Amazon Timestream

新機能紹介と実演

Yosuke Nishihara

アマゾン ウェブ サービス ジャパン合同会社

2023/09

Agenda

1. バッチロード
2. AWS Backup との統合
3. アンロード
4. ユーザ定義のパーティションキー

参考：https://docs.aws.amazon.com/timestream/latest/developerguide/batch-load.html

1. バッチロード

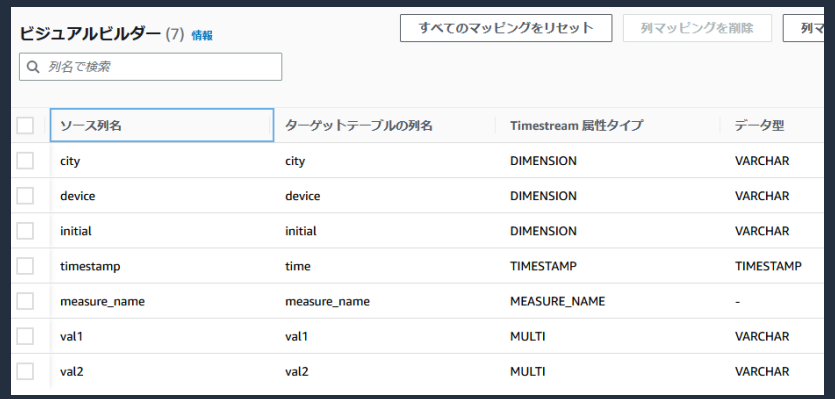
S3 に保存した CSV ファイルを直接 Amazon Timestream に取り込めるようになり、他データベースからのデータ移行がプログラミングレスで容易に実装できる。ジョブの並列実行により高速なローディングを実現。

1.データファイルの準備

```
id1,id2,timestamp,m_name,val1,val2
"009","K","1679273577275",”measure","27.8","-21.5"
"006","A","1679273577276"," measure ","13.4","21.5"
"005","H","1679273577278"," measure ","-11.9","-13.7"
"001",“B","1679273577279"," measure ",“8.3",“6.5"
"002",“M","1679273577288"," measure ","-4.1","-9.1"
…
```

S3 に CSV ファイルを配置。メジャー値とディメンジョン値が最低１つは必要。また、タイムスタンプは UNIX 時間で記述し、ファイルサイズは 5GB まで。

2.データマッピングを記述



ジョブの作成時にマッピングを定義。ロード先のテーブルとともに、各カラムと属性タイプ、データ型を指定。

3.ジョブ実行

ジョブ実行後、レコードの処理数やエラー発生数、エラー理由等が確認出来る。

デモの内容

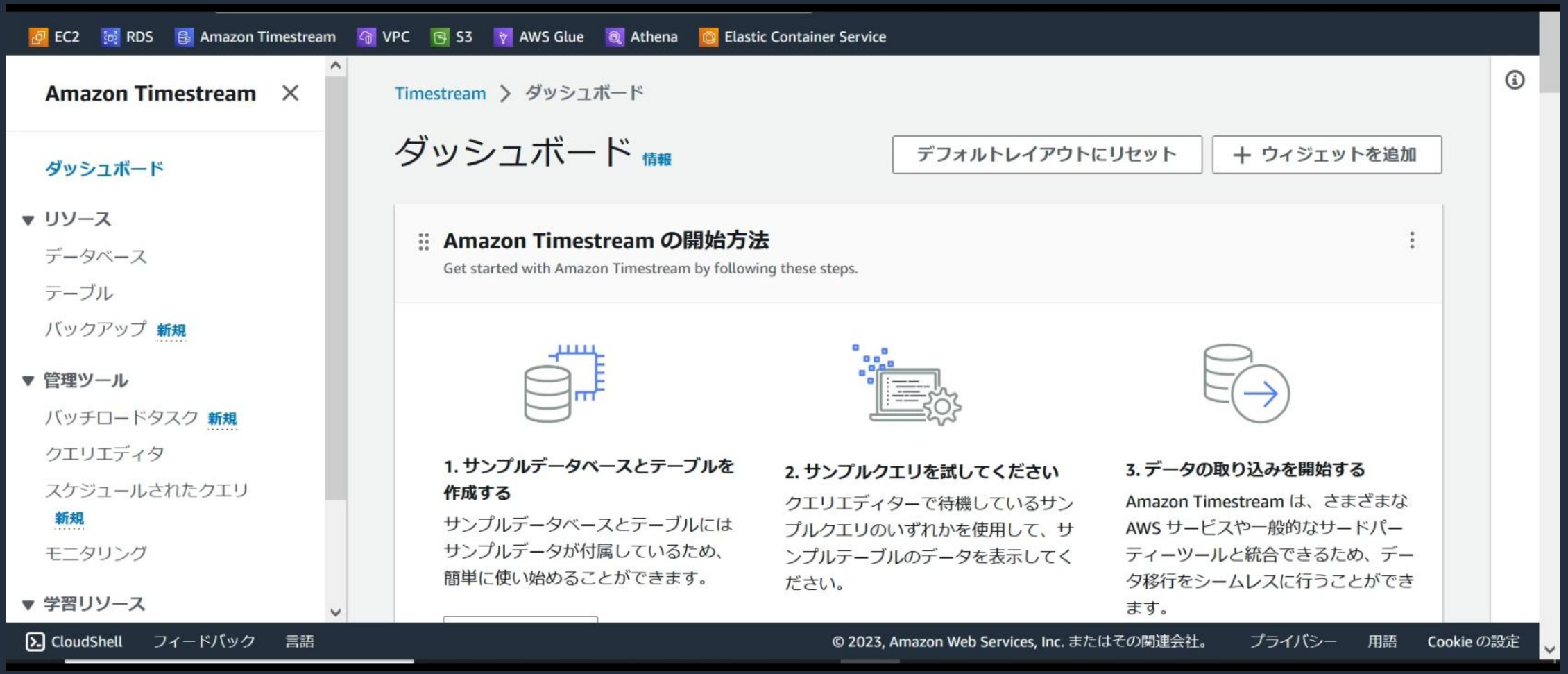
- Amazon S3 にテスト用データを配置
- Visual Editor を使ったデータマッピングを実施
- ロードの実行
- データの確認

