re:Generate

AWSマイスターシリーズ
AWS Client-side SDK
- Android, iOS and Javascript -

2014.2.5

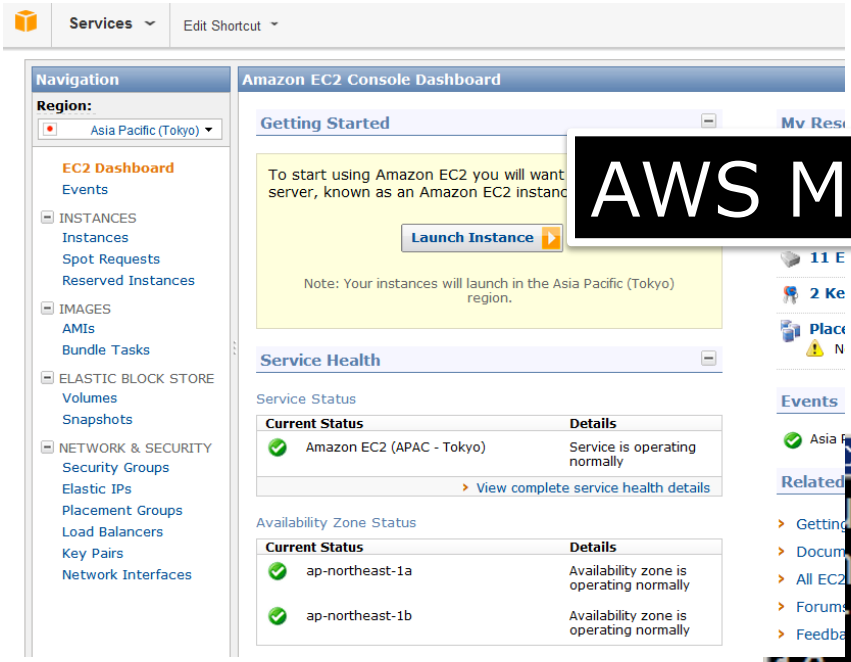
ソリューションアーキテクト
安川 健太　　今井 雄太

Agenda

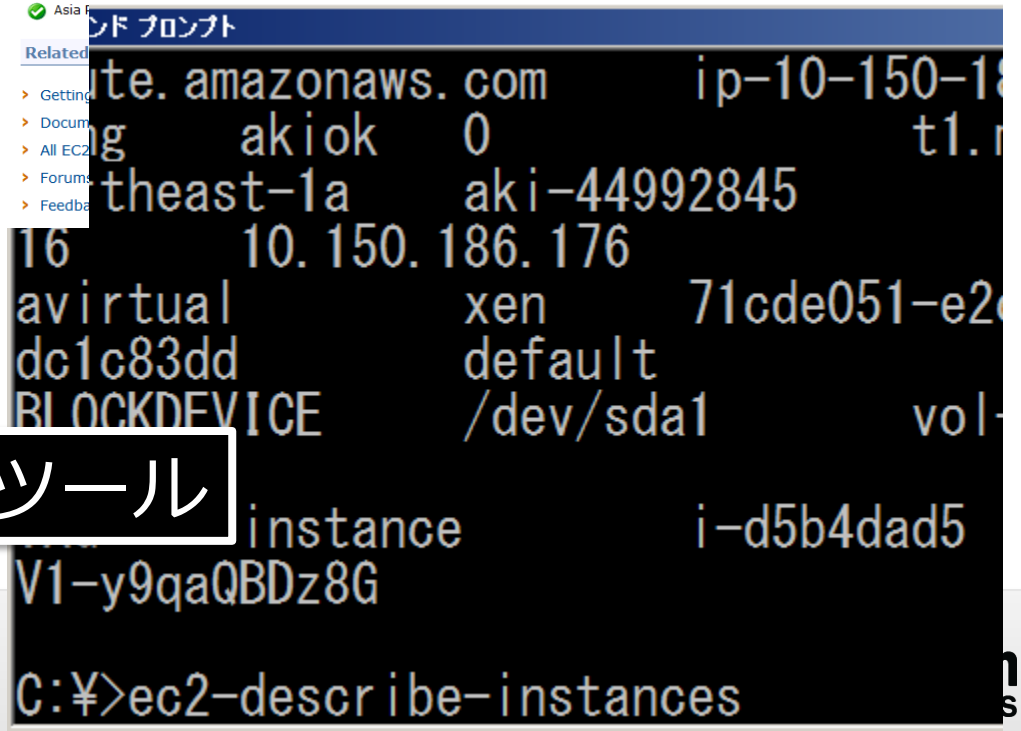
- AWS SDK オーバービュー
- クライアント側SDKの活用法
- クライアント側SDKにおけるAWS Credentialsの取り扱い
- 各クライアント側SDKの紹介
  - AWS SDK for Android
  - AWS SDK for iOS
  - AWS SDK for JavaScript in the Browser
- まとめ

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・・・

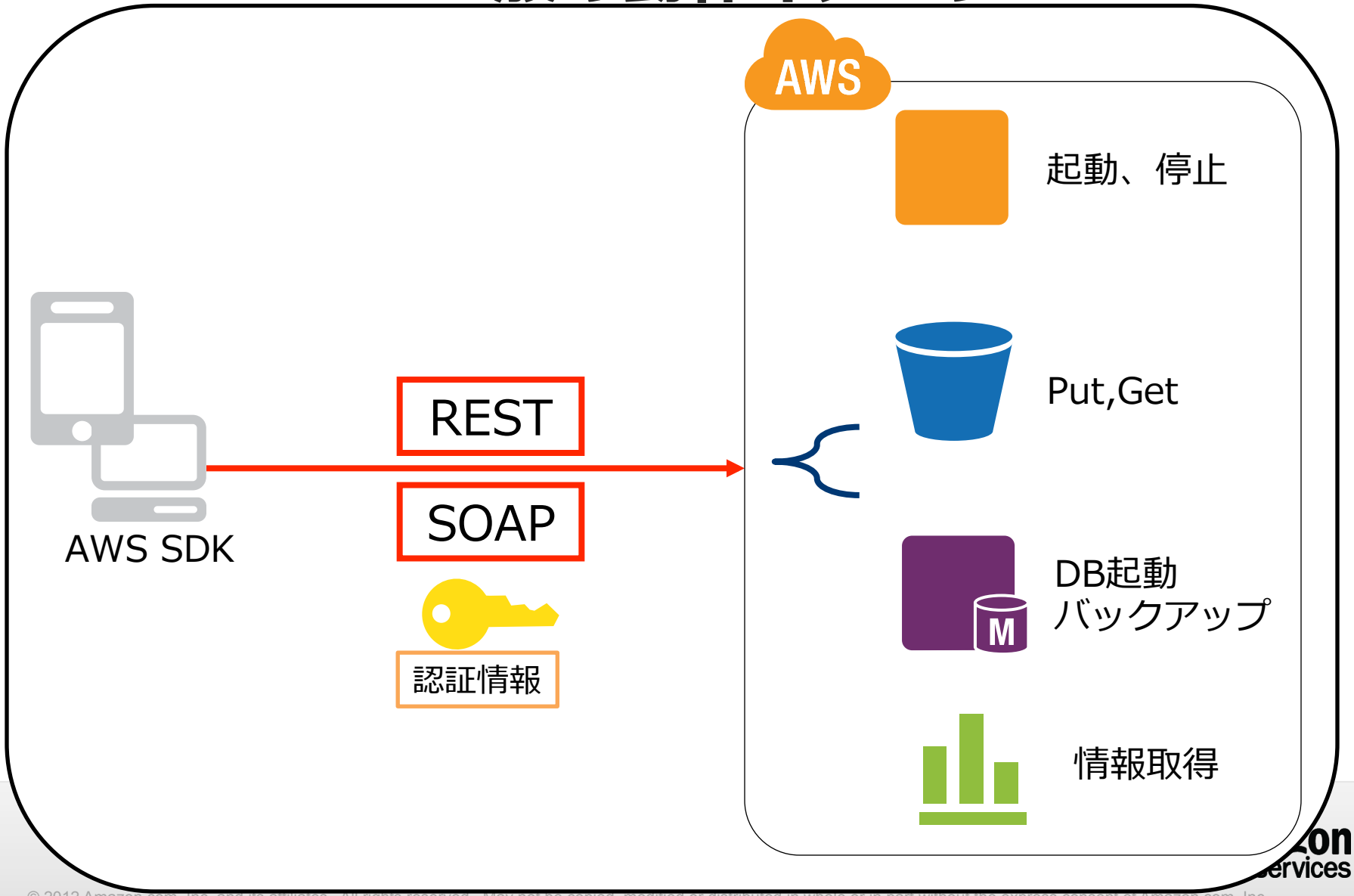
- 各サービスの各操作にAPIが定義されている
- AWSでは・・・
  - 人間がGUIまたはCUI越しに叩く
  - プログラミングしてそれを自動化・簡易化・カスタマイズ
    - 人間が手でやらなくてはいけない事をプログラミングして自動化できる、これがSDKで簡単に実現可能

    → 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
    - AWS SDK for Javascript in Browser
    - 有志の方による実装（ActionScript）も
  - 通信は原則HTTPS

