[AWS Black Belt Online Seminar] Amazon Timestream

サービスカットシリーズ

Solutions Architect 浅井 ももこ
2020/12/16

AWS 公式 Webinar
https://amzn.to/JPWebinar

過去資料
https://amzn.to/JPArchive

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

自己紹介

浅井 ももこ (Momoko Asai)

所属

- アマゾン ウェブ サービス ジャパン株式会社
  技術統括本部 ソリューションアーキテクト

好きなサービス

- Amazon Timestream
- Amazon Aurora

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

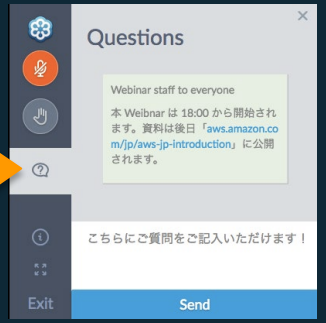
AWS Black Belt Online Seminar とは

「サービス別」「ソリューション別」「業種別」のそれぞれのテーマに分かれて、アマゾン ウェブ サービス ジャパン株式会社が主催するオンラインセミナーシリーズです。

質問を投げることができます！

- 書き込んだ質問は、主催者にしか見えません
- 今後のロードマップに関するご質問は
  お答えできませんのでご了承下さい

① 吹き出しをクリック
② 質問を入力
③ Sendをクリック



Twitter ハッシュタグは以下をご利用ください
#awsblackbelt

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

内容についての注意点

- 本資料では2020年12月16日現在のサービス内容および価格についてご説明しています。最新の情報はAWS公式ウェブサイト(http://aws.amazon.com)にてご確認ください。

- 資料作成には十分注意しておりますが、資料内の価格とAWS公式ウェブサイト記載の価格に相違があった場合、AWS公式ウェブサイトの価格を優先とさせていただきます。

- 価格は税抜表記となっています。日本居住者のお客様には別途消費税をご請求させていただきます。

- AWS does not offer binding price quotes. AWS pricing is publicly available and is subject to change in accordance with the AWS Customer Agreement available at http://aws.amazon.com/agreement/. Any pricing information included in this document is provided only as an estimate of usage charges for AWS services based on certain information that you have provided. Monthly charges will be based on your actual use of AWS services, and may vary from the estimates provided.

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

アジェンダ

Amazon Timestream

- イントロダクション
- アーキテクチャ概念と仕組み Demo
- データの入出力とインテグレーション
- 運用
- 価格と参考リソース
- まとめ

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

Agenda

- イントロダクション
- アーキテクチャ概念と仕組み
- データの入出力とインテグレーション
- 運用
- 価格と参考リソース
- まとめ

イントロダクション

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

AWSのデータベースサービス

時系列データを
専門に扱うデータベース

Relational	Key-value	Document	In-Memory	Graph	Time-series	Ledger	Wide Column
Amazon Aurora Amazon RDS	Amazon DynamoDB	Amazon DocumentDB	Amazon ElastiCache	Amazon Neptune	Amazon Timestream	Amazon QLDB	Amazon Keyspaces (for Apache Cassandra)
従来のアプリケーション、ERP、CRM、e コマース	トラフィックの多いウェブアプリ、e コマースシステム、ゲームアプリケーション	コンテンツ管理、カタログ、ユーザープロファイル	キャッシュ、セッション管理、ゲームのリーダーボード、地理空間アプリケーション	不正検出、ソーシャルネットワーク、レコメンデーションエンジン	IoT アプリケーション、DevOps、産業テレメトリ **New!**	記録システム、サプライチェーン、銀行トランザクション	産業用機器のメンテナンス、取引監視、フリート管理、ルート最適化

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

AWSのデータベースサービス

時系列データを
専門に扱うデータベース

Relational | Key-value | Document | In-Memory | Graph | Time-series | Ledger | Wide Column

Amazon Aurora

Amazon RDS

Amazon DynamoDB

Amazon DocumentDB

Amazon ElastiCache

Amazon Neptune

Amazon Timestream

Amazon QLDB

Amazon Keyspaces (for Apache Cassandra)

2020年10月に一般公開開始
現在時点で利用可能なリージョン
米国東部（バージニア州北部）、米国東部（オハイオ州）、
米国西部（オレゴン州）、EU（アイルランド）

IoT アプリケーション、DevOps、産業テレメトリ

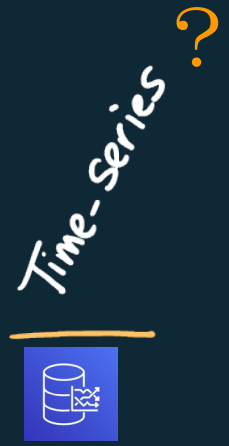
New!

記録システム、サプライチェーン、銀行トランザクション

産業用機器のメンテナンス、取引監視、フリート管理、ルート最適化

aws

AWSのデータベースサービス

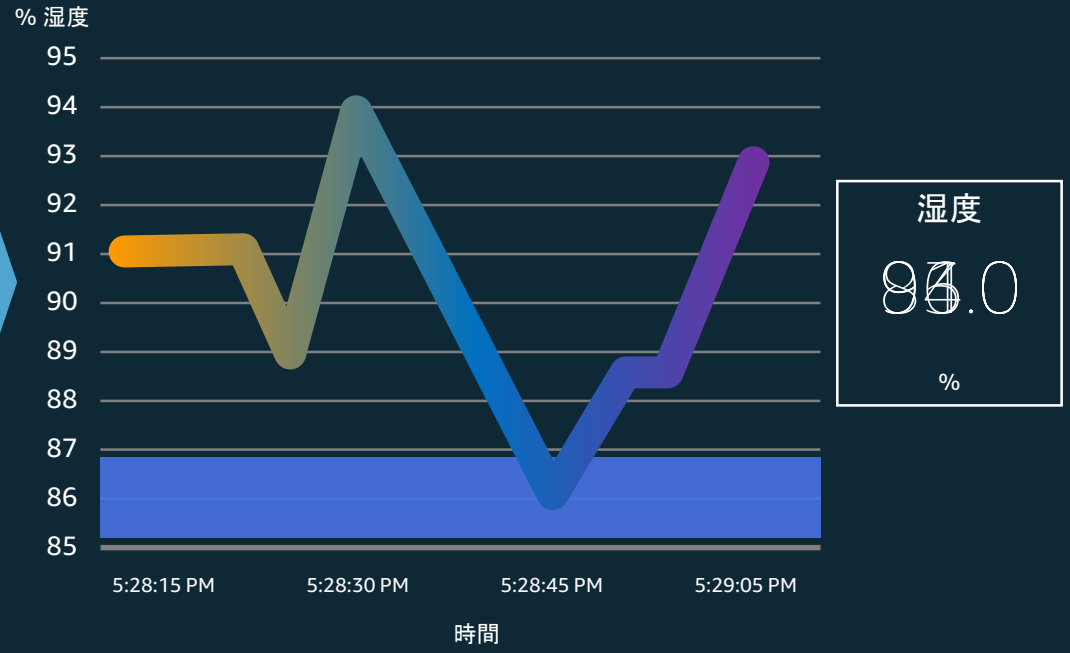


© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

時系列データとは

時間経過とともに変化する事象を測定するために、タイムスタンプとともに記録されたシーケンスデータ

```
timestamp, 測定情報, 属性,          . . .
2020-01-26 21:42:53, 57, Tokyo . . .
2020-01-26 21:42:54, 60, Tokyo . . .
2020-01-26 21:42:55, 59, Tokyo . . .
        ⋮
```



© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

時系列データの活用場所

IoT Application

デバイスのセンサーから動きや温度のデータを収集し、稼働期間の特定やエネルギー利用状況のモニタリング、今後のエネルギー需要の予測のために利用される。

Analytics Application

クリックストリームデータを保存・分析し、カスタマーの閲覧経路、サイト滞在時間や離脱ポイントなどを把握するデータとして利用される。サイト構成やUXの改善に活かす。

DevOps Application

CPU 、メモリ使用率、ネットワークデータ、IOPS などのパフォーマンスメトリックスを収集・分析し、サーバの監視や異常値を検知。システム稼働状況を最適化する。

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

時系列データの活用場所

①現状の傾向を知る
②これから先に起こることを予測する

IoT Application

- センサー
  ネットワーク監視
- 在庫計画
- 交通の混雑状況や
  遅延のモニタリング
- 製造・物流の
  管理

Analytics Application

- クリックストリーム
  分析
- 株市場の予測
- 商品需要予測
- 企業データの分析

DevOps Application

- 物理システムの
  モニタリング
- アプリケーションの
  モニタリング
- ソフトウェアシステムの
  モニタリング

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

時系列データを扱う上での課題

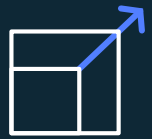
リレーショナルデータベース

時系列データを効率よく扱うための設計が必要

スキーマが固定され柔軟性に欠ける

時系列データ分析用の機能を必要に応じて独自で実装

既存の時系列データソリューション



大規模なデータに対してスケールすることが難しい

データライフサイクル管理

リアルタイムデータとヒストリカルデータの分離

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Amazon Timestream

高速かつスケーラブルな、サーバーレスの時系列専用データベース

サーバーレス

コスト最適化

スケーラブル

時系列専用

常時暗号化と
アクセス制御

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Amazon Timestream 特徴

サーバーレス

コスト最適化

スケーラブル

時系列専用

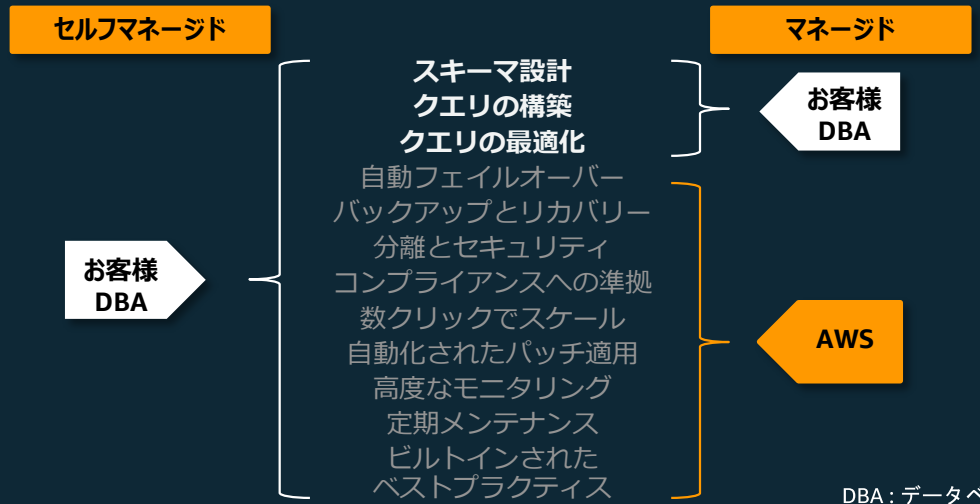
常時暗号化と
アクセス制御

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

サーバーレスによるインフラ運用負荷軽減

サーバーやインスタンスの構築・管理が不要

- 事前にノードをプロビジョニングせずに利用することが可能
- ソフトウェアパッチ適用、アップグレードを自動的に実施



DBA : データベース管理者

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Amazon Timestream 特徴

サーバーレス

コスト最適化

スケーラブル

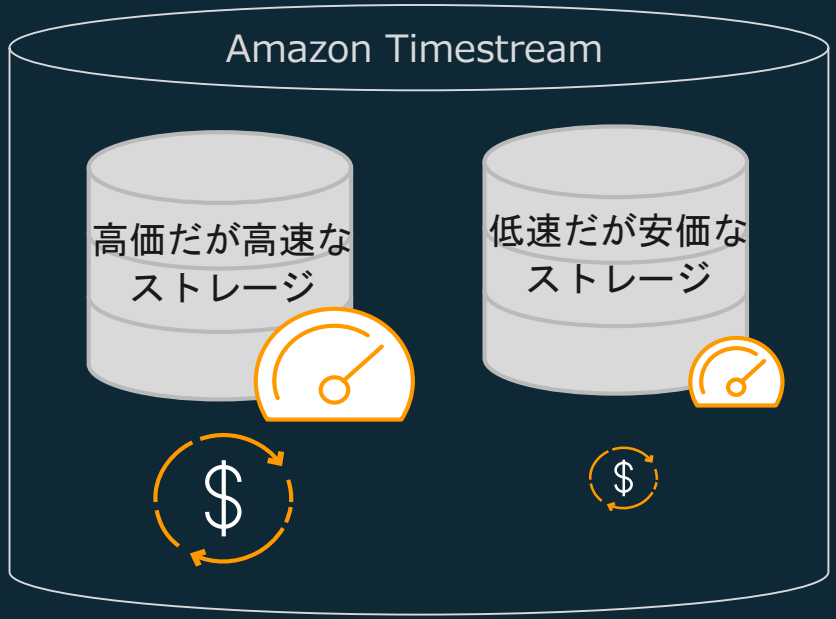
時系列専用

常時暗号化と
アクセス制御

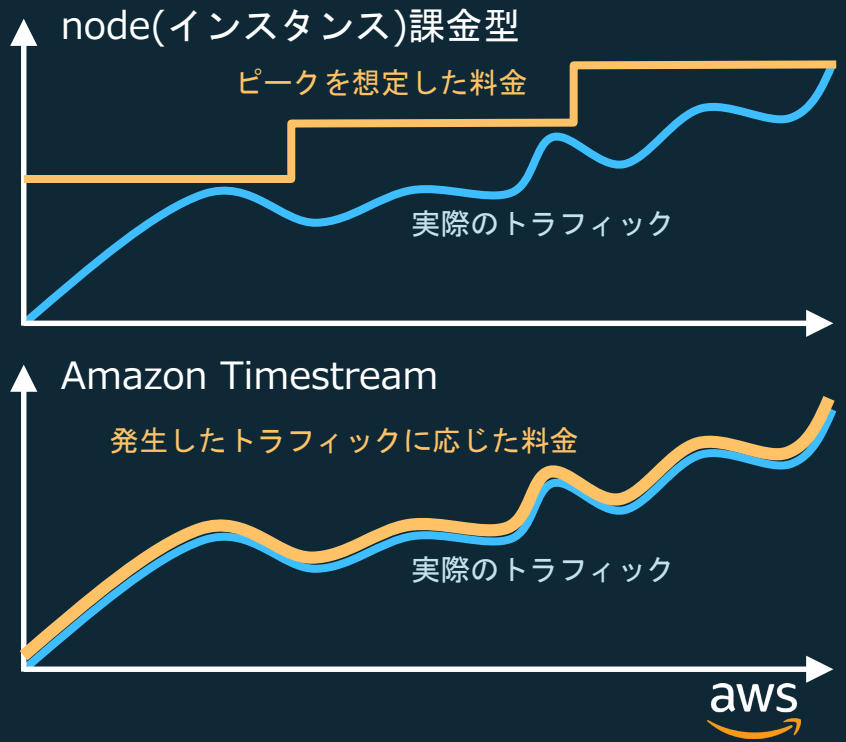
© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

コスト最適化

パフォーマンス要求に応じてコストが最適化された異なるストレージを提供



取り込んだデータ量、保存したデータ量、クエリしたデータ量に応じた利用料金



© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Amazon Timestream 特徴

サーバーレス

コスト最適化

スケーラブル

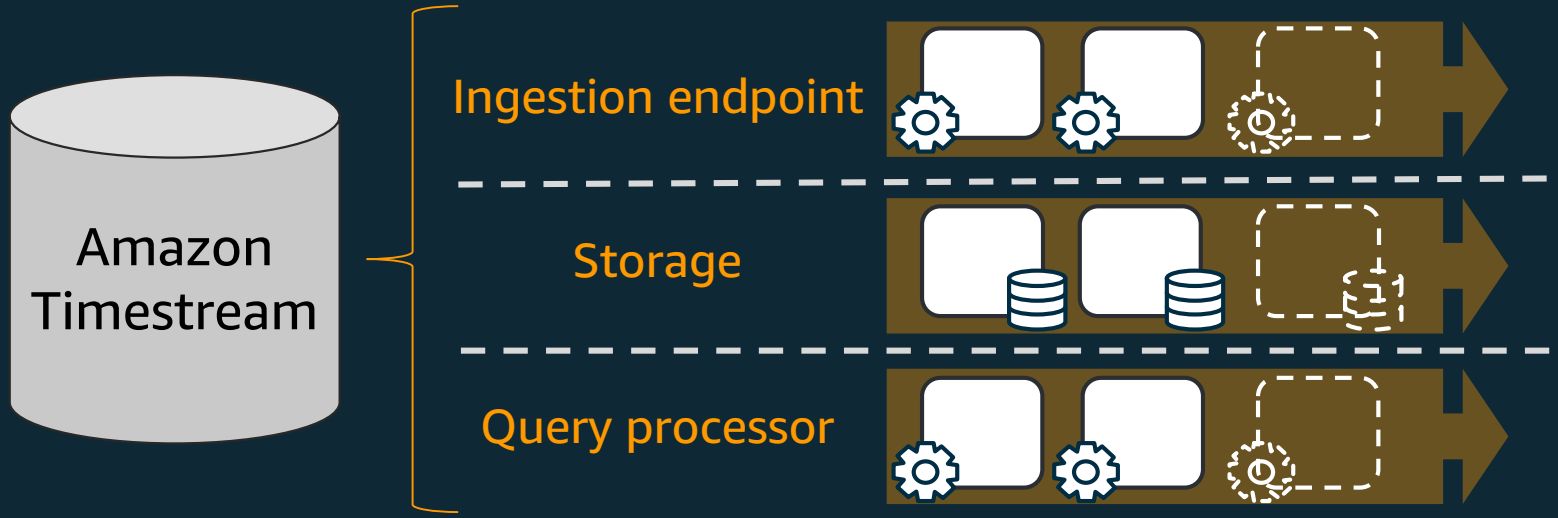
時系列専用

常時暗号化と
アクセス制御

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

スケーラブル

データ挿入・ストレージ・クエリプロセシングが分離されたアーキテクチャ構成。
それぞれ独立して拡張され、1日あたり数兆ものイベントに対応可能。



容量とパフォーマンス状況に合わせ、個別に自動でスケールリング

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Amazon Timestream 特徴

サーバーレス

コスト最適化

スケーラブル

時系列専用

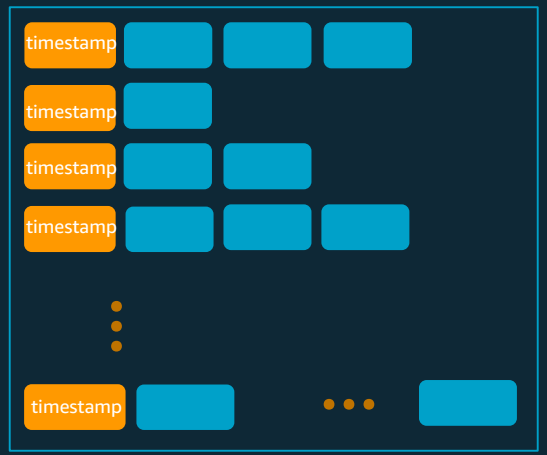
常時暗号化と
アクセス制御

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

時系列専用のデータモデルと関数

時系列データ指向のデータモデルや、平滑化/近似/補間などの時系列関数を提供。
アプリケーション側で複雑な計算ロジックの対応が不要。

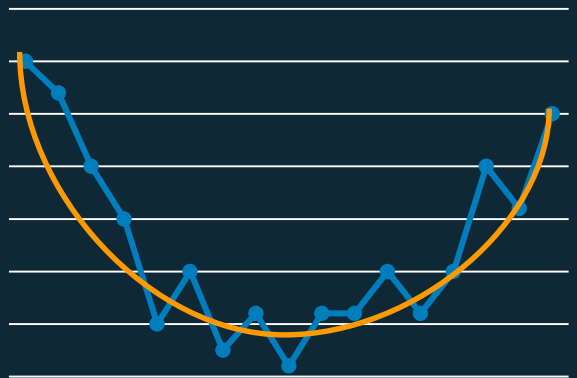
時系列データモデル



タイムスタンプをキーにした高速なデータの取得や、属性情報の付加が柔軟に可能なデータモデル

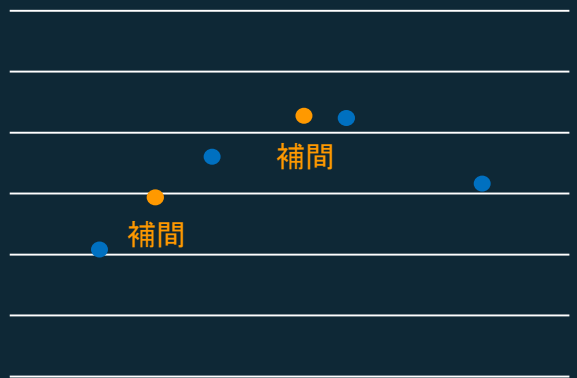
時系列専用関数

平滑化関数



ノイズやその他の微細なポイント、急激な現象を排除しながらデータの重要なパターンを算出し、結果データを返す関数

補間関数



欠落している値を補間したデータを返す関数

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Amazon Timestream 特徴

サーバーレス

コスト最適化

スケーラブル

時系列専用

常時暗号化と
アクセス制御

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

セキュリティ

- データは常時暗号化
  - 通信時：TLSによるクライアント/サーバ間の通信暗号化
  - 保管時：AWS Key Management Service (KMS)を利用して保存データを常時暗号化
- アクセス管理
  - AWS IAMユーザによる認証
    - リクエストは、IAMプリンシパルに関連付けられたアクセスキーIDとシークレットアクセスキーを使用した署名が必要
  - AWS IAMポリシーによるアクションベースの権限制御

