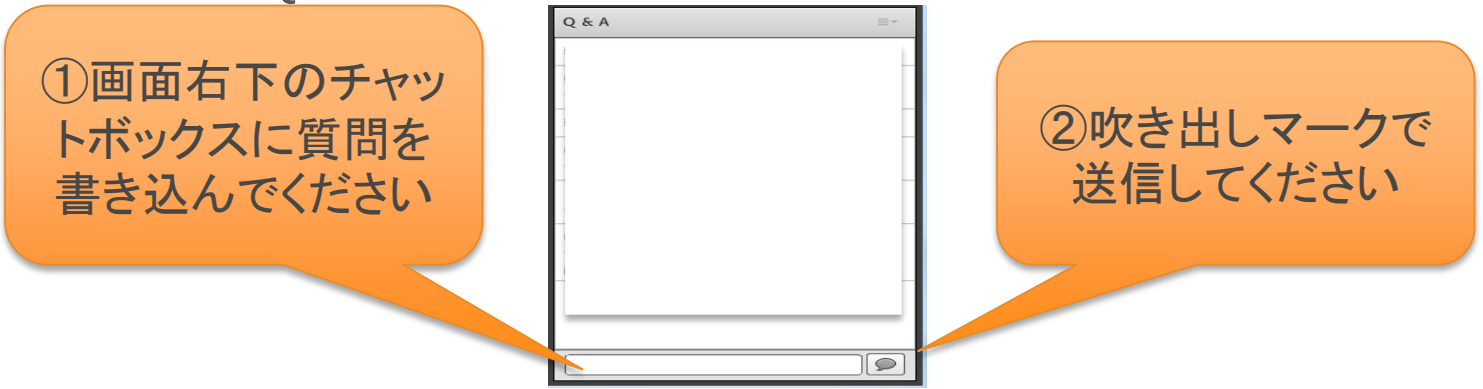
【 AWS 初心者向け Webinar 】
.NET 開発者のための AWS 超入門

2016/01/12
アマゾン ウェブ サービス ジャパン 株式会社
ソリューション アーキテクト
福井　厚

amazon web services

ご質問を受け付け致します！

- 質問を投げることができます！
  - Adobe Connect のチャット機能を使って、質問を書き込んでください。（書き込んだ質問は、主催者にしか見えません）
  - 最後の Q&A の時間で可能な限り回答させていただきます。



AWS 初心者向け Webinar のご紹介

- AWS についてこれから学ぶ方むけの
  ソリューションカットの Webinar です

- 過去の Webinar 資料
  - AWS クラウドサービス活用資料集ページにて公開
    http://aws.amazon.com/jp/aws-jp-introduction/

- イベントの告知
  - 国内のイベント・セミナースケジュールページにて告知
    http://aws.amazon.com/jp/about-aws/events/
    （オンラインセミナー枠）

自己紹介

名前：　福井 厚 （ふくい あつし）

fatushi@amazon.co.jp

所属：　アマゾンウェブサービスジャパン株式会社
技術本部 エンタープライズ ソリューション部
ソリューション アーキテクト

経歴：　PCメーカーサポート、ソフトハウス、SIベンダー（国産、外資）、開発系コンサルティング ファームを経て、2015年7月より現職。
2008年8月、Microsoft Certified Architect for Solutions Certification (MCA) に認定される。マイクロソフトMVPアワード受賞歴11回（2015年7月にMVP 終了）。C#を愛し、.NETが大好きなエンジニアとして .NET開発者向けにAWSを普及する活動を実施中。

好きなAWSサービス： AWS IoT、AWS Tools for .NET、AWS Lambda

amazon web services

アジェンダ

Agenda

- AWS概要
- AWS SDK for .NET と AWS Tools for Visual Studio
- AWS SDK による自動化
- AWS サービスの利用
- まとめ

アジェンダ

Agenda

- AWS概要
- AWS SDK for .NET / AWS Tools for Visual Studio
- AWS SDK による自動化
- AWS サービスの利用
- まとめ

AWSの特徴

初期投資が不要

実際の使用分のみ支払い

低額な変動価格

セルフサービスなインフラ

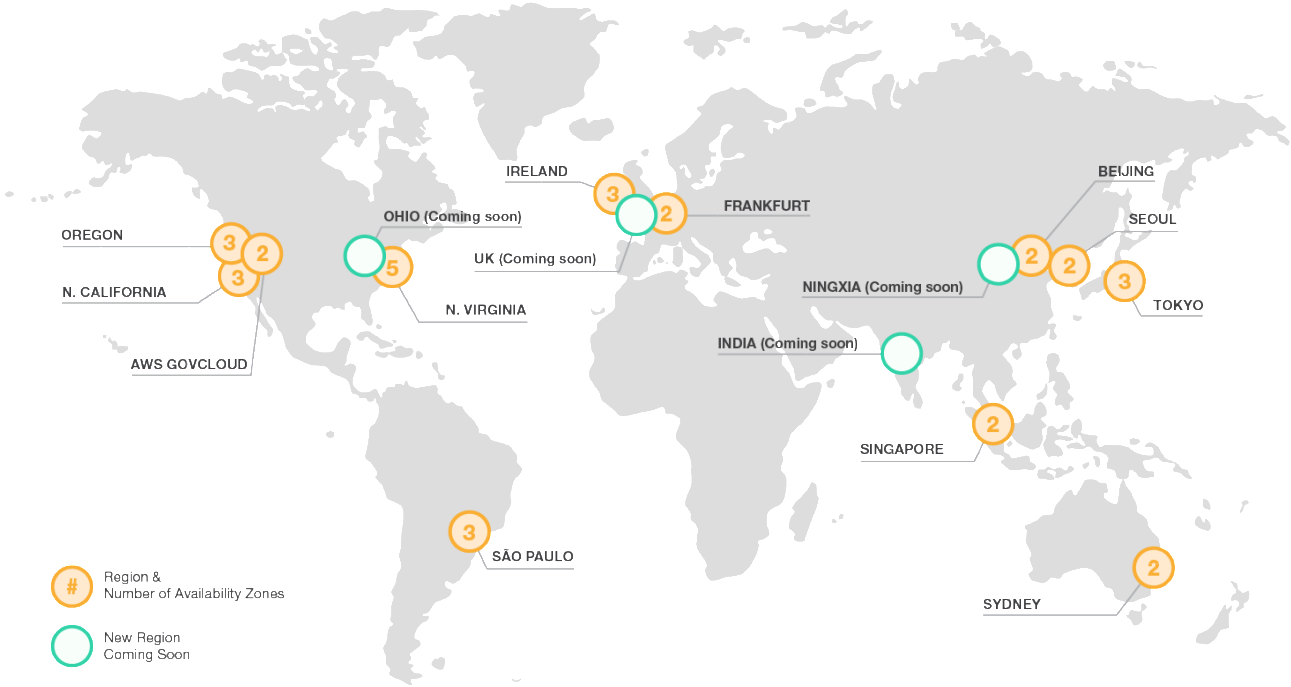
スケールアップ、ダウンが容易

市場投入と俊敏性の改善

世界中に広がるAWSの拠点

12 のリージョン

1. US EAST (Virginia)
2. US WEST （N. California)
3. US WEST 2 (Oregon)
4. EU WEST (Ireland)
5. JAPAN (Tokyo)
6. South America (Sao Paulo)
7. ASP 1 (Singapore)
8. ASP 2 (Sydney)
9. GovCloud
10. BJS 1 (Beijing China) limited preview
11. EU (Frankfurt)
12. Korea（SEOUL)