SDK一般の動作イメージ



AWS SDKをある観点でざっくり分けると

開発者の環境（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

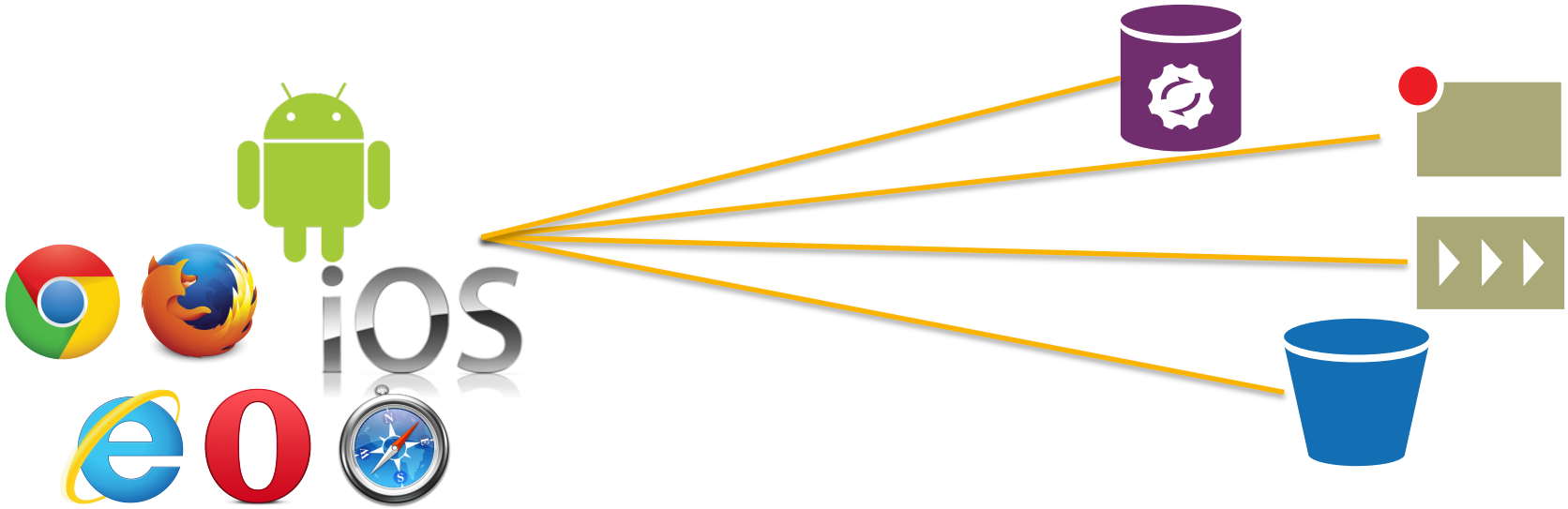

Javascript
in
Browser


クライアント側SDK

クライアント側SDKの活用法

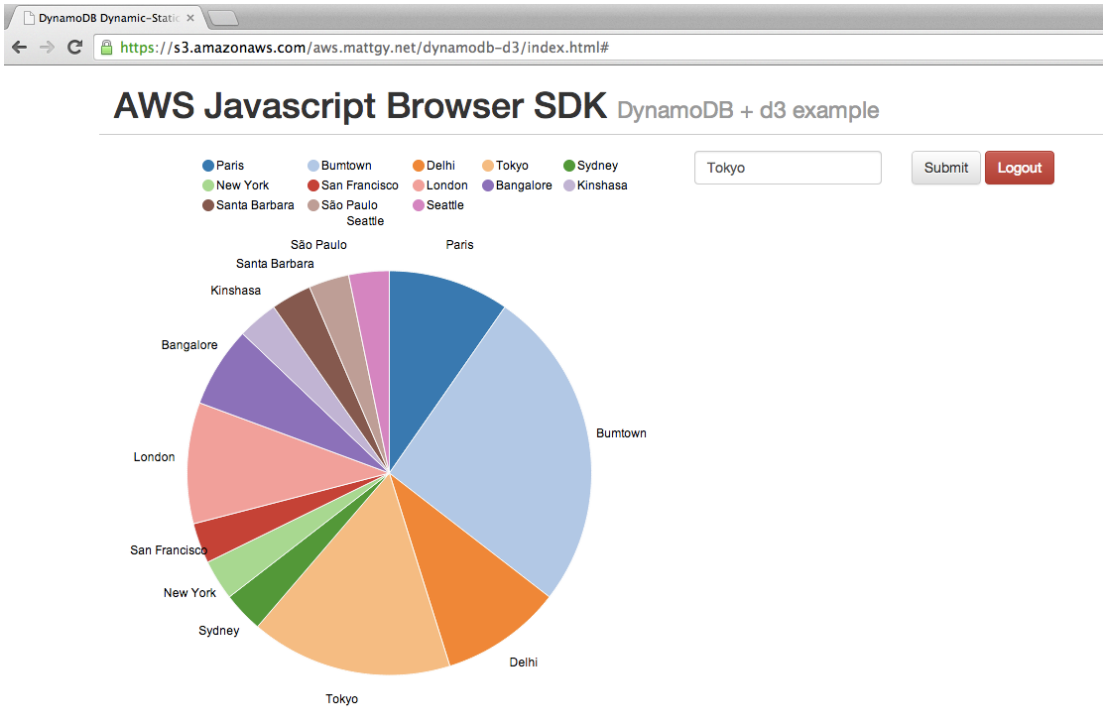
AWSのマネージドサービスを活用した2-tierのアーキテクチャが組める

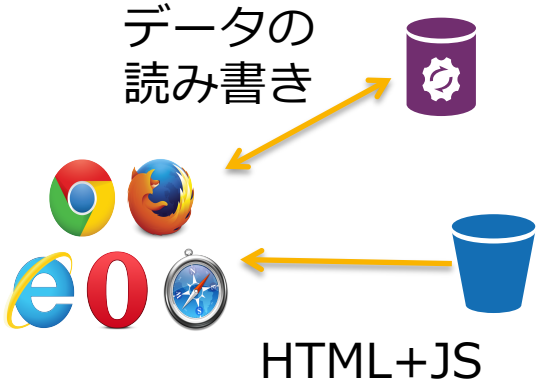
- モバイル端末やブラウザからAWSの各種サービスを呼ぶ
  - AWSのマネージドサービスを組み合わせてバックエンドに
  - しかもプラットフォーム横断で連携！



例1：静的ファイルだけで動的サイト

- DynamoDBやS3などをデータの保存先とするHTML+JSをS3に置けばWebサーバ無しで動的サイトを構築可





https://s3.amazonaws.com/aws.mattgy.net/dynamodb-d3/index.html

amazon web services

例2: モバイルアプリ間でメッセージング

- AWSサービスのみで作るチャットアプリ
  - サーバ側のコーディングなしで掲示板/チャットサービスを提供

AWS

DynamoDB
・メッセージの保存
・チャットグループとメッセージの紐付

Simple Notification Service (SNS)
・新規メッセージの通知

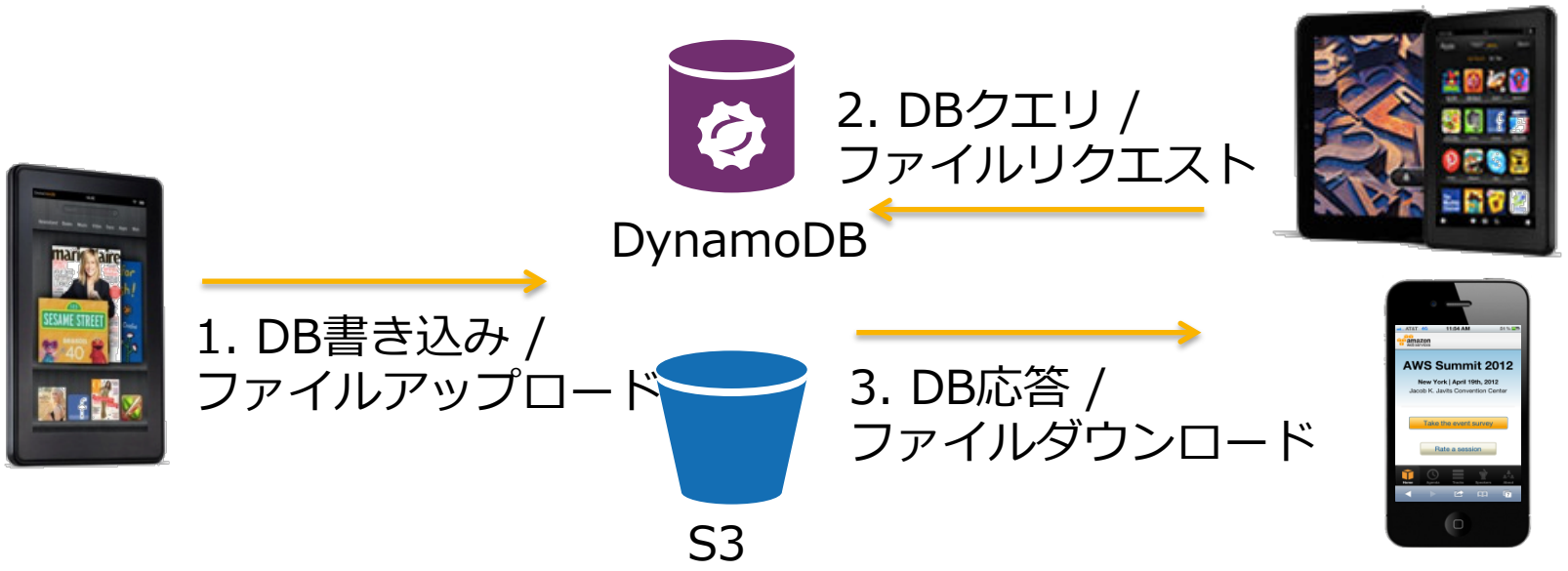
Simple Queue Service (SQS)
・SNSの通知受け口

Simple Storage Service (S3)
・画像等のデータの保存先・配信元

amazon web services

例2: モバイルアプリ間でメッセージング

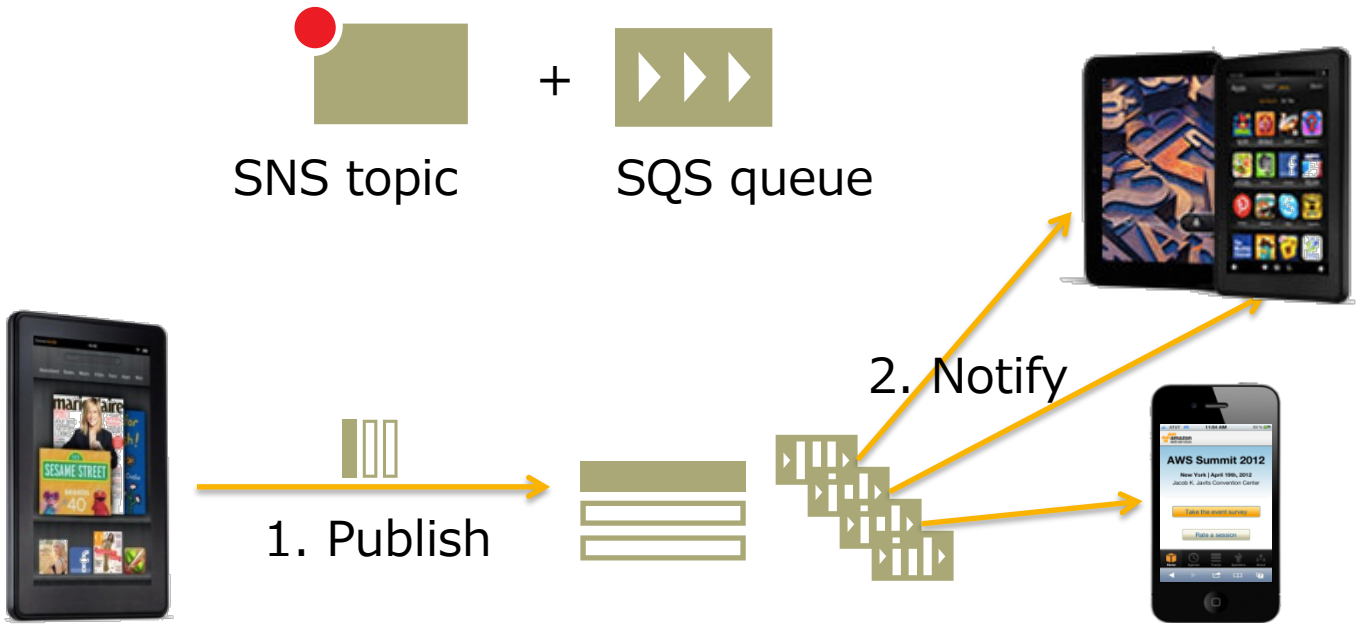
- DynamoDB, S3を使ってデータ共有



例2: モバイルアプリ間でメッセージング

- SNSとSQSを使ってPub/Sub

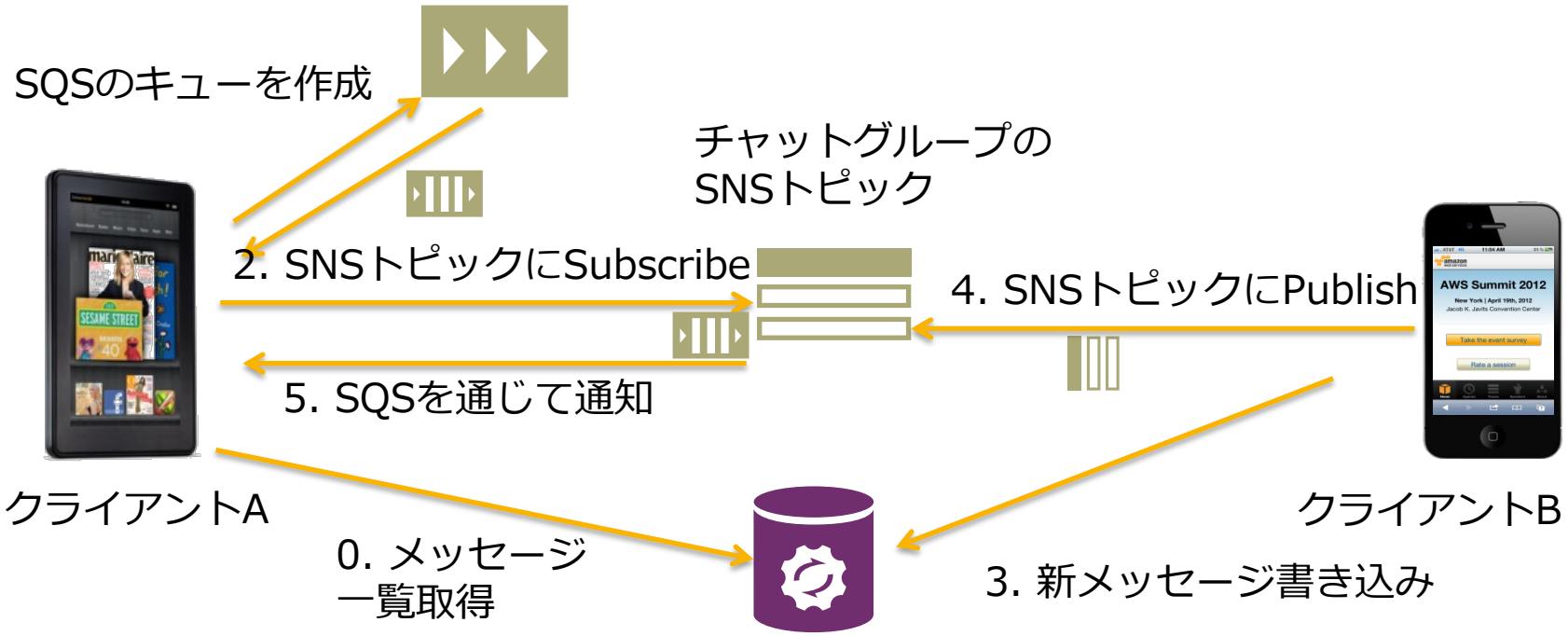
SNS: Simple Notification Service
SQS: Simple Queue Service