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

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

Agenda

- イントロダクション
- アーキテクチャ概念と仕組み
- データの入出力とインテグレーション
- 運用
- 価格と参考リソース
- まとめ

アーキテクチャ概念と仕組み

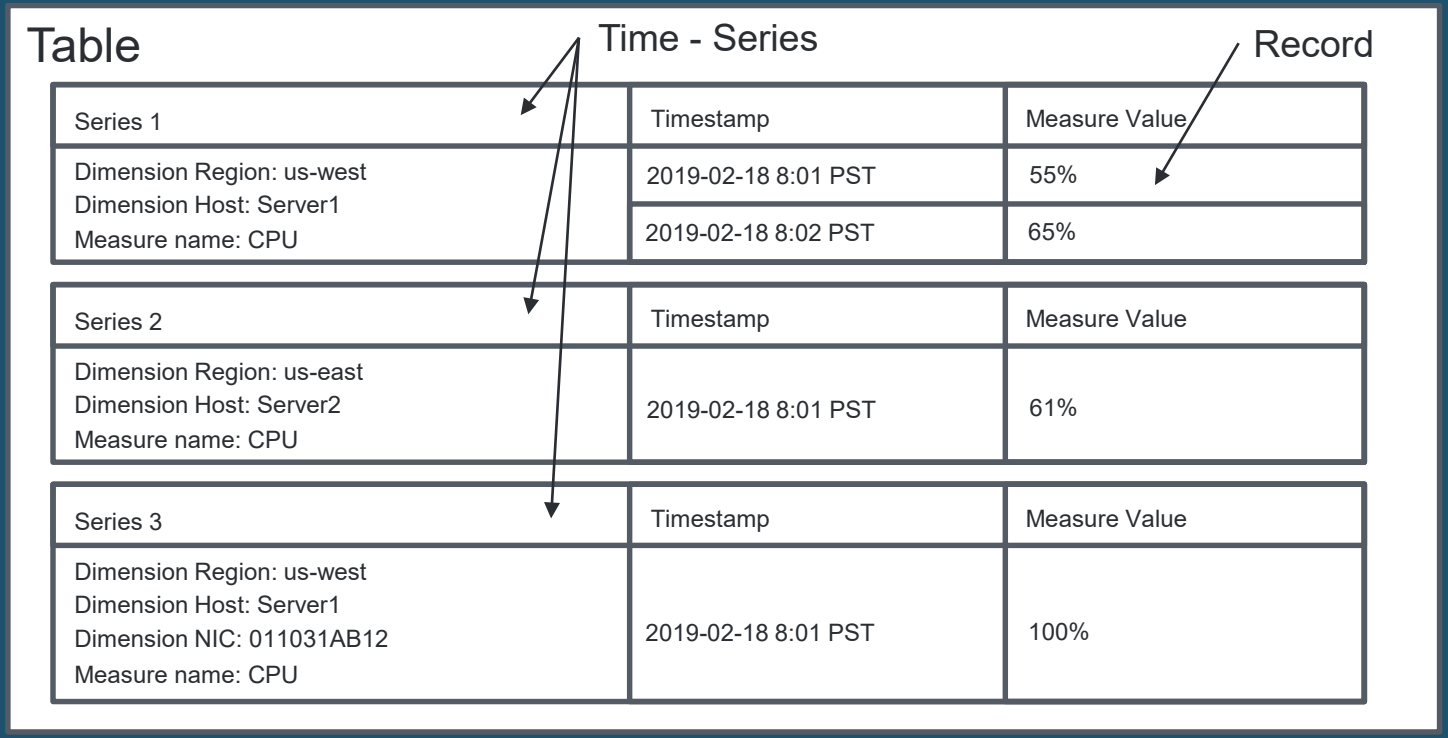
Demo付き

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

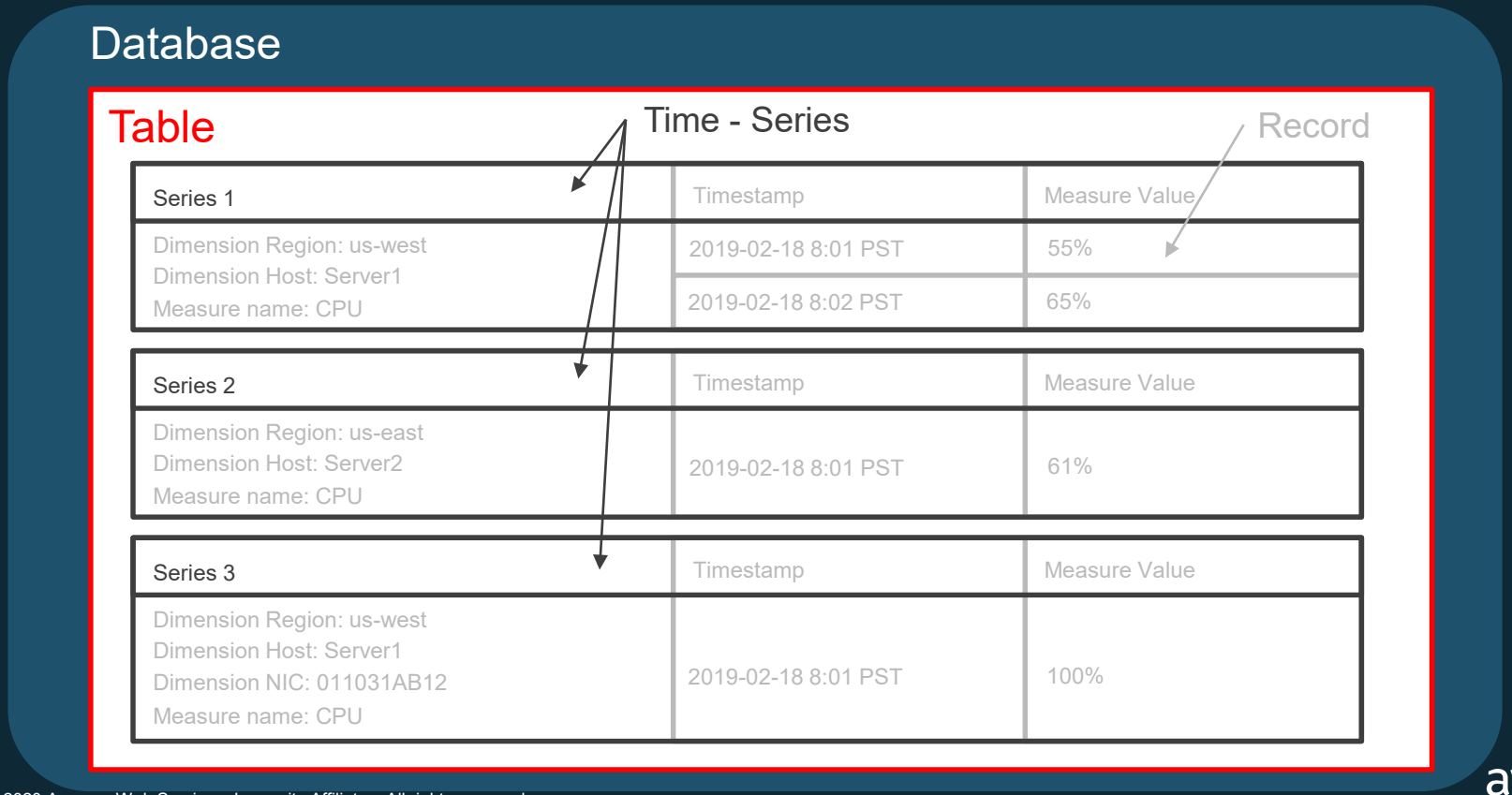
Amazon Timestreamの構成要素

Database



© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Amazon Timestreamの構成要素



© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

コンセプトと用語: テーブル (Tables)

タイムシリーズを保持するコンテナ

テーブル内は暗号化されている

テーブル作成時のスキーマ定義は不要。カラム定義やインデックス設定は自動で実施

ストレージ階層とデータ保持ポリシーに基づいたデータライフサイクル管理を自動で実施 (テーブル単位で保持ポリシーを設定)

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

ストレージ階層の種類

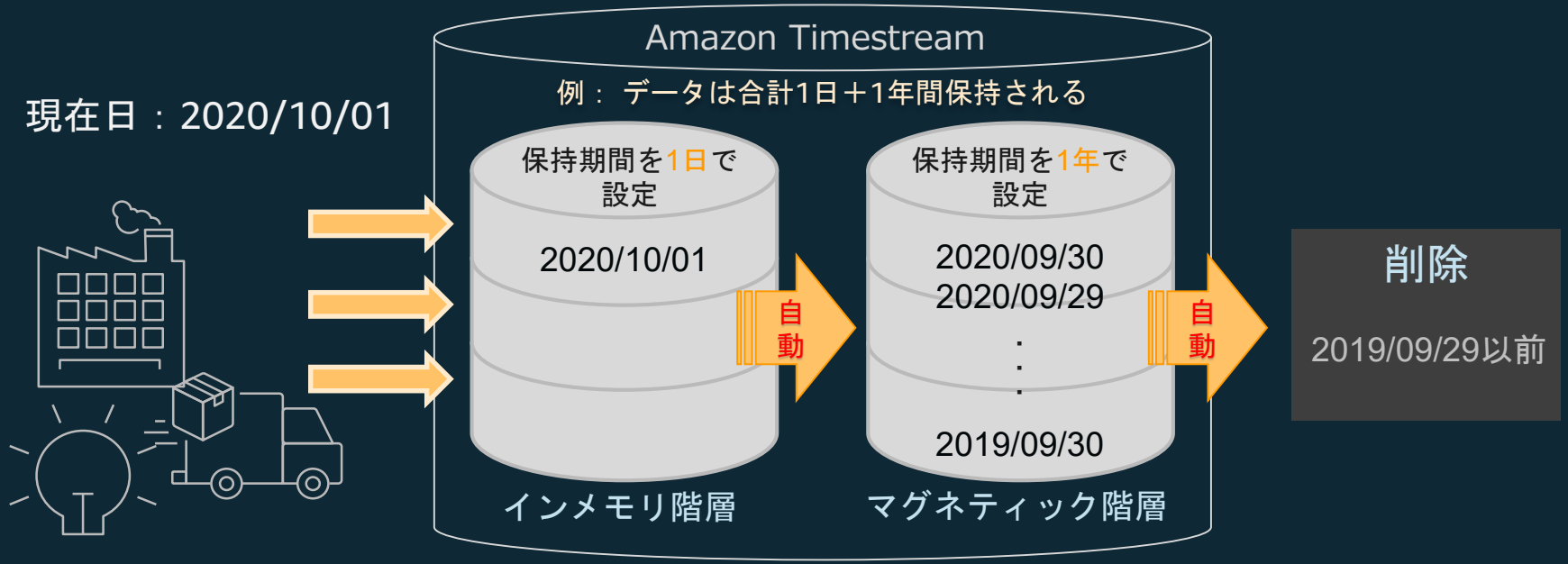
2種類のストレージ階層をサポート

		インメモリ階層	マグネティック階層
特徴	挿入	全てのデータ取り込みを処理	全てのデータはインメモリ階層から連携される
	クエリ	高速ポイントインタイムクエリに合わせて最適化	分析クエリに合わせてパフォーマンス最適化され、長期間データの保存に対してコスト効率が高い
データライフサイクル管理	保持ポリシー ※利用者側で設定	1時間 ～ 最大約１年	１日 ～ 最大200年
	サイクル	保持期間に達したデータをマグネティック階層へ移動する	保持期間に達したデータを削除する

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

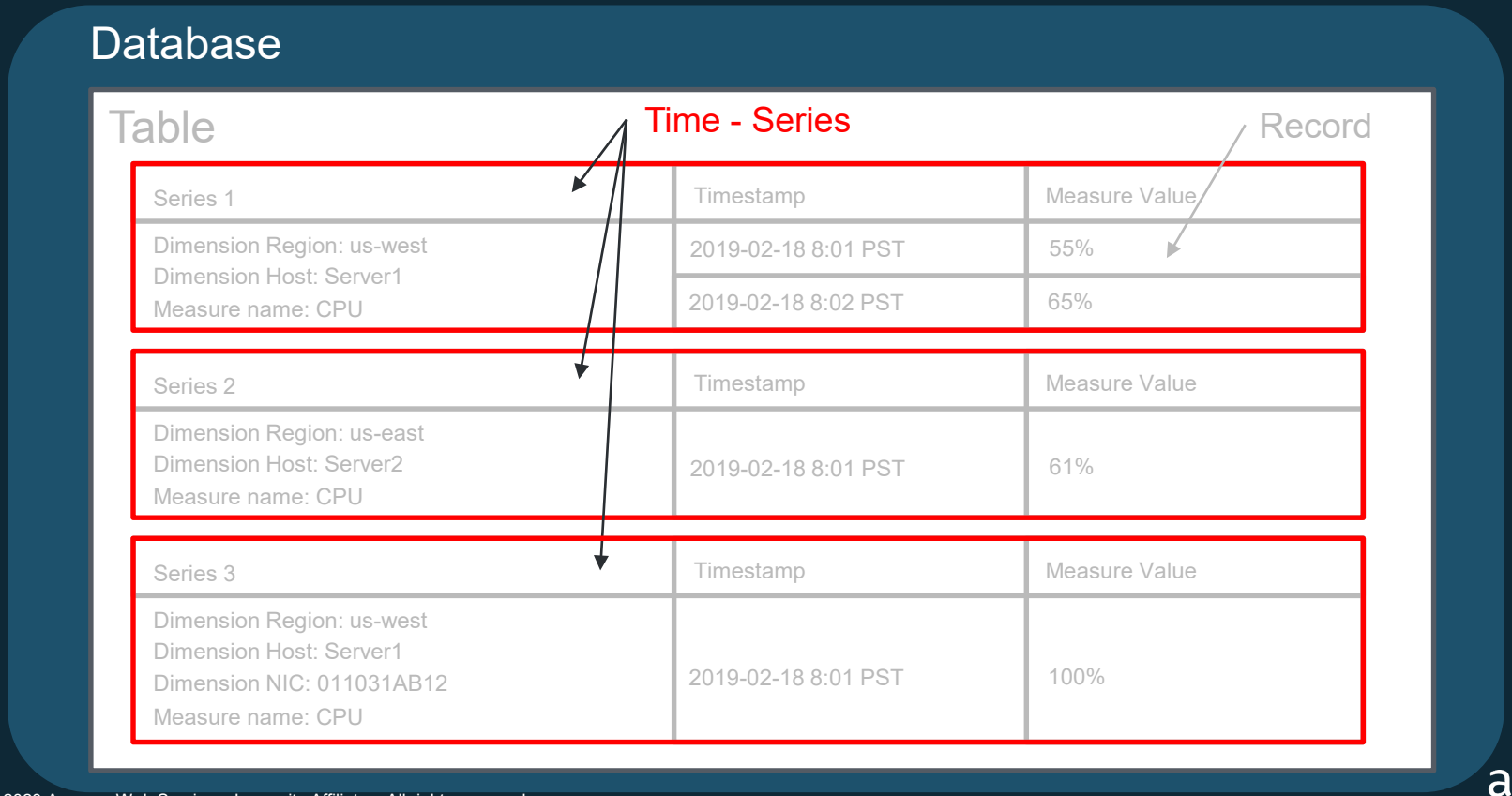
ストレージ階層間のデータライフサイクル

ストレージ階層毎にデータ保持ポリシーを構成することで、メモリ階層からマグネティック階層へのデータ移動、マグネティック階層からのデータ削除を自動で実施。



© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Amazon Timestreamの構成要素



© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

コンセプトと用語: タイムシリーズ(Time-Series)

ある属性値で説明できる、時系列に並んだレコードのまとまり

時系列データ内の欠損値は、Time-Series化したデータを使って組み込みの補間関数で値を埋めることが可能

Time-Seriesは組み込みの時系列関数を用いてクエリすることが可能

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

例: 表形式で表した場合のタイムシリーズ

time	location	store	sensor_id	measure_name	measure_value::double	measure_value::bigint
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	35.0	-
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	38.2	-
2020-06-17 19:00:02.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	45.3	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	51.8	-
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	54.9	-
2020-06-17 19:00:00.000000000	Osaka	-	sensor-31Hij	mem_utilization	42.6	-
2020-06-17 19:00:01.000000000	Osaka	-	sensor-31Hij	mem_utilization	33.3	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	30000
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	15200
2020-06-17 19:00:02.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	15200

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

例: 表形式で表した場合のタイムシリーズ

time	location	store	sensor_id	measure_name	measure_value::double	measure_value::bigint
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	35.0	-
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	38.2	-
2020-06-17 19:00:02.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	45.3	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	51.8	-
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	54.9	-
2020-06-17 19:00:00.000000000	Osaka	-	sensor-31Hij	mem_utilization	42.6	-
2020-06-17 19:00:01.000000000	Osaka	-	sensor-31Hij	mem_utilization	33.3	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	30000
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	15200
2020-06-17 19:00:02.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	15200

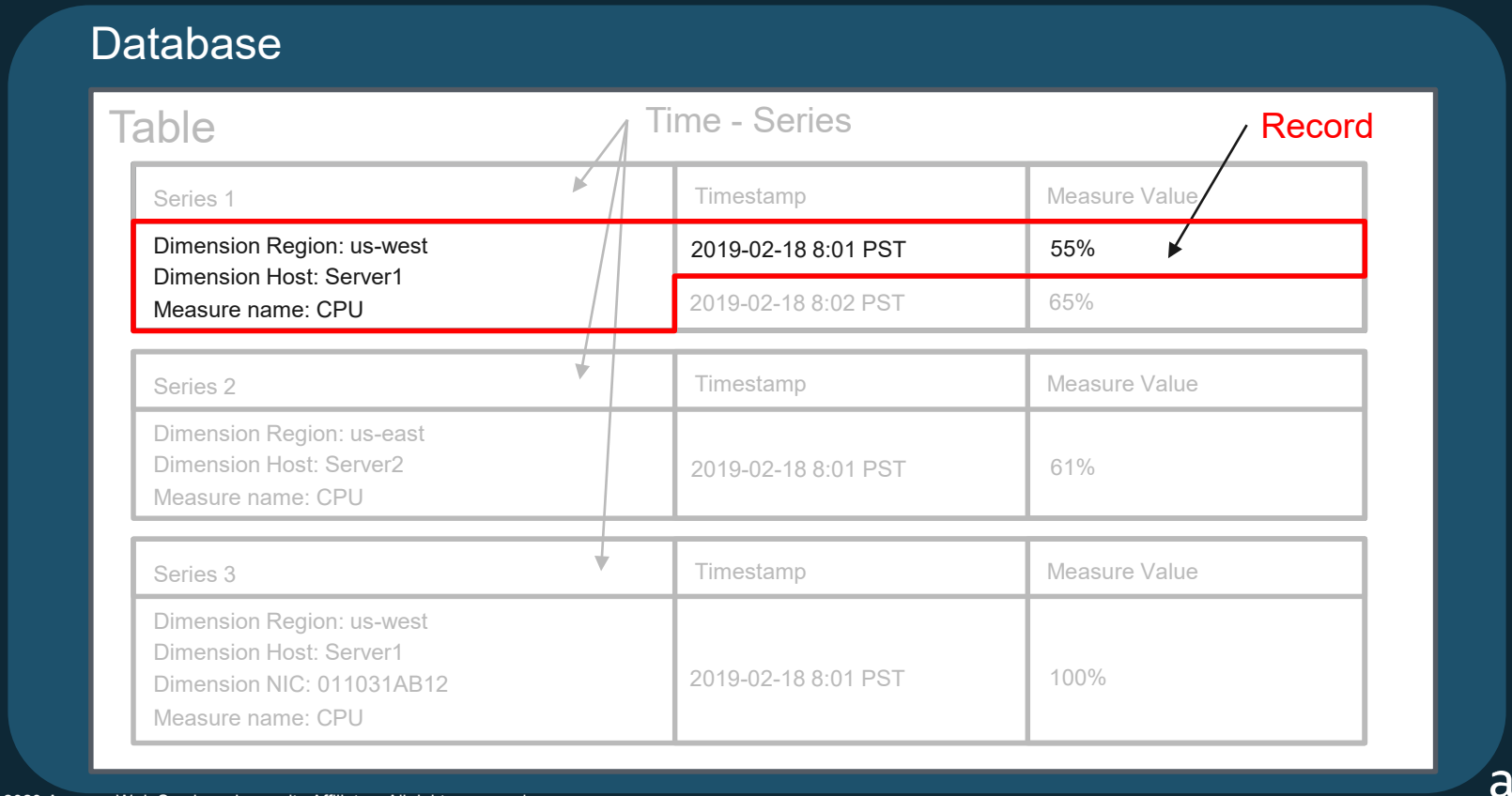
© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

