このコンテンツは公開から3年以上経過しており内容が古い可能性があります
最新情報については[サービス別資料]もしくはサービスのドキュメントをご確認ください

[AWS Black Belt Online Seminar] Amazon Athena

サービスカットシリーズ

Solutions Architect, Analytics
Makoto Kawamura
2020/06/17

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

過去資料
https://amzn.to/JPArchive

自己紹介

川村　誠

Hadoop/Spark スペシャリスト
ソリューションアーキテクト

- データ分析系サービスを担当
- 好きなサービス
  - Amazon EMR
  - Amazon Athena
  - AWS Glue
  - Amazon SageMaker

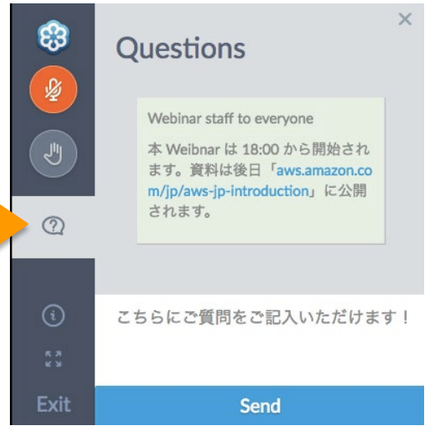
AWS Black Belt Online Seminar とは

「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

内容についての注意点

- 本資料では 2020 年 6 月 17 日時点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。

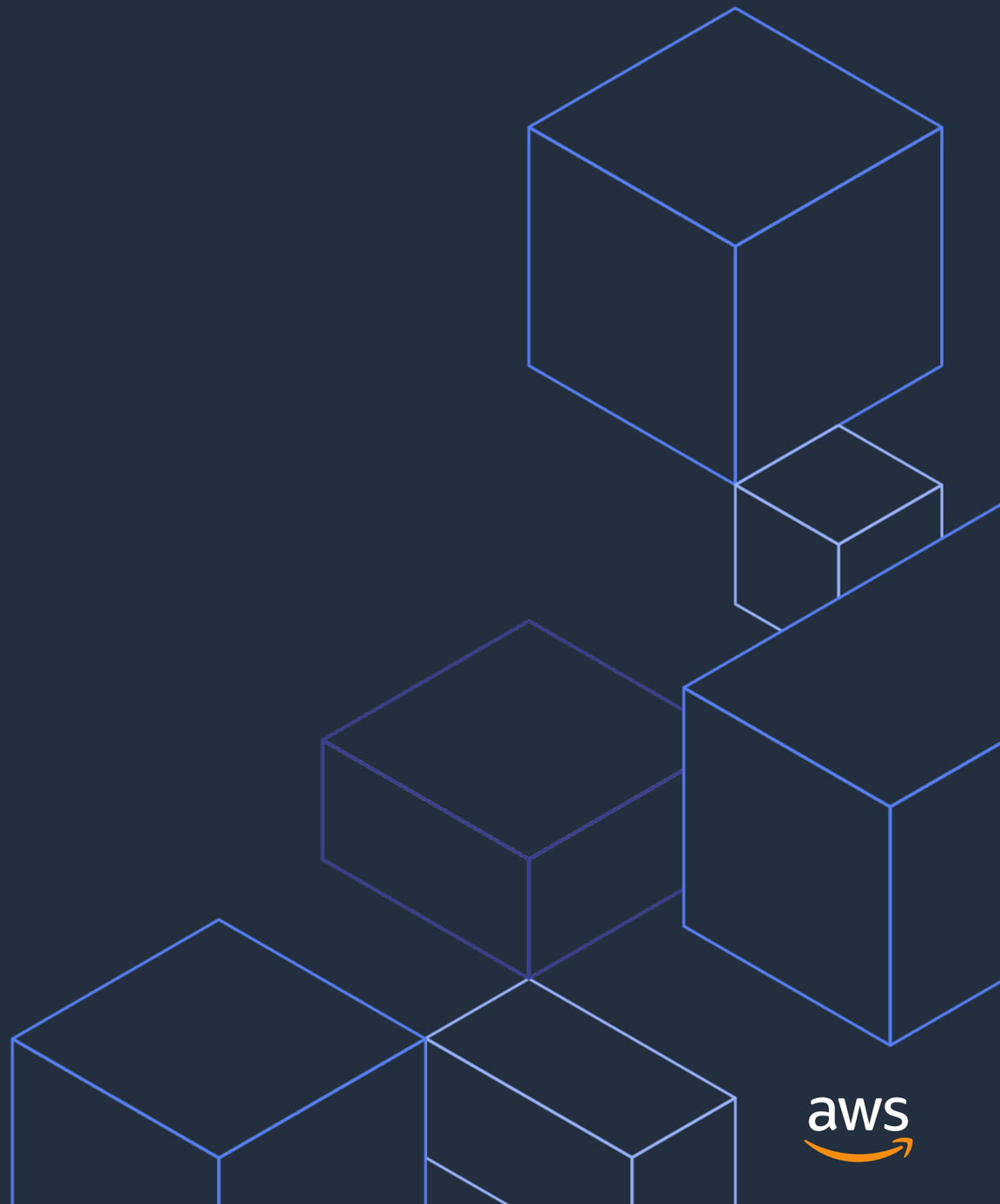
- 価格は税抜表記となっています。日本居住者のお客様には別途消費税をご請求させていただきます。

- 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.

アジェンダ

- Amazon Athena 概要
- Athena の使い方
- Athena のさまざまなクエリ
- パフォーマンスチューニング
- Athena の運用管理
- Athena のセキュリティ
- サービス上限と料金

