AWS
Black Belt
Online Seminar

AWS Black Belt Online Seminar

Amazon Athena

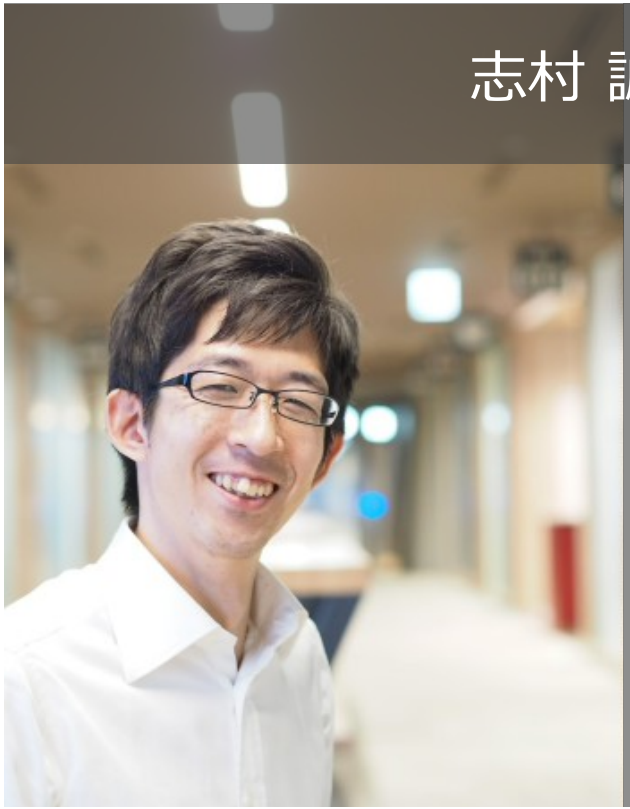
アマゾンウェブサービスジャパン株式会社
ソリューションアーキテクト, 志村 誠

2017.03.01

[2017.04.06更新]

amazon web services

自己紹介



志村 誠 (Makoto Shimura)

所属:
アマゾンウェブサービスジャパン株式会社

業務:
ソリューションアーキテクト
(データサイエンス領域)

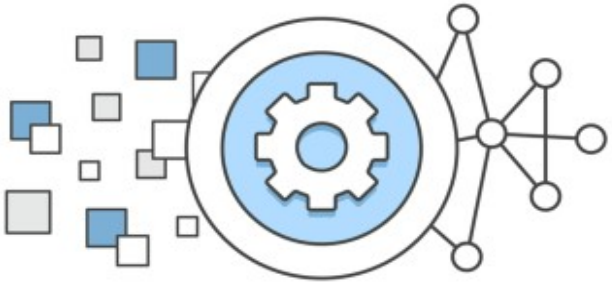
経歴:
Hadoopログ解析基盤の開発
データ分析
データマネジメントや組織のデータ活用

Agenda

- Amazon Athena 概要
- クイックスタート
- Amazon Athena の構成
- データ設計とクエリ特性
- 使いどころ / Update / Tips

Amazon Athena の概要

Amazon Athena とは



S3上のデータに対して，標準 SQL による
インタラクティブなクエリを投げて
データの分析を行うことができるサービス

Amazon Athena とは

- re:Invent 2016 のキーノートにて発表された新サービス
- バージニア北部，オレゴン，オハイオ(2017/2/16追加)の3リージョンで提供
- クエリエンジン Presto と，Hive メタストア互換のデータカタログを使用


Amazon Web Services ブログ

Amazon Athena – Amazon S3上のデータに対話的にSQLクエリを

Amazon Athena紹介

本日、Amazon Athenaについて紹介したいと思います。

https://aws.amazon.com/jp/blogs/news/amazon-athena-interactive-sql-queries-for-data-in-amazon-s3/

Athena の特徴

サーバレスでインフラ管理の必要なし

大規模データに対しても高速なクエリ

事前のデータロードなしにS3に直接クエリ

スキャンしたデータに対しての従量課金

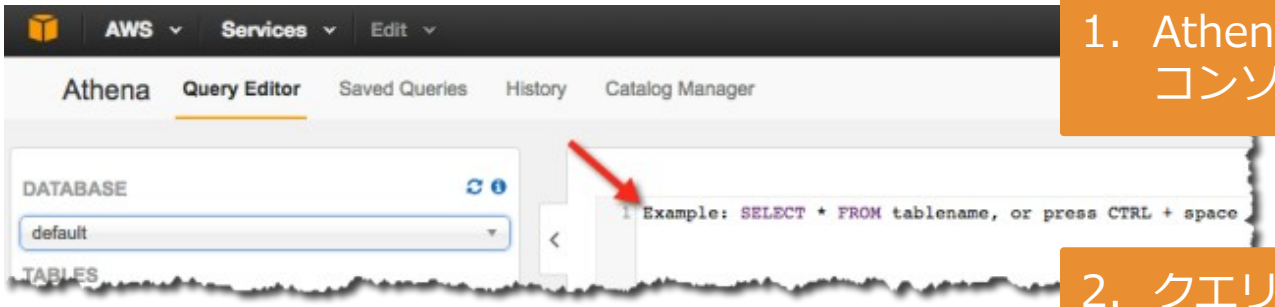
JDBC経由でBIツールから直接クエリ

Athena の想定ユースケース

ユースケース	データ	ユーザ
新しく取得したデータに対して， DWに入れる価値があるか探索的に検証	新しく取得したデータ	アナリスト
利用頻度の低い過去のデータに対する， BIツール経由のアドホックな分析	コールドデータ	アナリスト
Webサーバで障害が発生したときに， ログを漁って原因追求	アクセスログ	サーバ運用
大規模でないデータに対しての， 低頻度で実施するETL処理	生データ	開発者

