AWS マイスターシリーズ

AWS SDK for PHP & AWS SDK for Ruby & boto(Python) & JavaScript in Node.js

2013.12.20
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
今井　榎並　蒋

ウェビナー(Webセミナー)へようこそ！

- 参加者は、自動的にミュートになっています
- 質問を投げることができます！
  - GoToWebinarの仕組みを使って、書き込んでください
    - ただし環境によっては、日本語の直接入力ができないので、
      お手数ですが、テキストエディタ等に打ち込んでから、
      コピペしてください
  - 最後のＱ＆Ａの時間で、できるだけ回答させて頂きます
  - 書き込んだ質問は、主催者にしか見えません
- Twitterのハッシュタグは#jawsugでどうぞ

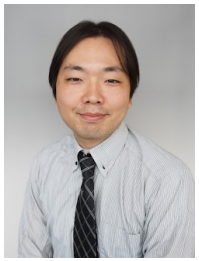
amazon web services

Webセミナー
週刊AWSマイスターシリーズ　re:Generate!

- AWSマイスターシリーズ
  - 12月4日　AWS re:Invent アップデート振り返り
  - 12月11日 AWS SDK for Java/.Net
  - 申し込みサイト
    - http://aws.amazon.com/jp/event_schedule/
  - 過去資料集
    - http://aws.amazon.com/jp/aws-meister/











Agenda

- AWS SDK 概要
- AWS SDK for PHP
- boto(Python)
- AWS SDK for Ruby
- AWS SDK for JavaScript in Node.js
- まとめ

amazon web services

AWS SDK概要

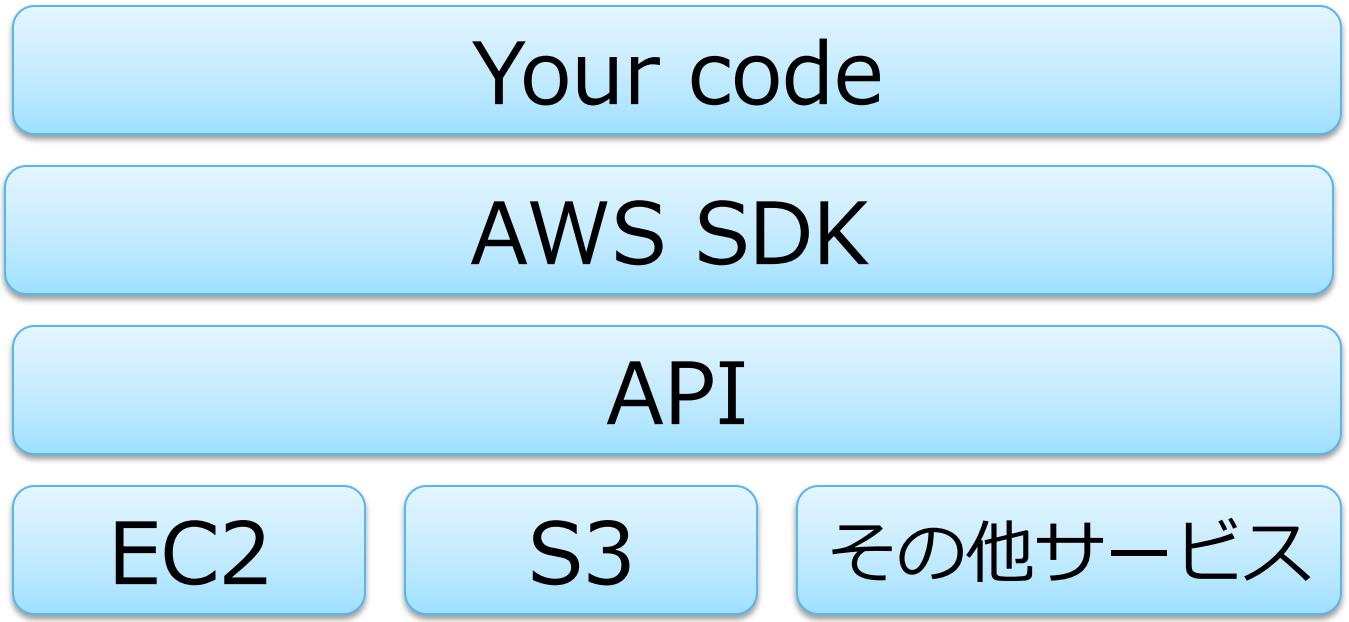
AWS SDKの重要性

- AWSはプログラマブルなインフラ
  - プログラムから扱うためにはSDKが必要
  - ほぼすべてのサービスはマネジメントコンソールから利用できるが、自動化しようと思ったらプログラムから扱うのが必須
  - 自動化するとAWSは何倍も便利になる

- 代表的な用途
  - インフラ構築/運用の自動化
    - EC2やRDSを上げたり下げたり、CloudFormationでスタックをデプロイしたり
  - アプリケーション的なサービスの利用
    - S3にデータをアップしたり、DynamoDBやSQSにデータ入れたり出したり

AWS SDK

- AWSのサービスはHTTP/HTTPSでREST/SOAP形式のAPIをサポート。
  例えば
  - EC2：Start/Stop/Terminate
  - S3：GetObject/PutObject
  - DynamoDB：Get/Put
- SDKはこれらのAPIを抽象化し、各言語からの利用を非常に簡単にしてくれる