Athena 概要



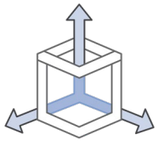
Amazon Athena とは

Amazon S3 にあるデータ（およびさまざまなデータソース*）に対して
標準 SQL を使用して簡単に分析可能とする
インタラクティブなクエリサービス

* Federated Query によるサポート

Amazon Athena の特徴

サーバレスでインフラ管理不要



大規模データに対しても高速なクエリ

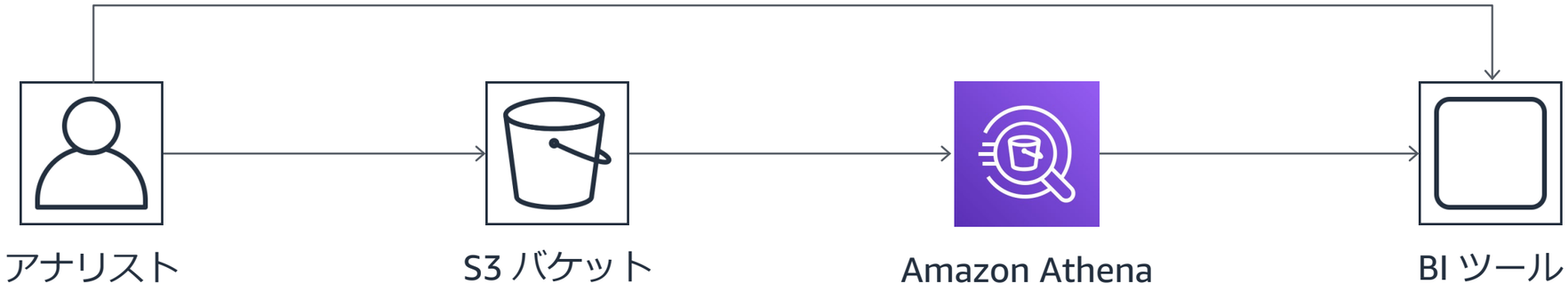
事前のデータロードなしに Amazon S3 に直接クエリ

スキャンしたデータに対しての従量課金

JDBC / ODBC / API 経由で BI ツールやシステムと連携

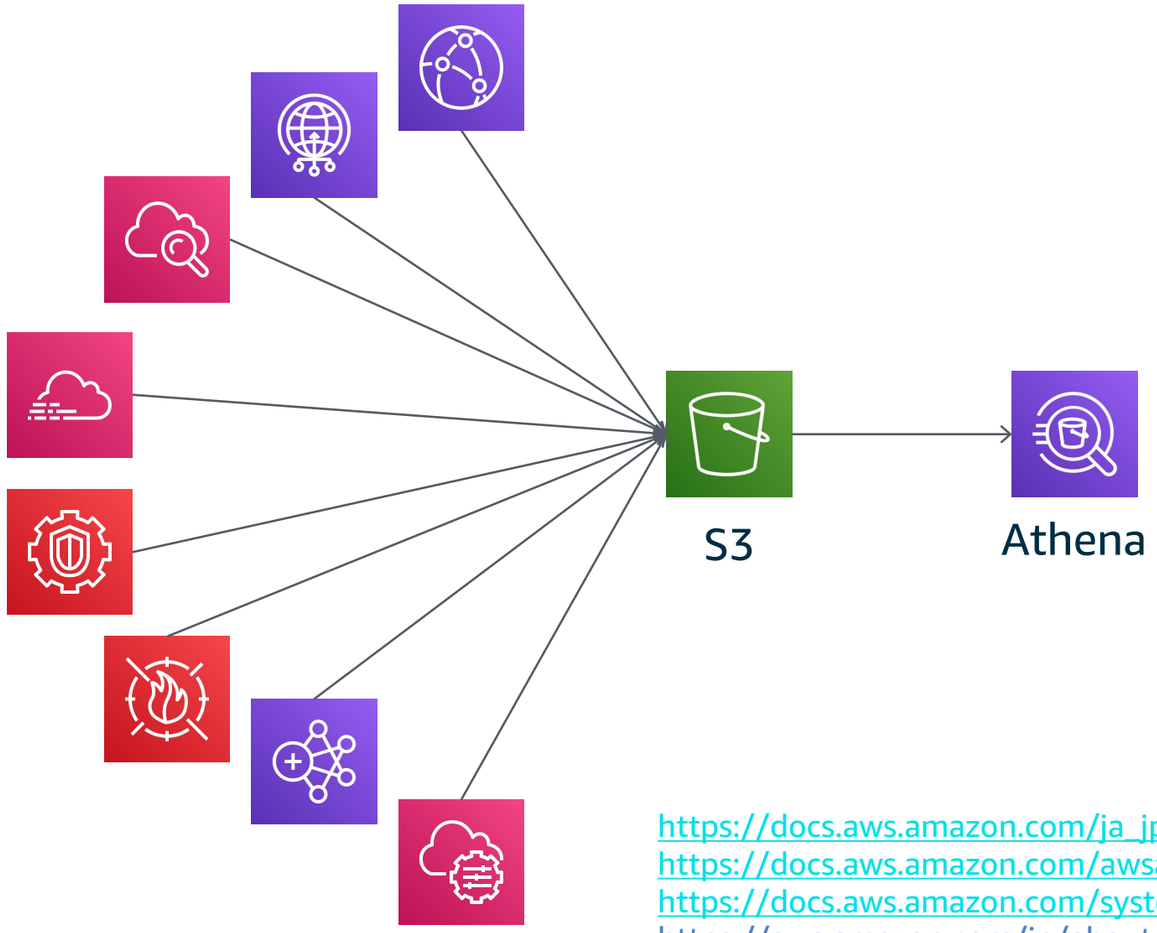
ユースケース 1: アナリストによるアドホックな分析

S3 にデータをアップロードしたら、テーブル定義をしてすぐにクエリを実行
さらに BI ツールから可視化



ユースケース 2: ログ分析

S3 に出力される様々な AWS サービスのログや収集データに対してクエリ

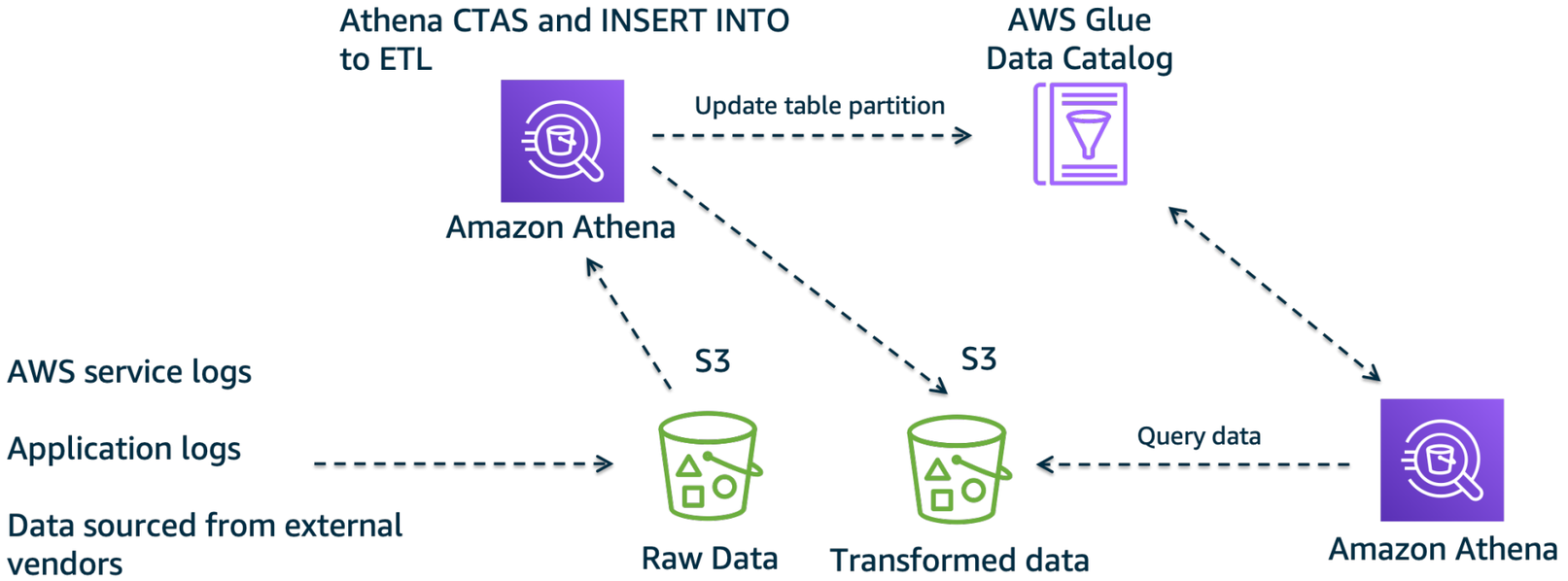


- Elastic Load Balancing (ALB/CLB/NLB) ログ
- Amazon CloudFront ログ
- AWS CloudTrail ログ
- Amazon EMR ログ
- AWS Global Accelerator ログ
- Amazon GuardDuty ログ
- Amazon VPC フローログ
- AWS WAF ログ
- AWS Cost and Usage Reports データ
- AWS Systems Manager インベントリデータ
- Amazon S3リクエスタ支払いバケットデータ

https://docs.aws.amazon.com/ja_jp/athena/latest/ug/querying-AWS-service-logs.html
https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/athena.html
https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-inventory-query.html
https://aws.amazon.com/jp/about-aws/whats-new/2019/08/amazon-athena-supports-querying-data-from-amazon-s3/