例: 表形式で表した場合のタイムシリーズ

time	location	store	sensor_id	measure_name	measure_value::double	measure_value::bigint
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	35.0	-
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	38.2	-
2020-06-17 19:00:02.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	45.3	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	51.8	-
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	54.9	-
2020-06-17 19:00:00.000000000	Osaka	-	sensor-31Hij	mem_utilization	42.6	-
2020-06-17 19:00:01.000000000	Osaka	-	sensor-31Hij	mem_utilization	33.3	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	30000
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	15200
2020-06-17 19:00:02.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	15200

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Amazon Timestreamの構成要素



© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

コンセプトと用語: レコード (Record)

単一の時系列データポイント

各レコードは、
タイムスタンプ、1つ以上のディメンション、および時間の経過とともに変化するメジャーで構成される

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

例: 表形式で表した場合のレコード

	ディメンション			メジャー		
time	location	store	sensor_id	measure_name	measure_value::double	measure_value::bigint
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	35.0	-
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	38.2	-
2020-06-17 19:00:02.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	45.3	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	51.8	-
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	54.9	-
2020-06-17 19:00:00.000000000	Osaka	-	sensor-31Hij	mem_utilization	42.6	-
2020-06-17 19:00:01.000000000	Osaka	-	sensor-31Hij	mem_utilization	33.3	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	30000
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	15200
2020-06-17 19:00:02.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	15200

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

例: 表形式で表した場合のレコード

	ディメンション			メジャー		
time	location	store	sensor_id	measure_name	measure_value::double	measure_value::bigint
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	35.0	-
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	38.2	-
2020-06-17 19:00:02.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	45.3	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	51.8	-
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	54.9	-
2020-06-17 19:00:00.000000000	Osaka	-	sensor-31Hij	mem_utilization	42.6	-
2020-06-17 19:00:01.000000000	Osaka	-	sensor-31Hij	mem_utilization	33.3	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	30000
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	15200
2020-06-17 19:00:02.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	15200

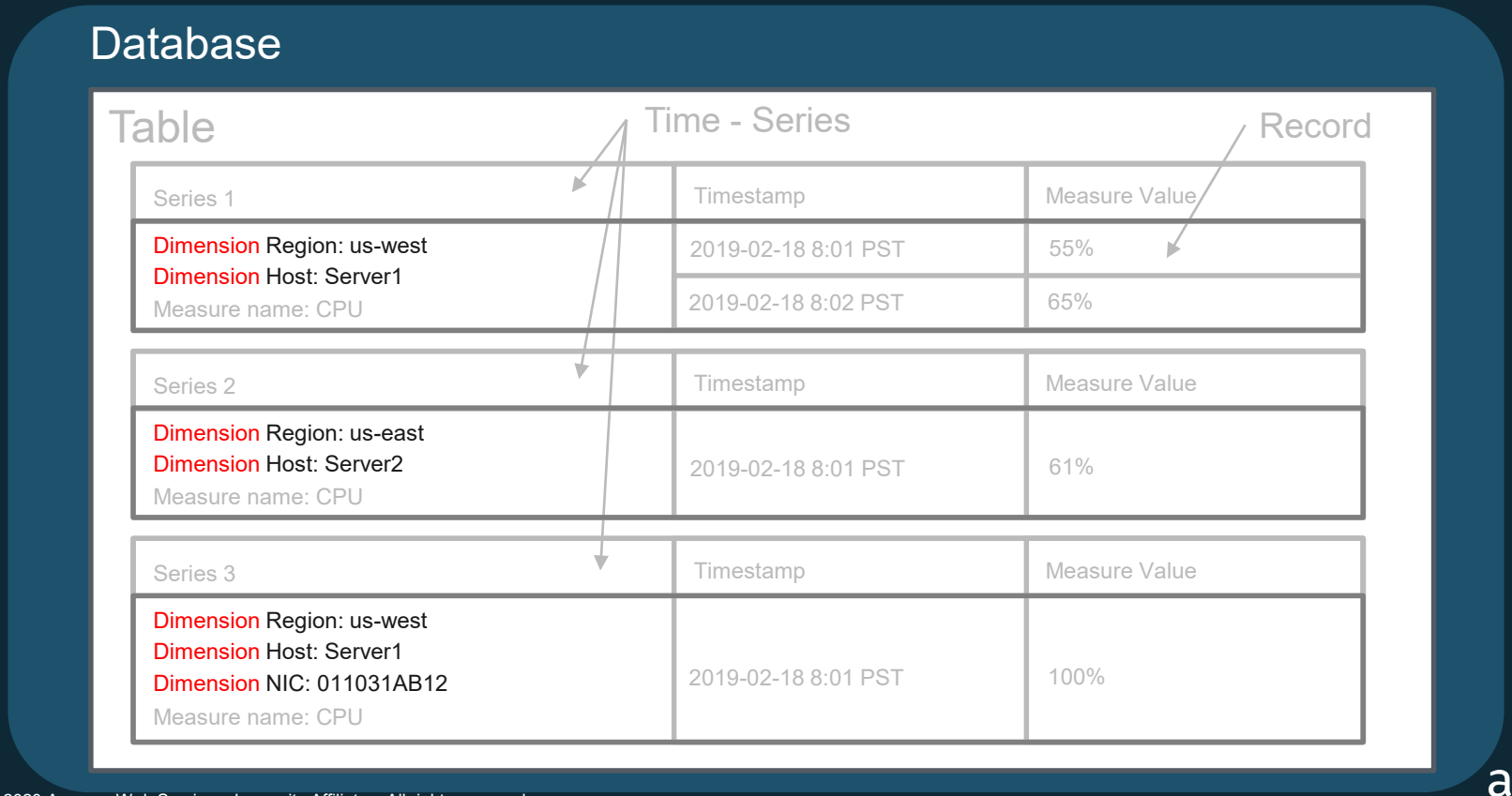
© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

例: 表形式で表した場合のレコード

	ディメンション			メジャー		
time	location	store	sensor_id	measure_name	measure_value::double	measure_value::bigint
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	35.0	-
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	38.2	-
2020-06-17 19:00:02.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	45.3	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	51.8	-
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	54.9	-
2020-06-17 19:00:00.000000000	Osaka	-	sensor-31Hij	mem_utilization	42.6	-
2020-06-17 19:00:01.000000000	Osaka	-	sensor-31Hij	mem_utilization	33.3	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	30000
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	15200
2020-06-17 19:00:02.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	15200

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Amazon Timestreamの構成要素



© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

コンセプトと用語: ディメンション (Dimensions)

測定値を識別するための属性情報セット

各テーブルで128個のディメンションが利用可能

全てのディメンションは文字列(varchars)として扱われる

ディメンションはテーブルにデータが取り込まれたとき、動的に追加される

一度定義されたディメンションは変更不可

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

例: 表形式で表した場合のディメンション

time	location	store	sensor_id	measure_name	measure_value::double	measure_value::bigint
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	35.0	-
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	38.2	-
2020-06-17 19:00:02.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	45.3	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	51.8	-
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	54.9	-
2020-06-17 19:00:00.000000000	Osaka	-	sensor-31Hij	mem_utilization	42.6	-
2020-06-17 19:00:01.000000000	Osaka	-	sensor-31Hij	mem_utilization	33.3	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	30000
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	15200
2020-06-17 19:00:02.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	15200

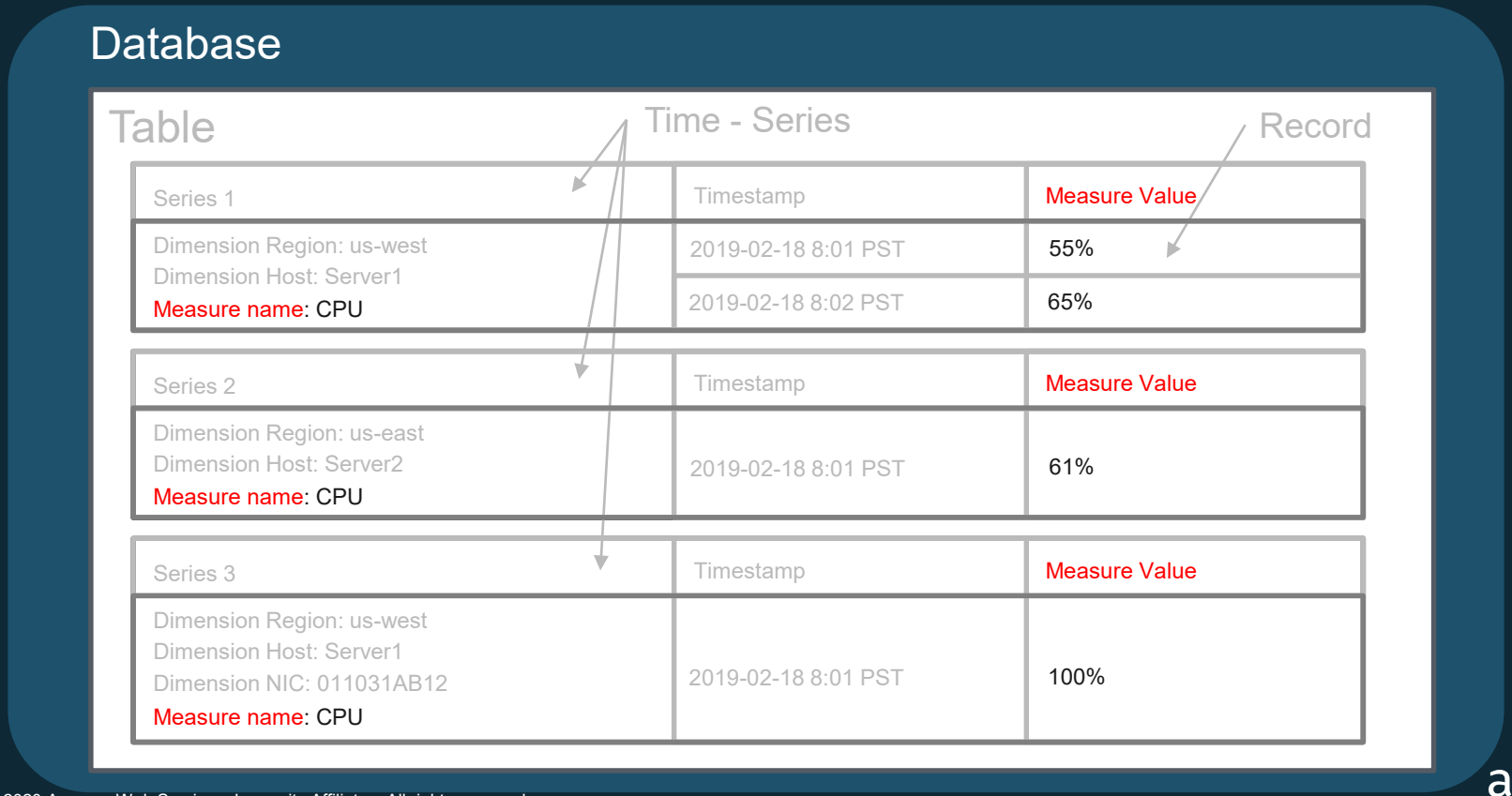
© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

例: 表形式で表した場合のディメンション

time	location	store	sensor_id	measure_name	measure_value::double	measure_value::bigint
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	35.0	-
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	38.2	-
2020-06-17 19:00:02.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	45.3	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	51.8	-
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	54.9	-
2020-06-17 19:00:00.000000000	Osaka	-	sensor-31Hij	mem_utilization	42.6	-
2020-06-17 19:00:01.000000000	Osaka	-	sensor-31Hij	mem_utilization	33.3	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	30000
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	15200
2020-06-17 19:00:02.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	15200

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Amazon Timestreamの構成要素



© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

コンセプトと用語: メジャー (Measures)

測定値。各レコードは名前(measure_name)と値(measure_value)から構成される単一の測定値を含む

※1レコードに１セット。メジャーを持たないレコードは存在しない

各テーブルは、1024のメジャーの種類を定義できる

サポートされるデータ型: boolean, bigint, double, varchar

メジャーはテーブルにデータが取り込まれたとき、動的に追加される(ディメンションと同様)

一度定義されたメジャーは変更不可。同一のメジャー名で異なる型の値を挿入することは出来ない

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

例: 表形式で表した場合のメジャー

time	location	store	sensor_id	measure_name	measure_value::double	measure_value::bigint
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	35.0	-
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	38.2	-
2020-06-17 19:00:02.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	45.3	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	51.8	-
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	54.9	-
2020-06-17 19:00:00.000000000	Osaka	-	sensor-31Hij	mem_utilization	42.6	-
2020-06-17 19:00:01.000000000	Osaka	-	sensor-31Hij	mem_utilization	33.3	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	30000
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	15200
2020-06-17 19:00:02.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	15200

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

※表形式で表した場合

例: 表形式で表した場合のメジャー

time	location	store	sensor_id	measure_name	measure_value::double	measure_value::bigint
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	35.0	-
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	38.2	-
2020-06-17 19:00:02.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	45.3	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	51.8	-
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	54.9	-
2020-06-17 19:00:00.000000000	Osaka	-	sensor-31Hij	mem_utilization	42.6	-
2020-06-17 19:00:01.000000000	Osaka	-	sensor-31Hij	mem_utilization	33.3	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	30000
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	15200
2020-06-17 19:00:02.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	15200

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

※表形式で表した場合

例: 表形式で表した場合のメジャー

time	location	store	sensor_id	measure_name	measure_value::double	measure_value::bigint
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	35.0	
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	38.2	
2020-06-17 19:00:02.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	45.3	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	51.8	-
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	54.9	-
2020-06-17 19:00:00.000000000	Osaka	-	sensor-31Hij	mem_utilization	42.6	-
2020-06-17 19:00:01.000000000	Osaka	-	sensor-31Hij	mem_utilization	33.3	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	30000
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	15200
2020-06-17 19:00:02.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	15200

メジャーの型毎に
バリューの列が追加される

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

※表形式で表した場合

例: 表形式で表した場合のメジャー

time	location	store	sensor_id	measure_name	measure_value::double	measure_value::bigint
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	35.0	
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	38.2	
2020-06-17 19:00:02.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	45.3	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	51.8	-
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	54.9	-
2020-06-17 19:00:00.000000000	Osaka	-	sensor-31Hij	mem_utilization	42.6	-
2020-06-17 19:00:01.000000000	Osaka	-	sensor-31Hij	mem_utilization	33.3	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	30000
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	15200
2020-06-17 19:00:02.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	15200

メジャーの型毎に
バリューの列が追加される

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

※表形式で表した場合

例: 表形式で表した場合のメジャー

time	location	store	sensor_id	measure_name	measure_value::double	measure_value::bigint
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	35.0	
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	38.2	
2020-06-17 19:00:02.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	45.3	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	51.8	-
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	54.9	-
2020-06-17 19:00:00.000000000	Osaka	-	sensor-31Hij	mem_utilization	42.6	-
2020-06-17 19:00:01.000000000	Osaka	-	sensor-31Hij	mem_utilization	33.3	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	30000
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	15200
2020-06-17 19:00:02.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	15200

メジャーの型毎に
バリューの列が追加される

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

※表形式で表した場合

	ディメンション			メジャー		
time	location	store	sensor_id	measure_name	measure_value::double	measure_value::bigint
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	35.0	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	51.8	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	30000
2020-06-17 19:00:00.000000000	Osaka	-	sensor-31Hij	cpu_utilization	51.8	-

time	location	store	sensor_id	measure_name	measure_value::double
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	37.0

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

	ディメンション			メジャー		
time	location	store	sensor_id	measure_name	measure_value::double	measure_value::bigint
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	35.0	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	51.8	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	30000
2020-06-17 19:00:00.000000000	Osaka	-	sensor-31Hij	cpu_utilization	51.8	-

time	location	store	sensor_id	measure_name	measure_value::double
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	37.0

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

	ディメンション			メジャー		
time	location	store	sensor_id	measure_name	measure_value::double	measure_value::bigint
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	35.0	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	51.8	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	30000
2020-06-17 19:00:00.000000000	Osaka	-	sensor-31Hij	cpu_utilization	51.8	-

time	location	store	sensor_id	measure_name	measure_value::double
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	37.0

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

レコードの重複判定

タイムスタンプ、ディメンション、メジャー名でユニーク判定。異なるメジャー値を挿入した場合はインジェストエラーとなる（"先勝ち"）。

	ディメンション			メジャー		
time	location	store	sensor_id	measure_name	measure_value::double	measure_value::bigint
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	35.0	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	51.8	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	30000
2020-06-17 19:00:00.000000000	Osaka	-	sensor-31Hij	cpu_utilization	51.8	-

ユニーク判定のキー

time	location	store	sensor_id	measure_name	measure_value::double
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	37.0

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

レコードの更新

2020/11/25 ~

Amazon Timestreamは各レコードのVersion番号を追跡。挿入リクエスト時に「Version」パラメータを含めることでメジャー値の更新が可能。

	ディメンション			メジャー		
time	**location**	**store**	**sensor_id**	**measure_name**	**measure_value::double**	**measure_value::bigint**
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	35.0	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	51.8	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	30000
2020-06-17 19:00:00.000000000	Osaka					-

ユニーク判定のキー

Version番号がより大きな書き込み要求を受信した際に値を更新。
“後勝ち” の実現には Version=現在時刻 が推奨。

Version= 1606433187

time	**location**	**store**	**sensor_id**	**measure_name**	**measure_value::double**
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	37.0

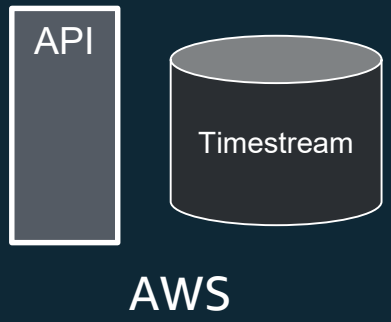
© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