使用するデータ

ディメンジョン

メジャー

time	deviceId	location	measure_name	temperature	humidity
2023-09-05 19:00:00.000498	394	東京	metrics	32.1	80
2023-09-05 19:00:01.020958	1543	大阪	metrics	35.1	81
2023-09-05 19:00:02.031958	9873	福岡	metrics	33.7	76
2023-09-05 19:00:03.487059	6819	奈良	metrics	35.8	80
2023-09-05 19:00:03.261059	49	佐賀	metrics	31.5	67
2023-09-05 19:00:03.994651	9200	京都	metrics	36.9	81
2023-09-05 19:00:03.019263	1048	北海道	metrics	24.1	54
2023-09-05 19:00:04.138948	99	埼玉	metrics	34.3	79
2023-09-05 19:00:04.713981	5081	高知	metrics	36.4	75
2023-09-05 19:00:05.669690	7756	徳島	metrics	35.5	74
…	…	…	…	…	…



特記事項

- CSV ファイルは 100MB ～ 1GB 程度が望ましい
- 並列にジョブを実行する事で、パフォーマンス改善が期待出来る
- バッチロードタスクの実行中は、同じテーブルへのデータ書き込みはタスクのエラーにつながる可能性があり、避けた方が良い
- バッチロードタスクの実行中は、対象の S3 バケットのファイルの変更、削除はタスクエラーの原因となる可能性があります