AWS SDK

- 下記の言語/環境で提供中


Java


Python


PHP


.NET


Ruby


JavaScript
in nodeJS


JavaScript
in the Browser


iOS


Android

AWS SDK

- 今回は下記についてお話します。

Java

Python

PHP

.NET

Ruby

JavaScript in Node.js

JavaScript in the Browser

iOS

Android

AWS SDKの使い方

- 必要な言語のSDKをインストール
  - 言語ごとにインストール方法が異なります。このあとの章で言語ごとに解説します。
- Credential(AWS APIの認証情報)を用意する
- ファクトリーメソッドを利用してサービス(例えばS3)のクライアントオブジェクトを生成
  - このときにCredentialを渡す
- クライアントオブジェクトのメソッドを使ってオペレーション(例えばPutObject)
- 言語によってはより高度に抽象化されているSDKもあります。

Credentialの取り扱い

- Credentialの取り扱いについてはいくつかのやり方がありますが、IAM Roleもしくは環境変数を使うのがオススメです。PHPを例にそれぞれの実装方法をご案内します。

- コード内に直接埋め込むパターン(コード内にCredentialが入り込んでしまうのでやっちゃダメなパターン)

```
<?php
$s3client = S3Client::factory(array(
        'key'    =>    'your-aws-access-key-id',
        'secret‘ =>    'your-aws-secret-access-key',
));
```

Credentialの取り扱い

- configファイルでCredentialを設定するパターン
  - PHPとbotoではこの方法が提供されている
  - これもCredentialがファイル化されるので良くないパターン

```
$ vim aws-config.php
<?php
return array(
    'includes' => array('_aws'),
    'services' => array(
        'default_settings' => array(
            'params' => array(
                'key'    => 'your-access-key-id',
                'secret' => 'your-secret-access-key',)
        )
    )
);

$ vim sample.php
<?php
$aws = Aws::factory(‘aws-config.php');
$s3client = $aws->get('s3');
```

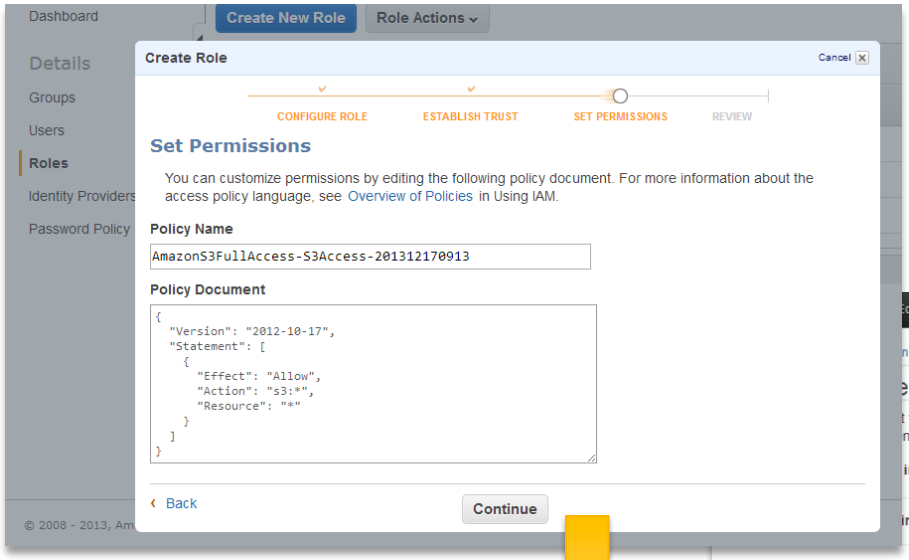
Credentialの取り扱い

- 環境変数でCredentialを設定するパターン
  - AWS_ACCESS_KEY_ID, AWS_SECRET_KEYという環境変数を実行ユーザーで設定しておくことによってコードからCredentialを追い出せる

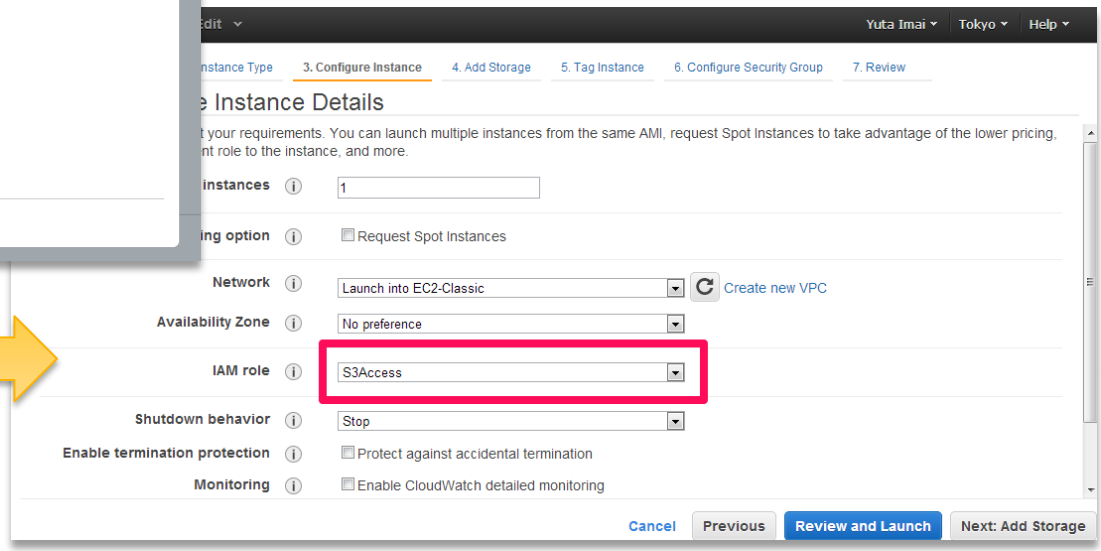
```
$ export AWS_ACCESS_KEY_ID="your-aws-access-key-id"
$ export AWS_SECRET_KEY="your-aws-secret-access-key”

$vim sample.php
<?php
$s3client = S3Client::factory();
```