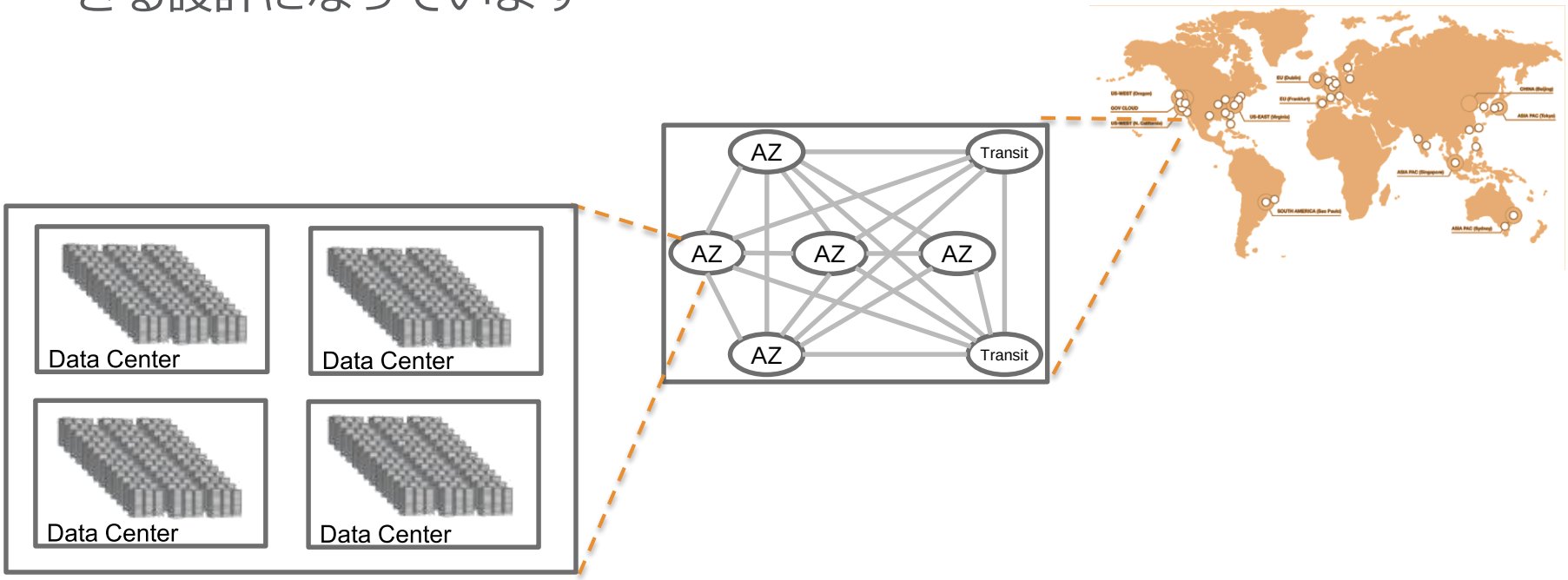


32 のアベイラビリティ・ゾーン
55 のエッジロケーション

https://aws.amazon.com/jp/about-aws/global-infrastructure/

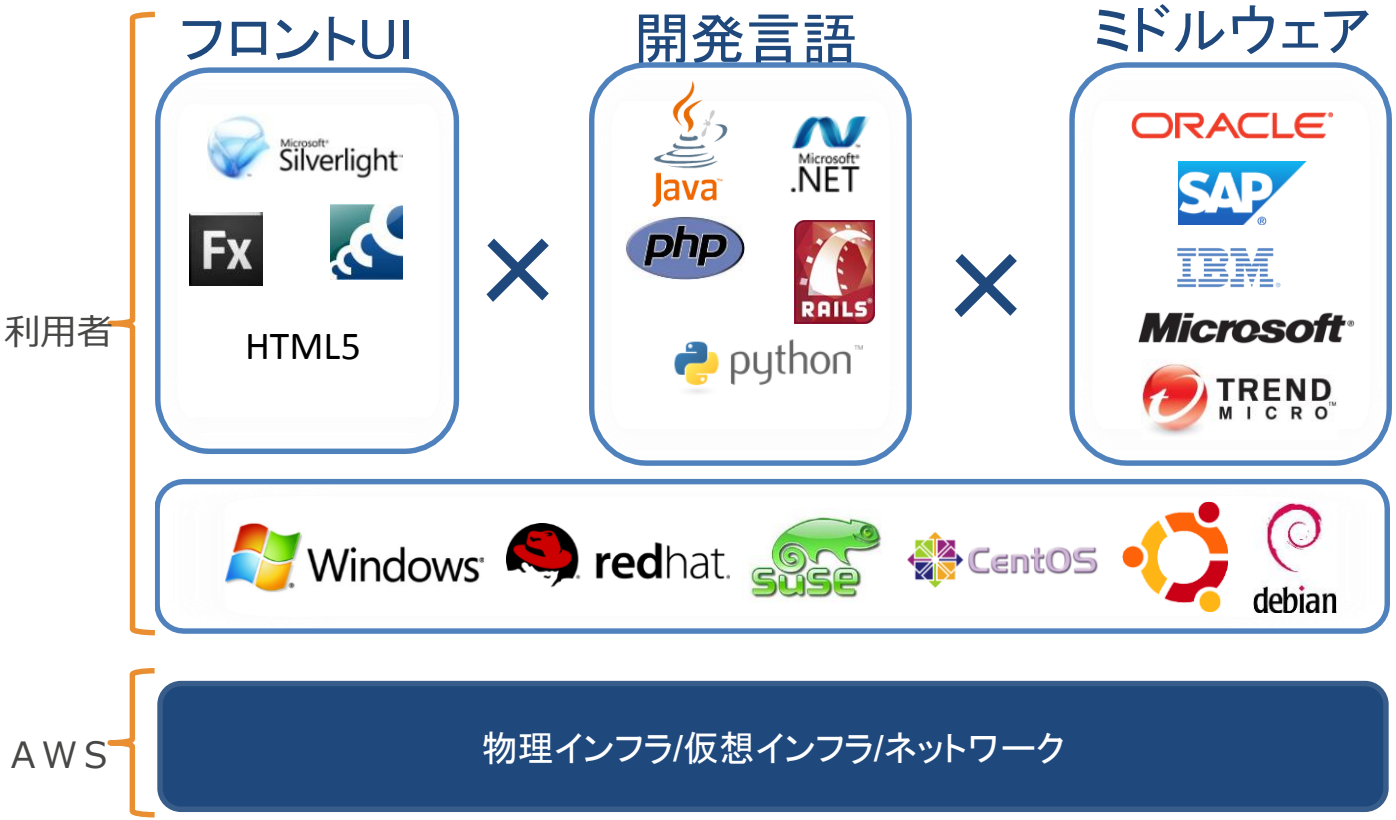
リージョンとアベイラビリティゾーン（AZ）

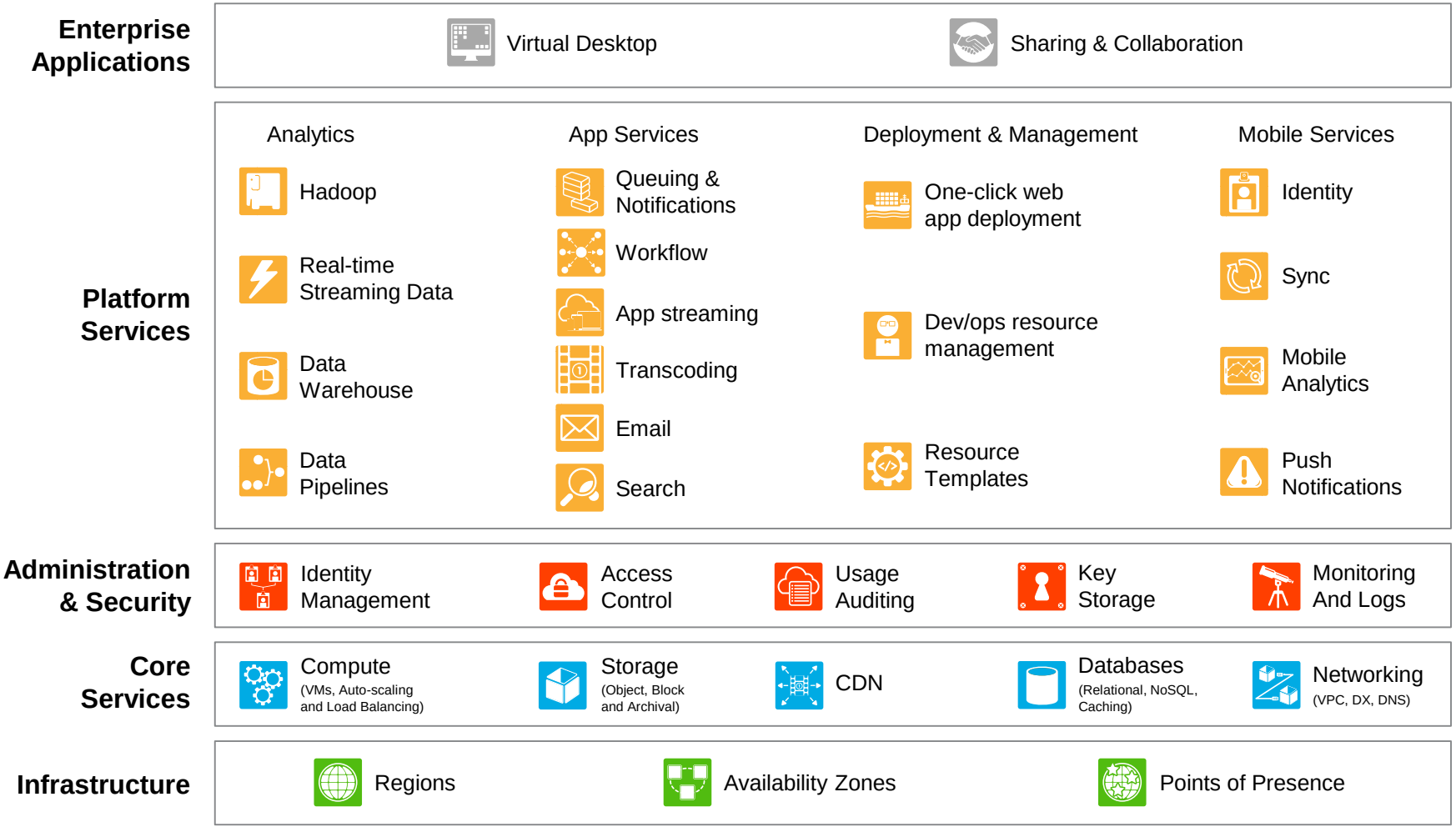
- リージョンは複数のアベイラビリティゾーン（AZ）で構成されています
- AZは、**複数のデータセンターによって構成され、高い耐障害性を提供**できる設計になっています



amazon web services

AWSは最も汎用性の高いクラウドの一つ

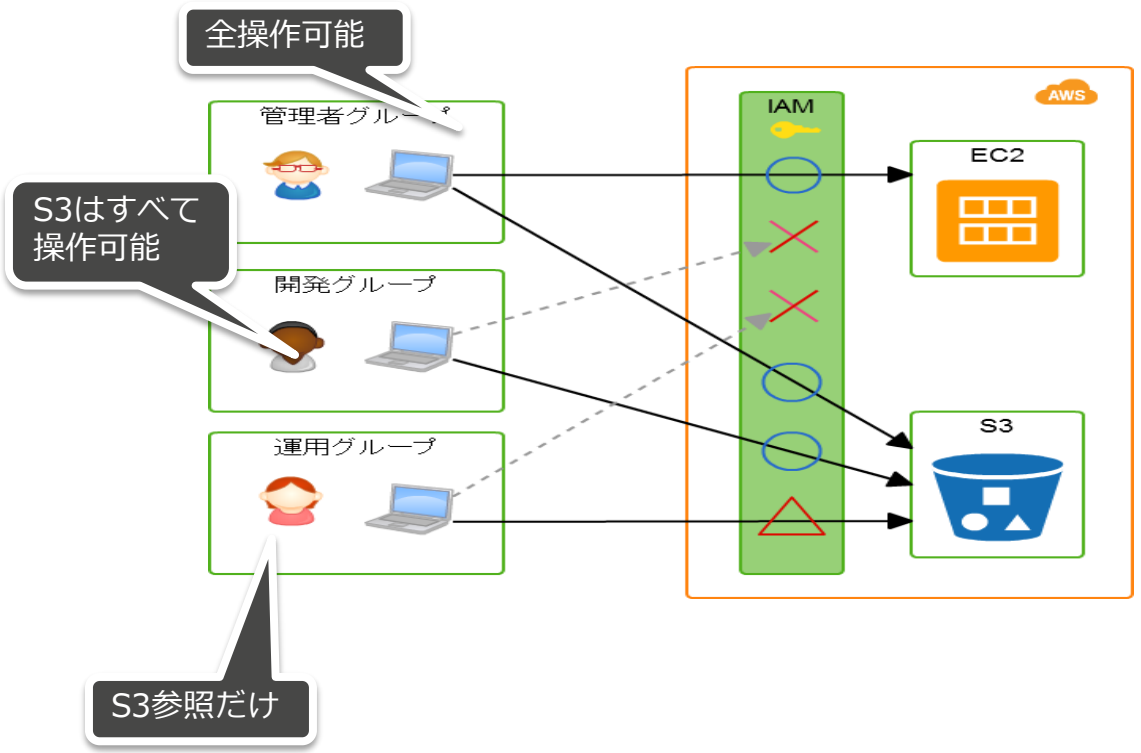



Enterprise Applications
Virtual Desktop
Sharing & Collaboration
Platform Services
Analytics
Hadoop
Real-time Streaming Data
Data Warehouse
Data Pipelines
App Services
Queuing & Notifications
Workflow
App streaming
Transcoding
Email
Search
Deployment & Management
One-click web app deployment
Dev/ops resource management
Resource Templates
Mobile Services
Identity
Sync
Mobile Analytics
Push Notifications
Administration & Security
Identity Management
Access Control
Usage Auditing
Key Storage
Monitoring And Logs
Core Services
Compute
(VMs, Auto-scaling and Load Balancing)
Storage
(Object, Block and Archival)
CDN
Databases
(Relational, NoSQL, Caching)
Networking
(VPC, DX, DNS)
Infrastructure
Regions
Availability Zones
Points of Presence

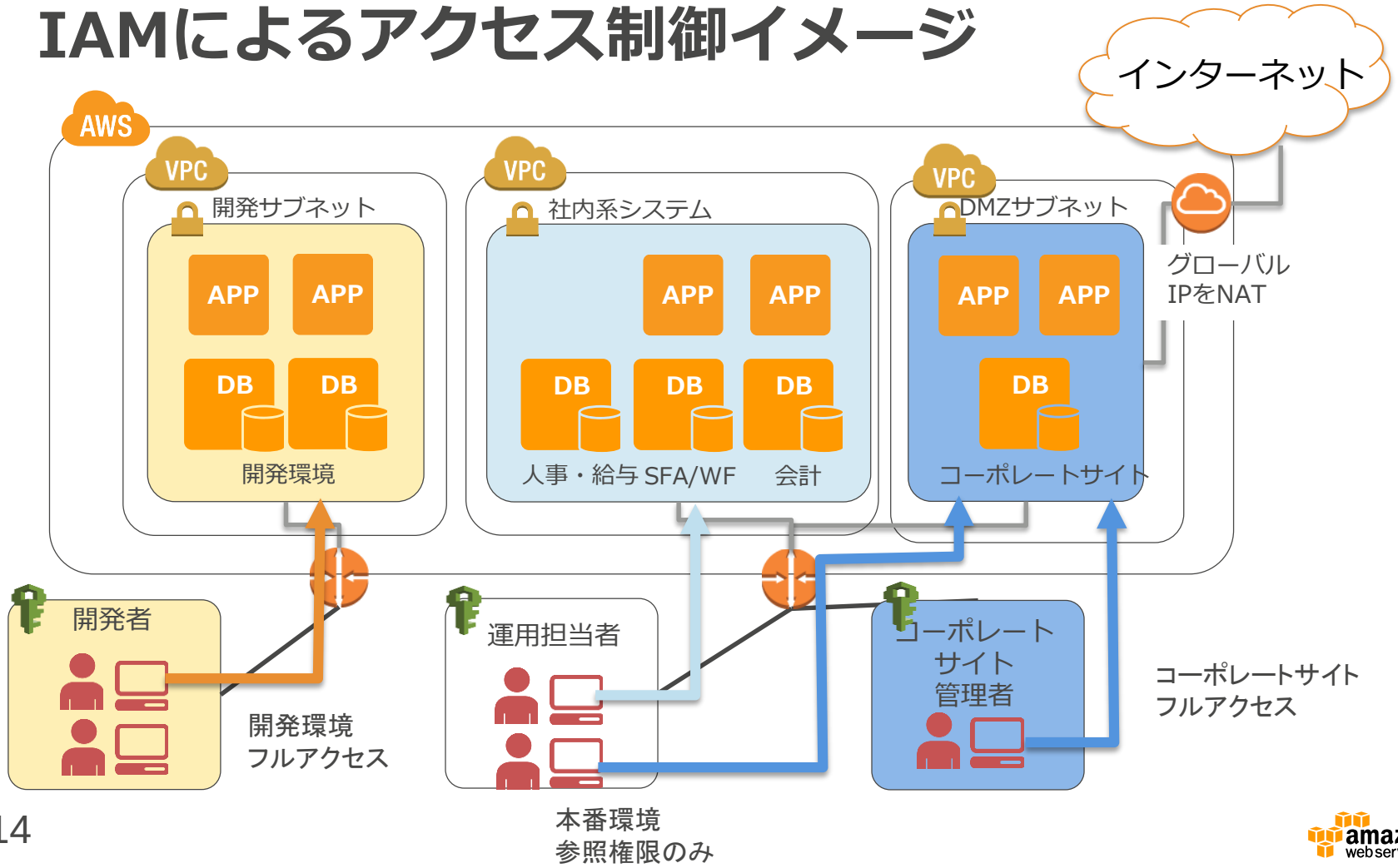

Enterprise Applications
Virtual Desktop
Sharing & Collaboration
Analytics
Hadoop
Real-time
Data Pipelines
App Services
Queuing & Notifications
Workflow
Search
Deployment & Management
One-click web app deployment
Resource Templates
Mobile Services
Identity
Sync
Push Notifications
サーバー、ストレージ、DBから、アプリケーションまで
50を超えるクラウドサービスを提供
Administration & Security
Identity Management
Access Control
Usage Auditing
Key Storage
Monitoring And Logs
Core Services
Compute
(VMs, Auto-scaling and Load Balancing)
Storage
(Object, Block and Archival)
CDN
Databases
(Relational, NoSQL, Caching)
Networking
(VPC, DX, DNS)
Infrastructure
Regions
Availability Zones
Points of Presence