ユースケース

- 別 DB から Timestream へデータ移行
- オンライン停止時間帯でのデータコピー
- 検証作業

参考： https://docs.aws.amazon.com/timestream/latest/developerguide/backups.html

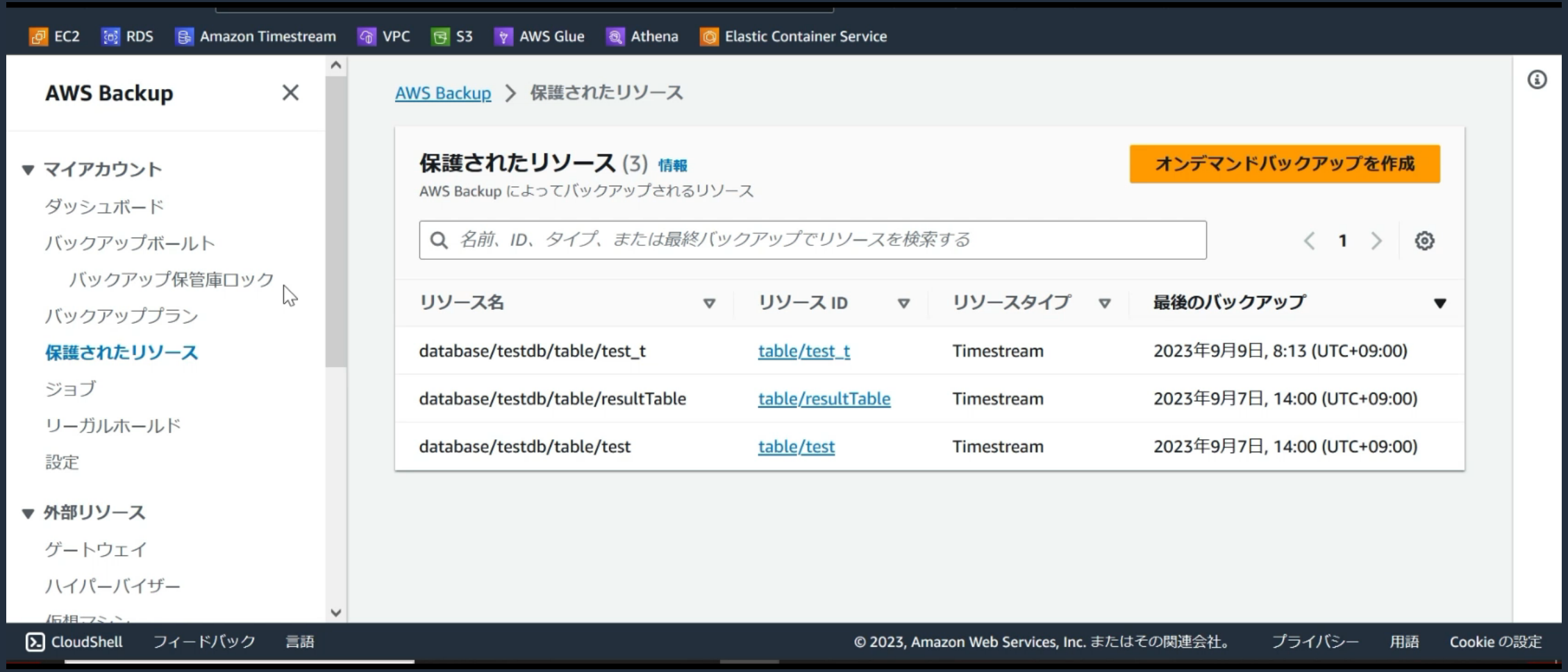
2. AWS Backup との統合

AWS Backup との統合により、フルマネージドで Timestream のバックアップを取得・管理できるようになった

- 表単位でのバックアップの作成
  - オンデマンド、もしくはスケジューリングによる自動取得が可能
  - Incremental Backup をサポート
- データの復元
  - ソースとは独立した新規の表として復元
  - リージョンやアカウントを跨いだ復元 (複製) も可能
  - 全てのデータの復元、もしくは特定期間のデータを選択した復元が可能

デモの内容

- AWS Backup で Timestream テーブルのバックアップを実行
- 取得したバックアップを使ってテーブルを復元



参考：https://docs.aws.amazon.com/timestream/latest/developerguide/export-unload.html

3. アンロード

SQL の実行結果を **Parquet／CSV フォーマットで S3 に出力**できる。出力データの圧縮や、パーティション化、暗号化にも対応。Amazon Athena や Amazon Redshift を利用した時系列データの分析を簡単に実装できる。

```
UNLOAD
(
    SELECT user_id, measure_name, time, quantity, product_id, channel
    FROM sample_clickstream.sample_shopping
    WHERE time BETWEEN ago(2d) AND now()
)
TO 's3://<bucket_name>/partitionbychannel/'
WITH
(
    partitioned_by = ARRAY ['channel'],
    format='CSV',
    compression='GZIP'
)
```