Credentialの取り扱い

- IAM RoleでCredentialを設定するパターン
  - コードが動くのがEC2上であれば、IAM Roleを使うことによって、EC2自体にAWS APIへのアクセス権限を付与できる。よってコードや環境変数にCredentialを持たせずに済む。



例えばS3フルアクセスの権限を持ったIAM Roleを作成して・・・



EC2にそのRoleを割り当てると・・・

Credentialの取り扱い

- IAM RoleでCredentialを設定するパターン
  - 前のページのようにS3アクセス権限を持ったRoleをEC2に割り当てると、下記を書くだけで利用可能。

```
$vim sample.php
<?php
$s3client = S3Client::factory();
```

  - これは、SDKが内部でSTS(Security Token Service)を利用しており、AccessKey,SecretAccessKey,Tokenを自動的に

AWS SDK for PHP

AWS SDK for PHP

- Amazon提供のAWS開発用のPHP向けSDK
- 現在は、Version2
- http://aws.amazon.com/jp/sdkforphp/
- 環境：PHP5.3.3以降
- Guzzle HTTP Client framework上で構築されている
- 依存Extension
  - cURL

操作可能サービス

Direct Connect	Elasticache	IAM
EC2	SimpleDB	OpsWorks
ELB	S3	Elastic Transcoder
Auto Scaling	Glacier	SQS
EMR	CloudFront	SNS
Route53	Storage Gateway	SES
VPC	Import/Export	SWF
DynamoDB	Elastic Beanstalk	CloudSearch
RDS	Cloud Formation	
Redshift	Cloud Watch	

amazon web services

利用方法

- **Composerによるインストール（推奨）**
  - http://docs.aws.amazon.com/aws-sdk-php/guide/latest/installation.html#installing-via-composer
- **Pharよるインストール**
  - http://docs.aws.amazon.com/aws-sdk-php/guide/latest/installation.html#installing-via-phar
- **Zipファイルからインストール**
  - http://docs.aws.amazon.com/aws-sdk-php/guide/latest/installation.html#installing-via-zip
- **PEARによるインストール**
  - http://docs.aws.amazon.com/aws-sdk-php/guide/latest/installation.html#installing-via-pear

amazon web services

初期設定

- インストール方法により以下のとおり異なります

インストール方法	インクルード
Composer	require '/path/to/vendor/autoload.php';
Phar	require '/path/to/aws.phar';
Zip	require '/path/to/aws-autoloader.php';
PEAR	'AWSSDKforPHP/aws.phar';

- コンフィグレーションファイルの設定
  - クレデンシャル情報などを設定可能
  - factory()の引数として指定することも可能
  - ‘src/Aws/Common/Resources/aws-config.php’ を参考に追加

amazon web services

サンプル：S3 - putObject

API仕様: http://docs.aws.amazon.com/aws-sdk-php/latest/class-Aws.S3.S3Client.html#_putObject

```
<?php
require '/path/to/vendor/autoload.php';
use Aws¥Common¥Aws;
use Aws¥S3¥Exception¥S3Exception;

try {
    $client = S3Client::factory;   ①
    $bucket = 'your_bucket';
    $file = 'test.txt';
    $result = $client->putObject(array(      ②
            'Bucket' => $bucket,
            'Key' => $file,
            'Body' => fopen($file, 'r'),
    ));
    // 結果表示
    var_dump($result);

} catch (S3Exception $e) {
    echo '*** Error ***' . "¥n";
    echo $e->getMessage();
}
```

① S3クライアント作成

② putObject実行

Bucketを指定、Keyは、ファイル名、Bodyにアップロードするファイルストリームを指定

(*) 事前にBucketが作成されている必要があります。

サンプル：S3 - getObject

API仕様: http://docs.aws.amazon.com/aws-sdk-php/latest/class-Aws.S3.S3Client.html#_getObject

```
<?php
require '/path/to/vendor/autoload.php';
use Aws¥Common¥Aws;
use Aws¥S3¥Exception¥S3Exception;

try {
    $client = S3Client::factory;①
    $bucket = 'your_bucket';
    $file = 'test.txt';
    $result = $client->getObject(array(    ②
            'Bucket' => $bucket,
            'Key' => $file,
));
    // 結果表示
    echo $result['Body'];  ③

} catch (S3Exception $e) {
    echo '*** Error ***' . "¥n";
    echo $e->getMessage();
}
```

① S3クライアント作成

② getObject実行

Bucketを指定、Keyは、ファイル名

③ 取得したコンテンツを表示

(*) 事前にBucketが作成されている必要があります。

サンプル：SQS - sendMessage

API仕様: http://docs.aws.amazon.com/aws-sdk-php/latest/class-Aws.Sqs.SqsClient.html#_sendMessage

```
<?php
require '/path/to/vendor/autoload.php';
use Aws¥Common¥Aws;
use Aws¥Sqs¥Exception¥SqsException;

try {
    $client = SqsClient::factory;          ①
    $queueUrl = 'https://YourQueue';
    $result = $client->sendMessage(array(  ②
            'QueueUrl' => $queueUrl,
            'MessageBody' => 'Send Message!',
    ));
    // 結果表示
    var_dump($result);

} catch (SqsException $e) {
    echo '*** Error ***' . "¥n";
    echo $e->getMessage();
}
```

