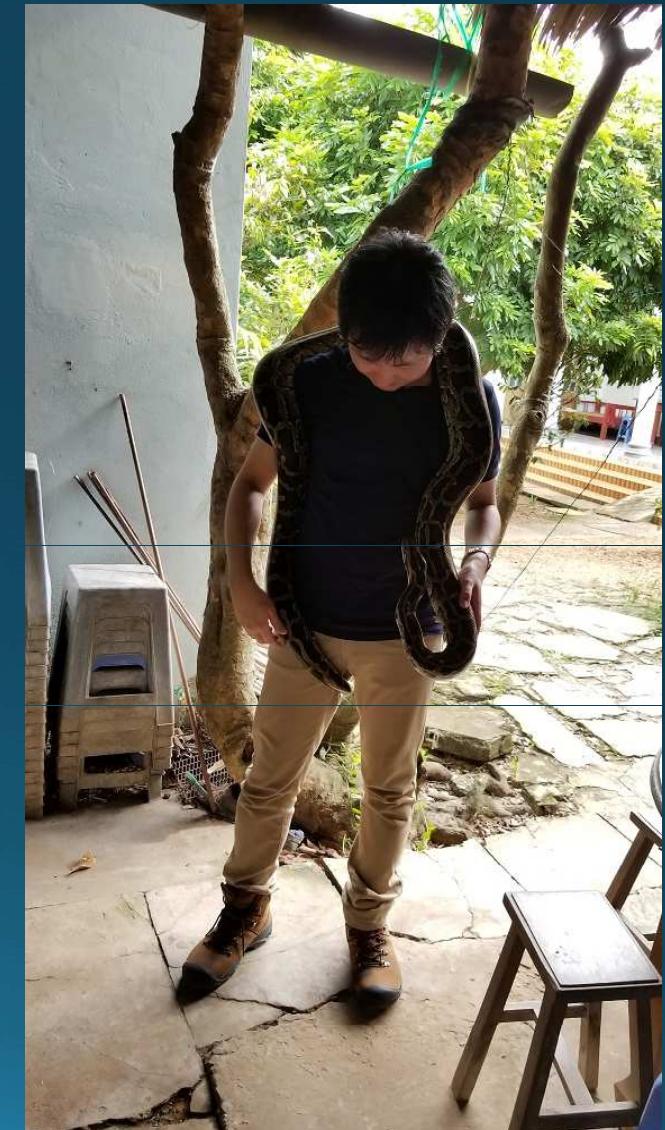


新日鉄住金ソリューションズ株式会社
SCC(ソリューション企画・コンサルティングセンター)

AWS Glue in Action

自己紹介

- 磯部俊行
- 新卒3年目
- SCC ・ アーキテクチャグループに所属
 - 技術よろずや
- AWSを中心に案件支援や検証
 - 前回日本経済新聞社様がお話しされた「Atlasプロジェクト」の技術ドメイン全般を担当
- 好きなAWSサービス: Beanstalk, DynamoDB
- 好きなプログラミング言語: Rust



所属部署のミッション



- ・全社ソリューション企画
- ・各事業部の支援

アーキテクチャグループは特に技術ドメインで支援をしています！

本日話すこと

- AWS Glueとは
- Glueの基本（さらっと行きます）
 - 機能
 - Glue用語まとめ
 - 基本的なワークフロー
 - 環境のセットアップとアクセスコントロール
- 実際に触ってみた&調べてみた
- データ分析案件で使おうとしている
- こんなのがあったら良いな

AWS Glue

2017/11/02 (木) 16:28



磯部 俊行

RE: Bigdata-JAWSの発表タイトルとアジェンダについて

宛先 ■高橋 慧 ■伊藤 宏樹; SCCアーキ系グループ

> 「今日の AWS Glue」
> 「孤独の AWS Glue」
> 「つかまえろ AWS Glue」
> 「AWS Glue 党」
> 「大図解! AWS Glue」
> 「AWS Glue まるわかり」
> 「はやわかり AWS Glue」
> 「交響詩 AWS Glue」
> 「あの日見た AWS Glue の名前を僕達はまだ知らない」
>
>
>

>> -----Original Message-----

>> From: 高橋 慧
>> Sent: Thursday, November 02, 2017 3:13 PM
>> To: 伊藤 宏樹; SCCアーキ系グループ

>> Subject: RE: Bigdata-JAWSの発表タイトルとアジェンダについて

>>
>> 「AWS Glue in Action」
>> 「今夜わかる AWS Glue」
>> 「エンジニアのための AWS Glue 入門」

>> 「誰も知らない AWS Glue」
>> 「AWS Glue で実現するさいきょうのバッチ基盤」
>> 「AWS Glue? なにそれ?美味しいの?」
>> 「re:Invent 2016 にて燐然と輝いた AWS Glue」
>> 「S3 と Redshift をピッタリくっつける AWS Glue」
>> 「AWS Glue で身長が10cm伸びました」
>> 「AWS Glue で彼女ができました」
>> 「AWS Glue の神話」
>> 「やさしい AWS Glue」
>> 「なれる!AWS Glue!」
>> 「いかにして AWS Glue を使うか」
>> 「AWS Glue への道」
>> 「AWS Glue は電気羊の夢を見るか」

2017/11/02 (木) 16:28

磯部 俊行

RE: Bigdata-JAWSの発表タイトルとアジェンダについて

宛先 ■高橋 慧 ■伊藤 宏樹; SCCアーキ系グループ

> 「今日の AWS Glue」
> 「孤独の AWS Glue」
> 「つかまえろ AWS Glue」
> 「AWS Glue 党」
> 「大図解! AWS Glue」
> 「AWS Glue まるわかり」
> 「はやわかり AWS Glue」
> 「交響詩 AWS Glue」
> 「あの日見た AWS Glue の名前を僕達はまだ知らない」
>
>
>

>> -----Original Message-----

>> From: 高橋 慧
>> Sent: Thursday, November 02, 2017 3:13 PM
>> To: 伊藤 宏樹; SCCアーキ系グループ

>> Subject: RE: Bigdata-JAWSの発表タイトルとアジェンダについて

>>
>> 「AWS Glue in Action」
>> 「今夜わかる AWS Glue」
>> 「エンジニアのための AWS Glue 入門」

>> 「誰も知らない AWS Glue」
>> 「AWS Glue で実現するさいきょうのバッチ基盤」
>> 「AWS Glue? なにそれ?美味しいの?」
>> 「re:Invent 2016 にて燐然と輝いた AWS Glue」
>> 「SSS と Redshift をどうやってつなげる AWS Glue」
>> 「AWS Glue で身長が 10cm 伸びました」
>> 「AWS Glue で彼女ができました」
>> 「AWS Glue の神話」
>> 「やさしい AWS Glue」
>> 「なれる! AWS Glue!」
>> 「いかにして AWS Glue を使うか」
>> 「AWS Glue への道」
>> 「AWS Glue は電気羊の夢を見るか」

>> --

re:Invent 2016で
大いに盛り上がった
比較的新しいサービス

S3からRedshiftに良い感じにロードする
方法を考えているんだけど…

今動かしてるバッチ、
Glueってやつに移行できる？

AWS Glueって結局どうなの？
そろそろ東京来る？使えるの？

AWS Glueとは？

- AWS上でデータレイク(S3)や各種データストア(RDS, Redshift)の間を繋ぐETL(Extract, Transform, Load)サービス
- 従来EMRクラスタを構築して行っていた大規模な分散処理をマネージド型サービスとしてサーバーレスに行うことができる
- 利用したリソース分だけ課金される