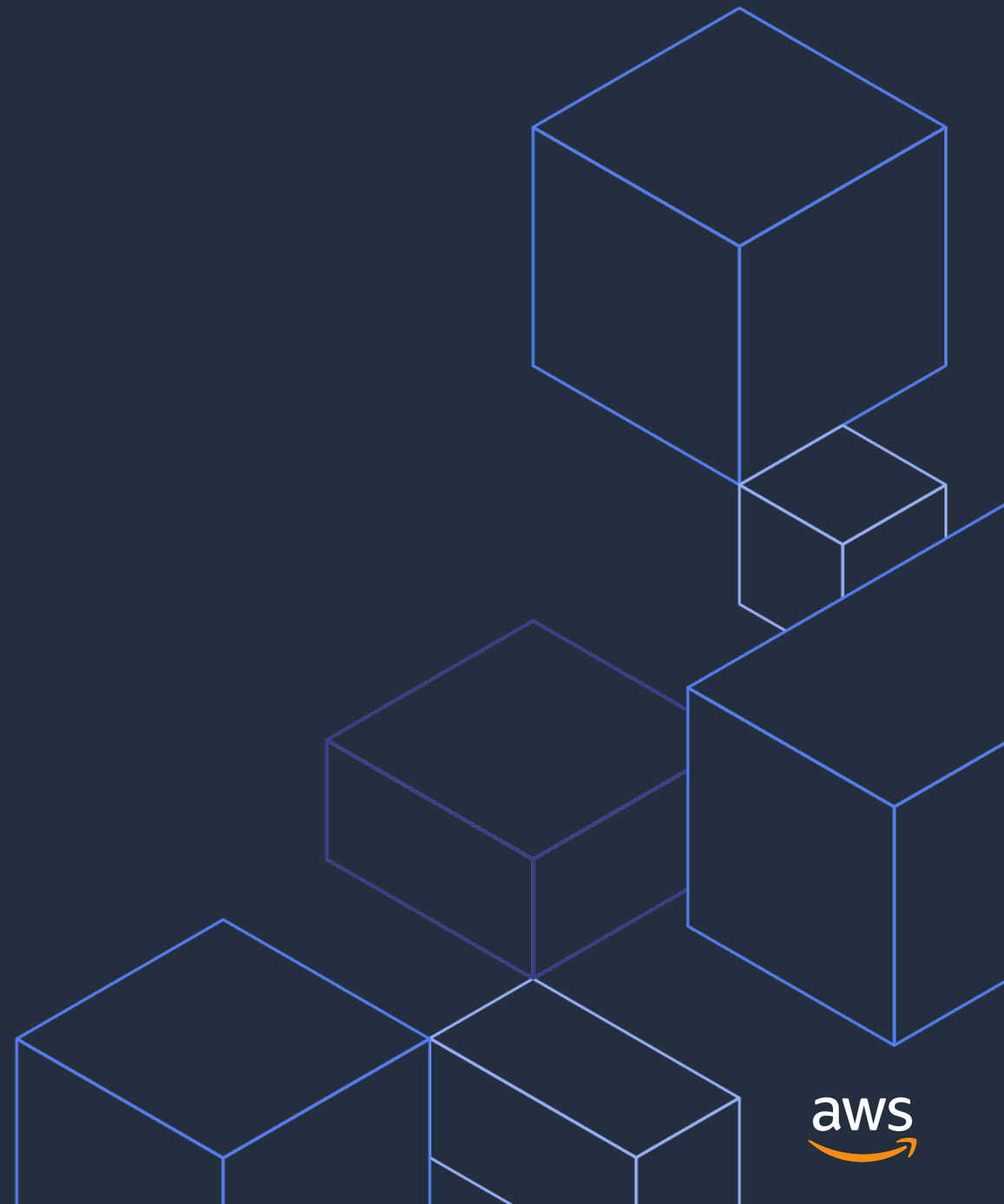
ETL パイプラインを Athena で構築

連携先サービスから定期的に送られてくるデータに対して、INSERT INTO 句を用いて継続的なデータ変換を行い、効率的にクエリ

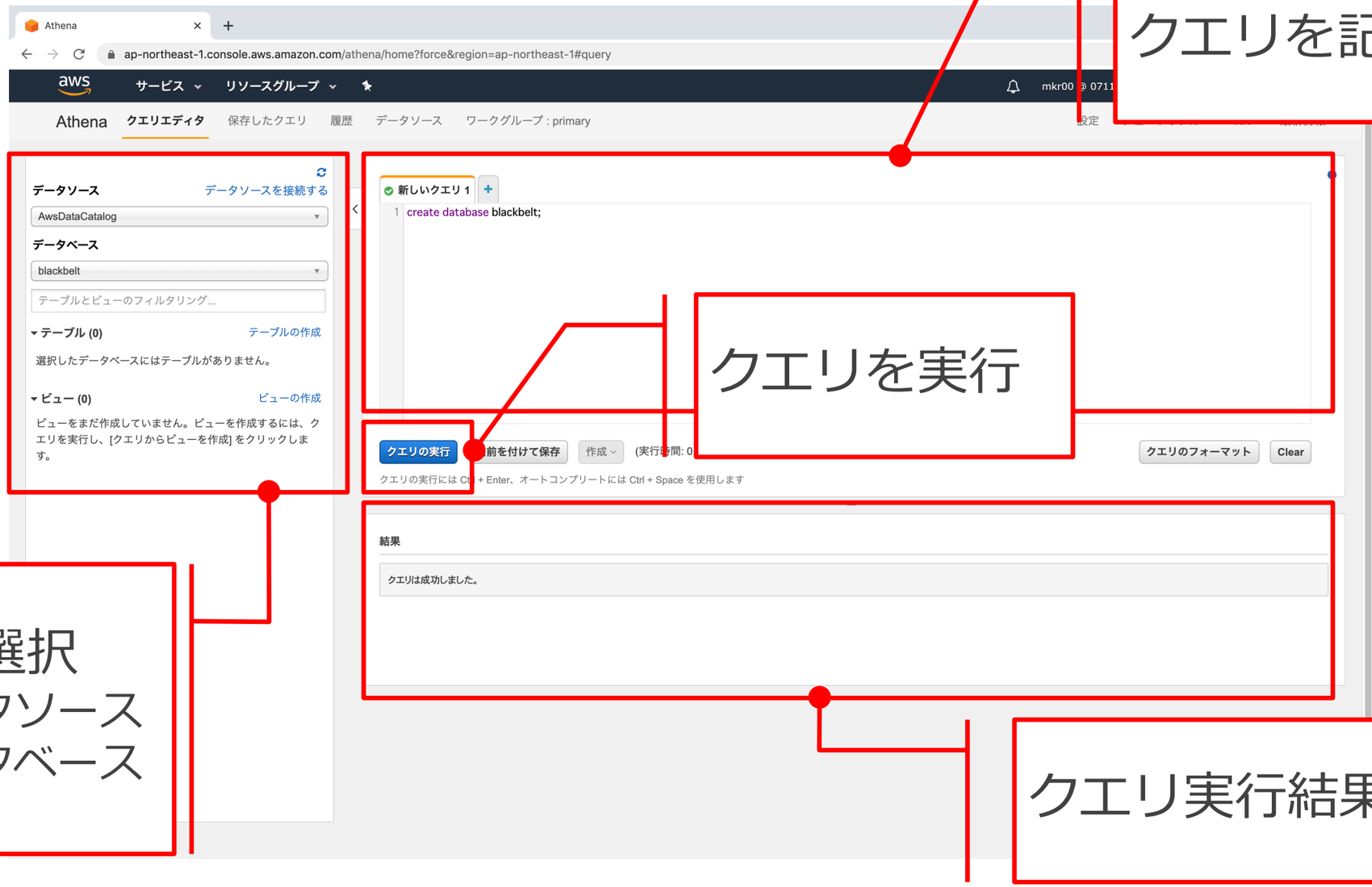


Athena の使い方

1. データを S3 に保存する
2. テーブルを定義する
3. クエリを実行する



Athena コンソール



クエリを記述

クエリを実行

以下を選択
- データソース
- データベース

クエリ実行結果を参照

1. データを Amazon S3 に保存する

Amazon S3 にデータを保存する方法

AWS Database Migration Service

Amazon Kinesis Data Firehose

Amazon Kinesis Data Streams

Amazon S3

AWS Storage Gateway

AWS Snowball

AWS Snowball Edge

AWS Snowmobile

AWS DataSync

AWS Transfer Family

2. テーブルを定義する

テーブルを定義する

- Athena ではクエリのためにテーブル定義が必要．デフォルトで AWS Glue Data Catalog 上のテーブル定義を使用

- AWS Glue Data Catalog は、Apache Hive Metastore という OSS と互換性のある、メタデータを管理するためのリポジトリ

- AWS Glue Data Catalog にテーブル定義を作成する方法は次の 3 つ
    - Athena DDL
    - AWS Glue Catalog API
    - AWS Glue Crawler

AWS Glue Data Catalog で定義されたテーブルの例

名前	sample_test_out01
説明	
データベース	sample-test
分類	parquet
場所	s3://test-sample-glue-bucket-ijgowerfvwigoev542637/test_out/out01/
接続	
廃止	いいえ
最終更新日	Thu Jun 27 10:35:27 GMT+900 2019
入力形式	org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat
出力形式	org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat
Serde シリアル化ライブラリ	org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe
Serde パラメータ	serialization.format 1
テーブルのプロパティ	sizeKey 18183 objectCount 63 UPDATED_BY_CRAWLER sample-test-data
	CrawlerSchemaSerializerVersion 1.0 recordCount 30 averageRecordSize 143
	CrawlerSchemaDeserializerVersion 1.0 compressionType none typeOfData file

スキーマ

表示中: 1 - 7 of 7

	列名	データ型	パーティションキー	Comment
1	deviceid	string		
2	platform	string		
3	email	string		
4	location	string	Partition (0)	
5	year	string	Partition (1)	
6	month	string	Partition (2)	
7	day	string	Partition (3)	

Athena DDL

- Athena の DDL は HiveQL 形式で記述
- 標準的なテーブル定義ステートメントの後に、パーティション定義、データ形式、データの場所、圧縮形式などを指定

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

Amazon Athena のデータ形式

データ形式	SerDe
CSV	通常は LazySimpleSerDe を、引用符付きカラムが含まれる場合は OpenCSVSerDe を使用
TSV	LazuSimpleSerDe を使用して ‘¥t’ を区切り文字に指定
カスタム区切り	LazuSimpleSerDe を使用して任意の区切り文字を指定
JSON	HiveJSONSerDe または OpenXJsonSerDe を指定
Apache Avro	AvroSerDe を指定
ORC	ORCSerDe を指定
Apache Parquet	ParquetSerDe を指定
Logstash ログ	Grok SerDe を指定
Apache ログ	RegexSerDe を指定
CloudTrail ログ	基本的に CloudTrailSerDe を指定．一部では OpenXJSONSerDe を指定

https://docs.aws.amazon.com/ja_jp/athena/latest/ug/supported-format.html

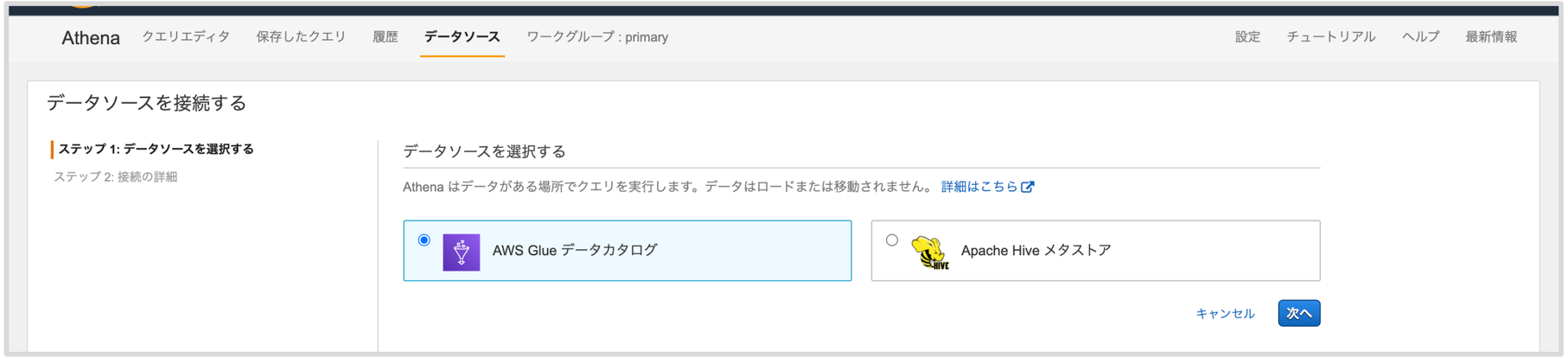
データソース

- Athena では、データを記述するデータカタログ、およびそこに含まれるデータをあわせて**データソース**と呼ぶ。デフォルトのデータソースは AWS Glue Data Catalog

- AWS Glue Data Catalog に登録されたデータのうち、操作しているユーザーが参照権限を持っているもののみが、Athena 上で表示される

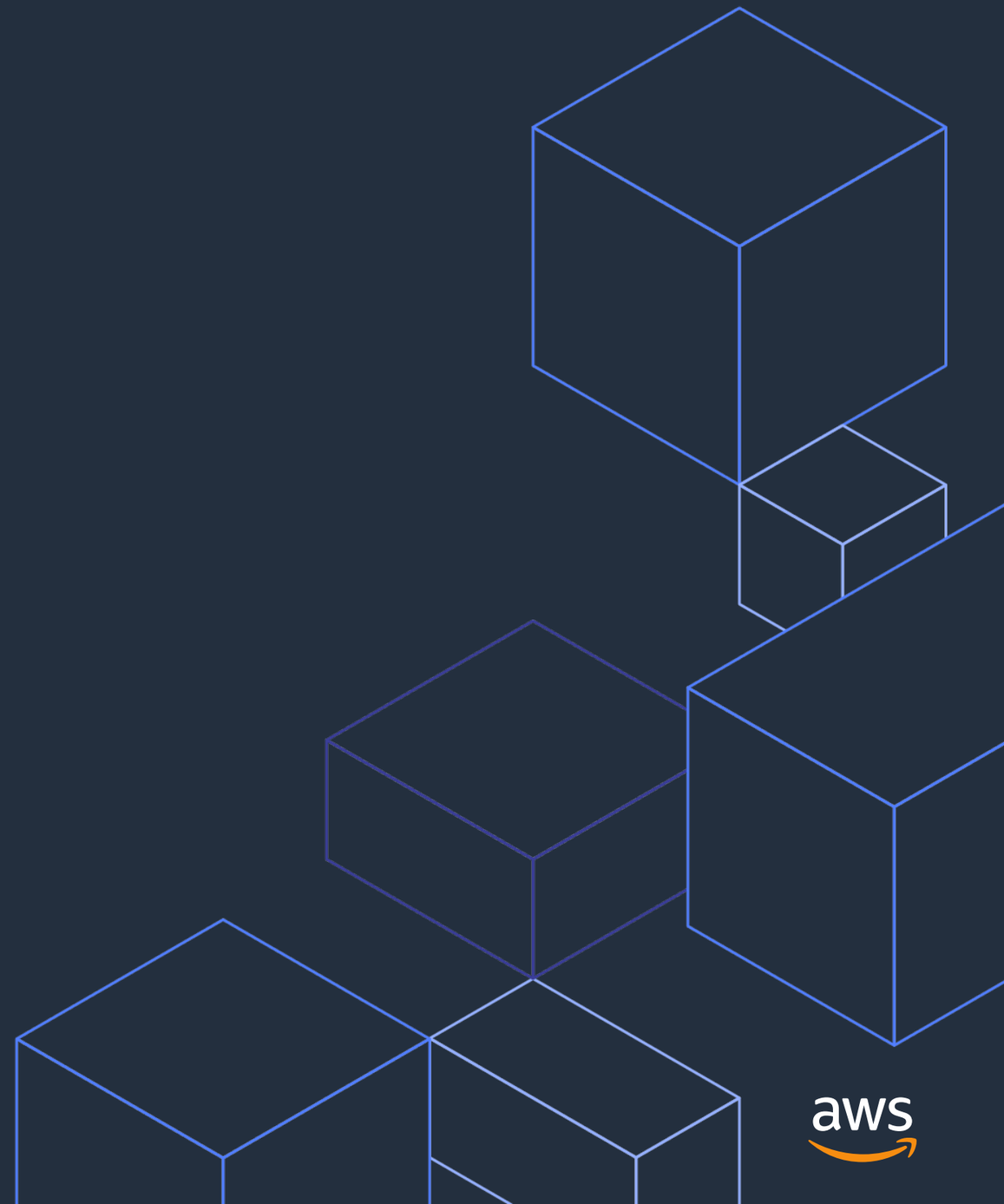
データソースの追加

- AWS Glue Data Catalog 以外のデータソースを、新たに追加することができる
- すでに既存の仕組みがあり、自分たちの Hive Metastore を持っている場合には、これを Athena に登録することで、Athena でも利用可能になる
    - 既存の Hive Metastore 資産を移行する必要がなくなる
- 外部メタストアへの接続には、Lambda 関数として実行されるコネクター (Athena Data Connector for External Hive Metastore) を利用する



https://docs.aws.amazon.com/athena/latest/ug/connect-to-data-source-hive.html

3. クエリを実行する



Amazon 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
- 基本的には Presto 0.172 に準拠
    - 一部サポートしていない機能等の詳細は以下ドキュメント参照
      https://docs.aws.amazon.com/ja_jp/athena/latest/ug/other-notable-limitations.html

```
[ 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 におけるクエリ実行の基礎