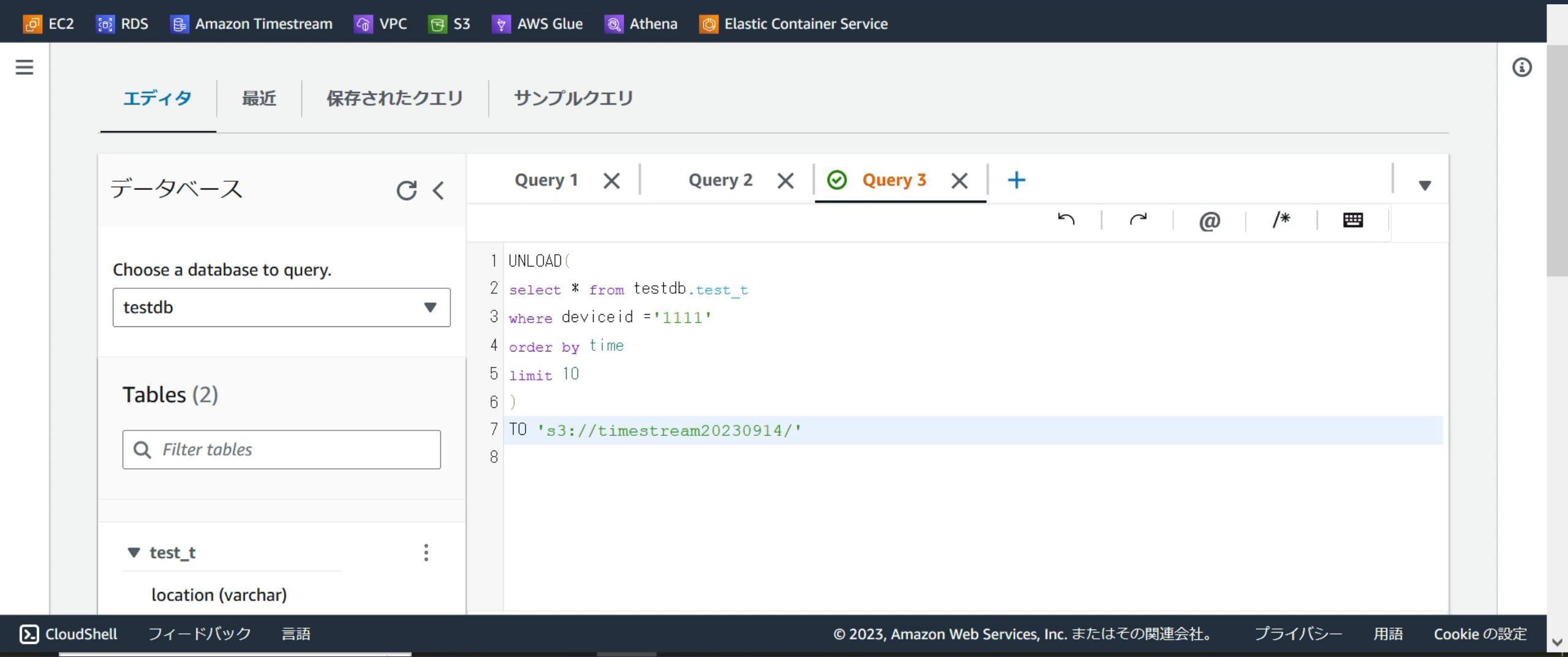
← S3 に出力する SQL 文を記述

← 出力先の S3 のパスを記述

各種オプションを指定
- 指定したカラムでのパーティション化
- 出力フォーマット
- 圧縮有無　等

デモの内容

- UNLOAD の実行
- S3 に出力されたデータの確認



UNLOAD 設定オプション

オプション名	内容	デフォルト
PARTIIONED_BY	パーティションの為のカラム名を ARRAY 句と併せて指定する	-
FORMAT	出力ファイルフォーマット（CSV または PARQUET）	CSV
COMPRESSION	圧縮有無（ GZIP または NONE ）	GZIP
ENCRYPTION	暗号化指定（ SSE_KMS または SSE_S3 ）	SSE_S3
KMS_KEY	顧客管理の KMS 鍵を記載	-
FIELD_DELIMITER	区切り文字	カンマ
ESCAPED_BY	エスケープ文字	バックスラッシュ

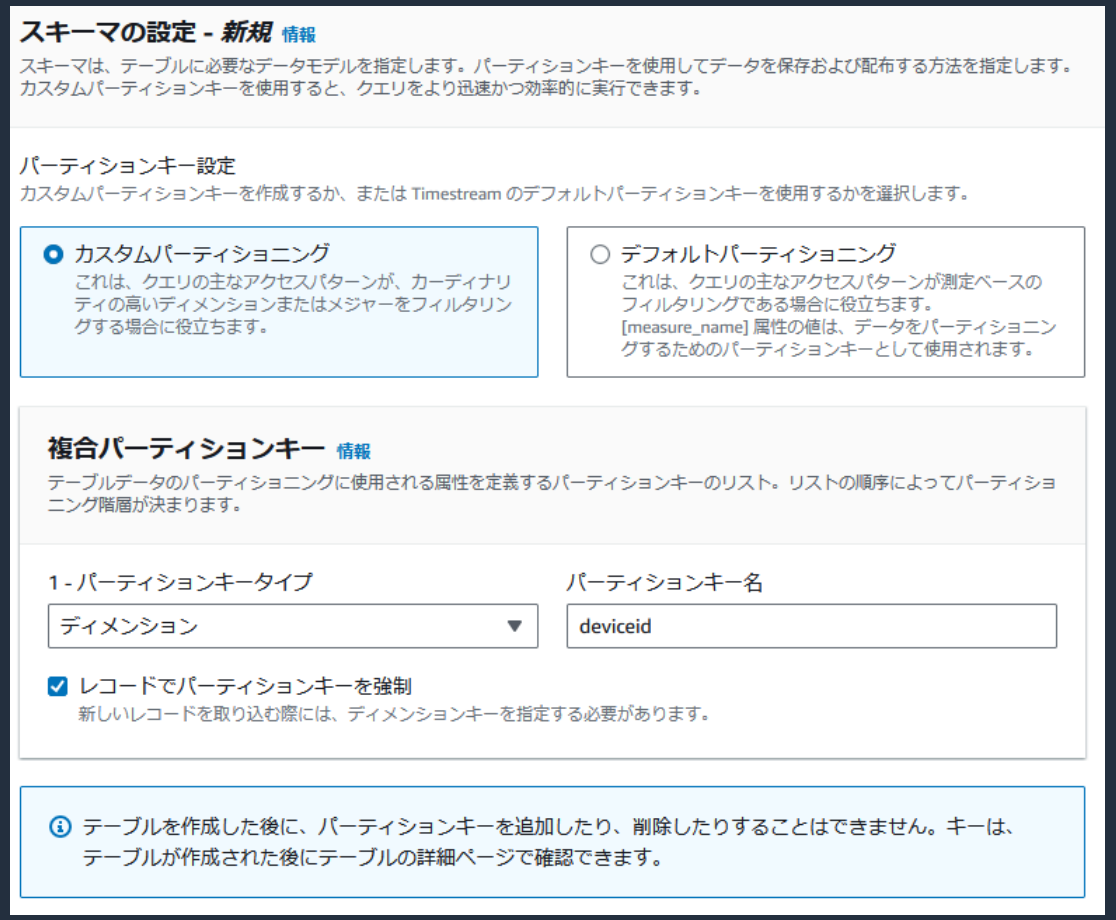
UNLOAD 利用時の注意点

- タイムアウト (60分) を考慮すると、出力データのサイズは 60 GB 以内に抑えた方が良い
- UNLOAD が何らかのエラーで終了した場合には、不完全なデータが S3 に出力される可能性有
- partition_by 句で指定するカラムは、参照クエリの最後に指定し、データは ASCII のみ許容。また、101 個以上のパーティションは作成出来ない。

