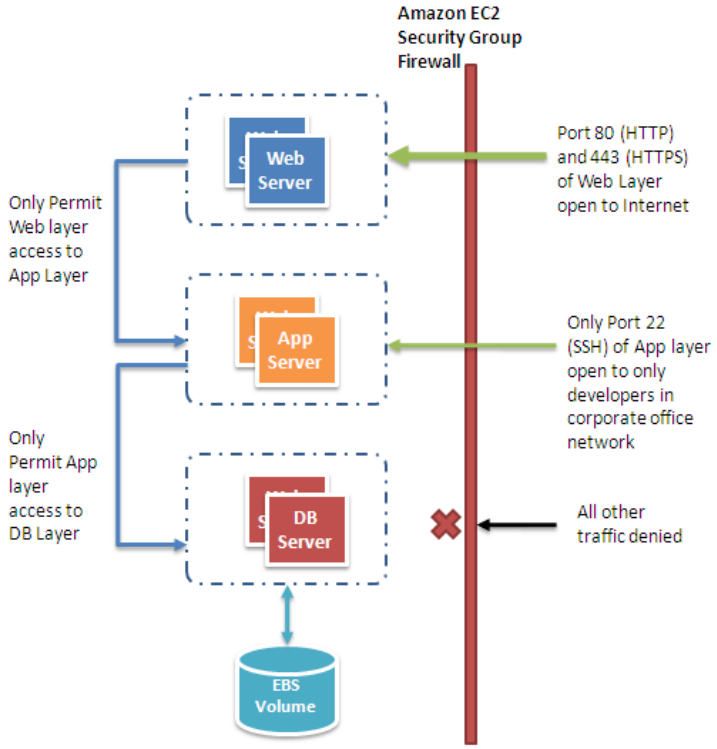


図 4: EC2 Security Groups を使用してウェブアプリケーションのセキュリティを確保

振り返ってみると、クラウドは物理的セキュリティの複雑さからユーザーを解放しており、ユーザーにツールと機能を通じた柔軟なコントロールを与えるので、アプリケーションのセキュリティを確保しやすくなっているのです。

# さらなるベストプラクティスを求めて

アプリケーションが物理的ハードウェアを気にしなくてもよくなる日は、そう遠くありません。マイクロウェーブのプラグインに電気の知識が必要ないように、アプリケーションの実行に必要な電源を取得するために、だれでも、ユーティリティプログラムのように、アプリケーションをクラウドにプラグインすることができます。アーキテクトは、物理的なサーバーではなく、抽象的なコンピュート、ストレージ、ネットワークのリソースを管理するようになります。アプリケーションは、基盤となる物理的なハードウェアが故障したり、取り外されたり、または交換されていても、機能し続けます。アプリケーションは、リソースを瞬時にかつ自動的に配置し、変動する需要パターンに自ら適応するため、常に最高レベルの利用率を達成します。スケーラビリティ、セキュリティ、高可用性、耐故障性、テスト可能性、弾力性は、アプリケーションアーキテクチャにおいて調整可能なプロパティとなり、そのアーキテクチャが構築されるプラットフォームにおいて自動化され実現されることになるでしょう。

ただし、私達はまだそこまでたどり着いていません。今日、本書で説明したベストプラクティスを実践することにより、上記で述べたようなレベルで、クラウドを用いたアプリケーション構築が可能です。クラウドコンピューティングアーキテクチャのベストプラクティスは、今後も進化し続けるでしょう。私達は研究者として、開発者やアーキテクトがクラウドにアプリケーションを容易にプラグインできるように、クラウドの強化だけではなく、仕事を楽にするツール、テクノロジー、プロセスについても重点的に研究すべきです。

# まとめ

本書は、効率的なクラウドアプリケーションを設計するクラウドアーキテクトに対する手引きとなるガイドラインを提供しています。

故障に備えた設計、アプリケーションコンポーネントの分離、弾力性のについての理解と実践、弾力性と並列化の組み合わせ、アプリケーションアーキテクチャのあらゆる面におけるセキュリティの統合、などの概念とベストプラクティスに集中して説明しています。クラウドアーキテクトにとっては、非常にスケーラブルなクラウドアプリケーションの構築に必要な設計上の考慮事項について、理解を深めることができるでしょう。

AWS クラウドは、非常に信頼性の高い従量課金制のインフラストラクチャサービスを提供します。本書で説明した AWS を活用した戦術群は、これらのサービスを使用したクラウドアプリケーションの設計に役立ちます。一人の研究者として、こうした商用サービスを利用し、先人の仕事から学び、それを活かして構築し、進化させ、さらなるクラウドコンピューティングを発明していかれることを期待いたしております。