```
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
```

- 標準 SQL を利用可能
- 行単位でのトランザクションではなく、あくまでデータ抽出や集計がメインの用途
- 典型的なクエリパターン
    - SELECT でデータを取得
    - COUNT(), SUM() 等で集計
    - JOIN で複数テーブルを結合
    - WHERE でデータのフィルタ
    - GROUP BY で集約

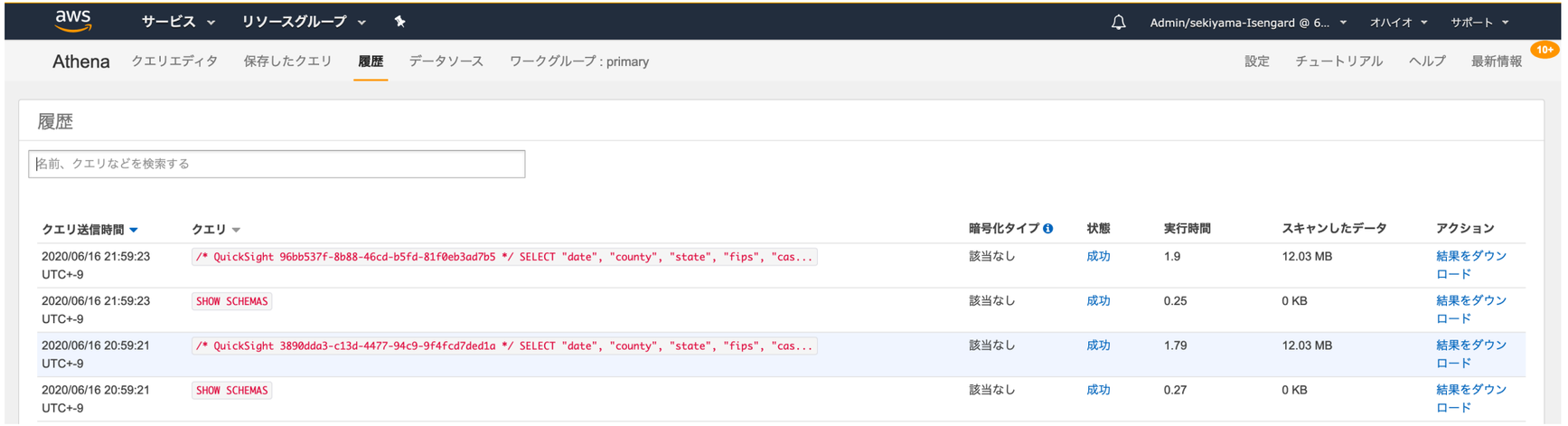
クエリ結果

- 実行される各クエリのクエリ結果とメタデータ情報を、指定した **S3 バケット**に自動的に保存
    - この保存自体をオフにすることはできない
    - 必要に応じて、この場所にあるファイルにアクセスして操作可能
    - Athena コンソール履歴画面から、クエリ結果ファイルを直接ダウンロードすることも可能

- クエリ出力ファイルへのアクセスには以下の権限が必要
    - クエリ結果の場所の Amazon S3 GetObject アクション
    - Athena GetQueryResults アクション

クエリ履歴

- Athena コンソールの履歴画面では下記情報を確認可能．結果は 45 日間保存
  - クエリ送信時刻
  - 送信クエリ
  - 暗号化タイプ
  - クエリの状態(成功/失敗)
  - 実行時間
  - スキャンしたデータ量
  - 成功したクエリの結果ファイルダウンロード
  - 失敗したクエリのエラー情報詳細表示



コンソール以外からの実行

JDBC / ODBC ドライバー経由

- JDBC: バージョン 2.0.9: JDBC API 4.1 および 4.2 のデータ基準に準拠
- ODBC: Windows、Linux、および OSX で使用可能

Athena API 経由

- AWS SDK 経由で Athena API を利用可能
- クエリ実行 API: 非同期処理 (JDBC 接続と異なる)
    1. **StartQueryExecution** でクエリを実行して **QueryExecutionId** を取得
    2. **GetQueryExecution** で実行状況を確認
    3. **State** が **SUCCEEDED** になったら **GetQueryResults** で結果を取得

https://docs.aws.amazon.com/ja_jp/athena/latest/ug/athena-bi-tools-jdbc-odbc.html
https://docs.aws.amazon.com/athena/latest/APIReference/Welcome.html

Athena のクエリパフォーマンスの基本

スキャンデータ量を減らす

クエリ速度を向上するには、クエリ結果を集計するために必要なデータのスキャン量を減らすのことが効果的

- 列指向フォーマットの利用
- データの圧縮
- データのパーティショニング(ファイルの配置方法の最適化)

スキャンするファイルを一定サイズ以上にまとめる

小さいファイルが大量にある場合、ファイル操作のオーバーヘッドが大きいため、128MB 以上のかたまりにまとめる

スキャンするファイルを分割可能な形式とする

Athena は複数のワーカーが並列で処理を行うため、ファイルを分割可能な形式として、複数のワーカーで並列処理できるようにする

Athena のさまざまなクエリ

CTAS (Create Table As Select)

クエリ結果をもとに、新しいテーブルを作成．ETL 処理を Athena で実行可能

- データ / 圧縮フォーマット変換
- データを複雑に加工整形するための多段クエリ

```
CREATE TABLE new_table WITH (
    format = 'PARQUET',
    external_location = 's3://my_athena_results/new_table/'
    partitioned_by = ARRAY['year', 'month', 'day'],
    bucketed_by = ARRAY['user_id', 'category'],
    bucket_count = 3,
)
AS
SELECT * FROM old_table;
```

CTAS のパーティショニングとバケッティング

パーティショニング

- 通常のパーティションと一緒、 ARRAY[‘key1’, ‘key2’, ‘key3’] で指定した順番に s3://MYBUCKET/key1=xxx/key2=yyy/key3=zzz という形で保存
- CTAS 句で作成可能なパーティション数は最大 **100** で、超えるとエラーになる

バケッティング

- 指定したカラム群を元に、出力のファイル数に振りわける
- **カラム内の同じキーは同一ファイルに入る**ため、カラムのカーディナリティに注意

使い分け

- パーティショニングは、WHERE 句で絞る対象のカラムを指定
- バケッティングは、個々のファイルサイズを調整（128-512MB 程度）
- 両方同時に使うことが可能（組み合わせた場合、パーティション内のファイル数が、バケッティングで指定した数で分割される）

INSERT INTO

クエリ結果をもとに、既存のテーブルにデータを追加

- SELECT クエリの結果を、新しい行として対象テーブルに挿入する形
- データは挿入先テーブルの形式に変換されるため、データ形式変換用のクエリとして INSERT INTO を利用可能
- INSERT INTO のたびに新しいファイルが生成されるため，少量データでの実行を繰り返すことで、大量の小さなファイルが生成され，クエリパフォーマンスに影響

```
INSERT INTO destination_table
SELECT select_query FROM source_table_or_view


INSERT INTO destination_table [(col1,col2,...)]
VALUES (col1value,col2value,...)[,
        (col1value,col2value,...)][,
        ...]
```


https://docs.aws.amazon.com/ja_jp/athena/latest/ug/insert-into.html

INSERT INTO によるパーティション追加

パーティション追加の例

1. CREATE EXTERNAL TABLE で、s3://MY_BUCKET/service_a/ にパーティションを設定したテーブル **table_service_a** を作成
2. 日次受信データ用のテーブル **tmp_table_service_a** を作成
3. INSERT INTO で **tmp_table_service_a** のデータを(処理した上で) **table_service_a** に追加
4. **tmp_table_service_a** を DELETE TABLE で削除
5. 2-4 を日次で実行

制約事項

- INSERT INTO できるパーティションの最大数は **100**
- バケット化されたテーブルは未サポート
- サポートされるデータ形式は Avro/JSON/ORC/Parquet/テキストファイル
  (対応する Serde の詳細はドキュメントを参照)

https://docs.aws.amazon.com/ja_jp/athena/latest/ug/insert-into.html

VIEW

クエリ内容自体を仮想的なテーブルとして登録

- VIEW を実行すると、裏側で元テーブルに対してクエリを実行
- VIEW からさらに VIEW を作成することも可能
- AWS Glue Data Catalog 側からは、テーブルとして認識される

```
CREATE VIEW my_view AS
SELECT
    col1,
    col2 / 100
FROM table1
INNER JOIN table2
ON table1.id = table2.id;
```

VIEW の活用シーン

- **データのサブセット**をクエリする。元のテーブルから列のサブセットがあるテーブルを作成して、データのクエリを単純化する
- **複数のテーブルを 1 つのクエリに結合する**。複数のテーブルを持ち、それらを UNION ALLと組み合わせる場合は、その式でビューを作成して、結合テーブルに対するクエリを簡素化できる
- 既存の基本クエリの複雑性を解消して、**ユーザーによるクエリの実行を単純化する**
- View をクエリ問い合わせのインタフェースとして利用し、より最適なクエリへの移行やクエリの仕様変更に対する**メンテナンスの影響を最小限に抑える**

地理空間データの分析

ESRI Java Geometry Library をサポート

- ジオメトリデータ型として point,line,multiline 等に対応
- ジオメトリデータ間の関係として distance,equals,overlaps 等に対応

具体的には、以下のような分析を行うことが可能

- 2 点間の距離を計算
- あるエリアに他のエリアが含まれているか否かをチェック

以下の例では、各地域内で起こった地震の回数を ST_CONTAINS, ST_POINT 関数で求めている