AWS IAM(Identity and Access Management)

- AWS利用者の認証とリソースへのきめ細かなアクセス制御を提供
  - AWS操作のためのグループ・ユーザー・ロールの作成が可能
  - グループ、ユーザーごとに、実行出来る操作を規定できる
  - ユーザーごとに認証情報の設定が可能
  - AWS サービス API や指定リソースへのアクセスをコントロール
  - 多要素認証機能の提供
- Active Directoryなどの認証機構との連携も可能

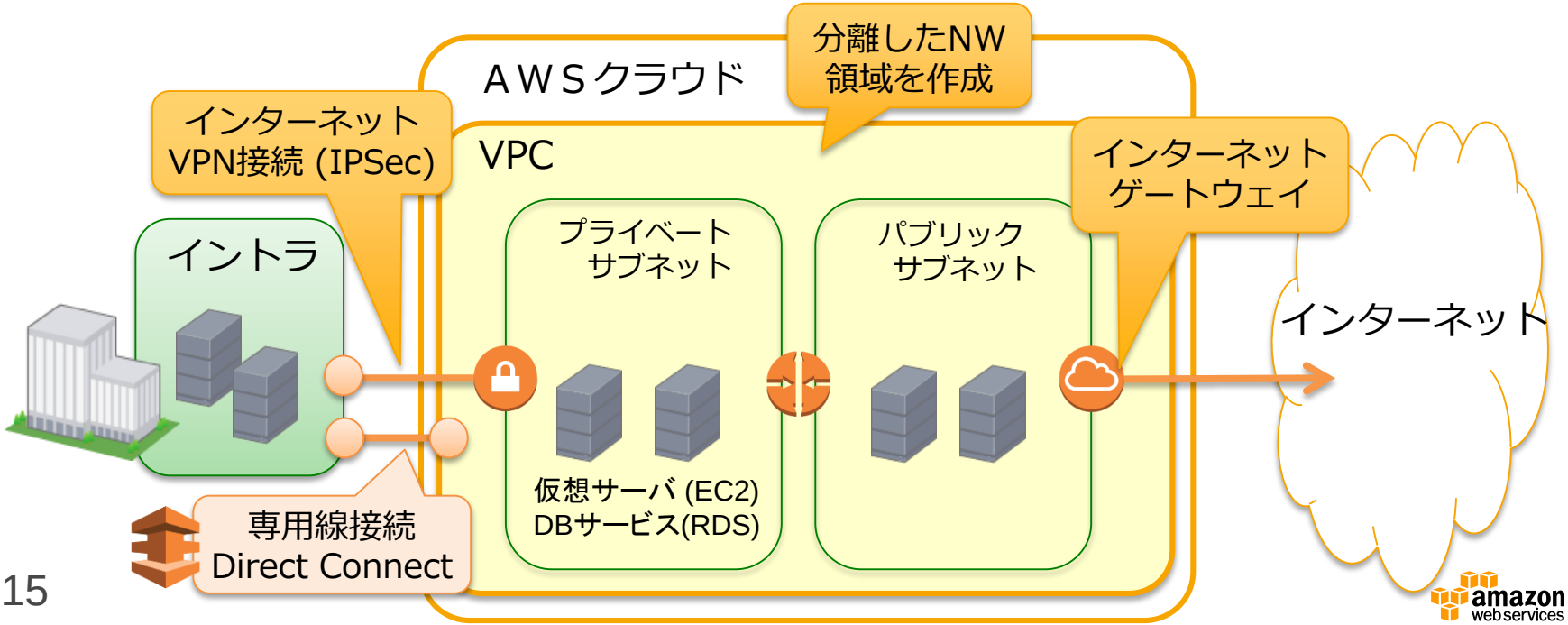


IAMによるアクセス制御イメージ



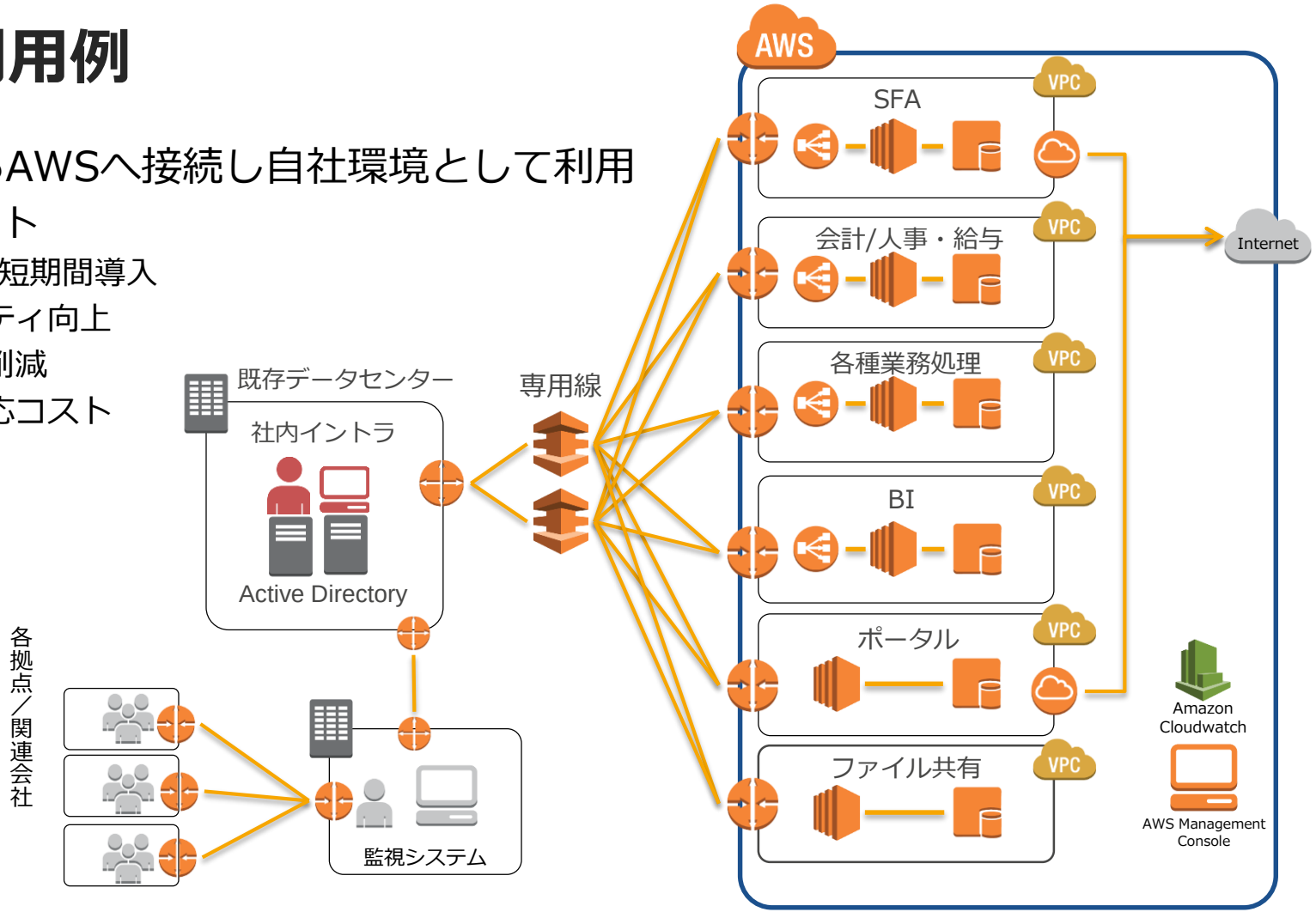
Amazon VPC (Virtual Private Cloud)

- クラウド内にプライベートネットワークを構築
- 既存データセンターの延長としてAWSを利用
- 認証の連携など、オンプレミスのシステムとの連携が容易



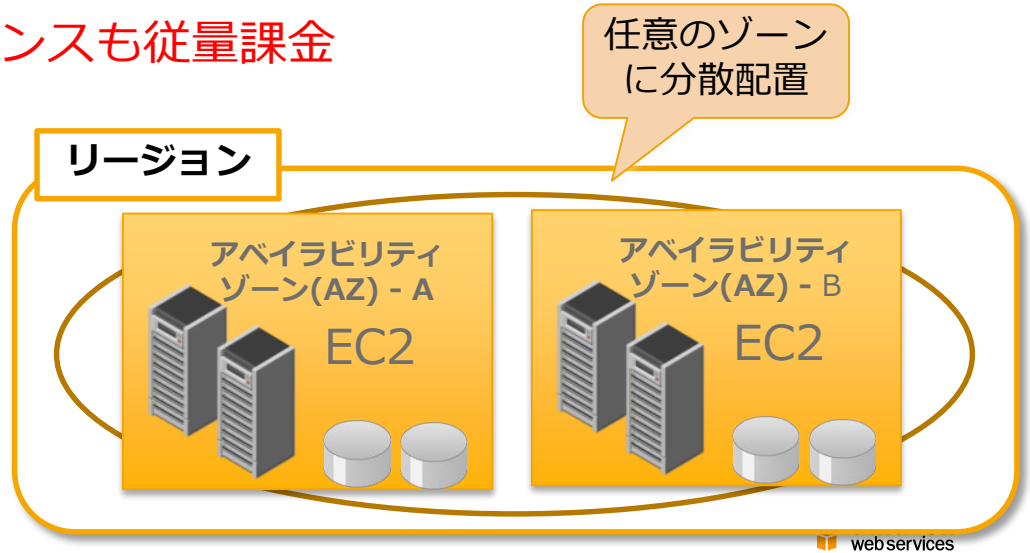
VPCの利用例

- 既存DCからAWSへ接続し自社環境として利用
- 様々なメリット
  - 低コスト/短期間導入
  - セキュリティ向上
  - 運用工数削減
  - 老朽化対応コスト