amazon web services

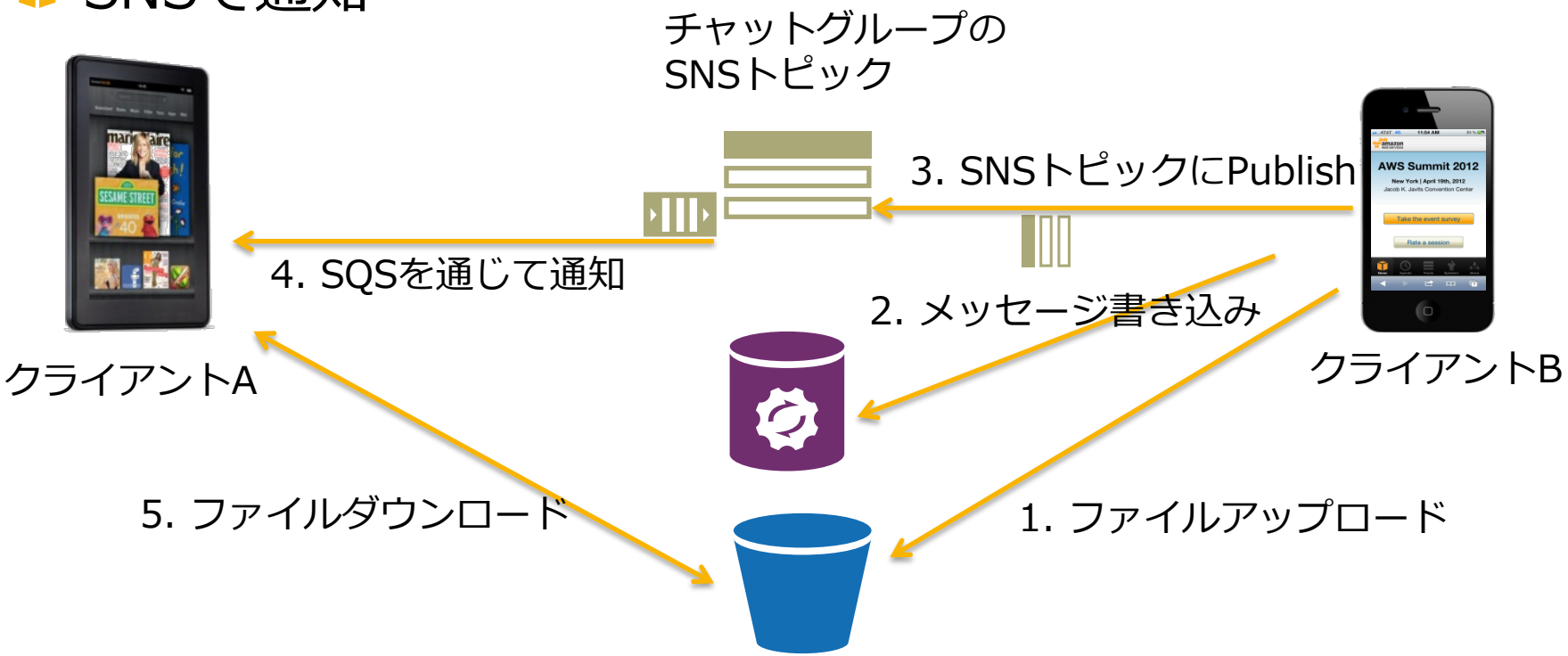
例2: モバイルアプリ間でメッセージング チャットグループへのSubscribe

- 過去メッセージはDynamoDBから取得
- SQSでキューを作成し、SNSにSubscribe



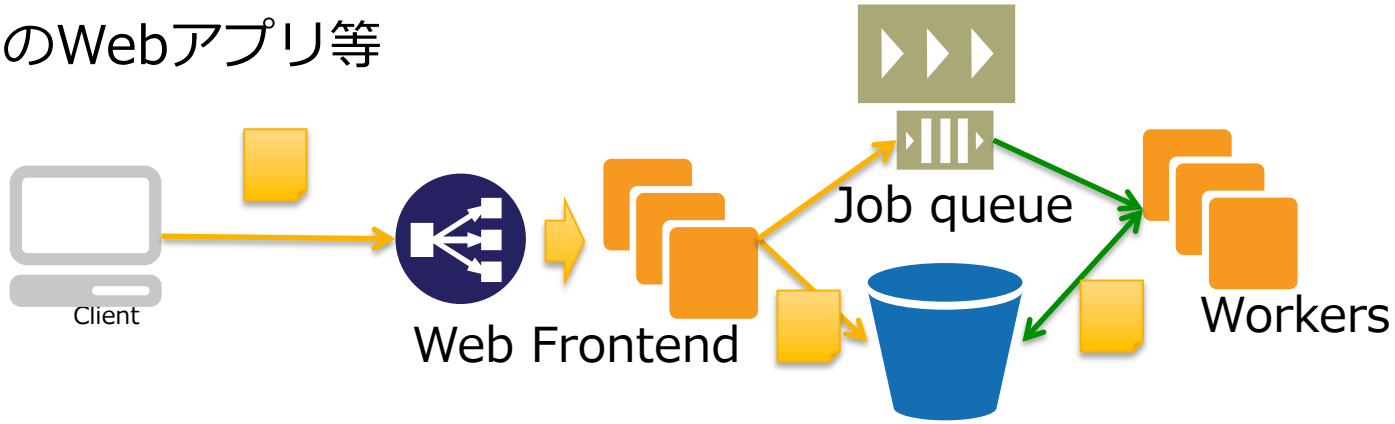
例2: モバイルアプリ間でメッセージング
チャットグループへのファイルアップロード