Demo

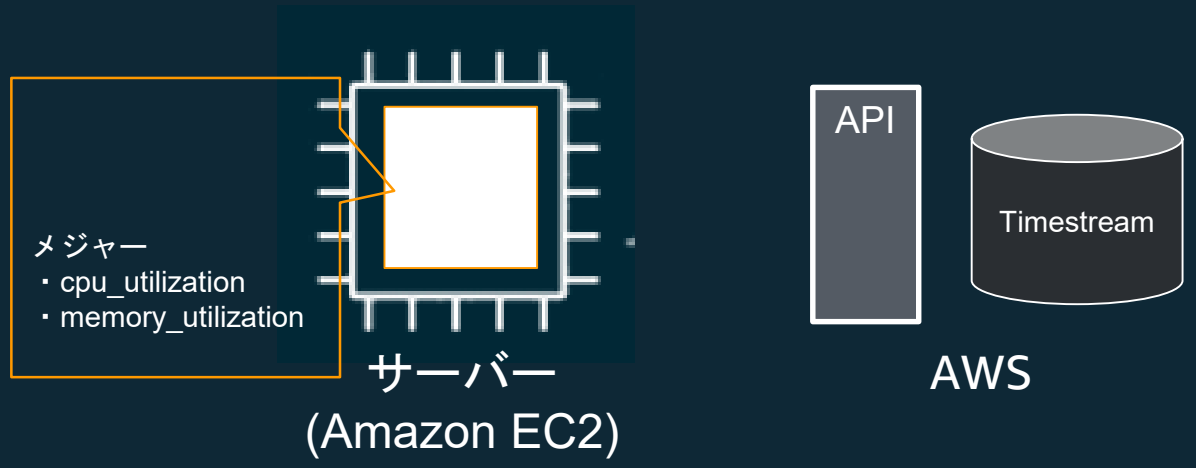
© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Demo



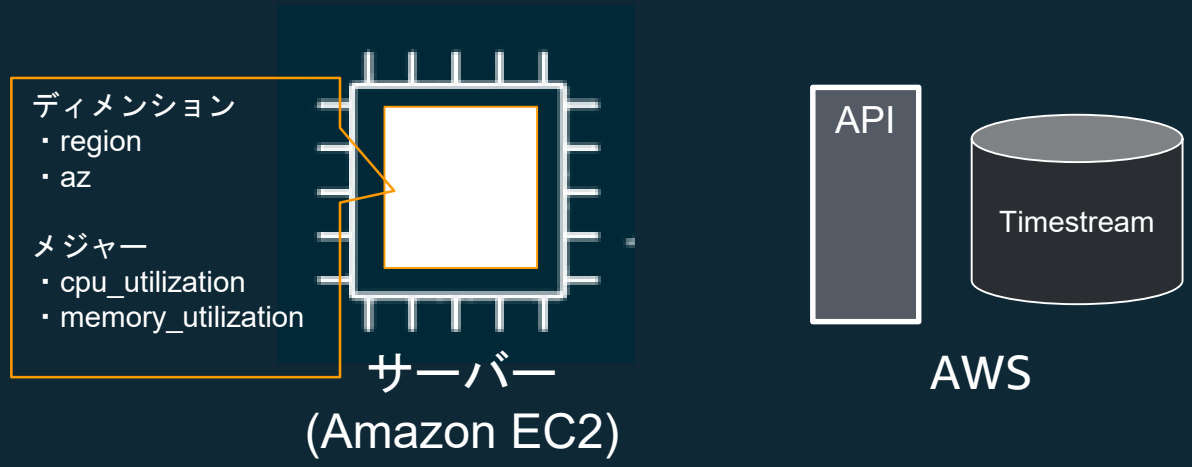
© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Demo



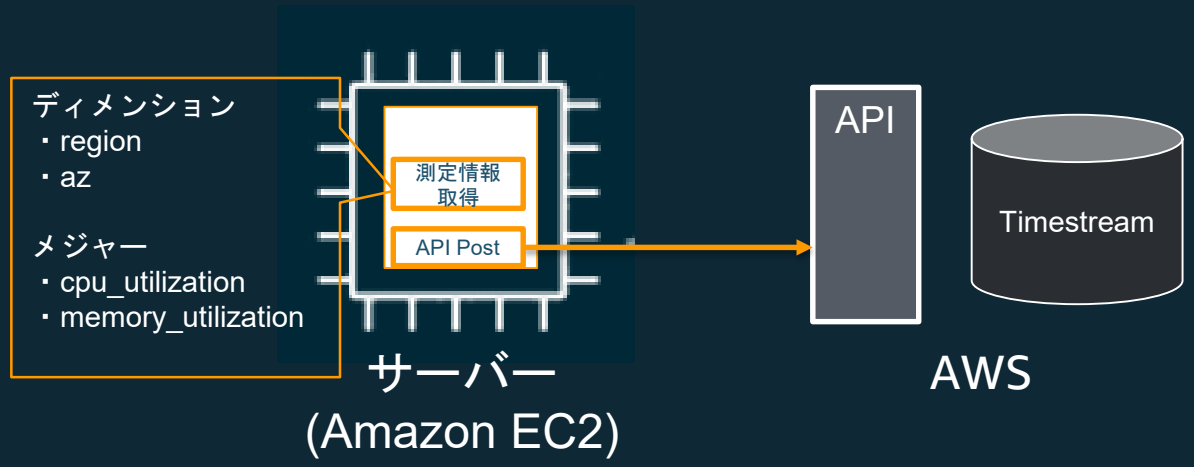
© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Demo



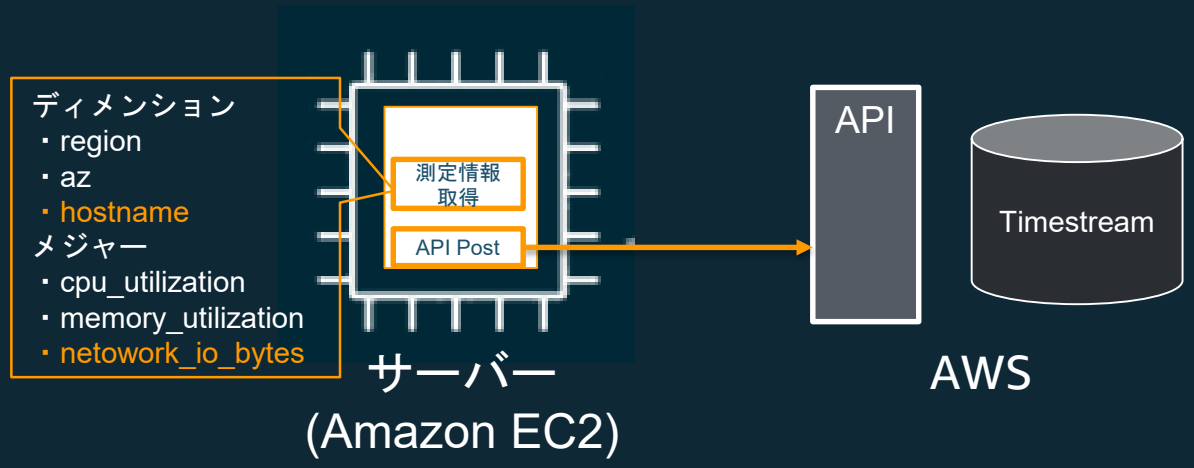
© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Demo



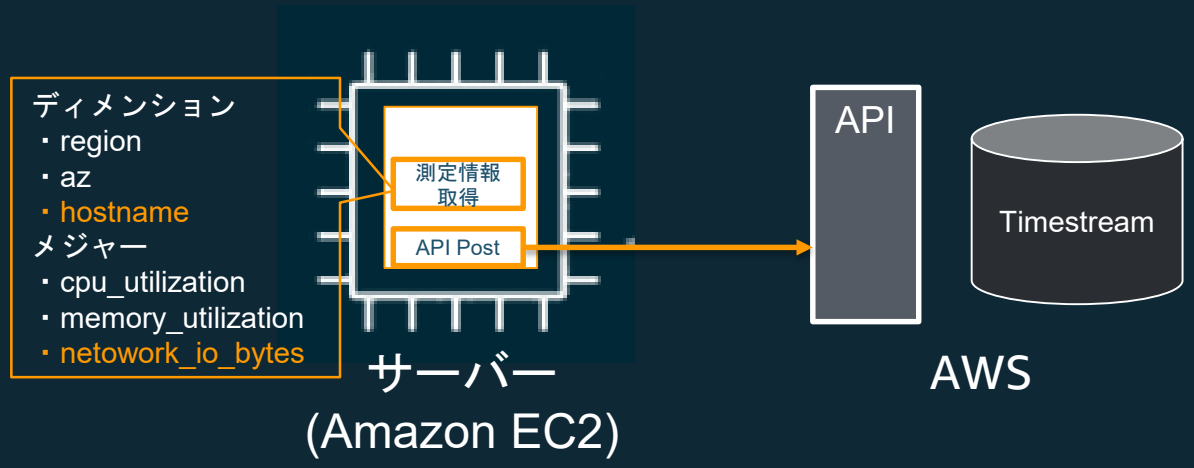
© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Demo



© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Demo



© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

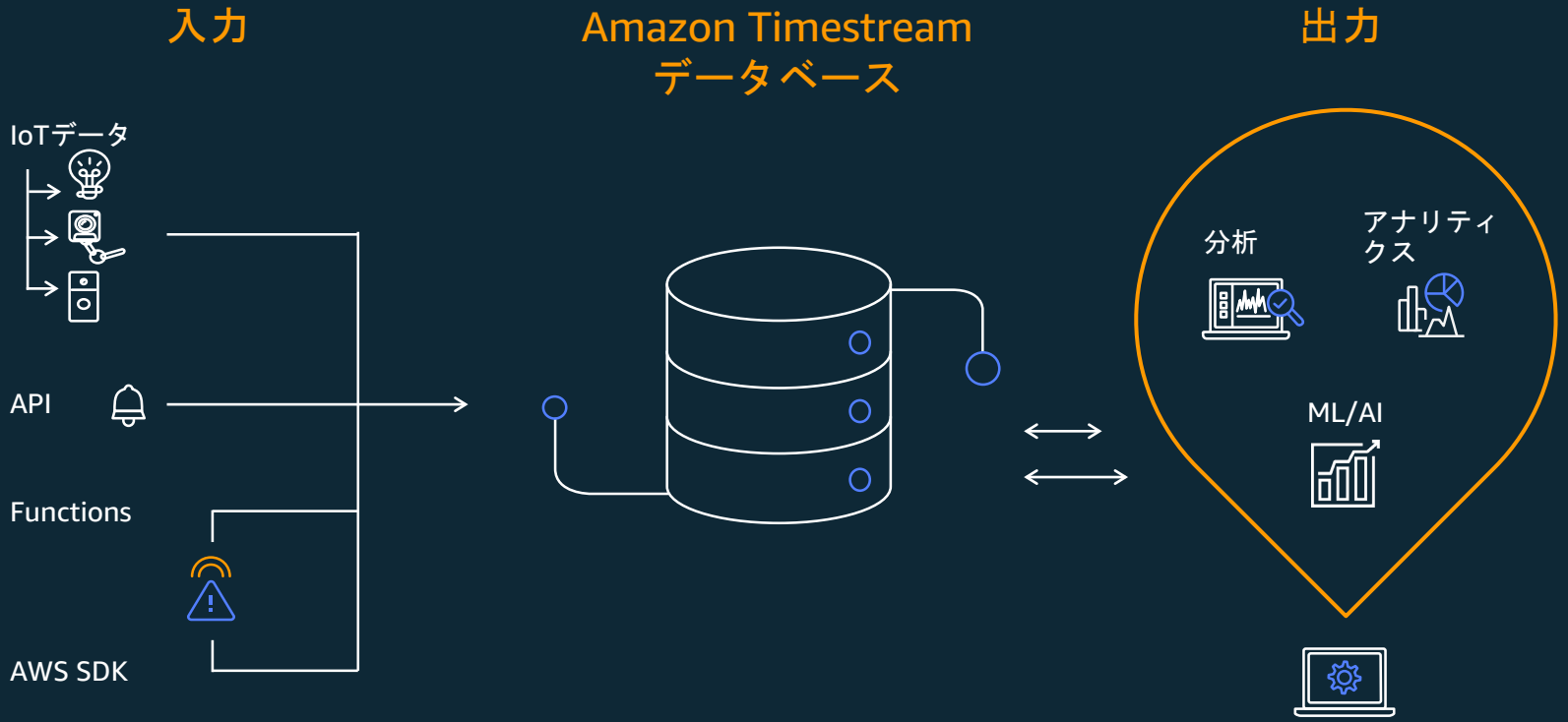
Agenda

- イントロダクション
- アーキテクチャ概念と仕組み
- データの入出力とインテグレーション
- 運用
- 価格と参考リソース
- まとめ

データの入出力とインテグレーション

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

データベースへのアクセス方法



© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

入力：データ挿入

標準

- データの挿入には AWS SDK / Timestream API を利用
    - SDK: Java, Python, Golang, Node.js, .NET, etc.
    - AWS CLI

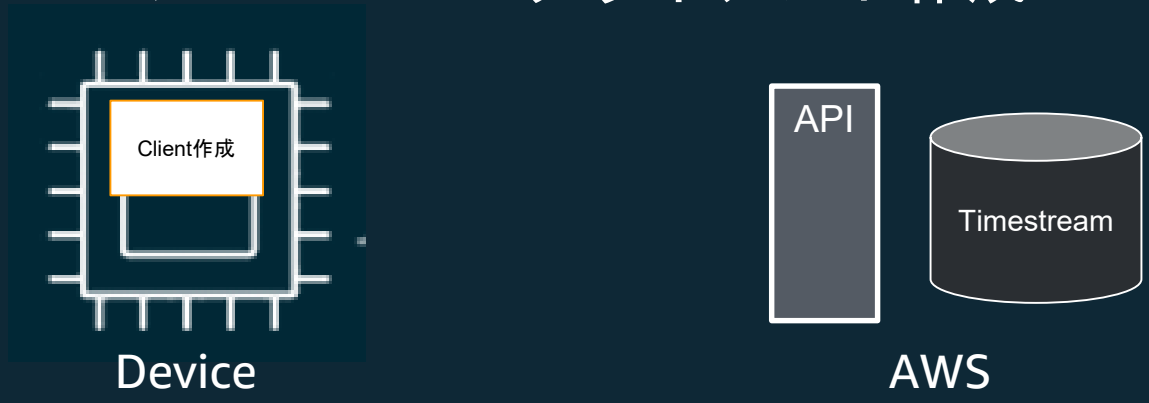
インテグレーション

- AWS IoT Core
- Amazon Kinesis Data Analytics for Apache Flink connector (AWSLabs GitHub Integration)
- Telegraf connector (AWSLabs GitHub plugin)

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

例: Python SDKを使ったデータ挿入
書き込みSDKのTimestreamクライアント作成

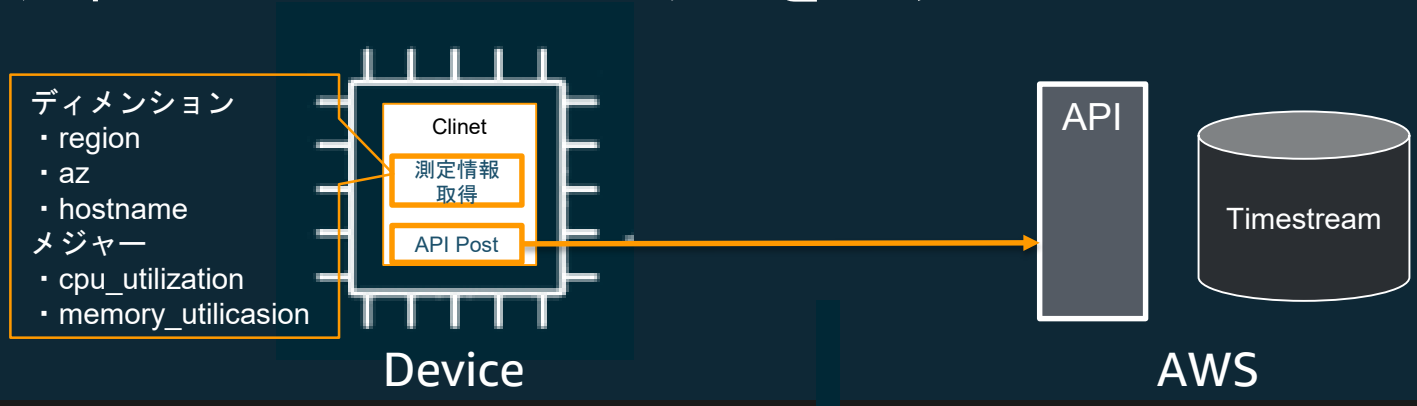



```
import boto3
from botocore.config import Config

session = boto3.Session()

write_client = session.client('timestream-write', region_name=region,
                              config=Config(read_timeout=20,                # リクエストタイムアウト(秒)
                                            max_pool_connections=5000,      # 最大接続数
                                            retries={'max_attempts': 10})) # 最大試行回数
```

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

※表現上、一部簡略化

例: Python SDKを使ったデータ挿入
ディメンション・メジャーをセット ～ Write API Call

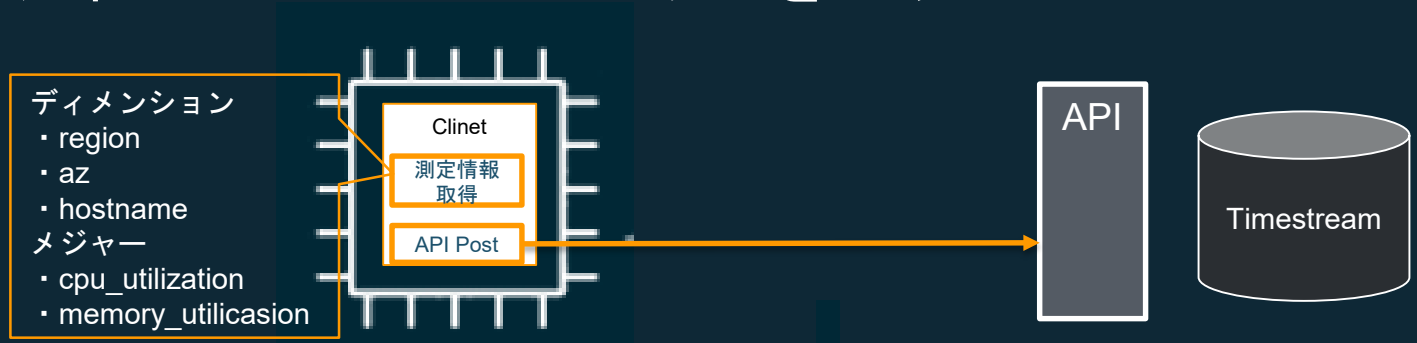



```
current_time = self._current_milli_time()

dimensions = {
    {'Name': 'region', 'Value': 'us-east-1'},
    {'Name': 'az', 'Value': 'az1'},
    {'Name': 'hostname', 'Value': 'host1'}
]

cpu_utilization = {
    'Dimensions': dimensions,
    'MeasureName': 'cpu_utilization',
    'MeasureValue': str(instance_cpu_util_value),
    'MeasureValueType': 'DOUBLE',
    'Time': current_time
}
```

```
memory_utilization = {
    'Dimensions': dimensions,
    'MeasureName': 'memory_utilization',
    'MeasureValue': str(instance_memory_util_value),
    'MeasureValueType': 'DOUBLE',
    'Time': current_time
}

records = [cpu_utilization, memory_utilization]

result =
self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
                          TableName=Constant.TABLE_NAME,
                          Records=records, CommonAttributes={})
```