AWS Glueの基本

機能

Glueの主な機能

- データカタログ
 - S3やRDS, Redshiftといったデータストアのメタデータをデータカタログの形で保存できる
 - メタデータの変更履歴を保持し、データの変化を把握できる
 - 生成されたデータカタログはRedshift Spectrum, Athena, EMRのカタログとしても利用できる
- スキーマの自動検出（クローラ）
 - クローラが各種データストアを自動的に走査し、データカタログにメタデータを追加・更新することができる
 - デフォルトで複数のフォーマットに対応し、またGrokパターンで独自のClassifierを記述することができる
- ジョブ
 - PySparkのスクリプトを利用し、Sparkクラスタを用いた分散処理のジョブを作成できる
 - GlueにはいくつかのTransformが用意されており、単純な処理であればほぼ自分でコードを書く必要がない
 - スケジュールベース、イベント(他ジョブの完了)ベース、オンデマンドでジョブをキックできる

Glue用語まとめ

用語(1/2)

Data Catalog	1つのアカウントに1つだけある、Glueのメタデータの格納先。Tableの定義等が含まれる。Apache Hiveのメタストアと同様のものである。
Data Store	S3やRDSなど、実データが格納されている場所
Table	データを表現するメタデータ。カラム名、データ型などから構成される。実データはTableとは関係なく存在し、GlueはTableを通して実データにアクセスできる。TableはJobのSourceやTargetとして使うことができる。
Crawler	Data Storeを走査して、Data Catalogにメタデータ(Table)を自動的に作成する。
Classifier	データのスキーマを決定する。GlueはCSV, JSON, AVRO, JDBCなどに対するClassifierを提供している。Grokパターンを利用してオリジナルのClassifierを作成することもできる。
Connection	Data Storeにアクセスするためのコネクション情報が格納される。Connectionに対してIAM RoleやSecurity Groupを設定できる。

用語(2/2)

Database	複数のTableから構成される論理的なグループ
Job	ETLを実行するロジック。Transform ScriptとData Source, Targetから構成される。
Script	PySparkにGlueの拡張を加えたETLのコード
Transform	データを別のフォーマットやスキーマに変換するもの。Built-in Transformとして基本的なものが提供されている。
Trigger	Jobを起動する。スケジュールベース、イベントベース、あるいはオンデマンドを設定できる。
Development endpoint	Scriptの開発やテストに使うための環境で、常時Glueのジョブを実行するクラスタが起動しているようなもの(環境の起動時間に対して時間課金される)
Notebook server	Development endpointにセットアップできるZeppelinサーバで、Scriptの開発環境として使うことができる。

基本的なワークフロー

1. Tableの定義を作成する

- 手動で作ることもできるが、Crawlerを作成して自動生成することが推奨されている
- Built-inのClassifierでかなり多くのフォーマット[†]とデータストア[‡]に対応している
- 対応していない場合、独自のclassifierを作成できる

2. 作成したTableからSourceとTargetを指定し、ETLを記述してJobを作成する

- 実際はPySparkコードであり、既存の資産をそのまま流用することもできる

3. 作成したJobのTriggerを作成し、Jobを実行する

- スケジュールベースの定期実行やイベント(他のジョブの終了など)ベースでJobを実行することができる

[†] http://docs.aws.amazon.com/ja_jp/glue/latest/dg/add-classifier.html

[‡] http://docs.aws.amazon.com/ja_jp/glue/latest/dg/add-crawler.html

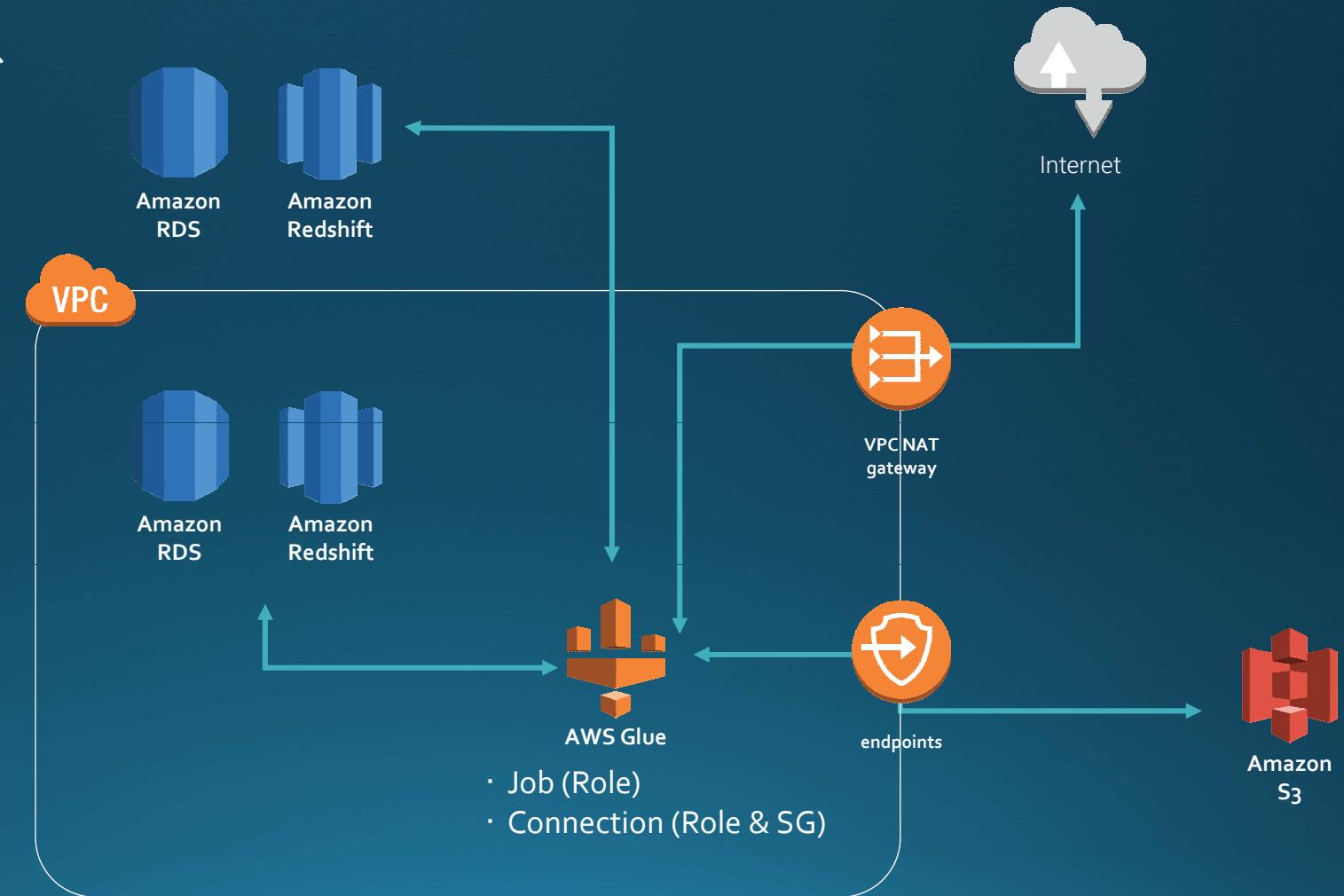
環境のセットアップと アクセスコントロール

リソースに対する権限設定

- 他のサービス同様IAM Policyを使って制御する
 - IAMのEntity(User, Group, Role)に設定するIdentity-basedのポリシーがサポートされている
- 開発用に便利なポリシーが用意されている
 - **AWSGlueConsoleFullAccess**
 - コンソールを使ったGlueのオペレーションに必要なポリシー。コンソールを利用するUserにアタッチする。
 - **AWSGlueServiceRole**
 - JobやCrawlerに設定するRoleにアタッチする。S3のキーとCloudWatch Logsのロググループ名がGlueの生成する規定のものである限り、それらのリソースへのアクセスが許可される。Role名はAWSGlueServiceRoleから始まる名前をつけるか、利用者にiam:PassRoleでRole名を指定する必要がある。
 - **AWSGlueServiceNotebookRole**
 - Notebook serverに設定するRoleにアタッチする。各リソースの名称がGlueの生成する規定のものである限り、Notebook serverからのアクセスを許可する。