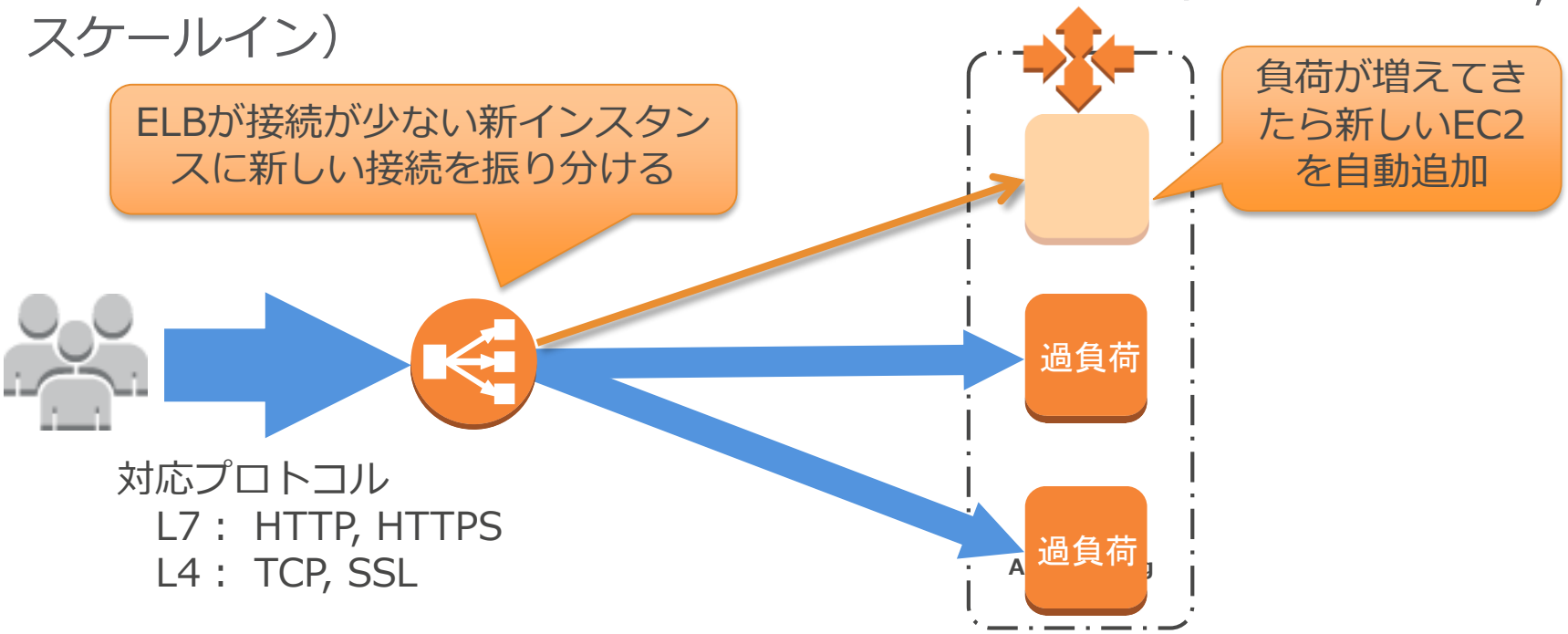
Amazon EC2 (Elastic Compute Cloud)

- 数分で起動可能な仮想サーバ
- 1時間ごとの従量課金で利用可能
- スケールアップ/ダウン、スケールアウト/インが可能
- Windows, LinuxなどのOSが利用可能
  - WindowsやRedHatのライセンスも従量課金
- OSより上はお客様の自由。
  お手持ちのソフトを
  そのまま利用可能



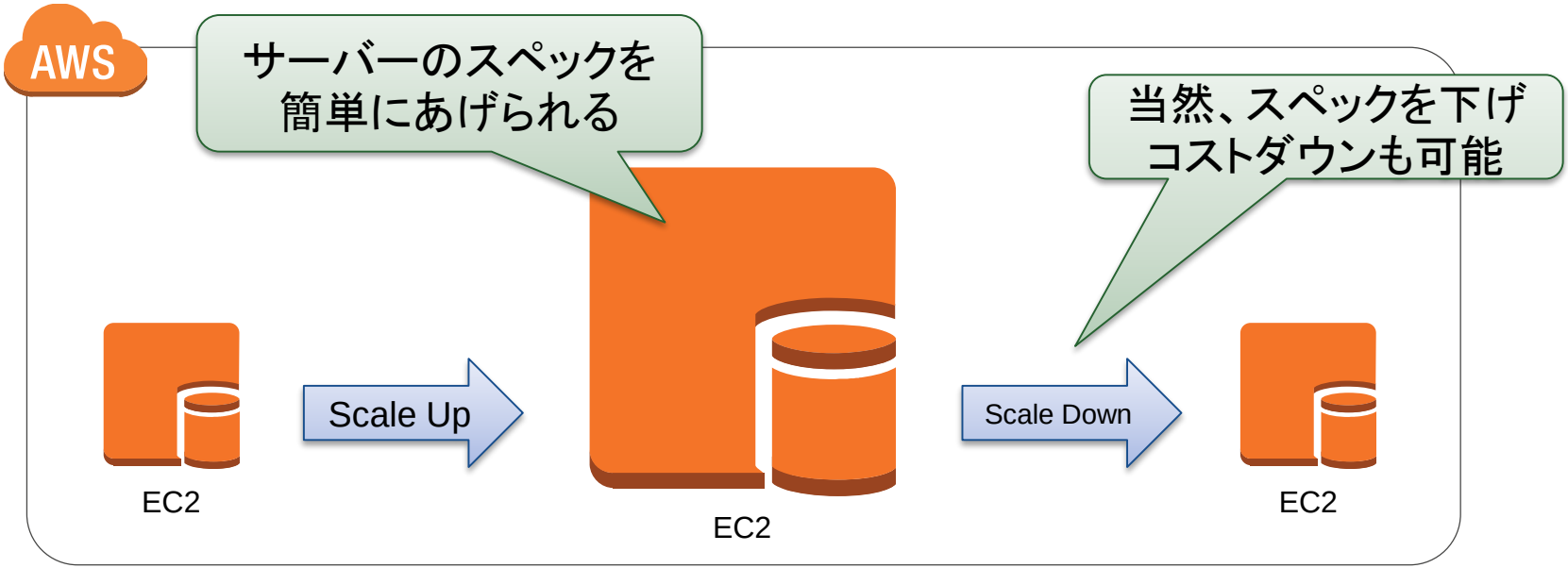
スケールアウト可能なシステム構成

Amazon Elastic Load Balancer (ELB)とAuto Scaling機能を組み合わせることで、EC2インスタンス数を自動増減可能（スケールアウト，スケールイン）