※表現上、一部簡略化

例: Python SDKを使ったデータ挿入
ディメンション・メジャーをセット ～ Write API Call

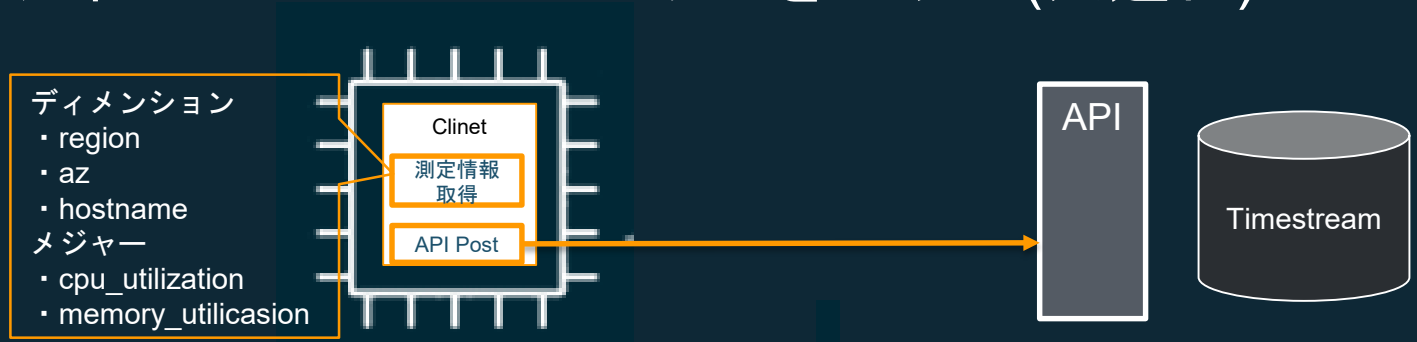



```
current_time = self._current_milli_time()

dimensions = {
    {'Name': 'region', 'Value': 'us-east-1'},
    {'Name': 'az', 'Value': 'az1'},
    {'Name': 'hostname', 'Value': 'host1'}
]

cpu_utilization = {
    'Dimensions': dimensions,
    'MeasureName': 'cpu_utilization',
    'MeasureValue': str(instance_cpu_util_value),
    'MeasureValueType': 'DOUBLE',
    'Time': current_time
}
```

```
memory_utilization = {
    'Dimensions': dimensions,
    'MeasureName': 'memory_utilization',
    'MeasureValue': str(instance_memory_util_value),
    'MeasureValueType': 'DOUBLE',
    'Time': current_time
}

records = [cpu_utilization, memory_utilization]

result =
self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
                          TableName=Constant.TABLE_NAME,
                          Records=records, CommonAttributes={})
```

※表現上、一部簡略化

例: Python SDKを使ったデータ挿入
ディメンション・メジャーをセット(共通化) ～ Write API Call



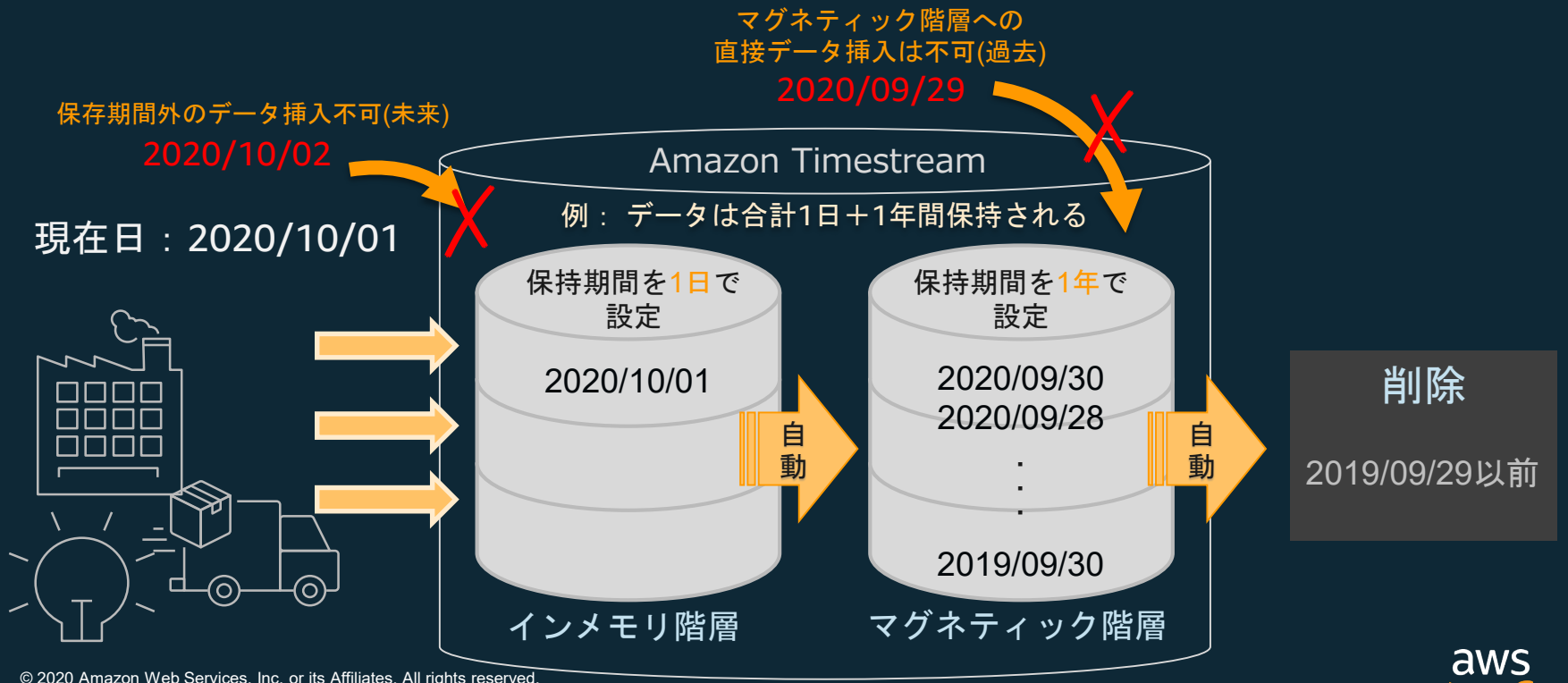

```
current_time = self._current_milli_time()

dimensions = [
    {'Name': 'region', 'Value': 'us-east-1'},
    {'Name': 'az', 'Value': 'az1'},
    {'Name': 'hostname', 'Value': 'host1'}
]
cpu_utilization = {
    'MeasureName': 'cpu_utilization',
    'MeasureValue ': str(instance_cpu_util_value)
}
memory_utilization = {
    'MeasureName': 'memory_utilization',
    'MeasureValue': str(instance_memory_util_value)
}
```

```
common_attributes = {
    'Dimensions': dimensions,
    'MeasureValueType': 'DOUBLE',
    'Time': current_time
}

records = [cpu_utilization, memory_utilization]

result =
self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
                          TableName=Constant.TABLE_NAME,
                          Records=records,
                          CommonAttributes=common_attributes)
```

※表現上、一部簡略化

データ挿入注意点①

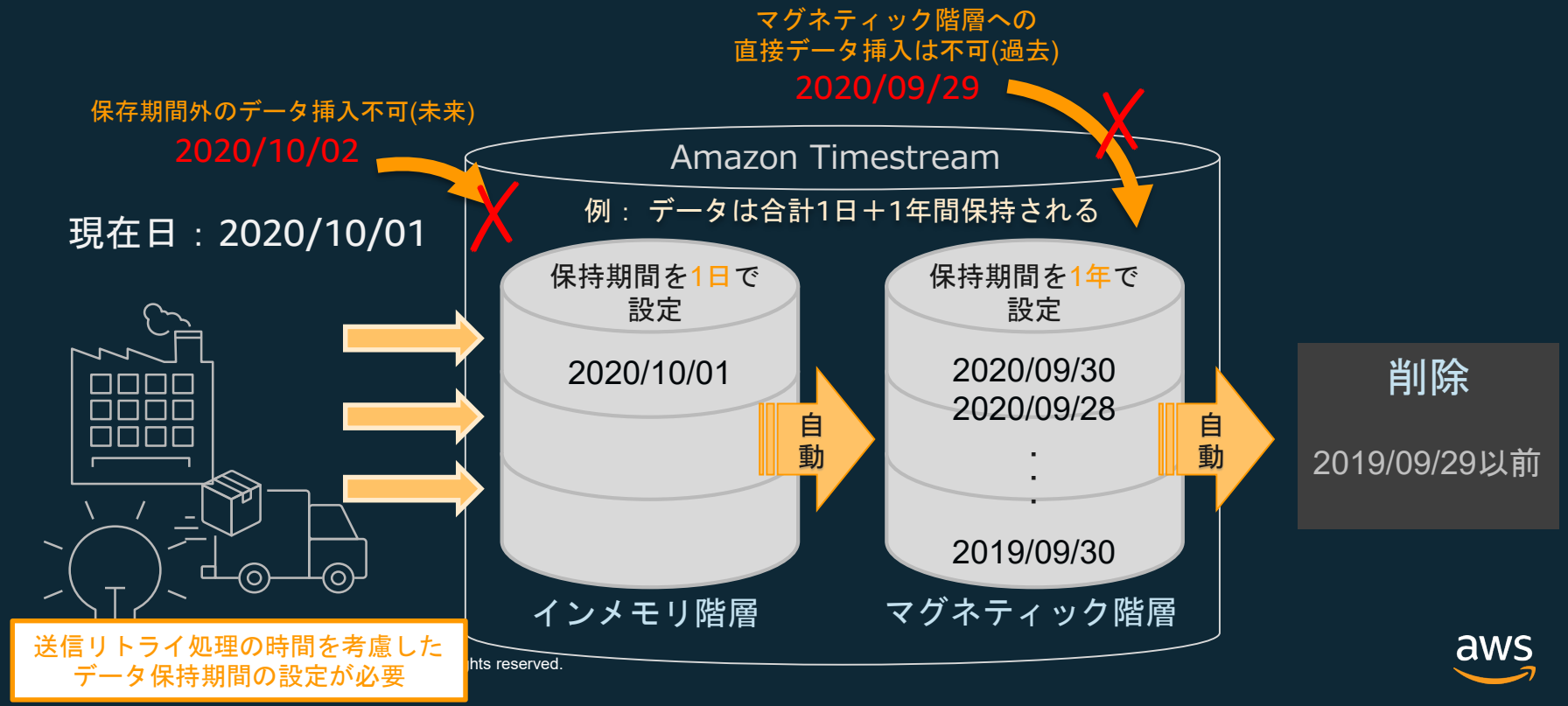
インメモリ階層の保持期間外のデータを挿入することが出来ない。



© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

データ挿入注意点①

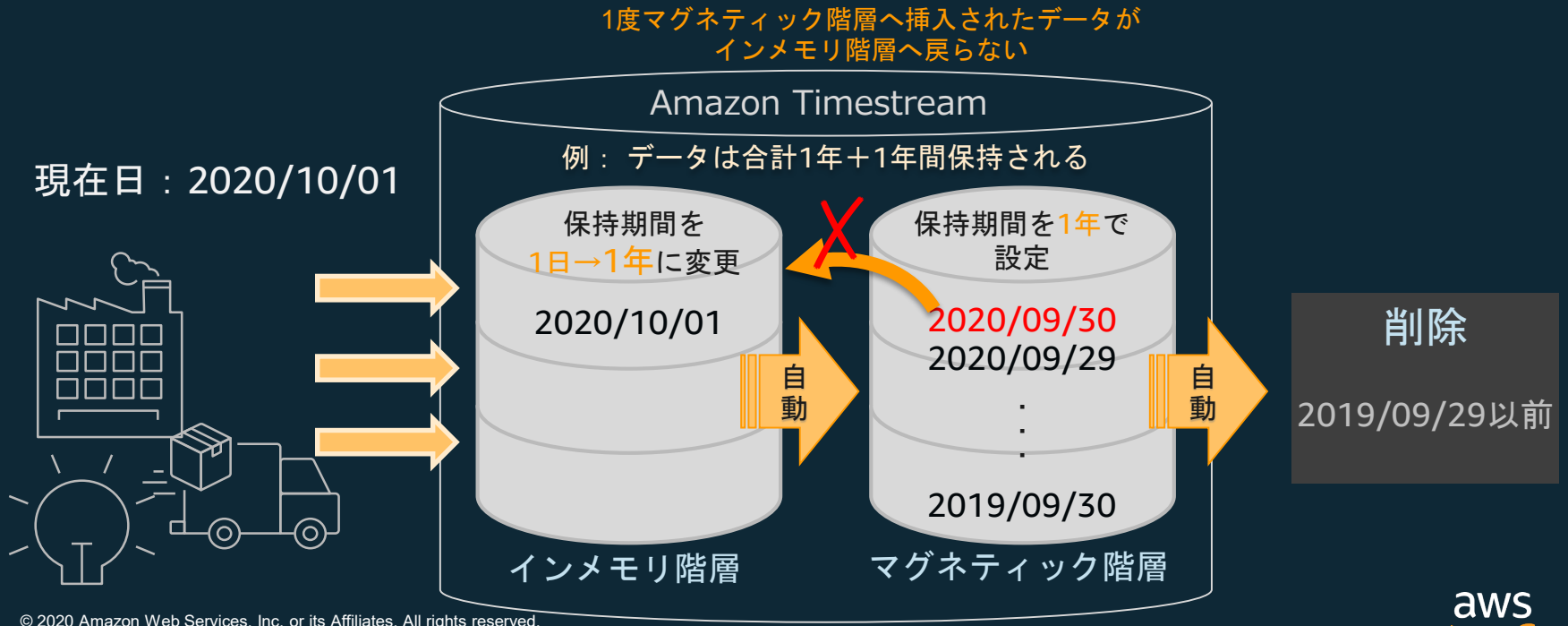
インメモリ階層の保持期間外のデータを挿入することが出来ない。



hts reserved.

データ挿入注意点②

保持期間を変更した場合は、変更したタイミング以降にTimestreamへ送信されたデータに対してのみ新しい保持期間が有効となる。



© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

出力：クエリプロセッシング

ANSI-2003 SQL 標準に準拠。独自クエリ言語の学習が不要

- 250以上のスカラー関数、集約関数、ウィンドウ関数をカバー
- 時系列機能、補間関数を提供
- SQL 結合句はSELF JOINのみをサポート
- UPDATE, DELETEステートメントは利用不可

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

時系列機能 - 2 種類の出力形式

フラットモデルと時系列モデルの2つのデータモデルをサポート。データはフラットモデルを使用して保存され、時系列分析には時系列モデルを使用。

フラットモデル

デフォルトデータモデル。時系列データを表形式で表す。各行は時系列内の特定の時間に対する測定値のアトミックなデータポイント。

time	vpc	instance_id	measure_name	measure_value :: double
2019-12-04 19:00:00.000000000	vpc-1a2b3c4d	i-1234567890abcdef0	cpu_utilization	35
2019-12-04 19:00:01.000000000	vpc-1a2b3c4d	i-1234567890abcdef0	cpu_utilization	38.2
2019-12-04 19:00:02.000000000	vpc-1a2b3c4d	i-1234567890abcdef0	cpu_utilization	45.3
2019-12-04 19:00:00.000000000	vpc-1a2b3c4d	i-1234567890abcdef0	memory_utilization	54.9
2019-12-04 19:00:01.000000000	vpc-1a2b3c4d	i-1234567890abcdef0	memory_utilization	42.6

時系列モデル(Time-Series)

時系列分析に使用されるデータモデル。特定のディメンションに対応したメジャーをタイムスタンプ付き値の配列で表す。

region	vpc	instance_id	cpu_utilization
us-east-1	vpc-1a2b3c4d	i-1234567890abcdef0	[{time：2019-12-04 19：00：00.000000000, value：35}, {time：2019-12-04 19：00：01.000000000,value：38.2}, {time：2019-12-04 19:00 ：02.000000000,value：45.3}, ...]

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

例：フラットモデルで取得

```
SELECT *
FROM MonitoringDB.sensor_metrics
AND time >= ago(2h)
```

```
SELECT *
FROM MonitoringDB.sensor_metrics
WHERE measure_name = 'cpu_utilization'
AND sensor_id = 'sensor-24Gju'
AND time >= ago(2h)
```

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

例：フラットモデルで取得

```
SELECT *
FROM MonitoringDB.sensor_metrics
AND time >= ago(2h)
```

```
SELECT *
FROM MonitoringDB.sensor_metrics
WHERE measure_name = 'cpu_utilization'
AND sensor_id = 'sensor-24Gju'
AND time >= ago(2h)
```

time	location	store	sensor_id	measure_name	measure_value::double	measure_value::bigint
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	35.0	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	51.8	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	30000
2020-06-17 19:00:00.000000000	Osaka	-	sensor-31Hij	cpu_utilization	51.8	-
2020-06-17 19:00:00.000000000	Osaka	-	sensor-31Hij	mem_utilization	32.1	-

aws

例：フラットモデルで取得

```
SELECT *
FROM MonitoringDB.sensor_metrics
AND time >= ago(2h)
```

time	location	store	sensor_id	measure_name	measure_value::double	measure_value::bigint
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	35.0	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	51.8	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	30000
2020-06-17 19:00:00.000000000	Osaka	-	sensor-31Hij	cpu_utilization	51.8	-
2020-06-17 19:00:00.000000000	Osaka	-	sensor-31Hij	mem_utilization	32.1	-

```
SELECT *
FROM MonitoringDB.sensor_metrics
WHERE measure_name = 'cpu_utilization'
AND sensor_id = 'sensor-24Gju'
AND time >= ago(2h)
```

time	location	store	sensor_id	measure_name	measure_value::double
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	35.0
2020-06-17 19:00:01.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	35.4
2020-06-17 19:00:02.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	36.1
2020-06-17 19:00:03.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	36.2
2020-06-17 19:00:04.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	35.8

例：時系列モデルへ変換

フラットモデル

time	location	store	sensor_id	measure_name	measure_value::double	measure_value::bigint
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	35.0	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	mem_utilization	51.8	-
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	networks_bytes	-	30000
2020-06-17 19:00:00.000000000	Osaka	-	sensor-31Hij	cpu_utilization	51.8	-
2020-06-17 19:00:00.000000000	Osaka	-	sensor-31Hij	mem_utilization	32.1	-

時系列モデル(Time-Series)

location	store	sensor_id	cpu_util_list
Tokyo	abc1a2b3c4d	sensor-24Gju	[{ time: 2020-11-03 21:35:33.000000000, value: 36.65 }, { time: 2020-11-03 21:36:51.000000000, value: 44.75 }... View 47 more row(s)]
Osaka	-	sensor-31Hij	[{ time: 2020-11-03 21:35:33.000000000, value: 8.75 }, { time: 2020-11-03 21:38:51.000000000, value: 1.78 }... View 47 more row(s)]
Okinawa	xyz5l5o5n5m	sensor-67Kju	{ time: 2020-11-03 21:34:33.000000000, value: 4.05 }, { time: 2020-11-03 21:36:51.000000000, value: 1.24 }... View 47 more row(s)]

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

例：時系列モデルへ変換

フラットモデル

time	location	store	sensor_id	measure_name	measure_value::double	measure_value::bigint
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	35.0	-
2020-06-17 19:00:00.000000000	Tokyo	a…				
2020-06-17 19:00:00.000000000	Tokyo	a…				
2020-06-17 19:00:00.000000000	Osaka	-				
2020-06-17 19:00:00.000000000	Osaka	-				

時系列モデル(Time-Series)

location	store	sensor_id	cpu_util_list
Tokyo	abc1a2b3c4d	sensor-24Gju	[{ time: 2020-11-03 21:35:33.000000000, value: 36.65 }, { time: 2020-11-03 21:36:51.000000000, value: 44.75 }... View 47 more row(s)]
	-	sensor-31Hij	[{ time: 2020-11-03 21:35:33.000000000, value: 8.75 }, { time: 2020-11-03 21:38:51.000000000, value: 1.78 }... View 47 more row(s)]
	xyz5l5o5n5m	sensor-67Kju	{ time: 2020-11-03 21:34:33.000000000, value: 4.05 }, { time: 2020-11-03 21:36:51.000000000, value: 1.24 }... View 47 more row(s)]

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

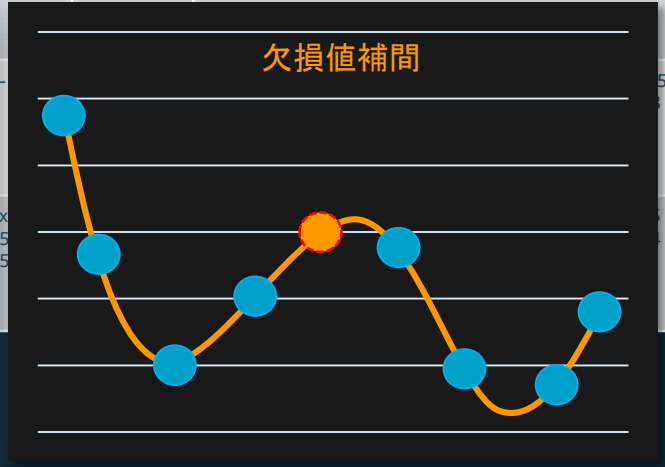
例：時系列モデルへ変換

フラットモデル

time	location	store	sensor_id	measure_name	measure_value::double	measure_value::bigint
2020-06-17 19:00:00.000000000	Tokyo	abc1a2b3c4d	sensor-24Gju	cpu_utilization	35.0	-
2020-06-17 19:00:00.000000000	Tokyo	ab...				
2020-06-17 19:00:00.000000000	Tokyo	ab...				
2020-06-17 19:00:00.000000000	Osaka	-				
2020-06-17 19:00:00.000000000	Osaka	-				