クイックスタート

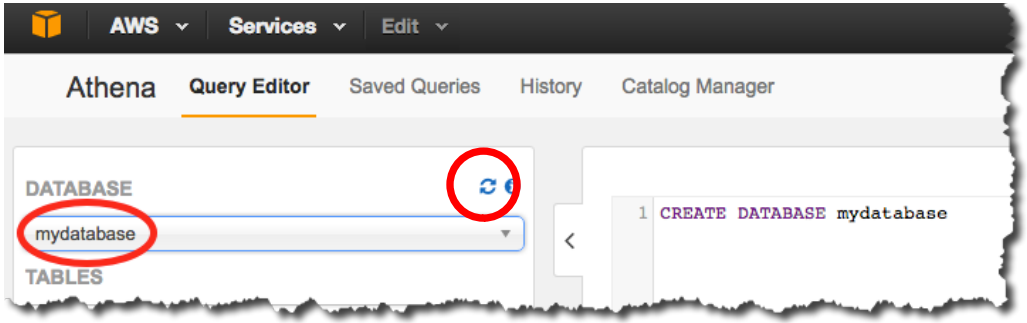
データベースの作成



1. Athenaのマネジメントコンソールを開く

2. クエリエディタからデータベースを作成

```
CREATE DATABASE mydatabase
```

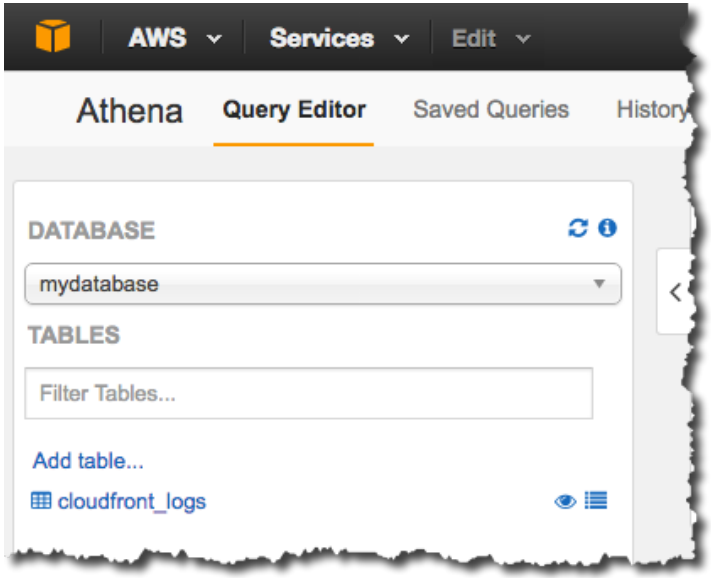


3. ページをリフレッシュ後データベースの存在を確認

テーブルの作成

```
CREATE EXTERNAL TABLE IF NOT EXISTS cloudfront_logs (
    `Date` DATE,
    Time STRING,
    Location STRING,
    Bytes INT,
    RequestIP STRING,
    Method STRING,
    Host STRING, Uri STRING,
    Status INT,
    Referrer STRING,
    Os STRING,
    Browser STRING,
    BrowserVersion STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
    "input.regex" = "^(?!#)([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^
]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^
]+)\\s+[^\(]+[\(]([^\;]+).*\%20([^\/]+)[\/](.*)$"
)
LOCATION 's3://athena-examples/cloudfront/plaintext/';
```

1. クエリエディタからテーブルを作成
2. ページをリフレッシュして，
   テーブルの存在を確認

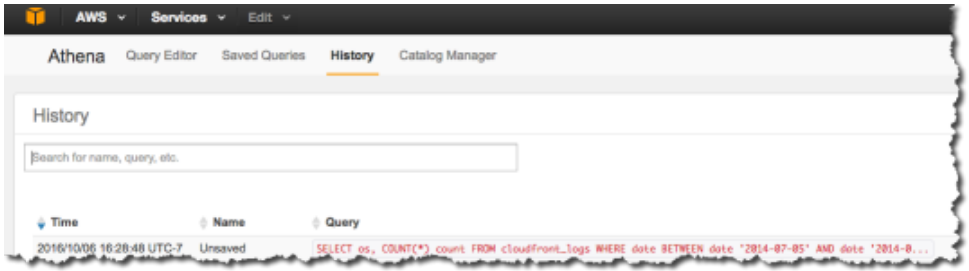


クエリの実行

```
SELECT os, COUNT(*) count FROM cloudfront_logs
WHERE date BETWEEN date '2014-07-05' AND date '2014-08-05' GROUP BY os;
```

Results

	os	count
1	iOS	794
2	MacOS	852
3	OSX	799
4	Windows	883
5	Linux	813
6	Android	855



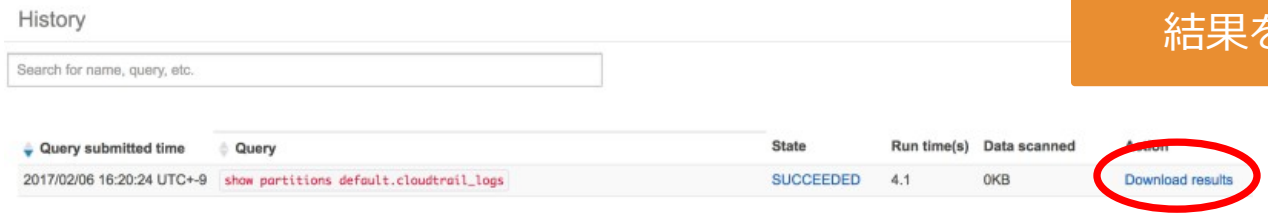
1. クエリエディタから
   集計クエリを実行

2. コンソールに結果が表示

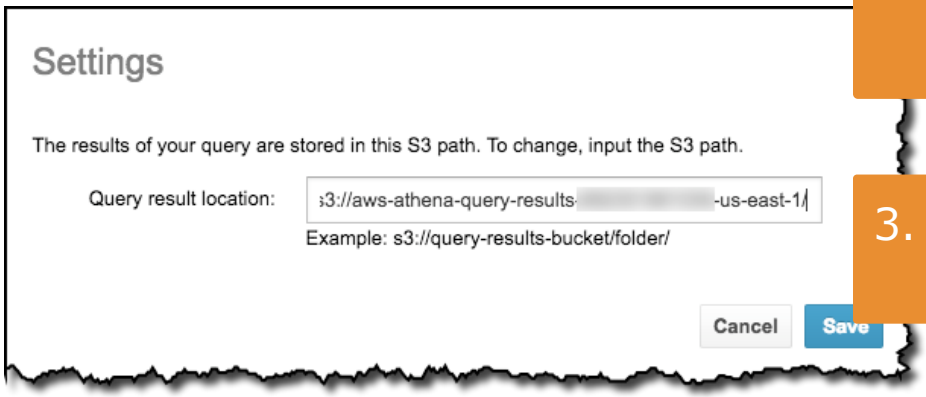
3. 過去のクエリについては
   History で確認可能
   (現状はコンソール確認のみ)