# 謝辞

筆者は、本書の初期の草案に対して意見を提供してくれた Jeff Barr、Steve Riley、Paul Horvath、Prashant Sridharan および Scot Marvin に深く謝意を表します。また貴重な見識を示してくれた Matt Tavis には、特に謝意を表します。彼の助力がなかったら、この記事は世に出ていなかった事でしょう。



# 参考文献

1. **Amazon S3 Team, Best Practices for using Amazon S3,** http://developer.amazonwebservices.com/connect/entry.jspa?externalID=1904, 2008-11-26
2. **Amazon S3 Team, Amazon S3 Error Best Practices,** http://docs.amazonwebservices.com/AmazonS3/latest/index.html?ErrorBestPractices.html, 2006-03-01
3. **Amazon SimpleDB Team, Query 201: Tips and Tricks for Amazon SimpleDB Query,** http://developer.amazonwebservices.com/connect/entry.jspa?externalID=1232&categoryID=176, 2008-02-07
4. **Amazon SimpleDB Team, Building for Performance and Reliability with Amazon SimpleDB,** http://developer.amazonwebservices.com/connect/entry.jspa?externalID=1394&categoryID=176, 2008-04-11
5. **Amazon SimpleDB Team, Query 101: Building Amazon SimpleDB Queries,** http://developer.amazonwebservices.com/connect/entry.jspa?externalID=1231&categoryID=176, 2008-02-07
6. **J. Varia, Cloud Architectures,** http://jineshvaria.s3.amazonaws.com/public/cloudarchitectures-varia.pdf, 2007-07-01

7. **Amazon Security Team, Overview of Security Processes,** http://awsmedia.s3.amazonaws.com/pdf/AWS_Security_Whitepaper.pdf, 2009-06-01
8. **Amazon Web Services Team, Creating HIPPA-Compliant Medical Data Applications With AWS,** http://awsmedia.s3.amazonaws.com/AWS_HIPAA_Whitepaper_Final.pdf, 2009-04-01
9. D. Obasanjo, Building Scalable Databases: Pros and Cons of Various Database Sharding Schemes, http://www.25hoursaday.com/weblog/2009/01/16/BuildingScalableDatabasesProsAndConsOfVariousDatabaseShardingSchemes.aspx, 2009-01-16
10. D. Pritchett, Shard Lessons, http://www.addsimplicity.com/adding_simplicity_an_engi/2008/08/shard-lessons.html, 2008-08-24
11. J. Hamilton, On Designing and Deploying Internet-Scale Services, 2007, 21st Large Installation System Administration conference (LISA '07), http://mvdirona.com/jrh/talksAndPapers/JamesRH_Lisa.pdf
12. J. Dean and S. Ghemawat, MapReduce: Simplified data processing on large clusters2004-12-01, In Proc. of the 6th OSDI, http://labs.google.com/papers/mapreduce-osdi04.pdf
13. T. Schlossnagle, Scalable Internet Architectures, Sams Publishing , 2006-07-31,
14. M. Lurie, The Federation: Database Interoperability, http://www.ibm.com/developerworks/data/library/techarticle/0304lurie/0304lurie.html, 2003-04-23
15. E. Hammond, Escaping Restrictive/Untrusted Networks with OpenVPN on EC2, http://alestic.com/2009/05/openvpn-ec2, 2009-05-02
16. R. Bragg, The Encrypting File System, http://technet.microsoft.com/en-us/library/cc700811.aspx, 2009
17. S. Swidler, How to keep your AWS credentials on an EC2 instance securely, http://clouddevelopertips.blogspot.com/2009/08/how-to-keep-your-aws-credentials-on-ec2.html, 2009-08-31
18. **Amazon SQS Team, Building Scalable, Reliable Amazon EC2 Applications with Amazon SQS,** http://sqs-public-images.s3.amazonaws.com/Building_Scalabale_EC2_applications_with_SQS2.pdf, 2008
19. Microsoft Support Team, Best Practices For Encrypting File System (Windows), http://support.microsoft.com/kb/223316, 2009
20. Solaris Security Team, ZFS Encryption Project (OpenSolaris), http://www.opensolaris.org/os/project/zfs-crypto/, 2009-05-01
21. **Amazon RDS Team, Amazon RDS Multi-AZ Deployments,** http://docs.amazonwebservices.com/AmazonRDS/latest/DeveloperGuide/Concepts.DBInstance.html#Concepts.MultiAZ . 2010 年 5 月 15 日
22. **Amazon SimpleDB Team, Amazon SimpleDB Consistency Enhancements** http://developer.amazonwebservices.com/connect/entry.jspa?externalID=3572 2010-02-24