```
SELECT counties.name, COUNT(*) cnt
FROM counties
CROSS JOIN earthquakes
WHERE ST_CONTAINS (
        counties.boundaryshape,
        ST_POINT(earthquakes.longitude, earthquakes.latitude))
GROUP BY counties.name
ORDER BY cnt DESC
```

https://docs.aws.amazon.com/ja_jp/athena/latest/ug/geospatial-query-what-is.html

SELECT $path / WHERE $path

- 予期しないデータの調査や、ソースデータに関する情報を必要とする場合に利用可能なクエリ
- SELECT 句や WHERE 句の中で利用可能

SELECT 句での利用

クエリ

```
SELECT "$path" FROM "my_database"."my_table" WHERE year=2019;
```

結果

```
s3://awsexamplebucket/datasets_mytable/year=2019/data_file1.json
```

WHERE 句での利用

クエリ

```
SELECT *,"$path" FROM "my_database"."my_table" WHERE "$path" LIKE
's3://awsexamplebucket/my_table/my_partition/file-01.csv'
```

結果

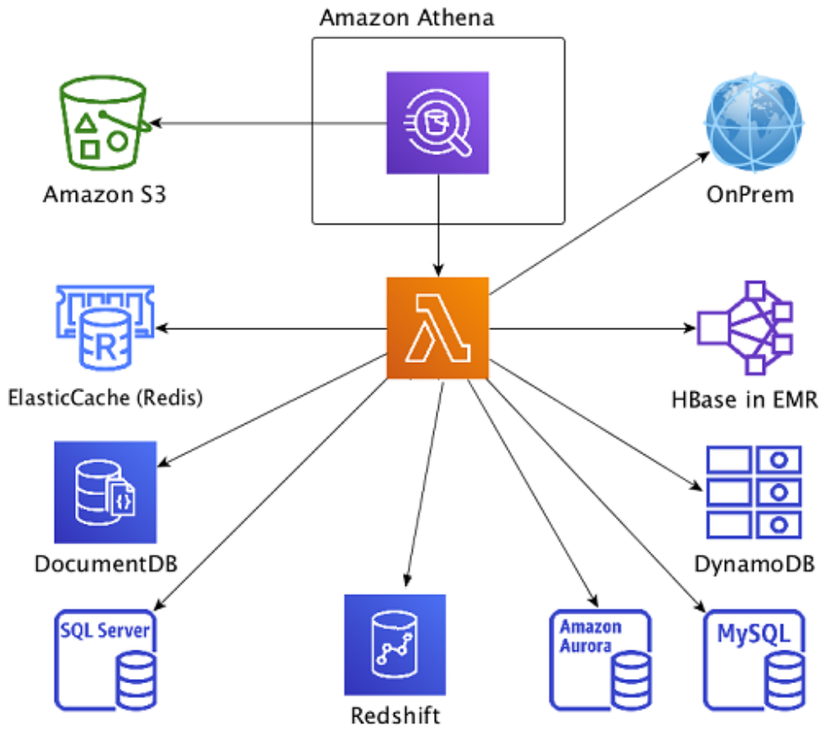
```
id name year $path
3 John 1999 's3://awsexamplebucket/my_table/my_partition/file-01.csv'
4 Jane 2000 's3://awsexamplebucket/my_table/my_partition/file-01.csv'
```

https://aws.amazon.com/jp/premiumsupport/knowledge-center/find-s3-source-file-athena-table-row/

Amazon Athena Federated Query

Preview

- 様々なデータソースに対して SQL クエリを実行可能
- AWS Lambda で動作するコネクタを利用して実行
- 標準のコネクタとしてさまざまなデータソースが対応済
  - Amazon DynamoDB
  - Amazon Redshift
  - Apache HBase
  - MySQL
  - PostgreSQL など
- Athena Query Federation SDK を利用して、独自コネクタを実装可能



対応リージョン: バージニア、ムンバイ、アイルランド、オレゴン

https://github.com/awslabs/aws-athena-query-federation
https://aws.amazon.com/jp/blogs/news/query-any-data-source-with-amazon-athenas-new-federated-query/

UDF (User Defined Function)

Preview

- ユーザ独自のスカラー関数を UDF として定義して、SQL クエリで呼び出すことが可能
- データの圧縮や解凍、機密データの編集、カスタマイズされた復号の適用など独自の処理を実行可能
- Athena Query Federation SDK を利用して Java で実装し、Lambda 関数として Athena から呼び出される

```
USING FUNCTION compress(col1 VARCHAR)
      RETURNS VARCHAR TYPE
       LAMBDA_INVOKE WITH (lambda_name = 'MyAthenaUDFLambda')
SELECT
     compress('StringToBeCompressed');
```

- 参照される Lambda 関数内の Java メソッドに対応した UDF 名を指定
- 名前付き変数を複数指定可能

対応リージョン: バージニア、ムンバイ、アイルランド、オレゴン

https://docs.aws.amazon.com/athena/latest/ug/querying-udf.html

Machine Learning (ML) with Amazon Athena

Preview

- Athena SQL クエリで SageMaker ML モデルを呼び出し、推論を実行可能
- 異常検出やコホート分析、販売予測などの複雑な作業が SQL クエリで関数を呼び出す感覚で利用できる

ML関数を定義、名前付き変数を複数指定可能

```
USING FUNCTION predict_customer_registration(age INTEGER)
      RETURNS DOUBLE TYPE
      SAGEMAKER_INVOKE_ENDPOINT WITH (sagemaker_endpoint = 'xgboost-2019-09-20-04-49-29-303')
SELECT predict_customer_registration(age) AS probability_of_enrolling, customer_id
      FROM "sampledb"."ml_test_dataset"
      WHERE predict_customer_registration(age) < 0.5;
```

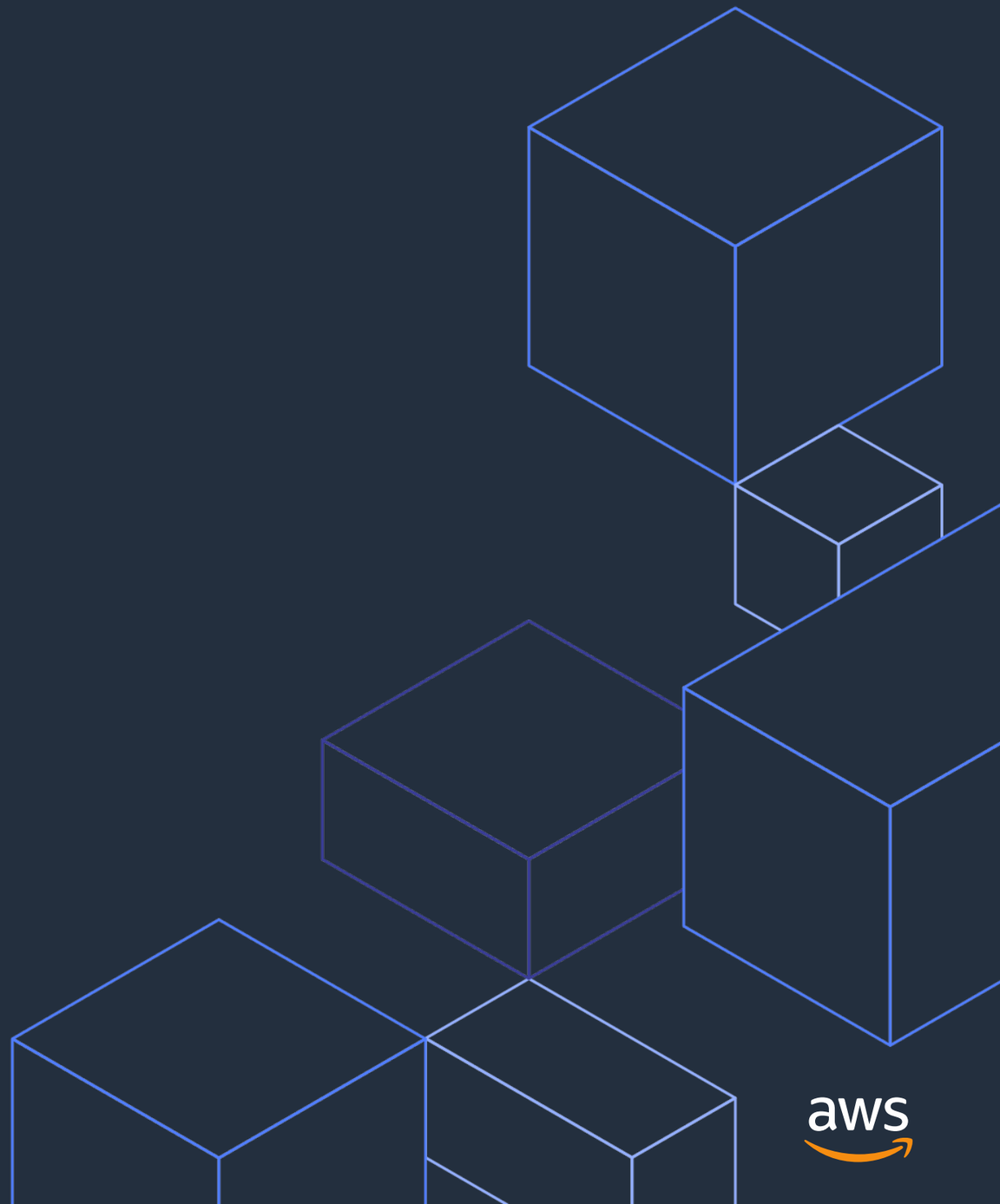
ML関数呼び出し

対応リージョン: バージニア、ムンバイ、アイルランド、オレゴン
https://docs.aws.amazon.com/ja_jp/athena/latest/ug/querying-mlmodel.html

パフォーマンスチューニング

- データの最適化
- データレイアウトの最適化
- クエリの最適化

データ最適化



列指向フォーマット

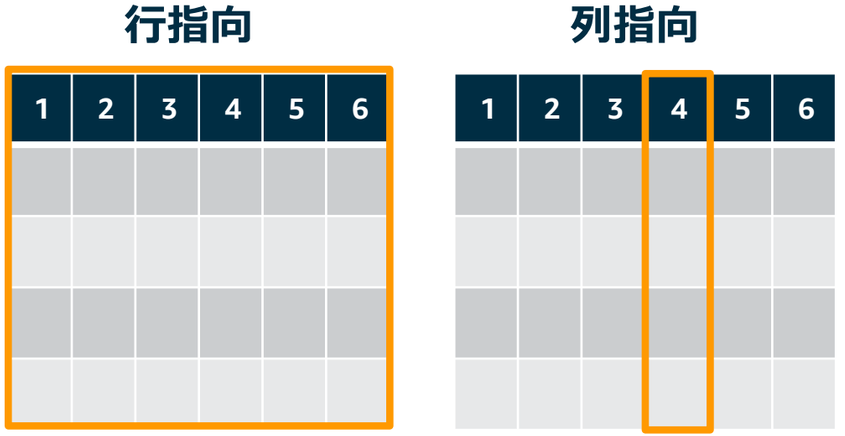
指向	特徴
列指向	• カラムごとにデータをまとめて保存 • 特定の列だけを扱う処理では、ファイル全体を読む必要がない • OLAP向き • ORC, Parquet など
行指向	• レコード単位でデータを保存 • 1カラムのみ必要でも、レコード全体を読み込む必要がある • OLTP向き • AVRO など

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

列指向フォーマットを使うメリット

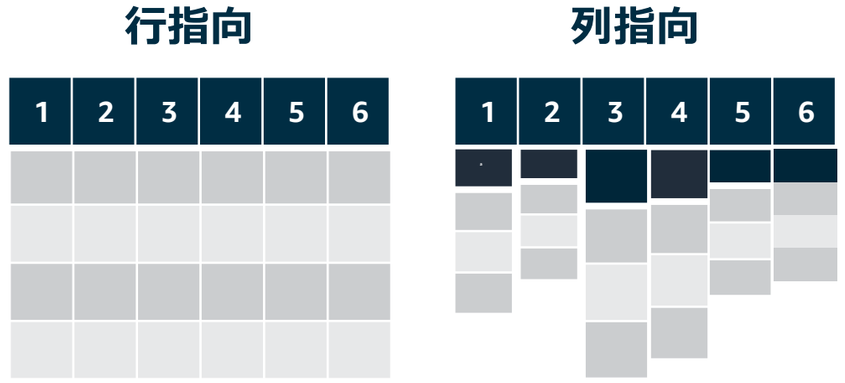
OLAP 系の分析クエリを効率的に実行できる

- 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



データ圧縮

- 最低限分割可能な圧縮形式を利用しておくと巨大なファイルがあったとしても、分散処理することが可能
- 分割不可の圧縮方式では、ファイル単位でしか分散処理できため、巨大なファイルは事前に分割しておく必要がある

圧縮形式	分割	説明
SNAPPY	可	Parquet フォーマットファイルのデフォルト圧縮形式
ZLIB	可	ORC フォーマットファイルのデフォルト圧縮形式
GZIP	不可	あらかじめデータを 1GB 未満のサイズに分割する必要あり
LZO	可	分割するためインデックスを付与する処理を行う必要あり
BZIP2	可	圧縮率と処理速度のバランスが良い圧縮形式

https://docs.aws.amazon.com/ja_jp/athena/latest/ug/compression-formats.html

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
- 分析クエリを実行する際に、使用するカラムだけを読み込むようになる