参考：https://docs.aws.amazon.com/timestream/latest/developerguide/customer-defined-partition-keys.html

4. ユーザ定義のパーティションキー

ディメンジョンをテーブルのパーティションのキーとして個別に設定できるようになり、設計時の選択肢が広がった



特徴

- 以下の条件に合致する場合、クエリのパフォーマンス改善に有用
  - デフォルトの measure_name でのパーティショニングでは目標のクエリパフォーマンスを達成出来ない
  - カーディナリティの高いディメンジョンがあり、かつ、改善が必要なクエリの where 条件に対象のディメンジョンが含まれる
- パーティションキーが必須かどうかをキー作成時に指定可能

注意点

- テーブル作成後にはパーティションキーを作成、変更、削除する事が出来ない為、入念な事前確認が必要

Amazon Timestream の構成要素と制約

ディメンジョン

メジャー

time	deviceId	location	measure_name	temperature	humidity	...
2023-09-05 19:00:00	49820	東京	metrics	32.1	80	
2023-09-05 19:00:01	1543	大阪	metrics	35.1	81	
2023-09-05 19:00:02	59873	福岡	metrics	33.7	76	
2023-09-05 19:00:03	6819	奈良	metrics	35.8	80	
2023-09-05 19:00:03	49	佐賀	metrics	31.5	67	
2023-09-05 19:00:03	9200	京都	metrics	36.9	81	
2023-09-05 19:00:03	1048	北海道	metrics	24.1	54	
2023-09-05 19:00:04	99	埼玉	metrics	34.3	79	
2023-09-05 19:00:04	439081	高知	metrics	36.4	75	
2023-09-05 19:00:05	4958	徳島	metrics	35.5	74	

入力可能なメジャー値は最大 256 個

最大で 8,192 種類の値

■ シングルメジャーレコード

単一のメジャーを1 つのテーブル行に格納するデータモデル

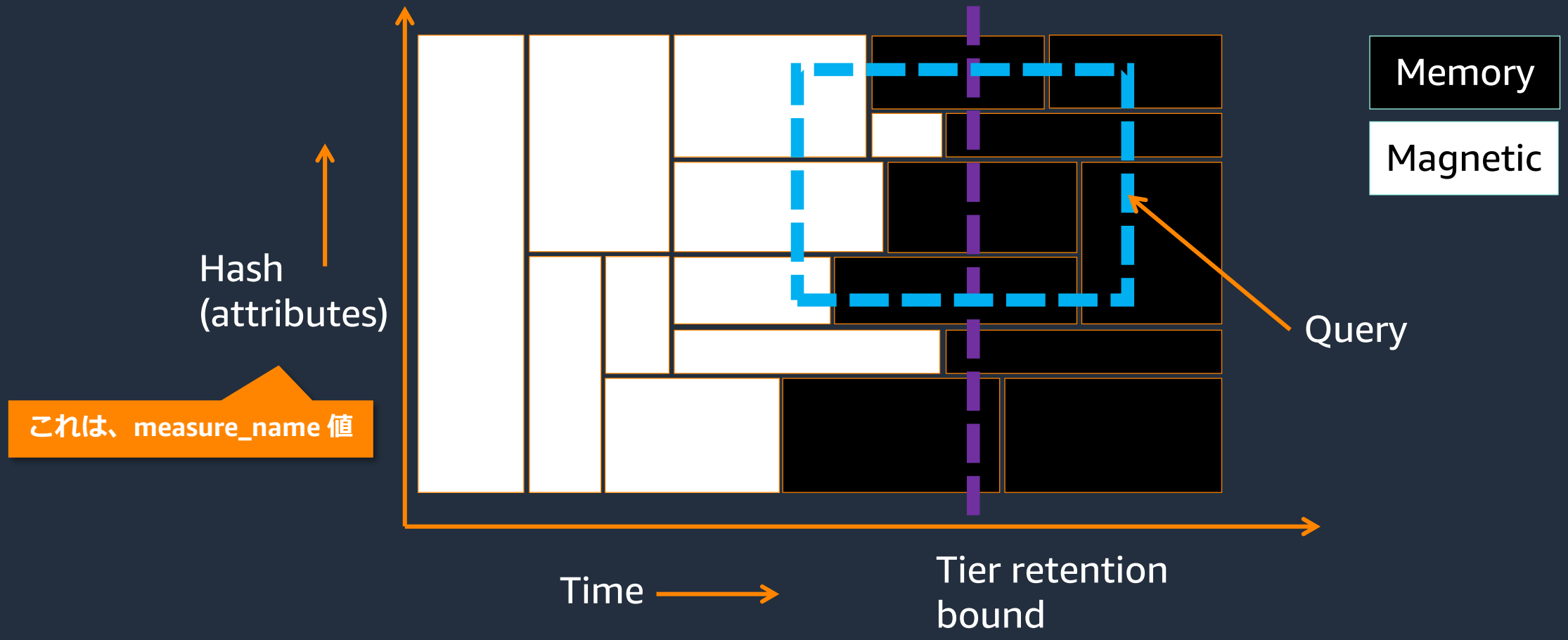
time	deviceId	location	measure_name	measure_value:double
2023-09-05 19:00:00	49820	東京	temperature	32.1
2023-09-05 19:00:00	49820	東京	humidity	80
2023-09-05 19:00:02	59873	福岡	temperature	33.7
2023-09-05 19:00:02	59873	福岡	humidity	76
2023-09-05 19:00:02	59873	福岡	temperature	33.7
2023-09-05 19:00:02	59873	福岡	humidity	76