- S3にファイルをアップロード
- SNSで通知

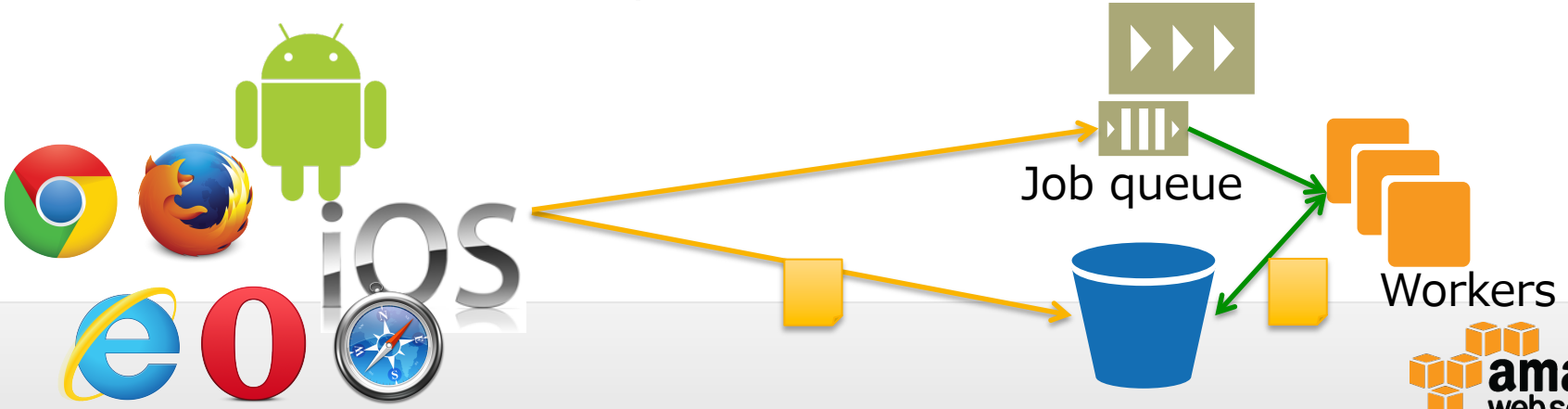


その他の例
非同期バッチ処理

- 非同期バッチ処理要求を直接登録
  - 通常のWebアプリ等



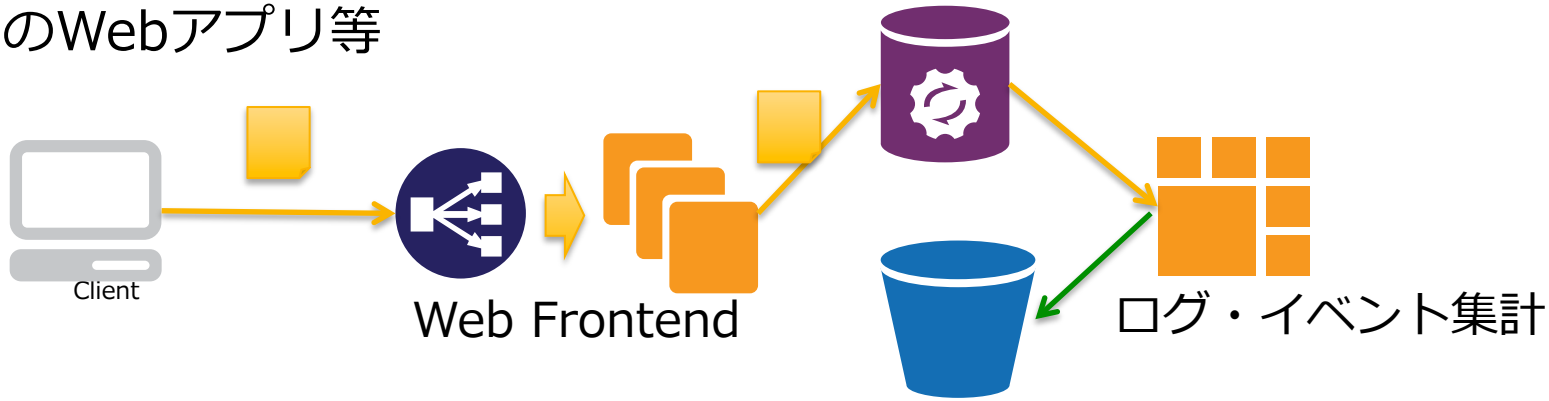
  - Android/iOS/Javascript SDKを使った場合



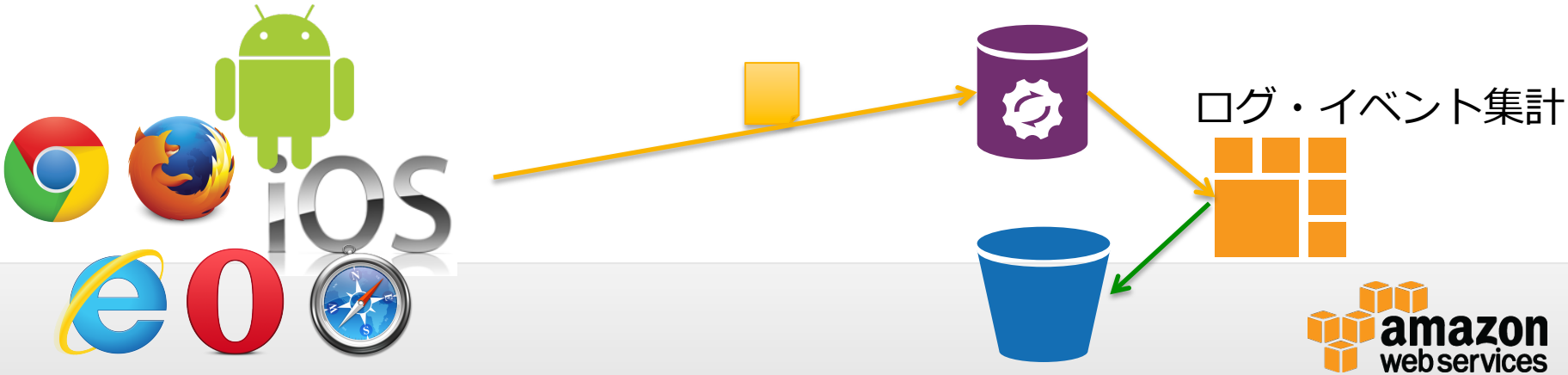
amazon web services

その他の例
ログやイベント情報の集計処理

- ログやイベント情報を直接登録
  - 通常のWebアプリ等

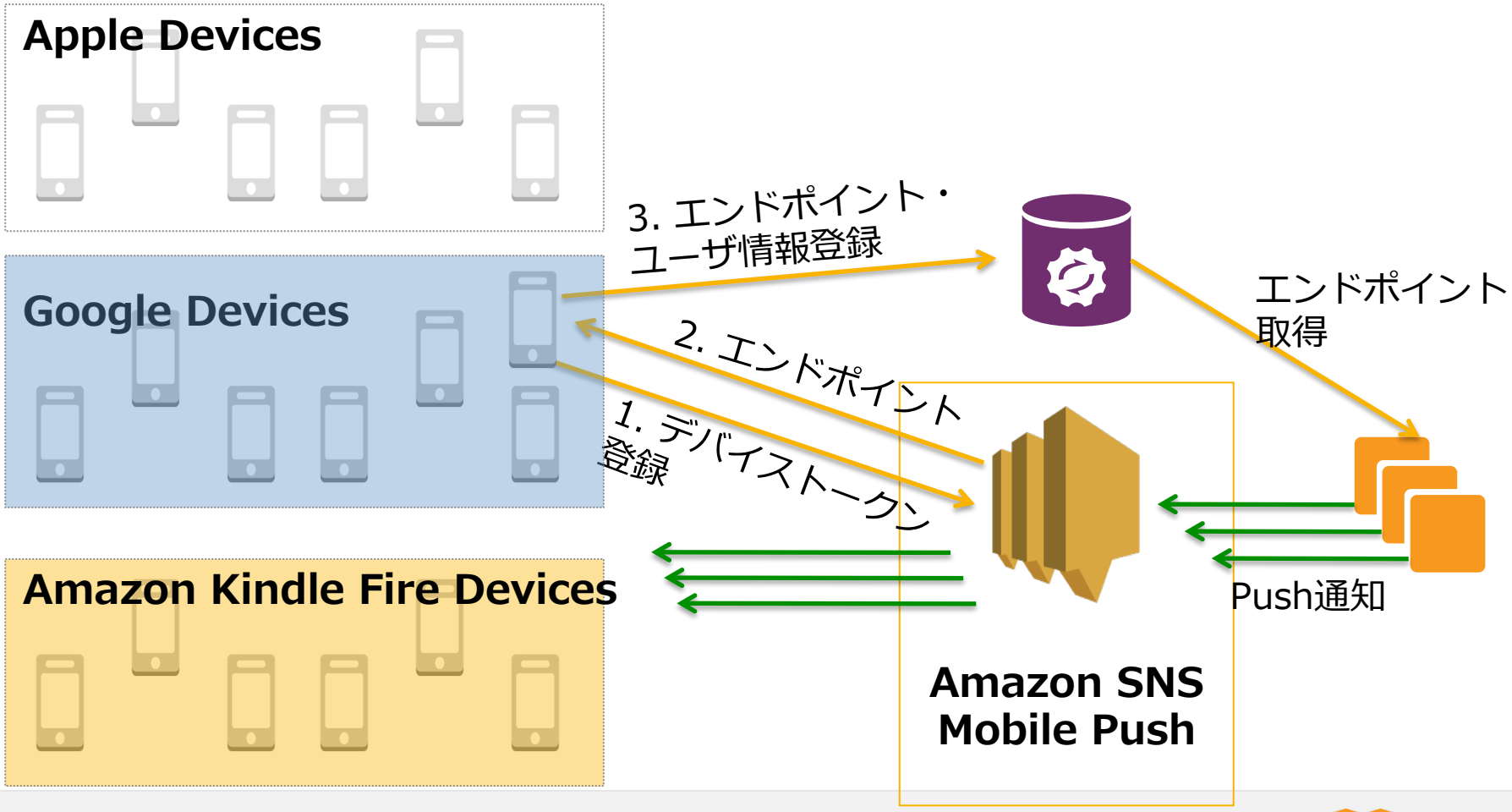


  - Android/iOS/Javascript SDKを使った場合



amazon web services

Amazon SNSへのデバイストークンの登録も直接可能



クライアント側SDK活用のメリット

- アプリの開発に多くのメリット：
  - バックエンド側の開発コストを最小化
  - バックエンド側の運用コストを最小化
  - スケーラビリティの心配なし
  - 金額面でもローコスト（当社比*）
- 必要に応じてEC2も導入できる安心感
  - 後からバックエンド側にロジックを入れてシステムの最適化

＊ EC2で同規模のサーバを立てる場合に比べ

amazon web services

AWS CREDENTIALSの取扱い

amazon web services

誰のAWSアカウントを使う？

- AWSのコンソールアプリを作る場合
  - エンドユーザはAWSユーザ
    →AWSのアクセスキーとシークレットキーを入力してもらえばOK
- AWSの各種サービスを使ったアプリを作る場合
  - AWSの各種サービスはあくまでバックエンド
  - エンドユーザは必ずしもAWSユーザではない
    →アプリは開発者のアカウントで認証・認可を受ける必要

開発者アカウントを使うとして…

- アプリに開発者アカウントのアクセスキー等を埋め込んだら
  - アクセスキーが広範にばら撒かれることに
  - アクセスキーの不正利用を止めるためにキーを無効化したら
    →全ユーザへのサービスが停止！！
- アクセスキーの定期的な更新で対処するにしても
  - 更新のたびにバージョンアップは非現実的
  - 更新前のアプリからはサービス利用不可に

セキュアなAWSアクセスを提供するには

- アプリに認証情報を埋め込むべきではない
- エンドユーザ/端末ごとに異なるCredentialsを提供すべき
  - ユーザごとに必要最小限の権限を与えるのは重要
  - 不正利用発覚時に不正ユーザのみ権限を停止
- Credentialsは期限が来たら無効化されるべき
  - 不正ユーザの影響も期限付きに

これを実現する仕組みあります！

amazon web services

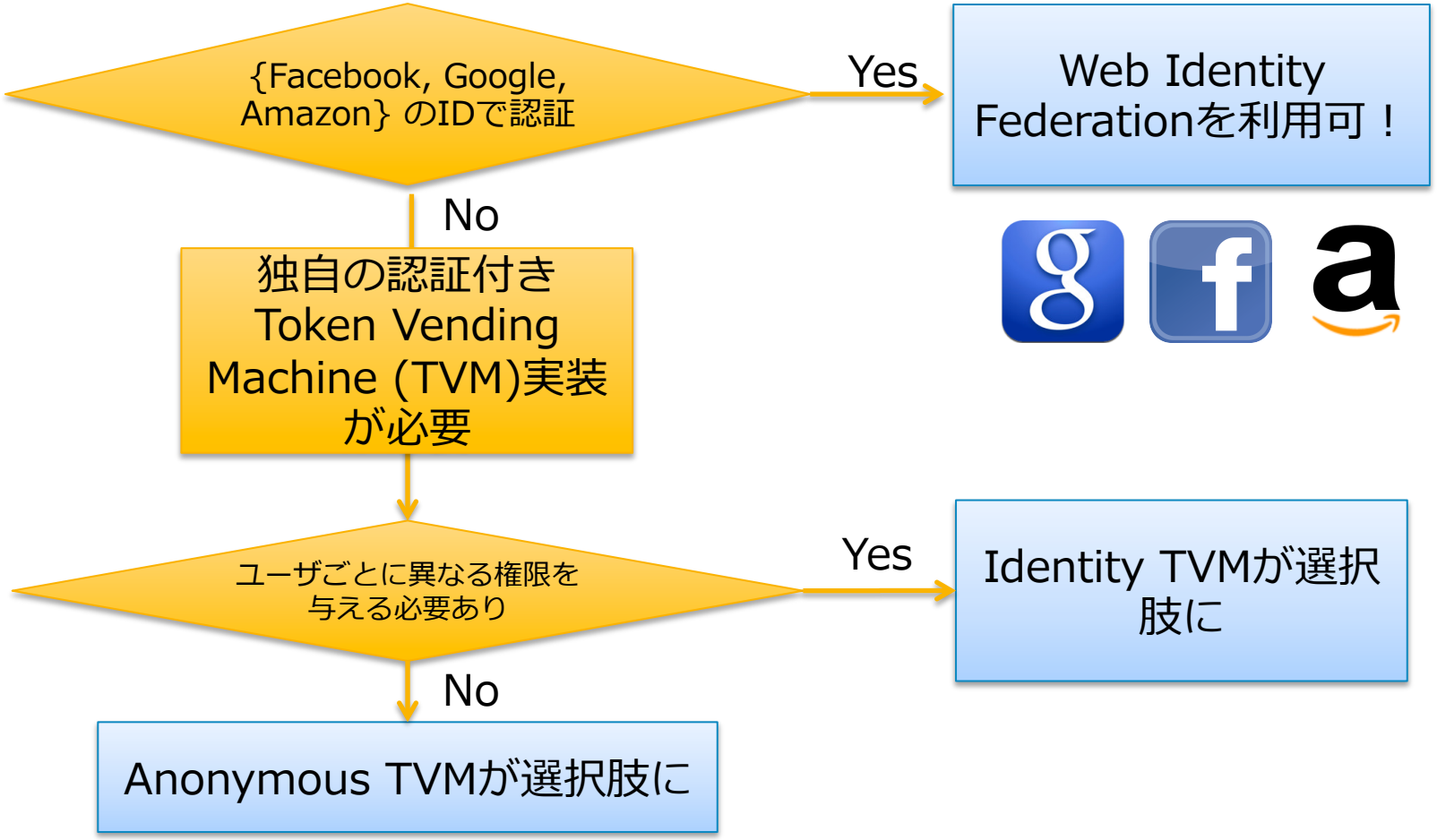
Security Token Service

- **AWSに対する、一時的な認証情報を作成する仕組み**
  - 期限付きの認証情報（認証チケット）を発行
  - Identity and Access Management (IAM)サービスの機能
- **ユーザーに対して、以下の3つのキーを発行**
  - アクセスID
  - シークレットキー
  - セッショントークン
- **作成した認証情報の有効期限設定が可能**
  - デフォルト１２時間　最小1時間　最大36時間
  - ただし延長・短縮は出来ない

但し、STS自体は認証の機能を持たないため、認証の仕組みは別途必要

amazon web services

認証機構とSTSとを インテグレーションする際の選択肢

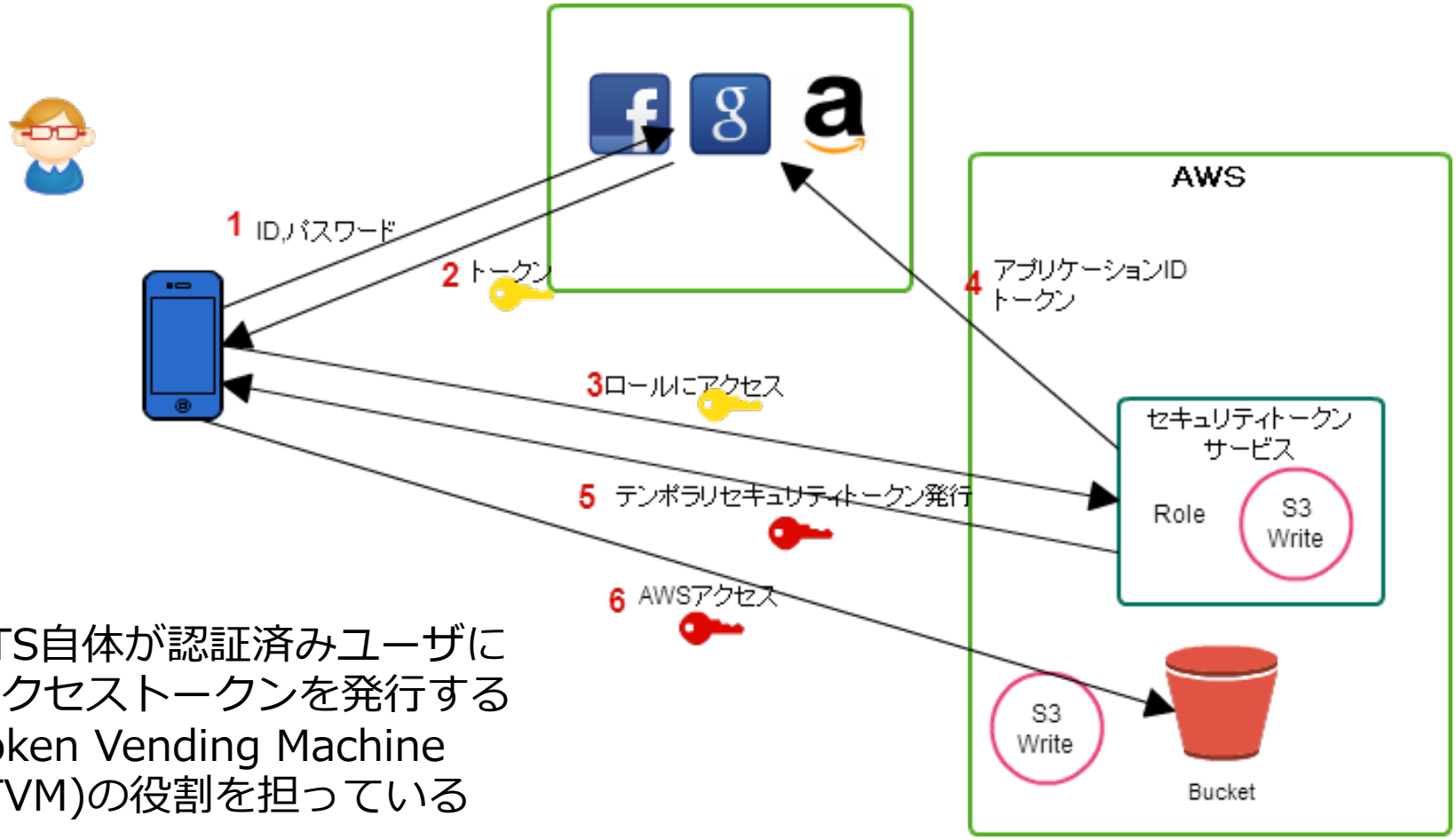


Web Identity Federation

- Web Identity Providerの認証結果を受けてAWSの各種サービスへのアクセスを認可する仕組み
- IAM RoleのAssume機能を利用して実現
  - Web Identity Providerによって認証されたユーザは特定のIAM Roleを持つものと見なす（Assume）する
- 対応Identity Providerは
  - Facebook
  - Google
  - Amazon