```
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
```

データレイアウトの最適化

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
- AWS Glue Data Catalog 使用時 1テーブルあたりの最大パーティション数は **10,000,000**

```
CREATE EXTERNAL TABLE IF NOT EXISTS action_log (
    user_id string,
    action_category string,
)
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.snappy.parquet

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 Amazon 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=06/day=30/

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、パーティション数実行する必要がある • 後からパーティションが増えた際、増えた分だけ ALTER TABLE ADD PARTITION

Partition Projection

Partition Projection を使用すると、非常に多くのパーティションがあるテーブルに対するクエリ処理を高速化し、パーティション管理を自動化することが可能

- パーティションプルーニングでは、メタデータを収集し、クエリに適用されるパーティションにだけに処理が実行されるので、多くの場合、クエリが高速化する
- しかしながら、テーブルに非常に多くのパーティションがある場合、AWS Glue Data Catalog に対するパーティション取得 API(GetPartitions) の呼び出しがクエリパフォーマンスに影響する可能性が生じる
- Partition Projection を利用するとパーティション情報を Athena 自身で計算できるようになるため、GetPartitions 呼び出しの影響を回避し、パーティションプルーニングによるクエリ高速化の恩恵を享受できるようになる

https://docs.aws.amazon.com/ja_jp/athena/latest/ug/partition-projection.html

Partition Projection

ユースケース

- 非常に多くのパーティションがあるテーブルに対するクエリが思ったほどすぐに完了しない場合
- データに新しいパーティションが作成されたとき、定期的にパーティションをテーブルに追加している場合
- 非常に多くのパーティション化されたデータが S3 に保存されていて、メタデータストアで管理するのが現実的でなく、クエリで参照する対象のデータがほんの一部の場合

Projection 可能なパーティション構造

Partition Projection はパーティションが予測可能なパターンに従う場合に利用する

Projection Type	パターン	例
整数	整数の連続シーケンス	[1, 2, 3, 4, ..., 1000] や [0500, 0550, ..., 2500] など
日付	日付/日時の 連続シーケンス	[20200101, 20200102, ..., 20201231]、 [1-1-2020 00:00:00, 1-1-2020 01:00:00, ..., 12-31-2020 23:00:00] など
列挙値	列挙値の有限セット	空港コードや AWS リージョンなど

https://docs.aws.amazon.com/ja_jp/athena/latest/ug/partition-projection.html

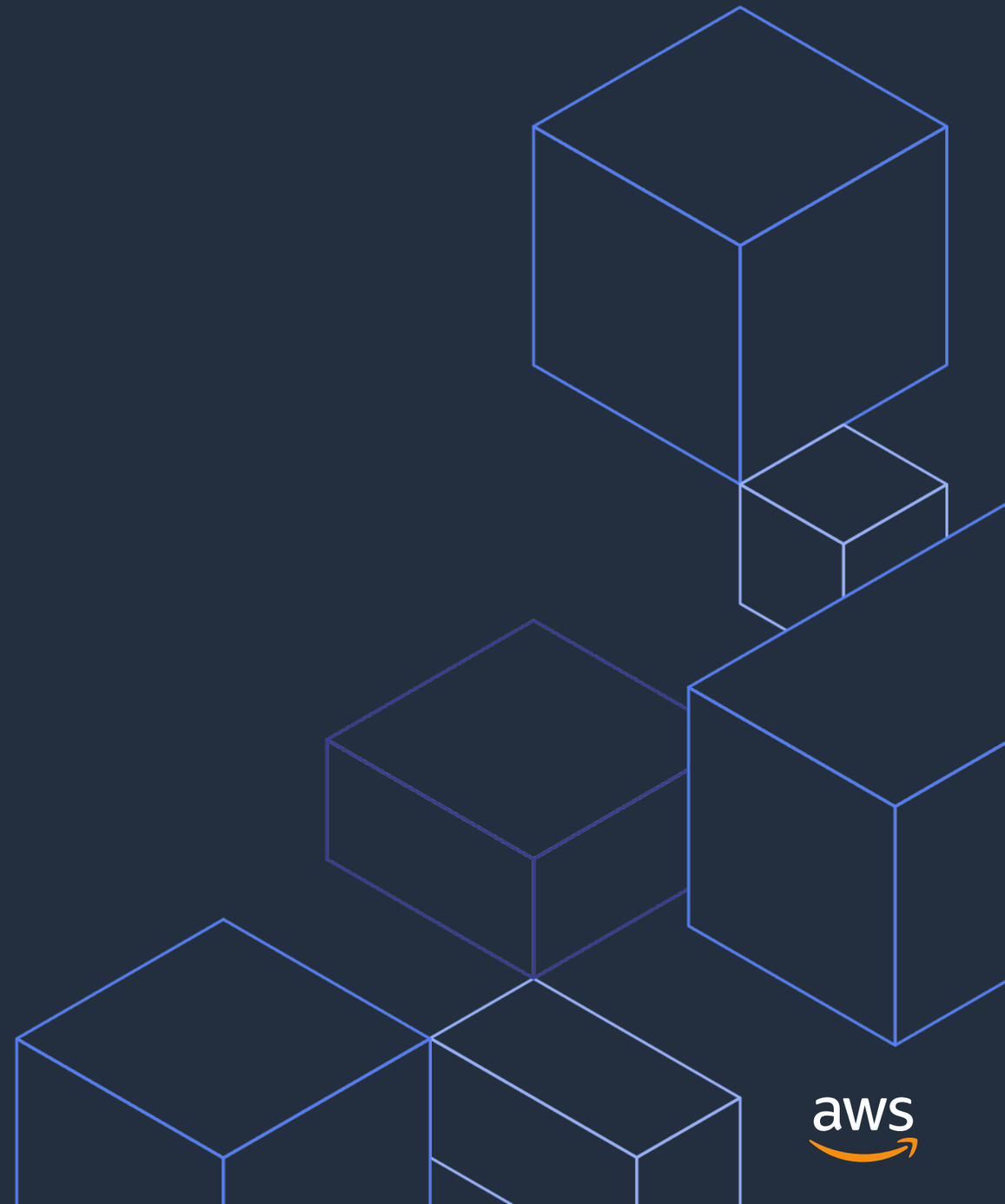
Partition Projection – 設定例

- テーブル作成時に Partition Projection を設定しておくと、設定に基づいてパーティションが存在する場所を Athena に通知することが可能
- 次の CREATE TABLE の設定例では、日付範囲の上限に **NOW** を使用することで、１時間毎に新しいデータが自動的にクエリ可能になる

```
...
LOCATION "s3://bucket/prefix/"
PARTITIONED BY (
  datehour STRING
)
TABLE PROPERTIES (
  "projection.enabled" = "true",
  "projection.datehour.type" = "date",
  "projection.datehour.range" = "2018/01/01/00,NOW",
  "projection.datehour.format" = ""yyyy/MM/dd/HH",
  "projection.datehour.interval" = "1",
  "projection.datehour.interval.unit" = "HOURS",
  "storage.location.template" = "s3://bucket/prefix/${datehour}”
```

https://docs.aws.amazon.com/ja_jp/athena/latest/ug/partition-projection-kinesis-firehose-example.html

