

MSL Image Explorer

デモをダウンロードしていただき、ありがとうございます。このチュートリアルでは、MSL Image Explorer の背後で実行される動作について説明します。このアプリケーションを使用して、NASA ジェット推進研究所が発行した火星の画像を簡単に保存してインデックスを作成することができます。このアプリケーションを使用する過程で、いくつかの DynamoDB API 呼び出しの使用方法について説明します。そして、JavaScript のコードスニペットを紹介します。また、このアプリケーションには、DynamoDB に JSON データを保存する方法を示すサンプル Java アプリケーションも用意されています。

開始する前に、ご利用のマシンで以下がサポートされている必要があります。

- a) Java 1.7+
- b) NodeJS
- c) Maven

エクスプローラを使用する

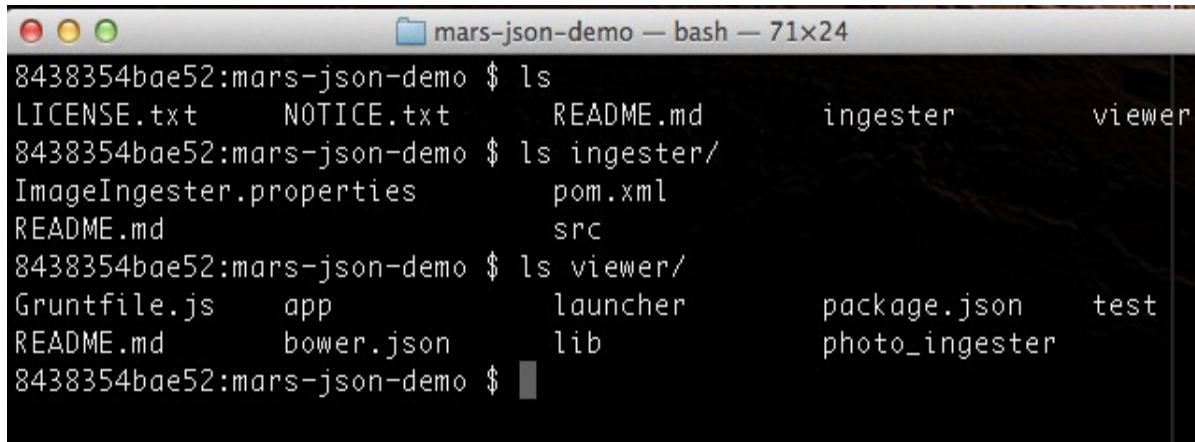
1) デモのファイルをすべてダウンロードします。

a. オプション 1、GitHub から取得できます

```
git clone https://github.com/aws-labs/aws-dynamodb-mars-json-demo.git
```

b. オプション 2、この [リンク](#) からダウンロードできます

2) すべてのファイルを解凍すると、以下のようになります。



```
mars-json-demo — bash — 71x24
8438354bae52:mars-json-demo $ ls
LICENSE.txt  NOTICE.txt  README.md  ingester  viewer
8438354bae52:mars-json-demo $ ls ingester/
ImageIngester.properties  pom.xml
README.md                  src
8438354bae52:mars-json-demo $ ls viewer/
Gruntfile.js  app  launcher  package.json  test
README.md     bower.json  lib  photo_ingester
8438354bae52:mars-json-demo $
```

3) 以下のコマンドを使用して、アプリケーションをビルドします。

a. Image Ingester - ./Ingester フォルダに移動して、次のコマンドを実行します。

```
mvn clean install
```

b. Image Viewer - ./viewer ディレクトリに移動して、次のコマンドを実行します。

```
sudo npm install -g grunt-cli bower
npm install
bower install
```

4) アプリケーションを起動する前に、./ingester/ フォルダの ImageIngester.properties というプロパティファイルを確認します。特徴的な設定を以下に示します。