■ マルチメジャーレコード (re:Invent2021 より追加されたデータモデル)

複数のメジャーを 1 つのテーブル行に格納することが可能な新しいデータモデル。デバイスが複数の測定値を同時に送信した場合に、データ書き込み・保存・クエリスキャンのコスト効率に優れる。

time	deviceId	location	measure_name	temperature	humidity
2023-09-05 19:00:00	49820	東京	metrics	32.1	80
2023-09-05 19:00:01	1543	大阪	metrics	35.1	81
2023-09-05 19:00:02	59873	福岡	metrics	33.7	76

自動パーティショニング/インデクシングによるスケーリング



ディメンジョン

メジャー

time	deviceId	location	measure_name	temperature	humidity	...		
2023-09-05 19:00:00	49820	東京	metrics	32.1	80			
2023-09-05 19:00:01	1543	大阪	metrics	35.1	81			
2023-09-05 19:00:02	59873	福岡	metrics	33.7	76			
2023-09-05 19:00:03	6819	奈良	metrics	35.8	80			
2023-09-05 19:00:03	49	佐賀	metrics	31.5	67			
2023-09-05 19:00:03	9200	京都	metrics	36.9	81			
2023-09-05 19:00:03	1048	北海道	metrics	24.1	54			
2023-09-05 19:00:04	99	埼玉	metrics	34.3	79			
2023-09-05 19:00:04	439081	高知	metrics	36.4	75			
2023-09-05 19:00:05	4958	徳島	metrics	35.5	74			

- measure_name が固定値だと、データが効率的に分散されない

(元々の) ベストプラクティス

データを分散させる為に、dimension の値の Hash 値を 8,192 で割った余りを measure_name に設定する事を推奨していた

time	deviceId	location	measure_name	temperature	humidity	...	
2023-09-05 19:00:00	49820	東京	3948	32.1	80		
2023-09-05 19:00:01	1543	大阪	6781	35.1	81		
2023-09-05 19:00:02	59873	福岡	8100	33.7	76		
2023-09-05 19:00:03	6819	奈良	376	35.8	80		
2023-09-05 19:00:03	49	佐賀	221	31.5	67		
2023-09-05 19:00:03	9200	京都	7530	36.9	81		
2023-09-05 19:00:03	1048	北海道	5550	24.1	54		
2023-09-05 19:00:04	99	埼玉	6210	34.3	79		
2023-09-05 19:00:04	439081	高知	2321	36.4	75		
2023-09-05 19:00:05	4958	徳島	4432	35.5	74		