クエリの最適化

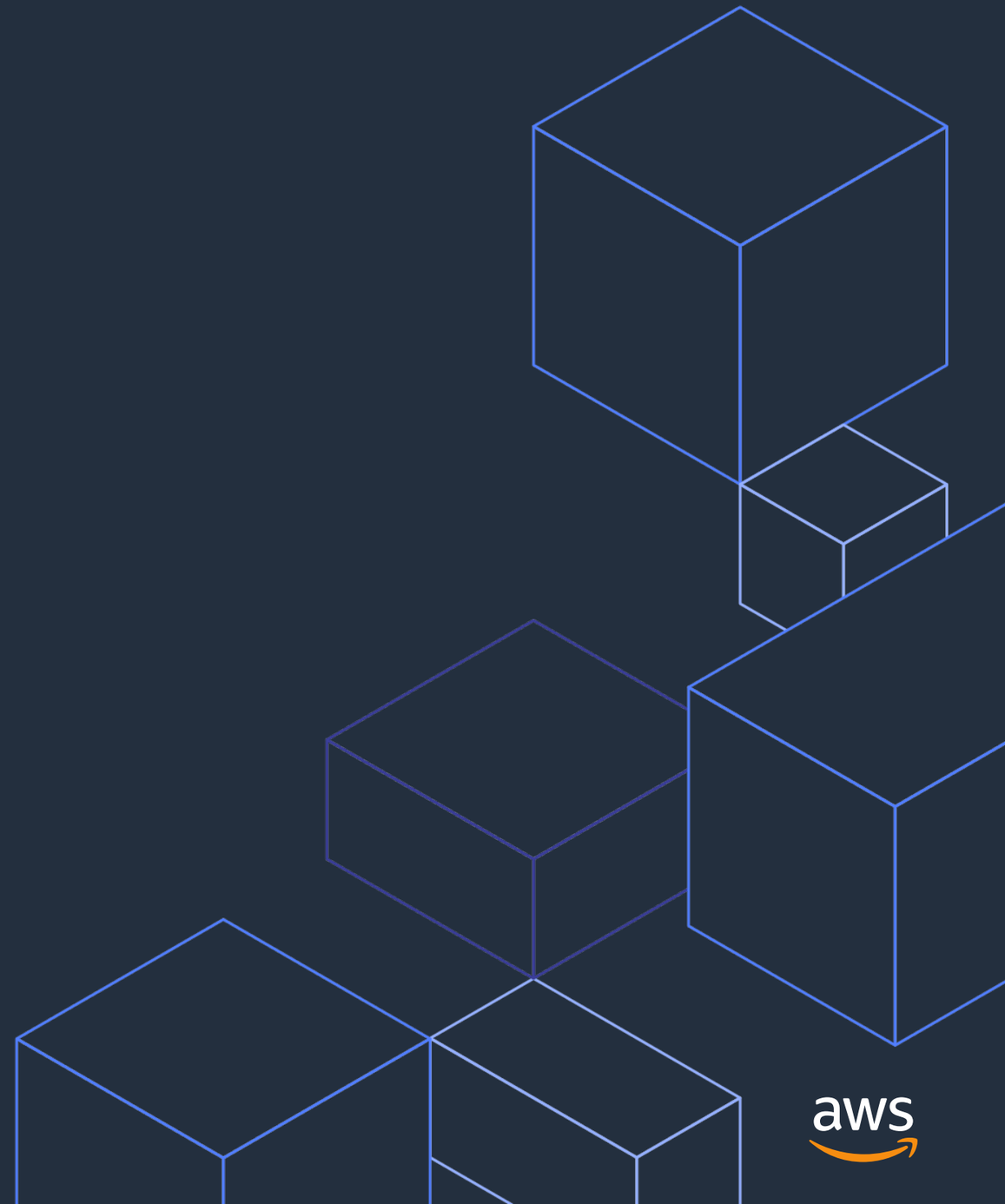


クエリの最適化

1. **ORDER BY を最適化する**
   LIMIT 句をつけることで、ORDER BY の負荷を軽減
2. **JOIN を最適化する**
   結合の際には、大きなテーブルを左側に、小さなテーブルを右にする
3. **GROUP BY を最適化する**
   複数カラムを指定する場合には、カーディナリティの高いカラムを前に持ってくる
4. **LIKE 演算子を最適化する**
   クエリ内で複数の LIKE 演算子を使う場合には、RegEx におきかえた方が高速になる
5. **近似関数を使う**
   多少の誤差を許容できるなら、COUNT DISTINCT でなく APPROX_DISTINCT() を使用
6. **必要なカラムだけを読みこむ**
   できるだけ * を使わず、必要なカラムだけを指定して SELECT 文を実行

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

Athena の運用管理



Workgroups を利用した利用量・コストの管理

- 同一アカウント内で、仮想的なワークグループを作成することが可能
- ワークグループごとに下記設定を実施可能
    - クエリワークロードの分離
    - クエリメトリクスの分離
    - クエリ毎のスキャン量上限設定
- スキャン量が一定を超えたらアラームをあげることも可能

https://docs.aws.amazon.com/ja_jp/athena/latest/ug/user-created-workgroups.html
https://aws.amazon.com/jp/about-aws/whats-new/2020/03/amazon-athena-adds-support-for-managing-athena-workgroups-using-aws-cloudformation/

Amazon EventBridge を利用したクエリの監視

- EventBridge (旧名 Amazon CloudWatch Events) を使用して、クエリの状態に関するリアルタイムの通知を受け取ることが可能に
- クエリの状態が移行する(たとえば、実行中から成功またはキャンセルなど)と、Athena はそのクエリ状態移行に関する情報を含むイベントを EventBridge に発行する
- 発行されたイベントに対応したルールを準備することで、下記のようなアクションを自動化することが可能
    - クエリが成功した場合に Lambda 関数を呼び出して後続処理を実行
    - クエリが失敗した場合にアラート通知を発行する

Amazon Athena のメトリクス

Athena の性能を把握するのに役立つ，以下の CloudWatch メトリクスがある

メトリクス	概要
EngineExecutionTime	クエリの実行にかかったミリ秒数
ProcessedBytes	クエリ内でスキャンされたデータのメガバイト数
Query Planning Time	クエリの計画を立てるためにかかるミリ秒数 データソースからテーブルパーティションを取得する時間も含まれる
Query Queuing Time	クエリがリソースを待ってキューに並んでいたミリ秒数
Service Processing Time	クエリエンジンがその実行を完了した後に結果を書き込むのにかかるミリ秒数
Total Execution Time	クエリを実行するのに Athena が費やした全体のミリ秒数

CloudTrail を利用した利用状況のロギング

- CloudTrail は、Athena の全ての API コールをイベントとしてキャプチャする

- CloudTrail によって収集される下記情報から、Athena に対してどのようなリクエストが行われたかを判断可能
    - リクエストの作成元の IP アドレス
    - リクエストの実行者
    - リクエストの実行日時
    - Athena で実行されたアクション

- 証跡を作成する場合は、Athena のイベントなど、Amazon S3 バケットへの CloudTrail イベントの継続的な配信を有効にすることが可能

https://docs.aws.amazon.com/ja_jp/athena/latest/ug/monitor-with-cloudtrail.html

Athena のセキュリティ

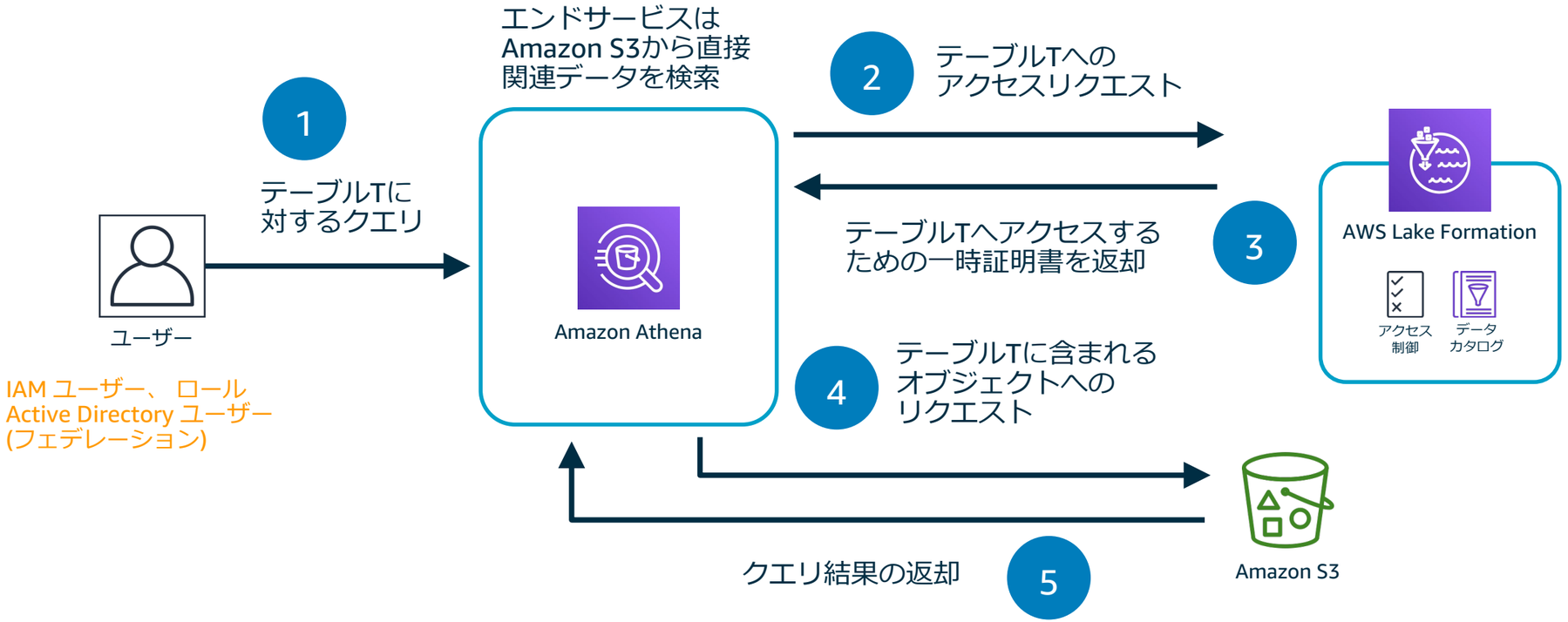
暗号化

- 保管時の暗号化
  - 暗号化データについて、以下に対応
    - 暗号化された Amazon S3 データに対するクエリ
    - 結果データを暗号化して Amazon S3 に保存
    - AWS Glue Data Catalog 内のデータ
  - 暗号化方式は以下の 3 種類
    - SSE-S3
    - SSE-KMS
    - CSE-KMS
  - 暗号化方式は、クエリと保存で別々のものを指定可能
- 転送時の暗号化
  - Athena と Amazon S3 間、Athena と Athena にアクセスするカスタマーアプリケーション間で、転送時のデータに Transport Layer Security (TLS) 暗号化を使用

https://docs.aws.amazon.com/ja_jp/athena/latest/ug/encryption.html

AWS Lake Formation による Fine Grained Access Control

- シンプルな権限の Grant/Revoke によりデータアクセスを制御
- データカタログのデータベース、テーブル、列に対して権限を指定可能
- 特定ユーザーに付与されたポリシーの確認も可能
- すべてのデータアクセスを1箇所で監査



https://docs.aws.amazon.com/ja_jp/lake-formation/latest/dg/access-control-overview.html