クエリの結果

1. Historyページから直接
   結果をダウンロード

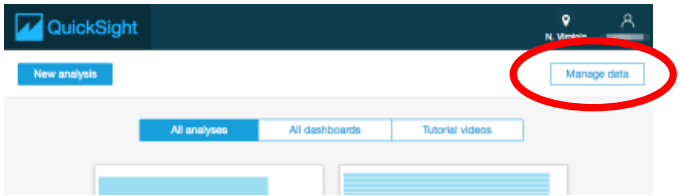


2. 結果はS3バケットに自動的に保存
   されており，IAMユーザごとに
   場所も確認 / 変更可能

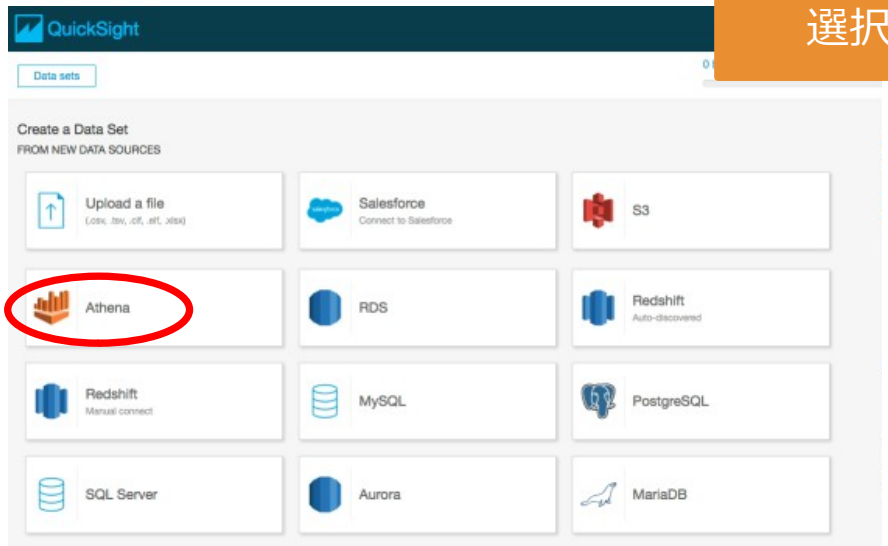


3. デフォルトの保存先バケットは
   aws-athena-query-results-${ACCOUNTID}-${REGION}

QuickSight から Athena に接続



1. QuickSight のメイン画面から右上の Manage data を選択

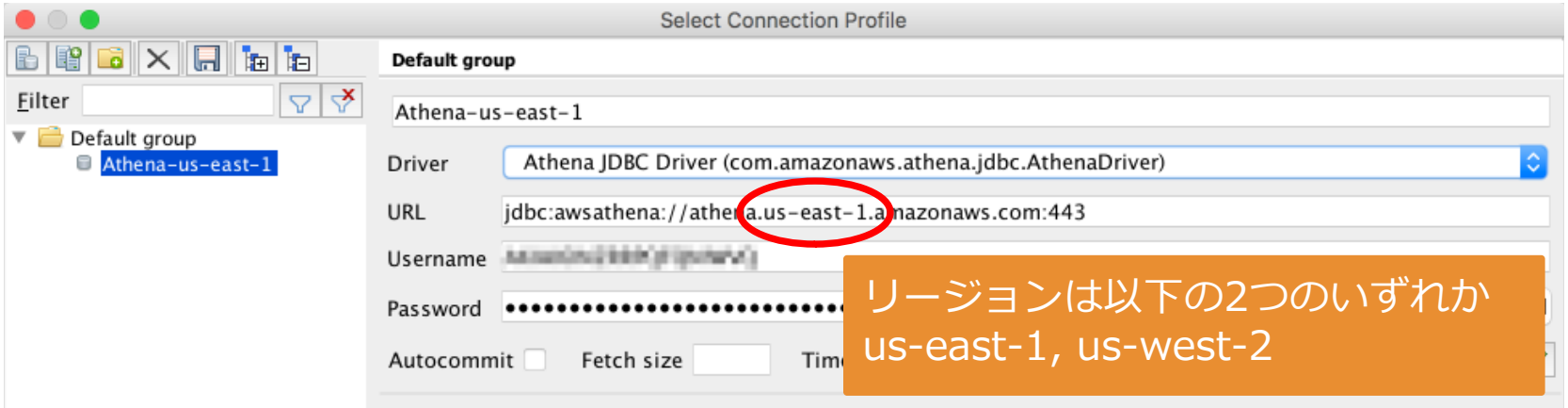


2. さらに左上の New data set を選択して，Athena をクリック

3. あとはガイダンスに従って，DB, table と選んでいく

JDBCでのAthenaへの接続（SQL Workbenchの例）

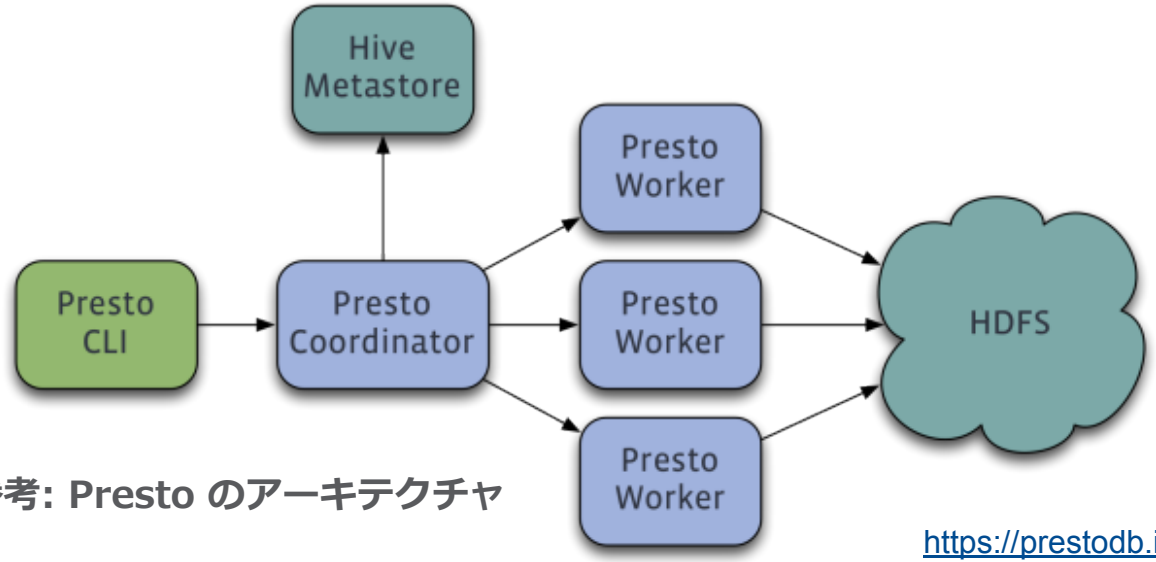
1. JDBC ドライバをダウンロード: s3://athena-downloads/drivers/AthenaJDBC41-1.0.0.jar
2. SQL Workbench の接続設定からドライバの追加
3. 接続先URL: jdbc:awsathena://athena.${REGION}.amazonaws.com:443
   ユーザ名 / パスワード: aws_access_key_id / aws_secret_access_key
   ドライバ: 2. で追加したドライバを指定



Amazon Athena の構成

Presto: 高速な分散クエリエンジン

- Athenaで使用しているクエリエンジン
- データをディスクに書き出さず，すべてメモリ上で処理
- ノード故障やメモリ溢れの場合にはクエリ自体が失敗
- バッチ処理ではなく，インタラクティブクエリ向け

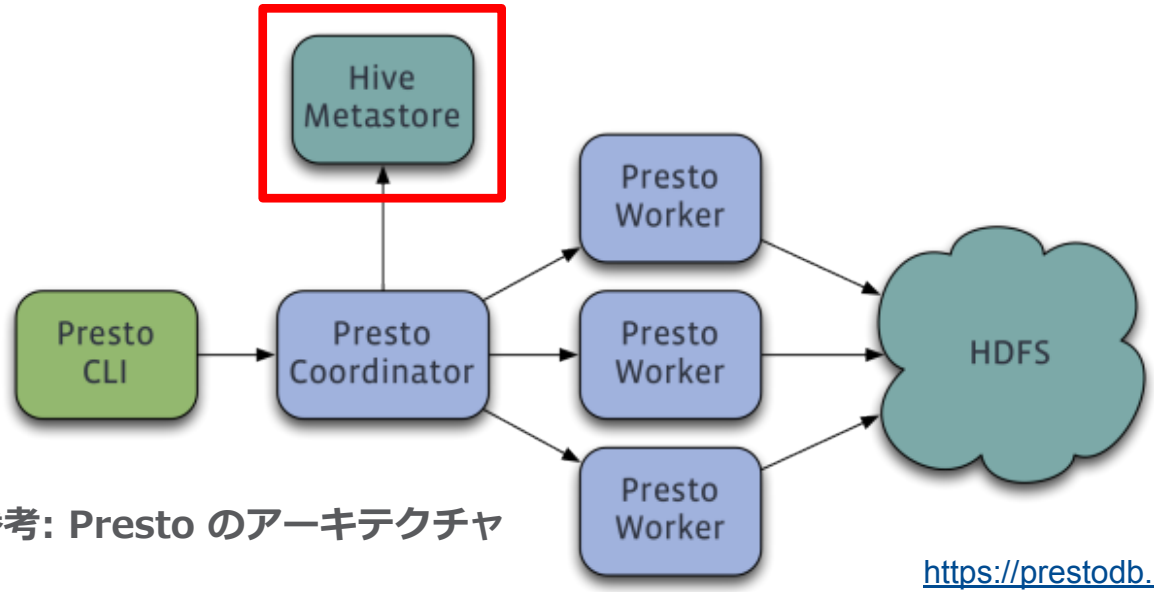


参考: Presto のアーキテクチャ

https://prestodb.io/overview.html

Hive メタストア

- Athena のデータカタログと互換性がある
- Hive は SQL ライクな記法で，Hadoop上のバッチ処理を記述可能
- データソースに対してスキーマを定義し，テーブルのように扱える
- HDFS上のデータをベースとしており，標準 SQL とは異なる DDL を持つ