① SQSクライアント作成

② sendMessage実行

QueueURLを指定、送りたいメッセージを指定

(*) 事前にQueueが作成されている必要があります。

サンプル：SQS - receiveMessage

API仕様: http://docs.aws.amazon.com/aws-sdk-php/latest/class-Aws.Sqs.SqsClient.html#_receiveMessage

```
<?php
require '/path/to/vendor/autoload.php';
use Aws¥Common¥Aws;
use Aws¥Sqs¥Exception¥SqsException;

try {
    $client = SqsClient::factory;  ①
    $queueUrl = 'https://YourQueue';
    $result = $client->receiveMessage(array(  ②
            'QueueUrl' => $queueUrl,
    ));
    // 結果表示
    var_dump($result);

} catch (SqsException $e) {
    echo '*** Error ***' . "¥n";
    echo $e->getMessage();
}
```

① コンフィグを指定してインスタンス作成

② SQSクライアント作成

③ receiveMessage実行

QueueURLを指定

(*) 事前にQueueが作成されている必要があります。

サンプル：DynamoDB - putItem

API仕様: http://docs.aws.amazon.com/aws-sdk-php/latest/class-Aws.DynamoDb.DynamoDbClient.html#_putItem

```
<?php
require '/path/to/vendor/autoload.php';
use Aws¥Common¥Aws;
use Aws¥DynamoDb¥Exception¥DynamoDbException;

try {
    $client = DynamoDbClient::factory;  ①
    $result = $client->putItem(array(
        'TableName' => 'table_name',
        'Item' => $client->formatAttributes(array( ②
            'id'        => 100,
            'timestamp' => 130699342,
            'message'      => 'Good Morning.',
        )),
    ));
    // 結果表示
    var_dump($result);

} catch (DynamoDbException $e) {
    echo '*** Error ***' . "¥n";
    echo $e->getMessage();
}
```

① DynamoDBクライアント作成

② putItem実行
- TableName: テーブル名
- Item: Putするアイテムを指定

（例）
id : Hash Key
timestamp : Range Key
message : Attribute

(*) 事前にTableが作成されている必要があります。

サンプル：DynamoDB - getItem

API仕様: http://docs.aws.amazon.com/aws-sdk-php/latest/class-Aws.DynamoDb.DynamoDbClient.html#_getItem

```
<?php
require '/path/to/vendor/autoload.php';
use Aws¥Common¥Aws;
use Aws¥DynamoDb¥Exception¥DynamoDbException;

try {
    $client = DynamoDbClient::factory; ①
    $result = $client->getItem(array(    ②
        'ConsistentRead' => true,
        'TableName' => 'table_name',
        'Key' => array(
              'id' => array('N' => '100'),
              'timestamp' => array('N' => '130699342')
              )
    ));
   // 結果表示
    var_dump($result);

} catch (DynamoDbException $e) {
    echo '*** Error ***' . "¥n";
    echo $e->getMessage();
}
```