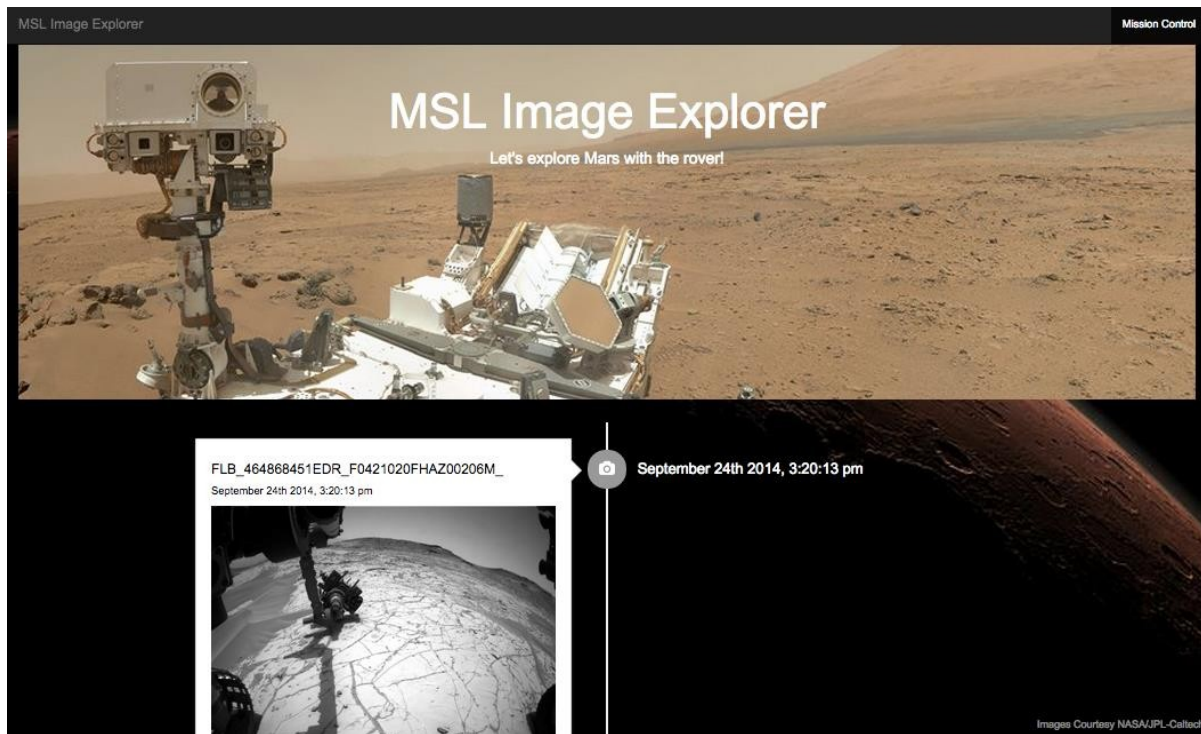
- a. JSON.root - これは、JSON の読み取り元を定義します。2 つの URL が事前定義されており、一方をコメントアウトし、もう一方をコメントアウトせずに残すことで、どちらか 1 つを選択できます。
- b. dynamodb.endpoint - これは、DynamoDB クライアントが接続するエンドポイントです。テーブルは、エンドポイントが提供するリージョンに配置されます。エンドポイントが <http://localhost:8000> に設定されている場合は、[DynamoDB Local](#)、（DynamoDB の無料のクライアント側のバージョン）を指します。この設定は、クラウドでデモアプリケーションを実行する際に必要に応じて変更できます。

- c. dynamodb.resource、dynamodb.image - これらは、リソースと画像の DynamoDB テーブルのテーブル名です。画像のテーブルは必須で、リソースのテーブルは `ingester.track-resources` オプションを有効にした場合に必須です。
- d. `ingester.store-thumbnails - true` に設定した場合、DynamoDB にサムネイルデータが保存されます。これは、データベースに大きなアイテムを保存するときに役立ちます。
- e. `ingester.manifest.threads`、`ingester.sol.threads`、`ingester.image.threads` - タスクごとに実行されるスレッドの数。これらの数は、コストと時間のバランスをとるため、テーブルのコンピューティング能力やプロビジョニングされたスループットに基づいて微調整できます。

