re:Generate

AWS マイスターシリーズ

AWS SDK for Java/.NET
Eclipse plugin,Visual Studio

2013.12.11

アマゾン データ 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
片山暁雄　渡邉源太

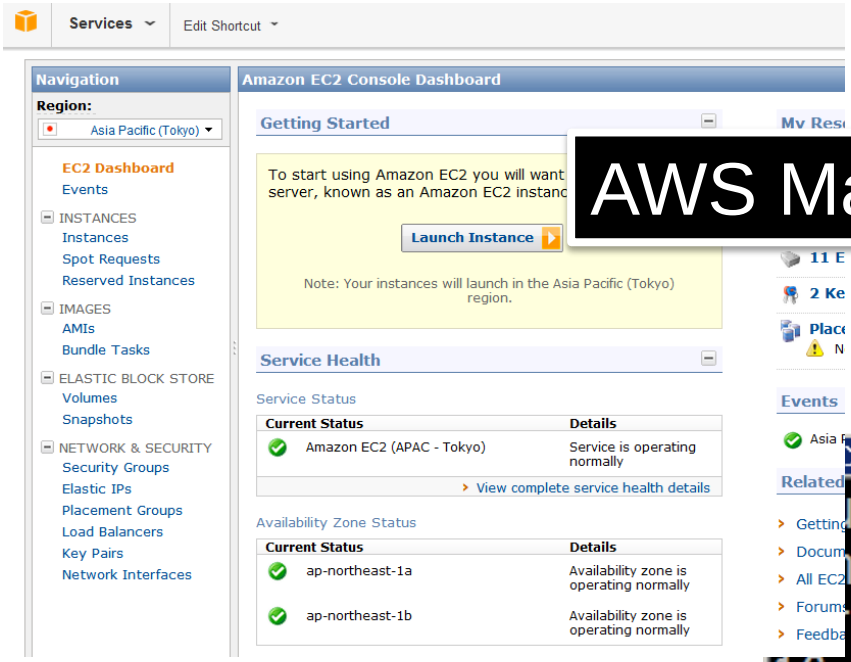
Agenda

- AWS SDK オーバービュー
- AWS SDK for Java
- AWS SDK for .NET
- まとめ

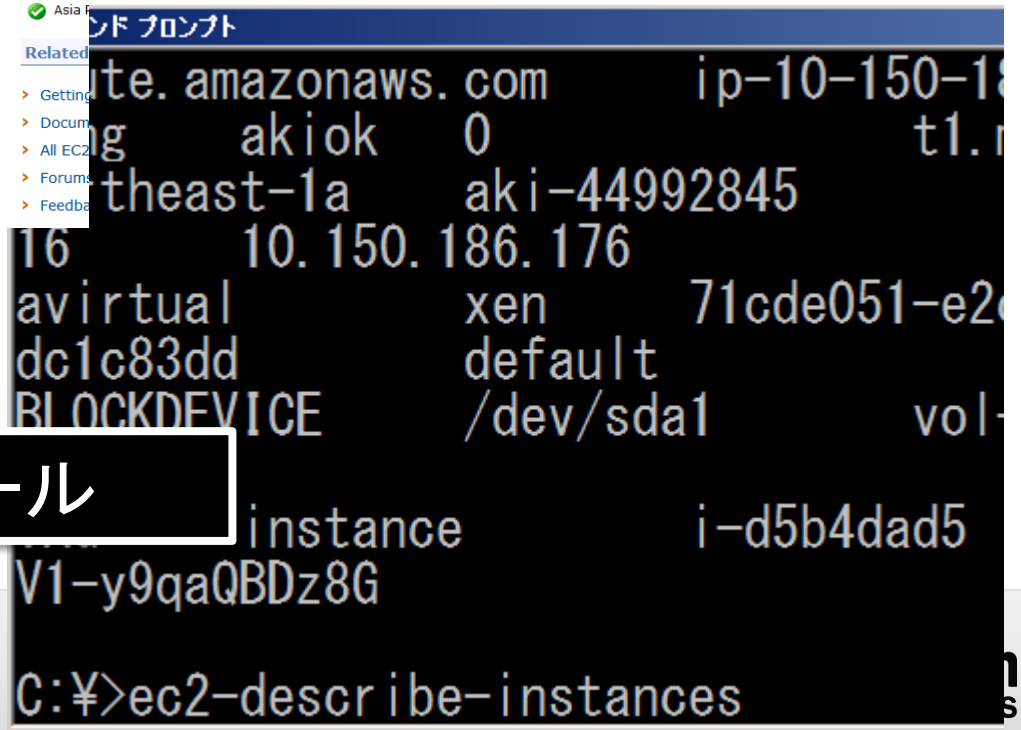
amazon web services

AWS SDK オーバービュー

AWSのサービス操作と言えば



AWS Management Console



AWSコマンドラインツール

これらの裏側では・・・

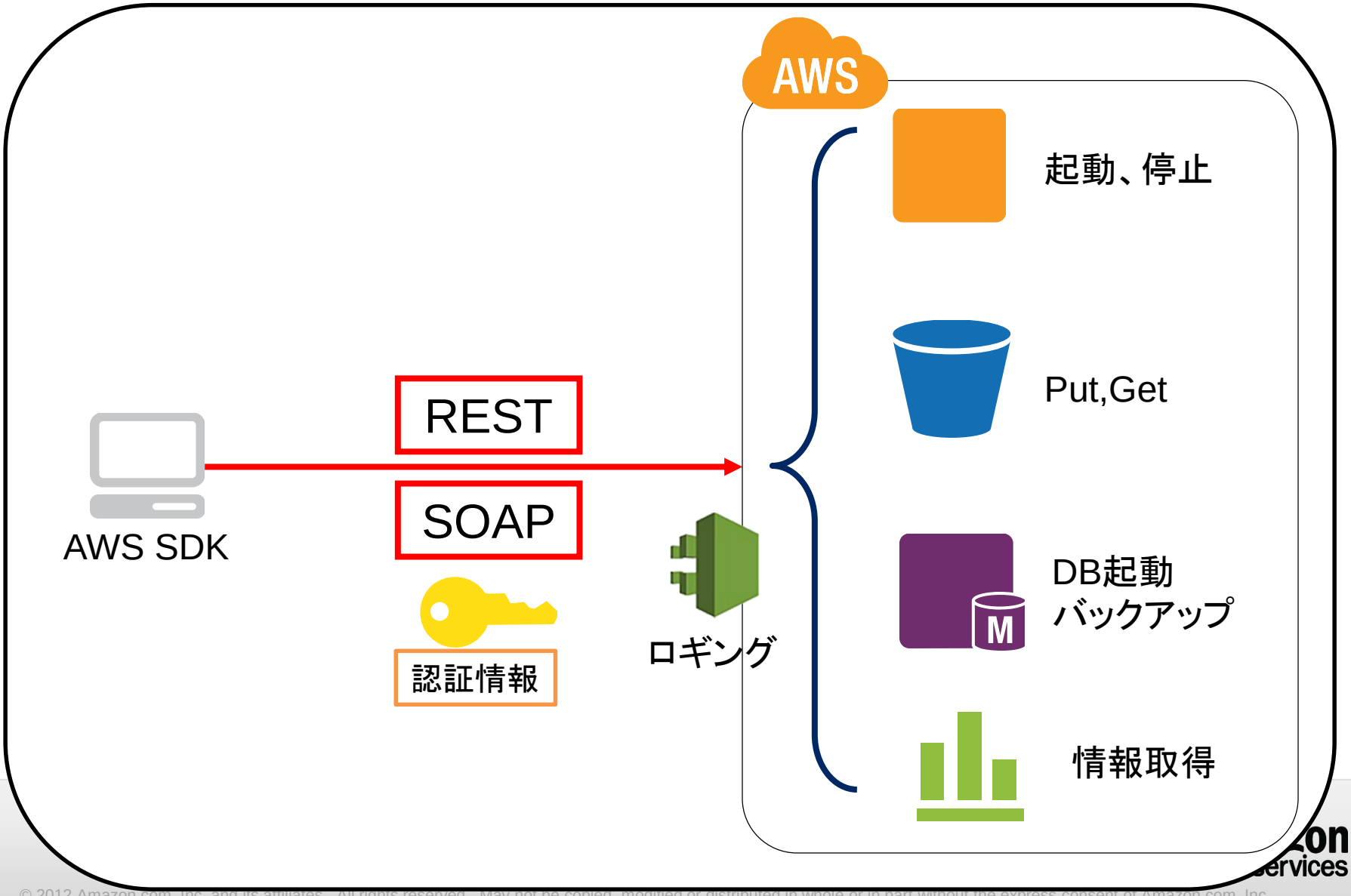
- 各サービスの各操作にAPIが定義されている
- AWSでは・・・
  - 人間がGUIまたはCUI越しに叩く