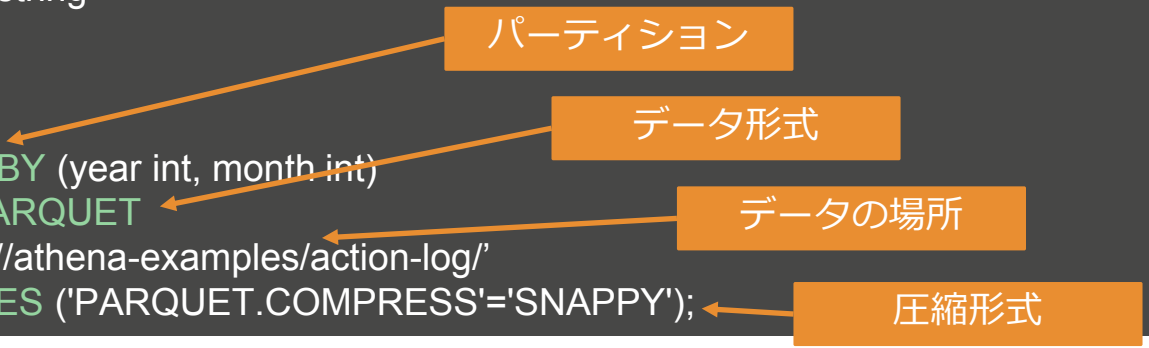
参考: Presto のアーキテクチャ

https://prestodb.io/overview.html

Athena のテーブル定義

- 標準のテーブル定義の後に，データ形式，圧縮形式，データの場所などを指定
- 既に Hive DDL がある場合，Athena で実行すれば，すぐクエリを投げられる
- 既存の EMR の Hive メタストア自体に，直接アクセスすることはできない
- schema-on-read なので，同一データに複数のスキーマを定義可能

```
CREATE EXTERNAL TABLE IF NOT EXISTS action_log (
      user_id string,
      action_category string,
      action_detail string
      year int,
      month int,
)
  PARTITIONED BY (year int, month int)
  STORED AS PARQUET
  LOCATION 's3://athena-examples/action-log/'
  TBLPROPERTIES ('PARQUET.COMPRESS'='SNAPPY');
```



Athena のデータ型

種類	値
プリミティブ型	tinyint, smallint, int, bigint, boolean, float, double, string, binary, timestamp, decimal, date, varchar, char
配列型	array<data_type>
map型	map<primitive_type, data_type>
struct型*	struct<col_name: data_type>
union型	UNIONTYPE<data_type, data_type…>

* https://aws.amazon.com/jp/blogs/big-data/create-tables-in-amazon-athena-from-nested-json-and-mappings-using-jsonserde/

Athena のデータ形式 / 圧縮形式

項目	値	注意点
データ形式	CSV, TSV, Parquet, ORC, JSON, Regex, Avro, Cloudtrail Logs	• 2017/2/16 に Avro と OpenCSV Serde* をサポート • JSONについては Hive-JsonSerDe と Openx-JsonSerDe の2つが利用可能 • CroudtrailSerDe**が利用可能
圧縮形式	Snappy, Zlib, GZIP, LZO	• 2017/4/6 に LZO をサポート

* Serialize/Deserialize の略で，データの入出力形式の変換クラス

** https://aws.amazon.com/jp/blogs/big-data/aws-cloudtrail-and-amazon-athena-dive-deep-to-analyze-security-compliance-and-operational-activity /

Athena のクエリ

- Presto と同様，標準 ANSI SQL に準拠したクエリ
- WITH句，Window関数，JOINなどに対応
- Presto で用意されている関数は，基本的に使用可能

```
[ WITH with_query [, ...] ]
SELECT [ ALL | DISTINCT ] select_expression [, ...]
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ GROUP BY [ ALL | DISTINCT ] grouping_element [, ...] ]
[ HAVING condition ]
[ UNION [ ALL | DISTINCT ] union_query ]
[ ORDER BY expression [ ASC | DESC ] [ NULLS FIRST | NULLS LAST] [, ...] ]
[ LIMIT [ count | ALL ] ]
```

Athena のDDL / クエリにおける注意点

- 現状，CREATE TABLE AS SELECT 句は使用できない
- EXTERNAL TABLE のみが利用可能
- トランザクション処理，UDF / UDAF，ストアドプロシージャ，VIEW等は未サポート
  - 未サポートのDDLについては，マニュアルを参照*
- テーブルに対する変更処理は ACID を保証
  - 複数人が同時に DDL を発行しても，成功するのは1つだけ

* http://docs.aws.amazon.com/athena/latest/ug/language-reference.html#unsupported-ddl

データ設計とクエリ特性

データ設計に影響する Athena の特性

- OLTP* ではなく OLAP** 向け
  - そもそもトランザクションは未サポート
- ETL ではなく 分析向け
  - データをフルスキャン & 変換するのは高コストな設計
  - リトライ機構がないため，安定的なバッチ処理には向かない
- いかにして読み込むデータ量を減らすかが重要
  - パーティション
  - 列指向フォーマット
  - 圧縮

* Online Transactional Processing ** Online Analytical Processing

典型的な OLAP の集計クエリ

```
SELECT
    col1
    , col2
    , COUNT(col3)
    , SUM(col3)
FROM
    table1
    INNER JOIN table2
    ON table1.id = table2.id
WHERE
    table1.id = table2.id
    AND col4 = 1
    AND col5 = “good”
GROUP BY
    col1
    , col2
```

- 基本パターンは，WHERE で条件を絞り，GROUP BY で集約する
- ポイントは，WHERE と SELECT
- 読み込むデータ量をどれだけ減らすかが，パフォーマンスに直結する
  - スキャンする量が減れば，当然クエリの速度は向上する
  - そしてクエリの費用も安くなる

パーティション

S3のオブジェクトキーの構成を CREATE TABLE に反映
WHERE で絞ったときに当該ディレクトリだけ読み込む
1テーブルあたりの最大パーティション数は**20000**

```
CREATE EXTERNAL TABLE IF NOT EXISTS action_log (
    user_id string,
    action_category string,
    action_detail string
    year int,
    month int,
    day int
)
PARTITIONED BY (year int, month int, day int)
STORED AS PARQUET
LOCATION 's3://athena-examples/action-log/'
TBLPROPERTIES ('PARQUET.COMPRESS'='SNAPPY');
```

s3://athena-examples/action-log/year=2017/month=03/day=01/data01.gz

パーティションの効果的な使い方

```
SELECT
    month
    , action_category
    , action_detail
    , COUNT(user_id)
FROM
    action_log
WHERE
    year = 2016
    AND month >= 4
    AND month < 7
GROUP BY
    month
    , action_category
    , action_detail
```