hash(deviceid) % 8192

ユーザ定義のパーティション機能が登場

任意の dimemsion 値にパーティションキーを設定できるようになった

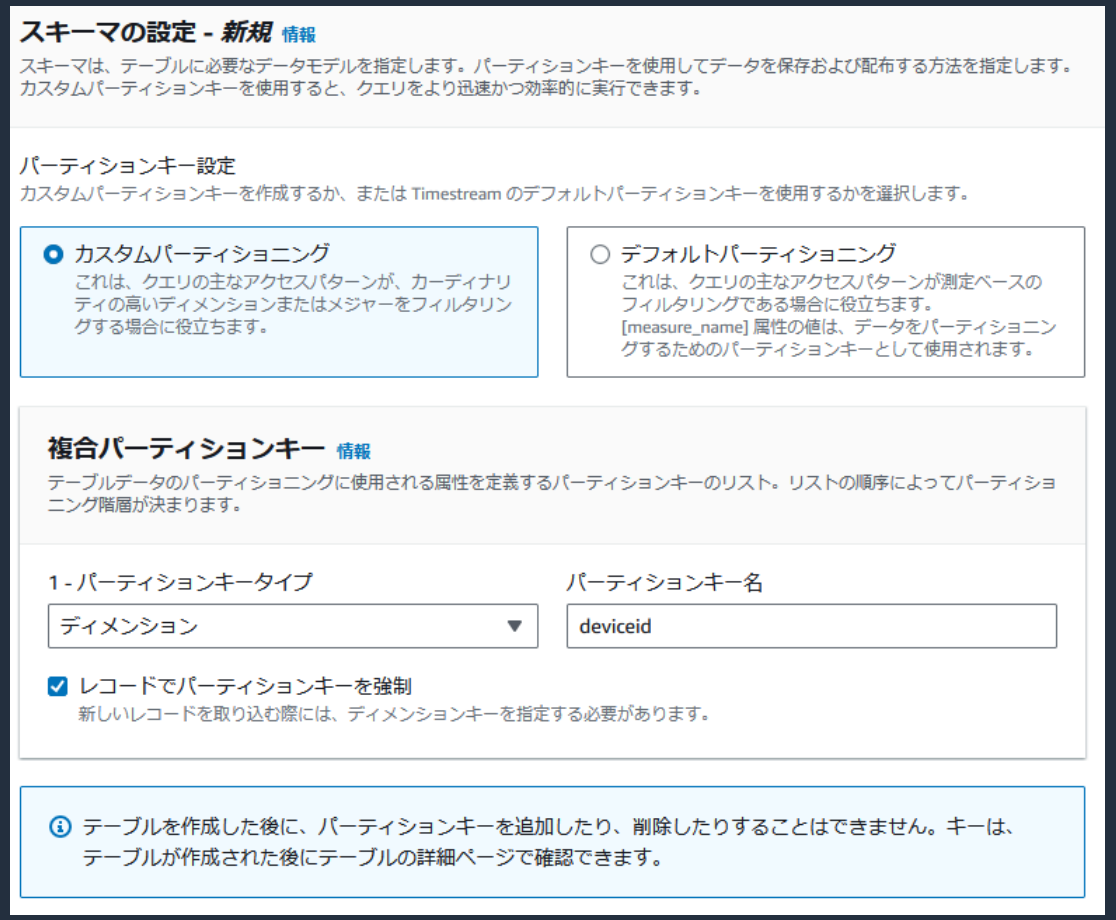
time	deviceId	location	measure_name	temperature	humidity	...	
2023-09-05 19:00:00	49820	東京	metrics	32.1	80		
2023-09-05 19:00:01	1543	大阪	metrics	35.1	81		
2023-09-05 19:00:02	59873	福岡	metrics	33.7	76		
2023-09-05 19:00:03	6819	奈良	metrics	35.8	80		
2023-09-05 19:00:03	49	佐賀	metrics	31.5	67		
2023-09-05 19:00:03	9200	京都	metrics	36.9	81		
2023-09-05 19:00:03	1048	北海道	metrics	24.1	54		
2023-09-05 19:00:04	99	埼玉	metrics	34.3	79		
2023-09-05 19:00:04	439081	高知	metrics	36.4	75		
2023-09-05 19:00:05	4958	徳島	metrics	35.5	74		

deviceid をパーティションキーに設定する事が出来る

参考：https://docs.aws.amazon.com/timestream/latest/developerguide/customer-defined-partition-keys.html

4. ユーザ定義のパーティションキー

ディメンジョンをテーブルのパーティションのキーとして個別に設定できるようになり、設計時の選択肢が広がった



特徴

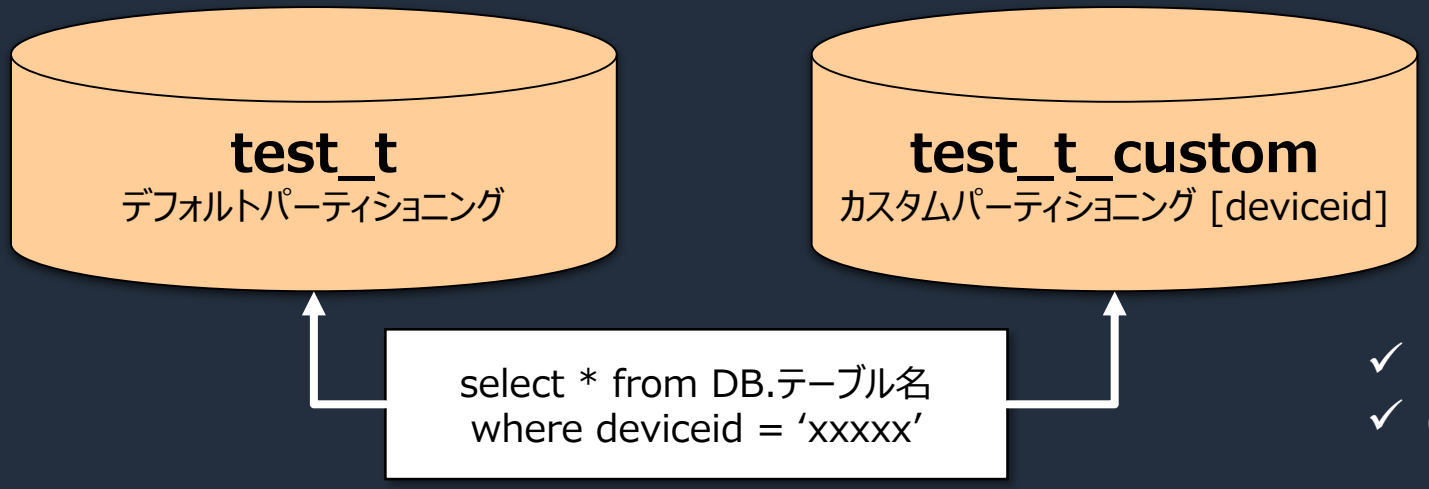
- 以下の条件に合致する場合、クエリのパフォーマンス改善に有用
  - デフォルトの measure_name でのパーティショニングでは目標のクエリパフォーマンスを達成出来ない
  - カーディナリティの高いディメンジョンがあり、かつ、改善が必要なクエリの where 条件に対象のディメンジョンが含まれる
- パーティションキーが必須かどうかをキー作成時に指定可能