① DynamoDBクライアント作成

② getItem実行
- TableName: テーブル名
- Key:GetするアイテムのKeyを指定

（例）
id : Hash Key
timestamp : Range Key

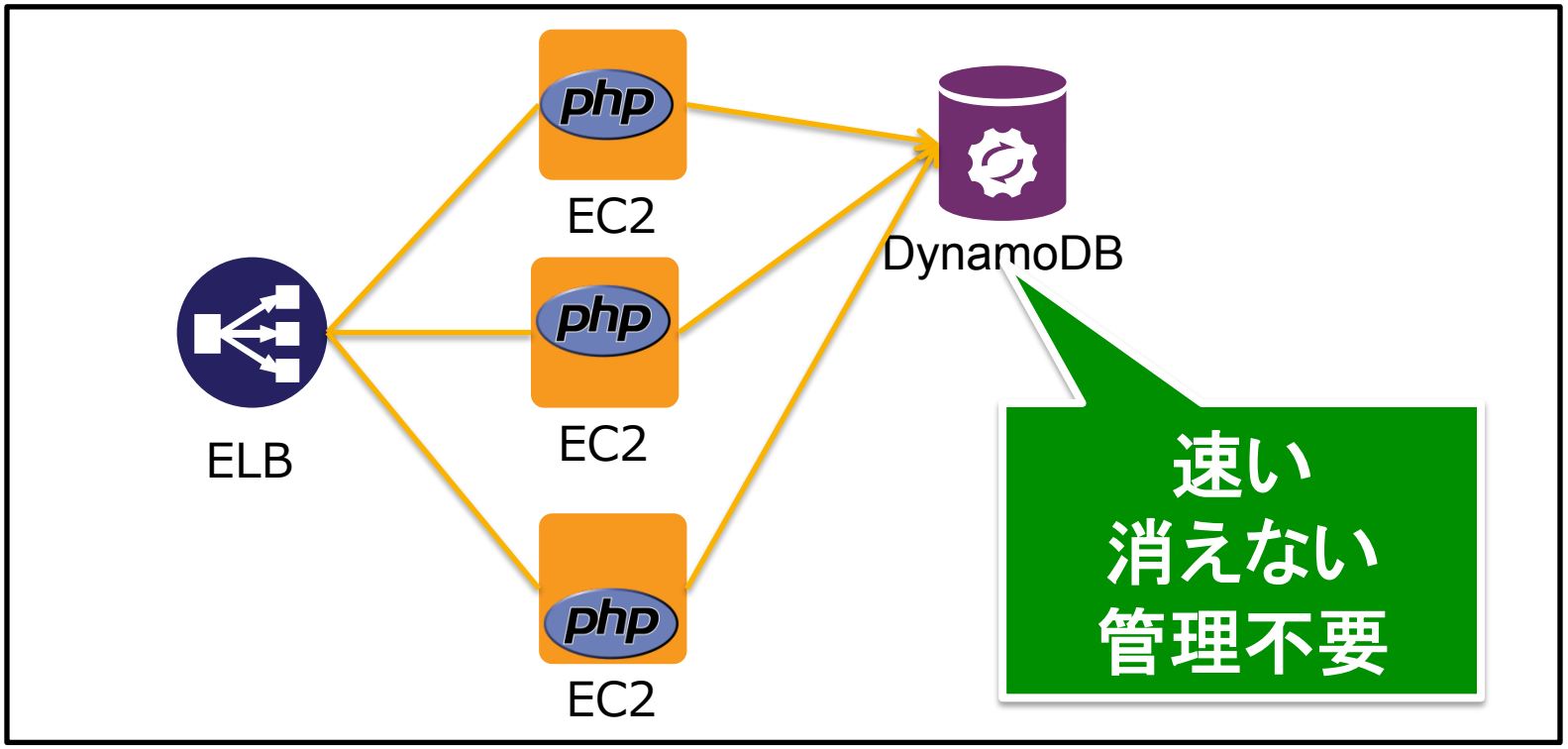
(*) 事前にTableが作成されている必要があります。

活用法

- **PHPアプリのバックエンドストレージ呼び出し**
  - DynamoDB,SQS,S3を簡単に呼び出し可能
- **PHPベースのアプリケーションの拡張**
  - WordpressやEC-CubeなどのPHPベースのアプリケーションとAWSの連携
- **独自の管理用Webインターフェースの作成**
  - 定型作業を簡略化したり、管理を容易にするための独自Webインターフェースの作成

PHP版SDKの便利なクラス

- DynamoDBSessionHandler
  - DynamoDBを使った、HTTPのセッション共有が可能



DynamoDBSessionHandlerの利用

```
<?php
require '/home/ec2-user/vendor/autoload.php';
use Aws¥Common¥Aws;
use Aws¥DynamoDb¥Exception¥DynamoDbException;
use Aws¥DynamoDb¥DynamoDbClient;
use Aws¥DynamoDb¥Session¥SessionHandler;

$aws = Aws::factory('aws-config.php');
$client = $aws->get('dynamodb');

try {
    $sessionHandler = SessionHandler::factory(array(
        'dynamodb_client' => $client,
        'table_name'      => 'sessions',
    ));
    $sessionHandler->register();
} catch (DynamoDbException $e) {
    echo '*** Error ***' . "¥n";
    echo $e->getMessage();
}
session_start();
//セッションにデータ登録
$_SESSION['username'] = 'jeremy';
$_SESSION['role'] = 'admin';
session_commit();
```

boto(Python)

botoとは

- Python用のAWS SDKのデファクトスタンダード
- 最新版は2.19.0 (2013年12月現在)
- Python 2.6.6, 2.7.3 on Mac OSX and Ubuntu Maverick で動作確認している。
- Python 2.5での互換性もなるべく考慮しているが保証外
- Python 3.x版はDeveloper Preview中

操作可能サービス

Direct Connect	Elasticache	IAM
EC2	SimpleDB	OpsWorks
ELB	S3	Elastic Transcoder
Auto Scaling	Glacier	SQS
EMR	CloudFront	SNS
Route53	Storage Gateway	SES
VPC	Import/Export	SWF
DynamoDB	Elastic Beanstalk	CloudSearch
RDS	Cloud Formation	Data Pipeline
Redshift	Cloud Watch	CloudTrail
FPS	EBS	

botoに関するリソース

- ソースコードリポジトリ
  - https://github.com/boto/boto
- PyPI
  - http://pypi.python.org/pypi/boto
- オンラインドキュメント
  - http://docs.pythonboto.org/
- IRC
  - http://webchat.freenode.net/?channels=boto

インストール

```
$ pip install boto
```

amazon web services

初期設定

- 設定ファイルの作成
- /etc/boto.cfg - グローバルな設定を記述
- ~/.boto ユーザー毎の設定を記述

```
[Boto]
debug = 0
num_retries = 10

[DynamoDB]
region = ap-northeast-1
```

サンプル：S3 – set_contents_from_string

```
>>> from boto.s3.connection import S3Connection
>>> conn = S3Connection()
>>> bucket_name = ”yourBucket"
>>> bucket =  conn.get_bucket(bucket_name)
>>> from boto.s3.key import Key
>>> k = Key(bucket)
>>> k.key = "test.txt"
>>> k.set_contents_from_string("Hello World!")
```