時系列モデル(Time-Series)

location	store	sensor_id	cpu_util_list
Tokyo	abc1a2b3c4d	sensor-24Gju	[{ time: 2020-11-03 21:35:33.000000000, value: 36.65 }, { time: 2020-11-03 21:36:51.000000000, value: 44.75 }... View 47 more row(s)]



© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

例：時系列モデルへ変換

CREATE_TIME_SERIES()関数で、時系列データに変換

```
SELECT location, store, sensor_id,
 CREATE_TIME_SERIES(time, measure_value::double) as cpu_util_list
FROM MonitoringDB.sensor_metrics
WHERE measure_name = 'cpu_utilization' AND time >= ago(2h)
GROUP BY location, store, sensor_id
```

location	store	sensor_id	cpu_util_list
Tokyo	abc1a2b3c4d	sensor-24Gju	[{ time: 2020-11-03 21:35:33.000000000, value: 36.65714366101487 }, { time: 2020-11-03 21:36:51.000000000, value: 44.751706964910525 }... View 47 more row(s)]
Osaka	-	sensor-31Hij	[{ time: 2020-11-03 21:35:33.000000000, value: 8.757432549200132 }, { time: 2020-11-03 21:38:51.000000000, value: 1.7819709974945508 }... View 47 more row(s)]
Okinawa	xyz5l5o5n5m	sensor-67Kju	{ time: 2020-11-03 21:34:33.000000000, value: 4.050995092013331 }, { time: 2020-11-03 21:36:51.000000000, value: 1.2426470512230736 }... View 47 more row(s)]

例：時系列モデルへ変換

CREATE_TIME_SERIES()関数で、時系列データに変換

```
SELECT location, store, sensor_id,
 CREATE_TIME_SERIES(time, measure_value::double) as cpu_util_list
FROM MonitoringDB.sensor_metrics
WHERE measure_name = 'cpu_utilization' AND time >= ago(2h)
GROUP BY location, store, sensor_id
```

location	store	sensor_id	cpu_util_list
Tokyo	abc1a2b3c4d	sensor-24Gju	[{ time: 2020-11-03 21:35:33.000000000, value: 36.65714366101487 }, { time: 2020-11-03 21:36:51.000000000, value: 44.751706964910525 }... View 47 more row(s)]
Osaka	-	sensor-31Hij	[{ time: 2020-11-03 21:35:33.000000000, value: 8.757432549200132 }, { time: 2020-11-03 21:38:51.000000000, value: 1.7819709974945508 }... View 47 more row(s)]
Okinawa	xyz5l5o5n5m	sensor-67Kju	{ time: 2020-11-03 21:34:33.000000000, value: 4.050995092013331 }, { time: 2020-11-03 21:36:51.000000000, value: 1.2426470512230736 }... View 47 more row(s)]

時系列機能 - 補間関数

時系列データ型を利用して欠損値の補間が可能。4種の補間手法を提供し、利用者側での複雑な計算処理が不要。

補間手法	線形補間	キュービック スプライン補間 (3次スプライン補間)	Last Observation Carried Forward補間 (LOCF補間)	固定値補間
対応関数	interpolate_linear()	interpolate_spline_cubic()	interpolate_locf()	interpolate_fill()

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

例：線形補間を利用して15秒間隔で挿入したデータを10秒の粒度に補間して取得

```
SELECT location, store, sensor_id,
       INTERPOLATE_LINEAR(                    -- 補間関数
         CREATE_TIME_SERIES(time, measure_value::double),  -- 対象データの時系列データ型
         SEQUENCE(ago(2h), now(), 10s)        -- 補間したい対象のタイムスタンプ(単体/複数可)
       ) AS interpolated_cpu_utilization
FROM MonitoringDB.sensor_metrics
WHERE measure_name = 'cpu_utilization' AND time >= ago(2h)
GROUP BY location, store, sensor_id
```

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

例：複数メジャーを1行にまとめて表示

要件によってはメジャー毎にレコードを分けたくない場合がある

Amazon Timestreamのデータ構造

time	sensor_name	measure_name	measure_value::double	measure_value::bigint
2020-06-17 19:00:00.000000000	sensor-24Gju	cpu_utilization	35.0	-
2020-06-17 19:00:00.000000000	sensor-24Gju	mem_utilization	51.8	-
2020-06-17 19:00:00.000000000	sensor-24Gju	networks_bytes	null	30000
2020-06-17 19:00:01.000000000	sensor-24Gju	cpu_utilization	34.8	-
2020-06-17 19:00:01.000000000	sensor-24Gju	mem_utilization	52.4	-

実現したい形式

time	sensor_name	cpu_utilization	mem_utilization	networks_bytes
2020-06-17 19:00:00.00	sensor-24Gju	35.0	51.8	30000
2020-06-17 19:00:01.00	sensor-24Gju	34.8	52.4	26400

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

例：複数メジャーを1行にまとめて表示

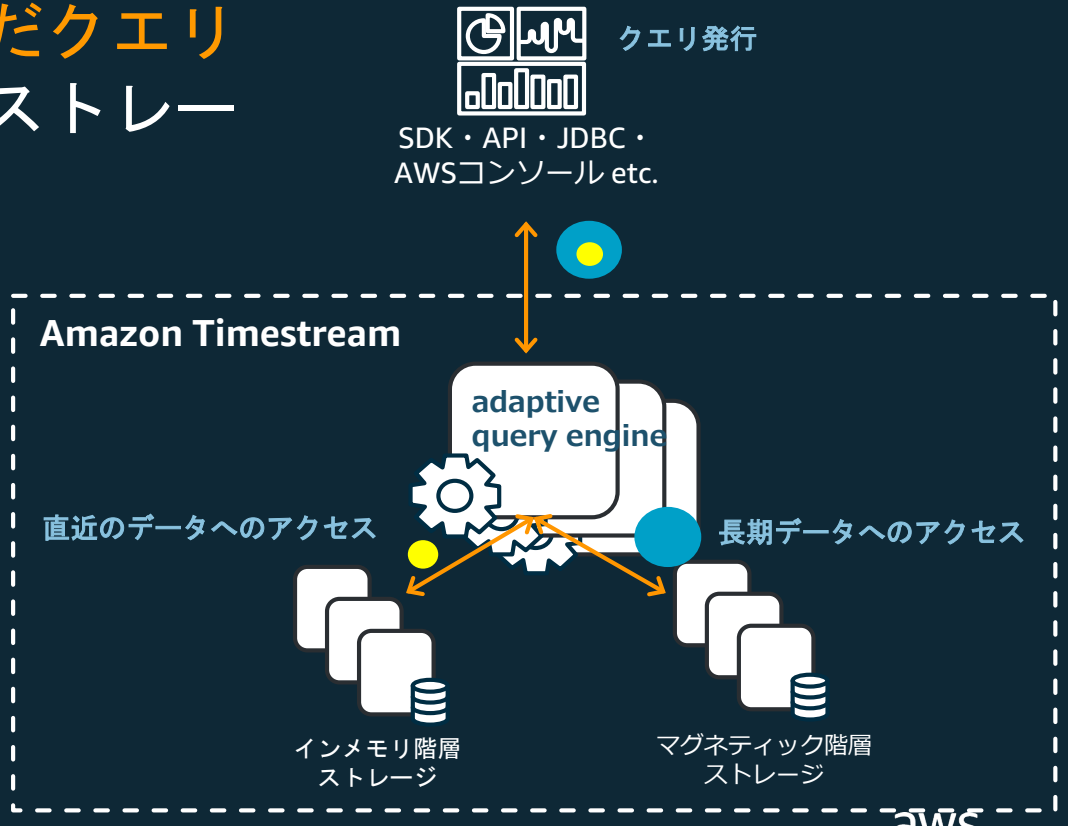
```
SELECT sensor_id, bin(time, 5s) AS time_bin,
  avg(CASE WHEN measure_name = 'cpu_utilization' THEN measure_value::double ELSE NULL END) AS avg_5s_cpu_util,
  avg(CASE WHEN measure_name = 'mem_utilization' THEN measure_value::double ELSE NULL END) AS avg_5s_mem_util,
  avg(CASE WHEN measure_name = 'networks_bytes' THEN measure_value::double ELSE NULL END) AS avg_5s_networks,
FROM MonitoringDB.sensor_metrics
WHERE measure_name IN ('cpu_utilization', 'mem_utilization', 'networks_bytes')
AND time > ago(2h)
AND sensor_id = 'sensor-24Gju'
GROUP BY sensor_id, bin(time, 5s)
```

bin()関数で5秒毎にまとめ、その平均値を取得

time_bin	sensor_id	cpu_utilization	mem_utilization	mem_utilization
2020-06-17 19:00:00.000000000	sensor-24Gju	20.3	10.9	9.42
2020-06-17 19:00:05.000000000	sensor-24Gju	23.6	12.3	9.41
2020-06-17 19:00:10.000000000	sensor-24Gju	24.2	14.1	9.39
2020-06-17 19:00:15.000000000	sensor-24Gju	21.2	11.4	9.33

Adaptive Query Engineによる最適化

ストレージ階層間を跨いだクエリを実施。クエリ発行時にストレージの指定が不要。



© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

出力：クエリ発行

標準

- データへのクエリはAWS SDK / Timestream API を利用
  - SDK: Java, Python, Golang, Node.js, .NET, etc.
  - AWS CLI

インテグレーション

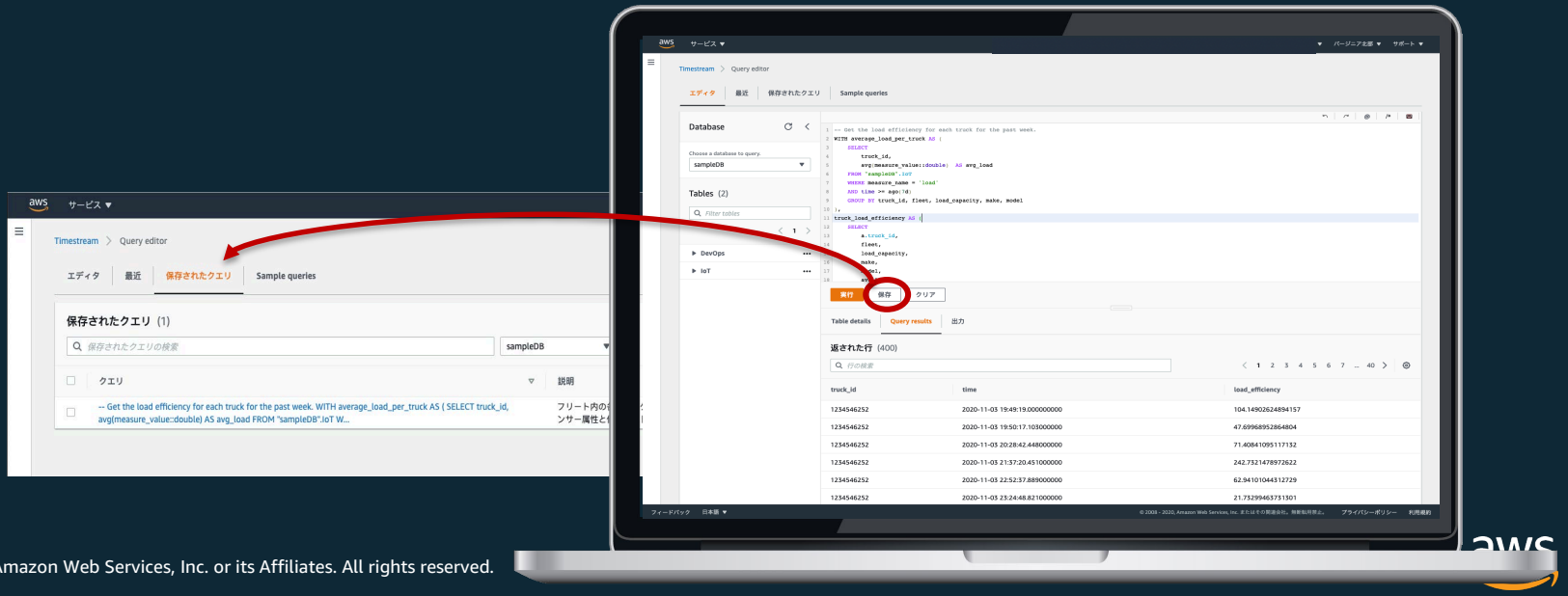
- JDBC Driver
  - SQL Workbench/J, DataGrip, DBVisualizer, etc.

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

Query Editor機能

- 専用クライアントを使用せずにマネジメントコンソールから直接テーブルにクエリを発行することが可能
- クエリの保存機能で実行頻度の高いクエリを保存可能



© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

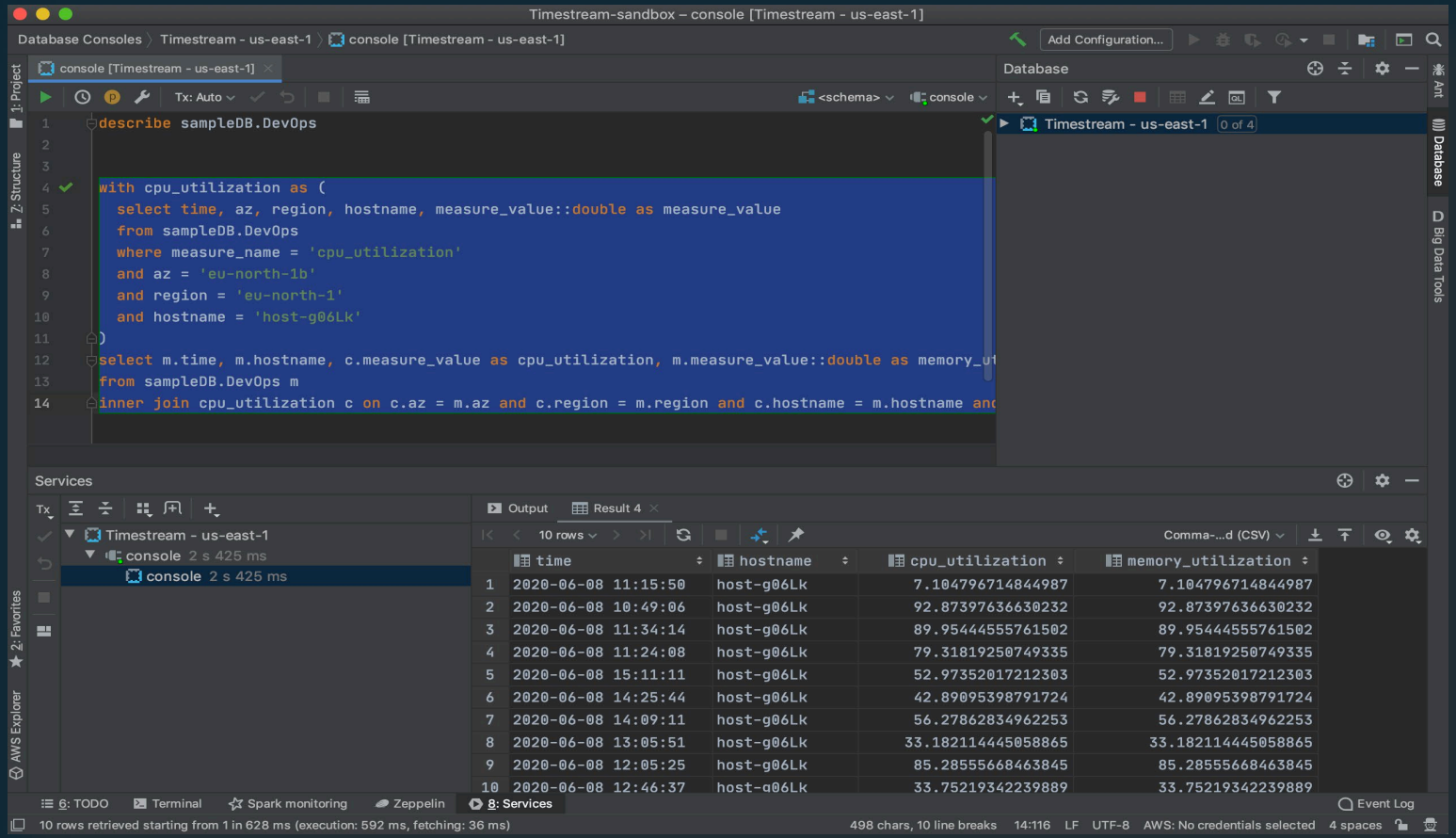
出力：分析・可視化ツール

インテグレーション

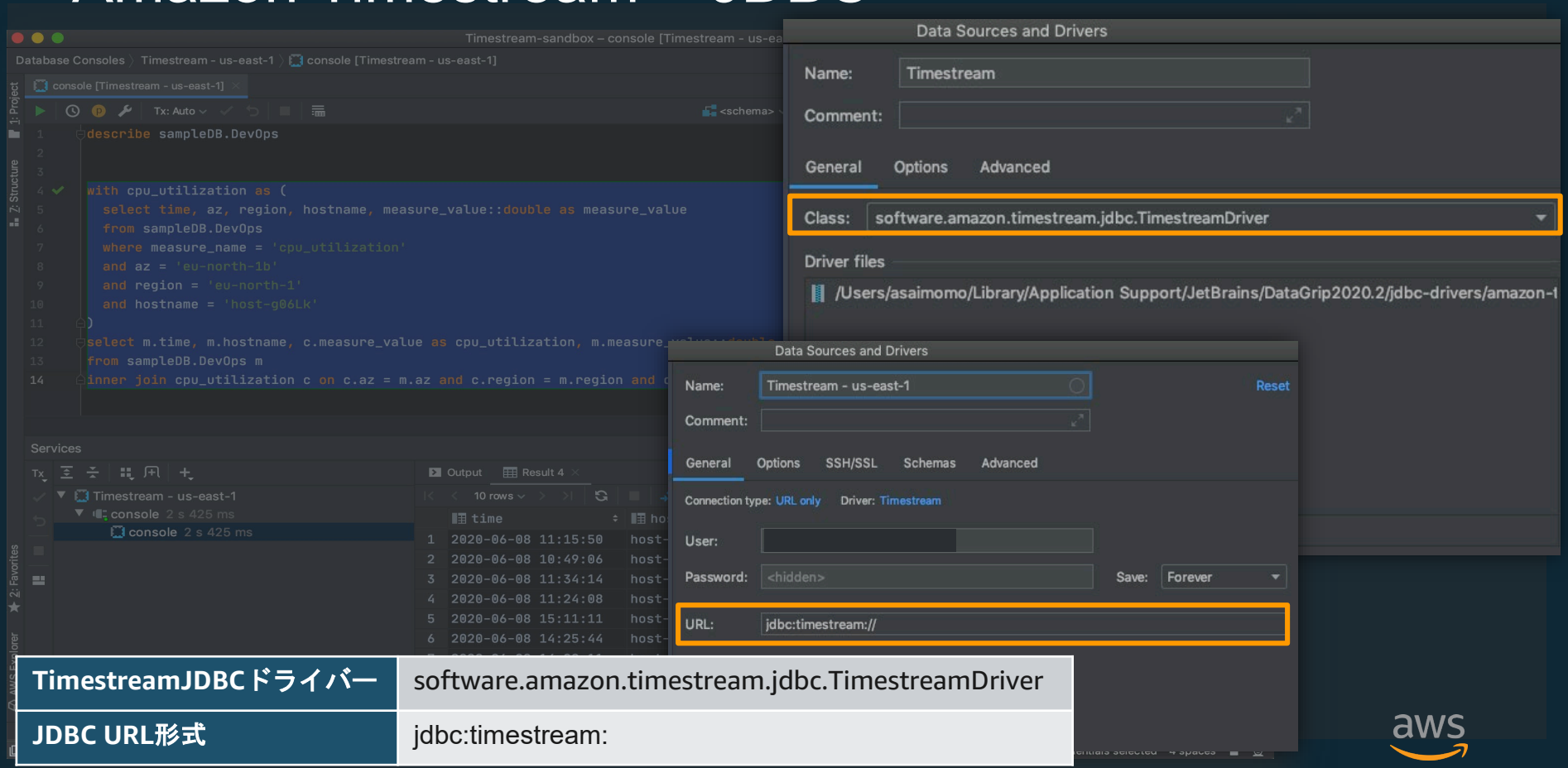
- Amazon QuickSight
- Grafana (Open Source Edition)
- Amazon SageMaker Notebook (AWSLabs GitHub Integration)