VPC & Security Group

- Connection作成時にVPCを指定すると、Glueは指定したSubnetにPrivate IPを持ったENIをセットアップして起動する
(Public IPは持たない)
- Connectionと各Data Store間が通信できるようにSG (Security Group)を設定する
 - 自身との通信を許可するSGを作成し、Data StoreとConnectionに設定すると良い
- Internetにアクセスする必要がある場合、NATを設定する
- VPC内からS3にアクセスする必要がある場合、VPC Endpointsを設定する
- VPCのenableDnsHostnamesとenableDnsSupportをtrueに設定する



実際に触ってみた
&
調べてみた

弊社でよく聞くユースケース



- S3にあるオブジェクトに処理をしてからRedshiftにロードするという処理へのニーズが多い
 - データレイクとしてデータをとりあえずS3に置き、それを分析したいというニーズ
 - 現状はそれぞれの案件で3rd partyのETLツールが使われていたり、EMRで処理をしてからCOPY文を流したり、手製のスクリプトをジョブスケジューラでキックして処理している
- 今回は主にS3 => Redshiftの処理に主眼を置いて、案件を進める上でよく尋ねられる事柄について調べてみました

変なデータが入ってきても
ちゃんと処理できる？

DynamicFrame

- Glueのcreate_dynamic_frameで作られるDataFrameの拡張のようなもの
- レコード1行に相当するDynamicRecordから成る
- DataFrameと異なり、 Schemaを定義しなくて良い
 - 自動で型を検出
 - 同一カラムに複数の型が見つかると、両方のデータ型を保持する
 - ResolveChoiceによって複数の型があるカラムの扱いを選択できる
 - cast: 型のcastを試み、失敗したらnullにする
 - make_cols: 型毎にカラムを作る e.g. id_long, id_string, ...
 - make_struct: カラムをStruct型として両方を保持する
 - project: castと同様にcastを試みるが、失敗したらカラムを捨てる
- DataFrameと相互に変換できる
 - toDF(), fromDF()

Redshiftへのロード

- Redshiftのテーブルに合わせて次の処理を行う(デフォルトだと自動でコードが生成される)
 - ApplyMapping: カラムのマッピングを定義
 - ResolveChoice: 1つのカラムに複数の方がある場合の振る舞いを選択
- glueContext.write_dynamic_frame.from_jdbc_conf()で DynamicFrameをRedshiftにロードできる
 - テーブルがない場合新たに作られる
 - String型の列はvarchar(65535)型として作られる
 - make_col等で新たなカラムが生じると、COPYの前にALTER TABLEが実行され、カラムが追加される
 - S3の一時フォルダに書き出されてからCOPY文が実行される

```
COPY dest_table FROM 's3://aws-glue-scripts-286101162442-us-east-1/tmp/...'  
CREDENTIALS " FORMAT AS CSV DATEFORMAT 'YYYY-MM-DD HH:MI:SS' NULL AS 'null'
```

[参考] S3へのロード

- `write_dynamic_frame.from_options`を使う
 - 「後で分析をする時に高速に並列読み込みをするために複数ファイルに書き出す」とあるが、実際に使ったところDPU 16で数百KBのオブジェクトが1440個書き出された…
 - 大量の小さいファイルはアンチパターンでは
 - Top 10 Performance Tuning Tips for Amazon Athena
<https://aws.amazon.com/jp/blogs/big-data/top-10-performance-tuning-tips-for-amazon-athena/>
- DataFrameWriterを使う
 - `repartition(n)`で分割数を指定した後`write`で書き出すと指定した分割数で書き出すことができる
 - パフォーマンスは劣化する可能性がある
 - より細かく出力フォーマットを指定することができる

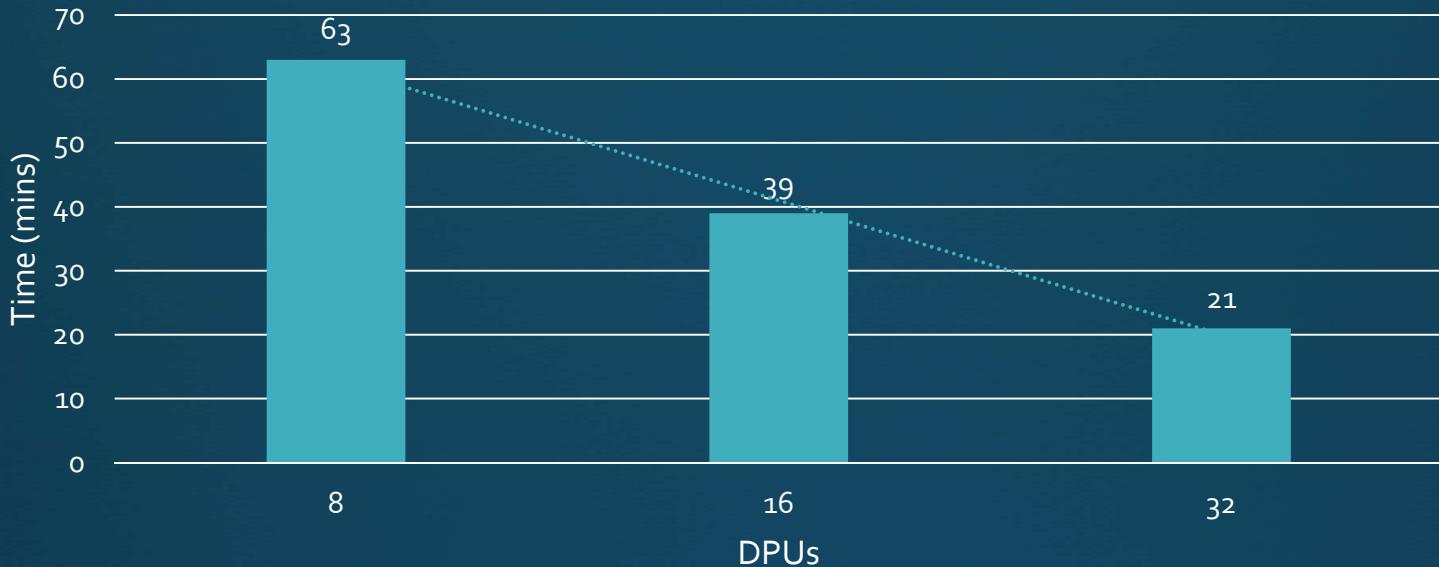
Glueって速いの？

テストケース

- apache_log_gen(https://github.com/tamtam180/apache_log_gen)で生成したApacheダミーログを利用
- 500 reqs/secの想定で、1分間(30000 reqs)毎のログファイルを生成(JSON, GZIP)し、S3に格納
 - 1日分 =4320万レコード， 約10GB (非圧縮)
- 次の処理を行う
 - DataFrameに変換し、UDFとwithColumnsで処理
 - UserAgentをua_parser(<https://github.com/ua-parser/uap-python>)でパースし、OS Family, Device Family, UA Familyのカラムを追加
 - IPとUserAgentの結合文字列のSHA256ハッシュを計算し追加
 - Redshiftへロードする

DPUs	Data Volume	1st	2nd
8	24 hours	63	79
16	24 hours	46	39
32	24 hours	29	21
32	12 hours	18	12
32	1 hour	7	11

- 初回実行時はコールドスタートのような挙動
 - 2度目は速くなることが多いが、そうでないケースもあり、詳細は不明（詳細な仕様は非公開とのこと）
 - 予想実行時間に対して10~20分程度の実行時間の差異を許容できないケースでは使うのは難しい