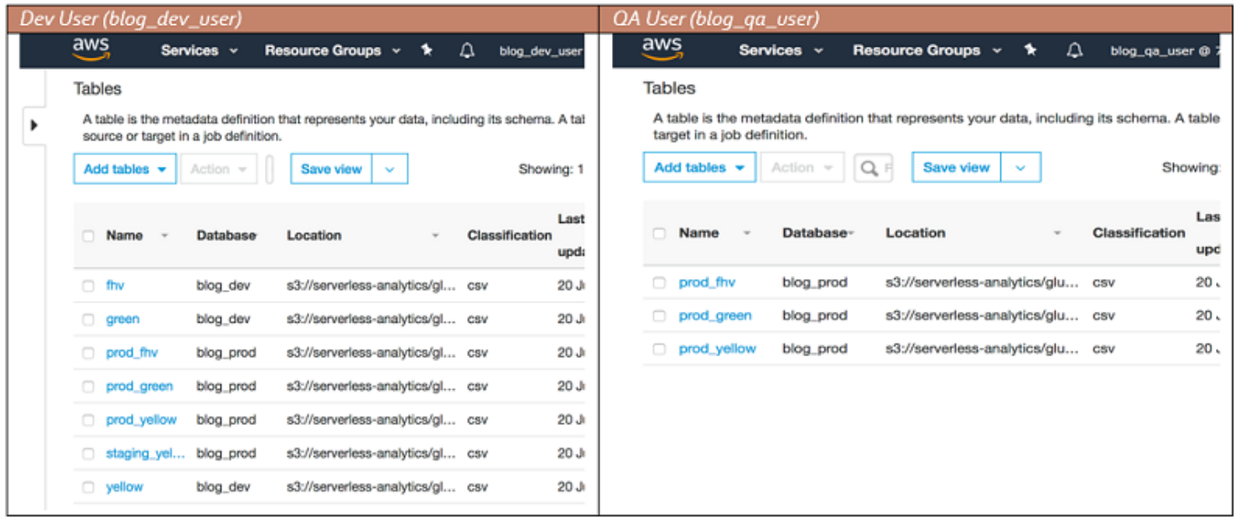
AWS Glue Data Catalog のリソースベースポリシー

- AWS Glue Data Catalog を用いることで、IAM ユーザーやロールに対して、テーブル単位でのアクセス権限を設定することが可能
- 権限のないテーブルは、Amazon Athena 側でも表示されなくなる
- AWS Glue Data Catalog はメタデータの管理しか行わないので、別途 Amazon S3 へのアクセスをバケットポリシーで制限する必要がある
- 制限されたバケットに対する Athena からのアクセスは aws:CalledVia 条件キーで制御する

Permissions

Add a policy to define fine-grained access control of the data catalog.

```
{
  "Version" : "2012-10-17",
  "Statement" : [ {
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "arn:aws:iam::123456789012:user/test_user"
    },
    "Action" : "glue:*",
    "Resource" : [
        "arn:aws:glue:us-east-1:123456789012:catalog",
        "arn:aws:glue:us-east-1:123456789012:database/default",
        "arn:aws:glue:us-east-1:123456789012:database/sandbox",
        "arn:aws:glue:us-east-1:123456789012:table/default/*"
    ]
  } ]
}
```



VPCエンドポイント

- インターネットを介さずに、Virtual Private Cloud (VPC) のインターフェイス VPC エンドポイントを使用して、Athena に直接接続可能

- インターフェイス VPC エンドポイントを使用すると、VPC と Athena の間の通信は完全に AWS ネットワーク内で実施されます

- VPC エンドポイントのポリシーを作成して、以下を指定できます
    - アクションを実行できるプリンシパル
    - 実行可能なアクション
    - このアクションを実行できるリソース

https://docs.aws.amazon.com/ja_jp/athena/latest/ug/interface-vpc-endpoint.html

サービス上限と料金

リソース制限（クエリ関連）

Amazon Athena の 1 アカウントあたりのクエリ関連のリソース制限は以下の通り

- 最大同時クエリ実行数: **20** (DDL とクエリそれぞれで 20 ずつ)
- クエリタイムアウト時間: **30** 分
- 1 秒あたりの API 実行数制限（下表）

API 名	デフォルト値	バースト時の値
BatchGetNamedQuery,BatchGetQueryExecution ListNamedQueries, ListQueryExecutions	5	最大 10
CreateNamedQuery, DeleteNamedQuery, GetNamedQuery StartQueryExecution, StopQueryExecution	5	最大 20
BatchGetQueryExecution	20	最大 40
StartQueryExecution, StopQueryExecution	20	最大 80
GetQueryExecution, GetQueryResults	100	最大 200

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

リソース制限（データベース/テーブル関連）

AWS Glue の 1 アカウントあたりのデータベース/テーブル関連の制限は
以下の通り

- データベース数: **10,000**
- データベースあたりテーブル数: **200,000**
- テーブルあたりパーティション数: **10,000,000**
- テーブル数: **1,000,000**
- パーティション数: **20,000,000**
- 関数の数: **100**

https://docs.aws.amazon.com/ja_jp/general/latest/gr/glue.html#limits_glue

料金

- クエリ単位の従量課金で、基本は S3 のデータスキャン 1TB につき 5 USD
    - 最低スキャン量は 10MB で、それ以下のものは 10MB に切り上げ
    - スキャン量の MB 未満の端数は切り上げて計算
    - スキャンの計算は、S3 上のデータサイズに基づいて行われるため、圧縮するだけで課金額を削減可能
- DDL および実行に失敗したクエリは、課金の対象外
- 別リージョンの Amazon S3 バケットからデータを読み込む場合は、別途 S3 のデータ転送料金が課金される
- 裏側で実行される Amazon S3 API リクエストについては、S3 側で別途課金
- AWS Glue Data Catalog の利用についても、別途課金
- VPC エンドポイント利用時、インターフェイス VPC エンドポイント料金が適用、課金される

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

リージョン

Amazon Athena は、東京リージョンを含む以下のリージョンで利用可能

アメリカ大陸

- 北バージニア
- オハイオ
- オレゴン
- 北カリフォルニア
- モントリオール
- サンパウロ
- GovCloud (米国西部)
- GovCloud (米国東部)

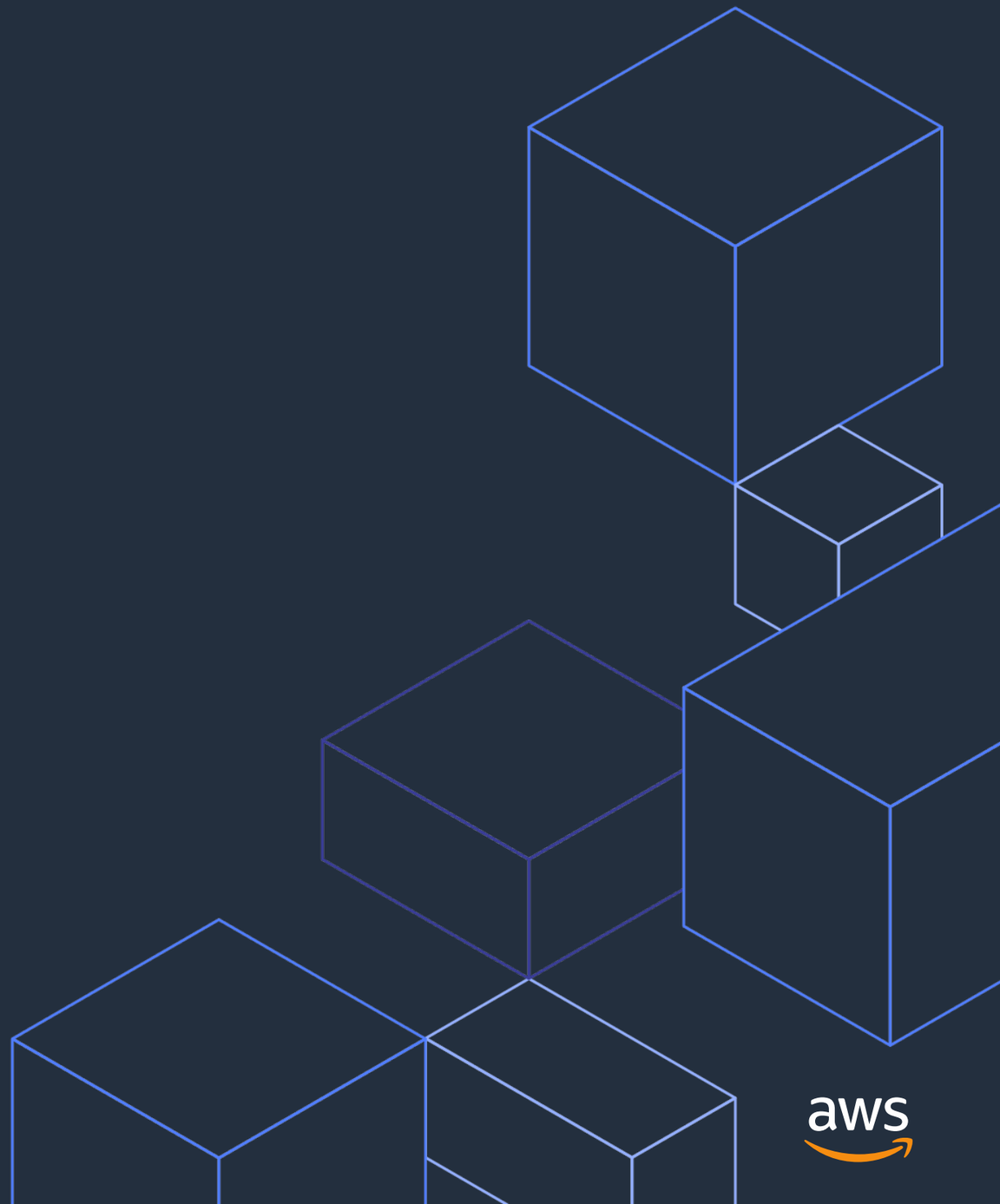
欧州/中東/アフリカ

- フランクフルト
- アイルランド
- ロンドン
- パリ
- ストックホルム
- バーレーン

アジアパシフィック

- シンガポール
- 東京
- シドニー
- ソウル
- ムンバイ
- 香港
- 北京
- 寧夏

まとめ



まとめ

- Amazon Athena は、Amazon S3 上のデータに対して、標準 SQL によるインタラクティブクエリを実行して、データ分析を行うことができるサービス

- データの持ち方やクエリの記述を工夫することで、より低いコストで、より高速にクエリを実行することが可能

- 性能改善・運用改善につながる様々な機能追加を行なっている

- 安全にデータ分析を実施可能な機能が揃っている

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
資料公開と併せて、後日掲載させていただきます。

AWS の日本語資料の場所「AWS 資料」で検索

https://amzn.to/JPArchive

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]

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