Web Identity Federationの動作フロー



STS自体が認証済みユーザに
アクセストークンを発行する
Token Vending Machine
(TVM)の役割を担っている

詳細はマイスターシリーズIAM回を参照：
http://www.slideshare.net/AmazonWebServicesJapan/20130716-aws-meisterregenerateiampublic

Web Identity Federation利用時のコード例

- Android

```
WebIdentityFederationSessionCredentialsProvider wif =
        new WebIdentityFederationSessionCredentialsProvider(fbSession.getAccessToken(),
                                                            "graph.facebook.com",
                                                            ROLE_ARN);
String subjectFromWIF = wif.getSubjectFromWIF();
s3 = new AmazonS3Client(wif);
```

- iOS

```
AmazonWIFCredentialsProvider *wif = [[AmazonWIFCredentialsProvider alloc]
            initWithRole:ROLE_ARN
            andWebIdentityToken:self.session.accessTokenData.accessToken
                        fromProvider:@"graph.facebook.com"];

NSString *subjectFromWIF = wif.subjectFromWIF;
s3 = [[[AmazonS3Client alloc] autorelease] initWithCredentialsProvider:wif];
```

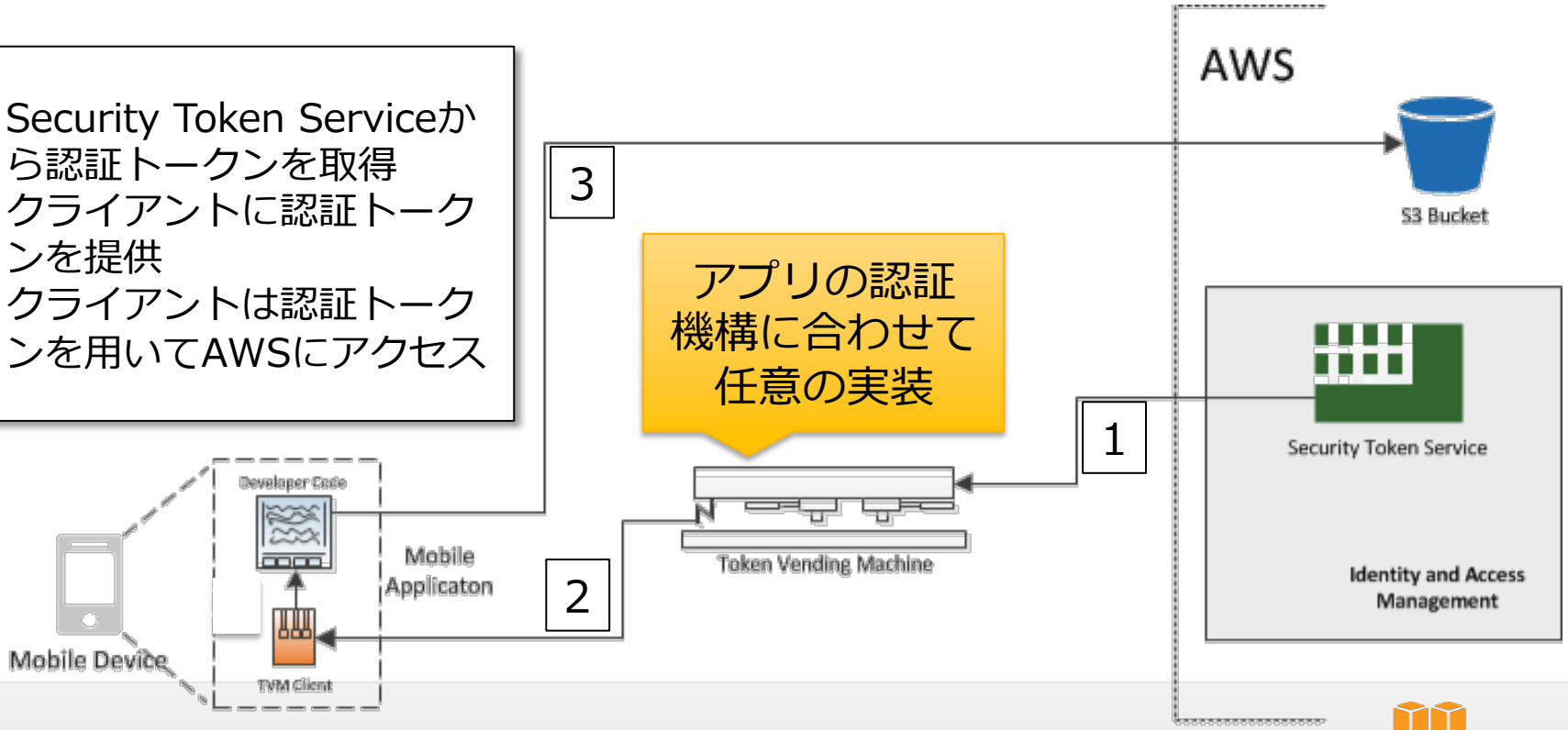
amazon web services

独自の認証機構を用いる場合

Token Vending Machine (TVM)を導入

- ユーザ/端末の認証とトークンの発行を実施
- アプリケーションごとの認証とSTSを結びつけるための仕組み

1. Security Token Serviceから認証トークンを取得
2. クライアントに認証トークンを提供
3. クライアントは認証トークンを用いてAWSにアクセス



TVMの実装方針を大別すると

- AnonymousTVM
  - ユーザ認証を必要とせず、一時認証トークンの機能のみ利用したいユースケース向け
  - 例：誰でも参加出来る掲示板システム、ログの記録

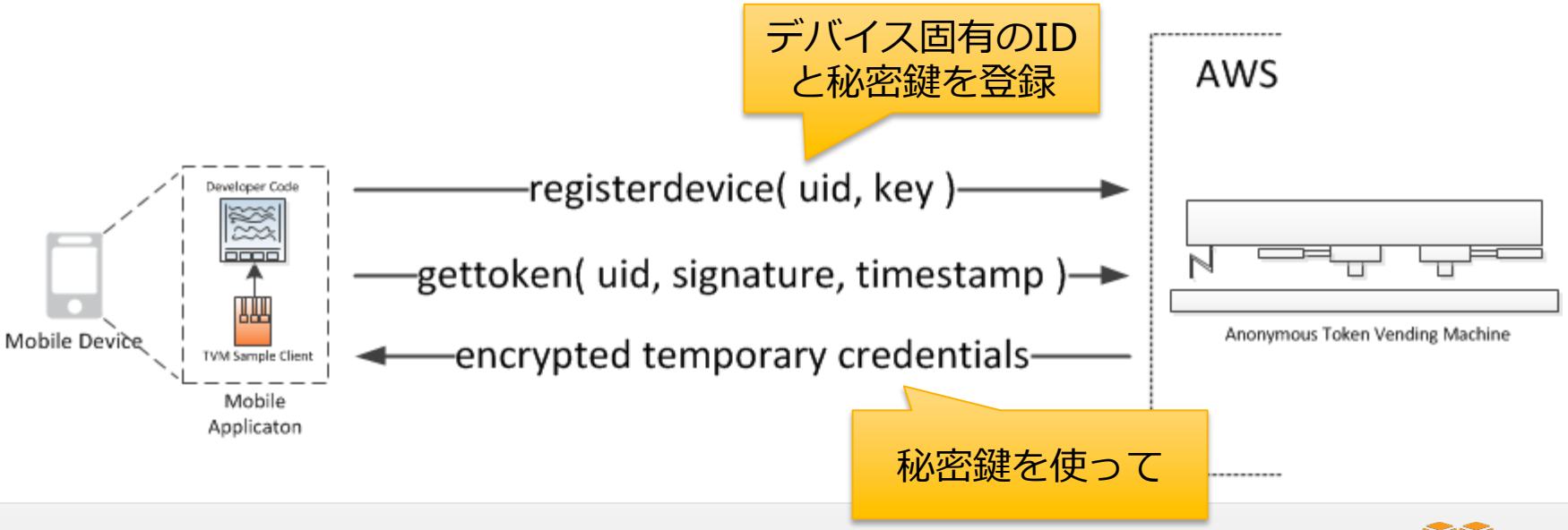
- IdentityTVM
  - ユーザ認証を実施し、認証されたユーザにのみ認証トークンを発行するユースケース向け
  - 例：ユーザ認証を伴うソーシャルアプリ、ゲーム等

どちらもサンプルの実装がサーバ側・クライアント側とも公開されている

amazon web services

Anonymous TVM

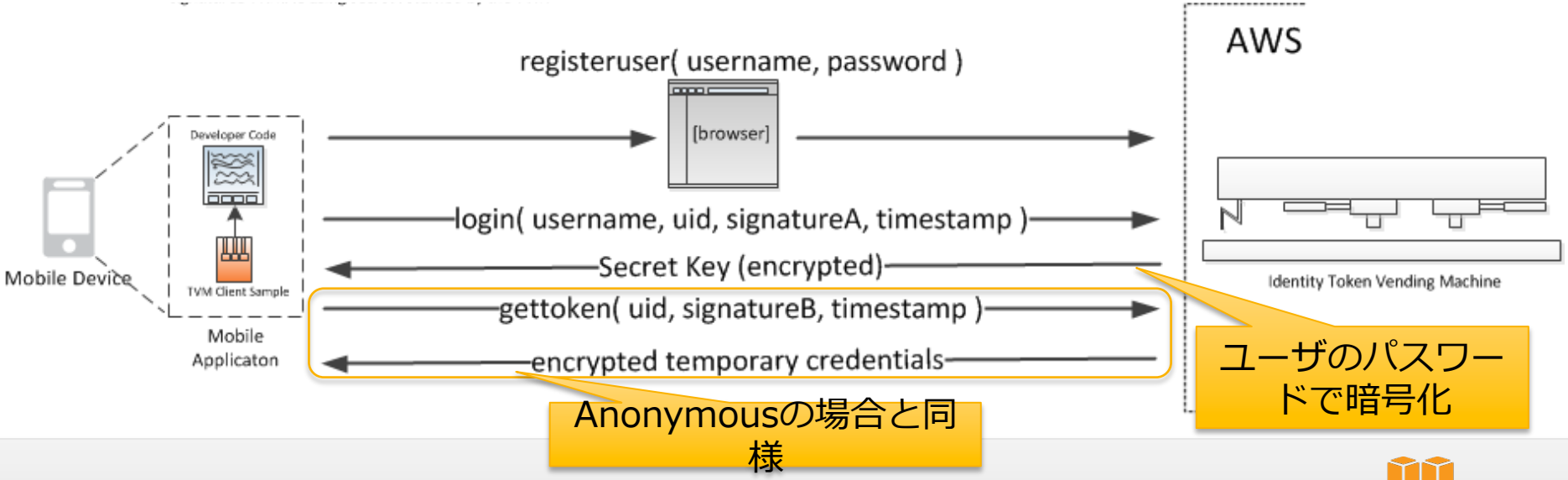
- デバイス固有のIDと対応する秘密鍵をTVMに登録
- 秘密鍵を使って認証トークンを暗号化して提供



Identity TVM

- Out-of-bandでユーザ登録
- 登録したユーザ情報でログイン処理
- TVM用秘密鍵の共有
- 秘密鍵を使って認証トークンを暗号化して提供

Facebook, Twitter, LDAPやAD等の基幹システムによる認証で置き換え可



Web Identityを用いた Fine Grained Access Control

- 例1: S3バケット内にユーザごとのプライベートな領域を作成

```
{ "Version":"2012-10-17",
  "Statement":[
  { "Effect":"Allow",
    "Action":["s3:GetObject", "s3:PutObject", "s3:DeleteObject"],
    "Resource":[
       "arn:aws:s3:::myBucket/myApp/${graph.facebook.com:id}",
       "arn:aws:s3:::myBucket/myApp/${graph.facebook.com:id}/*“
    ]
  } ]
}
```

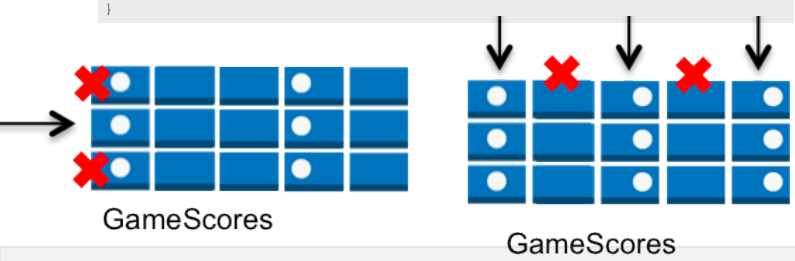
Facebookにログインしたユーザーの**ID**が入る

Web Identityを用いた Fine Grained Access Control

- 例2: DynamoDBのテーブル内にユーザごとにプライベートな行/列を定義する

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "dynamodb:GetItem",
            "dynamodb:BatchGetItem",
            "dynamodb:Query",
            "dynamodb:PutItem",
            "dynamodb:UpdateItem",
            "dynamodb:DeleteItem",
            "dynamodb:BatchWriteItem"
         ],
         "Resource": [
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition": {
            "ForAllValues:StringEquals": {
               "dynamodb:LeadingKeys":  ["${www.amazon.com:user_id}"],
               "dynamodb:Attributes": [
                  "UserId","GameTitle","Wins","Losses",
                  "TopScore","TopScoreDateTime"
               ]
            },
            "StringEqualsIfExists": {"dynamodb:Select": "SPECIFIC_ATTRIBUTES"}
         }
      }
   ]
}
```