5) これで、アプリケーションを起動する準備ができました。すべてのダウンロードファイルが置かれたディレクトリに戻ります。"grunt" コマンドを使用して、Mars Image Explorer アプリケーションを起動します。

```
8438354bae52:mars-json-demo $ cd viewer/
8438354bae52:viewer $ ls
Gruntfile.js          bower.json            node_modules
README.md              bower_components      package.json
app                   lib                    test
8438354bae52:viewer $ grunt serve
```

6) ブラウザでこの美しいアプリケーションが開かれます。



7) `viewer/app/scripts/services/mars-photos.js` というファイルで以下のコードスニペットを参照できます。

- 8) これで、エクスプローラアプリケーションから参照できる画像を制御できます。
 - a. 異なる方法を選択するには、"Mission control ☐ Select Instrument" に移動します。
 - b. 背後で実行されているコードスニペットは以下のとおりです。

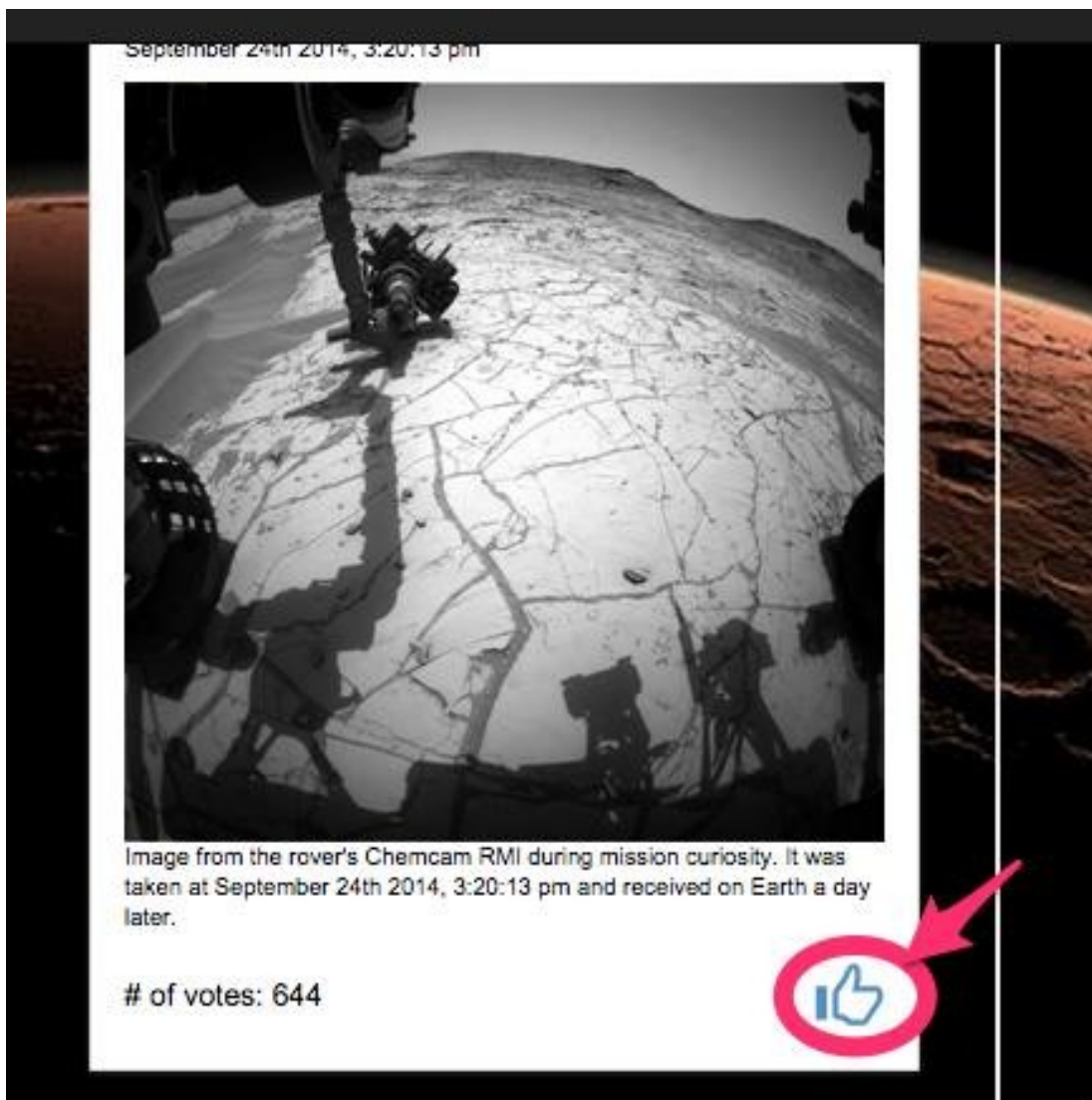
```

/**
 * Queries photos table with the date index and give the result
 * to callback function.
 * @param {object} queryParams - The query parameters used in the query. The parameter
 * hashKey is required. The lastEvaluatedKey and rangeKey are optional.
 * @param {function} callback - The callback function used to return the result.
 */
queryWithDateIndex: function(queryParams, callback){
    assertHashKey(queryParams);
    assertFunction(callback);

    var params = {
        TableName: 'marsDemoImages',
        KeyConditions: [
            AWS.dynamoDB.Condition('Mission+InstrumentID', 'EQ', queryParams.hashKey)
        ],
        IndexName: 'date-gsi',
        Limit: 5,
        ScanIndexForward: false
    };
    if(hasLastEvaluatedKey(queryParams)){
        params.ExclusiveStartKey = queryParams.lastEvaluatedKey;
    }
    if(hasRangeKey(queryParams)){
        params.KeyConditions.push (AWS.dynamoDB.Condition('TimeStamp', 'LE', queryParams.rangeKey));
    }
    logRequest('query', params);
    AWS.dynamoDB.query(params, callback);
}