- WHERE で読み込み範囲を絞るときに**頻繁に**使われるカラムを，キーに指定する
- 絞り込みの効果が高いものが向いている
- ログデータの場合，日付が定番
- “year/month/day” と階層で指定する

以下のS3パスだけが読み込まれる

s3://athena-examples/action-log/year=2016/month=04/day=01/
s3://athena-examples/action-log/year=2016/month=04/day=02/
s3://athena-examples/action-log/year=2016/month=04/day=03/
…
s3://athena-examples/action-log/year=2016/month=07/day=31/

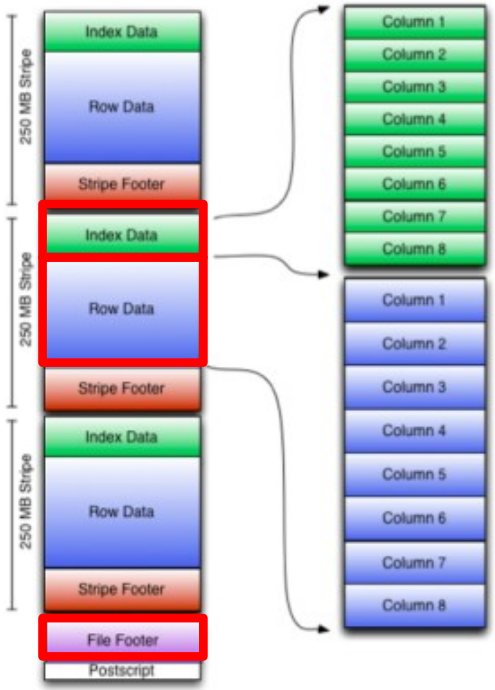
パーティションの分け方

タイプ	形式	特徴
カラム名あり (Hive 標準)	col1=val1/col2=val2/	• CREATE TABLE をしてから，MSCK REPAIR TABLE を実行すればOK • パーティションが増えた際も，MSCK REPAIR TABLE を1回実行すればOK • この形式にするために前処理が必要
カラム名なし	val1/val2/	• 自然な形式 • MSCK REPAIR TABLE が使えないため，ALTER TABLE ADD PARTITION を，パーティションの数だけ実行する必要がある • 後からパーティションが増えた際も，増えたぶんだけ ALTER TABLE ADD PARTITION*

* あらかじめ先の期間までパーティションを作成しておけば，ファイルを置いた瞬間クエリ可能になる

列指向フォーマット

指向	特徴
列指向	• カラムごとにデータをまとめて保存 • 特定の列だけを扱う処理では，ファイル全体を読む必要がない • OLAP向き • ORC, Parquet など
行指向	• レコード単位でデータを保存 • 1カラムのみ必要でも，レコード全体を読み込む必要がある • OLTP向き • TEXTFILE(CSV, TSV) など



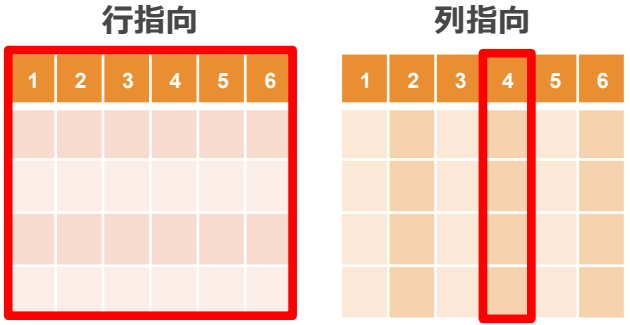
ORCのデータ構造

https://orc.apache.org/docs/spec-intro.html

列指向フォーマットを使うメリット

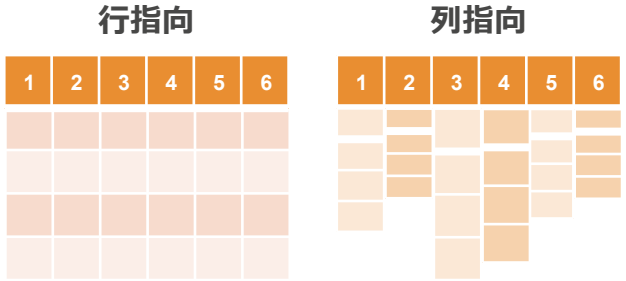
OLAP 系の分析クエリを効率的に実行できる

- たいていの分析クエリは，一度のクエリで一部のカラムしか使用しない
- 単純な統計データなら，メタデータで完結する



I/O の効率があがる

- 圧縮と同時に使うことで I/O 効率がさらに向上
- カラムごとに分けられてデータが並んでいる
- 同じカラムは，似たような中身のデータが続くため，圧縮効率がよくなる



列指向フォーマット & 圧縮の使い方

- CREATE 文で指定するだけ
- データは事前にフォーマット変換 & 圧縮の必要あり
- 分析クエリを投げる際に，使用するカラムだけを読みこむようになる

```
CREATE EXTERNAL TABLE IF NOT EXISTS action_log (
    user_id string,
    action_category string,
    action_detail string
    year int,
    month int,
    day int
)
PARTITIONED BY (year int, month int, day int)
STORED AS PARQUET
LOCATION 's3://athena-examples/action-log/'
TBLPROPERTIES ('PARQUET.COMPRESS'='SNAPPY');
```

```
SELECT
    month
    , action_category
    , COUNT(action_category)
FROM
    action_log
WHERE
    year = 2016
    AND month >= 4
    AND month < 7
GROUP BY
    month
    , action_category
```

パフォーマンスの比較

Parquet & Snappy

```
CREATE EXTERNAL TABLE IF NOT EXISTS elb_logs_pq (
    request_timestamp string,
    elb_name string,
    request_ip string,
    request_port int,
    …
    user_agent string,
    ssl_cipher string,
    ssl_protocol string
)
PARTITIONED BY (year int, month int, day int)
STORED AS PARQUET
LOCATION 's3://athena-examples/elb/parquet/'
TBLPROPERTIES("parquet.compress"="SNAPPY");
```

Regex & Raw file

```
CREATE EXTERNAL TABLE IF NOT EXISTS elb_logs_raw
(
    request_timestamp string,
    elb_name string,
    request_ip string,
    request_port int,
    backend_ip string,
    …
    user_agent string,
    ssl_cipher string,
    ssl_protocol string
  )
  PARTITIONED BY(year string, month string, day string)
  ROW FORMAT SERDE
'org.apache.hadoop.hive.serde2.RegexSerDe'
  WITH   SERDEPROPERTIES (
    'serialization.format' = '1',
    'input.regex' = '([^ ]*) ([^ ]*) ([^ ]*):… ([A-Za-z0-9.-]*)$'
  )
  LOCATION 's3://athena-examples/elb/raw/';
```