  - プログラミングしてそれを自動化・簡易化・カスタマイズ
    - 人間が手でやらなくてはいけない事をプログラミングして自動化できる、これがSDKで簡単に実現可能

    → AWSの実は最も優れた一面

AWS SDKとは

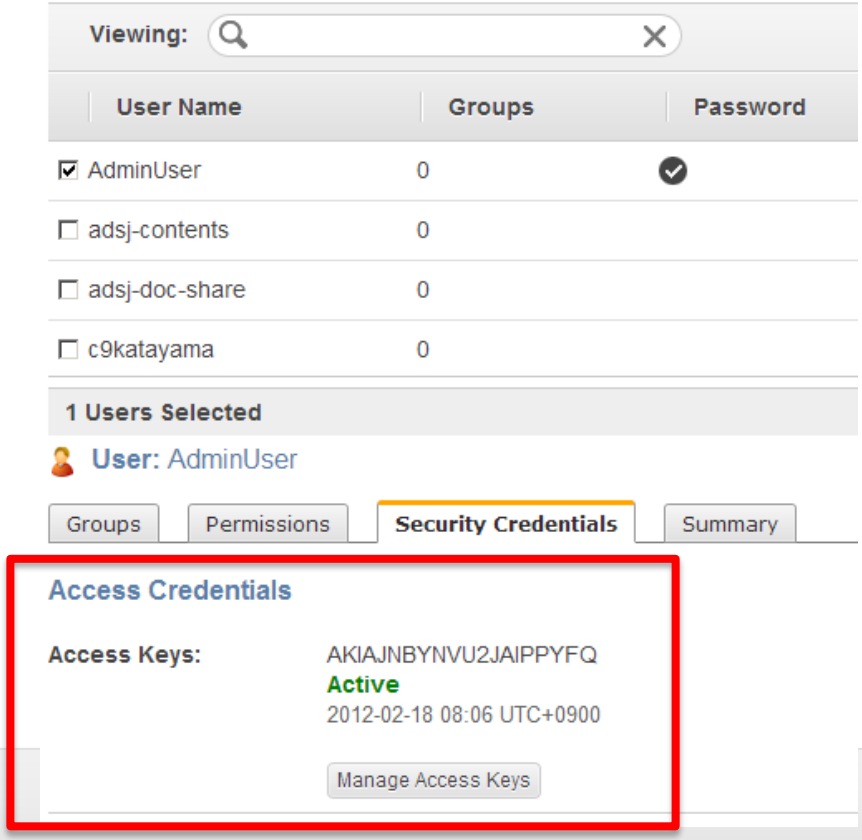
- AWSのサービスをプログラムで操作できるSDK
  - さまざまな言語で
    - AWS SDK for Java
    - AWS SDK for .NET
    - AWS SDK for Ruby
    - AWS SDK for PHP
    - AWS SDK for Python (boto)
    - AWS SDK for node.js
    - AWS SDK for Android
    - AWS SDK for iOS
    - 有志の方による実装（ActionScript）も
  - 通信は原則HTTPS

動作イメージ



認証情報

- AWSマネジメントコンソールのIAMタブで認証情報作成
  - アクセスキー、シークレットキー、(optional:MFA)
  - シークレットキーは作成時のみ取得可能