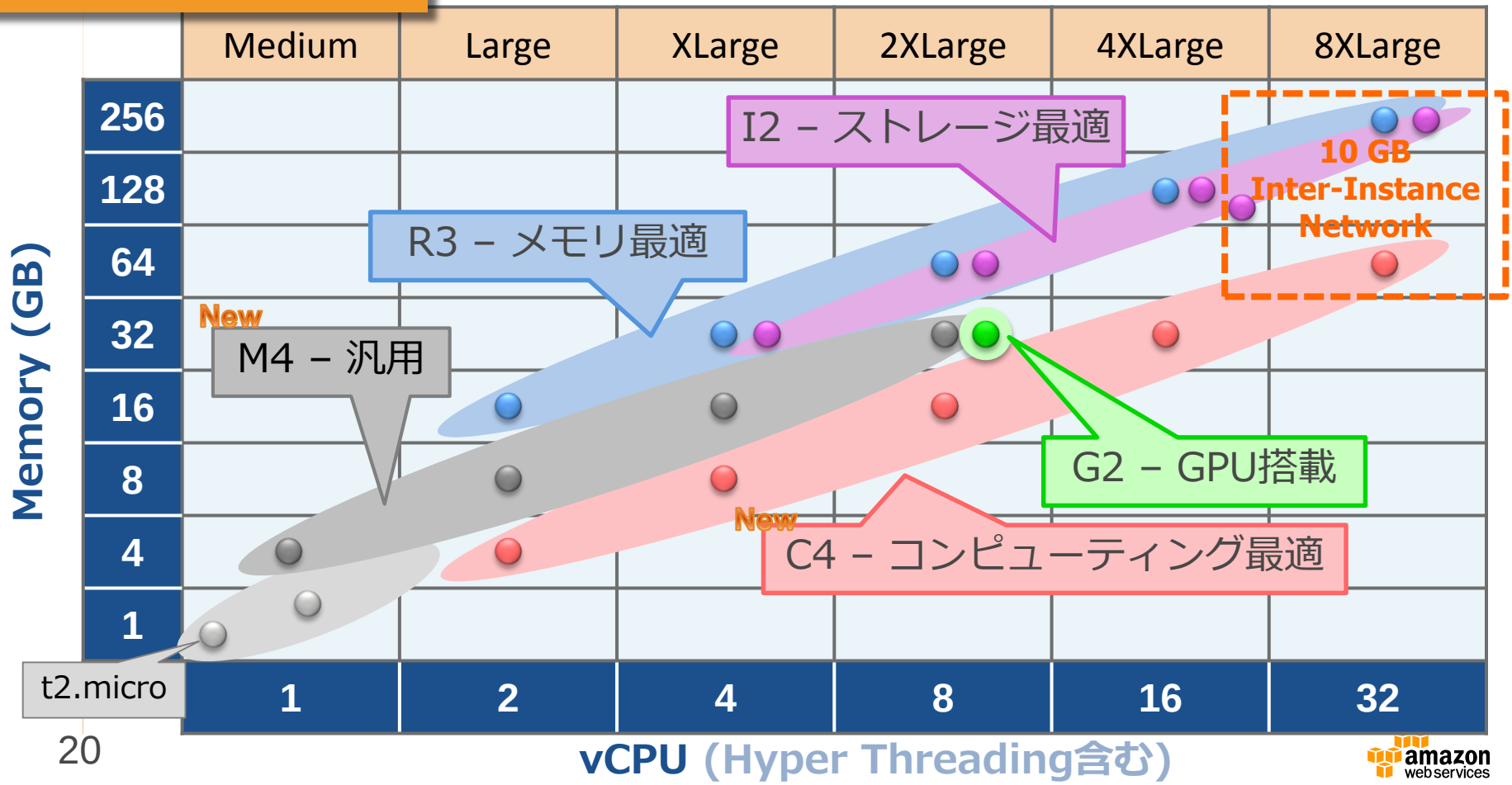


柔軟なキャパシティ変更で環境の変化に対応

- EC2のスペック変更は容易

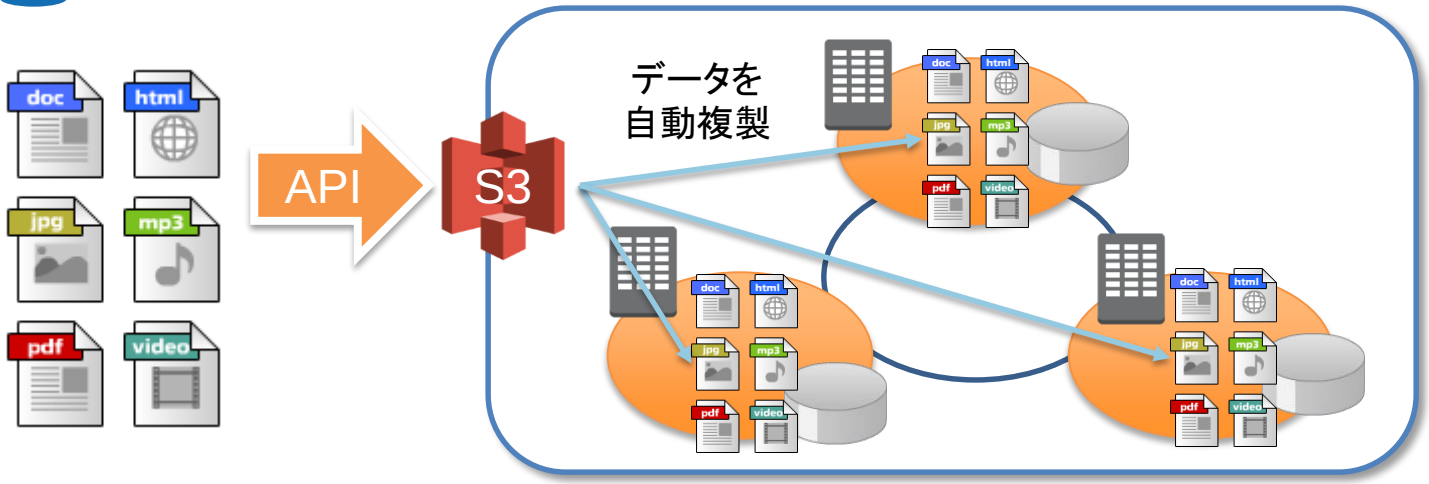


最新インスタンス



amazon web services

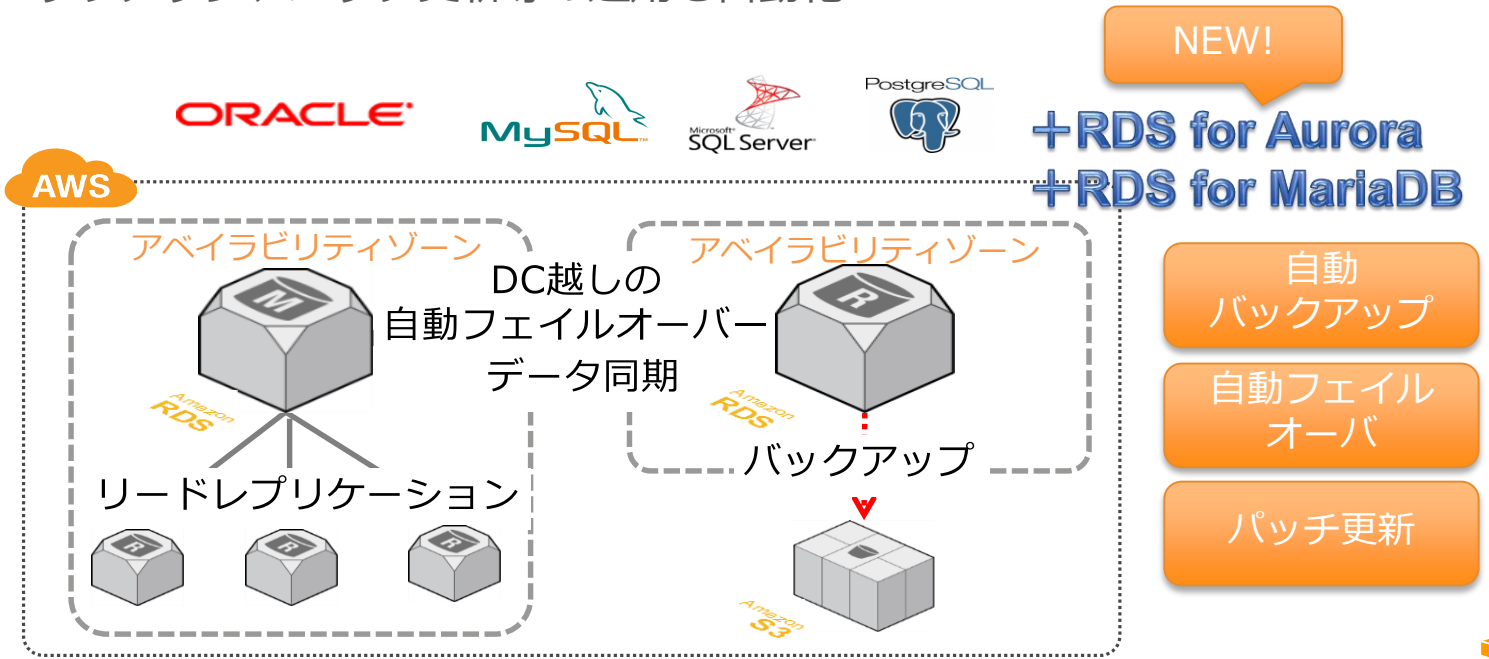
Amazon S3(Simple Storage Service)



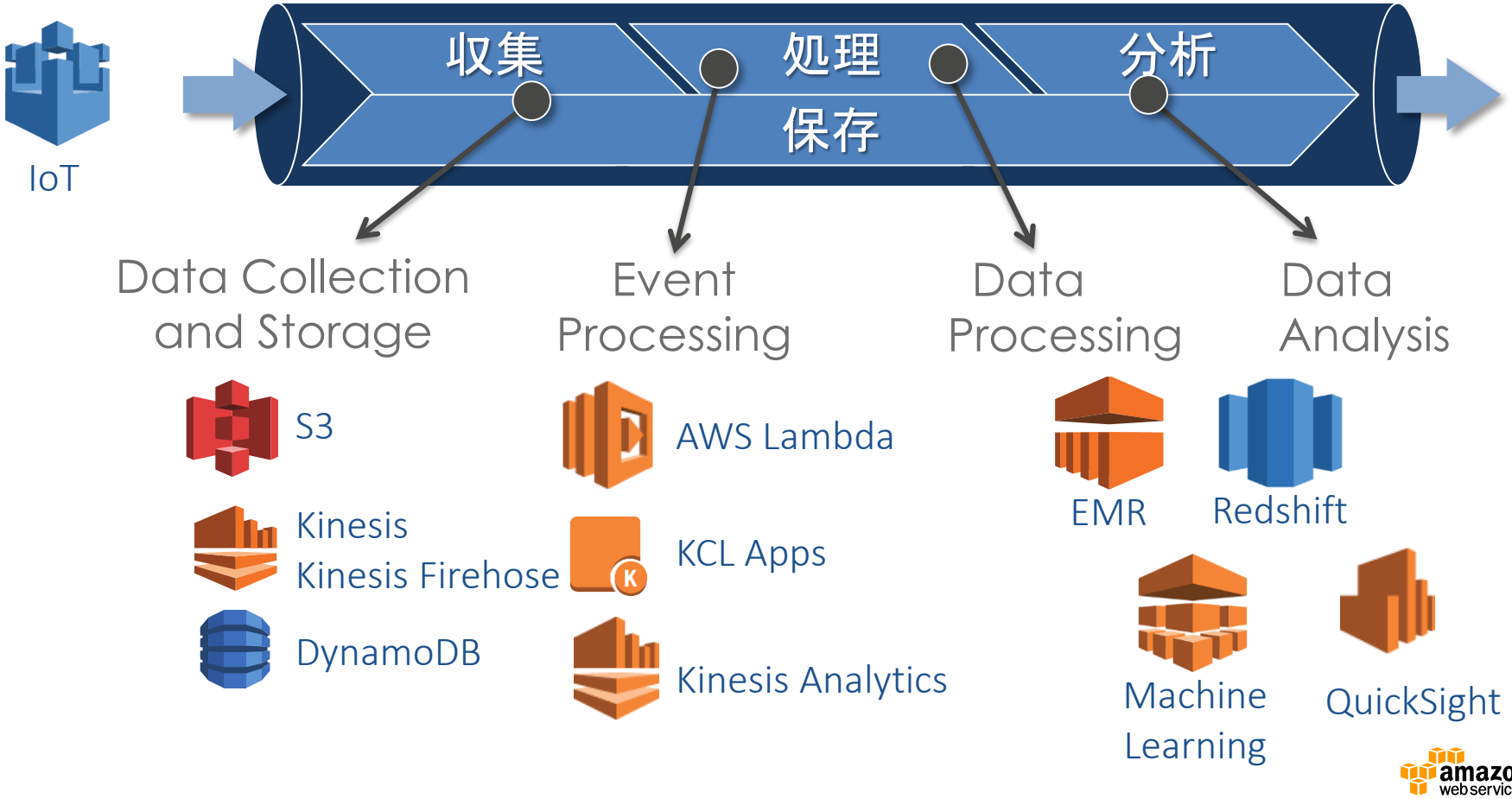
- データは2か所で障害が発生しても復旧可能
- 設計上のデータ耐久性は、99.999999999%
- **1GByteあたり月額約4円**　容量は無制限
- 現在世界中で２兆以上のファイルを格納

Amazon RDS(Relational Database Service)

- マネージドRDBMSサービス
  - MySQL, Oracle, SQLServer, PostgreSQL
- DCレベルでのMaster-Slave構成を数クリックで構築可能
- バックアップ、パッチ更新等の運用も自動化



AWS サービスの利用例

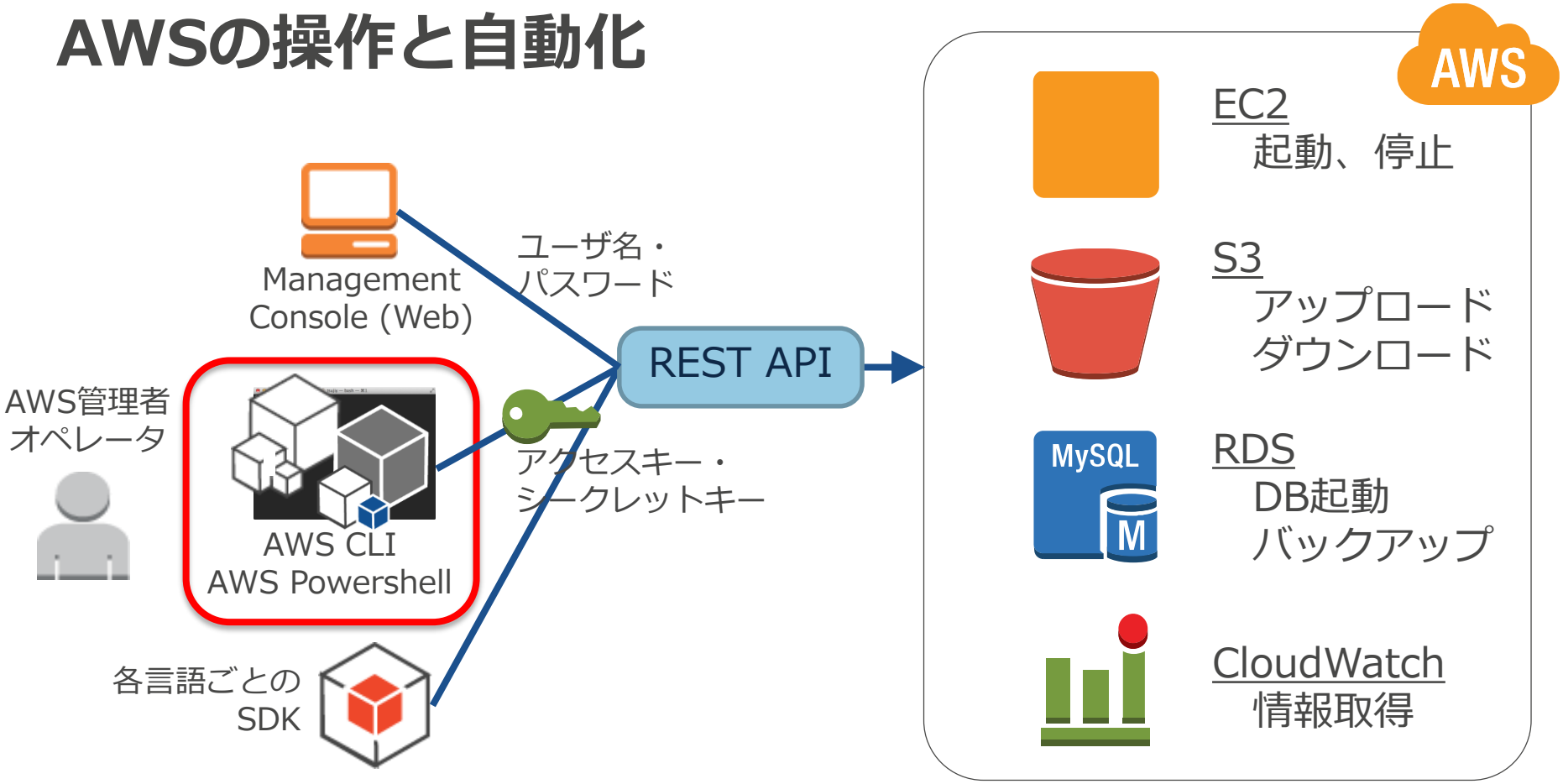


アジェンダ

Agenda

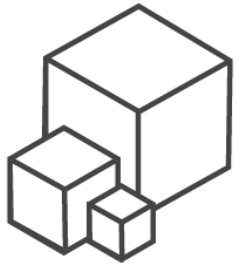
- AWS概要
- AWS SDK for .NET / AWS Tools for Visual Studio
- AWS SDK による自動化
- AWS サービスの利用
- まとめ

AWSの操作と自動化



AWS Command Line Interface (CLI)

- “aws”という名前の単一コマンドでAWSサービスを操作可能
- プラットフォームや開発言語などが限定されない
  Windows, Linux, Mac, Unixなど
- S3用にはsyncなどの便利な機能あり



AWS Tools for Windows PowerShell

- “AWSPowerShell”モジュール内のコマンドレットから、ほとんどのAWSサービスを操作可能
- PowerShellの強力なシェル機能が利用できる

AWS SDK

サーバやバッチ処理ワーカーなどで動かすコードで利用


Java


NodeJS


.NET


PHP


Python


Ruby

エンドユーザの端末あるいはサービスのクライアント側で動くコードで利用


Android


iOS


Javascript
in
Browser

AWS SDKの用途

- AWSリソースのコントロール
  - インフラ構築/運用の自動化をプログラムで実装可能
    - EC2 インスタンスの起動や Cloud Formation の実行
- AWSサービスの利用
  - アプリケーションの機能として AWS のサービスを利用する
    - S3へのデータ保存、DynamoDB や SQS へのデータ送信など

AWS SDK 利用のユースケース

- 複雑な条件分岐が必要な場合
  - データベースのデータに応じて処理条件が変更されるなど
- 複雑なバッチ処理
  - ジョブ コントローラーで実行されるバッチ処理をプログラミングする必要がある場合
- AWSサービスを利用した独自API を公開
- 様々な UI でサービスを提供
  - スマートフォンを操作してEC2インスタンスを起動するなど