(*) 事前にBucketが作成されている必要があります。

サンプル : S3 – get_contents_as_string

```
>>> from boto.s3.connection import S3Connection
>>> conn = S3Connection()
>>> bucket_name = "yourBucket"
>>> bucket =  conn.get_bucket(bucket_name)
>>> from boto.s3.key import Key
>>> k = Key(bucket)
>>> k.key = "test.txt"
>>> k.get_contents_as_string()
```

(*) 事前にBucketが作成されている必要があります。

サンプル：SQS – write

```
>>> import boto.sqs
>>> conn = boto.sqs.connect_to_region("ap-northeast-1")
>>> q = conn.get_queue('YourQueue')
>>> from boto.sqs.message import Message
>>> m = Message()
>>> m.set_body('Hello World!!')
>>> q.write(m)
```

(*) 事前にQueueが作成されている必要があります。

サンプル：SQS – get_messages

```
>>> import boto.sqs
>>> conn = boto.sqs.connect_to_region("ap-northeast-1")
>>> q = conn.get_queue('YourQueue')
>>> rs = q.get_messages()
>>> m = rs[0]
>>> m.get_body()
'Hello World!!'
```

(*) 事前にQueueが作成されている必要があります。

サンプル：DynamoDB – putItem

```
>>> from boto.dynamodb2.items import Item
>>> from boto.dynamodb2.table import Table
>>> table = Table('table_name')
>>> item = Item(table , data={
... 'id': 100,
... 'timestamp': 130699342,
... 'message' : 'Good Morning.',
... })
>>> item.save()
True
```

(*) 事前にTableが作成されている必要があります。

サンプル：DynamoDB – getItem

```
>>> from boto.dynamodb2.table import Table
>>> table= Table('table_name')
>>> item = table.get_item(id=100,timestamp=130699342)
>>> print item['message']
Good Morning.
```

(*) 事前にTableが作成されている必要があります。

その他Tips

- AWSコマンドラインインタフェース(CLI)もbotoを一部利用している

AWS SDK for Ruby

AWS SDK for Rubyとは

- Amazon提供のAWS開発用のRuby向けSDK
- 現在のバージョン
  - Version 1 (stable)
  - Version 2 (developer preview)
- http://aws.amazon.com/sdkforruby/
- 環境：Ruby 1.8.7 以降

操作可能サービス

Direct Connect	Elasticache	IAM
EC2	SimpleDB	OpsWorks
ELB	S3	Elastic Transcoder
Auto Scaling	Glacier	SQS
EMR	CloudFront	SNS
Route53	Storage Gateway	SES
VPC	Import/Export	SWF
DynamoDB	Elastic Beanstalk	CloudSearch
RDS	Cloud Formation	Data Pipeline
Redshift	Cloud Watch	

amazon web services

利用方法

- **Ruby環境のセットアップ**
  - Rbenvを利用する
    - https://github.com/sstephenson/rbenv
  - RVMを利用する
    - https://rvm.io/

- aws-sdk gemをインストール

  $ gem install aws-sdk

設定の初期化

- デフォルトのリージョンをセッティング

```
require 'aws’

AWS.config({
:region => ‘YOUR_DEFAULT_REGION’
})
```

amazon web services

サンプル：S3 - write

API仕様: http://docs.aws.amazon.com/AWSRubySDK/latest/AWS/S3/S3Object.html#write-instance_method

```
#!/usr/bin/env ruby
require 'aws'

                                                    ①
AWS.config({
  :access_key_id =>
ENV['AWS_ACCESS_KEY_ID'],
  :secret_access_key =>
ENV['AWS_SECRET_ACCESS_KEY'],
  :region => ENV['AWS_REGION']
})
                                                    ②
s3 = AWS::S3.new
bucket = s3.buckets['your-bucket']
obj = bucket.objects['key']

obj.write(Pathname.new('/path/to/file.txt'))  ③
```

① コンフィグを指定

② S3クライアント作成し、bucketやkeyを指定

③ putするfile名を指定し、s3にput

(*) 事前にBucketが作成されている必要があります。

サンプル：S3 - read

API仕様: http://docs.aws.amazon.com/AWSRubySDK/latest/AWS/S3/S3Object.html#read-instance_method

```
#!/usr/bin/env ruby
require 'aws'

AWS.config({                                    ①
  :access_key_id =>
ENV['AWS_ACCESS_KEY_ID'],
  :secret_access_key =>
ENV['AWS_SECRET_ACCESS_KEY'],
  :region => ENV['AWS_REGION']
})

s3 = AWS::S3.new                                ②
bucket = s3.buckets['your-bucket']
obj = bucket.objects['key']

obj.read do |chunk|                             ③
  puts chunk
  #=> data of s3://your-bucket/key
end
```

① コンフィグを指定

② S3クライアント作成し、bucketやkeyを指定

③ s3にあるobjectのデータを取得

(*) 事前にBucketが作成されている必要があります。

サンプル：SQS – send_message

API仕様: http://docs.aws.amazon.com/AWSRubySDK/latest/AWS/SQS/Client.html#send_message-instance_method

```
#!/usr/bin/env ruby
require 'aws'
require 'pp'

AWS.config({                                    ①
  :access_key_id =>
ENV['AWS_ACCESS_KEY_ID'],
  :secret_access_key =>
ENV['AWS_SECRET_ACCESS_KEY'],
  :region => ENV['AWS_REGION']
})

sqs = AWS::SQS.new                              ②

queue_url = 'https://your_queue'                ③
result = sqs.client.send_message({
  :queue_url    => queue_url,
  :message_body => 'Send Message!'
})
pp result
```