操作の種類

- 例：EC2
  - インスタンス起動・・・RunInstances
  - リブート・・・ RebootInstances
  - IPアドレス付与・・AllocateAddress
  - などさまざまな操作が、プログラムから実行可能
  - http://docs.amazonwebservices.com/AWSEC2/latest/APIReference/Welcome.html
  - SDKによってサポートする操作が異なる
- APIコールに対する制限は、 Identity and Access Management(IAM) のドキュメントを参照

AWS SDK for Java

AWS SDK for Java

- AWS提供の開発用Java SDK
  - http://aws.amazon.com/sdkforjava/
  - https://github.com/aws/aws-sdk-java
- 環境：Java6以降
- 最新版:1.6.8
- 依存ライブラリ
  - stax
  - aspectj
  - commons-codec
  - commons-logging
  - freemarker
  - httpcomponents-client
  - jackson
  - java-mail
  - spring

amazon web services

サポートしているサービス

Compute & Networking


AWS Direct Connect »


Amazon EC2 »


Elastic Load Balancing »


Auto Scaling »


Amazon EMR »


Amazon Route 53 »


Amazon VPC »

Storage & Content Delivery


Amazon S3 »


Amazon Glacier »


Amazon CloudFront »


AWS Storage Gateway »

AWS Import/Export »

App Services


Amazon Elastic Transcoder »


Amazon SQS »


Amazon SNS »


Amazon SES »


Amazon SWF »


Amazon CloudSearch »

Database


Amazon DynamoDB »


Amazon RDS »


Amazon Redshift »


Amazon ElastiCache »


Amazon SimpleDB »

Deployment & Management


AWS Elastic Beanstalk »


AWS CloudFormation »


Amazon CloudWatch »

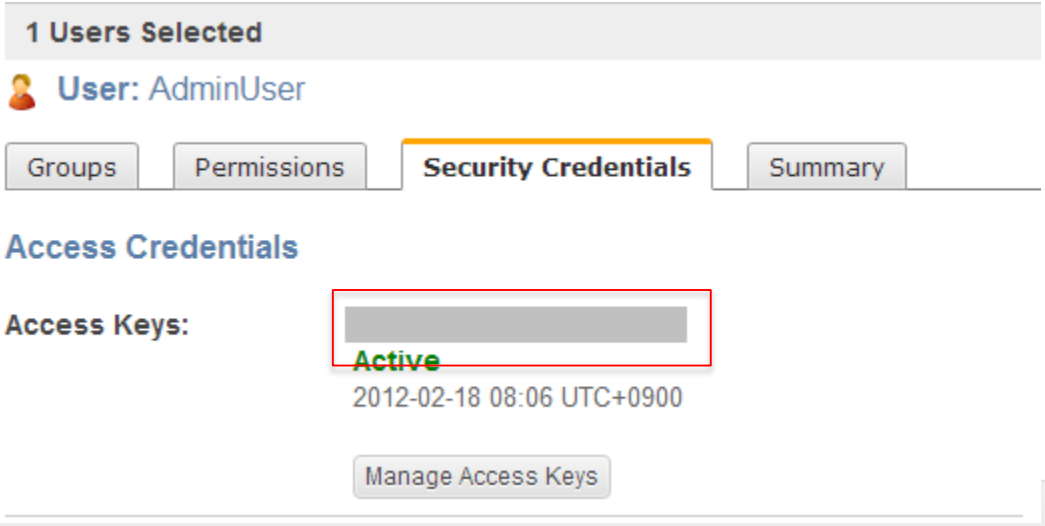

AWS Data Pipeline »


AWS Identity and Access Management »


AWS OpsWorks »

使い方

- SDKをダウンロード
  - クラスパスにjarファイルを設定
- 認証情報の取得
  - AWSマネジメントコンソールからIAMユーザーを作成し、クレデンシャル情報（Access Key, Secret Access Key） を取得



AWS Toolkit for Eclipse

- EclipseにAWS SDKを使ったプロジェクトを追加するプラグイン
- AWSを使用したアプリの開発/テストを効率化
- EC2やS3などのサービス管理コンソールも付属

amazon web services

Java SDKクライアントクラスの基本

- クレデンシャル情報を、クライアントクラスに設定
  - アクセスキー
  - シークレットアクセスキー
- 設定方法：
  - BasicAWSCredentialsで直接キーをコードに入れる（非推奨）
  - PropertiesCredentialsでプロパティファイルに記述する
  - STS(一時トークン）を取得して設定
  - IAM Roleを利用する

コード例

```
AWSCredentials credentials = new null;
try {
    credentials = new PropertiesCredentials(Util.class
.getResourceAsStream("AwsCredentials.properties"));
} catch (IOException e) {
    //例外処理
}
//このクレデンシャルを使ってClientを生成する
```

IAM利用時は、IAMユーザーのアクセスキー、シークレットキーを利用。
IAMユーザーでアクセス権のないメソッドを呼ぶと、例外が発生。

リージョンの設定

- デフォルトはus-east
- クライアントクラスへの設定方法は2種類
  - setEndpointメソッドで指定（旧来の方法）
    - setEndpoint("ec2.ap-southeast-1.amazonaws.com")
    - サービス毎のエンドポイントは下記URL参照
    - http://docs.amazonwebservices.com/general/latest/gr/rande.html
  - setRegionメソッドで指定（新しい指定）
    - setRegion(Region.getRegion(Regions.AP_NORTHEAST_1));

SetとWith

- API呼び出しパラメータのデータセットは、set～とwith～メソッドがある
  - 例： setInstanceType() と withInstanceType()
  - set ～は、通常のJavaのsetterとして利用
  - with ～は、戻り値としてパラメータのインスタンスを返す
    - with ～. with ～というように、メソッドを連続で呼び出せる

```
// 立ち上げたいインスタンス情報の作成
RunInstancesRequest req =
    new RunInstancesRequest("ami-xxxxx", 5, 5);

req.setKeyName(“yourkeyname")
req.withSecurityGroupIds(“yoursecgroup")
       .withMonitoring(true)
       .withInstanceType(InstanceType.M1Small);

// インスタンスの起動
ec2.runInstances(runInstancesRequest);
```