AWS SDK for .NET のご紹介

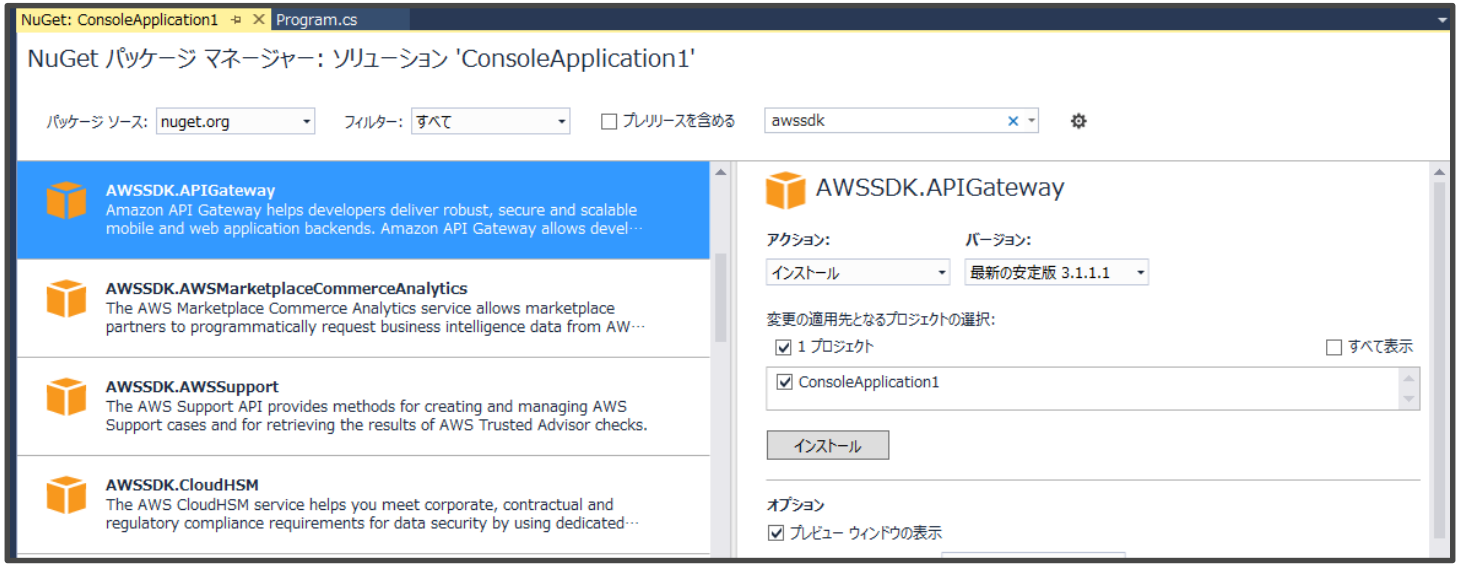
- 以下のコンポーネントをバンドル
  - 現在のバージョンの AWS SDK for .NET （2016/1/3現在は、場
    バージョン 3.1.36 ）
  - 以前のバージョンの AWS SDK for .NET
  - AWS SDK for .NET のサンプル コード
  - AWS Tools for Windows PowerShell
  - AWS Tools for Visual Studio

.NET 開発環境のインストール

- Microsoft.NET Framework 3.5 以上（必須）
- Microsoft Visual Studio 2010 以降（必須）
- AWS SDK for .NET （必須）
- AWS Tools for Visual Studio プラグイン（推奨）

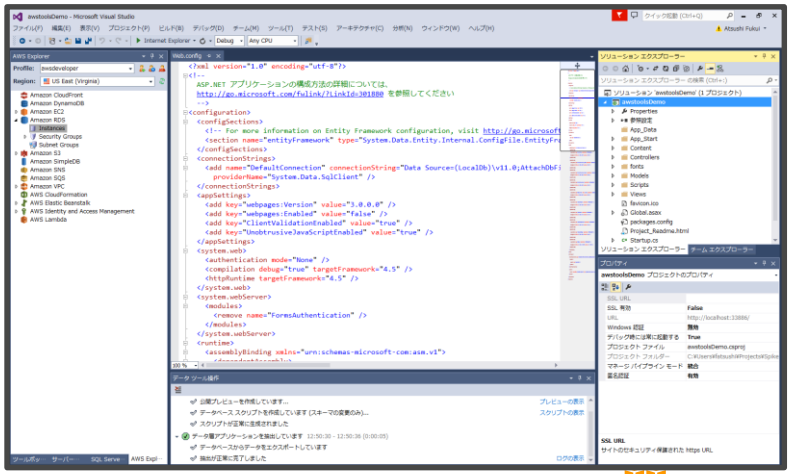
NuGet Package Manager の利用

- NuGet Package Manager で AWS SDK の各アセンブリをプロジェクトに個別にインストールすることも可能



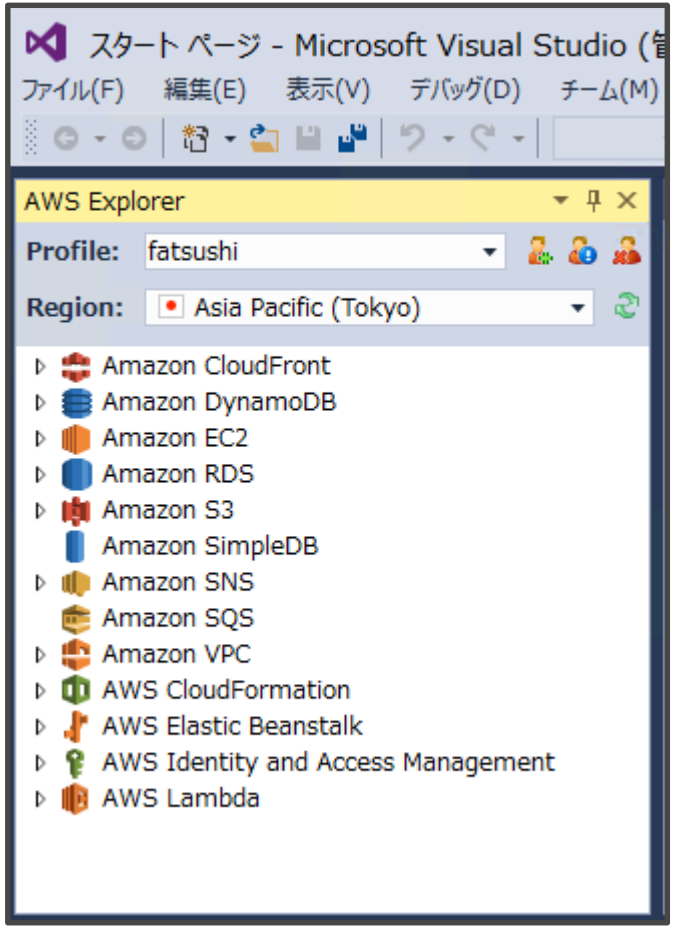
AWS Tools for Visual Studio のご紹介

- Visual Studio プラグイン
  - Visual Studio 2010 以降のバージョンに対応
- AWS Explorer ウィンドウ
- Elastic Beanstalk を利用してWeb アプリケーションを展開可能



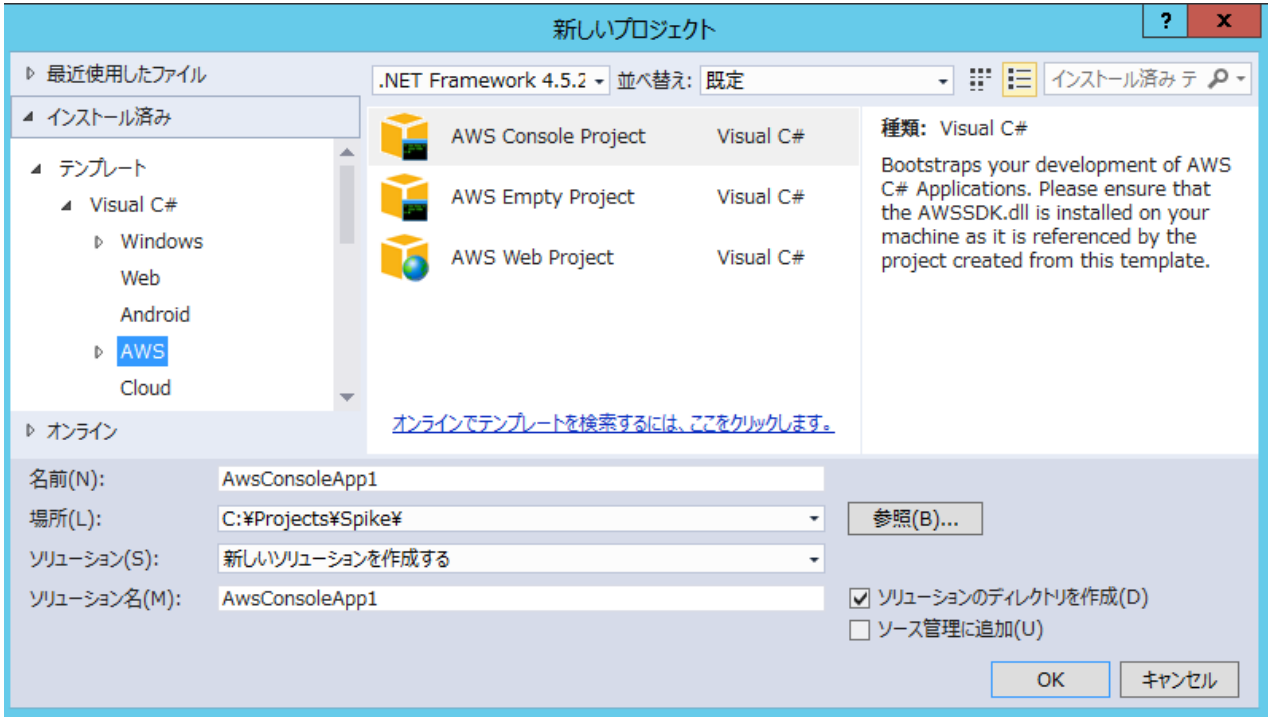
AWS Explorer ウィンドウ

- EC2 インスタンスなど AWS の各サービスを Visual Studio 内から閲覧、編集可能
- AWS CloudFormation や AWS Elastic Beanstalk による展開をウィザード画面で実行可能