- DPUsの数によって処理時間はほぼ線形にスケールする
 - 1つのDPU(Data Processing Unit)あたり4vCPUと16GBメモリが割り当てられる
 - e.g. 16DPUs => 64vCPU, 256GBメモリ
 - DPUsの数(N)とSparkクラスタの起動オプションの関係(推測)
 - `spark.dynamicAllocation.maxExecutors = 2 * N - 2`
 - `spark.executor.memory = 5g` (全体のメモリ量からすると少なめ?)
- 処理がDPUs数に応じてスケールするかはSparkの時と同じ考え方で良い
 - Shuffle等が入ると線形にはスケールしないと考えられる

コード書かなくて良いってホント？

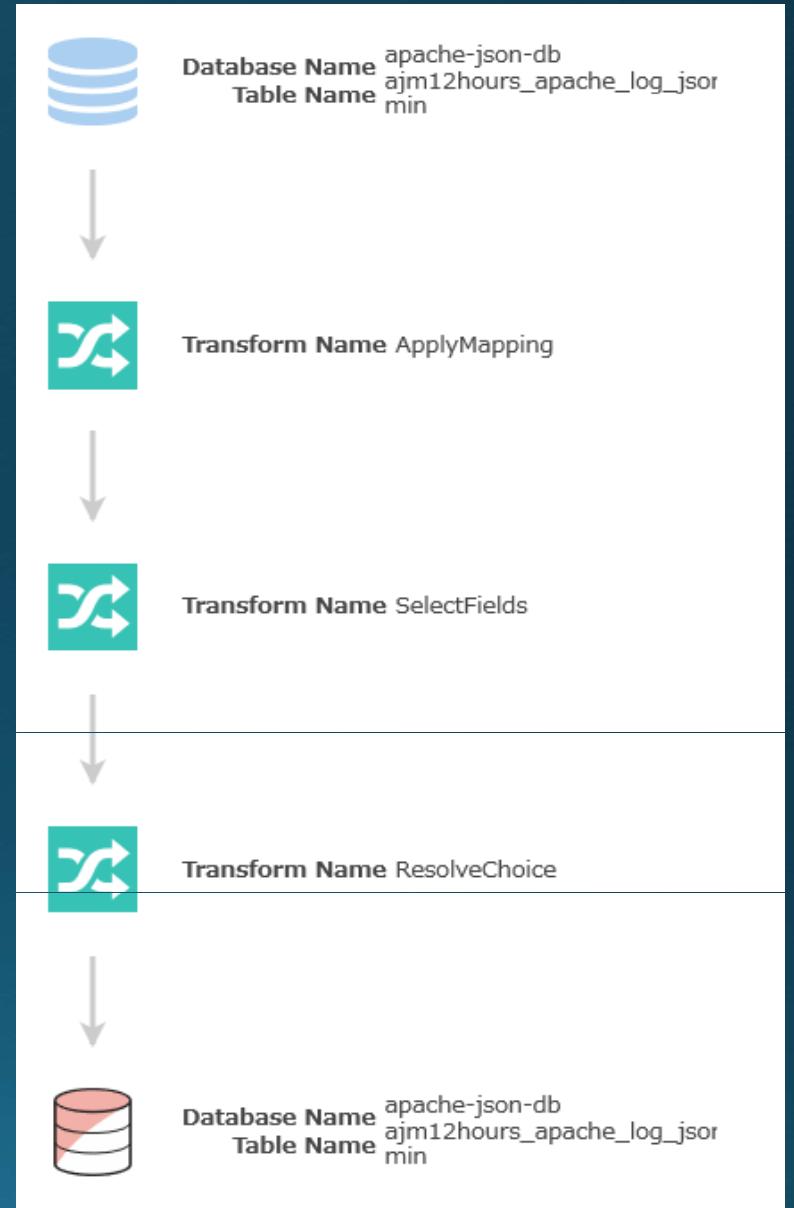
- GlueではData SourceとTargetを指定すると単純なETLコードを生成してくれるため、そのままJobとして実行できる。
- また、次頁に示すBuilt-in Transformで対応できる処理であれば、Transformの引数を記述するだけで処理を変更することができる。
 - TransformのAPIリファレンスは読む必要があるが、PythonやSparkの知識はそこまで求められない。

```

13 ## @type: ApplyMapping
14 ## @args: [mappings = <mappings>, transformation_ctx = "<transformation_ctx>"]
15 ## @return: <output>
16 ## @inputs: [frame = <frame>]
17 <output> = ApplyMapping.apply(frame = <frame>, mappings = <mappings>, transformation_ctx = "<transformation_ctx>")

```

<value>を埋めるだけで規定の処理を実行できる



デフォルトのJob

Built-in Transform(1/3)

ApplyMapping	sourceとなるDynamicFrameに対してカラム名とデータ型のマッピングを指定して、新しいDynamicFrameを得る。マッピングは("sourceのカラム名", "sourceのデータ型", "destinationのカラム名", "destinationのデータ型")というタプルの配列で指定できる。
DropFields	プロパティを指定して削除することができる。例えばField Aの子のField Bを削除する場合、A.BとPathに指定することで削除できる。
DropNullFields	型がnull typeであるfieldsをDynamicFrameから削除する。
Filter	特定の条件にマッチするレコードのみのDynamicFrameを取得する。
Join	2つのDynamicFrameに対して、双方に存在する特定のKeyフィールドを指定し、値が等しいレコードを結合した1つのDynamicFrameを取得する。
Map	入力されたDynamicFrameの各レコードに対して指定した関数を適用したDynamicFrameを返す。関数の実行時にエラーが発生した場合、当該レコードがエラーとしてマークされ、処理は継続される。

<http://docs.aws.amazon.com/glue/latest/dg/aws-glue-programming-python-transforms.html>

Built-in Transform(2/3)

MapToCollection	入力されたDynamicFrameCollectionに含まれる各DynamicFrameに対してTransformを適用した新しいDynamicFrameCollectionを返す。
Relationalize	DynamicFrameをrowsとcolumnsから構成されるリレーションナル形式に変換する。ネストされたstructはそれぞれのプロパティが新しいカラムとして作られ、配列はそれぞれ別のDynamicFrameとして生成される。DynamicFrameCollectionが取得できる。
RenameField	pathで指定した特定のfieldをリネームする。
ResolveChoice	あるDynamicFrameのcolumnが複数の型を持っている場合に、どのように処理するかを指定し、処理されたDynamicFrameを返す。
SelectFields	指定したpathのfieldsからのみなるDynamicFrameを返す。
SelectFromCollection	Indexを指定して、DynamicFrameCollectionから特定のDynamicFrameを取得する。

<http://docs.aws.amazon.com/glue/latest/dg/aws-glue-programming-python-transforms.html>

Built-in Transform(3/3)

Spigot	DynamicFrameからS3にサンプルデータを書き出す。S3のロケーションとサンプリング方法(レコード数など)を指定する。
SplitFields	あるDynamicFrameを特定のFieldsで2つに分割する。出力は2つのDynamicFrameのCollectionで、1つは指定したfieldsのみからなるDynamicFrameで、他方は残りのfieldsからなるDynamicFrameである。
SplitRows	あるDynamicFrameを特定の条件を指定して2つに分割する。出力は2つのDynamicFrameのCollectionで、1つは条件に合致するrowsのみからなるDynamicFrameで、他方は合致しなかったrowsからなるDynamicFrameである。
Unbox	あるStringのfieldをペース(unbox)したDynamicFrameを返す。例えばあるCSVの特定のカラムが3つのプロパティからなるJSON文字列だった場合に、このカラムを3つのプロパティに対応するカラムに分割することができる

<http://docs.aws.amazon.com/glue/latest/dg/aws-glue-programming-python-transforms.html>

Built-inのTransform使った方が良い?
既存のPySpark使いたいんだけど…

複数の方法で同一処理を試してみる

1. Built-in TransformのMapを使う

```
map1 = Map.apply(frame = datasource, f = map_function, transformation_ctx = "map1")
```

2. DataFrameに変換してUDFとwithColumn()を使う

```
df = datasource.toDF()
df.withColumn("hash", udf_get_hash("host", "agent"))
...
dest = DynamicFrame.fromDF(df, glueContext, "back_to_dynamic_frame")
```

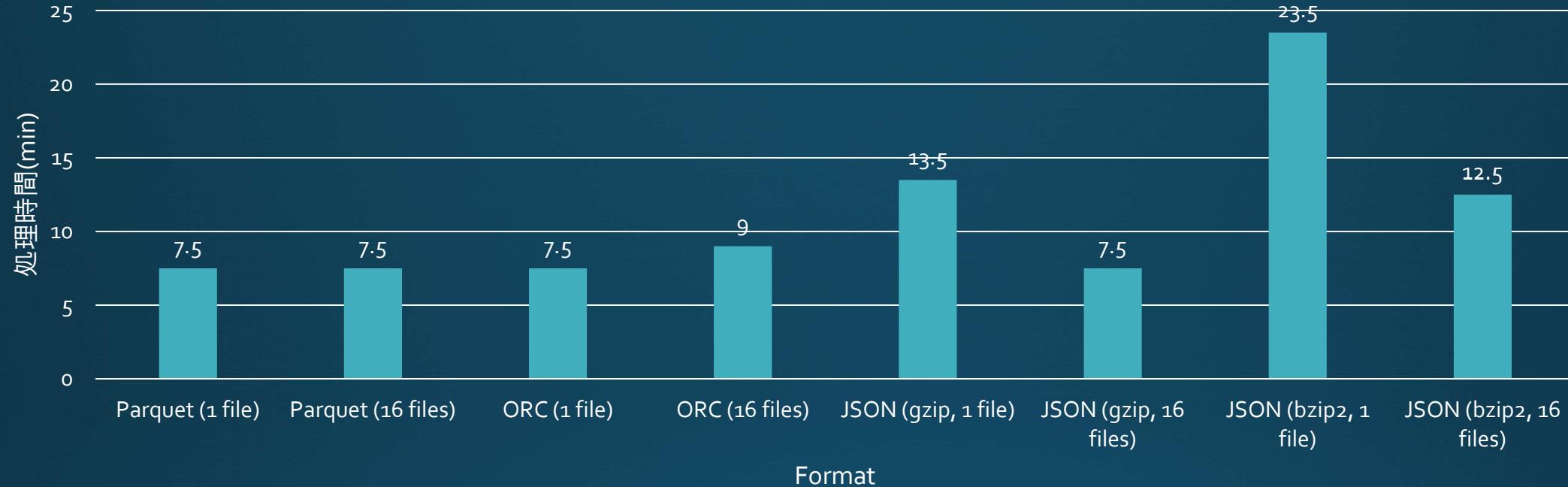
3. RDDに変換してmap()を使う

```
rdd = datasource.toDF().rdd
rdd = rdd.map(add_columns)
df = spark.createDataFrame(rdd, schema)
dest = DynamicFrame.fromDF(df, glueContext, "back_to_dynamic_frame")
```

Type	DPUs	Data Volume	1st	2nd	3rd
1. DynamicFrame	8	1 hour	31	13	15
2. DataFrame	8	1 hour	15	7	7
3. RDD	8	1 hour	27	7	7

- 少なくともカラムの追加処理に関しては、DynamicFrameのBuilt-in TransformのMapよりもDataFrameやRDDに変換してから処理を行った方が速かった
- 既存のPySparkの資産がある場合、ExtractとLoadの時だけGlueの DynamicFrameを用い、TransformはDataFrameに変換して既存のコードを活用するといった使いができる
- また、既存のPythonライブラリ、JavaライブラリもS3から読み込んで使用することができる

S3にはどういう形式でファイルを
置くのが良い？



- S3 => RedshiftのGlueのデフォルトのJobを実行
 - 12時間分のデータ: 2160万レコード, 約5GB (非圧縮)
 - DPUs: 16
 - 複数回実行した内の最小値を記録 (RedshiftのCOPY時間除く)
- ParquetやORCといった列指向フォーマットが安定してパフォーマンスが良く、また複数に分割したGZIPファイルも同等の結果
- ただしテストデータ量が小さく、オーバーヘッドの影響が大きい

- 基本的にはEMRやAthenaのベストプラクティスがGlueにも適用できる
 - https://media.amazonwebservices.com/jp/wp/AWS_Amazon_EMR_Best_Practices.pdf
 - <https://aws.amazon.com/jp/blogs/big-data/top-10-performance-tuning-tips-for-amazon-athena/>
- ファイルサイズと分割数を最適化する
 - 分割数が小さいとS3からのReadがボトルネックになる
 - 一般にHadoopは大量の小さなファイルを扱うのに向いておらず、分割数を増やせば良いわけではない
 - Executorが並列に読み込める数に分割する
- bzip2などのSplitableな形式で圧縮する
 - 圧縮することでネットワークトラフィックを軽減できる
 - bzip2などの分割可能な圧縮形式にする
- ParquetやORCといった列指向フォーマットを使う
 - Athena等を使う場合にもメリットを享受できる

Kinesis Streamsから次々オブジェクトが到着するんだけど、抜け漏れなく継続的に処理できる？

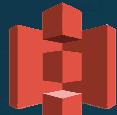
Ingest



Amazon
Kinesis



AWS
Lambda



Amazon
S3

Time Series



Analysis



AWS
Glue



Amazon
Redshift

Job Bookmarks

- S3のSourceが以前のJobで処理されたかの状態を保存しておく機能
- Jobに対してJob Bookmarksを設定できる
 - Enable: 未処理の新しいデータに対して処理をする。処理後に状態を更新し、次回以降のRunで今回処理したデータが再処理されないようにする
 - Disable: 状態を保存しない。常に全データセットを処理する
 - Pause: 未処理の新しいデータに対して処理をする。ただし、Enableと異なり処理後も状態は更新せず、処理したデータも未処理として扱われる

Job Bookmarksを有効にし、順次S3にオブジェクトを追加しながら定期的にJobを走らせてみた

- Jobの同時実行数であるConcurrentを1にした場合、以前のJobが終了しないと次のJobは実行されず、次々追加されるS3もそれぞれ一度ずつ処理された
- Concurrentを2以上に設定した場合、Jobの実行が重なるとそれぞれ同じオブジェクトを処理してしまった
 - Concurrentを2以上にする場合、Job Bookmarksを有効にしても重複を排除する処理を後段で行う必要がある
- Jobの実行に失敗した場合(リトライ上限に達した場合)、その時処理の対象だったオブジェクトは処理済みとされず、次のRunで再び処理された

1件もレコードを漏らしちゃダメなん
だけどGlue使っていい？

Job BookmarksとErrorsAsDynamicFrameを組み合わせてみる

- Jobにはリトライ回数を指定することができ、ネットワークやData Storeの一時的なダウンで失敗しても自動リトライできる
 - Job Bookmarksと組み合わせることで、At Least Onceは実現できそう
- DynamicFrameはエラーを示すDynamicRecordを持つことができる
 - 例えばMap処理中にあるレコードで例外が発生すると、Glueはエラーをマークして処理は継続する
 - ErrorsAsDynamicFrameでエラー情報とエラーレコードを含んだDynamicFrameを抽出できるので、別の場所に書き出して後からエラーコードのみに処理をするといったことが可能

監視と運用どうしたら良い？

CloudWatch Events & Logs

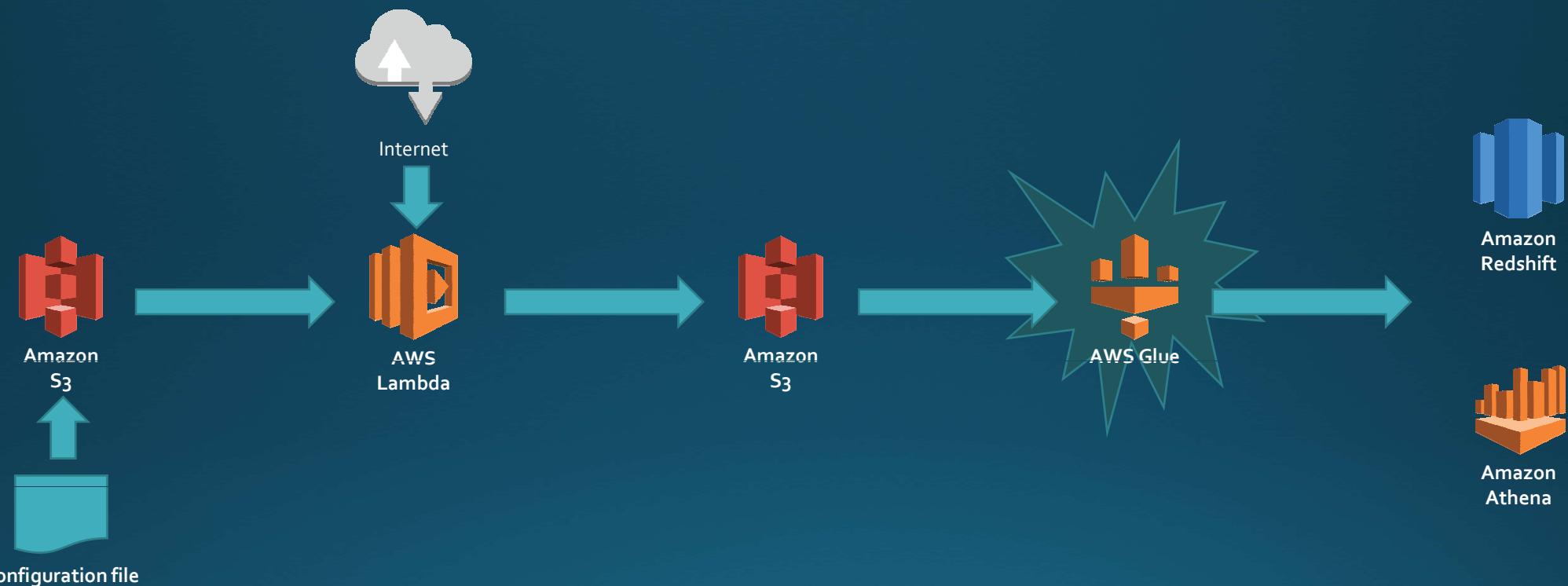
- CloudWatch Eventsで次のイベントを捕捉できる
 - Glue Job State Change
 - SUCCEEDED, FAILED, STOPPED
 - Glue Crawler State Change
 - Started, Succeeded, Failed
- CloudWatch Logsのaws-glueロググループに各JobのログがJob ID のストリームに出力される
 - メトリクスフィルタでイベントを捕捉できる
- ErrorsAsDynamicFrameでエラーレコードをS3に書き出すようにJob を設定し、Putイベントを捕捉すればJob内のエラーもモニターすることができる
- その他、Jobの実行時間等のメトリクスはGlue APIから取得する
<http://docs.aws.amazon.com/glue/latest/dg/aws-glue-api.html>

データ分析案件で使おうとしている

データ分析案件向けユースケース

- 分析でよく使うオープンデータを半自動取り込みしたい
 - プロジェクト毎に取り込みを行っており非効率的
 - 必要となるのはオープンデータ全体ではなく、分析トピックに応じた項目だけ
- よく使うオープンデータの例
 - 気象庁提供「過去の気象データ・ダウンロード」
 - <http://www.data.jma.go.jp/gmd/risk/obsdl/index.php>
 - 総務省提供「国勢調査データ」
 - <http://www.e-stat.go.jp/SG1/estat/eStatTopPortal.do>

やりたいこと



- ユーザが取得のための設定情報をS3 PUTしたら、Lambdaを用いてデータをスクレイピングして、データレイク用S3に保存
- データレイク用S3に保存されたらAWS GlueのCrawlerをまわし、データを利用できる状態にする

オープンデータにありがとうこと

- ・不要なヘッダ/複数行にまたがるヘッダ



GlueでCrawlすると…

- 余計なヘッダが入っていると、Classifierが識別してくれない

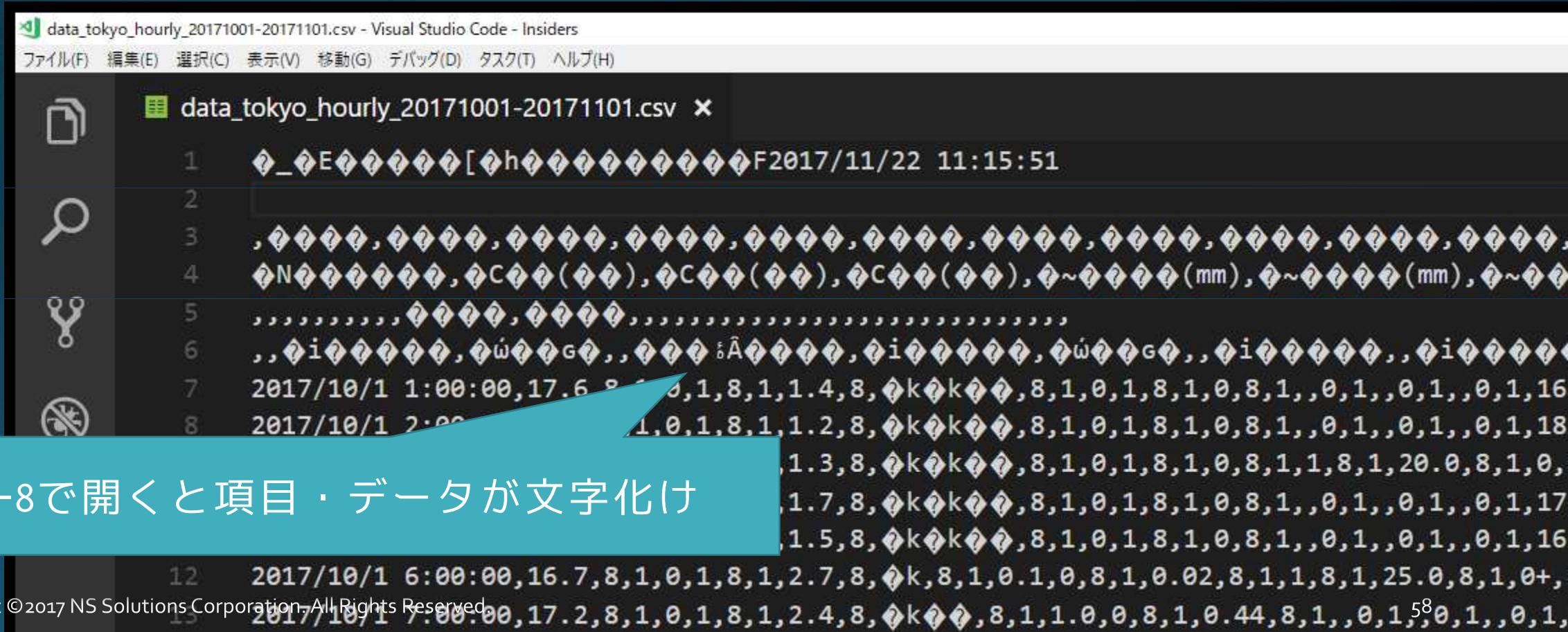
Name	Database	Location	Classification
sjis_census	opendata	s3://scc-batchsystem-test-sjis/census/	csv
sjis_data_tokyo_ho...	opendata	s3://scc-batchsystem-test-sjis/jma/2017...	Unknown