EC2

AmazonEC2Client

```
// EC2操作用のクライアント
AmazonEC2 ec2
                = new AmazonEC2Client(credentials);
// リージョン設定
ec2.setRegion(Region.getRegion(Regions.AP_NORTHEAST_1));

// 立ち上げたいインスタンス情報の作成
RunInstancesRequest runInstancesRequest = new
RunInstancesRequest("ami-xxxxx", 5, 5)
    .withKeyName(“yourkeyname")
    .withSecurityGroupIds(“yoursecgroup")
    .withMonitoring(true)
    .withInstanceType(InstanceType.M1Small);

// インスタンスの起動
ec2.runInstances(runInstancesRequest);
```

S3

- AmazonS3Client

```
// S3操作用クライアント
AmazonS3 s3 = new AmazonS3Client(credentials);

// bucket作成
s3.createBucket(“mybucket");

//オブジェクトのPUT
PutObjectResult ret = client.putObject("mybucket",
"aaa.txt", file);
```

S3での巨大ファイルの分割アップロード

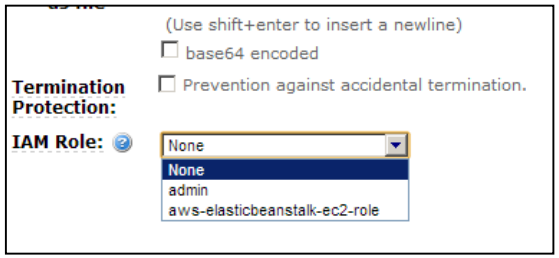
```
//TransferManagerを作成
AmazonS3Client client = new AmazonS3Client(cred);
TransferManager manager = new TransferManager(client);

//最低1チャンクを5MBに設定
TransferManagerConfiguration c =
                    new TransferManagerConfiguration();
c.setMinimumUploadPartSize(5 * 1024L * 1024L);
manager.setConfiguration(configuration);

//巨大ファイルのアップロード実行と進捗の監視
Upload upload =
          manager.upload(bucketName, bigfileName, target, o);
while (upload.isDone() == false) {…
```

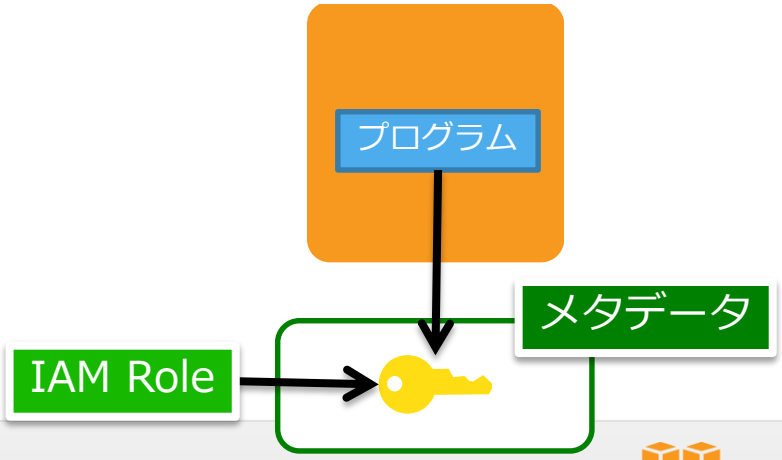
SDKならではの機能

- S3のクライアントサイド暗号化
- DynamoDBのオブジェクトマッパー
  - Plain Old Java Object (POJOs)とDynamoDBのテーブルをマッピング
- Amazon SQS クライアントサイドバッファリング
  - メッセージをクライアント側でバッファリング。パフォーマンスの改善
- AWS SimpleWorkflowService FlowFramework
- IAMロールを使った認証

IAM Role for EC2 instances

- EC2インスタンスに、指定のロールを付与する機能
  - EC2起動時にロールを指定すると、認証情報がメタデータに設定される
  - 認証情報はSTS(Security Token Service)で生成
    - インスタンス毎に異なるキー
    - 有効期限付きで、期限が来るとローテート
  - アプリケーションから認証情報を取得し、AWSサービスへアクセス
    - インスタンス内からメタデータにアクセス
    - アクセスキーID、シークレットアクセスキー、セッショントークンを取得
    - 3つの認証情報でAPI呼び出し





IAM Roleを使った認証

- EC2起動時に、IAM Roleを設定
- インスタンス上でSDKを利用。認証設定なしでAPIコール可能
  - インスタンスメタデータから、STS(AWS Security Token Service）をSDKが自動取得

```
AWSCredentials credentials =
        new BasicAWSCredentials(“アクセスキー”,”シークレットキーID”);
AmazonEC2 ec2 = new AmazonEC2Client(credentials);
ec2.describeInstances();
```

IAM Role利用後

```
AmazonEC2 ec2 = new AmazonEC2Client();
ec2.describeInstances();
```

IAM Role適用のインスタンス上では、認証情報の設定が不要

MFAデバイスを使ったAPIコール

- APIコール時に、MFAデバイスでの認証を必須とする機能
- 以下の流れで利用
  1. IAMユーザーを作成
  2. IAMユーザーのMFAデバイスを登録
  3. IAMユーザーに対して、ポリシーを登録
     - API呼び出しにMFAが必要とする制限を付与
  4. AWSSecurityTokenServiceClientを使用して、一時トークンを取得
     - ここでMFAデバイスのコードを使用
  5. 以降のAPIコールに、取得した一時トークンを使用

IAMポリシー設定例

```
{ "Version":"2012-10-17",
  "Statement":[
    { "Action":["ec2:*"],
      "Effect":"Allow",
      "Resource":["*"],
      "Condition":{
        "NumericLessThan":{"aws:MultiFactorAuthAge":"3600"}}
      }
  ]
}
```