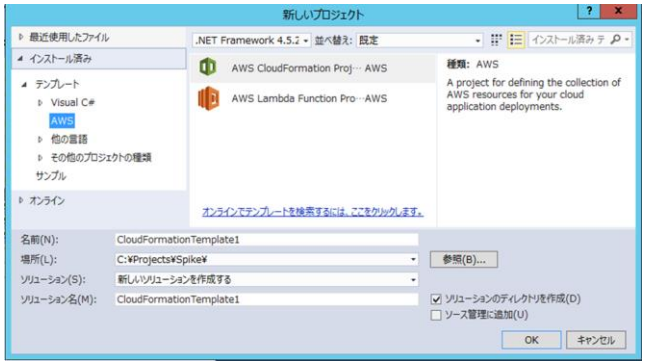
amazon web services

Visual Studio に追加されるテンプレート

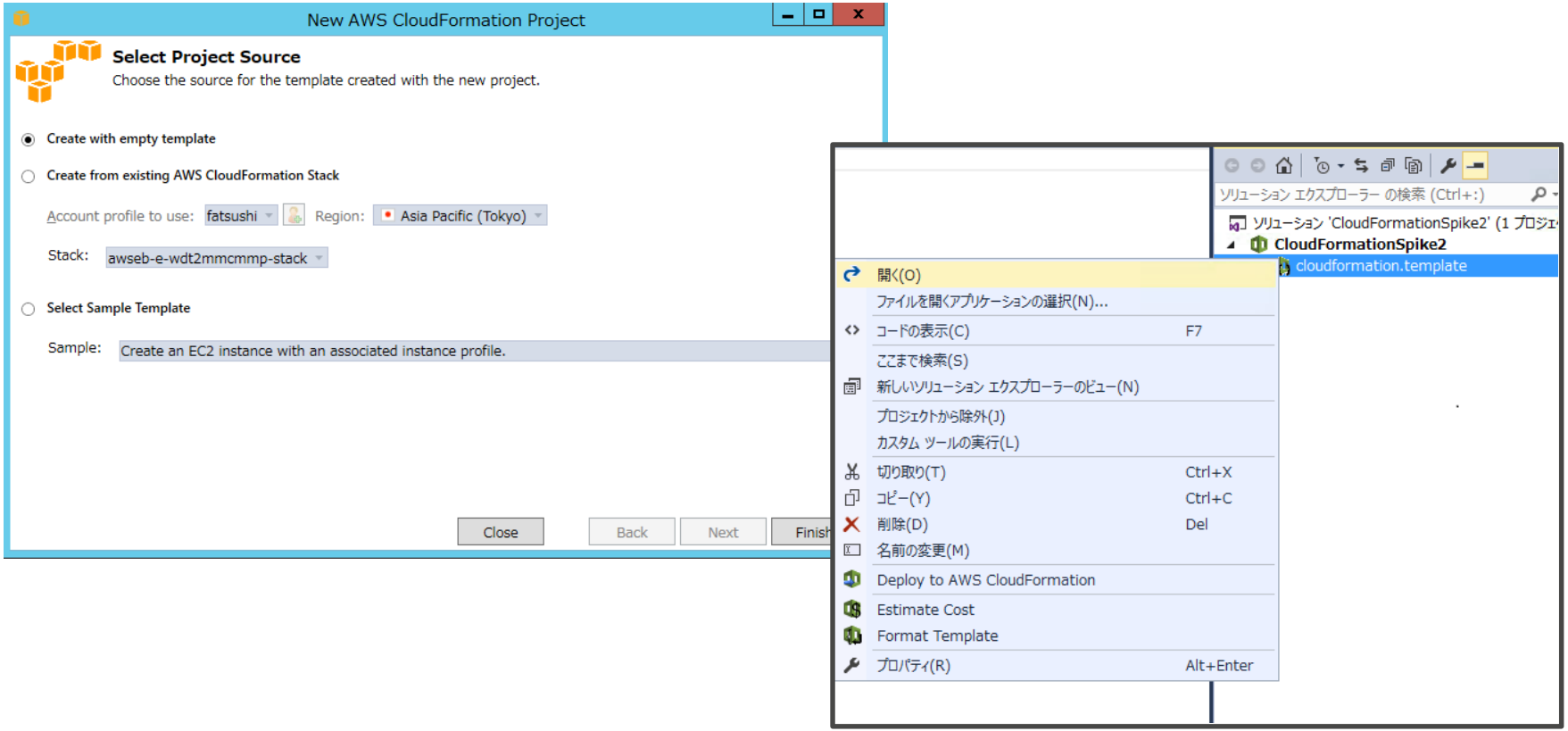


Visual Studio に追加されるテンプレート

- AWS CloudFormation プロジェクト
- AWS Lambda Function プロジェクト
  - Node.js Tools for Visual Studio を事前にインストールする
  - https://github.com/Microsoft/nodejstools/releases/tag/v1.1



CloudFormation テンプレート



AWS アカウントとクレデンシャルの作成

- AWS へアクセスするために AWS アカウントを作成
  - http://aws.amazon.com/ ページを開いてアカウントを作成
- アカウント作成後、コンソールにログイン
- AWS サービスへプログラムからアクセスするためには、クレデンシャル（アクセスキーとシークレットキー）を作成する必要がある
  - AWS ルートアカウントでもクレデンシャルは作成できるが、専用の IAM User を作成することを推奨
  - 作成したIAM User に適切な IAM Role を割り当て
- AWS Tools for Visual Studio に作成したクレデンシャルを指定する

AWS SDK for .NET アプリケーションの構成ファイル

```
<configuration>
  <configSections>
    <section name="aws" type="Amazon.AWSSection, AWSSDK.Core"/>
  </configSections>
  <aws region="us-west-2">
    <logging logTo="Log4Net"/>
  </aws>
</configuration>
```

アジェンダ

Agenda

- AWS概要
- AWS SDK for .NET / AWS Tools for Visual Studio
- AWS SDK による自動化
- AWS サービスの利用
- まとめ

AWS SDK を利用した自動化の例
EC2 インスタンスの操作

EC2 インスタンスの起動

```
List<string> groupIds = new List<string>() { securityGroup.GroupId };
//リクエスト オブジェクトの組み立て
RunInstancesRequest runInstanceRequest = new RunInstancesRequest(){
    ImageId = amiId,
    InstanceType = instanceType,
    MinCount = minCount,
    MaxCount = maxCount,
    KeyName = keypairName,
    SecurityGroupIds = groupIds
};
//EC2インスタンスを起動
RunInstancesResponse runInstanceResponse = ec2Client.RunInstances(runInstanceRequest);
//起動したインスタンス情報を返す
return runInstanceResponse.Reservation.Instances;
```

条件を指定してEC2 インスタンス情報を取得

```
//検索用のフィルタ リストを作成
List<Amazon.EC2.Model.Filter> filters = new List<Amazon.EC2.Model.Filter>();
//インスタンス タイプを指定
Amazon.EC2.Model.Filter filter = new Amazon.EC2.Model.Filter();
filter.Name = "instance-type";
filter.Values = new List<string>() { InstanceType.T2Micro };
filters.Add(filter);

//イメージIDを指定
filter = new Amazon.EC2.Model.Filter();
filter.Name = "image-id";
filter.Values = new List<string>() { "ami-383c1956" };
filters.Add(filter);
```

条件を指定してEC2 インスタンス情報を取得

```
//リクエスト オブジェクトの作成
DescribeInstancesRequest descInstRequest = new DescribeInstancesRequest();
descInstRequest.Filters.AddRange(filters);
//EC2インスタンス情報の取得
DescribeInstancesResponse descInstResponse =
                              ec2Client.DescribeInstances(descInstRequest);
List<Instance> instances = new List<Instance>();
foreach (var reservation in descInstResponse.Reservations)
{
    foreach (var instance in reservation.Instances)
    {
        instances.Add(instance);
    }
}
//マッチしたインスタンス情報を返す
return instances;
```

EC2 インスタンスの削除

```
//インスタンス情報からインスタンスIDを取得
List<string> instanceIds = new List<string>();
foreach (var instance in instances)
{
    instanceIds.Add(instance.InstanceId);
}
//リクエスト オブジェクトの作成
TerminateInstancesRequest termInstRequest = new TerminateInstancesRequest()
{
    InstanceIds = instanceIds
};
//インスタンスを削除
TerminateInstancesResponse termInstResponse =
                            ec2Client.TerminateInstances(termInstRequest);
//削除したインスタンス情報を返す
return termInstResponse.TerminatingInstances;
```

アジェンダ

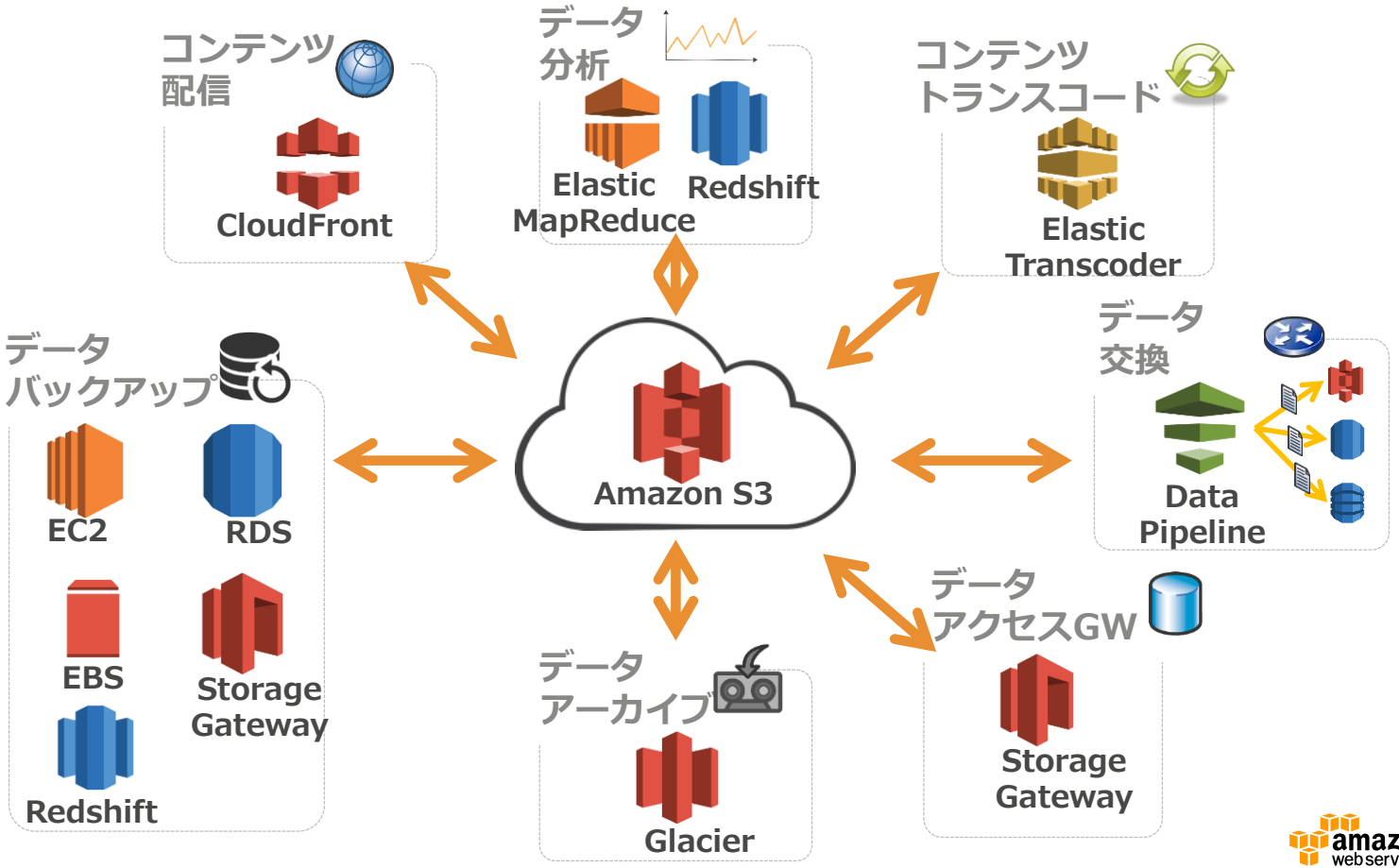
Agenda

- AWS概要
- AWS SDK for .NET / AWS Tools for Visual Studio
- AWS SDK による自動化
- AWS サービスの利用
- まとめ

クラウド ネイティブなサービスの利用

- .NET開発者にとってのメリット
  - RDS for SQL Server を利用することで数クリックでデータベースを構築し利用可能、かつ運用の手間も省ける。
  - DynamoDB をスケーラブルで高速、安価なNoSQLデータベースとして利用可能、データ容量も無制限。
  - S3 を安価なデータの保存先として利用、S3から他のAWSサービスと容易に連携可能
  - Amazon Machine Learning や Amazon Elastic Transcorder などクラウドのパワーを有効活用可能

Amazon S3をデータの集積ポイントに



amazon web services

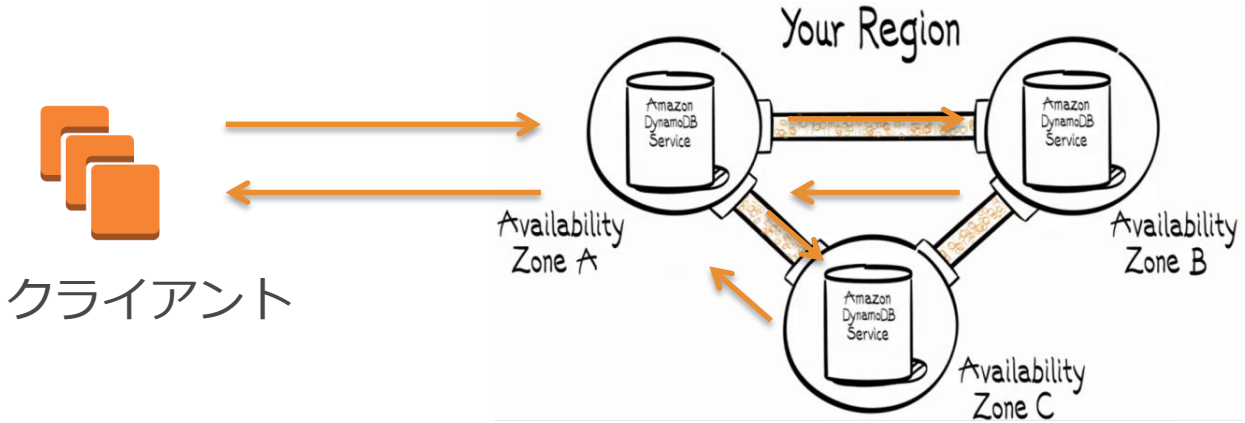
AWS サービスの利用例
Amazon DynamoDB の利用

Amazon DynamoDB とは

- 完全マネージド型の NoSQL データベースサービス
- ハイスケーラブル、低レイテンシー
- 高可用性– 3x レプリケーション
- シンプル且つパワフルAPI
- ストレージの容量制限がない
- 運用管理の必要なし