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Amazon Timestream + JDBC DataGripを利用した例

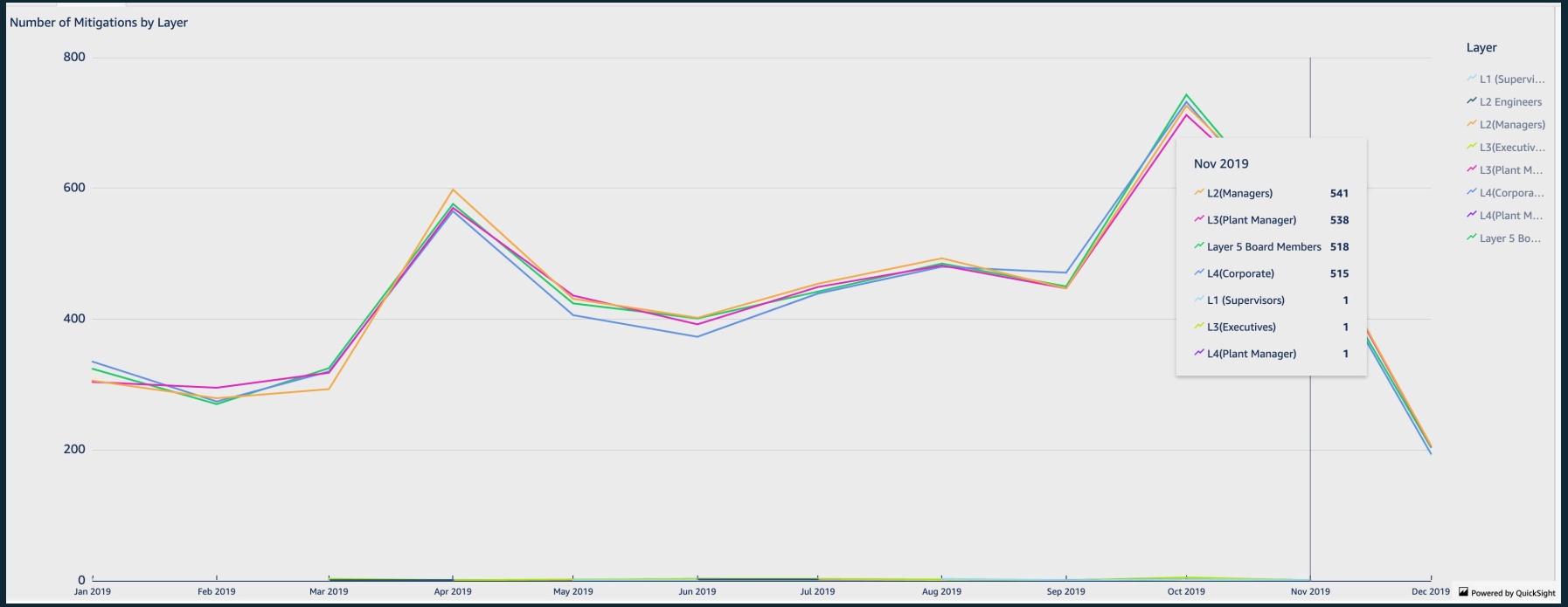


Amazon Timestream + JDBC DataGripを利用した例



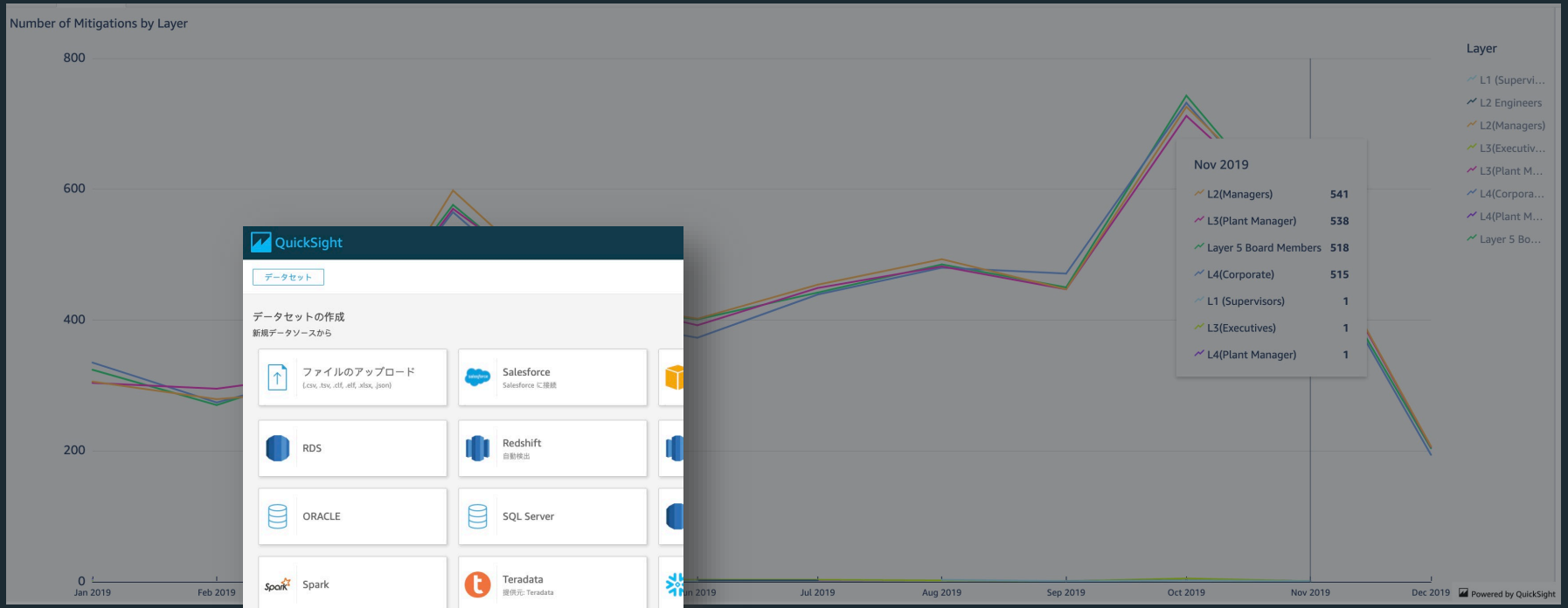
TimestreamJDBCドライバー	software.amazon.timestream.jdbc.TimestreamDriver
JDBC URL形式	jdbc:timestream:

Amazon Timestream + Amazon QuickSight



© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

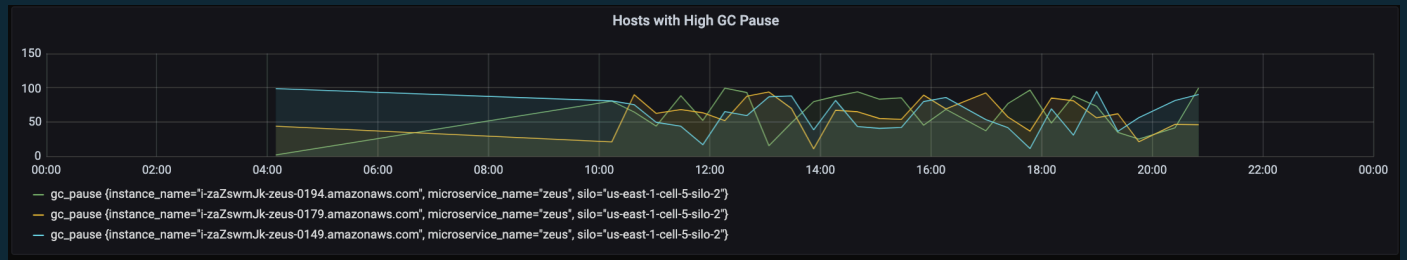
Amazon Timestream + Amazon QuickSight

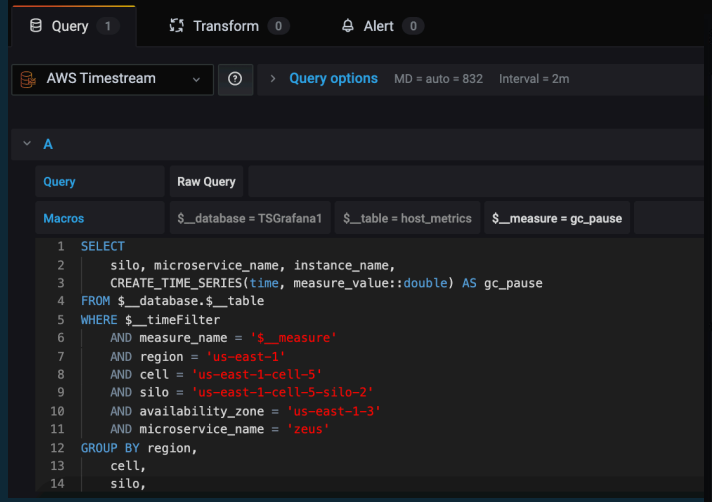


QuickSightの画面でTimestreamのデータセットが選択可能

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

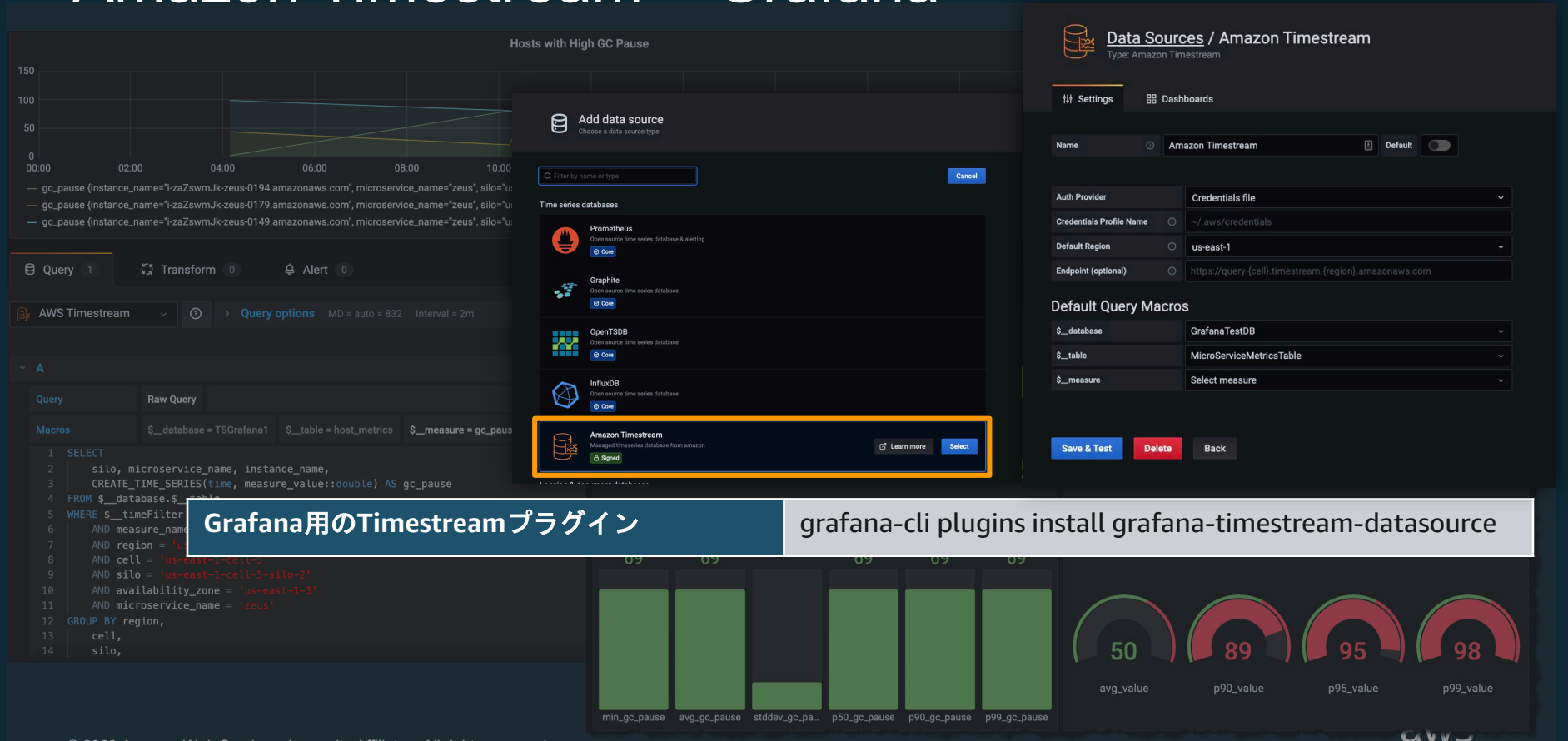
Amazon Timestream + Grafana





© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Amazon Timestream + Grafana



Grafana用のTimestreamプラグイン

grafana-cli plugins install grafana-timestream-datasource

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

再掲

時系列データの主なユースケース

IoT Application

デバイスのセンサーから動作中のエネルギーデータを収集し、稼働期間の特定やエネルギー利用状況をモニタリングし、今後のエネルギー需要を予測する。

Analytics Application

クリックストリームデータを保存・分析し、カスタマーの閲覧経路、サイト滞在時間や離脱ポイントなどを把握。サイト構成やUXの改善に活かす。

DevOps Application

CPU 、メモリ使用率、ネットワークデータ、IOPS などのパフォーマンスメトリックスを収集・分析し、サーバの監視や異常値を検知。システム稼働状況を最適化する。

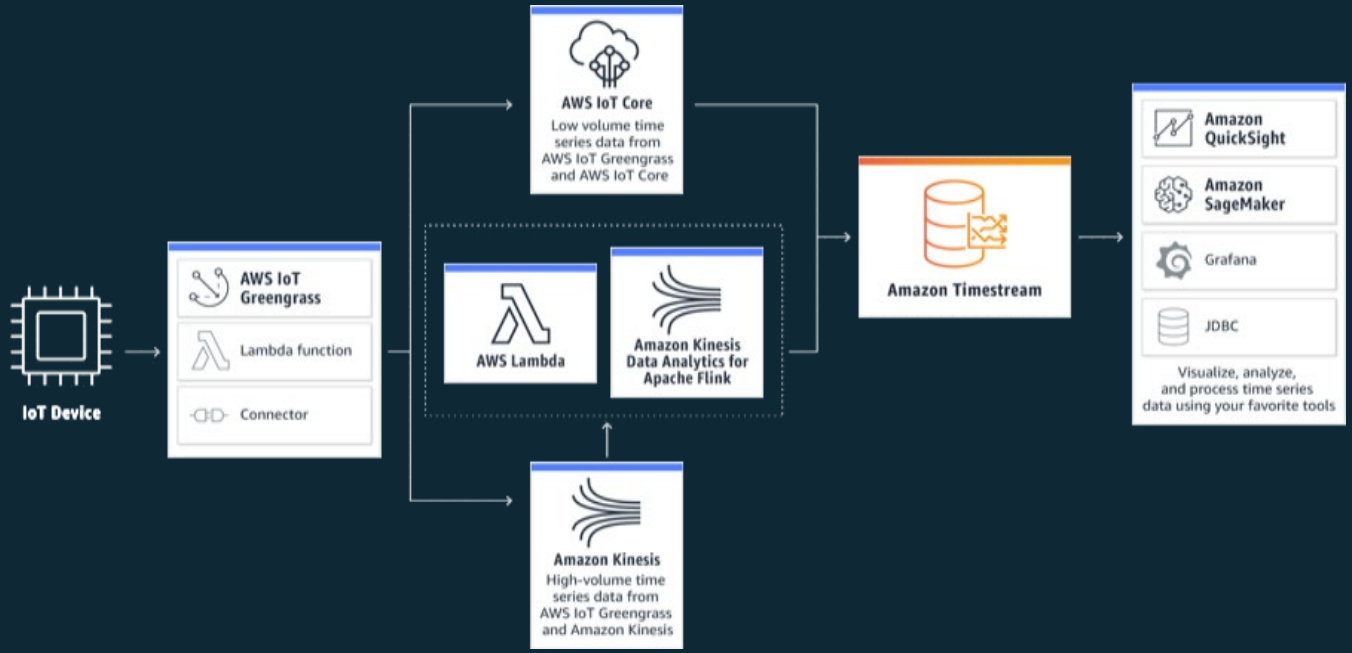
© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

IoTソリューション x Timestream 構成例

IoT Application | Analytics Application | DevOps Application

IoTデバイスから収集したデータをAWS IoT Coreルールアクションを介して
Amazon Timestreamにルーティング可能。



© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

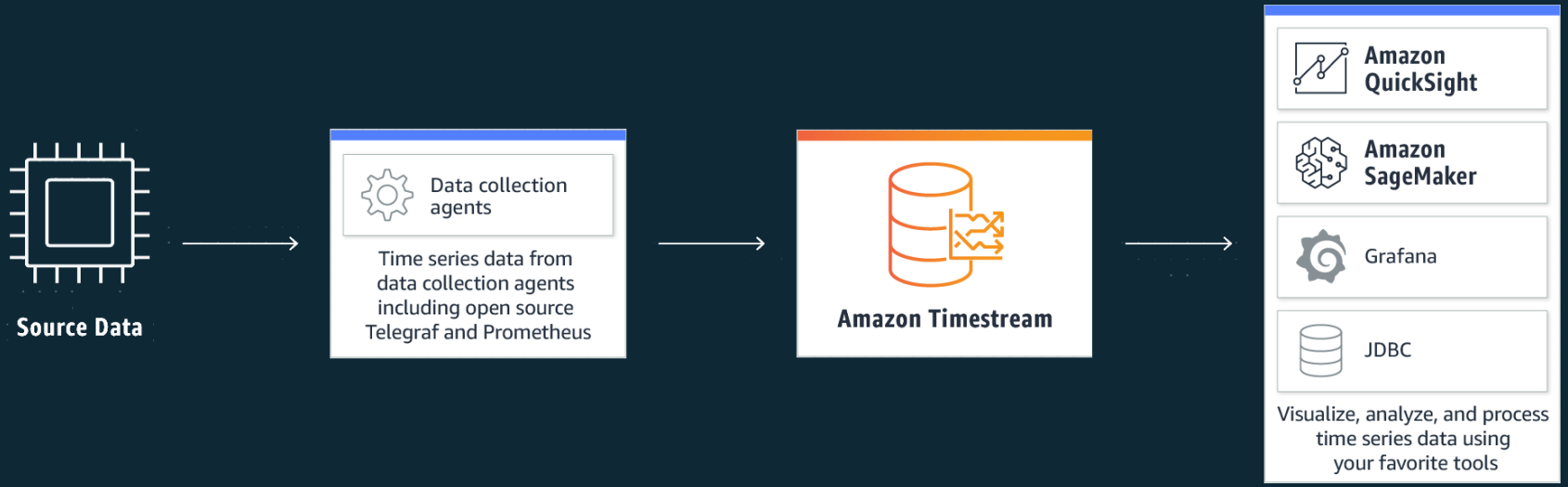
Analytics x Timestream 構成例

Apache Flinkを使用して、Amazon Kinesis、Amazon MSK、Apache Kafka等のストリーミング処理サービスからAmazon Timestreamへデータの直接転送が可能。

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

DevOps x Timestream 構成例

Telegraf(OSS)のエージェントを監視対象リソースへインストールしてAmazon Timestreamへ直接データを転送することが可能。



© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Agenda

- イントロダクション
- アーキテクチャ概念と仕組み
- データの入出力とインテグレーション
- 運用
- 価格と参考リソース
- まとめ

運用

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

ロギングとモニタリング

- Amazon CloudWatch
  - メトリクスの集計(1分毎)、アラームの作成が可能

メトリクス	メトリクス概要	集計可能単位
SuccessfulRequestLatency	成功したリクエストの経過時間 成功したリクエストの数	TableName/DatabaseName/Operation
SystemErrors	内部サービスエラー	Operation
UserErrors	InvalidRequestエラー	Operation
DataScannedBytes	スキャンされたデータの量	Operation