MFA認証後1時間（3600秒）以内のAPIコールのみ許可

コード例

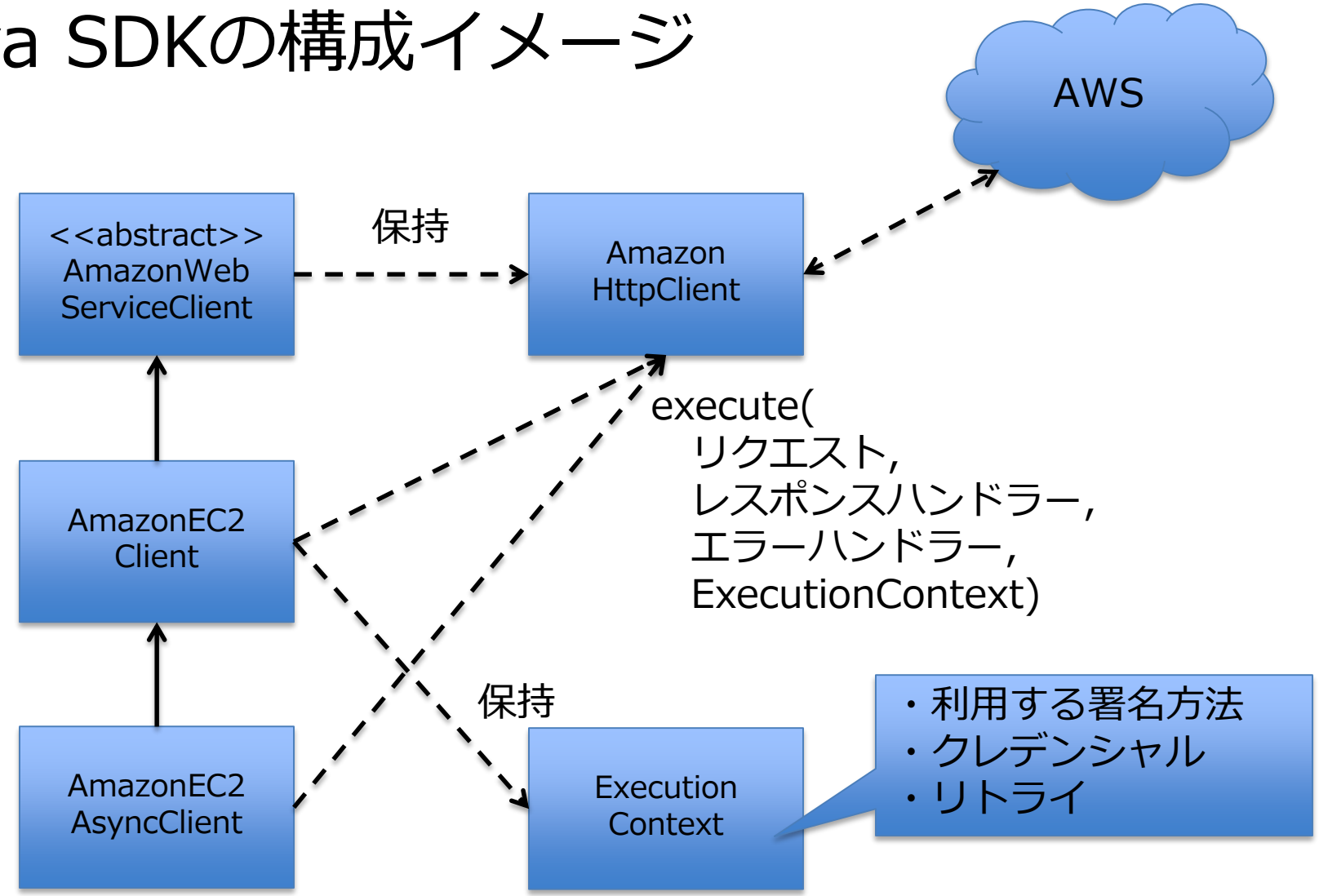
```
// 認証情報の読み込み
AWSCredentials credentials = new BasicAWSCredentials(“アクセスキー”,”シークレットキー”);
// Security Token Service呼び出し
AWSSecurityTokenServiceClient stsClient =
        new AWSSecurityTokenServiceClient(credentials);
GetSessionTokenRequest stsRequest = new GetSessionTokenRequest();
stsRequest.setDurationSeconds(3600);
stsRequest.setSerialNumber(“GAKTXXXXXXXX”);//登録したMFAデバイスのシリアルコード
stsRequest.setTokenCode(“000000”);//MFAデバイスの数値

GetSessionTokenResult stsResult = stsClient.getSessionToken(stsRequest);
Credentials stsCredentials = stsResult.getCredentials();

BasicSessionCredentials credentials = new BasicSessionCredentials(
            stsCredentials.getAccessKeyId(),
            stsCredentials.getSecretAccessKey(),
            stsCredentials.getSessionToken());

// EC2クライアントの作成
AmazonEC2Client ec2 = new AmazonEC2Client(credentials);
//以下通常のAPIコール
```

amazon web services

Java SDKの構成

- 各サービス毎にクライアントクラスが付属
  - 面倒な下記のようなことは利用者から隠ぺいしてくれる
    - HTTP通信及び例外ハンドリング
    - 署名
    - エラー時のリトライ処理
- サービスどれでもほぼ同一で直感的な使い勝手の提供
- 極力、利用者は単純に呼び出すだけでよい
- プラガブルな構成で、部分的に入れ替えられる
  - 挙動を変えたい
  - 例外のハンドリング方法を変えたい
  - 処理速度を上げたい

Java SDKの構成イメージ



Java SDKのパッケージ構成

- com.amazonaws
  - auth：署名関連
  - handlers：SDKの挙動をカスタマイズするハンドラー
  - http：実際のHTTPリクエストを処理する
  - internal：リトライ関連
  - sdk：バージョン情報
  - services：
  - 　各サービス毎の実体（クライアントクラス、データクラス）
  - transform：API経由で返ってくるXMLをパースする実体
  - util：ユーティリティ
- 各サービスのクライアントクラスは実体が2つ
  - 同期通信を中心としたクライアント：XxxClient
  - 非同期通信を追加した非同期クライアント：XxxAsyncClient

SDKの有効な利用方法

- 利用方法（運用面）
  - 決まったインスタンスを起動するバッチ
  - 固定ディスク（EBS）のスナップショットを定期取得
  - S3からデータを定期的に取得
  - S3上のデータを監視して、無くなったら通知
- 利用方法（アプリから）
  - アプリのバックエンドとして、DynamoDBを使う
  - メール送信のためにSMSを使う
  - データ保存のためにS3を使う