パフォーマンスの比較

データセット	S3上のサイズ	クエリ実行時間	スキャンデータ	コスト
Regex & Raw file	1TB	236s	1.15TB	$5.75
Parquet & Snappy	130GB	6.78s	2.51GB	$0.013
削減 / スピードアップ	87%削減	34倍高速	99%削減	99.7%削減

```
SELECT elb_name, uptime, downtime, cast(downtime as double)/cast(uptime as double) uptime_downtime_ratio
FROM (SELECT elb_name,
            sum(case elb_response_code WHEN '200' THEN 1 ELSE 0 end) AS uptime,
            sum(case elb_response_code WHEN '404' THEN 1 ELSE 0 end) AS downtime
        FROM elb_logs_pq GROUP BY elb_name
)
```

https://aws.amazon.com/jp/blogs/news/analyzing-data-in-s3-using-amazon-athena/

Hive on EMR で列指向フォーマットに変換

- EMRを使えば，列指向フォーマットへの変換が手軽に実行可能
- Hive でテーブル定義して，データをロード
- INSERT OVERWRITE で書き込み
- パーティション分け等も同様に実施可能
- Spark での変換* も同様に可能

```
CREATE EXTERNAL TABLE parquet_hive (
    requestBeginTime string,
    adId string, impressionId string,
    referrer string,
    userAgent string,
    userCookie string,
    ip string
)
STORED AS PARQUET
LOCATION 's3://myBucket/myParquet/'
TBLPROPERTIES ('PARQUET.COMPRESS'='SNAPPY');
```

```
INSERT OVERWRITE TABLE parquet_hive
SELECT
    requestbegintime
    , adid
    , impressionid
    , referrer
    , useragent
    , usercookie
    , ip
FROM impressions
WHERE dt='2009-04-14-04-05';
```


* https://github.com/awslabs/aws-big-data-blog/tree/master/aws-blog-spark-parquet-conversion

使いどころ

再掲: Athena の想定ユースケース

ユースケース	データ	ユーザ
新しく取得したデータに対して， DWに入れる価値があるか探索的に検証	新しく取得したデータ	アナリスト
利用頻度の低い過去のデータに対する， BIツール経由のアドホックな分析	コールドデータ	アナリスト
Webサーバで障害が発生したときに， ログを漁って原因追求	アクセスログ	サーバ運用
大規模でないデータに対しての， 低頻度で実施するETL処理	生データ	開発者

Athena が向いていない処理

- リトライ機構がなく，データを絞って高速にスキャンするアーキテクチャのため，バッチ処理には向かない
- 分析処理でも，大量データを長時間処理するのには向かない

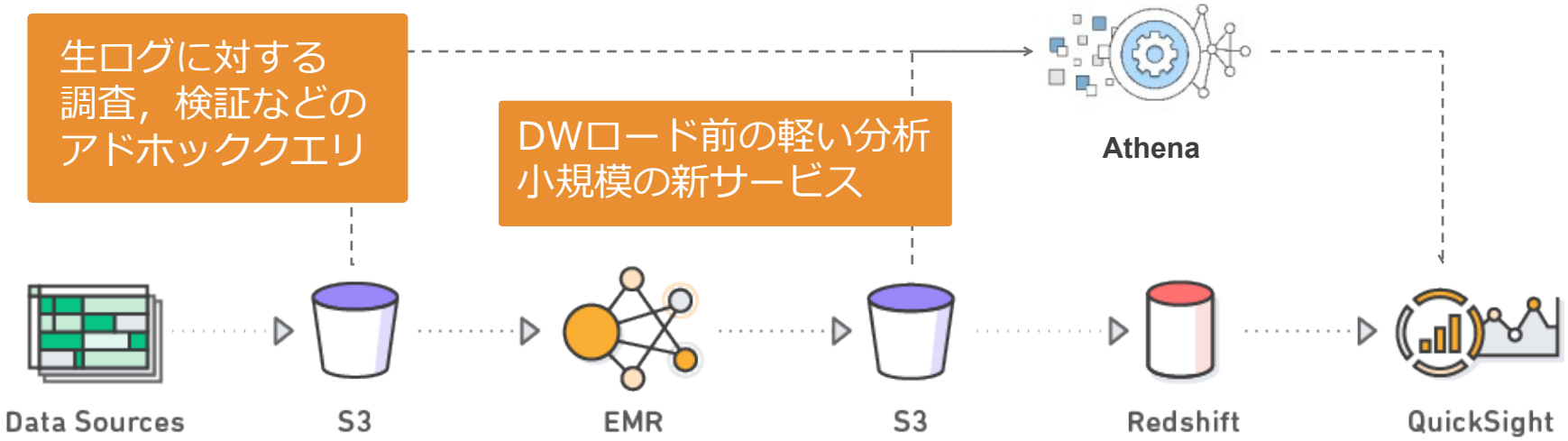
ユースケース	適したサービス
大規模なデータに対して，フルスキャンを定期的に行う処理	EMR
テンポラリテーブルを活用した多段のETL処理	EMR, Glue*
サブクエリやJOINを駆使した複雑な集計処理	Redshift
高頻度なレポーティングのための大量の分析処理	Redshift