```

- c. 更新の方法を確認するには、気に入った画像に投票します。
- d. 画像に投票するには、投票ボタンをクリックします。



- e. 投票するたびに、背後で更新のコードが実行されます。以下ようになります。

```
/**
 * Increments the voting count in the photo table. It gets the updated
 * votes count in return. The function is meant to be used privately
 * in MarsPhoto service.
 * @param {String} imageid - The ID of the photo to increment vote count.
 * @param {function} callback - The callback function used to return the result or error.
 */
var incrementVotesCount = function(imageid, callback) {
    var params = {
        TableName: 'marsDemoImages',
        Key: { imageid: imageid },
        UpdateExpression: 'add votes :v',
        ExpressionAttributeValues: { ':v': 1 },
        ReturnValues: 'UPDATED_NEW'
    };

    logRequest('updateItem', params);
    AWS.dynamoDB.updateItem(params, callback);
};
```

- f. 同様に条件付き更新も簡単に記述できます。複数回投票しないようにする場合、テーブルでユーザーの投票を追跡し、ユーザーが特定の画像に投票したかどうかをテーブルで確認します。コードスニペットは以下のとおりです。

```
item.userid = userid;
var params = {
    TableName: 'userVotes',
    Item: item,
    Expected: [
        AWS.dynamoDB.Condition('imageid', 'NULL')
    ]
};
logRequest('putItem', params);
AWS.dynamoDB.putItem(params, function(error) {
    if (!error) {
        $log.debug('Liked image successfully');
        incrementVotesCount(photo.imageid, callback);
    } else {
        if(error.code === 'ConditionalCheckFailedException'){
            callback('You have already voted on this image');
        } else {
            $log.error(error);
        }
    }
});
```

MSL Image Explorer アプリケーションと DynamoDB コマンドの使用を楽しんでいただけたでしょうか。DynamoDB のヘルプと取扱い説明書については、[『開発者ガイド』](#)を参照してください。