```
"Condition": {
      "ForAllValues:StringEquals": {
        "dynamodb:LeadingKeys": [
          "${www.amazon.com:user_id}"
        ],
        "dynamodb:Attributes": [
          "UserId”,"GameTitle”, "Wins”, "Losses”
        ]
      },
      "StringEqualsIfExists": {
        "dynamodb:Select": "SPECIFIC_ATTRIBUTES"
      }
    }
```



各クライアント側SDKの紹介

AWS SDK FOR ANDROID

AWS SDK for Androidとは

- Amazon提供のAWS開発用のAndroid向けSDK
- 公式ページ：http://aws.amazon.com/jp/sdkforandroid/
- 環境：Android 2.3 (API level 10）以上
- 利用方法
  - 公式ページからダウンロード
  - Gitレポジトリから

Mavenをお使いの場合：
公式Mavenレポジトリ Localレポジトリにインストールしての利用は可（但し、正式サポートの範囲外）

amazon web services

操作可能サービス

EC2	S3
DynamoDB	SimpleDB
SNS	SQS
SES	ELB
CloudWatch	Autoscaling

※サポートするサービスの範囲はiOS SDKと同様

SDKに含まれるもの

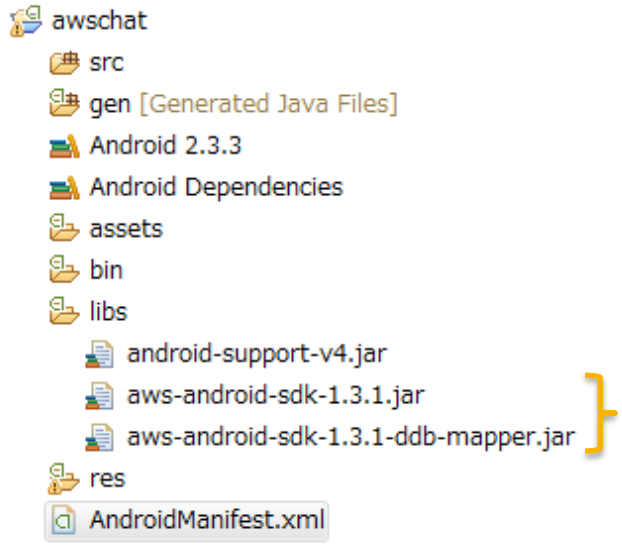
- **AWS Android ライブラリ**
  - 各種サービス向けAPIを提供するJavaライブラリ
  - Androidアプリのプロジェクトから参照
  - 最終的にはAPKに内包
- **ソースコード**
- **ドキュメント**

（コードサンプルはGitレポジトリに移動）
https://github.com/awslabs/aws-sdk-android-samples

- 各種サービスの操作
- SNSとSQSを使ったメッセージボード
- S3アップローダ
- SimpleDB ハイスコア
- SESフィードバックフォーム
- DynamoDBを使ったユーザプリファレンス

初期設定

- プロジェクトの依存関係にAndroid SDKを追加
  - EclipseとAndroid SDKを使っている場合の例：



必要なライブラリをプロジェクトのlibs配下にコピー

注：他の場所に配置する場合は最終的なAPKにクラスファイルが含まれるようにすること

依存するJarファイルについてTips

- 開発時はデバッグシンボル付きのライブラリを使用

  aws-android-sdk-<version>-debug.jar
  aws-android-sdk-<version>-ddb-mapper-debug.jar

  DynamoDBMapper を使う場合に必要

- 配布時は必要なサービスのみ同梱

  全部入り

  aws-android-sdk-<version>.jar
  aws-android-sdk-<version>-ddb-mapper.jar

  必要なサービスのみ

  aws-android-sdk-<version>-core.jar

  +

  aws-android-sdk-<version>-ddb.jar
  aws-android-sdk-<version>-s3.jar
  aws-android-sdk-<version>-sns.jar
  aws-android-sdk-<version>-sqs.jar

  あるいはproguardを使って必要なパッケージのみ同梱

コード例

- 基本的な流れ（iOSも同様）
  - 利用するサービスのクライアントオブジェクトの作成
  - サービスのリクエストオブジェクトを作成
  - クライアントオブジェクトを通じてリクエスト
  - レスポンスを受け取る

S3にファイルをアップロードする例

```
void uploadToS3 (String bucketName, String objectName, File file){
   AmazonCredentials credentials =
            new BasicAmazonCredentials(ACCESS_KEY, SECRET_KEY);
   AmazonS3Client s3 = new AmazonS3Client(credentials);

   PutObjectRequest req = new PutObjectRequest(bucketName, objectName, file));

   PutObjectResponse resp = s3.putObject(req);
}
```

amazon web services

コード例：S3へのアップロード (非同期)

```
S3UploadTask uploadTask = new S3UploadTask();
uploadTask.execute(new PutObjectRequest(bucketName, objectName, tempFile));

private class S3UploadTask extends AsyncTask<PutObjectRequest, Long, Long>
        implements ProgressListener {
        protected Long totalSent;

        // AsyncTask#doInBackground()
        protected Long doInBackground (PutObjectRequest... reqs) {    totalSent = 0L;

        reqs[0].setProgressListener(this);

        S3.getInstance().putObject(reqs[0]);
        return totalSent;
        }

         // ProgressListener#progressChanged()
        public void progressChanged (ProgressEvent progressEvent) {   totalSent
        progressEvent.getBytesTransfered();       publishProgress(totalSent);
        }
}
```

バックグラウンド実行

進行状況の更新

コード例：DynamoDBへのQuery

ハッシュキーとレンジキーのMaxを指定して特定のテーブルにクエリを送る例

```
QueryResult doQuery (String hashKey, String max) {

    AttributeValue hashKey = new AttributeValue().withS(hashKey);

    Condition lessThanMax = new Condition()
            .withComparisonOperator(ComparisonOperator.LT)
            .withAttributeValueList(new AttributeValue().withN(max));

    QueryResult result = dynamoClient.query(new QueryRequest()
            .withTableName(TABLE_NAME)
            .withLimit(NUMBER_OF_ITEMS_TO_GET_AT_ONCE)
            .withHashKeyValue(hashKey)
            .withRangeKeyCondition(lessThanMax)
            );
    return result;
}
```

ハッシュキーを指定

条件を指定

クエリを実行

コード例：DynamoDBMapperで DynamoDBにオブジェクトを永続化

POJOを定義

```
@DynamoDBTable(tableName = “meisters_items”)
public class MeistersItem {

   private String id;
   private long value;

   @DynamoDBHashKey(attributeName = “id”)
   public String getId() {
      return id;
   }

   @DynamoDBHashKey(attributeName =  “id”)
   public void setId(String id) {
      this.id = id;
   }

   @DynamoDBAttribute(attributeName = “value”)
   public long getValue() {
      return time;
   }

   @DynamoDBAttribute(attributeName = “value”)
   public void setValue(long time) {
      this.time = time;
   }
}
```

DynamoDBMapperを初期化

```
AmazonDynamoDBClient dynamoClient =
            new AmazonDynamoDBClient(getCredentials());
AmazonDynamoDBMapper dbMapper =
            new DynamoDBMapper(getDynamoClient());
```

POJOの保存

```
MeisterItem item = new MeisterItem();
item.setId(“Expensive item”);
item.setValue(10000);
dbMapper.save(item);
```

POJOの読み込み

```
MeisterItem item = dbMapper.load(MeisterItem.class,
“Expensive item”);
```

amazon web services

Android SDK利用時のTips

- UIスレッドからAWSのAPIを呼ぶのは避ける
  - 適宜AsyncTaskや別Threadを使用
- 各サービスクライアントのインスタンスは再利用
  - サービスクライアントの実装はThread Safe
  - Singletonにすることで重いオブジェクト生成を最小限に
  - 例：サンプル内のAmazonClientManager

```
public class AmazonClientManager { // 下記コードは簡易版
    private AmazonS3Client s3Client = null;

    public AmazonS3Client s3() {
        if (s3Client == null) s3Client = new AmazonS3Client( credentials );
        return s3Client;
    }
}
```

- ユースケース別の解説付きコードサンプルがこちらに：
  - http://aws.amazon.com/articles/SDKs/Android

AWS SDK FOR IOS

AWS SDK for iOSとは

- Amazon提供のAWS開発用iOS SDK
- 公式ページ：
  http://aws.amazon.com/jp/sdkforios/
- 動作環境：iOS 4.3以上
- 開発環境：Xcode v4以上
- 利用方法
  - 公式ページからダウンロード
  - Gitレポジトリから

操作可能サービス

EC2	S3
DynamoDB	SimpleDB
SNS	SQS
SES	ELB
CloudWatch	Autoscaling

※サポートするサービスの範囲はAndroid SDKと同様

SDKに含まれるもの

- **AWS iOSライブラリ**
  - 各種サービス向けAPIを提供するObjective-Cライブラリ
  - iOSアプリプロジェクトのフレームワークの1つとして参照
- **ドキュメント**
- **ソースコード**

（コードサンプルはGitレポジトリに移動）

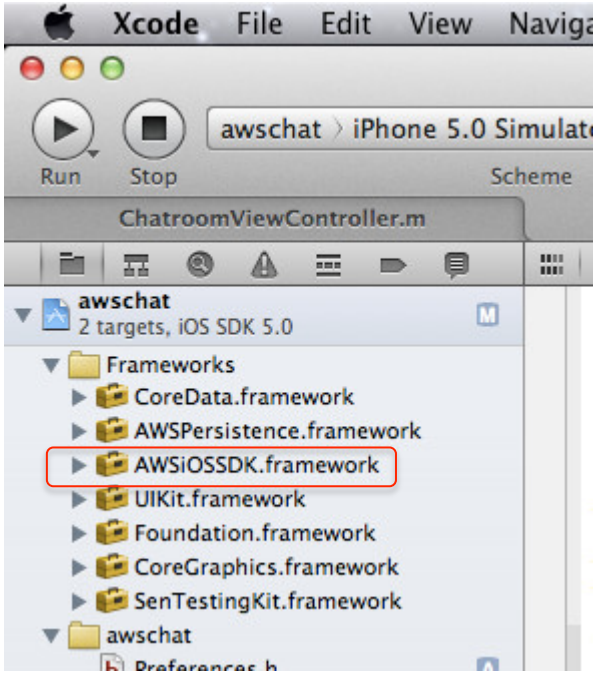
https://github.com/awslabs/aws-sdk-ios-samples

- 各種サービスの操作
- SNSとSQSを使ったメッセージボード
- S3アップローダ
- SimpleDB ハイスコア
- SESフィードバックフォーム
- DynamoDBを使ったユーザプリファレンス
- DynamoDB を CoreDataフレームワークのバックエンドに

初期設定
～Frameworkの追加～

- プロジェクトにSDKを追加
  1. Xcodeにてプロジェクトを開く
  2. Frameworks GroupをCtrlを押しながらクリック
  3. AWSiOSSDK.frameworkを追加
     - AWSPersistence.frameworkは任意（DynamoDB + CoreDataを使ったPersistenceを利用する場合）
  4. ソースコードで必要なヘッダファイルをインクルード
     - 例

```
#import <AWSiOSSDK/S3/AmazonS3Client.h>
#import <AWSiOSSDK/SimpleDB/AmazonSimpleDBClient.h>
#import <AWSiOSSDK/SQS/AmazonSQSClient.h>
#import <AWSiOSSDK/SNS/AmazonSNSClient.h>
```



初期設定
～ドキュメントのインストール～

- ドキュメントセットをXcodeにインストール
  - SDKのディレクトリからdocsetファイルをコピー
    - これを
      - Documentation/com.amazon.aws.ios.docset
    - ここに
      - $HOME/Library/Developer/Shared/Documentaion/DocSets

```
$ mkdir –p $HOME/Library/Developer/Shared/Documentaion/DocSets
$ cp Documentation/com.amazon.aws.ios.docset¥
$HOME/Library/Developer/Shared/Documentaion/DocSets
```

  - Xcodeを再起動

amazon web services

コード例

- 基本的な流れ （Androidも同様）
  - 利用するサービスのクライアントオブジェクトの作成
  - サービスのリクエストオブジェクトを作成
  - クライアントオブジェクトを通じてリクエスト
  - レスポンスを受け取る

S3にファイルをアップロードする例

```
- void uploadToS3: (NSString*) bucketName keyName: (NSString *) keyName fileName:
(NSString *) fileName {
   S3PutObjectRequest *putObjectRequest =
      [[[S3PutObjectRequest alloc] initWithKey:keyName inBucket:bucketName] autorelease];
    putObjectRequest.filename = fileName;