アドバンスドなトピック

- Java SDKは拡張性をかなり確保した状態になっている
- Client/AsyncClient自体も自分で拡張できる
  - インターフェースをそのまま実装する
- 細かく拡張ポイントを入れ替えることも可能
  - ClientConfiguration
  - RequestHandler
  - CustomBackoffStrategy
  - Marshaller/UnMarshaller

ClientConfiguration

- 各Clientクラスの基本的な挙動を決定する設定
  - ソケットのタイムアウト値
  - 最大コネクション数
  - 最大リトライ数
  - 使用するプロトコル(デフォルトHTTPS)
  - プロキシ関連設定
  - コネクションタイムアウト
- 各Client生成時にClientConfigurationを渡す

RequestHandler

- ServletでいうところのいわゆるFilter
  - デバッグ時や、何か統一的に処理させたい場合など便利
  - beforeRequest：リクエスト送信前
  - afterResponse：レスポンス受信後
  - afterError：エラー受信後
- 各サービスごとに定義可能
  - Clientで普通に渡す
  - com/amazonaws/services/xxx/request.handlersというファイルをおいて、これに定義するとロードされる
- 標準で既に定義されているものも
  - EC2RequestHandler
  - QueueUrlHandler
  - Route53IdRequestHandler

amazon web services

CustomBackoffStrategy

- AWSクラウド側へのリトライ間隔をコントロールする
  - カスタマイズして現状使っているのはDynamoDBのみ
  - 実行コンテキストであるExecutionContextに設定して渡す
- AmazonHttpClientの600行目付近
  - pauseExponentiallyメソッド

```
 private void pauseExponentially(int retries, AmazonServiceException
previousException, CustomBackoffStrategy backoffStrategy) {
        long delay = 0;
        if (backoffStrategy != null) {
            delay = backoffStrategy.getBackoffPeriod(retries);
        } else {
            long scaleFactor = 300;
            …
            delay = (long) (Math.pow(2, retries) * scaleFactor);
        }
        …
}
```

Marshaller/Unmarshaller

- AWSが提供しているXMLパース(一部JSONパース)では遅いので色々いらないものを省略してパースしたい
  - Java SDKではStAXというプルモデル型のパーサー
  - Clientを書いて、自分でMarshaller/Unmarshallerする
- 各サービス.model.transformの下
  - ここにMarshaller/Unmarshallerが大量にいる
- コーディングポリシー的には
  - Clientの各メソッド内でMarshallしてRequest<X>を作成
  - UnmarshallerはResponseHandlerにセットされて、AmazonHttpClientに渡されてレスポンス帰ってきたら実行

AWS SDK for .NET

AWS SDK for .NET

- AWS SDK for .NET
  - Amazon提供のAWS開発用.NET SDK
  - http://aws.amazon.com/sdkfornet/
  - 環境：
    - .NET Framework 3.5以降
    - Visual Studio 2010以降
  - 最新版2.0.2
  - C#およびVisual Basicをサポート

amazon web services

AWS SDK for .NET V2

- Windows StoreおよびWindows Phone 8アプリおよび非同期処理をサポートしたあたらしいバージョンのSDK
- .NET 4.5をターゲットにしたタスクベースの非同期パターンのサポート
- ターゲットとするプラットフォームごとに別個のアセンブリを提供
- あたらしいS3暗号化クライアントの提供

サポートしているサービス

Compute & Networking


AWS Direct Connect »


Amazon EC2 »


Elastic Load Balancing »


Auto Scaling »


Amazon EMR »


Amazon Route 53 »


Amazon VPC »

Storage & Content Delivery


Amazon S3 »


Amazon Glacier »


Amazon CloudFront »


AWS Storage Gateway »

AWS Import/Export »

App Services


Amazon Elastic Transcoder »


Amazon SQS »


Amazon SNS »


Amazon SES »


Amazon SWF »


Amazon CloudSearch »

Database


Amazon DynamoDB »


Amazon RDS »


Amazon Redshift »


Amazon ElastiCache »


Amazon SimpleDB »

Deployment & Management


AWS Elastic Beanstalk »


AWS CloudFormation »


Amazon CloudWatch »


AWS Data Pipeline »


AWS Identity and Access Management »


AWS OpsWorks »

AWS SDK for .NETに含まれるもの

- AWS Toolkit for Microsoft Visual Studio
- Visual Studioプロジェクトテンプレート
- AWS Tools for Windows PowerShell
- AWS .NETライブラリ
- C#コードサンプル
- ドキュメント

インストール方法

- 以下のページの右上隅にある「AWS .NET for SDK」ボタンをクリック
  - http://aws.amazon.com/jp/sdkfornet/
- ファイルを保存するかどうかをたずねるメッセージがブラウザに表示されたら、ローカルのディスクに保存
- 保存したインストーラを開いてインストールプロセスを開始