- 項目名が複数行にまたがっていると、項目がデータに含まれてしまう

col0	col1	col2	col3	col4	col5	col6
1	1	"平成27年国勢調査人口等基本集計（総務省統計局）"	"	"	"	"
2	2	"	"	"	"	"
3	3	"	"	"	"	"
4	4	"	"	"	"	"

オープンデータにありがちなこと

- エンコーディングはもちろんShift-JIS



The screenshot shows a Visual Studio Code window with the title bar "data_tokyo_hourly_20171001-20171101.csv - Visual Studio Code - Insiders". The menu bar includes "ファイル(F)", "編集(E)", "選択(C)", "表示(V)", "移動(G)", "デバッグ(D)", "タスク(T)", and "ヘルプ(H)". The left sidebar has icons for file, search, and other code-related functions. The main editor area displays the following CSV data:

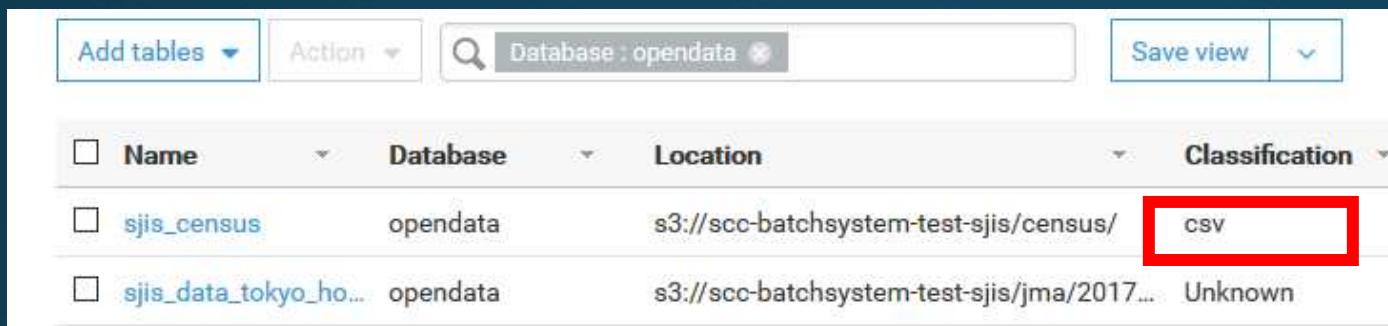
	項目	データ
1	日付	2017/11/22 11:15:51
2	データ	（完全な日本語が表示されない）
3	データ	（完全な日本語が表示されない）
4	データ	（完全な日本語が表示されない）
5	データ	（完全な日本語が表示されない）
6	データ	（完全な日本語が表示されない）
7	データ	2017/10/1 1:00:00,17.6,8,1,8,1,1.4,8,0k0k,8,1,0,1,8,1,0,8,1,,0,1,,0,1,,0,1,16
8	データ	2017/10/1 2:00,1.0,1,8,1,1.2,8,0k0k,8,1,0,1,8,1,0,8,1,,0,1,,0,1,,0,1,18
12	データ	1.3,8,0k0k,8,1,0,1,8,1,0,8,1,1,8,1,20.0,8,1,0,1.7,8,0k0k,8,1,0,1,8,1,0,8,1,,0,1,,0,1,17
13	データ	,1.5,8,0k0k,8,1,0,1,8,1,0,8,1,,0,1,,0,1,,0,1,16
15	データ	2017/10/1 6:00:00,16.7,8,1,0,1,8,1,2.7,8,0k,8,1,0.1,0,8,1,0.02,8,1,1,8,1,25.0,8,1,0+,2017/10/1 7:00:00,17.2,8,1,0,1,8,1,2.4,8,0k,8,1,1.0,0,8,1,0.44,8,1,,0,1,,0,1,58,0,1,,0,1,

A large blue callout box points from the bottom-left towards the data in rows 8, 12, and 13, containing the Japanese text "UTF-8で開くと項目・データが文字化け".

UTF-8で開くと項目・データが文字化け

GlueでCrawlすると…

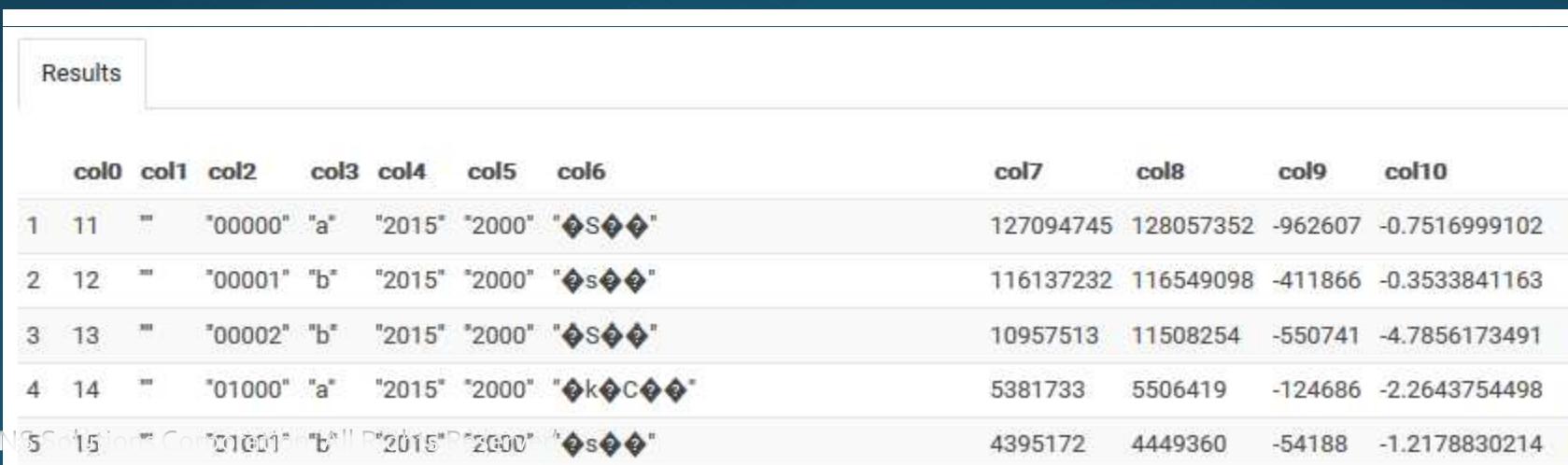
- Crawlそのものは上手く行く



The screenshot shows the AWS Glue Catalog interface. At the top, there are buttons for 'Add tables', 'Action', a search bar with 'Database: opendata', and a 'Save view' button. Below this is a table with columns: Name, Database, Location, and Classification. Two rows are listed:

Name	Database	Location	Classification
sjis_census	opendata	s3://scc-batchsystem-test-sjis/census/	csv
sjis_data_tokyo_ho...	opendata	s3://scc-batchsystem-test-sjis/jma/2017...	Unknown