    [[AmazonClientManager s3] putObject:putObjectRequest];
}
```

amazon web services

コード例：S3へのアップロード（非同期）1/2

NSOperationのサブクラスとしてS3 Uploaderを実装

```
#import <Foundation/Foundation.h>
#import <AWSiOSSDK/AmazonServiceRequest.h>

@interface AsyncImageUploader:NSOperation<AmazonServiceRequestDelegate>
{
…
```

```
- (void) start {
….
    // Puts the file as an object in the bucket.
    S3PutObjectRequest *putObjectRequest =
      [[[S3PutObjectRequest alloc] initWithKey:keyName inBucket:bucketName] autorelease];
    putObjectRequest.filename = filename;
    putObjectRequest.delegate = self;

    [[AmazonClientManager s3] putObject:putObjectRequest];
}
```

Delegateとして途中経過や結果を受け取れるように登録

コード例：S3へのアップロード（非同期）2/2

Delegateパターンで途中経過や結果を受け取り

完了時の処理

```
-(void) request:(AmazonServiceRequest *)request
        didCompleteWithResponse:(AmazonServiceResponse *)response
{
    [self performSelectorOnMainThread:@selector(hideProgressView)
                           withObject:nil waitUntilDone:NO];

    [self finish];
}
```

進行状況の更新処理

```
-(void) request:(AmazonServiceRequest *)request
        didSendData:(NSInteger)bytesWritten
        totalBytesWritten:(NSInteger) totalBytesWritten
        totalBytesExpectedToWrite:(NSInteger) totalBytesExpectedToWrite
{
    [self performSelectorOnMainThread:@selector(updateProgressView:)
            withObject:
            [NSNumber numberWithFloat:
              (float)totalBytesWritten / totalBytesExpectedToWrite] waitUntilDone:NO];
}
```

コード例：DynamoDBへのQuery

```
-(DynamoDynamoDBQueryResponse *) doQuery: (NSString*) hashKey
                                      max: (NSString *) max {

    DynamoDBQueryRequest *req = [[DynamoDBQueryRequest alloc] autorelease];
    req.tableName = TABLE_NAME;
    req.limit = [[NSNumber alloc] initWithInt: NUMBER_OF_ITEMS_TO_GET_AT_ONCE];
    req.hashKeyValue = [[DynamoDBAttributeValue alloc] initWithS: hashKey] autorelease];

    DynamoDBCondition *lessThanMax = [[DynamoDBCondition alloc] autorelease];
    [req.rangeKeyCondition setComparisonOperator: @"LT"];
    [req.rangeKeyCondition addAttributeValueList:
                        [[[DynamoDBAttributeValue alloc] initWithN: max] autorelease]];
    req.rangeKeyCondition = lessThanMax;

    DynamoDBQueryResponse *resp = [[AmazonClientManager dynamodb] query: req];
    return resp;
}
```

ハッシュキーを指定

条件を指定

クエリを実行

iOS SDK利用時のTips

- UIスレッドからAWSのAPIを呼ぶのは避ける
  - 適宜NSOperationやGCD等を使用
- 各サービスクライアントのインスタンスは再利用
  - サービスクライアントの実装はThread Safe
  - 例：サンプル内のAmazonClientManager

```
#import "AmazonClientManager.h"
static AmazonS3Client       *s3  = nil;

@implementation AmazonClientManager
+(AmazonS3Client *)s3
{
    if(s3 == nil) s3  = [[AmazonS3Client alloc] initWithCredentials:credentials];
    return s3;
}
```

- ユースケース別の解説付きコードサンプルがこちらに：
  - http://aws.amazon.com/articles/SDKs/iOS

AWS SDK FOR JAVASCRIPT IN WEB BROWSER

AWS SDK for JavaScript in the Browserとは

- Amazon提供のブラウザで動作するJavaScript環境のためのAWS SDK
  - node.js環境のためのSDKは別に存在する
- 公式ページ：http://aws.amazon.com/jp/sdkforbrowser/
- 環境：

Google Chrome	28.0+	**Microsoft Internet Explorer**	10.0+
Mozilla Firefox	23.0+	**Apple Safari**	5.1+
Opera	17.0+	**Android Browser**	4.3+

- 利用方法
  - 公式ページからダウンロード
  - Gitレポジトリから

amazon web services

操作可能サービス

DynamoDB	S3
SQS	SNS

初期設定(1/2)

- HTML内にJavaScriptを読み込む
  - SDKをインターネット上の任意の場所に配置し、Scriptタグで読み込む
    - <script type="text/javascript" src=”PATH/TO/YOUR/SDK"></script>
  - AWSで用意するCDN上に配置されたSDKを使う
    - <script type="text/javascript" src="https://sdk.amazonaws.com/js/aws-sdk-2.0.0-rc1.min.js"></script>
- 読み込まれると、“AWS”というグローバル変数が利用可能になる

初期設定(2/2)

- WebIdentityFederationでAWS Credentialを取得

```
// FacebookのAccessTokenを使ってAWSのCredentialを取得する
window.fbAsyncInit = function() {
        FB.init({ appId: appId });
        FB.login(function(response) {
          AWS.config.credentials = new AWS.WebIdentityCredentials({
            ProviderId: 'graph.facebook.com',
            RoleArn: roleArn,    // IAM RoleのARN
            WebIdentityToken: response.authResponse.accessToken
          });
          fbUserId = response.authResponse.userID;
          button.style.display = 'block';
        });
};

// FacebookのSDK読み込み
(function(d, s, id){
    var js, fjs = d.getElementsByTagName(s)[0];
    if (d.getElementById(id)) {return;}
    js = d.createElement(s); js.id = id;
    js.src = "//connect.facebook.net/en_US/all.js";
    fjs.parentNode.insertBefore(js, fjs);
}(document, 'script', 'facebook-jssdk'));
```

コード例：S3へのアップロード

```
<input type="file" id="file-chooser" />
<button id="upload-button" style="display: none”>Upload to S3</button>

<script type="text/javascript”>
//AWS.config.credentialにCredentialが取得済みの状態で
var bucket = new AWS.S3({params: {Bucket: “YOUR_BUCKET_NAME”}});

button.addEventListener('click', function() {
        var file = fileChooser.files[0];
        if (file) {
          results.innerHTML = '';
          var objKey = “KEY_NAME”;
          var params = {Key: objKey, Body: file, ACL: 'public-read'};
          bucket.putObject(params, function (err, data) {
            if (err) {
              console.log(err);
            } else {
              console.log(data);
            }
          });
      }
}, false);
</script>
```

コード例：DynamoDBへのputItem

```
<button id="upload-button" style="display: none">Put to DynamoDB</button>

<script type="text/javascript”>
//AWS.config.credentialにCredentialが取得済みの状態で
var dynamodb = new AWS.DynamoDB({region:'ap-northeast-1'});

button.addEventListener('click', function() {
    var params = {
        TableName: 'test',
            Item: {
                'title': {S: appId + 'aaa'},
                'message': {S: 'Good Morning'}
        }
    };
   dynamodb.putItem(params, function(err, data) {
        if (err) {
                console.log(err);
        }else{
                console.log(data);
        }
    });
}, false);
</script>
```

まとめ

まとめ

- クライアント側SDKを用いて2-tierアーキテクチャを実現
  - バックエンド側の開発コストを最小化
  - バックエンド側の運用コストを最小化
  - スケーラビリティの心配なし
  - 金額面でもコスト削減に
- Web Identity Federationや認証トークン機構を使うことで
  クライアントアプリからのAWSへのアクセスも安心
- 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
  - その恩恵はサーバ側だけでなくクライアントアプリにも！
  - HAPPY CODING!!

ご参加ありがとう
ございました

TVM導入の流れ

- 認証トークンに割り当てるユーザ権限を設定
  - Identity and Access Management (IAM)でユーザを作成
  - アプリケーションに合わせたポリシーを割り当て

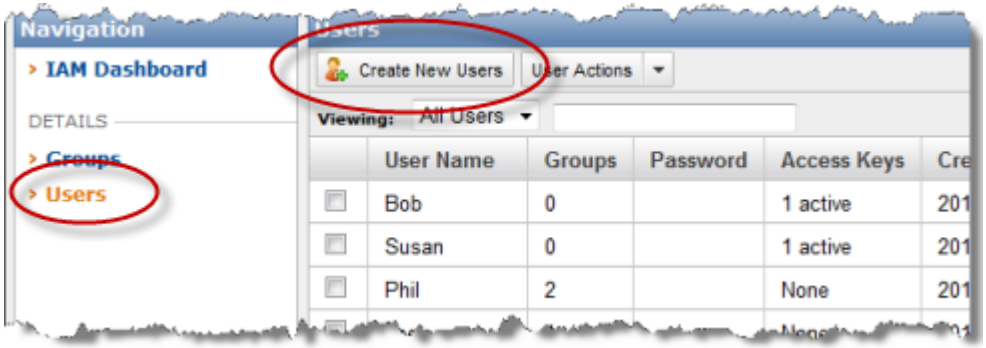
- TVMを実装し、デプロイ
  - サンプルはWARファイルをデプロイするだけで実行可
    - AnonymousTVM
    - IdentityTVM

  →ElasticBeanstalkを使うと簡単！

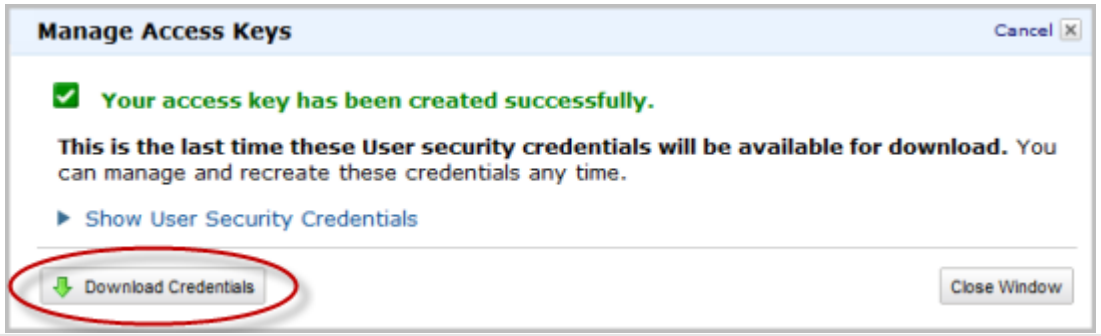
- TVMの各種設定
  - Security Token Serviceにアクセスするためのアクセスキー設定
  - SSLの設定（推奨）

認証トークンに割り当てるユーザ権限を設定 (1/2)

- IAMでTVM用の新しいユーザを作成（例：TVMUser）



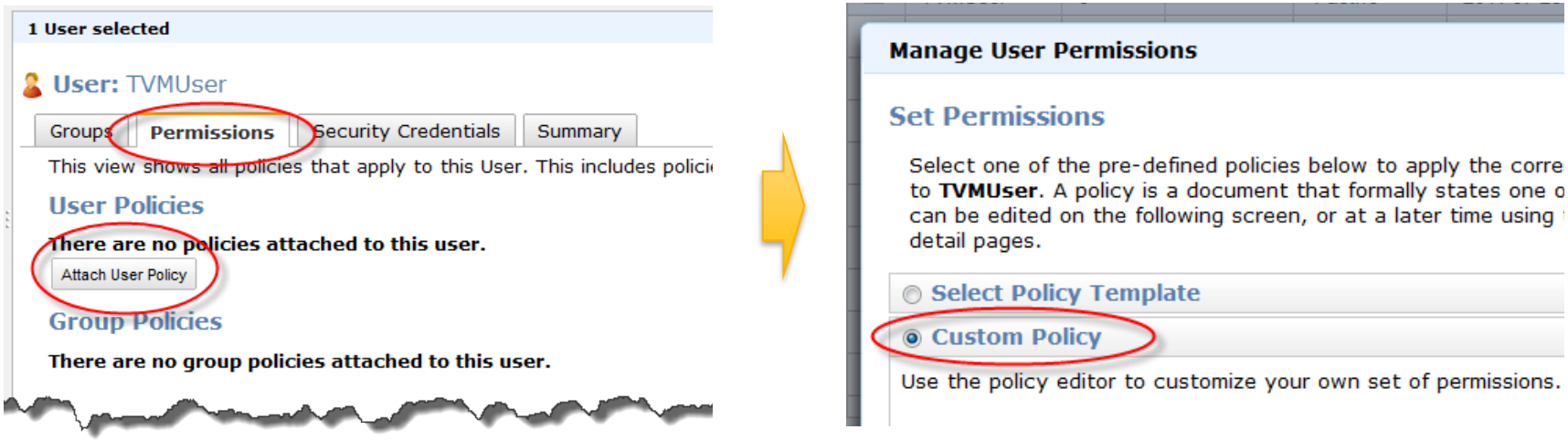
- 認証情報（アクセスキー・シークレットキー）を保存



後でTVMアプリケーション側に設定するので大切に保存

認証トークンに割り当てるユーザ権限を設定 (2/2)

- カスタムポリシーを作成して割り当て



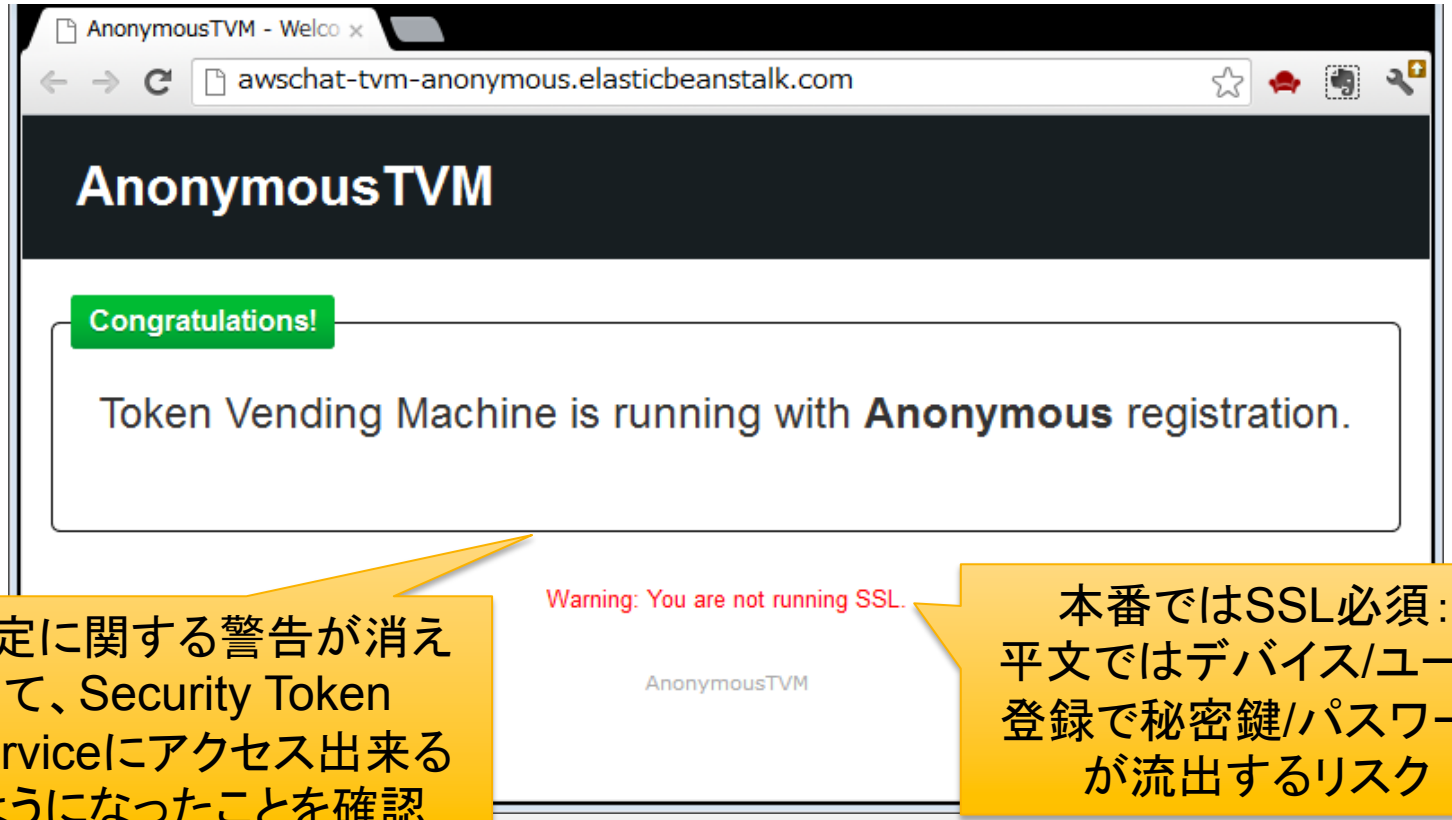
- カスタムポリシーの例 (TVMのサンプルに同梱)