AWS Toolkit for Visual Studio

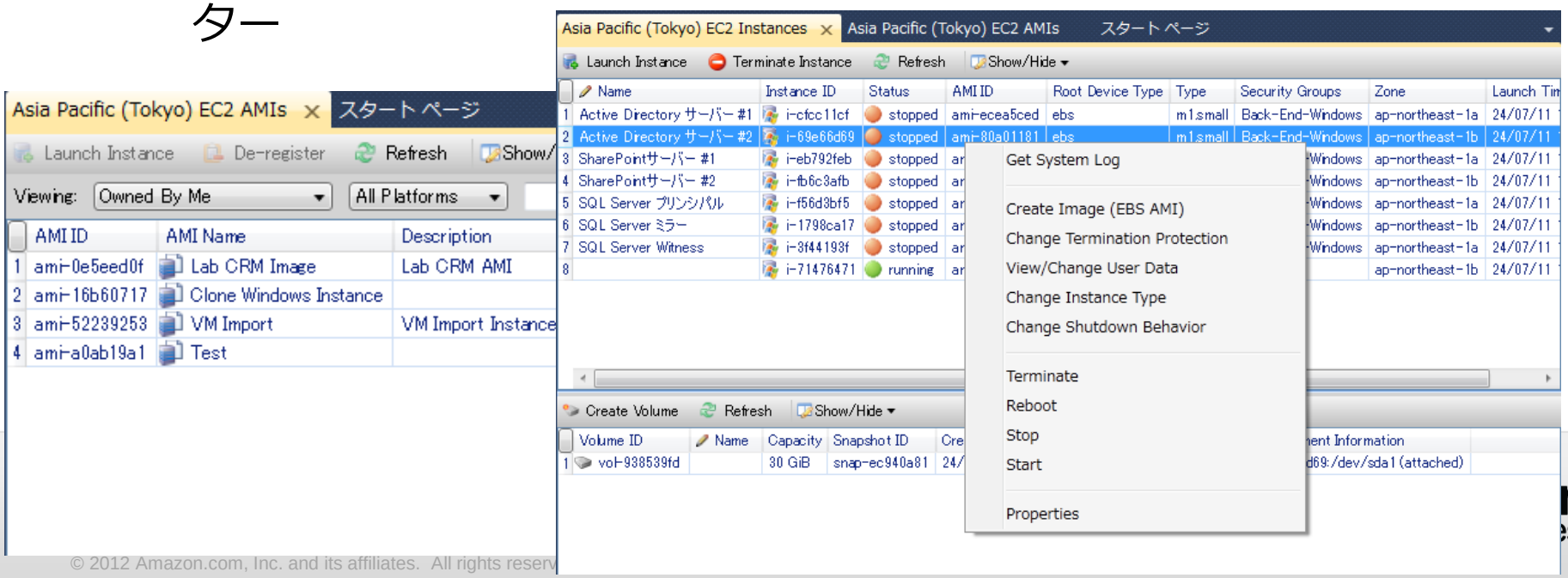
- Microsoft Visual Studioを使用してのサービスの管理が可能
- AWS SDK for .NETによるアプリケーション開発に対応
- AWS Elastic Beanstalk/AWS CloudFormationによる.NETアプリケーションのデプロイに対応

機能

- アカウント管理
- AWS Explorer
  - 各種サービスの情報表示/操作
- EC2 コンソール
  - EC2のコントロールパネル
- AWS CloudFormation
- AWS Elastic Beanstalk
  - デプロイ、デバッグ、環境構築

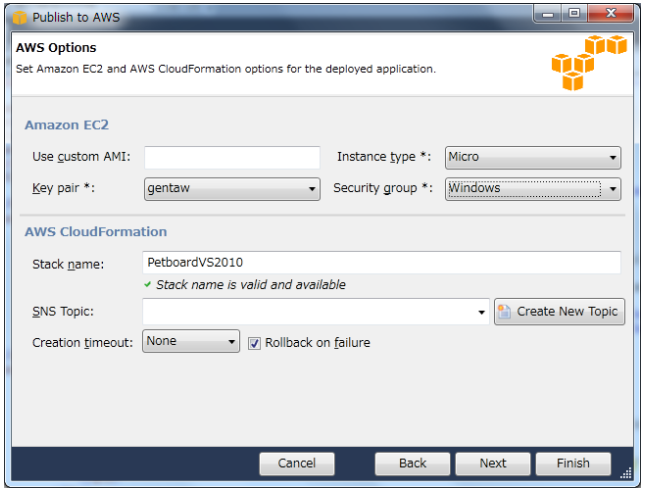
EC2 コンソール

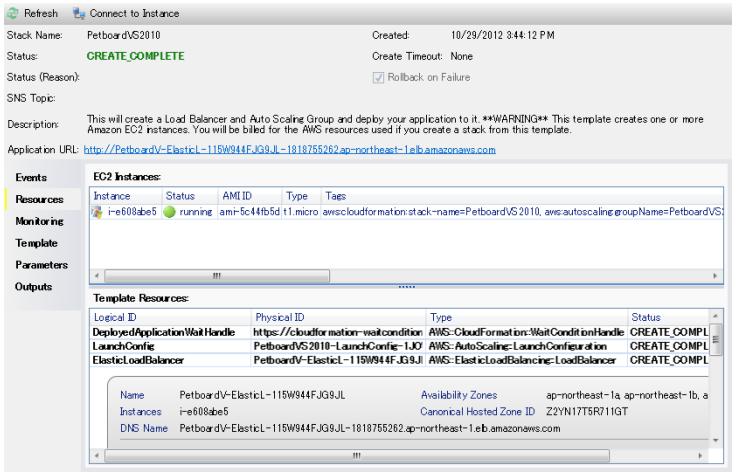
- AMI管理
  - 検索、削除、AMIからの起動
- インスタンス管理
  - 右クリックから状態変更、AMI作成、EBSの操作
  - インスタンスタイプおよびセキュリティグループによるフィルター



AWS CloudFormation

- AWS CloudFormationテンプレートによるアプリケーションのデプロイ
- Single Instance TemplateまたはLoad Balanced Templateから選択
- .NET Frameworkのバージョンが指定可能
- Application URLからアプリにアクセス可能





.NET SDKにおけるアクセスキー管理

- Service Clientにクレデンシャルを設定

```
var accessKey = "";// Get access key from a secure store
var secretKey = "";// Get secret key from a secure store
var s3Client =
AWSClientFactory.CreateAmazonS3Client(accessKey,secretKey,R
egionEndpoint.APNortheast1;
```

- アプリケーションコンフィグ（App.config）に記述
- IAM Roleを利用する

App.configの設定

- App.configにアクセスキーとシークレットキーを以下のように設定

```
<configuration>
 <appSettings>
        <add key="AWSAccessKey" value="[ENTER YOUR ACCESS KEY ID HERE]"/>
        <add key="AWSSecretKey" value="[ENTER YOUR SECRET ACCESS KEY HERE]"/>
 </appSettings>
</configuration>
```

リージョンの設定

- リージョンの選択は.NET SDK V2では必須
- App.configで指定

```
<configuration>
  <appsettings>
    <add key="AWSRegion" value="ap-northeast-1">
  </add></appsettings>
</configuration>
```

- RegionEndPointパラメータで指定
  - var s3Client = new AmazonS3Client(RegionEndpoint.APNortheast1);

- サービス毎のエンドポイントは下記URL参照
  - http://docs.amazonwebservices.com/general/latest/gr/rande.html