* Glue は 2017/3/1 時点ではプレビュー中で，正式リリースされてはいません

リファレンスアーキテクチャ

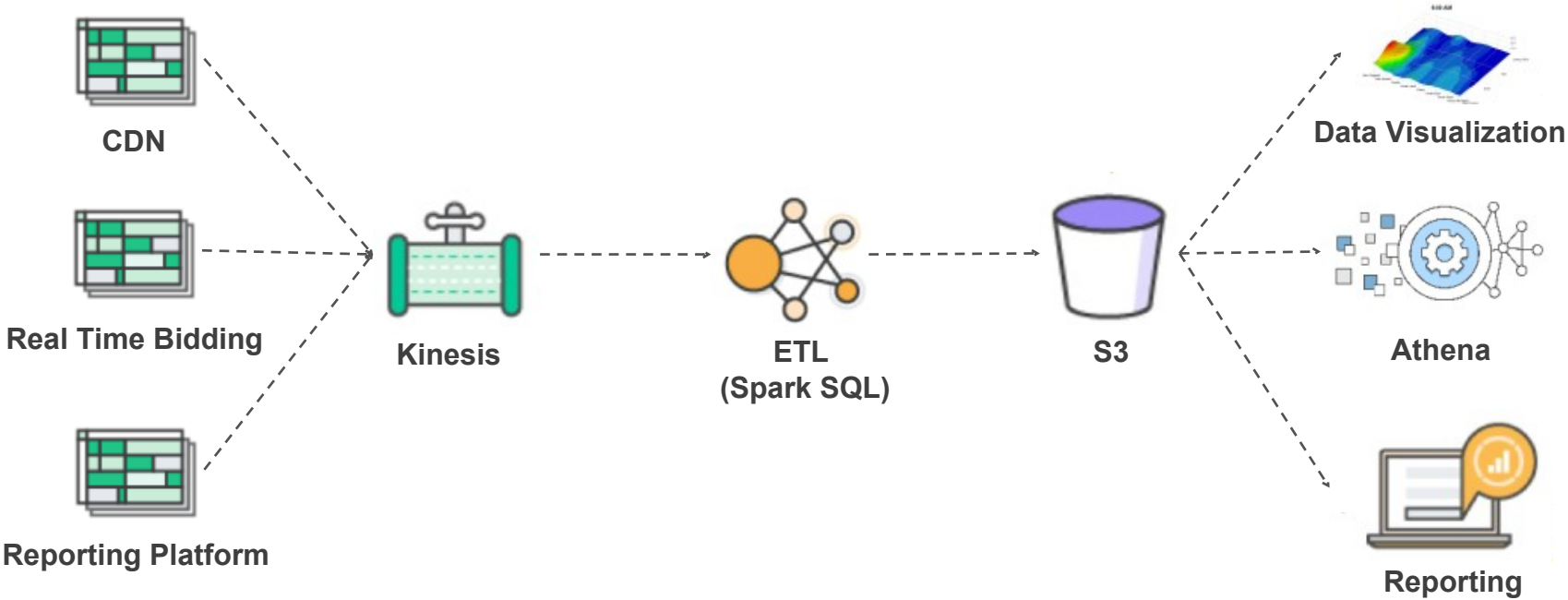
既存のデータパイプラインを補完する形で活用

ちょっとしたダッシュボードのバックは Athena が活躍



ユースケース: DataXu 社

180TB / 日のデータを Athena で分析



http://www.slideshare.net/AmazonWebServices/announcing-amazon-athena-instantly-analyze-your-data-in-s3-using-sql

利用企業のコメント

“Amazon S3 上のデータを直接クエリ処理しているのに、これまで使用してきたシステムよりも高速にクエリ結果を得られたことに大変感銘を受けています。当社はワークロードを積極的に AWS に移行し、今後、Amazon Athena を当社の分析プラットフォームの中核に据えるつもりです”

Gunosy 社

“オープンソースフォーマットを使用していることから、Amazon EMR のような、その他のソリューションで同じデータを活用し、相互運用することもできます。Amazon Athena は特別な管理者を必要としないことから、すぐに使い始めることができました”

Japan Taxi 社

https://aws.amazon.com/jp/about-aws/whats-new/2016/12/introducing-amazon-athena/

リソースに関する制約

Athena のリソースには現状，以下のような制約がある*

- 1アカウントあたりの最大クエリ同時実行数は5個
- クエリは30分でタイムアウト
- 1アカウントあたりの最大データベース数は100個
- 1データベースあたりの最大テーブル数は100個
- 1テーブルあたりの最大パーティション数は20000個

* 上限緩和申請によって増やすことは可能

http://docs.aws.amazon.com/athena/latest/ug/service-limits.html

I/Oに関する制約

入力に関する制約

- どのリージョンの S3 からでもデータの読み込みは可能
- ただしリージョンをまたぐ場合，データ転送時間と，転送料金がかかる
- 暗号化データ読み込みは SSE-S3, SSE-KMS, CSE-KMS に対応

出力に関する制約

- Athena 実行リージョンの S3 にヘッダ付き csv で出力のみ
- 結果ファイルは SSE-S3, SSE-KMS, CSE-KMS の3方式で暗号化して出力可能
- SELECT INSERT には未対応

データアクセスに関する制約

Athena のデータアクセスには現状，次のような制約がある

- アクセス手段はコンソールまたはJDBCドライバ経由のみ
  APIやCLIは提供していない
- データベース単位，テーブル単位など，リソースごとのアクセス制御は未対応
- CREATE TABLE の際に，読み込ませない行を指定できない
  (Hive DDL の ‘skip.header.line.count’ オプションは未対応*)

* Presto 自体が Hive の上記オプションに対応していないことに起因するもの

料金体系

- クエリ単位の従量課金
- S3 のデータスキャンに対して，$5 / 1TB の料金
  - バイト数はメガバイト単位で切り上げられ，10MB 未満のクエリは 10MB と計算される
- 別リージョンからデータを読み込む場合には，別途S3のデータ転送料金がかかる
- DDL のクエリや，実行に失敗したクエリの料金は無料
- パーティション，列指向フォーマット，圧縮を活用することで，スキャンするデータ量を減らして，コスト削減が可能

Update

2017/2/16 に以下の新機能＆性能改善をリリース

新機能

- 新たな SerDe のサポート（Avro, OpenCSVSerDe）
- オハイオリージョン（us-east-2）のローンチ
- テーブル追加ウィザードで，カラム名/型をまとめて入力可能に

性能改善

- 大規模な Parquet テーブルのクエリ速度を改善

http://docs.aws.amazon.com/athena/latest/ug/release-notes.html

OpenCSVSerDe

- 囲み文字を quoteChar で指定できる
- OpenCSVSerDe 自体の制約として，以下の2点に注意
  - カラム型として string しか選択できない
  - 文字列内の改行文字はサポートされていない

```
"a1", "a2", "a3", "a4”
,"1" ,"2" "abc", "def”
"a", "a1", "abc3", "ab4"
```

```
CREATE EXTERNAL TABLE myopencsvtable (
   col1 string,
   col2 string,
   col3 string,
   col4 string,
)
ROW FORMAT SERDE  'org.apache.hadoop.hive.serde2.OpenCSVSerde’
WITH SERDEPROPERTIES (
   'separatorChar' = ',',
   'quoteChar' = '\"',
   'escapeChar' = '\\’
   )
STORED AS TEXTFILE
LOCATION 's3://location/of/csv/';
```

オハイオリージョンのローンチ

オハイオリージョンで Athena が実行可能に

2017/3/24 に以下の新機能＆性能改善をリリース

新機能

- 新たな SerDe のサポート（CloudTrailSerDe*）

性能改善

- パーティション数が非常に多い場合のスキャン速度を改善
- MSCK REPAIR TABLE の速度を改善
- オハイオリージョンでも他リージョンの S3 からのデータ読み込みが可能に

http://docs.aws.amazon.com/athena/latest/ug/release-notes.html
* https://aws.amazon.com/blogs/big-data/aws-cloudtrail-and-amazon-athena-dive-deep-to-analyze-security-compliance-and-operational-activity /

2017/4/6 に以下の新機能をリリース

新機能

- 入出力データの暗号化に対応
- 新バージョンの JDBC ドライバ (AthenaJDBC41-1.0.1.jar) リリース
- ALTER TABLE で カラムの追加，置き換え，変更処理が可能に
- LZO 圧縮形式に対応

http://docs.aws.amazon.com/athena/latest/ug/release-notes.html
* https://aws.amazon.com/blogs/big-data/aws-cloudtrail-and-amazon-athena-dive-deep-to-analyze-security-compliance-and-operational-activity /

入出力データの暗号化に対応

- 暗号化方式として， SSE-S3, SSE-KMS, CSE-KMS に対応
- 暗号化データへのクエリ，とクエリ結果の暗号化保存，は**個別に設定**する必要がある
  - そのため，**両者で暗号化方式が違っていてもOK**
- マネジメントコンソール，JDBCの両方に対応

入出力	手段	ユーザ
入力	共通	CREATE TABLE の tablproperties で指定
出力	コンソール	画面右上の Settings から暗号化方式を指定
	JDBC	JDBC ドライバのオプションで指定

Tips

JOIN クエリのテーブルの並べ方

JOIN は大きなテーブルを左側に持ってくるほうがよい

右側のテーブルを各ノードに Broadcast して JOIN するため

Presto には Cost-based optimizer がないため，自動で最適化されない

ただし，結合条件によっては CROSS JOIN が行われる場合もあるので注意*

```
FROM
  small_table
  INNER JOIN large_table
  ON large_table.id = small_table.id
```

```
FROM
  large_table
  INNER JOIN small_table
  ON large_table.id = small_table.id
```