- 利用するAthena/Redshiftはutf-8のみ対応のため文字化けする

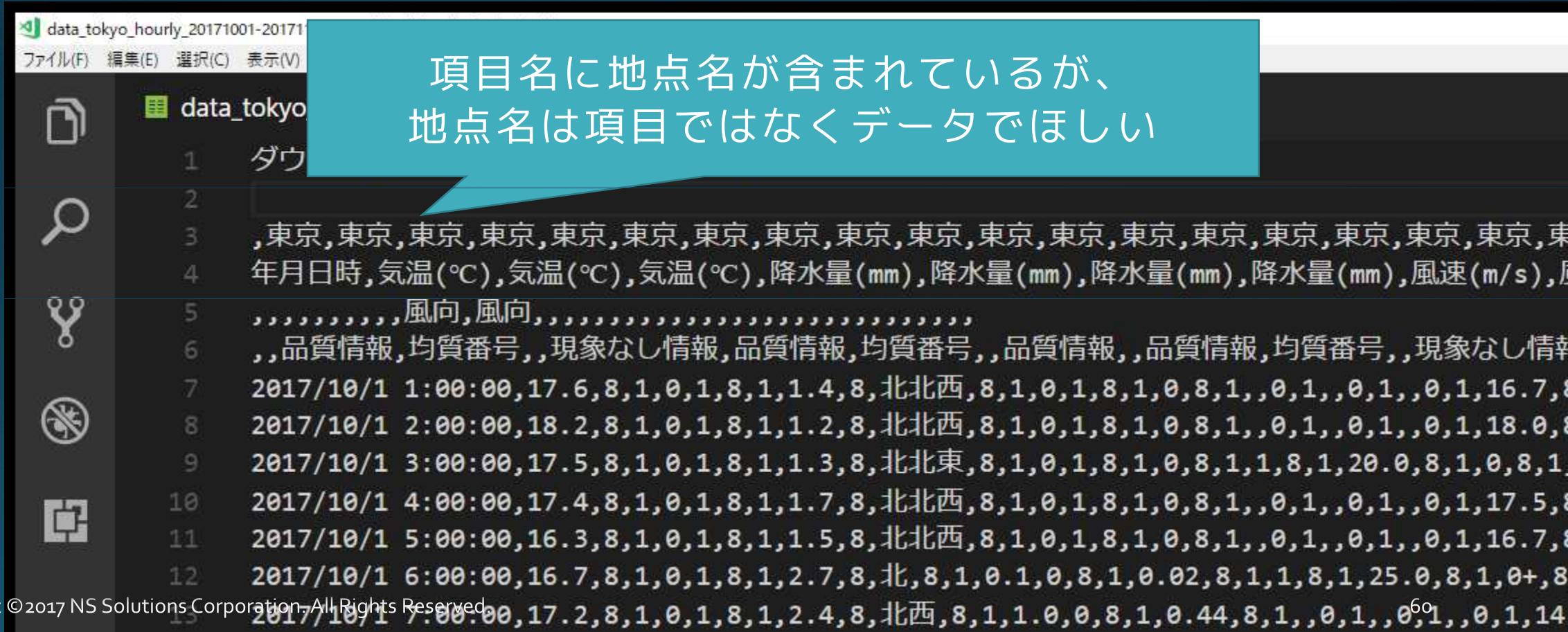


The screenshot shows the results of an Athena query. The table has 11 columns labeled col0 through col10. The first few rows of data are as follows:

col0	col1	col2	col3	col4	col5	col6	col7	col8	col9	col10	
1	11	"	"00000"	"a"	"2015"	"2000"	"♦S♦?♦"	127094745	128057352	-962607	-0.7516999102
2	12	"	"00001"	"b"	"2015"	"2000"	"♦s♦?♦"	116137232	116549098	-411866	-0.3533841163
3	13	"	"00002"	"b"	"2015"	"2000"	"♦S♦?♦"	10957513	11508254	-550741	-4.7856173491
4	14	"	"01000"	"a"	"2015"	"2000"	"♦keCe♦?♦"	5381733	5506419	-124686	-2.2643754498
5	15	"	"01001"	"b"	"2015"	"2000"	"♦s♦?♦"	4395172	4449360	-54188	-1.2178830214

オープンデータにありがちなこと

- ・縦横変換が必要



data_tokyo_hourly_20171001-20171130.csv

ファイル(F) 編集(E) 選択(C) 表示(V)

data_tokyo

1 ダウ

2

3 ,東京,東京,東京,東京,東京,東京,東京,東京,東京,東京,東京,東京,東京,東京,東京,東京,東

4 年月日時,気温(°C),気温(°C),気温(°C),降水量(mm),降水量(mm),降水量(mm),降水量(mm),風速(m/s),風

5 ,,,,風向,風向,,,,,風向,風向,風向,風向,風向,風向,風向,風向,風向,風向,風向,風向,現象なし情

6 ,,品質情報,均質番号,,現象なし情報,品質情報,均質番号,,品質情報,,品質情報,均質番号,,現象なし情

7 2017/10/1 1:00:00,17.6,8,1,0,1,8,1,1.4,8,北北西,8,1,0,1,8,1,0,8,1,,0,1,,0,1,,0,1,16.7,8

8 2017/10/1 2:00:00,18.2,8,1,0,1,8,1,1.2,8,北北西,8,1,0,1,8,1,0,8,1,,0,1,,0,1,,0,1,18.0,8

9 2017/10/1 3:00:00,17.5,8,1,0,1,8,1,1.3,8,北北東,8,1,0,1,8,1,0,8,1,1,8,1,20.0,8,1,0,8,1

10 2017/10/1 4:00:00,17.4,8,1,0,1,8,1,1.7,8,北北西,8,1,0,1,8,1,0,8,1,,0,1,,0,1,,0,1,17.5,8

11 2017/10/1 5:00:00,16.3,8,1,0,1,8,1,1.5,8,北北西,8,1,0,1,8,1,0,8,1,,0,1,,0,1,,0,1,16.7,8

12 2017/10/1 6:00:00,16.7,8,1,0,1,8,1,2.7,8,北,8,1,0,1,0,8,1,0,02,8,1,1,8,1,25.0,8,1,0+,8

13 2017/10/1 7:00:00,17.2,8,1,0,1,8,1,2.4,8,北西,8,1,1.0,0,8,1,0,44,8,1,,0,1,,0,1,,0,1,14

項目名に地点名が含まれているが、
地点名は項目ではなくデータでほしい

オープンデータにありがちなこと

- ・ 節度あるデータ取得が求められる
 - ・ 必要なデータが複数ファイルにわかれ、それぞれに処理が必要

The screenshot shows a web browser window for the Japan Meteorological Agency's historical meteorological data download page (www.data.jma.go.jp/gmd/risk/obsdl/index.php). A large teal callout box on the left side contains the text "一度にダウンロードできるデータ容量に限り" (Limited by the data volume that can be downloaded at once) and "人とのインタラクション前提のUI" (UI designed for interaction with humans). Another teal callout box points to a progress bar labeled "選択済みのデータ量 0% - 100% (上限)" (Selected data volume 0% - 100% (maximum)). The UI includes a search bar, a sidebar with links like "過去の気象データ", "地域平均気象データ", and "気候リスク管理", and a bottom section for selecting locations and projects.

Glueで…

- ・全部やることは結局諦めました
- ・以下のようなファイル毎の前処理はこれまで通り、LambdaやBatch用に処理を書いて、Glueで取り込める状態にしています
 - Shift-JIS→UTF-8変換
 - ヘッダ削除
 - 項目に応じたデータ追加(縦横変換)

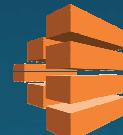


データ分析案件向けユースケースまとめ

- AWS Glueを活用することによって必要なデータを、分析用途に応じた柔軟なスキーマですぐに使える用にできるのは魅力的
- ただオープンデータのような、自社でフォーマットの制御が効かないデータを分析に使える状態にするにはデータの前処理が不可欠
- 全部Glueでやろうとせず、AWS LambdaやAWS Batchを適切に使う



AWS
Lambda



AWS Batch

- そもそもちゃんと機械判読可能なフォーマットで提供してくれればこんな苦労は…

こんなのがあったら良いな

コンソールUI

- Tablesには属性での行のソートがあるが、CrawlersやJobsにはない
- Tableの詳細画面のrecordCountが間違った値になっている
 - Parquet等の列指向のフォーマットでは正しく出るが、JSONやCSVでは間違った値が表示される
- JobのNameでのフィルタが完全一致でしか検索できない

その他

- `glueContext.write_dynamic_frame.from_jdbc_conf()`では RedshiftのCOPYのオプションを指定できない
- TableとJobからパフォーマンスが劣化する可能性を指摘する機能
- より視認性の良いCloudWatch Logsの出力
- Spark Web UIのようなものが欲しい

Thank you!