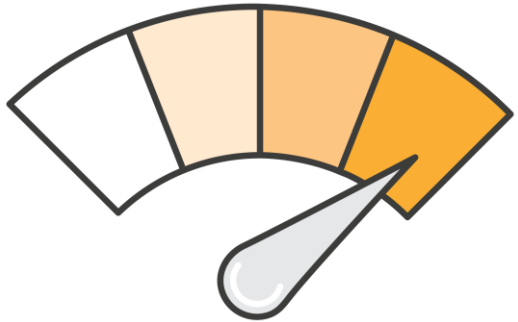
特長１：管理不要で信頼性が高い

- SPOFの存在しない構成
- データは3箇所のAZに保存されるので信頼性が高い
- ストレージは必要に応じて自動的にパーティショニングされる



特長２：プロビジョンドスループット

- テーブルごとにReadとWriteそれぞれに対し、必要な分だけのスループットキャパシティを割り当てる（＝プロビジョンする）ことができる
- 例えば下記のようにプロビジョンする
  - Read：1,000
  - Write：100
- 書き込みワークロードが上がってきたら
  - Read：500
  - Write：1,000
- この値はDB運用中にオンラインで変更可能
  - ただし、スケールダウンに関しては日に４回までしかできないので注意



特長３：ストレージの容量制限がない

- 使った分だけの従量課金制のストレージ
- データ容量の増加に応じたディスクやノードの増設作業は一切不要

DynamoDBの整合性モデル

- Write
  - 少なくとも２つのAZでの書き込み完了が確認とれた時点でAck
- Read
  - デフォルト
    - 結果整合性のある読み込み
    - 最新の書き込み結果が反映されない可能性がある
  - Consistent Readオプションを付けたリクエスト
    - GetItem、Query、Scanでは強力な整合性のある読み込みオプションが指定可能
    - Readリクエストを受け取る前までのWriteがすべて反映されたレスポンスを保証
    - Capacity Unitを２倍消費する

DynamoDBが使われているユースケース

- KVSとして
  - ユーザー情報の格納するデータベース
- 広告やゲームなどのユーザー行動履歴DBとして
  - ユーザーIDごとに複数の行動履歴を管理するためのデータベース
- モバイルアプリのバックエンドとして
  - モバイルアプリから直接参照できるデータベースとして
- 他にも
  - バッチ処理のロック管理
  - ストレージのインデックス

提供されているAPI

Table API

- CreateTable
- UpdateTable
- DeleteTable
- DescribeTable
- ListTables

- Query
- Scan

- BatchGetItem
- BatchWriteItem

- GetItem
- PutItem
- UpdateItem
- DeleteItem

DynamoDB

Stream API

- Liststreams
- DescribeStream
- GetShardIterator
- GetRecords

テーブル操作について

よく使われるオペレーションは以下のとおり

- GetItem
  - Hash keyを条件として指定し、一件のアイテムを取得
- PutItem
  - 1件のアイテムを書き込む
- Update
  - 1件のアイテムを更新
- Delete
  - 1件のアイテムを削除
- Query
  - Hash keyとRange keyの複合条件にマッチするアイテム群を取得
- BatchGet
  - 複数のプライマリキーを指定してマッチするアイテム群を取得
- Scan
  - テーブル総ナメする

※Query または Scan オペレーションで、最大 1 MB のデータを取得可能

AWS SDK for .NET によるプログラミング

- DynamoDB に対する 3 種類のAPIを提供

API	説明
低レベル API	プロトコルレベルの API であり、DynamoDB API に緊密にマッピングされます。低レベル API は、テーブルおよび項目のすべてのオペレーション（テーブルおよび項目の作成、更新、削除など）に使用できます。テーブルに対するクエリおよびスキャンも可能です。
ドキュメントモデル API	低レベル API の周囲にラッパークラスを提供し、プログラミングタスクをさらに簡素化します。Table と Document が主要なラッパークラスです。 ドキュメントモデルは、項目の作成、取得、更新、削除などのデータオペレーションに使用できます。テーブルを作成、更新、および削除するには、低レベル API を使用する必要があります。
オブジェクト永続性 API	オブジェクト永続性 API を使用すると、クライアント側クラスを DynamoDB のテーブルにマッピングできます。各オブジェクトインスタンスが、対応するテーブルの項目にマッピングされます。この API 内にある DynamoDBContext クラスによって、クライアント側オブジェクトをテーブルに保存し、項目をオブジェクトとして取得し、クエリおよびスキャンを実行するメソッドが提供されます。 オブジェクト永続性モデルは、項目の作成、取得、更新、削除などのデータオペレーションに使用できます。まず、低レベル API を使用してテーブルを作成し、次に、オブジェクト永続性モデルを使用して、クラスをテーブルにマッピングします。

低レベル API（テーブルの作成）

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
CreateTableRequest request = new CreateTableRequest{
    TableName = tableName,
    AttributeDefinitions = new List<AttributeDefinition>(){
        new AttributeDefinition{ AttributeName = "Id", AttributeType = "N"}
    },
    KeySchema = new List<KeySchemaElement>(){
        new KeySchemaElement{
            AttributeName = "Id",KeyType = "HASH"
        }
    },
    ProvisionedThroughput = new ProvisionedThroughput{
        ReadCapacityUnits = 10, WriteCapacityUnits = 5
    }
};
CreateTableResponse response = client.CreateTable(request);
```

低レベル API（テーブルの更新）

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
UpdateTableRequest request = new UpdateTableRequest()
{
    TableName = tableName,
    ProvisionedThroughput = new ProvisionedThroughput()
{
        //更新する値を指定する
        ReadCapacityUnits = 20,
        WriteCapacityUnits = 10
    }
};
UpdateTableResponse response = client.UpdateTable(request);
```

低レベル API（テーブルの削除）

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
DeleteTableRequest request = new DeleteTableRequest
{
    TableName = tableName
};
//テーブルを削除
DeleteTableResponse response = client.DeleteTable(request);
```

ドキュメント モデル API（項目の取得）

```
Table table = Table.LoadTable(client, "tableName");

GetItemOperationConfig config = new GetItemOperationConfig()
{
    AttributesToGet = new List<string>() { "Id", "Title", "Authors" },
    ConsistentRead = true
};
Document doc = table.GetItem(101, config);
```

ドキュメントモデル API（項目の削除）

```
Table table = Table.LoadTable(client, "TableName");
//ドキュメントのインスタンスを取得
Document document = table.GetItem(primaryKey);

//方法１：ドキュメントのインスタンスを使用して削除
table.DeleteItem(document);
//方法２：プライマリキーを使用して削除
int primaryKey = 222;
Table.DeleteItem(primaryKey);
```

ドキュメントモデル API（テーブルのクエリ）

```
DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
QueryFilter filter = new QueryFilter("Id", QueryOperator.Equal, primaryKeyValue);
filter.AddCondition("ReplyDateTime", QueryOperator.GreaterThan, twoWeeksAgoDate);
Search search = table.Query(filter);
List<Document> documentSet = new List<Document>();
do
{
    documentSet = search.GetNextSet();
    foreach (var document in documentSet)
    {
        //documentを利用した処理
    }

} while (!search.IsDone);
```

オブジェクト永続化 API

- .NET のクラスを DynamoDB の項目にマッピング
- クラスのプロパティに属性を指定してハッシュキー、レンジキーと紐づけ
  - DynamoDBHashKeyAttribute
  - DynamoDBRangeKeyAttribute
- .NET のプリミティブなデータ型やコレクション型がDynamoDBの型に自動的にマップされる
- 任意のデータ型のコンバーターも記述可能

オブジェクト永続化 API（クラス マッピング）

```
[DynamoDBTable("ProductCatalog")]
public class Book
{
    [DynamoDBHashKey]
    public int Id { get; set; }
    public string Title { get; set; }
    public int ISBN { get; set; }
    [DynamoDBProperty("Authors")]
    public List<string> BookAuthors { get; set; }
    [DynamoDBIgnore]
    public string CoverPage { get; set; }
}
```

オブジェクト永続化 API（オブジェクトの保存）

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
DynamoDBContext context = new DynamoDBContext(client);
Book myBook = new Book
{
    Id = 1001,
    Title = "object persistence-AWS SDK for.NET SDK-Book 1001",
    ISBN = "111-1111111001",
    BookAuthors = new List<string> { "Author 1", "Author 2" },
};
context.Save(myBook);
```

オブジェクト永続化 API（オブジェクトの更新）

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
DynamoDBContext context = new DynamoDBContext(client);

Book bookRetrieved = context.Load<Book>(1001);
bookRetrieved.ISBN = "222-2222221001";
bookRetrieved.BookAuthors = new List<string> { " Author 1", "Author x" };
context.Save(bookRetrieved);
```

オブジェクト永続化 API（オブジェクトの削除）

```
//強力な整合性のある読み取りオプションを指定してオブジェクトをロード
Book updatedBook = context.Load<Book>(1001,
            new DynamoDBContextConfig { ConsistentRead = true });
//オブジェクトを削除
context.Delete<Book>(1001);
//削除されたオブジェクトの読み取り。NULLが返るはず。
Book deletedBook = context.Load<Book>(1001,
            new DynamoDBContextConfig { ConsistentRead = true });
if (deletedBook == null)
    Console.WriteLine("削除されています。");
```

アジェンダ

Agenda

- AWS概要
- AWS SDK for .NET と AWS Tools for Visual Studio
- AWS SDK による自動化
- AWS サービスの利用
- まとめ

まとめ

- AWSの概要を理解し活用する
- AWS Tools for Visual Studio によって Web アプリケーションが簡単に展開できる
- AWS SDK を利用することでAWSの運用を自動化できる
- AWS SDK を利用することでAWS サービスを利用し、クラウド ネイティブなアプリケーションを開発することができる

Q&A

AWSトレーニングでは様々な学習方法をご提供しています

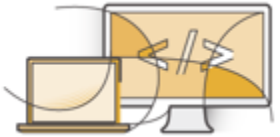
メリット

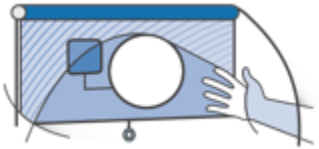
- AWS について**実習**や**実践練習**を通じて学習できる
- **AWS を熟知したエキスパート**から直接 AWS の機能について学び、疑問の答えを得られる
- **自信をもって** IT ソリューションに関する決定を下せるようになる

提供方法

e ラーニングや動画

セルフペースラボ





クラスルーム
トレーニング

詳しくは、http://aws.amazon.com/training をご覧ください

公式Twitter/Facebook
AWSの最新情報をお届けします

@awscloud_jp

もしくは
http://on.fb.me/1vR8yWm

最新技術情報、イベント情報、お役立ち情報、お得なキャンペーン情報などを
日々更新しています！

AWS 初心者向け Webinar

- AWSをこれからご使用になる技術者向け、ソリューションカットのセミナー
- 今後の配信予定

  1/28（木） 【AWS 初心者向け Webinar】AWS クラウドでの Windows の実行

  ※12時~13時分の時間帯です！
- 申し込みサイト
  - http://aws.amazon.com/jp/about-aws/events/

AWS Black Belt Tech Webinar 2016

AWSのサービスをディープにご紹介（19:00開始です）

- 1/13（水）AWS Black Belt Tech Webinar 2016 ～ Amazon Route 53 ～
- 1/20（水）　AWS Black Belt Tech Webinar 2016 ～ Amazon SQS & SNS ～
- 1/27（水）　AWS Black Belt Tech Webinar 2016 ～ Amazon CloudFront ～

- 申し込みサイト
  - http://aws.amazon.com/jp/about-aws/events/

AWSの導入、お問い合わせのご相談

- AWSクラウド導入に関するご質問、お見積り、資料請求をご希望のお客様は、以下のリンクよりお気軽にご相談ください
  https://aws.amazon.com/jp/contact-us/aws-sales/

※「AWS　問い合わせ」で検索してください

ご清聴ありがとうございました！