* https://aws.amazon.com/jp/blogs/news/top-10-performance-tuning-tips-for-amazon-athena/

GROUP BY のカラムの並べ方

GROUP BY に複数のカラムを指定する場合には，カーディナリティが高い順番に並べるほうがよい

前に並んだカラムの値に応じて，各ノードにデータを割り振るため，カーディナリティが低いデータだと，処理が十分に並列化されない

```
FROM
    action_log
GROUP BY
    low_card_col
    , high_card_col
```

```
FROM
    action_log
GROUP BY
    high_card_col
    , low_card_col
```


https://aws.amazon.com/jp/blogs/news/top-10-performance-tuning-tips-for-amazon-athena/

ORDER BY の件数を制限する

集計クエリで ORDER BY を行う場合，実際にはトップ100件くらいまでしか確認しないようなケースが多い

ORDER BY を行う際には，データを単一ノードに集約してソートするため，基本的に計算を並列化できない

そのため，LIMIT 100 をつけてあげることで，ソート処理を高速に行うことができる

```
SELECT
    *
FROM
    lineitem
ORDER BY
    l_shipdate
```

```
SELECT
    *
FROM
    lineitem
ORDER BY
    l_shipdate
LIMIT
    100
```


https://aws.amazon.com/jp/blogs/news/top-10-performance-tuning-tips-for-amazon-athena/

LIKE 演算子を使うべきではないケース

LIKE 演算子で多数の比較を行う場合には，RegEx で一括処理をしたほうが効率がよい

それぞれの LIKE 演算子について捜査が走るため，数が増えるほど，また文字列が長くなるほど，処理が重くなる

```
WHERE
    comment LIKE ‘%wake%’
    , comment LIKE ‘%regular%’
    , comment LIKE ‘%expres%’
    , comment LIKE ‘%sleep%’
    , comment LIKE ‘%hello%’
```

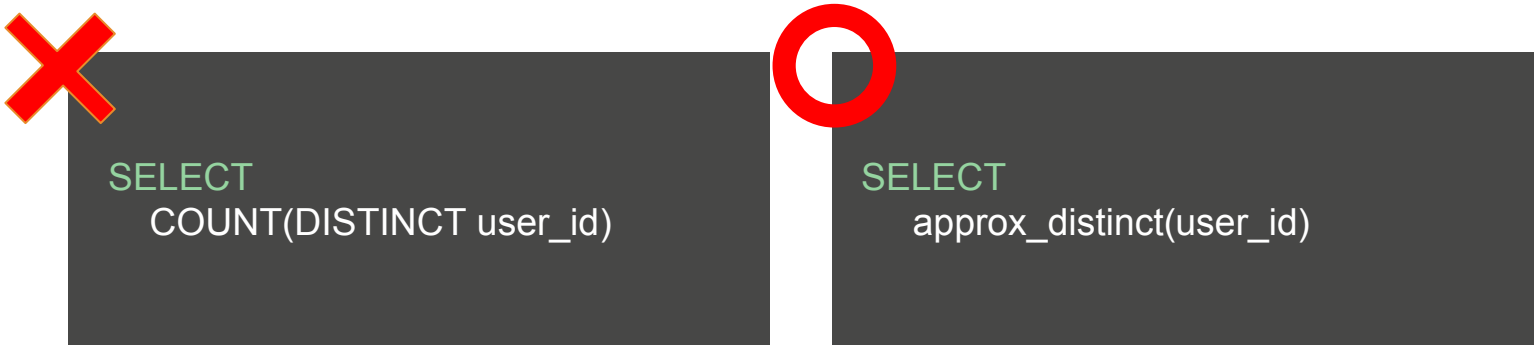
```
WHERE
    regexp_like(comment, 'wake|regular|express|sleep|hello')
```


https://aws.amazon.com/jp/blogs/news/top-10-performance-tuning-tips-for-amazon-athena/

近似関数で計算コストを下げる

COUNT DISTINCT でユニーク数を求める際に，正確性より計算速度のほうが大事なときは，approx_distinct() で求めた近似値を使うとよい

approx_distinct() は HyperLogLog で近似値を求めているため，実際のユニーク数が多い場合でも高速に動作する



https://github.com/prestodb/presto/blob/master/presto-main/src/main/java/com/facebook/presto/operator/aggregation/ApproximateCountDistinctAggregations.java
https://github.com/airlift/airlift/blob/master/stats/src/main/java/io/airlift/stats/cardinality/HyperLogLog.java
https://github.com/airlift/airlift/blob/master/stats/docs/hll.md

テーブル / カラム名についての注意点

アンダースコアから始まるときは，バッククォートで囲む
アンダースコア以外の記号を使うことはできない

```
CREATE EXTERNAL TABLE _action_log (
    _user_id string,
    _action_category string,
```

```
CREATE EXTERNAL TABLE `_action_log`
(
    `_user_id` string,
    action_category string,
```

大文字小文字の区別はない

```
SELECT ACTION_CATEGORY
FROM `_ACTION_LOG`
```

テーブル / カラム名についての注意点

予約語を使うことも可能
ただし，DDL ではバッククォートで，クエリではダブルクォートで囲む必要がある
クエリでシングルクォートで囲むと，単なる文字列として扱われる

```
CREATE EXTERNAL TABLE `join` (
    `where` string,
    action_category string,
```

```
SELECT “where”, ‘where’ TABLE FROM “join”;
```

Japan	where
China	where
Brazil	where
Japan	where
France	where

データの場所についての注意点

S3のキーについては，必ず最後を `/` で終える必要がある
‘/’ までが一致するキーが，すべてテーブルの中身として認識される
ワイルドカードや，オブジェクト名の直指定は不可
パーティションがある場合には，MSCK や ADD PARTITION を忘れずに

```
LOCATION 's3://athena-examples/action-log/’
```

```
LOCATION 's3://athena-examples/action-log’
LOCATION ‘s3://athena-examples/action-log/*’
LOCATION 's3://athena-examples/action-log/data.csv’
```

JSON データ読み込みについての注意点

正しい SerDe を指定する

```
ROW FORMAT SERDE ’org.apache.hive.hcatalog.data.JsonSerDe’
ROW FORMAT SERDE ’org.openx.data.jsonserde.JsonSerDe’
```

JSON レコード内に改行が入っているとエラーになる (SerDeの仕様)

```
{
    “col1”: “val1”,
    “col2”: “val2”
}
```

```
{ “col1”: “val1”, “col2”: “val2” }
```

不正な形式のレコードはオプションで無視することが可能

```
WITH SERDEPROPERTIES ('ignore.malformed.json' = 'true')
```

https://aws.amazon.com/blogs/big-data/create-tables-in-amazon-athena-from-nested-json-and-mappings-using-jsonserde/
https://aws.amazon.com/premiumsupport/knowledge-center/error-json-athena/

まとめ

まとめ

- Athena は S3 上のデータに対して，標準 SQL によるインタラクティブなクエリを投げてデータの分析を行うことができるサービス
- 分析系のアドホッククエリに向いており，QuickSight や JDBC 経由で SQL クライアントから，簡単にアクセスできる
- パーティション，データフォーマット，圧縮形式によりパフォーマンスが向上
- すべてを Athena でやるのではなく，適材適所で使い分ける


amazon
web services