① コンフィグを指定

② SQSクライアント作成

③ send_message実行

queue_urlを指定、送りたいメッセージを指定

(*) 事前にQueueが作成されている必要があります。

サンプル：SQS – receive_message

API仕様: http://docs.aws.amazon.com/AWSRubySDK/latest/AWS/SQS/Client.html#receive_message-instance_method

```
#!/usr/bin/env ruby
require 'aws'
require 'pp'

AWS.config({                                    ①
  :access_key_id =>
ENV['AWS_ACCESS_KEY_ID'],
  :secret_access_key =>
ENV['AWS_SECRET_ACCESS_KEY'],
  :region => ENV['AWS_REGION']
})

sqs = AWS::SQS.new                              ②

queue_url = 'https://your_queue'                ③
result = sqs.client.receive_message({
  :queue_url    => queue_url
})
pp result
```

① コンフィグを指定

② SQSクライアント作成

③ receive_message実行

queue_urlを指定

(*) 事前にQueueが作成されている必要があります。

サンプル：DynamoDB – put_item

API仕様: http://docs.aws.amazon.com/AWSRubySDK/latest/AWS/DynamoDB/Client.html#put_item-instance_method

```
#!/usr/bin/env ruby
require 'aws'
require 'pp'

AWS.config({
  :access_key_id => ENV['AWS_ACCESS_KEY_ID'],
  :secret_access_key => ENV['AWS_SECRET_ACCESS_KEY'],
  :region => ENV['AWS_REGION']
})

client = AWS::DynamoDB::Client.new

result = client.put_item({
  :table_name => 'table_name',
  :item => {
    'id'        => {:n => '100'},
    'timestamp' => {:n => '130699342'},
    'message'   => {:s => 'Good Morning'}
  }
})
pp result
#=> {"ConsumedCapacityUnits"=>1.0}
```

① コンフィグを指定

② DynamoDBクライアント作成

③ put_item実行
- :table_name: テーブル名
- :item: Putするアイテムを指定

（例）
id : Hash Key
timestamp : Range Key
message : Attribute

(*) 事前にTableが作成されている必要があります。

サンプル：DynamoDB – get_item

API仕様: http://docs.aws.amazon.com/AWSRubySDK/latest/AWS/DynamoDB/Client.html#get_item-instance_method

```
#!/usr/bin/env ruby
require 'aws'
require 'pp'

AWS.config({
  :access_key_id => ENV['AWS_ACCESS_KEY_ID'],
  :secret_access_key => ENV['AWS_SECRET_ACCESS_KEY'],
  :region => ENV['AWS_REGION']
})

client = AWS::DynamoDB::Client.new

result = client.get_item({
  :consistent_read => true,
  :table_name => 'table_name',
  :key => {
    :hash_key_element => {:n => '100'},
    :range_key_element => {:n => '130699342'}
  }
})
pp result
#=> {"Item"=>{"id"=>{"N"=>"100"},
"message"=>{"S"=>"Good Morning"},
"timestamp"=>{"N"=>"130699342"}},
"ConsumedCapacityUnits"=>1.0}
```

① コンフィグを指定

② DynamoDBクライアント作成

③ get_item実行
- :table_name テーブル名
- :consistent_read 一貫性
- :key GutするKeyを指定

（例）
id : Hash Key
timestamp : Range Key

(*) 事前にTableが作成されている必要があります。

その他Tips

- 新しいRuby SDK (version 2)
  - Developer Preview
  - バリデーション、ドキュメント、拡張性など改善
  - プラグイン方式をとっている
  - Ruby 1.9以降が必須

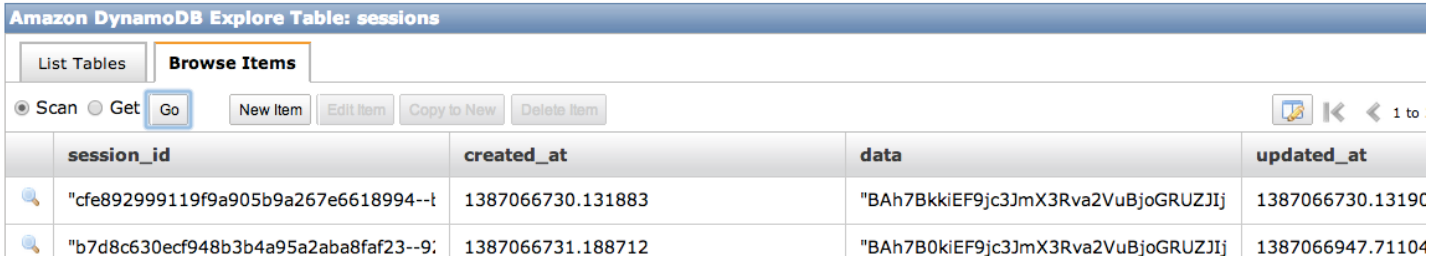
- 以下のように利用可能

```
$ gem install aws-sdk-core
AWS::DynamoDB.new(api_version: '2012-08-10')
```

Re:Inventで発表された新Ruby SDKのスライドはおすすめ

その他Tips

- RailsのセッションをDynamoDBに保存
  - スケーラブルかつ耐障害に優れたセッションストア
  - Rails 3.x か 4.x から利用可能
  - Rackベースのアプリも利用可能