注意点

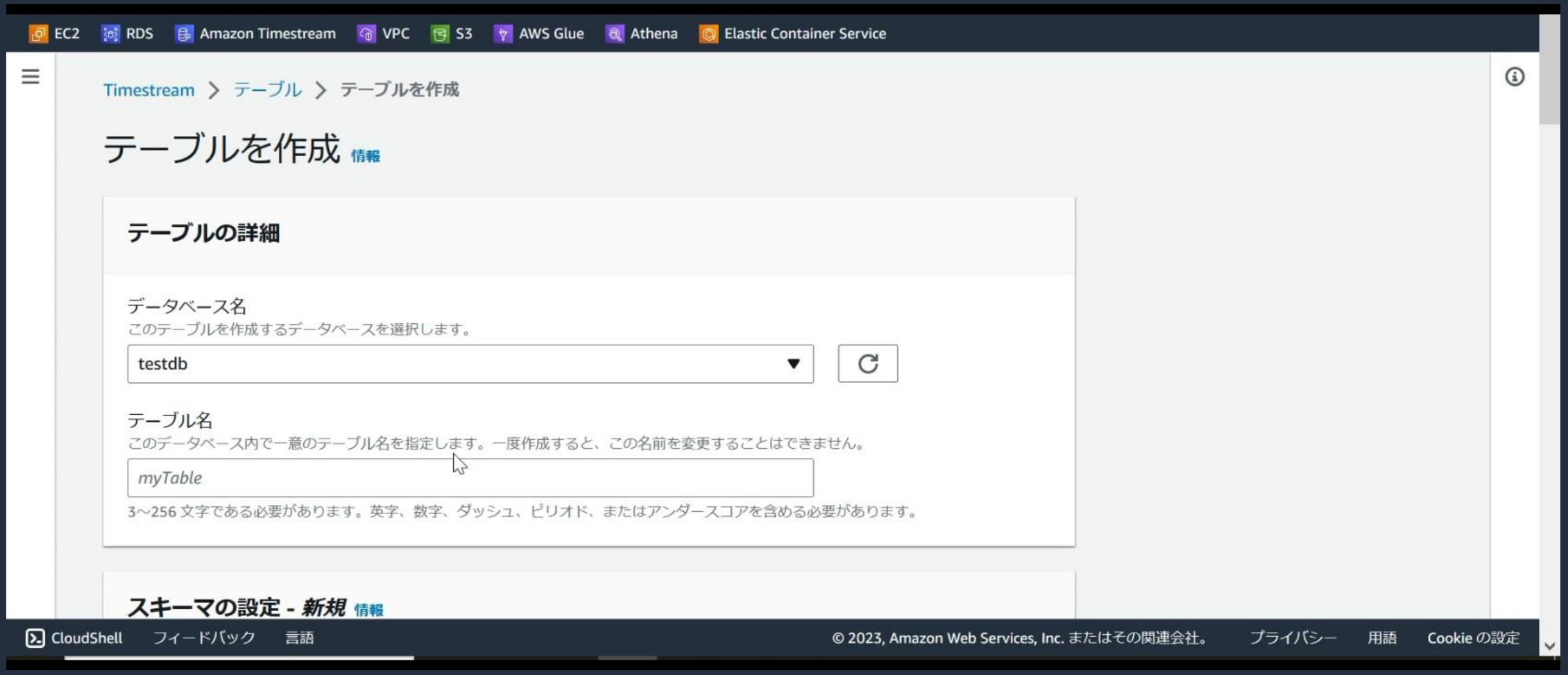
- テーブル作成後にはパーティションキーを作成、変更、削除する事が出来ない為、入念な事前確認が必要

デモの内容

- カスタムパーティションキーの設定したテーブルの作成
- SQL の実行
  - デフォルト／カスタムパーティショニングを設定したそれぞれのテーブルに対して同一 SQL を実行して性能差を確認



✓ レコード数は 1,000 万件
✓ deviceid の取りうる値は 100万個



Amazon Timestream の日本語 Blog を提供しています

～ “aws blog timestream” で検索してみて下さい ～

Thank you!