.NET SDK V2でサポートされない機能

- V1でサポートされていたWithメソッドは使用できない

```
TransferUtilityUploadRequest uploadRequest = new
TransferUtilityUploadRequest()
    .WithBucketName("my-bucket")
    .WithKey("test")
    .WithFilePath("c:¥test.txt");
```

- Withメソッドのかわりにコンストラクタ初期化子（Constructor Initializer）をつかう

```
TransferUtilityUploadRequest uploadRequest = new
TransferUtilityUploadRequest
{
    BucketName = "my-bucket",
    Key = "test",
    FilePath = "c:¥test.txt"
};
```

amazon web services

サンプル：EC2インスタンス起動

```
// EC2操作用のクライアント
var ec2Client = new AmazonEC2Client();

// 立ち上げたいインスタンス情報の作成
 var runInstanceRequest = new RunInstancesRequest{
        ImageId = "ami-xxx",
        MaxCount = 1,
        MinCount = 1,
        KeyName = “YourKeyName",
        SecurityGroups =  { “yourSecurityGroup” },
        InstanceType = InstanceType.M1Small
        };

// インスタンスの起動
ec2Client.RunInstances(runInstanceRequest);
```

サンプル：S3バケットの作成

```
// S3操作用クライアント
var s3Client = new AmazonS3Client();

// bucket情報の設定
 var putBucketRequest = new PutBucketRequest
        {
           BucketName = "Mybucket",
           BucketRegion = S3Region.APN1
        };

// bucket作成
s3Client.PutBucket(putBucketRequest);
```

サンプル：S3オブジェクトのアップロード

```
// S3操作用クライアント
var s3Client = new AmazonS3Client();

//オブジェクト情報の設定
 var putObjectRequest = new PutObjectRequest
        {
            BucketName = "Mybucket",
            Key = "Item1",
            FilePath = "contents.txt"
        };

//
var response = s3Client.PutObject(putObjectRequest);
```

amazon web services

サンプル：S3オブジェクトのアップロード（非同期）

```
// 非同期アップロード用のメソッド
async Task UploadFile(string bucketName, string filepath)
{
var s3Client = new AmazonS3Client();
var request = new PutObjectRequest()
{
BucketName = bucketName,
FilePath = filepath
};

await s3Client.PutObjectAsync(request);
Console.WriteLine("File Uploaded");
}
```

```
// UploadFileメソッドの呼び出し
await UploadFile(bucketName, filepath);
```

サンプル：RDSインスタンスの作成

```
// RDS操作用クライアント
var rdsclient = new AmazonRDSClient();

// DB instance情報の設定
 var createDBInstanceRequest = new CreateDBInstanceRequest
         {
            AllocatedStorage = 30,
            DBInstanceClass = "db.m1.small",
            DBInstanceIdentifier = "sqlserver",
            Engine = "sqlserver-ex",
            MasterUsername = "sa",
            MasterUserPassword = "password"
          };

// DB instance作成
 rdsclient.CreateDBInstance(createDBInstanceRequest);
```

.NET SDKの特徴

- Amazon DynamoDB オブジェクト永続フレームワーク
  - .NETクラスをAmazon DynamoDBの項目にマッピングしてデータを格納および取得
- Amazon S3 TransferUtility
  - マルチスレッド化されたAmazon S3マルチパートアップロード
- Amazon Glacier Archive TranferManager
  - 高レベルAPIによる自動的なファイル分割
- Amazon DynamoDB Session State Provider
  - ASP .NETのセッション状態をDynamoDBに格納

amazon web services

サンプル：Amazon DynamoDBオブジェクト永続フレームワークの利用（1/2）

- DynamoDBのテーブルに”Book”タイプを設定
- ハッシュキーは”Id”, “Title”, “Authors”, “Price”

```
[DynamoDBTable("Books")]
class Book
{
    [DynamoDBHashKey]
    public int Id { get; set; }
    public string Title { get; set; }
    public List Authors { get; set; }
    public double Price { get; set; }
}
```

サンプル：Amazon DynamoDBオブジェクト永続化フレームワークの利用（2/2）

- DynamoDBContextから”Book”タイプを利用

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

// アイテムの格納
Book book = new Book
{
    Title = "Cryptonomicon",
    Id = 42,
    Authors = new List { "Neal Stephenson" },
    Price = 12.95
};
context.Save(book);

// アイテムの取得
book = context.Load(42);
Console.WriteLine("Id = {0}", book.Id);
Console.WriteLine("Title = {0}", book.Title);
Console.WriteLine("Authors = {0}", string.Join(", ", book.Authors));
```

AWS SDK for .NET
プラットフォームによる差異

- AWS SDK for .NET Framework 3.5
  - V1の機能および同様の非同期処理をサポート
- AWS SDK for .NET Framework 4.5
  - V1の機能に加えてC# 5.0のasync/awaitによるタスクベースの非同期処理
- AWS SDK for Windows RT
  - 非同期処理はasync/awaitのみ
  - Amazon S3とAmazon DynamoDBの一部機能が利用不可
    - Transfer Utility
    - IO Namespace
  - GetDecryptedPasswordによるWindowsパスワード取得が利用不可
- AWS SDK for Windows Phone 8
  - Windows RTと同様の制限
  - AWS SDK for Android/iOSと同等のサービスをサポート

まとめ

- SDKを使うことで
  - 運用管理での煩雑な手間をコーディングで楽にすることが可能
  - どのサービスでも同じような使い勝手で利用可能
  - 実際のサービス開発のお供に
    - S3
    - SNS/SQS/SWF
    - DynamoDB
- AWSはコーディングする方の力を最大限に引き出すインフラ
  - HAPPY CODING!!

amazon web services

参考情報

- AWS SDK for .NET
  - http://aws.amazon.com/jp/sdkfornet/
- AWS SDK for .NET API Reference
  - http://docs.aws.amazon.com/sdkfornet/latest/apidocs/Index.html
- AWS Forum development in .NET
  - https://forums.aws.amazon.com/forum.jspa?forumID=61
- AWS blog .NET Development
  - http://blogs.aws.amazon.com/net/