```
{ "Statement": [
          { "Effect": "Allow", "Action": "sts:GetFederationToken", "Resource": "*" },
          { "Effect": "Allow", "Action": "iam:GetUser", "Resource": "*" },
          { "Effect": "Allow", "Action": "sdb:*", "Resource": "*" },
          { "Effect":"Allow", "Action":"dynamodb:*", "Resource":"*" },
          { "Effect": "Allow", "Action": "sqs:*", "Resource": "*" },
          { "Effect": "Allow", "Action": "s3:*", "Resource": "*" },
          {"Effect": "Allow", "Action": "sns:*", "Resource": "*" }
          ]
}
```

アプリケーションで必要最小限な権限になるよう更新

TVMがデプロイされたことを確認

- TVMをデプロイしたHTTPコンテナにアクセス



設定に関する警告が消えて、Security Token Serviceにアクセス出来るようになったことを確認

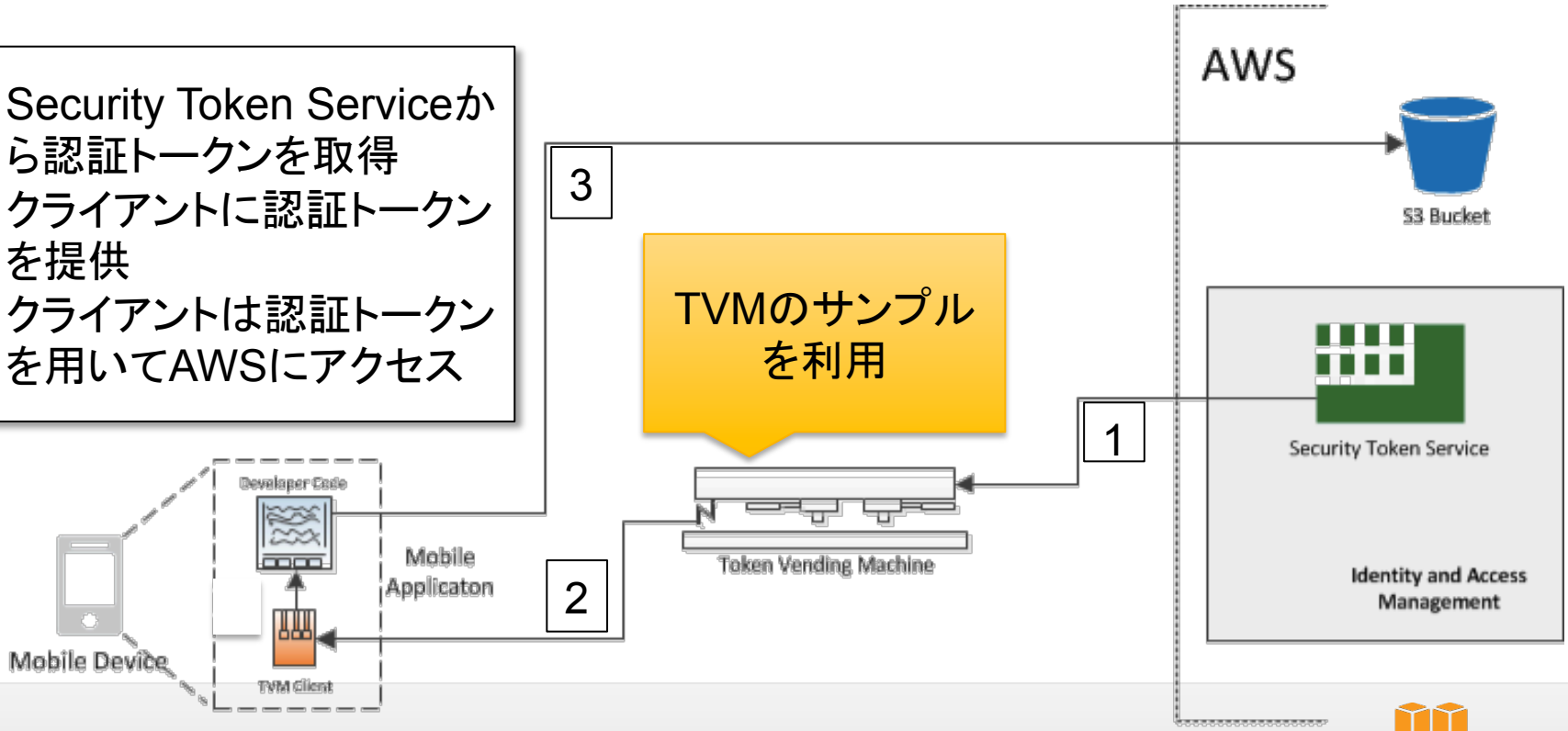
本番ではSSL必須：
平文ではデバイス/ユーザ登録で秘密鍵/パスワードが流出するリスク

[再掲]
認証トークンを用いたサービス利用の流れ

Token Vending Machine (TVM)を導入

- ユーザ/端末の認証とトークンの発行を実施
- アプリケーションごとの認証とAWSの認証機構を結びつけるサービス

1. Security Token Serviceから認証トークンを取得
2. クライアントに認証トークンを提供
3. クライアントは認証トークンを用いてAWSにアクセス



TVMを用いたサンプルを試す AnonymousTVMの動作確認

- samples/S3_SimpleDB_SNS_SQS_DemoTVM
  - 設定例：
    - TVM URL: http://anon-example.elasticbeanstalk.com

Android: AwsCredentials.propertiesを編集
tokenVendingMachineURL=anon-example.elasticbeanstalk.com

iOS: Constants.hを編集
#define TOKEN_VENDING_MACHINE_URL @“anon-example.elasticbeanstalk.com”

/registerdevice?uid=<UID>&key=<KEY>

/gettoken?
uid=<UID>&timestamp=<timestamp>&signature=<Signature>

<Encrypted token credentials>

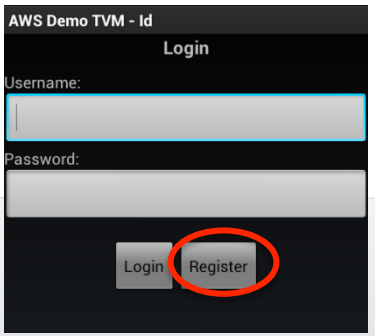
http://anon-example.elasticbeanstalk.com

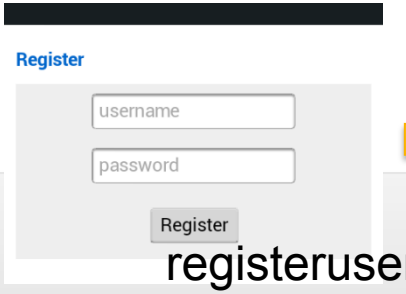
TVMを用いたサンプルを試す IdentityTVMの動作確認

- samples/S3_SimpleDB_SNS_SQS_DemoTVMIdentity
  - 設定例：
    - TVM URL: http://id-example.elasticbeanstalk.com
    - アプリ名: exampleApp (TVMの環境変数”PARAM1”に設定した値)

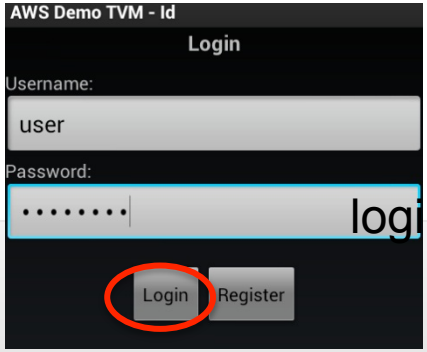
Android: AwsCredentials.propertiesを編集
tokenVendingMachineURL=id-example.elasticbeanstalk.com
appName=exampleApp

iOS: Constants.hを編集
#define TOKEN_VENDING_MACHINE_URL @“id-example.elasticbeanstalk.com”
#define APP_NAME @“exampleApp"

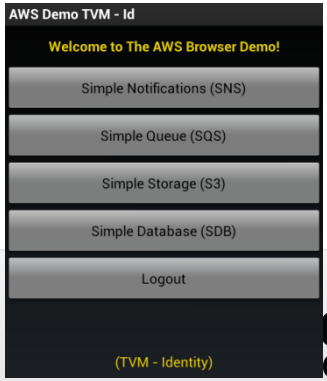




registeruser()



login()



TVM Clientのインテグレーション

- Android / iOSともにTVM Clientの実装がサンプルに
  - 各サンプル内のAmazonClientManagerの実装はTVM Clientを利用
    - S3_SimpleDB_SNS_SQS_DemoTVM → Anonymous TVM
    - S3_SimpleDB_SNS_SQS_DemoTVMIdentity → IdentityTVM

- おすすめのステップは…
  1. ベースとするTVMの実装を決定 (Anonymous or Identity)
  2. 対応するサンプルコード内の実装をコピー
     - AmazonClientManager
     - TVMClient
  3. 必要に応じてTVM, TVM Client, 認証メカニズムをカスタマイズ
     - Identity TVMをカスタマイズすればSNSとのID連携も


amazon web services