- 以下の手順で利用可能

```
gem ‘aws-sessionstore-dynamodb’
$ bundle install
```

参考：https://github.com/aws/aws-sessionstore-dynamodb-ruby

amazon web services

AWS SDK for JavaScript in Node.js

AWS SDK for JavaScript in Node.jsとは

- Amazon提供のAWS開発用のnode.js環境で動くJavaScript向けSDK
  - http://aws.amazon.com/sdkfornodejs/

操作可能サービス

Direct Connect	Elasticache	IAM
EC2	SimpleDB	OpsWorks
ELB	S3	Elastic Transcoder
Auto Scaling	Glacier	SQS
EMR	CloudFront	SNS
Route53	Storage Gateway	SES
VPC	Import/Export	SWF
DynamoDB	Elastic Beanstalk	CloudSearch
RDS	Cloud Formation	Data Pipeline
Redshift	Cloud Watch	

amazon web services

利用方法

- **Node.js環境のセットアップ**
  - OSのパッケージマネージャーでnpmをインストール
  - Mac の場合：

    $ brew install npm

- aws-sdk packageをインストール

  $ npm install aws-sdk

サンプル：S3 - putObject

```
var AWS = require('aws-sdk');
var fs = require('fs');

var s3 = new AWS.S3();                                          ①

var bodyStream = fs.createReadStream( '/path/                   ②
to/file.txt' );
var params = {
  Bucket: 'your-bucket',
  Key: 'key',
  Body: bodyStream
};

s3.putObject(params, function(err, data) {                      ③
  if (err)
    console.log(err);
  else
    console.log("Successfully uploaded.");
});
```

① S3クライアント作成

② put時の引数
- Bucket: s3 bucket
- Key: s3 key
- Body: 実際のデータ

③ S3へPUT

(*) 事前にBucketが作成されている必要があります。

サンプル：S3 - getObject

```
var AWS = require('aws-sdk');
var fs = require('fs');

var s3 = new AWS.S3();                                    ①

var params = {                                            ②
  Bucket: 'your-bucket',
  Key: 'key'
};

s3.getObject(params, function(err, data) {                ③
  if (err)
    console.log(err);
  else
    console.log(data.Body.toString());
});
```

① S3クライアント作成

② gut時の引数
- Bucket: s3 bucket
- Key: s3 key

③ S3からデータ取得

(*) 事前にBucketが作成されている必要があります。

サンプル：SQS - sendMessage

```
var AWS = require('aws-sdk');
var fs = require('fs');

var s3 = new AWS.SQS();                                    ①

var params = {                                             ②
  QueueUrl: 'https://your_queue',
  MessageBody: 'Send Message!'
}
sqs.sendMessage(params, function(err, data) {
  if (err)
    console.log(err);
  else
    console.log("Successfully sent.");
})
```

① SQSクライアント作成

② queue_urlを指定、送りたいメッセージを指定

③ sendMessage実行

(*) 事前にQueueが作成されている必要があります。

サンプル：SQS - receiveMessage

```
var AWS = require('aws-sdk');
var fs = require('fs');

var s3 = new AWS.SQS();                                    ①

var params = {
  QueueUrl: 'https://your_queue'                           ②
}
sqs.receiveMessage(params, function(err, data)
{                                                          ③
  if (err)
    console.log(err);
  else
    console.log(data);
})
```

① SQSクライアント作成

② queue_urlを指定

③ receiveMessage実行

(*) 事前にQueueが作成されている必要があります。

サンプル：DynamoDB – putItem

```
var AWS = require('aws-sdk');
var fs = require('fs');

var ddb = new AWS.DynamoDB();                                  ①

var params = {                                                 ②
  TableName: 'my_table',
  Item: {
    'id': {N: '100'},
    'timestamp': {N: '130699342'},
    'message': {S: 'Good Morning'}
  }
};

ddb.putItem(params, function(err, data) {                      ③
  if (err)
    console.log(err);
  else
    console.log(data);
});
```

① DynamoDBクライアント作成

② パラメータ指定
- :TableName: テーブル名
- :Item: Putするアイテムを指定

（例）
id：Hash Key
timestamp：Range Key
message：Attribute

③ Put実行

(*) 事前にTableが作成されている必要があります。

サンプル：DynamoDB – getItem

```
var AWS = require('aws-sdk');
var fs = require('fs');

var ddb = new AWS.DynamoDB();                                    ①

var params = {                                                   ②
  TableName: 'my_table',
  Key: {
    'id': {N: '100'},
    'timestamp': {N: '130699342'}
  }
};

ddb.getItem(params, function(err, data) {                        ③
  if (err)
    console.log(err);
  else
    console.log(data);
});
```

① DynamoDBクライアント作成

② パラメータ指定
- :TableName: テーブル名
- :Item: Putするアイテムを指定

（例）
id：Hash Key
timestamp：Range Key
message：Attribute

③ Get実行

(*) 事前にTableが作成されている必要があります。

まとめ

今回のまとめ

- AWSはプログラマブルなインフラ
  - ほぼすべてのサービスはマネジメントコンソールから利用できるが、自動化しようと思ったらプログラムから扱うのが必須
  - 自動化するとAWSは何倍も便利になる

- AWSのサービスはHTTP/HTTPSでREST/SOAP形式のAPIをサポート。
  例えば
  - EC2：Start/Stop/Terminate
  - S3：GetObject/PutObject
  - DynamoDB：Get/Put

- SDKはこれらのAPIを抽象化し、各言語からの利用を非常に簡単にしてくれる

Q&A