- AWSCloudTrail
  - API操作とユーザーアクティビティのログ監視

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

可用性・耐障害性

- リージョン内で複数AZ構成をとり、データのレプリケーションや障害時のフェイルオーバーなどをフルマネージドで実現
    - データ挿入時、3 つのAZでの書き込み完了確認が取れた時点でオペレーションの成功が返される
    - データレプリケーションはクォーラムベース
      →ノードまたは AZ 全体が損失されても耐久性や可用性が損なわれない
- 99.99%のSLAを提供

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

バックアップ・リストア

自動	各階層のデータは複数AZに自動で複製 インメモリ階層のデータはS3へ継続的なバックアップを実施 障害発生時は自動でデータの復旧・リストアを実施
手動	Timestreamの機能では手動による静止点バックアップやリストアを行う機能は提供されていない

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

パフォーマンスとチューニング

- 容量とパフォーマンス状況に合わせ、読み込み・書き込み・ストレージを個別に自動でスケールリング
- データ挿入時に自動でパーティション分割を実施。パーティション内でインデックスを自動設定

チューニングポイント

入力 (書き込み)	• 1つのリクエストで複数レコードを送信(バッチ処理) • 各レコード共通の属性は、共通項目としてまとめて送信 • タイムスタンプを昇順にソートして送信
出力 (クエリ)	• 取得期間と対象 (SQL where句の範囲)を最小限にして、スキャンデータ量を削減

Agenda

- イントロダクション
- アーキテクチャ概念と仕組み
- データの入出力とインテグレーション
- 運用
- 価格と参考リソース
- まとめ

価格と参考リソース

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Timestream の価格設定

対象リージョン：米国東部（バージニア州北部）、米国東部（オハイオ州）、米国西部（オレゴン州）
※ EU（アイルランド）は異なり価格設定となります。詳細はドキュメントをご参照ください
https://aws.amazon.com/jp/timestream/pricing/
https://docs.aws.amazon.com/ja_jp/timestream/latest/developerguide/metering-and-pricing.html

データの保持

インメモリ階層(Tier)

1 GB あたりの料金は、
1 時間あたり $0.036

マグネティック階層(Tier)

1 GB あたりの料金は、
1 か月あたり $0.03

SSD階層(Tier)

近日公開予定

1 GB あたりの 料金は、
1 日あたり $0.01

書き込みとクエリ

書き込み

書き込みバイト数に応じて料金発生。1 回の書き込みは1 KB

100 万回書き込みあたりの料金は $0.50

1 KB未満の書き込みは1 KBの単位に切り上げ

クエリ

スキャンされたバイト数に対して料金発生

クエリでスキャンされたデータ 1 GB あたりの料金は $0.01

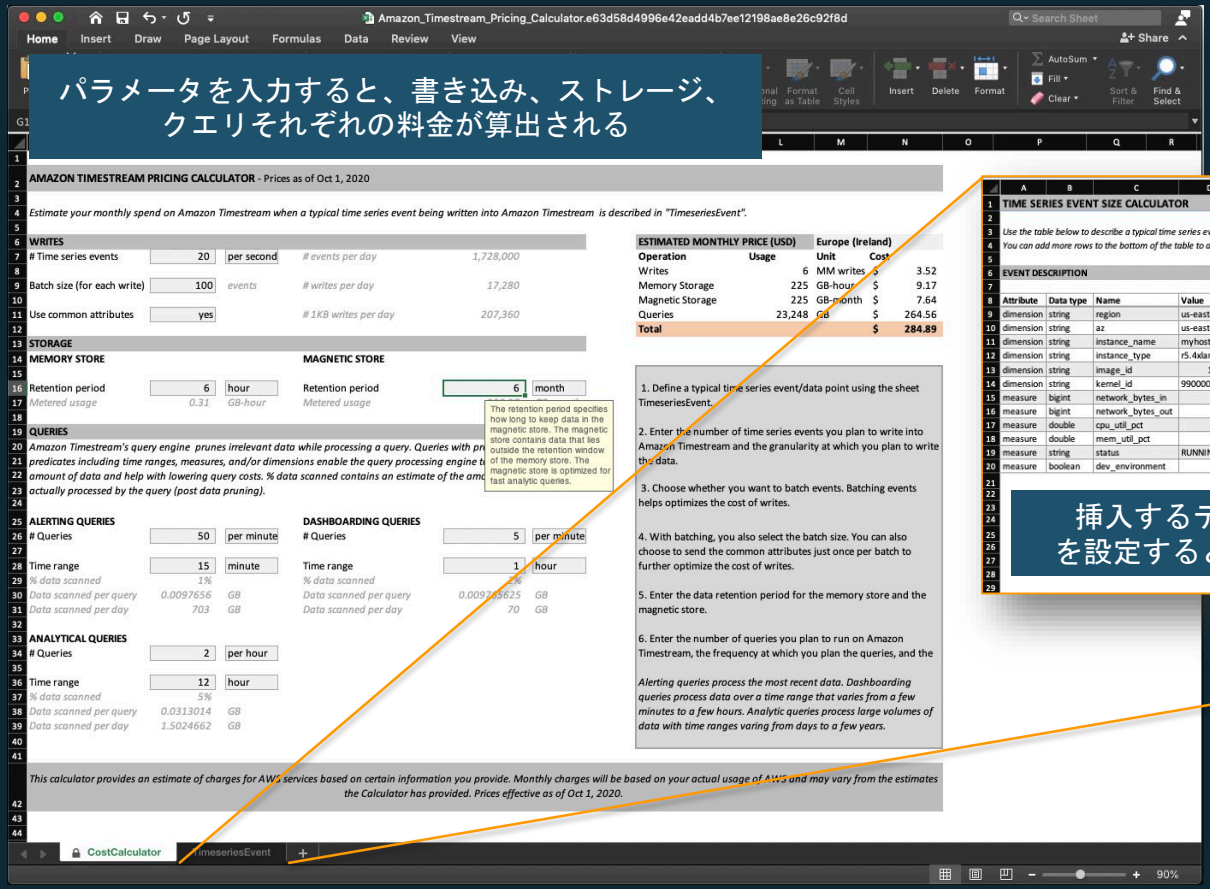
10 MB 未満のクエリは 10 MB の単位に切り上げ

コスト最適化ベストプラクティス
https://docs.aws.amazon.com/timestream/latest/developerguide/metering-and-pricing.cost-optimization.html

料金計算ツール

https://aws.amazon.com/jp/timestream/pricing/

パラメータを入力すると、書き込み、ストレージ、
クエリそれぞれの料金が算出される



AMAZON TIMESTREAM PRICING CALCULATOR - Prices as of Oct 1, 2020

Estimate your monthly spend on Amazon Timestream when a typical time series event being written into Amazon Timestream is described in "TimeseriesEvent".

WRITES

# Time series events	20	per second	# events per day	1,728,000
Batch size (for each write)	100	events	# writes per day	17,280
Use common attributes	yes		# 1KB writes per day	207,360

STORAGE

MEMORY STORE			MAGNETIC STORE		
Retention period	6	hour	Retention period	6	month
Metered usage	0.31	GB-hour	Metered usage		

The retention period specifies how long to keep data in the magnetic store. The magnetic store contains data that lies outside the retention window of the memory store. The magnetic store is optimized for fast analytic queries.

QUERIES

Amazon Timestream's query engine prunes irrelevant data while processing a query. Queries with pr... predicates including time ranges, measures, and/or dimensions enable the query processing engine t... amount of data and help with lowering query costs. % data scanned contains an estimate of the amo... actually processed by the query (post data pruning).

ALERTING QUERIES

# Queries	50	per minute			
Time range	15	minute			
% data scanned	1%				
Data scanned per query	0.0097656	GB			
Data scanned per day	703	GB			

DASHBOARDING QUERIES

# Queries	5	per minute			
Time range	1	hour			
% data scanned	3%				
Data scanned per query	0.009765625	GB			
Data scanned per day	70	GB			

ANALYTICAL QUERIES

# Queries	2	per hour			
Time range	12	hour			
% data scanned	5%				
Data scanned per query	0.0313014	GB			
Data scanned per day	1.5024662	GB			

ESTIMATED MONTHLY PRICE (USD) — Europe (Ireland)

Operation	Usage	Unit	Cost
Writes	6	MM writes	$ 3.52
Memory Storage	225	GB-hour	$ 9.17
Magnetic Storage	225	GB-month	$ 7.64
Queries	23,248	GB	$ 264.56
Total			**$ 284.89**

1. Define a typical time series event/data point using the sheet TimeseriesEvent.

2. Enter the number of time series events you plan to write into Amazon Timestream and the granularity at which you plan to write the data.

3. Choose whether you want to batch events. Batching events helps optimizes the cost of writes.

4. With batching, you also select the batch size. You can also choose to send the common attributes just once per batch to further optimize the cost of writes.

5. Enter the data retention period for the memory store and the magnetic store.

6. Enter the number of queries you plan to run on Amazon Timestream, the frequency at which you plan the queries, and the

Alerting queries process the most recent data. Dashboarding queries process data over a time range that varies from a few minutes to a few hours. Analytic queries process large volumes of data with time ranges varing from days to a few years.

This calculator provides an estimate of charges for AWS services based on certain information you provide. Monthly charges will be based on your actual usage of AWS and may vary from the estimates the Calculator has provided. Prices effective as of Oct 1, 2020.

CostCalculator | TimeseriesEvent

TIME SERIES EVENT SIZE CALCULATOR

Use the table below to describe a typical time series event being sent to Amazon Timestream. The time series event must have one or more dimensions and measures.
You can add more rows to the bottom of the table to add dimensions and measures. The timestamp will be added automatically.

EVENT DESCRIPTION

Attribute	Data type	Name	Value	NameLen	ValueLen	IsCommon	WriteBytes	RecordStorageBytes
dimension	string	region	us-east-1	6	9	yes	15	
dimension	string	az	us-east-1a	2	10	yes	12	
dimension	string	instance_name	myhost1	13	7	yes	20	
dimension	string	instance_type	r5.4xlarge	13	10	yes	23	
dimension	string	image_id	123456	8	6	yes	14	
dimension	string	kernel_id	990000AAQ	9	9	yes	18	
measure	bigint	network_bytes_in	800	16	0	no	24	134
measure	bigint	network_bytes_out	1440	17	0	no	25	135
measure	double	cpu_util_pct	23.5	12	0	no	20	130
measure	double	mem_util_pct	45.6	12	0	no	20	130
measure	string	status	RUNNING	6	7	yes	13	123
measure	boolean	dev_environment	1	15	0	no	16	126

EVENT SIZE IN BYTES

Type	Value	Unit
Write size	220	bytes
Write size excluding common attributes	120	bytes
Common attributes size	116	bytes
Storage size	778	bytes

1. Describe a typical time series event by adding rows to the table. Each row is either a dimension or measure.

2. When creating a dimension or measure, enter the data type, name, value, and if it is a common attribute.

3. Common attributes are the dimensions and measure names ... common across multiple events written to Amazon ...

...columns NameLen, ValueLen, WriteBytes, and ...StorageBytes are automatically calculated.

挿入するディメンションやメジャー
を設定するとbytes数が自動算出される

Amazon Timestream Documents

Amazon Timestreamドキュメント
https://aws.amazon.com/jp/timestream

始め方(デモ動画付き。Integrationの接続方法も動画でご提供)
https://aws.amazon.com/jp/timestream/getting-started/?nc=sn&loc=4

FAQ
https://aws.amazon.com/jp/timestream/faq

Developer Guide
https://docs.aws.amazon.com/timestream/latest/developerguide/what-is-timestream.html

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Amazon Timestream Blog

Amazon Timestreamであらゆる規模の時系列データを保存・アクセス（Store and Access Time Series Data at Any Scale with Amazon Timestream – Now Generally Available）

https://aws.amazon.com/jp/blogs/aws/store-and-access-time-series-data-at-any-scale-with-amazon-timestream-now-generally-available/

Amazon Timestreamを使用して、ペタバイト級の時系列データからリアルタイムの洞察を導き出（Deriving real-time insights over petabytes of time series data with Amazon Timestream）

https://aws.amazon.com/jp/blogs/database/deriving-real-time-insights-over-petabytes-of-time-series-data-with-amazon-timestream/

Amazon Timestreamと、OSSのTelegrafやGrafanaを利用してDevOpsワークロードのデー収集やモニタリングを行う（Collecting, storing, and analyzing your DevOps workloads with open-source Telegraf, Amazon Timestream, and Grafana）

https://aws.amazon.com/jp/blogs/database/collecting-storing-and-analyzing-your-devops-workloads-with-open-source-telegraf-amazon-timestream-and-grafana/

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

AWS Labs on GitHub: Amazon Timestream

Sample Applications (available in various languages):
https://github.com/awslabs/amazon-timestream-tools/tree/master/sample_apps

Working with other tools and services:
https://github.com/awslabs/amazon-timestream-tools/tree/master/integrations

Kinesis Data Analytics for Apache Flink connector example:
https://github.com/awslabs/amazon-timestream-tools/tree/master/integrations/flink_connector

SageMaker example:
https://github.com/awslabs/amazon-timestream-tools/tree/master/integrations/sagemaker

Telegraf example:
https://github.com/awslabs/amazon-timestream-tools/tree/master/integrations/telegraf

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

Agenda

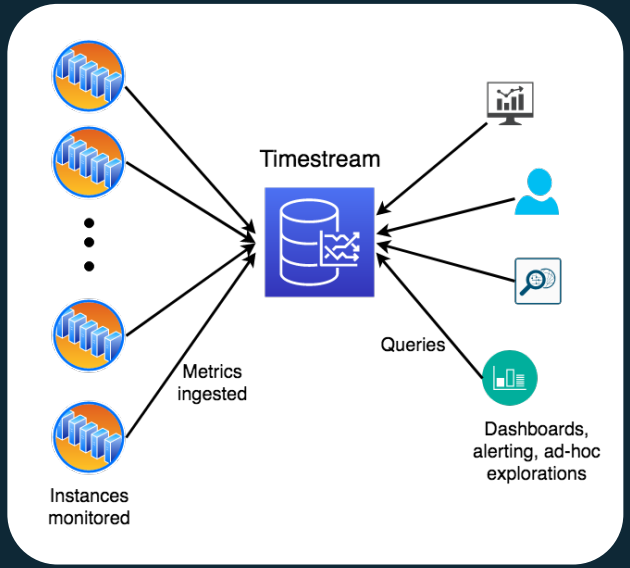
- イントロダクション
- アーキテクチャ概念と仕組み
- データの入出力とインテグレーション
- 運用
- 価格と参考リソース
- まとめ

まとめ

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Amazon Timestream まとめ

高速でスケーラブルな時系列専用のサーバーレスデータベース



- サーバーレスであり、基盤となるインフラストラクチャの管理が不要
- 自動スケーリング機能により、1 日あたり数兆のイベント処理が可能
- 時系列専用データベースで、標準SQLと組み込みの時系列関数を用意

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

構築・設計・運用にかかる時間を削減して

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

本来のビジネス価値を高める仕事へ

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

投資していきましょう！

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Q&A

お答えできなかったご質問については

AWS Japan Blog

「https://aws.amazon.com/jp/blogs/news/」にて

後日掲載します。

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

AWS の日本語資料の場所「AWS 資料」で検索

https://amzn.to/JPArchive

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

AWS Well-Architected 個別技術相談会

毎週”W-A個別技術相談会”を実施中

- AWSのソリューションアーキテクト(SA)に対策などを相談することも可能
- 申込みはイベント告知サイトから

(https://aws.amazon.com/jp/about-aws/events/)

AWS イベント で[検索]

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

ご視聴ありがとうございました

AWS 公式 Webinar
https://amzn.to/JPWebinar

過去資料
https://amzn.to/JPArchive

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Appendix

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Appendix

データベース・テーブル設計時の考慮ポイント

- セキュリティ要件
  - データ毎に異なるAWS KMSキーを使用した暗号化を実施したい
    → AWS KMSキーはデータベース単位に設定するため、データベースで分離が必要
- データ保持要件
  - データ毎に異なる保持ポリシー(保持期間)が必要
    → 保持ポリシーはテーブル単位に設定するため、テーブルで分離が必要
- データの関係性
  - データ取得/分析時に関連するデータ
    → 同一テーブルに格納することを推奨
    （またはアプリケーションサイドで結合を実施）

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

Appendix

利用開始事例

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

お客様のユースケース

Guardian Life Insurance Company は、フォーチュン250の相互会社であり、人生、障害、歯科、その他の個人給付の リーディング プロバイダーです。

お客様は、ビルディングシステムやアーティファクトリポジトリからメトリクスを収集して処理するアプリケーションを構築しています。このデータをセルフホスト時系列データベースに格納していました。
Amazon Timestream を利用し、サーバーレス、自動スケーリング、データライフサイクル管理機能に感銘を受けました。
Grafana を使用して Amazon Timestream に格納されている時系列データを視覚化できることを嬉しく感じています。

©, 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Appendix

お客様のユースケース

Appendix

オートデスク は、建築、エンジニアリング/建設、製品設計、製造業界のお客様向けのソフトウェアのグローバルリーダーです。

Amazon Timestream は、クラウドでホストされたスケーラブルな時系列データベースを提供することで、新しいワークフローを提供できる可能性を秘めています。時系列データにより、製品の性能が向上し、製造の無駄が削減されます。Timestream が提供する主な差別化要因は、オートデスク様のデータ管理の負担軽減です。

©, 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

お客様のユースケース

Appendix

Trimble Inc. は、 建設、農業、地理空間、輸送業界向けの生産性ソリューションを提供する大手技術プロバイダーです。

彼らは、IoT モニタリングソリューションをサポートするために、サーバーレスの時系列データベースとして Amazon Timestream を使用しています。
Timestreamは、IOTで生成された時系列データのために特別に設計されており、既存の監視システムによる管理オーバーヘッドの削減、パフォーマンスの向上、コストの削減を実現します。

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

お客様のユースケース

Appendix

D2L は、K-12、高等教育、医療、政府、企業部門の顧客が使用する業界をリードする世界クラスの学習プラットフォームを備えた教育技術のグローバルリーダーです。

現在、リレーショナルデータベースにデータを格納する内部モニタリングツールに Amazon Timestream を使用しています。完全に管理された時系列データベースであるTimestreamに切り替えることで、コストを **80% 以上削減しながら、パフォーマンスを維持できます。**

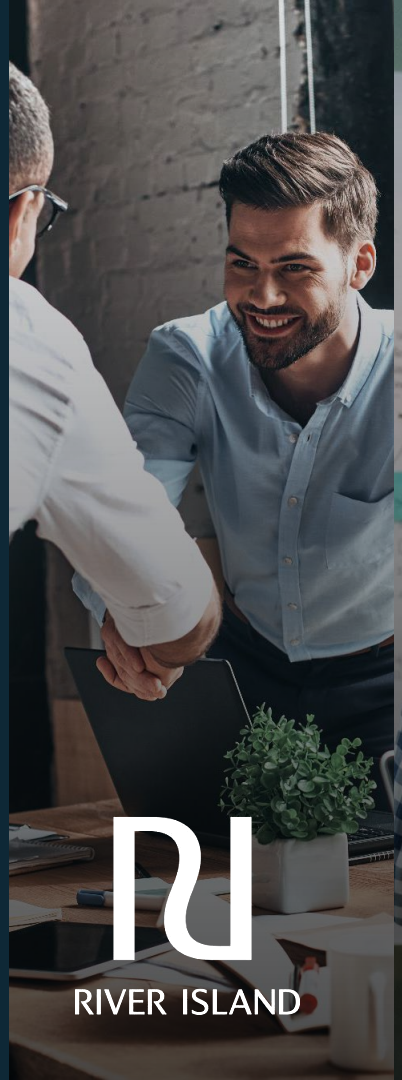
© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

お客様のユースケース

Appendix

River Island は、60年以上のファッション小売業経験を持つRiver Islandは、ヨーロッパ、アジア、中東 **の350** 以上の店舗と、6つの専用オンラインサイトを持つ最も有名で愛されているブランドの1つです。

彼らは、シンプルで、簡単で、手頃な価格の時系列データストアを見つけるのに苦労しました。
Timestream では、彼らはより多くのことを得ました。
Timestream は、AWS がホストするマイクロサービスだけでなく、すべての遺産システム全体にわたって集中モニタリング機能を構築する必要性を実現しました。



© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Appendix

Timestream パートナー

Analytics

システムインテグレータ

© 2020 Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws