での Apache Iceberg の使用 AWS

AWS 規範ガイダンス

AWS 規範ガイダンス: での Apache Iceberg の使用 AWS

Table of Contents

での Apache Iceberg の使用 AWS

アマゾン ウェブ サービス (寄稿者)

2025 年 11 月 (ドキュメント履歴)

Apache Iceberg は、パフォーマンスを向上させながらテーブル管理を簡素化するオープンソースのテーブル形式です。Amazon EMR、Amazon Athena AWS Glue、Amazon Redshift などの AWS 分析サービスには Iceberg のネイティブサポートが含まれているため、Amazon Simple Storage Service (Amazon S3) 上にトランザクションデータレイクを簡単に構築できます AWS。

さらに、次世代の Amazon SageMaker は、データレイク、データウェアハウス、サードパーティーおよびフェデレーティッドソース間のデータアクセスを統合するオープンレイクハウスアーキテクチャ上に構築されています。 AWS レイクハウスは Iceberg と完全に互換性があり、Iceberg REST API を使用して、所定のデータに柔軟にアクセスしてクエリを実行できます。

このテクニカルガイドでは、さまざまな で Iceberg の使用を開始するためのガイダンスを提供します。また AWS のサービス、コストとパフォーマンスを最適化しながら、Iceberg AWS を大規模に実行するためのベストプラクティスと推奨事項が含まれています。

Iceberg の使用を開始したばかりのユーザーでも、既存の Iceberg ワークロードを最適化しようとしている経験豊富なユーザーでも AWS、このガイドはプロジェクトのすべての段階で貴重なインサイトを提供します。

このガイドの内容

最新のデータレイク

最新のデータレイクにおける高度なユースケース

データストレージの進化は、データベースからデータウェアハウス、データレイクへと進み、各テクノロジーが固有のビジネス要件とデータ要件に対処しています。従来のデータベースは構造化データとトランザクションワークロードの処理に優れていましたが、データ量が増えるにつれてパフォーマンスの課題に直面していました。データウェアハウスは、パフォーマンスとスケーラビリティの問題に取り組むために出現しましたが、データベースと同様に、垂直統合システム内の独自の形式に依存していました。

データレイクは、コスト、スケーラビリティ、柔軟性の観点からデータを保存するための最適なオプションの 1 つです。データレイクを使用すると、大量の構造化データと非構造化データを低コストで保持し、ビジネスインテリジェンスレポートからビッグデータ処理、リアルタイム分析、機械学習、生成人工知能 (AI) まで、さまざまなタイプの分析ワークロードにこのデータを使用して、より良い意思決定に役立てることができます。

これらの利点にもかかわらず、データレイクは当初、データベースのような機能で設計されていませんでした。データレイクは、アトミック性、整合性、分離性、耐久性 (ACID) 処理セマンティクスをサポートしていません。これは、さまざまなテクノロジーを使用して、数百人または数千人のユーザー間でデータを大規模に効果的に最適化および管理するために必要になる場合があります。データレイクは、以下の機能をネイティブにサポートしていません。

- ビジネスでのデータ変更に伴うレコードレベルの効率的な更新と削除の実行
- 数百万のファイルと数十万のパーティションへのテーブルの増加に伴うクエリパフォーマンスの管理
- 複数の同時ライターとリーダー間でのデータ整合性の確保
- オペレーションの途中で書き込みオペレーションが失敗した場合のデータ破損の防止
- データセットを (部分的に) 書き換えることなく、時間の経過とともにテーブルスキーマを進化させる

これらの課題は、変更データキャプチャ (CDC) の処理や、プライバシー、データの削除、ストリーミングデータ取り込みに関するユースケースなど、ユースケースで特に一般的になっており、最適ではないテーブルが発生する可能性があります。

従来の Hive 形式のテーブルを使用するデータレイクは、ファイル全体の書き込みオペレーションのみをサポートします。これにより、更新と削除の実装が難しく、時間がかかり、コストがかかります。さらに、データの整合性と一貫性を確保するために、ACID 準拠システムで提供される同時実行制御と保証が必要です。

これらの課題により、ユーザーはジレンマに陥ります。つまり、完全に統合された独自のプラットフォームを選択するか、ベンダーに依存しないがリソースを大量に消費する自己構築型のデータレイクを選択して、潜在的な価値を実現するには、継続的なメンテナンスと移行が必要です。

これらの課題を克服するために、Iceberg には、Amazon S3 などの費用対効果の高いシステム上のストレージをサポートしながら、データレイクの最適化と管理のオーバーヘッドを簡素化するデータベースのような機能が追加されています。

Apache Iceberg の概要

Apache Iceberg は、これまでデータベースまたはデータウェアハウスでのみ使用されていたデータレイクテーブルの機能を提供するオープンソースのテーブル形式です。スケーリングとパフォーマンスのために設計されており、数百ギガバイトを超えるテーブルの管理に適しています。Iceberg テーブルの主な機能は次のとおりです。

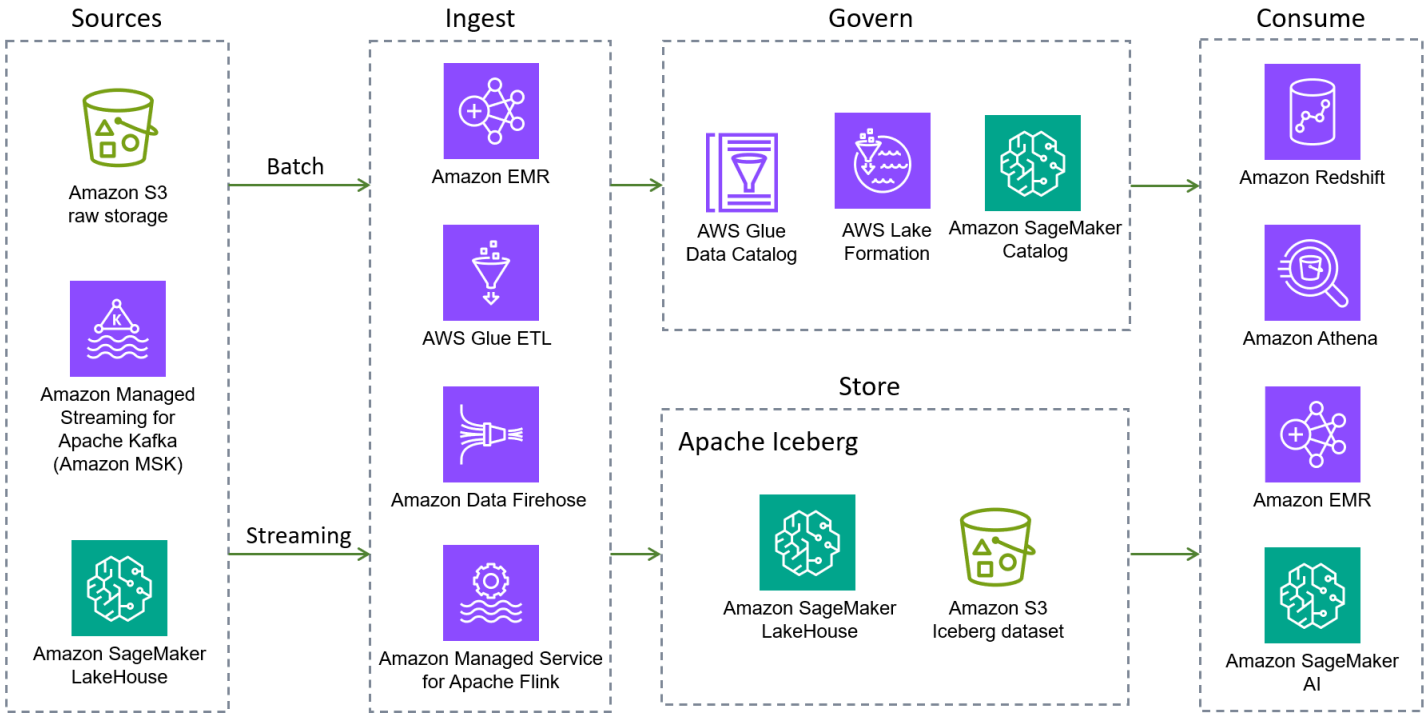
- 削除、更新、マージします。 Iceberg は、データレイクテーブルで使用するデータウェアハウス用の標準 SQL コマンドをサポートしています。
- 高速スキャン計画と高度なフィルタリング。Iceberg は、クエリの計画と実行を高速化するためにエンジンで使用できるパーティションや列レベルの統計などのメタデータを保存します。
- 完全なスキーマの進化。Iceberg は、副作用のない列の追加、削除、更新、名前変更をサポートしています。
- パーティションの進化。データボリュームまたはクエリパターンの変化に応じて、テーブルのパーティションレイアウトを更新できます。Iceberg は、テーブルがパーティション分割されている列の変更、複合パーティションへの列の追加、複合パーティションからの列の削除をサポートしています。
- 非表示のパーティショニング。 この機能は、不要なパーティションを自動的に読み取るのを防ぎます。これにより、ユーザーがテーブルのパーティショニングの詳細を理解したり、クエリにフィルターを追加したりする必要がなくなります。
- バージョンロールバック。ユーザーは、トランザクション前の状態に戻すことで問題をすばやく修正できます。
- タイムトラベル。ユーザーは、特定の以前のバージョンのテーブルをクエリできます。

- シリアル化可能な分離。テーブルの変更はアトミックであるため、読者は部分的またはコミットされていない変更を見ることはありません。
- 同時ライター。Iceberg はオプティミスティック同時実行を使用して、複数のトランザクションを成功させます。競合が発生した場合、ライターの 1 人がトランザクションを再試行する必要があります。
- ファイル形式を開きます。Iceberg は、Apache Parquet、Apache Avro、Apache ORC など、複数のオープンソースファイル形式をサポートしています。

要約すると、Iceberg 形式を使用するデータレイクは、トランザクションの一貫性、速度、スケール、スキーマの進化の恩恵を受けます。これらの機能やその他の Iceberg 機能の詳細については、Apache Iceberg のドキュメントを参照してください。

AWS Apache Iceberg のサポート

Apache Iceberg は、Amazon EMR、Amazon Athena、Amazon Redshift、AWS Glue、Amazon SageMaker AWS のサービス などの でサポートされています。次の図は、Iceberg に基づくデータレイクの簡略化されたリファレンスアーキテクチャを示しています。



以下は AWS のサービス 、Iceberg のネイティブ統合を提供します。間接的に、または Iceberg ライブラリをパッケージ化することによって、Iceberg とやり取り AWS のサービス できる追加 があります。

- Amazon S3 は、耐久性、可用性、スケーラビリティ、セキュリティ、コンプライアンス、監査機能を備えているため、データレイクを構築するのに最適な場所です。Iceberg は Amazon S3 とシームレスにやり取りするように設計および構築されており、Iceberg ドキュメントに記載されている多くの Amazon S3 機能をサポートしています。さらに、Amazon S3 Tables は、Iceberg のサポートが組み込まれた最初のクラウドオブジェクトストアを提供し、大規模な表形式データの保存を合理化します。Iceberg の S3 Tables のサポートにより、一般的なクエリエンジン AWS とサードパーティーのクエリエンジンを使用して、表形式のデータを簡単にクエリできます。
- 次世代の SageMaker は、Amazon S3 データレイク、Amazon Redshift データウェアハウス、サードパーティーおよびフェデレーティッドデータソース間のデータアクセスを統合するオープンレイクハウスアーキテクチャ上に構築されています。これらの機能は、データの 1 つのコピーで強力な分析と AI/ML アプリケーションを構築するのに役立ちます。レイクハウスは Iceberg と完全に互換性があるため、Iceberg REST API を使用してデータにアクセスしてクエリを実行する柔軟性があります。
- Amazon EMR は、Apache Spark、Flink、Trino、Hive などのオープンソースフレームワークを使用してペタバイト規模のデータ処理、インタラクティブ分析、機械学習を行うためのビッグデータソリューションです。Amazon EMR は、カスタマイズされた Amazon Elastic Compute Cloud (Amazon EC2) クラスター、Amazon Elastic Kubernetes Service (Amazon EKS)、 AWS Outposts、または Amazon EMR Serverless で実行できます。
- Amazon Athena は、オープンソースフレームワーク上に構築されたサーバーレスのインタラクティブな分析サービスです。オープンテーブル形式とファイル形式をサポートし、ペタバイト単位のデータを分析するためのシンプルで柔軟な方法を提供します。Athena は Iceberg の読み取り、タイムトラベル、書き込み、DDL クエリをネイティブにサポートし、Iceberg メタストア AWS Glue Data Catalog に を使用します。
- Amazon Redshift は、クラスターベースとサーバーレスの両方のデプロイオプションをサポートするペタバイト規模のクラウドデータウェアハウスです。Amazon Redshift Spectrum は、 に登録 AWS Glue Data Catalog されAmazon S3に保存されている外部テーブルをクエリできます。Redshift Spectrum は Iceberg ストレージ形式もサポートしています。
- AWS Glue は、分析、機械学習 (ML)、アプリケーション開発のために複数のソースからのデータを簡単に検出、準備、移動、統合できるサーバーレスデータ統合サービスです。Iceberg と完全に統合されています。具体的には、 AWS Glue ジョブを使用して Iceberg テーブルの読み取りおよび書き込みオペレーションを実行し、 AWS Glue Data Catalog (Hive メタストア互換) を使用してテーブルを管理し、 AWS Glue クローラを使用してテーブルを自動的に検出して登録し、 AWS Glue Data Quality 機能を使用して Iceberg テーブルのデータ品質を評価できます。は、列統計の収集、Iceberg テーブルの各列の個別値 (NDVs) の数の計算と更新、およびテーブルの自動最適化 (圧縮、スナップショット保持、孤立ファイル削除) AWS Glue Data Catalog もサポートしていま

す。 は、 AWS のサービス およびサードパーティーアプリケーションのリストから Iceberg テーブルへのゼロ ETL 統合 AWS Glue もサポートしています。

- Amazon Data Firehose は、Amazon S3、Amazon Redshift、Amazon OpenSearch Service、Amazon OpenSearch Serverless、Splunk、Apache Iceberg テーブルなどの送信先、および Datadog、Dynatrace、LogicMonitor、MongoDB、New Relic、Coralogix、Elastic など、サポートされているサードパーティーサービスプロバイダーが所有するカスタム HTTP または HTTP エンドポイントにリアルタイムのストリーミングデータを配信するためのフルマネージドサービスです。Firehose では、アプリケーションを記述したり、リソースを管理したりする必要はありません。Firehose にデータを送信するデータプロデューサーを作成すると、それにより、指定した送信先にデータが自動配信されます。データを配信前に変換するように、Firehose を設定することもできます。
- Amazon Managed Service for Apache Flink は、Apache Flink アプリケーションを使用してストリーミングデータを処理できるフルマネージド型の Amazon サービスです。Iceberg テーブルとの読み取りと書き込みの両方をサポートし、リアルタイムのデータ処理と分析を可能にします。
- Amazon SageMaker AI は、Iceberg 形式を使用して Amazon SageMaker AI Feature Store の機能セットのストレージをサポートします。
- AWS Lake Formation は、Athena または Amazon Redshift によって消費される Iceberg テーブルなど、データにアクセスするための粗くきめ細かなアクセスコントロール許可を提供します。Iceberg テーブルのアクセス許可のサポートの詳細については、Lake Formation ドキュメントを参照してください。

AWS には Iceberg をサポートする幅広いサービスがありますが、これらのサービスをすべてカバーすることは、このガイドの範囲外です。以下のセクションでは、Amazon EMR および の Spark (バッチおよび構造化ストリーミング) AWS Glueと Athena SQL について説明します。次のセクションでは、Athena SQL での Iceberg サポートについて簡単に説明します。

Amazon Athena SQL での Iceberg テーブルの開始方法

Amazon Athena は Iceberg の組み込みサポートを提供します。Athena ドキュメントの「開始方法」セクションで説明されているサービスの前提条件を設定する以外は、追加のステップや設定なしで Iceberg を使用できます。このセクションでは、Athena でテーブルを作成する方法を簡単に説明します。詳細については、このガイドの後半の「Athena SQL を使用した Iceberg テーブルの操作」を参照してください。

異なるエンジン AWS を使用して、 で Iceberg テーブルを作成できます。これらのテーブルはシームレスに動作します AWS のサービス。Athena SQL で最初の Iceberg テーブルを作成するには、次の定型コードを使用できます。

```
CREATE TABLE <table_name> (
    col_1 string,
    col_2 string,
    col_3 bigint,
    col_ts timestamp)
PARTITIONED BY (col_1, <<<partition_transform>>>(col_ts))
LOCATION 's3://<bucket>/<folder>/<table_name>/'
TBLPROPERTIES (
    'table_type' ='ICEBERG'
)
```

以下のセクションでは、Athena でパーティション分割された Iceberg テーブルとパーティション分割されていない Iceberg テーブルを作成する例を示します。詳細については、Athena ドキュメントで詳述されている Iceberg 構文を参照してください。

パーティション化されていないテーブルの作成

次のステートメント例では、定型 SQL コードをカスタマイズして、Athena でパーティション分割されていない Iceberg テーブルを作成します。このステートメントを Athena コンソールのクエリエディタに追加して、テーブルを作成できます。

```
CREATE TABLE athena_iceberg_table (
    color string,
    date string,
    name string,
    price bigint,
```

```
    product string,
    ts timestamp)
LOCATION 's3://DOC_EXAMPLE_BUCKET/ice_warehouse/iceberg_db/athena_iceberg_table/'
TBLPROPERTIES (
    'table_type' ='ICEBERG'
)
```

クエリエディタを使用するstep-by-stepについては、Athena ドキュメントの「開始方法」を参照してください。

パーティションテーブルの作成

次のステートメントは、Iceberg の非表示パーティショニングの概念を使用して、日付に基づいてパーティションテーブルを作成します。day( ) 変換を使用して、タイムスタンプ列から dd-mm-yyyy形式を使用して日次パーティションを取得します。Iceberg はこの値をデータセットの新しい列として保存しません。代わりに、データを書き込んだりクエリしたりすると、その場で値が算出されます。

```
CREATE TABLE athena_iceberg_table_partitioned (
    color string,
    date string,
    name string,
    price bigint,
    product string,
    ts timestamp)
PARTITIONED BY (day(ts))
LOCATION 's3://DOC_EXAMPLE_BUCKET/ice_warehouse/iceberg_db/athena_iceberg_table/'
TBLPROPERTIES (
    'table_type' ='ICEBERG'
)
```

単一の CTAS ステートメントを使用してテーブルを作成し、データをロードする

前のセクションのパーティション分割された例とパーティション分割されていない例では、Iceberg テーブルは 空のテーブルとして作成されます。INSERT または MERGEステートメントを使用して、テーブルにデータをロードできます。または、CREATE TABLE AS SELECT (CTAS)ステートメントを使用して、1 つのステップで Iceberg テーブルにデータを作成してロードすることもできます。

CTAS は、Athena でテーブルを作成し、1 つのステートメントにデータをロードする最適な方法です。次の例は、CTAS を使用して Athena の既存の Hive/Parquet テーブル (iceberg_ctas_table) から Iceberg テーブル (hive_table) を作成する方法を示しています。

```
CREATE TABLE iceberg_ctas_table WITH (
  table_type = 'ICEBERG',
  is_external = false,
  location = 's3://DOC_EXAMPLE_BUCKET/ice_warehouse/iceberg_db/iceberg_ctas_table/'
) AS
SELECT * FROM "iceberg_db"."hive_table" limit 20
---
SELECT * FROM "iceberg_db"."iceberg_ctas_table" limit 20
```

CTAS の詳細については、Athena CTAS ドキュメントを参照してください。

データの挿入、更新、削除

Athena は、、、INSERT INTO、、MERGE INTOおよび DELETE FROM ステートメントを使用して Iceberg UPDATEテーブルにデータを書き込むさまざまな方法をサポートしています。

> **Note**
>
> Athena SQL は現在copy-on-writeアプローチをサポートしていません。UPDATE、MERGE INTO、および DELETE FROMオペレーションでは、指定されたテーブルプロパティに関係なく、位置削除で常にmerge-on-readアプローチを使用します。write.update.mode、、 などのテーブルプロパティをcopy-on-writeを使用するwrite.merge.modewrite.delete.modeように設定した場合、クエリは失敗しませんが、Athena はそれらを無視してmerge-on-readを使用し続けます。

次のステートメントではINSERT INTO、 を使用して Iceberg テーブルにデータを追加します。

```
INSERT INTO "iceberg_db"."ice_table" VALUES (
    'red', '222022-07-19T03:47:29', 'PersonNew', 178, 'Tuna', now()
)

SELECT * FROM "iceberg_db"."ice_table"
where color = 'red' limit 10;
```

サンプル出力:

詳細については、Athena のドキュメントを参照してください。

Iceberg テーブルのクエリ

前の例に示すように、Athena SQL を使用して Iceberg テーブルに対して通常の SQL クエリを実行できます。

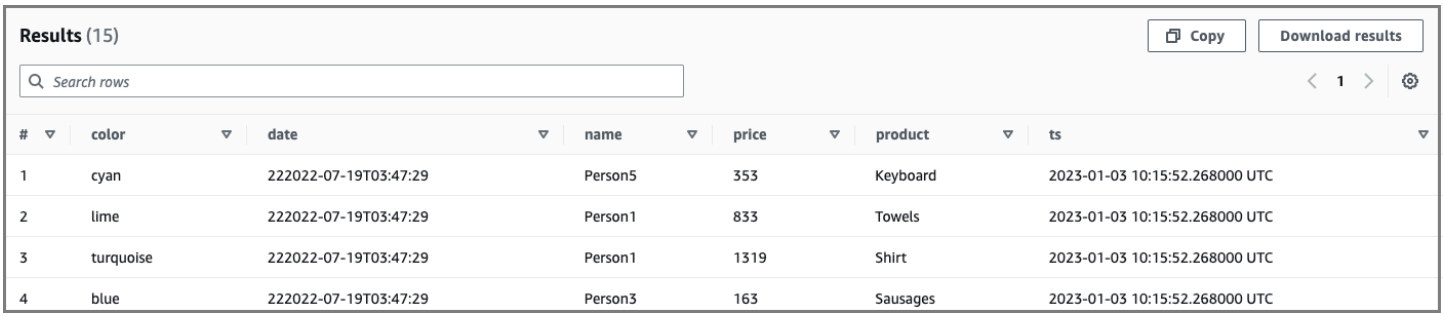
通常のクエリに加えて、Athena は Iceberg テーブルのタイムトラベルクエリもサポートしています。前述のように、Iceberg テーブルの更新または削除を通じて既存のレコードを変更できるため、タイムトラベルクエリを使用して、タイムスタンプまたはスナップショット ID に基づいてテーブルの古いバージョンをさかのぼるのが便利です。

たとえば、次のステートメントは の色値を更新しPerson5、2023 年 1 月 4 日以前の値を表示します。

```
UPDATE ice_table SET color='new_color' WHERE name='Person5'

SELECT * FROM "iceberg_db"."ice_table" FOR TIMESTAMP AS OF TIMESTAMP '2023-01-04
 12:00:00 UTC'
```

サンプル出力:



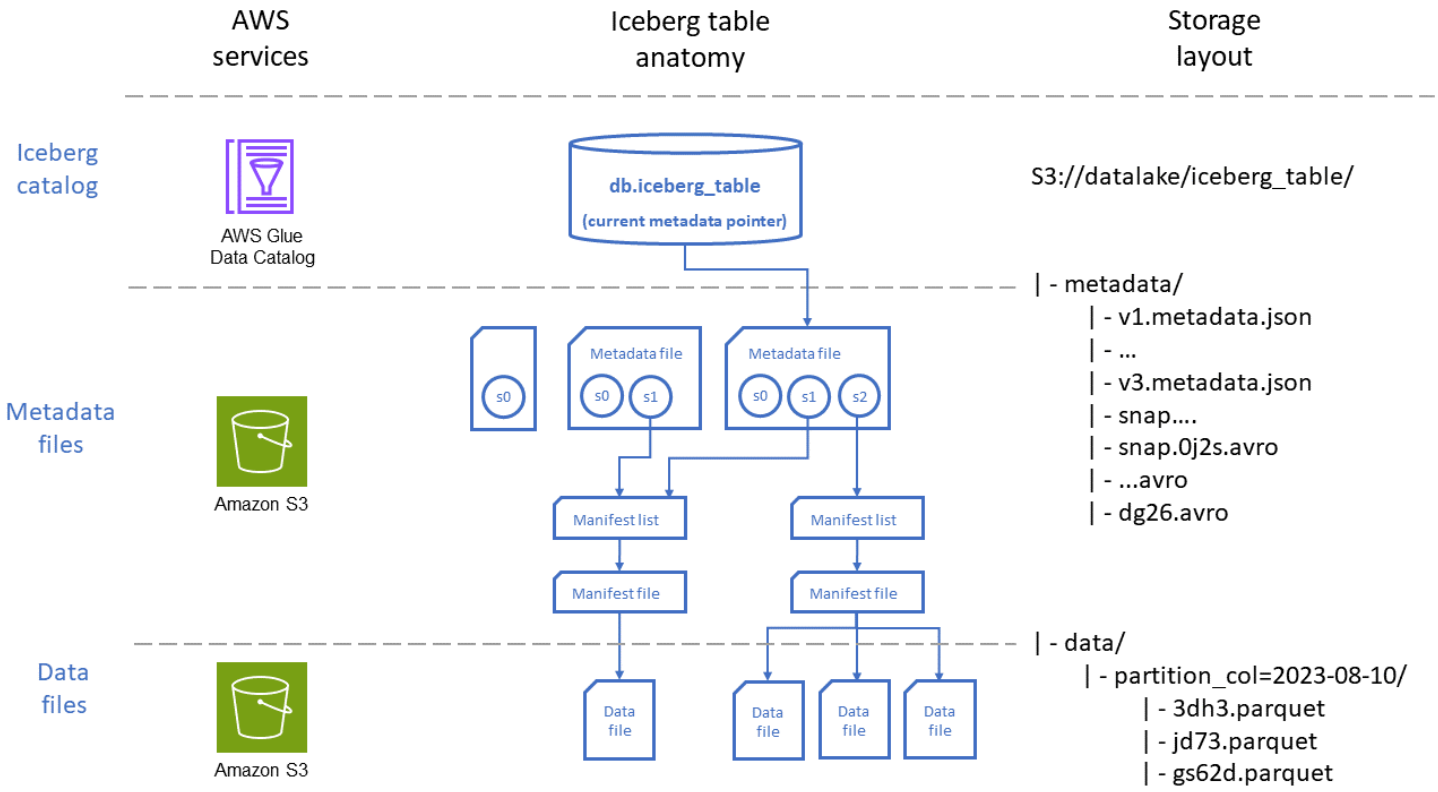
構文とタイムトラベルクエリのその他の例については、Athena ドキュメントを参照してください。

Iceberg テーブルの構造

Iceberg テーブルを使用するための基本的なステップを説明したので、Iceberg テーブルの複雑な詳細と設計について詳しく見てみましょう。

このガイドで前述した機能を有効にするために、Iceberg はデータとメタデータファイルの階層レイヤーを使用して設計されています。これらのレイヤーはメタデータをインテリジェントに管理し、クエリの計画と実行を最適化します。

次の図は、Iceberg テーブルの編成を 2 つの視点で示しています。テーブルの保存 AWS のサービスに使用される と、Amazon S3 でのファイル配置です。



図に示すように、Iceberg テーブルは 3 つのメインレイヤーで構成されます。

- Iceberg Catalog: は Iceberg とネイティブ AWS Glue Data Catalog に統合されており、ほとんどのユースケースでは、 で実行されるワークロードに最適なオプションです AWS。Iceberg テーブルとやり取りするサービス (Athena など) は、カタログを使用してテーブルの現在のスナップショットバージョンを検索し、データの読み取りまたは書き込みを行います。
- メタデータレイヤー: メタデータファイル、つまりマニフェストファイルとマニフェストリストファイルは、テーブルのスキーマ、パーティション戦略、データファイルの場所などの情報と、各

データファイルに保存されているレコードの最小範囲と最大範囲などの列レベルの統計情報を追跡します。これらのメタデータファイルは、テーブルパス内の Amazon S3 に保存されます。

- マニフェストファイルには、場所、形式、サイズ、チェックサム、その他の関連情報など、各データファイルのレコードが含まれます。
- マニフェストリストは、マニフェストファイルのインデックスを提供します。テーブル内のマニフェストファイルの数が増えるにつれて、その情報を小さなサブセクションに分割することで、クエリでスキャンする必要があるマニフェストファイルの数を減らすことができます。
- メタデータファイルには、マニフェストリスト、スキーマ、パーティションメタデータ、スナップショットファイル、およびテーブルのメタデータの管理に使用されるその他のファイルなど、Iceberg テーブル全体に関する情報が含まれます。

- データレイヤー: このレイヤーには、クエリが実行されるデータレコードを持つファイルが含まれています。これらのファイルは、Apache Parquet、Apache Avro、Apache ORC など、さまざまな形式で保存できます。
  - データファイルには、テーブルのデータレコードが含まれます。
  - Iceberg テーブルの行レベルの削除および更新オペレーションをエンコードするファイルを削除します。Iceberg には、Iceberg ドキュメントで説明されているように、2 種類の削除ファイルがあります。これらのファイルは、merge-on-readモードを使用して オペレーションによって作成されます。

Amazon EMR での Iceberg の使用

Amazon EMR は、Apache Spark、Apache Hive、Flink、Trino などのオープンソースフレームワークを使用して、ペタバイト規模のデータ処理、インタラクティブ分析、機械学習をクラウドで提供します。

> **Note**
>
> このガイドでは、Apache Spark を例として使用します。

Amazon EMR は、EC2 上の Amazon EMR、EKS 上の Amazon EMR、Amazon EMR Serverless、Amazon EMR の複数のデプロイオプションをサポートしています AWS Outposts。ワークロードのデプロイオプションを選択するには、Amazon EMR のよくある質問を参照してください。

バージョンと機能の互換性

Amazon EMR バージョン 6.5.0 以降のバージョンでは、Apache Iceberg がネイティブにサポートされています。各 Amazon EMR リリースでサポートされている Iceberg バージョンのリストについては、Amazon EMR ドキュメントの「Iceberg リリース履歴」を参照してください。また、「Iceberg でクラスターを使用する」のセクションを参照して、さまざまなフレームワークで Amazon EMR でサポートされている Iceberg 機能を確認してください。

サポートされている最新の Iceberg バージョンを活用するには、最新の Amazon EMR バージョンを使用することをお勧めします。このセクションのコード例と設定は、Amazon EMR リリース emr-7.8.0 を使用していることを前提としています。

Iceberg を使用した Amazon EMR クラスターの作成

Iceberg がインストールされた Amazon EC2 に Amazon EMR クラスターを作成するには、Amazon EMR ドキュメントの指示に従います。

具体的には、クラスターは次の分類で設定する必要があります。

```
[{
    "Classification": "iceberg-defaults",
    "Properties": {
```

```
        "iceberg.enabled": "true"
    }
}]
```

Amazon EMR Serverless または Amazon EMR on EKS を、Amazon EMR 6.6.0 から Iceberg ワークロードのデプロイオプションとして使用することもできます。

Amazon EMR での Iceberg アプリケーションの開発

Iceberg アプリケーションの Spark コードを開発するには、Amazon EMR Studio を使用できます。Amazon EMR Studio は、Amazon EMR クラスターで実行されるフルマネージド Jupyter ノートブック用のウェブベースの統合開発環境 (IDE) です。

Amazon EMR Studio ノートブックの使用

Amazon EMR Studio Workspace ノートブックで Spark アプリケーションをインタラクティブに開発し、それらのノートブックを EC2 クラスターの Amazon EMR または EKS マネージドエンドポイントの Amazon EMR に接続できます。EC2 での Amazon EMR 用 EMR Studio と EKS での Amazon EMR のセットアップ手順については、 AWS のサービス ドキュメントを参照してください。

EMR Studio で Iceberg を使用するには、次の手順に従います。

1. 「Iceberg がインストールされたクラスターを使用する」の説明に従って、Iceberg を有効にした Amazon EMR クラスターを起動します。
2. EMR Studio をセットアップします。手順については、「Amazon EMR Studio のセットアップ」を参照してください。
3. EMR Studio Workspace ノートブックを開き、ノートブックの最初のセルとして次のコードを実行して、Iceberg を使用するように Spark セッションを設定します。

```
%%configure -f
{
    "conf": {
        "spark.sql.catalog.<catalog_name>": "org.apache.iceberg.spark.SparkCatalog",
        "spark.sql.catalog.<catalog_name>.warehouse": "s3://YOUR-BUCKET-NAME/YOUR-
FOLDER-NAME/",
        "spark.sql.catalog.<catalog_name>.type": "glue",
        "spark.sql.extensions":
 "org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions"
    }
}
```

各パラメータの意味は次のとおりです。

- <catalog_name> は Iceberg Spark セッションカタログ名です。任意の名前に置き換え、このカタログに関連付けられているすべての設定で参照を必ず変更してください。コードでは、次のように Spark セッションカタログ名を含む完全修飾テーブル名を使用して Iceberg テーブルを参照できます。

```
<catalog_name>.<database_name>.<table_name>
```

  または、デフォルトのカタログを、 をカタログ名に設定して定義した Iceberg カタログspark.sql.defaultCatalogに変更することもできます。この 2 番目の方法では、カタログプレフィックスなしでテーブルを参照できるため、クエリを簡素化できます。

- <catalog_name>.warehouse は、データとメタデータを保存する Amazon S3 パスを指します。
- カタログを にするには AWS Glue Data Catalog、spark.sql.catalog.<catalog_name>.type を に設定しますglue。このキーは、カスタムカタログ実装の実装クラスを指すために必要です。このガイドの後半にある一般的なベストプラクティスセクションでは、Iceberg がサポートするさまざまなカタログについて説明します。

4. 他の Spark アプリケーションと同様に、ノートブックで Iceberg 用の Spark アプリケーションをインタラクティブに開発できるようになりました。

Amazon EMR Studio を使用して Spark for Apache Iceberg を設定する方法の詳細については、ブログ記事「Build a high-performance, ACID compliant, evolving data lake using Apache Iceberg on Amazon EMR」を参照してください。

Amazon EMR での Iceberg ジョブの実行

Iceberg ワークロードの Spark アプリケーションコードを作成したら、Iceberg をサポートする任意の Amazon EMR デプロイオプションで実行できます (Amazon EMR のよくある質問を参照) 。

他の Spark ジョブと同様に、ステップを追加するか、Spark ジョブをマスターノードにインタラクティブに送信することで、Amazon EMR on EC2 クラスターに作業を送信できます。Spark ジョブを実行するには、次の Amazon EMR ドキュメントページを参照してください。

- EC2 クラスターで Amazon EMR に作業を送信するためのさまざまなオプションの概要と各オプションの詳細な手順については、「クラスターに作業を送信する」を参照してください。

- Amazon EMR on EKS については、StartJobRun を使用した Spark ジョブの実行」を参照してください。
- EMR Serverless については、「ジョブの実行」を参照してください。

以下のセクションでは、各 Amazon EMR デプロイオプションの例を示します。

Amazon EMR on EC2

以下の手順を使用して Iceberg Spark ジョブを送信できます。

1. ワークステーションに次のコンテンツemr_step_iceberg.jsonを含む ファイルを作成します。

```
[{
    "Name": "iceberg-test-job",
    "Type": "spark",
    "ActionOnFailure": "CONTINUE",
    "Args": [
        "--deploy-mode",
        "client",
        "--conf",

  "spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions",
        "--conf",
        "spark.sql.catalog.<catalog_name>=org.apache.iceberg.spark.SparkCatalog",
        "--conf",
        "spark.sql.catalog.<catalog_name>.type=glue",
        "--conf",
        "spark.sql.catalog.<catalog_name>.warehouse=s3://YOUR-BUCKET-NAME/YOUR-
FOLDER-NAME/",
        "s3://YOUR-BUCKET-NAME/code/iceberg-job.py"
    ]
}]
```

2. 太字で強調表示されている Iceberg 設定オプションをカスタマイズして、特定の Spark ジョブの設定ファイルを変更します。
3. AWS Command Line Interface () を使用してステップを送信しますAWS CLI。emr_step_iceberg.json ファイルが配置されているディレクトリで コマンドを実行します。

```
aws emr add-steps --cluster-id <cluster_id> --steps file://emr_step_iceberg.json
```

Amazon EMR Serverless

を使用して Iceberg Spark ジョブを EMR Serverless に送信するには AWS CLI :

1. ワークステーションに次のコンテンツemr_serverless_iceberg.jsonを含む ファイルを作成します。

```
{
    "applicationId": "<APPLICATION_ID>",
    "executionRoleArn": "<ROLE_ARN>",
    "name": "iceberg-test-job",
    "jobDriver": {
        "sparkSubmit": {
            "entryPoint": "s3://YOUR-BUCKET-NAME/code/iceberg-job.py",
            "entryPointArguments": []
        }
    },
    "configurationOverrides": {
        "applicationConfiguration": [{
            "classification": "spark-defaults",
            "properties": {
                "spark.sql.extensions":
 "org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions",
                "spark.sql.catalog.<catalog_name>":
 "org.apache.iceberg.spark.SparkCatalog",
                "spark.sql.catalog.<catalog_name>.type": "glue",
                "spark.sql.catalog.<catalog_name>.warehouse": "s3://YOUR-BUCKET-NAME/
YOUR-FOLDER-NAME/",
                "spark.jars":"/usr/share/aws/iceberg/lib/iceberg-spark3-runtime.jar",

 "spark.hadoop.hive.metastore.client.factory.class":"com.amazonaws.glue.catalog.metastore.AWS
            }
        }],
        "monitoringConfiguration": {
            "s3MonitoringConfiguration": {
                "logUri": "s3://YOUR-BUCKET-NAME/emr-serverless/logs/"
            }
        }
    }
```

```
}
```

2. 太字で強調表示されている Iceberg 設定オプションをカスタマイズして、特定の Spark ジョブの設定ファイルを変更します。
3. を使用してジョブを送信します AWS CLI。`emr_serverless_iceberg.json` ファイルが配置されているディレクトリで コマンドを実行します。

```
aws emr-serverless start-job-run --cli-input-json file://emr_serverless_iceberg.json
```

EMR Studio コンソールを使用して Iceberg Spark ジョブを EMR Serverless に送信するには：

1. EMR Serverless ドキュメントの指示に従ってください。
2. ジョブ設定では、 に用意されている Spark の Iceberg 設定を使用し AWS CLI 、Iceberg の強調表示されたフィールドをカスタマイズします。詳細な手順については、Amazon EMR ドキュメントの「Using Apache Iceberg with EMR Serverless」を参照してください。

Amazon EMR on EKS

を使用して Iceberg Spark ジョブを Amazon EMR on EKS に送信するには AWS CLI

1. ワークステーションに次のコンテンツ`emr_eks_iceberg.json`を含む ファイルを作成します。

```
{
    "name": "iceberg-test-job",
    "virtualClusterId": "<VIRTUAL_CLUSTER_ID>",
    "executionRoleArn": "<ROLE_ARN>",
    "releaseLabel": "emr-6.9.0-latest",
    "jobDriver": {
        "sparkSubmitJobDriver": {
            "entryPoint": "s3://YOUR-BUCKET-NAME/code/iceberg-job.py",
            "entryPointArguments": [],
            "sparkSubmitParameters": "--jars local:///usr/share/aws/iceberg/lib/
iceberg-spark3-runtime.jar"
        }
    },
    "configurationOverrides": {
        "applicationConfiguration": [{
            "classification": "spark-defaults",
            "properties": {
```

```
                "spark.sql.extensions":
 "org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions",
                "spark.sql.catalog.<catalog_name>":
 "org.apache.iceberg.spark.SparkCatalog",
                "spark.sql.catalog.<catalog_name>.type": "glue",
                "spark.sql.catalog.<catalog_name>.warehouse": "s3://YOUR-BUCKET-NAME/
YOUR-FOLDER-NAME/",
                "spark.hadoop.hive.metastore.client.factory.class":
 "com.amazonaws.glue.catalog.metastore.AWSGlueDataCatalogHiveClientFactory"
            }
        }],
        "monitoringConfiguration": {
            "persistentAppUI": "ENABLED",
            "s3MonitoringConfiguration": {
                "logUri": "s3://YOUR-BUCKET-NAME/emr-serverless/logs/"
            }
        }
    }
}
```

2. 太字で強調表示されている Iceberg 設定オプションをカスタマイズして、Spark ジョブの設定ファイルを変更します。
3. を使用してジョブを送信します AWS CLI。`emr_eks_iceberg.json` ファイルが配置されているディレクトリで次のコマンドを実行します。

```
aws emr-containers start-job-run --cli-input-json file://emr_eks_iceberg.json
```

詳細な手順については、Amazon EMR on EKS ドキュメントの「Amazon EMR on EKS での Apache Iceberg の使用」を参照してください。

Amazon EMR のベストプラクティス

このセクションでは、Amazon EMR で Spark ジョブを調整して、Iceberg テーブルへのデータの読み取りと書き込みを最適化するための一般的なガイドラインを提供します。Iceberg 固有のベストプラクティスについては、このガイドの後半にある「ベストプラクティス」セクションを参照してください。

- Amazon EMR の最新バージョンを使用する – Amazon EMR は、Amazon EMR Spark ランタイムで Spark 最適化をすぐに提供します。 AWS は、新しいリリースごとに Spark ランタイムエンジンのパフォーマンスを向上させます。

- Spark ワークロードに最適なインフラストラクチャを決定する – Spark ワークロードでは、最適なパフォーマンスを確保するために、ジョブ特性ごとに異なるタイプのハードウェアが必要になる場合があります。Amazon EMR は、すべてのタイプの処理要件をカバーするために、複数のインスタンスタイプ (コンピューティング最適化、メモリ最適化、汎用、ストレージ最適化など) をサポートしています。新しいワークロードをオンボードするときは、M5 や M6g などの一般的なインスタンスタイプでベンチマークすることをお勧めします。Ganglia と Amazon CloudWatch のオペレーティングシステム (OS) と YARN メトリクスをモニタリングして、ピーク負荷時のシステムのボトルネック (CPU、メモリ、ストレージ、I/O) を判断し、適切なハードウェアを選択します。
- チューニング `spark.sql.shuffle.partitions` — `spark.sql.shuffle.partitions`プロパティをクラスター内の仮想コア (vCores の合計数、またはその値の倍数 (通常は vCoresの合計数の 1~2 倍) に設定します。この設定は、書き込み分散モードとしてハッシュパーティショニングと範囲パーティショニングを使用する場合の Spark の並列処理に影響します。データを整理するために書き込む前にシャッフルをリクエストするため、パーティションの整列が保証されます。
- マネージドスケーリングを有効にする – ほとんどのユースケースでは、マネージドスケーリングと動的割り当てを有効にすることをお勧めします。ただし、予測可能なパターンを持つワークロードがある場合は、自動スケーリングと動的割り当てを無効にすることをお勧めします。マネージドスケーリングが有効になっている場合は、コストを削減するためにスポットインスタンスを使用することをお勧めします。コアノードまたはマスターノードの代わりに、タスクノードにスポットインスタンスを使用します。スポットインスタンスを使用する場合は、フリートごとに複数のインスタンスタイプのインスタンスフリートを使用して、スポットの可用性を確保します。
- 可能な場合はブロードキャスト結合を使用する – テーブルの 1 つが最小ノードのメモリに収まるほど小さく (MBs の順) 、等号 (=) 結合を実行している限り、ブロードキャスト (マップサイド) 結合が最も最適な結合です。完全外部結合を除くすべての結合タイプがサポートされています。ブロードキャスト結合は、メモリ内のすべてのワーカーノードに小さなテーブルをハッシュテーブルとしてブロードキャストします。小さなテーブルがブロードキャストされると、そのテーブルを変更することはできません。ハッシュテーブルは Java 仮想マシン (JVM) にローカルに存在するため、ハッシュ結合を使用して結合条件に基づいて大きなテーブルと簡単にマージできます。ブロードキャスト結合は、シャッフルオーバーヘッドが最小限であるため、高いパフォーマンスを提供します。
- ガベージコレクターを調整する – ガベージコレクション (GC) サイクルが遅い場合は、パフォーマンスを向上させるために、デフォルトの並列ガベージコレクターから G1GC に切り替えることを検討してください。GC のパフォーマンスを最適化するために、GC パラメータを微調整できます。GC のパフォーマンスを追跡するには、Spark UI を使用してモニタリングできます。理想的には、GC 時間はタスクランタイム全体の 1% 以下である必要があります。

での Iceberg の使用 AWS Glue

AWS Glue は、分析、機械学習 (ML)、アプリケーション開発のために複数のソースからのデータを簡単に検出、準備、移動、統合できるサーバーレスデータ統合サービスです。のコア機能の 1 つは、抽出、変換、ロード (ETL) オペレーションをシンプルで費用対効果の高い方法で実行する機能 AWS Glue です。これにより、データを分類し、クリーンアップして強化し、さまざまなデータストアとデータストリーム間で確実に移動できます。

AWS Glue ジョブは、Apache Sparkor Python runtime. AWS Glue jobs を使用して変換ロジックを定義するスクリプトをカプセル化します。ジョブは、バッチモードとストリーミングモードの両方で実行できます。

で Iceberg ジョブを作成する場合 AWS Glue、 のバージョンに応じて AWS Glue、ネイティブ Iceberg 統合またはカスタム Iceberg バージョンを使用して Iceberg 依存関係をジョブにアタッチできます。

ネイティブ Iceberg 統合の使用

AWS Glue バージョン 3.0、4.0、5.0 は、 AWS Glue for Spark の Apache Iceberg、Apache Hudi、Linux Foundation Delta Lake などのトランザクションデータレイク形式をネイティブにサポートしています。この統合機能により、これらのフレームワークの使用を開始するために必要な設定手順が簡素化されます AWS Glue。

AWS Glue ジョブの Iceberg サポートを有効にするには、ジョブを設定します。 AWS Glue ジョブの詳細タブを選択し、高度なプロパティのジョブパラメータにスクロールして、キーを に設定`--datalake-formats`し、その値を に設定します`iceberg`。

ノートブックを使用してジョブを作成する場合は、次のように `%%configure`マジックを使用して、最初のノートブックセルで パラメータを設定できます。

```
%%configure
{
  "--conf" : <job-specific Spark configuration discussed later>,
  "--datalake-formats" : "iceberg"
}
```

`--datalake-formats` の AWS Glue `iceberg`の設定は、バージョンに基づいて特定の Iceberg AWS Glue バージョンに対応しています。

AWS Glue バージョン	Iceberg のデフォルトバージョン
5.0	1.7.1
4.0	1.0.0
3.0	0.13.1

カスタム Iceberg バージョンの使用

状況によっては、ジョブの Iceberg バージョンに対するコントロールを保持し、自分のペースでアップグレードする場合があります。たとえば、新しいバージョンにアップグレードすると、新機能やパフォーマンスの強化へのアクセスを ロック解除できます。で特定の Iceberg バージョンを使用するには AWS Glue、独自の JAR ファイルを指定できます。

カスタム Iceberg バージョンを実装する前に、ドキュメントAWS Glue のバージョンセクション AWS Glue を確認して、環境との AWS Glue 互換性を確認してください。たとえば、 AWS Glue 5.0 には Spark 3.5.4 との互換性が必要です。

例えば、Iceberg バージョン 1.9.1 を使用する AWS Glue ジョブを実行するには、次の手順に従います。

1. 必要な JAR ファイルを取得して Amazon S3 にアップロードします。
   a. Apache Maven リポジトリから iceberg-spark-runtime-3.5_2.12-1.9.1.jar と iceberg-aws-bundle-1.9.1.jar をダウンロードします。
   b. これらのファイルを指定された S3 バケットの場所 (例:) にアップロードしますs3://your-bucket-name/jars/。
2. ジョブの AWS Glue ジョブパラメータを次のように設定します。
   a. --extra-jars パラメータで両方の JAR ファイルへの完全な S3 パスをカンマで区切って指定します (例: s3://your-bucket-name/jars/iceberg-spark-runtime-3.5_2.12-1.9.1.jar,s3://your-bucket-name/jars/iceberg-aws-bundle-1.9.1.jar) 。
   b. --datalake-formats パラメータの値として、iceberg を含めないようにしてください。
   c. AWS Glue 5.0 を使用する場合は、 --user-jars-firstパラメータを に設定する必要がありますtrue。

での Iceberg の Spark 設定 AWS Glue

このセクションでは、Iceberg データセットの AWS Glue ETL ジョブを作成するために必要な Spark 設定について説明します。これらの設定は、Spark `--conf` キーとすべての Spark 設定キーと値のカンマ区切りリストを使用して設定できます。ノートブックで `%%configure`マジックを使用するか、AWS Glue Studio コンソールの ジョブパラメータ セクションを使用できます。

```
%glue_version 5.0

%%configure
{
  "--conf" : "spark.sql.extensions=org.apache.iceberg.spark.extensions...",
  "--datalake-formats" : "iceberg"
}
```

次のプロパティを使用して Spark セッションを設定します。

- `<catalog_name>` は Iceberg Spark セッションカタログ名の名前です。任意の名前に置き換え、このカタログに関連付けられているすべての設定で参照を必ず変更してください。コードでは、次のように Spark セッションカタログ名を含む完全修飾テーブル名を使用して Iceberg テーブルを参照できます。

  ```
  <catalog_name>.<database_name>.<table_name>
  ```

  または、デフォルトカタログを、 をカタログ名に設定して定義した Iceberg カタログ`spark.sql.defaultCatalog`に変更することもできます。この 2 番目のアプローチを使用して、カタログプレフィックスのないテーブルを参照できるため、クエリを簡素化できます。
- `<catalog_name>.<warehouse>` は、データとメタデータを保存する Amazon S3 パスを指します。
- カタログを にするには AWS Glue Data Catalog、`spark.sql.catalog.<catalog_name>.type` を に設定しますglue。このキーは、カスタムカタログ実装の実装クラスを指すために必要です。Iceberg でサポートされているカタログについては、このガイドの後半にある「一般的なベストプラクティス」セクションを参照してください。

たとえば、 というカタログがある場合は`glue_iceberg`、次のように複数の`--conf`キーを使用してジョブを設定できます。

```
%%configure
```

```
{
  "--datalake-formats" : "iceberg",
  "--conf" :
 "spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions
 --conf spark.sql.catalog.glue_iceberg=org.apache.iceberg.spark.SparkCatalog --
conf spark.sql.catalog.glue_iceberg.warehouse=s3://<your-warehouse-dir>/ --conf
 spark.sql.catalog.glue_iceberg.type=glue"
}
```

または、次のようにコードを使用して、上記の設定を Spark スクリプトに追加することもできます。

```
spark = SparkSession.builder\

 .config("spark.sql.extensions","org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensi
                    .config("spark.sql.catalog.glue_iceberg",
 "org.apache.iceberg.spark.SparkCatalog")\
                    .config("spark.sql.catalog.glue_iceberg.warehouse","s3://<your-
warehouse-dir>/")\
                    .config("spark.sql.catalog.glue_iceberg.type", "glue") \
                    .getOrCreate()
```

AWS Glue ジョブのベストプラクティス

このセクションでは、Iceberg テーブルへのデータの読み取りと書き込みを最適化 AWS Glue するために、 で Spark ジョブを調整するための一般的なガイドラインを提供します。Iceberg 固有のベストプラクティスについては、このガイドの後半にある「ベストプラクティス」セクションを参照してください。

- の最新バージョンを使用し AWS Glue 、可能な限りアップグレードする – の新しいバージョン AWS Glue では、パフォーマンスの向上、起動時間の短縮、新機能が提供されます。また、最新の Iceberg バージョンに必要な新しい Spark バージョンもサポートしています。利用可能な AWS Glue バージョンとそれらがサポートする Spark バージョンのリストについては、 AWS Glue ドキュメントを参照してください。
- AWS Glue ジョブメモリの最適化 – ブログ記事「 のメモリ管理の最適化 AWS Glue」の推奨事項に従ってください。 AWS
- Use AWS Glue Auto Scaling – Auto Scaling を有効にすると、 はワークロードに基づいて AWS Glue ワーカーの数 AWS Glue を自動的に調整します。これにより、ワークロードが小さく、ワーカーがアイドル状態のときにワーカー数がスケールダウンされるため AWS Glue 、ピーク負荷時

の AWS Glue ジョブのコストを削減できます。 AWS Glue Auto Scaling を使用するには、ジョブをスケーリングできる AWS Glue ワーカーの最大数を指定します。詳細については、 ドキュメントの AWS Glue 「 の自動スケーリングAWS Glue の使用」を参照してください。

- 目的の Iceberg バージョンを使用する – Iceberg の AWS Glue ネイティブ統合は、Iceberg の使用を開始するのに最適です。ただし、本番環境のワークロードでは、ライブラリの依存関係 (このガイドで前述) を追加して、Iceberg バージョンを完全に制御することをお勧めします。このアプローチは、ジョブにおける最新の Iceberg 機能とパフォーマンスの向上のメリットを享受する AWS Glue のに役立ちます。
- モニタリングとデバッグのために Spark UI を有効にする – の Spark UI AWS Glue を使用して、Spark ジョブのさまざまなステージを有向非巡回グラフ (DAG) で視覚化し、ジョブを詳細にモニタリングすることで、Iceberg ジョブを検査することもできます。Spark UI は、Iceberg ジョブのトラブルシューティングと最適化の両方に効果的な方法を提供します。例えば、シャッフルやディスクスピルが大きいボトルネックステージを特定して、調整の機会を特定できます。詳細については、 ドキュメントの「Apache Spark ウェブ UI を使用したジョブのモニタリング AWS Glue 」を参照してください。

Apache Spark を使用した Iceberg テーブルの操作

このセクションでは、Apache Spark を使用して Iceberg テーブルを操作する方法の概要を説明します。例は、Amazon EMR または で実行できる定型コードです AWS Glue。

注: Iceberg テーブルを操作するためのプライマリインターフェイスは SQL であるため、ほとんどの例では Spark SQL と DataFrames API が組み合わされます。

Iceberg テーブルの作成と書き込み

Spark SQL と Spark DataFrames を使用して、Iceberg テーブルにデータを作成して追加できます。

Spark SQL の使用

Iceberg データセットを記述するには、 `CREATE TABLE`や などの標準の Spark SQL ステートメントを使用します`INSERT INTO`。

パーティション分割されていないテーブル

Spark SQL を使用してパーティション分割されていない Iceberg テーブルを作成する例を次に示します。

```
spark.sql(f"""
    CREATE TABLE IF NOT EXISTS {CATALOG_NAME}.{DB_NAME}.{TABLE_NAME}_nopartitions (
        c_customer_sk             int,
        c_customer_id             string,
        c_first_name              string,
        c_last_name               string,
        c_birth_country           string,
        c_email_address           string)
    USING iceberg
    OPTIONS ('format-version'='2')
""")
```

パーティション化されていないテーブルにデータを挿入するには、標準`INSERT INTO`ステートメントを使用します。

```
spark.sql(f"""
INSERT INTO {CATALOG_NAME}.{DB_NAME}.{TABLE_NAME}_nopartitions
SELECT c_customer_sk, c_customer_id, c_first_name, c_last_name, c_birth_country,
 c_email_address
```

```
FROM another_table
""")
```

パーティションテーブル

Spark SQL を使用してパーティション化された Iceberg テーブルを作成する例を次に示します。

```
spark.sql(f"""
    CREATE TABLE IF NOT EXISTS {CATALOG_NAME}.{DB_NAME}.{TABLE_NAME}_withpartitions (
        c_customer_sk             int,
        c_customer_id             string,
        c_first_name              string,
        c_last_name               string,
        c_birth_country           string,
        c_email_address           string)
    USING iceberg
    PARTITIONED BY (c_birth_country)
    OPTIONS ('format-version'='2')
""")
```

Spark SQL を使用してパーティション化された Iceberg テーブルにデータを挿入するには、標準`INSERT INTO`ステートメントを使用します。

```
spark.sql(f"""
INSERT INTO {CATALOG_NAME}.{DB_NAME}.{TABLE_NAME}_withpartitions
SELECT c_customer_sk, c_customer_id, c_first_name, c_last_name, c_birth_country,
 c_email_address
FROM another_table
""")
```

> **Note**
>
> Iceberg 1.5.0 以降では、パーティションテーブルにデータを挿入する場合、`hash`書き込み分散モードがデフォルトになります。詳細については、Iceberg ドキュメントの「Writing Distribution Modes」を参照してください。

DataFrames API の使用

Iceberg データセットを記述するには、 `DataFrameWriterV2` API を使用できます。

Iceberg テーブルを作成し、そのテーブルにデータを書き込むには、df.writeTo(t) 関数を使用します。テーブルが存在する場合は、 .append()関数を使用します。そうでない場合は、.create().次の例で を使用します。これは .createOrReplace().create()に相当するのバリエーションですCREATE OR REPLACE TABLE AS SELECT。

パーティション分割されていないテーブル

DataFrameWriterV2 API を使用してパーティション分割されていない Iceberg テーブルを作成して入力するには:

```
input_data.writeTo(f"{CATALOG_NAME}.{DB_NAME}.{TABLE_NAME}_nopartitions") \
    .tableProperty("format-version", "2") \
    .createOrReplace()
```

DataFrameWriterV2 API を使用して 、パーティション分割されていない既存の Iceberg テーブルにデータを挿入するには:

```
input_data.writeTo(f"{CATALOG_NAME}.{DB_NAME}.{TABLE_NAME}_nopartitions") \
    .append()
```

パーティションテーブル

DataFrameWriterV2 API を使用してパーティション化された Iceberg テーブルを作成して入力するには:

```
input_data.writeTo(f"{CATALOG_NAME}.{DB_NAME}.{TABLE_NAME}_withpartitions") \
    .tableProperty("format-version", "2") \
    .partitionedBy("c_birth_country") \
    .createOrReplace()
```

DataFrameWriterV2 API を使用してパーティション化された Iceberg テーブルにデータを挿入するには:

```
input_data.writeTo(f"{CATALOG_NAME}.{DB_NAME}.{TABLE_NAME}_withpartitions") \
    .append()
```

Iceberg テーブルのデータの更新

次の例は、Iceberg テーブルのデータを更新する方法を示しています。この例では、c_customer_sk列に偶数を持つすべての行を変更します。

```
spark.sql(f"""
UPDATE {CATALOG_NAME}.{db.name}.{table.name}
SET c_email_address = 'even_row'
WHERE c_customer_sk % 2 == 0
""")
```

このオペレーションはデフォルトのcopy-on-write戦略を使用するため、影響を受けるすべてのデータファイルを書き換えます。

Iceberg テーブルのデータの更新

データの更新とは、新しいデータレコードを挿入し、既存のデータレコードを 1 回のトランザクションで更新することです。Iceberg テーブルにデータをアップサートするには、 SQL MERGE INTOステートメントを使用します。

次の例では、テーブル 内のテーブル {UPSERT_TABLE_NAME} の内容をアップサートします{TABLE_NAME}。

```
spark.sql(f"""
    MERGE INTO {CATALOG_NAME}.{DB_NAME}.{TABLE_NAME} t
    USING {UPSERT_TABLE_NAME} s
        ON t.c_customer_id = s.c_customer_id
    WHEN MATCHED THEN UPDATE SET t.c_email_address = s.c_email_address
    WHEN NOT MATCHED THEN INSERT *
""")
```

- にある顧客レコードが同じ {TABLE_NAME}に{UPSERT_TABLE_NAME}既に存在する場合c_customer_id、{UPSERT_TABLE_NAME}レコードc_email_address値は既存の値を上書きします (更新オペレーション) 。
- にある顧客レコード{UPSERT_TABLE_NAME}が に存在しない場合{TABLE_NAME}、{UPSERT_TABLE_NAME}レコードは {TABLE_NAME} (オペレーションを挿入) に追加されます。

Iceberg テーブルのデータの削除

Iceberg テーブルからデータを削除するには、 DELETE FROM式を使用して、削除する行に一致するフィルターを指定します。

```
spark.sql(f"""
DELETE FROM {CATALOG_NAME}.{db.name}.{table.name}
WHERE c_customer_sk % 2 != 0
""")
```

フィルターがパーティション全体と一致する場合、Iceberg はメタデータのみの削除を実行し、データファイルを配置したままにします。それ以外の場合は、影響を受けるデータファイルのみを書き換えます。

delete メソッドは、 WHERE句の影響を受けるデータファイルを取得し、削除されたレコードなしでそれらのコピーを作成します。次に、新しいデータファイルを指す新しいテーブルスナップショットを作成します。したがって、削除されたレコードはテーブルの古いスナップショットにまだ存在します。たとえば、テーブルの前のスナップショットを取得すると、先ほど削除したデータが表示されます。クリーンアップの目的で関連データファイルを使用して不要な古いスナップショットを削除する方法については、このガイドの後半の「圧縮を使用してファイルを維持する」セクションを参照してください。

データの読み込み

Spark SQL と DataFrames の両方で、Spark の Iceberg テーブルの最新ステータスを読み取ることができます。

Spark SQL の使用例:

```
spark.sql(f"""
SELECT * FROM {CATALOG_NAME}.{db.name}.{table.name} LIMIT 5
""")
```

DataFrames API の使用例:

```
df = spark.table(f"{CATALOG_NAME}.{DB_NAME}.{TABLE_NAME}").limit(5)
```

タイムトラベルの使用

Iceberg テーブルの各書き込みオペレーション (挿入、更新、アップサート、削除) は、新しいスナップショットを作成します。その後、これらのスナップショットをタイムトラベルに使用できます。タイムトラベルをさかのぼって、過去のテーブルのステータスを確認できます。

`snapshot-id` およびタイミング値を使用してテーブルのスナップショットの履歴を取得する方法については、このガイドの後半にある「メタデータへのアクセス」セクションを参照してください。

次のタイムトラベルクエリは、特定の に基づいてテーブルのステータスを表示します `snapshot-id`。

Spark SQL の使用:

```
spark.sql(f"""
SELECT * FROM {CATALOG_NAME}.{DB_NAME}.{TABLE_NAME} VERSION AS OF {snapshot_id}
""")
```

DataFrames API の使用:

```
df_1st_snapshot_id = spark.read.option("snapshot-id", snapshot_id) \
     .format("iceberg") \
     .load(f"{CATALOG_NAME}.{DB_NAME}.{TABLE_NAME}") \
     .limit(5)
```

次のタイムトラベルクエリは、特定のタイムスタンプの前に作成された最後のスナップショットに基づいて、テーブルのステータスをミリ秒 () 単位で表示しますas-of-timestamp。

Spark SQL の使用:

```
spark.sql(f"""
SELECT * FROM dev.{db.name}.{table.name} TIMESTAMP AS OF '{snapshot_ts}'
""")
```

DataFrames API の使用:

```
df_1st_snapshot_ts = spark.read.option("as-of-timestamp", snapshot_ts) \
                          .format("iceberg") \
                          .load(f"dev.{DB_NAME}.{TABLE_NAME}") \
                          .limit(5)
```

増分クエリの使用

Iceberg スナップショットを使用して、追加したデータを段階的に読み取ることもできます。

注： 現在、このオペレーションはappendスナップショットからのデータの読み取りをサポートしています。replace、、 overwriteなどのオペレーションからのデータの取得はサポートされていませんdelete。 さらに、増分読み取りオペレーションは Spark SQL 構文ではサポートされていません。

次の の例では、スナップショット start-snapshot-id (排他的) と end-snapshot-id (包括的) の間に Iceberg テーブルに追加されたすべてのレコードを取得します。

```
df_incremental = (spark.read.format("iceberg")
    .option("start-snapshot-id", snapshot_id_start)
    .option("end-snapshot-id", snapshot_id_end)
    .load(f"glue_catalog.{DB_NAME}.{TABLE_NAME}")
)
```

メタデータへのアクセス

Iceberg は SQL を介してメタデータへのアクセスを提供します。名前空間 をクエリすることで、任意のテーブル (<table_name>) のメタデータにアクセスできます<table_name>.<metadata_table>。メタデータテーブルの完全なリストについては、Iceberg ドキュメントの「テーブルの検査」を参照してください。

次の例は、Iceberg テーブルのコミット (変更) の履歴を示す Iceberg 履歴メタデータテーブルにアクセスする方法を示しています。

Amazon EMR Studio ノートブックからの Spark SQL (%%sqlマジックを使用) の使用:

```
Spark.sql(f"""
SELECT * FROM {CATALOG_NAME}.{DB_NAME}.{TABLE_NAME}.history LIMIT 5
""")
```

DataFrames API の使用:

```
spark.read.format("iceberg").load("{CATALOG_NAME}.{DB_NAME}.
{TABLE_NAME}.history").show(5,False)
```

サンプル出力:

Type: Table Pie Scatter Line Area Bar

made_current_at	snapshot_id	parent_id	is_current_ancestor
2023-01-09 02:50:17.547000+00:00	7501027970051178613	6598755163776233735	True
2023-01-12 05:39:29.567000+00:00	7069175828427777019	7501027970051178613	True
2023-01-12 05:39:58.807000+00:00	5173022175861138222	7069175828427777019	True
2023-01-12 05:40:18.499000+00:00	3703414997660223390	5173022175861138222	True
2023-01-12 05:40:41.827000+00:00	3807904412292252460	3703414997660223390	True

Trino を使用した Iceberg テーブルの操作

このセクションでは、Amazon EMR で Trino を使用して Iceberg テーブルをセットアップおよび操作する方法について説明します。例は、EC2 クラスター上の Amazon EMR で実行できる定型コードです。このセクションのコード例と設定は、Amazon EMR リリース emr-7.9.0 を使用していることを前提としています。

EC2 での Amazon EMR のセットアップ

1. 以下のコンテンツを含む `iceberg.properties` ファイルを作成します。`iceberg.file-format=parquet` 設定は、 `CREATE TABLE` ステートメントで形式が明示的に指定されていない場合に、新しいテーブルのデフォルトのストレージ形式を決定します。

```
connector.name=iceberg
iceberg.catalog.type=glue
iceberg.file-format=parquet
fs.native-s3.enabled=true
```

2. `iceberg.properties` ファイルを S3 バケットにアップロードします。
3. S3 バケットから`iceberg.properties` ファイルをコピーし、作成する Amazon EMR クラスターに Trino 設定ファイルとして保存するブートストラップアクションを作成します。 を S3 バケット名<S3-bucket-name> に置き換えてください。

```
#!/bin/bash
set -ex
sudo aws s3 cp s3://<S3-bucket-name>/iceberg.properties /etc/trino/conf/catalog/
iceberg.properties
```

4. Trino がインストールされた Amazon EMR クラスターを作成し、ブートストラップアクションとして前のスクリプトの実行を指定します。クラスターを作成するための sample AWS Command Line Interface (AWS CLI) コマンドを次に示します。

```
aws emr create-cluster --release-label emr-7.9.0 \
--applications Name=Trino \
--region <region> \
--name Trino_Iceberg_Cluster \
--bootstrap-actions '[{"Path":"s3://<S3-bucket-name>/bootstrap.sh","Name":"Add
 iceberg.properties"}]' \
```

```
--instance-groups
 '[{"InstanceGroupType":"MASTER","InstanceCount":1,"InstanceType":"m5.xlarge"},
{"InstanceGroupType":"CORE","InstanceCount":3,"InstanceType":"m5.xlarge"}]' \
--service-role "<IAM-service-role>" \
--ec2-attributes '{"KeyName":"<key-name>","InstanceProfile":"<EMR-EC2-instance-
profile>"}'
```

を置き換える場所:

- <S3-bucket-name> S3 バケット名
- <region> 特定の AWS リージョン
- <key-name> キーペアを使用します。キーペアが存在しない場合は作成されます。
- <IAM-service-role> 最小特権の原則に従う Amazon EMR サービスロール。
- <EMR-EC2-instance-profile> インスタンスプロファイルを使用します。

5. Amazon EMR クラスターが初期化されたら、次のコマンドを実行して Trino セッションを初期化できます。

```
trino-cli
```

6. Trino CLI では、以下を実行してカタログを表示できます。

```
SHOW CATALOGS;
```

Iceberg テーブルの作成

Iceberg テーブルを作成するには、 CREATE TABLEステートメントを使用できます。 Iceberg の非表示パーティショニングを使用するパーティションテーブルを作成する例を次に示します。

```
CREATE TABLE iceberg.iceberg_db.iceberg_table (
              userid int,
              firstname varchar,
              city varchar)
        WITH (
              format = 'PARQUET',
              partitioning = ARRAY['city', 'bucket(userid, 16)'],
              location = 's3://<S3-bucket>/<prefix>');
```

> **Note**
>
> 形式を指定しない場合、前のセクションで設定したiceberg.file-format値が使用されます。

データを挿入するには、 INSERT INTO コマンドを使用します。例を示します。

```
INSERT INTO iceberg.iceberg_db.iceberg_table (userid, firstname, city)
VALUES
    (1001, 'John', 'New York'),
    (1002, 'Mary', 'Los Angeles'),
    (1003, 'Mateo', 'Chicago'),
    (1004, 'Shirley', 'Houston'),
    (1005, 'Diego', 'Miami'),
    (1006, 'Nikki', 'Seattle'),
    (1007, 'Pat', 'Boston'),
    (1008, 'Terry', 'San Francisco'),
    (1009, 'Richard', 'Denver'),
    (1010, 'Pat', 'Phoenix');
```

Iceberg テーブルからの読み取り

次のように、 SELECTステートメントを使用して Iceberg テーブルの最新のステータスを読み取ることができます。

```
SELECT * FROM iceberg.iceberg_db.iceberg_table;
```

Iceberg テーブルへのデータの更新

MERGE INTO ステートメントを使用して、アップサートオペレーション (同時に新しいレコードを挿入し、既存のレコードを更新) を実行できます。例を示します。

```
MERGE INTO iceberg.iceberg_db.iceberg_table target
USING (
      VALUES
        (1001, 'John Updated', 'Boston'),       -- Update existing user
        (1002, 'Mary Updated', 'Seattle'),      -- Update existing user
        (1011, 'Martha', 'Portland'),           -- Insert new user
```

```
        (1012, 'Paulo', 'Austin')               -- Insert new user
    ) AS source (userid, firstname, city)
  ON target.userid = source.userid
WHEN MATCHED THEN
     UPDATE SET
       firstname = source.firstname,
       city = source.city
WHEN NOT MATCHED THEN
     INSERT (userid, firstname, city)
     VALUES (source.userid, source.firstname, source.city);
```

Iceberg テーブルからのレコードの削除

Iceberg テーブルからデータを削除するには、 DELETE FROM式を使用して、削除する行に一致するフィルターを指定します。例を示します。

```
DELETE FROM iceberg.iceberg_db.iceberg_table WHERE userid IN (1003, 1004);
```

Iceberg テーブルデータのクエリの実行

Iceberg は SQL を介してメタデータへのアクセスを提供します。名前空間 をクエリすることで、任意のテーブル (<table_name>) のメタデータにアクセスできます"<table_name>.$<metadata_table>"。メタデータテーブルの完全なリストについては、Iceberg ドキュメントの「テーブルの検査」を参照してください。

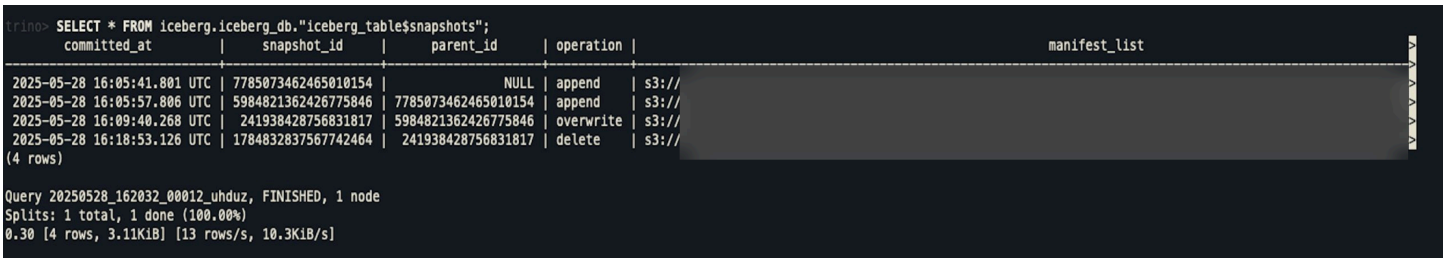
Iceberg メタデータを検査するクエリの例を次に示します。

```
SELECT  FROM iceberg.iceberg_db."iceberg_table$snapshots";
SELECT  FROM iceberg.iceberg_db."iceberg_table$history";
SELECT  FROM iceberg.iceberg_db."iceberg_table$partitions";
SELECT  FROM iceberg.iceberg_db."iceberg_table$files";
SELECT  FROM iceberg.iceberg_db."iceberg_table$manifests";
SELECT  FROM iceberg.iceberg_db."iceberg_table$refs";
SELECT * FROM iceberg.iceberg_db."iceberg_table$metadata_log_entries";
```

例えば、次のクエリは、

```
 SELECT * FROM iceberg.iceberg_db."iceberg_table$snapshots";
```

は出力を提供します。



タイムトラベルの使用

Iceberg テーブル内の書き込みオペレーション (挿入、更新、アップサート、または削除) ごとに、新しいスナップショットが作成されます。その後、これらのスナップショットをタイムトラベルに使用できます。タイムトラベルをさかのぼって、過去のテーブルのステータスを確認できます。

次のタイムトラベルクエリは、特定の に基づいてテーブルのステータスを表示しますsnapshot_id。

```
SELECT *
FROM iceberg.iceberg_db.iceberg_table FOR VERSION AS OF 241938428756831817;
```

次のタイムトラベルクエリは、特定のタイムスタンプに基づいてテーブルのステータスを表示します。

```
SELECT *
FROM iceberg.iceberg_db.iceberg_table FOR TIMESTAMP AS OF TIMESTAMP '2025-05-28
 16:09:40.268 UTC'
```

Trino で Iceberg を使用する場合の考慮事項

Iceberg テーブルに対する Trino 書き込みオペレーションはmerge-on-read 設計に従うため、更新または削除の影響を受けるデータファイル全体を書き換えるのではなく、位置削除ファイルを作成します。copy-on-write方式を使用する場合は、書き込みオペレーションに Spark を使用することを検討してください。

Amazon Data Firehose を使用した Iceberg テーブルの操作

Amazon Data Firehose は、 AWS WAF ログ、Amazon CloudWatch Logs、Amazon Kinesis Data Streams AWS IoT、Amazon Managed Streaming for Apache Kafka (Amazon MSK) などの 20 を超えるソースから Amazon S3、Amazon Redshift、Snowflake、Splunk などの宛先にデータストリームを配信するためのサーバーレスのノーコードサービスです。

Firehose を使用して、ストリーミングデータを Amazon S3 の Apache Iceberg テーブルに直接配信できます。Firehose を使用すると、単一のストリームから異なる Apache Iceberg テーブルにレコードをルーティングし、テーブル内のレコードに挿入、更新、削除オペレーションを自動的に適用できます。Firehose は、Iceberg テーブルへの 1 回限りの配信を保証します。この機能を使用するには、AWS Glue Data Catalogを使用する必要があります。

Firehose は、ストリーミングデータを Amazon S3 テーブルに直接配信することもできます。これらのテーブルは、大規模な分析ワークロードに最適化されたストレージを提供し、クエリのパフォーマンスを継続的に改善し、表形式のデータのストレージコストを削減する機能が含まれています。

Apache Iceberg テーブルにデータを配信するように Firehose ストリームを設定する方法については、Firehose ドキュメントの「Firehose ストリームを設定する」またはブログ記事「Amazon Data Firehose を使用して Amazon S3 の Apache Iceberg テーブルにリアルタイムデータをストリーミングする」を参照してください。

Athena SQL を使用した Iceberg テーブルの操作

Amazon Athena は Apache Iceberg の組み込みサポートを提供し、追加のステップや設定は必要ありません。このセクションでは、サポートされている機能の詳細と、Athena を使用して Iceberg テーブルを操作するための大まかなガイダンスについて説明します。

バージョンと機能の互換性

Iceberg テーブル仕様のサポート

Apache Iceberg テーブル仕様は、Iceberg テーブルの動作を指定します。Athena はテーブル形式バージョン 2 をサポートしているため、コンソール、CLI、または SDK で作成した Iceberg テーブルは、本質的にそのバージョンを使用します。

Amazon EMR の Apache Spark や などの別のエンジンで作成された Iceberg テーブルを使用する場合は AWS Glue、テーブルプロパティを使用してテーブル形式バージョンを設定してください。参考として、このガイドの前半の「Iceberg テーブルの作成と書き込み」セクションを参照してください。

Iceberg 機能のサポート

Athena を使用して Iceberg テーブルの読み取りと書き込みを行うことができます。`UPDATE`、、および `DELETE FROM`ステートメントを使用してデータを変更する`MERGE INTO`と、Athena はmerge-on-readモードのみをサポートします。このプロパティは変更できません。copy-on-write でデータを更新または削除するには、Amazon EMR の Apache Spark や などの他のエンジンを使用する必要があります AWS Glue。次の表は、Athena での Iceberg 機能のサポートをまとめたものです。

		DDL サポート		DML サポート		AWS Lake Formation セキュリティ用 (オプション)
	テーブル形式	テーブルの作成	スキーマ進化	データの読み込み	データの書き込み	行/列のアクセスコントロール

		DDL サポート		DML サポート		AWS Lake Formation セキュリティ用 (オプション)
Amazon Athena	バージョン 2	✓	✓	✓	X Copy-on-write	✓
					Merge-on-read	✓

> **Note**
>
> - Athena は増分クエリをサポートしていません。
> - Athena では、CoW がサポートされていないため、テーブルプロパティの書き込み時コピー (CoW) 設定に関係なく、更新、削除、マージオペレーションは常にデフォルトで読み取り時マージ (MoR) CoW になります。

Iceberg テーブルの操作

Athena で Iceberg を使用するためのクイックスタートについては、このガイドの前半の「Athena SQL での Iceberg テーブルの開始方法」セクションを参照してください。

次の表に、制限と推奨事項を示します。

シナリオ	制限	レコメンデーション
テーブル DDL 生成	他のエンジンで作成された Iceberg テーブルには、Athena で公開されていないプロパティを含めることができます。これらのテーブルでは、DDL を生成できません。	テーブルを作成したエンジンで同等のステートメント (Spark の `SHOW CREATE TABLE`ステートメントなど) を使用します。

シナリオ	制限	レコメンデーション
Iceberg テーブルに書き込まれるオブジェクトのランダム Amazon S3 プレフィックス	デフォルトでは、Athena で作成された Iceberg テーブルでは、 `write.object-storage.enabled` プロパティが有効になっています。	この動作を無効にして Iceberg テーブルプロパティを完全に制御するには、Amazon EMR の Spark や などの別のエンジンで Iceberg テーブルを作成します AWS Glue。
増分クエリ	Athena では現在サポートされていません。	増分クエリを使用して増分データ取り込みパイプラインを有効にするには、Amazon EMR または で Spark を使用します AWS Glue。

PyIceberg を使用した Iceberg テーブルの操作

このセクションでは、PyIceberg を使用して Iceberg テーブルを操作する方法について説明します。提供されている例は、Amazon Linux 2023 EC2 インスタンス、AWS Lambda関数、または適切に設定されたAWS 認証情報を持つ任意の Python 環境で実行できる定型コードです。

前提条件

> **Note**
>
> これらの例ではPyIceberg 1.9.1 を使用しています。

PyIceberg を使用するには、PyIceberg と AWS SDK for Python (Boto3) がインストールされている必要があります。PyIceberg と で動作するように Python 仮想環境を設定する方法の例を次に示します AWS Glue Data Catalog。

1. pip python パッケージインストーラを使用して PyIceberg をダウンロードします。Boto3 の操作も必要です AWS のサービス。次のコマンドを使用して、テストするローカル Python 仮想環境を設定できます。

```
python3 -m venv my_env
cd my_env/bin/
source activate
pip install "pyiceberg[pyarrow,pandas,glue]"
pip install boto3
```

2. を実行してpython Python シェルを開き、コマンドをテストします。

データカタログへの接続

で Iceberg テーブルの操作を開始するには AWS Glue、まず に接続する必要があります AWS Glue Data Catalog。

このload_catalog関数は、すべての Iceberg オペレーションのプライマリインターフェイスとして機能するカタログオブジェクトを作成して、データカタログへの接続を初期化します。

```
from pyiceberg.catalog import load_catalog
```

```
region = "us-east-1"

glue_catalog = load_catalog(
    'default',
    **{
        'client.region': region
    },
    type='glue'
)
```

データベースの一覧表示と作成

既存のデータベースを一覧表示するには、 `list_namespaces`関数を使用します。

```
databases = glue_catalog.list_namespaces()
print(databases)
```

新しいデータベースを作成するには、 `create_namespace`関数を使用します。

```
database_name="mydb"
s3_db_path=f"s3://amzn-s3-demo-bucket/{database_name}"

glue_catalog.create_namespace(database_name, properties={"location": s3_db_path})
```

Iceberg テーブルの作成と書き込み

パーティション化されていないテーブル

`create_table` 関数を使用してパーティション分割されていない Iceberg テーブルを作成する例を次に示します。

```
from pyiceberg.schema import Schema
from pyiceberg.types import NestedField, StringType, DoubleType

database_name="mydb"
table_name="pyiceberg_table"
s3_table_path=f"s3://amzn-s3-demo-bucket/{database_name}/{table_name}"

schema = Schema(
```

```
    NestedField(1, "city", StringType(), required=False),
    NestedField(2, "lat", DoubleType(), required=False),
    NestedField(3, "long", DoubleType(), required=False),
)

glue_catalog.create_table(f"{database_name}.{table_name}", schema=schema,
 location=s3_table_path)
```

`list_tables` 関数を使用して、データベース内のテーブルのリストを確認できます。

```
tables = glue_catalog.list_tables(namespace=database_name)
print(tables)
```

関数と PyArrow を使用してappend、Iceberg テーブル内にデータを挿入できます。

```
import pyarrow as pa
df = pa.Table.from_pylist(
    [
        {"city": "Amsterdam", "lat": 52.371807, "long": 4.896029},
        {"city": "San Francisco", "lat": 37.773972, "long": -122.431297},
        {"city": "Drachten", "lat": 53.11254, "long": 6.0989},
        {"city": "Paris", "lat": 48.864716, "long": 2.349014},
    ],
)

table = glue_catalog.load_table(f"{database_name}.{table_name}")
table.append(df)
```

パーティションテーブル

`create_table` 関数 と を使用して、パーティション分割を非表示にしてパーティション分割された Iceberg テーブルを作成する例を次に示しますPartitionSpec。 https://iceberg.apache.org/docs/1.4.0/partitioning/#icebergs-hidden-partitioning

```
from pyiceberg.schema import Schema
from pyiceberg.types import (
    NestedField,
    StringType,
    FloatType,
    DoubleType,
    TimestampType,
```

```
)

# Define the schema
schema = Schema(
    NestedField(field_id=1, name="datetime", field_type=TimestampType(),
 required=True),
    NestedField(field_id=2, name="drone_id", field_type=StringType(), required=True),
    NestedField(field_id=3, name="lat", field_type=DoubleType(), required=False),
    NestedField(field_id=4, name="lon", field_type=DoubleType(), required=False),
    NestedField(field_id=5, name="height", field_type=FloatType(), required=False),
)

from pyiceberg.partitioning import PartitionSpec, PartitionField
from pyiceberg.transforms import DayTransform

partition_spec = PartitionSpec(
    PartitionField(
        source_id=1,  # Refers to "datetime"
        field_id=1000,
        transform=DayTransform(),
        name="datetime_day"
    )
)

database_name="mydb"
partitioned_table_name="pyiceberg_table_partitioned"
s3_table_path=f"s3://amzn-s3-demo-bucket/{database_name}/{partitioned_table_name}"

glue_catalog.create_table(
    identifier=f"{database_name}.{partitioned_table_name}",
    schema=schema,
    location=s3_table_path,
    partition_spec=partition_spec
)
```

パーティション分割されていないテーブルの場合と同じ方法で、パーティション分割されたテーブルにデータを挿入できます。パーティショニングは自動的に処理されます。

```
from datetime import datetime
arrow_schema = pa.schema([
    pa.field("datetime", pa.timestamp("us"), nullable=False),
    pa.field("drone_id", pa.string(), nullable=False),
    pa.field("lat", pa.float64()),
```

```
    pa.field("lon", pa.float64()),
    pa.field("height", pa.float32()),
])

data = [
    {
        "datetime": datetime(2024, 6, 1, 12, 0, 0),
        "drone_id": "drone_001",
        "lat": 52.371807,
        "lon": 4.896029,
        "height": 120.5,
    },
    {
        "datetime": datetime(2024, 6, 1, 12, 5, 0),
        "drone_id": "drone_002",
        "lat": 37.773972,
        "lon": -122.431297,
        "height": 150.0,
    },
    {
        "datetime": datetime(2024, 6, 2, 9, 0, 0),
        "drone_id": "drone_001",
        "lat": 53.11254,
        "lon": 6.0989,
        "height": 110.2,
    },
    {
        "datetime": datetime(2024, 6, 2, 9, 30, 0),
        "drone_id": "drone_003",
        "lat": 48.864716,
        "lon": 2.349014,
        "height": 145.7,
    },
]

df = pa.Table.from_pylist(data, schema=arrow_schema)

table = glue_catalog.load_table(f"{database_name}.{partitioned_table_name}")
table.append(df)
```

データの読み込み

PyIceberg `scan`関数を使用して Iceberg テーブルからデータを読み取ることができます。行をフィルタリングし、特定の列を選択し、返されるレコードの数を制限できます。

```
table= glue_catalog.load_table(f"{database_name}.{table_name}")
scan_df = table.scan(
    row_filter=(
        f"city = 'Amsterdam'"
    ),
    selected_fields=("city", "lat"),
    limit=100,
).to_pandas()

print(scan_df)
```

データの削除

PyIceberg `delete`関数を使用すると、 を使用してテーブルからレコードを削除できますdelete_filter。

```
table = glue_catalog.load_table(f"{database_name}.{table_name}")
table.delete(delete_filter="city == 'Paris'")
```

メタデータへのアクセス

PyIceberg には、テーブルメタデータにアクセスするための関数がいくつか用意されています。テーブルスナップショットに関する情報を表示する方法は次のとおりです。

```
#List of snapshots
table.snapshots()

#Current snapshot
table.current_snapshot()

#Take a previous snapshot
second_last_snapshot_id=table.snapshots()[-2].snapshot_id
print(f"Second last SnapshotID: {second_last_snapshot_id}")
```

使用可能なメタデータの詳細なリストについては、PyIceberg ドキュメントのメタデータコードリファレンスセクションを参照してください。

タイムトラベルの使用

タイムトラベルにテーブルスナップショットを使用して、テーブルの以前の状態にアクセスできます。最後のオペレーションの前にテーブルの状態を表示する方法は次のとおりです。

```
second_last_snapshot_id=table.snapshots()[-2].snapshot_id

time_travel_df = table.scan(
    limit=100,
    snapshot_id=second_last_snapshot_id
).to_pandas()

print(time_travel_df)
```

使用可能な関数の完全なリストについては、PyIcebergPython API ドキュメントを参照してください。

Iceberg テーブル形式仕様バージョン 3 の使用

Apache Iceberg テーブル形式の仕様の最新バージョンはバージョン 3 です。このバージョンでは、ペタバイト規模のデータレイクを構築するための高度な機能が導入され、パフォーマンスが向上し、運用オーバーヘッドが削減されます。バージョン 2 で発生する一般的なパフォーマンスのボトルネック、特にバッチ更新とコンプライアンス削除オペレーションに関するボトルネックに対処します。

AWS では、Iceberg バージョン 3 仕様で定義されている削除ベクトルと行リネージュがサポートされています。これらの機能は、以下の Apache Spark で使用できます AWS のサービス。

AWS のサービス	バージョン 3 のサポート
Amazon EMR for Apache Spark	Amazon EMR リリース 7.12 以降
AWS Glue	はい
AWS Glue: Iceberg REST API、テーブルメンテナンス	はい
Amazon SageMaker Unified Studio ノートブック	はい
Amazon S3 Tables: Iceberg REST API、テーブルメンテナンス	はい
Amazon Athena (Trino)	いいえ

バージョン 3 の主な機能

削除ベクトルは、バージョン 2 で使用された位置削除ファイルを、Puffin ファイルとして保存された効率的なバイナリ形式に置き換えます。これにより、ランダムバッチ更新や一般データ保護規則 (GDPR) コンプライアンスの削除による書き込み増幅がなくなり、新しいデータを維持するオーバーヘッドが大幅に削減されます。高頻度の更新を処理する組織では、書き込みパフォーマンスがすぐに向上し、小さなファイルが少なくてストレージコストが削減されます。

行リネージュにより、行レベルで正確な変更追跡が可能になります。ダウンストリームシステムでは段階的に変更を処理できるため、データパイプラインが高速化され、変更データキャプチャ (CDC)

ワークフローのコンピューティングコストが削減されます。この組み込み機能により、カスタム変更追跡実装が不要になります。

バージョン互換性

バージョン 3 は、バージョン 2 テーブルとの下位互換性を維持します。AWS のサービスでは、バージョン 2 とバージョン 3 の両方のテーブルを同時にサポートしているため、次のことが可能になります。

- バージョン 2 とバージョン 3 の両方のテーブルでクエリを実行します。
- データ書き換えなしで、既存のバージョン 2 テーブルをバージョン 3 にアップグレードします。
- バージョン 2 とバージョン 3 のスナップショットにまたがるタイムトラベルクエリを実行します。
- スキーマの進化とテーブルバージョン間の非表示パーティショニングを使用します。

バージョン 3 の開始方法

前提条件

バージョン 3 のテーブルを使用する前に、以下があることを確認してください。

- 適切な AWS Identity and Access Management (IAM) アクセス許可 AWS アカウント を持つ 。
- 1 つ以上の AWS 分析サービス (Amazon EMR、 AWS Glue Amazon SageMaker Unified Studio ノートブック、または Amazon S3 Tables) へのアクセス。
- テーブルデータとメタデータを保存するための S3 バケット。
- 独自の Iceberg インフラストラクチャを構築する場合は、Amazon S3 Tables または汎用 S3 バケットの使用を開始するためのテーブルバケット。
- 設定済み AWS Glue カタログ。

バージョン 3 テーブルの作成

新しいテーブルの作成

新しい Iceberg バージョン 3 テーブルを作成するには、`format-version`テーブルプロパティを 3 に設定します。

Spark SQL の使用:

```
CREATE TABLE IF NOT EXISTS myns.orders_v3 (
    order_id bigint,
    customer_id string,
    order_date date,
    total_amount decimal(10,2),
    status string,
    created_at timestamp
)
USING iceberg
TBLPROPERTIES (
    'format-version' = '3'
)
```

バージョン 2 テーブルをバージョン 3 にアップグレードする

データを書き換え ることなく、既存のバージョン 2 テーブルをバージョン 3 にアトミックにアップグレードできます。

Spark SQL の使用:

```
ALTER TABLE myns.existing_table
SET TBLPROPERTIES ('format-version' = '3')
```

> **Important**
>
> バージョン 3 は一方向アップグレードです。テーブルをバージョン 2 からバージョン 3 にアップグレードした後は、標準オペレーションを通じてバージョン 2 にダウングレードすることはできません。

アップグレード中に何が起こるか:

- 新しいメタデータスナップショットがアトミックに作成されます。
- 既存の Parquet データファイルは再利用されます。
- 行の系統フィールドがテーブルメタデータに追加されます。

アップグレード後:

- 次の圧縮では、古いバージョン 2 の削除ファイルを削除します。
- 新しい変更では、バージョン 3 の削除ベクトルファイルが使用されます。

アップグレードでは、行系統変更追跡レコードの履歴バックフィルは実行されません。

削除ベクトルの有効化

更新、削除、マージの削除ベクトルを利用するには、書き込みモードを設定します。

Spark SQL の使用:

```
ALTER TABLE myns.orders_v3
SET TBLPROPERTIES ('format-version' = '3',
                   'write.delete.mode' = 'merge-on-read',
                   'write.update.mode' = 'merge-on-read',
                   'write.merge.mode' = 'merge-on-read'
                  )
```

これらの設定により、データファイル全体を書き換えるのではなく、更新、削除、マージオペレーションによって削除ベクトルファイルが作成されます。

変更の追跡に行リネージュを使用する

バージョン 3 では、行系統メタデータフィールドが自動的に追加され、変更を追跡します。

Spark SQL の使用:

```
# Query with parameter value provided
last_processed_sequence = 47

SELECT
    id,
    data,
    _row_id,
    _last_updated_sequence_number
FROM myns.orders_v3
WHERE _last_updated_sequence_number > :last_processed_sequence
```

_row_id フィールドは各行を一意に識別し、行が最後に変更された日時 _last_updated_sequence_numberを追跡します。これらのフィールドを使用して、次の操作を行います。

- 増分処理のために変更された行を特定します。
- コンプライアンスに関するデータ系統を追跡します。
- CDC パイプラインを最適化します。
- 変更のみを処理することで、コンピューティングコストを削減します。

バージョン 3 のベストプラクティス

バージョン 3 を使用するタイミング

次の場合は、バージョン 3 にアップグレードするか、バージョン 3 から開始することを検討してください。

- バッチ更新や削除を頻繁に実行します。
- GDPR またはコンプライアンス削除の要件を満たす必要があります。
- ワークロードには、高頻度のアップサートが含まれます。
- 効率的な CDC ワークフローが必要です。
- 小さなファイルからストレージコストを削減したい。
- より良い変更追跡機能が必要です。

書き込みパフォーマンスの最適化

- 更新が多いワークロードの削除ベクトルを有効にします。

```
SET TBLPROPERTIES (
'write.delete.mode' = 'merge-on-read',
'write.update.mode' = 'merge-on-read',
'write.merge.mode' = 'merge-on-read'
)
```

- 適切なファイルサイズを設定します。

```
SET TBLPROPERTIES (
'write.target-file-size-bytes' = '536870912'  — 512 MB
)
```

読み取りパフォーマンスの最適化

- 増分処理には行リネージュを使用します。
- タイムトラベルを使用して、コピーせずに履歴データにアクセスします。
- 統計収集を有効にして、クエリ計画を改善します。

移行戦略

バージョン 2 からバージョン 3 に移行する場合は、次のベストプラクティスに従ってください。

- まず非本番環境でテストして、アップグレードプロセスとパフォーマンスを検証します。
- アクティビティが少ない時間帯にアップグレードして、同時オペレーションへの影響を最小限に抑えます。
- 初期パフォーマンスをモニタリングし、アップグレード後のメトリクスを追跡します。
- 圧縮を実行して、アップグレード後に削除ファイルを統合します。
- バージョン 3 の機能を反映するようにチームのドキュメントを更新します。

互換性に関する考慮事項

- エンジンバージョン – テーブルにアクセスするすべてのエンジンがバージョン 3 をサポートしていることを確認します。
- サードパーティー製ツール – アップグレードする前に、ツールのバージョン 3 の互換性を確認してください。
- バックアップ戦略 – スナップショットベースの復旧手順をテストします。
- モニタリング – バージョン 3 固有のメトリクスのモニタリングダッシュボードを更新します。

トラブルシューティング

一般的な問題

エラー: "format-version 3 is not supported"

- エンジンバージョンがバージョン 3 をサポートしていることを確認します。 詳細については、このセクションの冒頭にある表を参照してください。

- カタログの互換性を確認します。
- の最新バージョンを使用していることを確認してください AWS のサービス。

アップグレード後のパフォーマンスの低下

- 圧縮圧縮の失敗がないことを確認します。詳細については、Amazon S3 ドキュメントの「S3 テーブルのログ記録とモニタリング」を参照してください。 Amazon S3
- 削除ベクトルが有効になっていることを確認します。次のプロパティを設定する必要があります。

```
SET TBLPROPERTIES (
'write.delete.mode' = 'merge-on-read',
'write.update.mode' = 'merge-on-read',
'write.merge.mode' = 'merge-on-read'
)
```

  テーブルのプロパティは、次のコードで確認できます。

```
DESCRIBE FORMATTED myns.orders_v3
```

- パーティション戦略を確認します。パーティショニングが過剰になると、ファイルが小さくなる可能性があります。次のクエリを実行して、テーブルの平均ファイルサイズを取得します。

```
SELECT avg(file_size_in_bytes) as avg_file_size_bytes
FROM myns.orders_v3.files
```

サードパーティー製ツールとの非互換性

- ツールがバージョン 3 仕様をサポートしていることを確認します。
- サポートされていないツールのバージョン 2 テーブルの管理を検討してください。
- バージョン 3 のサポートタイムラインについては、ツールベンダーにお問い合わせください。

ヘルプの利用

- AWS のサービス固有の問題については、 にお問い合わせくださいAWS サポート。
- Iceberg コミュニティからサポートを受けるには、Iceberg Slack チャネルを使用します。

- AWS のサービス を使用して分析ワークロードを管理する方法については、「 の分析 AWS」を参照してください。

料金

- Amazon EMR コンピューティングとストレージの料金
- Amazon SageMaker の料金
- AWS Glue ジョブ実行とデータカタログの料金
- S3 Tables ストレージとリクエストの料金

利用可能な状況

Iceberg テーブル形式の仕様バージョン 3 のサポートは、Amazon EMR AWS Glue、 AWS Glue Data Catalog、および S3 Tables が動作するすべての AWS リージョン で利用できます。利用可能なリージョンについて、詳しくは「AWS のサービス by Region」をご確認ください。

その他のリソース

- Apache Iceberg ドキュメント
- Apache Iceberg テーブル仕様
- Amazon S3 から S3 Tables への表形式データの移行に関するガイダンス
- チュートリアル: S3 テーブルの開始方法

既存のテーブルを Iceberg に移行する

このセクションでは、既存の Hive スタイルのテーブルを Iceberg 形式に移行することに重点を置いています。これは、Apache Parquet や Apache ORC などの従来の Hive 互換形式を使用するテーブルに適用されます。この情報は、Linux Foundation Delta Lake や Apache Hudi などの最新のテーブル形式を既に使用しているテーブルには適用されません。

現在の Hive スタイルのテーブルを Iceberg 形式に移行するには、インプレースまたはフルデータ移行を使用できます。

- インプレース移行は、既存のデータファイルの上に Iceberg のメタデータファイルを生成するプロセスです。
- フルデータ移行は Iceberg メタデータレイヤーを作成し、既存のデータファイルを元のテーブルから新しい Iceberg テーブルに書き換えます。

以下のセクションでは、実装に関するstep-by-stepの手順や考慮事項など、各移行方法の詳細な概要を説明します。これらの移行戦略の詳細については、Iceberg ドキュメントの「テーブル移行」セクションを参照してください。

インプレースおよびフルデータ移行方法の詳細を確認したら、次の 2 つの主要なセクションを参照して意思決定プロセスに役立ててください。

- 移行戦略を選択すると、一連の質問とシナリオを通じてガイダンスが提供され、特定の要件とユースケースに基づいて最適な移行アプローチを決定できます。
- 移行オプションの概要は、さまざまな移行オプションの主要な特性と考慮事項を比較する包括的な表を提供します。この表はクイックリファレンスガイドとして機能し、メソッド間の技術的なトレードオフを理解するのに役立つ機能比較を提供します。

インプレース移行

インプレース移行により、すべてのデータファイルを書き直す必要がなくなります。代わりに、Iceberg メタデータファイルが生成され、既存のデータファイルにリンクされます。この方法は通常、特に Parquet、Avro、ORC などの互換性のあるファイル形式を持つ大規模なデータセットやテーブルの場合、より高速でコスト効率が高くなります。

> **Note**
>
> Amazon S3 Tables への移行時にインプレース移行を使用することはできません。

Iceberg には、インプレース移行を実装するための 2 つの主要なオプションがあります。

- スナップショットプロシージャを使用して、ソーステーブルを変更せずに新しい Iceberg テーブルを作成します。詳細については、Iceberg ドキュメントの「スナップショットテーブル」を参照してください。
- 移行手順を使用して、ソーステーブルの代替として新しい Iceberg テーブルを作成します。詳細については、Iceberg ドキュメントの「テーブルの移行」を参照してください。この手順は Hive メタストア (HMS) で機能しますが、現在 と互換性がありません AWS Glue Data Catalog。このガイドの後半のセクションの「テーブル移行手順のレプリケート AWS Glue Data Catalog」は、データカタログと同様の結果を達成するための回避策を提供します。

`snapshot` または を使用してインプレース移行を実行した後`migrate`、一部のデータファイルは移行されないままになる場合があります。これは通常、ライターが移行中または移行後にソーステーブルに書き込むときに発生します。これらの残りのファイルを Iceberg テーブルに組み込むには、add_files プロシージャを使用できます。詳細については、Iceberg ドキュメントの「ファイルの追加」を参照してください。

次のように、Athena で作成および入力された Parquet ベースの`products`テーブルがあるとします。

```
CREATE EXTERNAL TABLE mydb.products (
    product_id INT,
    product_name STRING
)
PARTITIONED BY (category STRING)
STORED AS PARQUET
LOCATION 's3://amzn-s3-demo-bucket/products/';

INSERT INTO mydb.products
VALUES
    (1001, 'Smartphone', 'electronics'),
    (1002, 'Laptop', 'electronics'),
    (2001, 'T-Shirt', 'clothing'),
    (2002, 'Jeans', 'clothing');
```

以下のセクションでは、この表で snapshotおよび migrateの手順を使用する方法について説明します。

オプション 1: スナップショット手順

snapshot この手順では、別の名前の新しい Iceberg テーブルを作成しますが、ソーステーブルのスキーマとパーティション化をレプリケートします。このオペレーションでは、アクション中とアクション後の両方でソーステーブルが完全に変更されません。テーブルの軽量コピーを効果的に作成します。これは、元のデータソースへの変更をリスクにさらすことなく、シナリオやデータ探索をテストする場合に特に役立ちます。このアプローチにより、元のテーブルと Iceberg テーブルの両方が利用可能な移行期間が可能になります (このセクションの最後にある注記を参照) 。テストが完了したら、すべてのライターとリーダーを新しいテーブルに移行することで、新しい Iceberg テーブルを本番環境に移行できます。

snapshot この手順は、任意の Amazon EMR デプロイモデル (EC2 上の Amazon EMR、EKS 上の Amazon EMR、EMR Serverless など) および で Spark を使用して実行できます AWS Glue。

Spark snapshot 手順を使用してインプレース移行をテストするには、次の手順に従います。

1. Spark アプリケーションを起動し、次の設定で Spark セッションを設定します。
   - "spark.sql.extensions":"org.apache.iceberg.spark.extensions.IcebergSparkSessi
   - "spark.sql.catalog.spark_catalog":"org.apache.iceberg.spark.SparkSessionCatal
   - "spark.sql.catalog.spark_catalog.type":"glue"
   - "spark.hadoop.hive.metastore.client.factory.class":"com.amazonaws.glue.catalo
2. snapshot 手順を実行して、元のテーブルデータファイルを指す新しい Iceberg テーブルを作成します。

```
spark.sql(f"""
CALL system.snapshot(
source_table => 'mydb.products',
table => 'mydb.products_iceberg',
location => 's3://amzn-s3-demo-bucket/products_iceberg/'
)
"""
).show(truncate=False)
```

   出力データフレームには、 imported_files_count (追加されたファイルの数) が含まれます。
3. クエリを実行して新しいテーブルを検証します。

```
spark.sql(f"""
SELECT * FROM mydb.products_iceberg LIMIT 10
"""
).show(truncate=False)
```

注意事項

- プロシージャを実行すると、ソーステーブルのデータファイルを変更すると、生成されたテーブルが同期解除されます。追加した新しいファイルは Iceberg テーブルに表示されず、削除したファイルは Iceberg テーブルのクエリ機能に影響します。同期の問題を回避するには:
  - 新しい Iceberg テーブルが本番稼働用である場合は、元のテーブルに書き込むすべてのプロセスを停止し、新しいテーブルにリダイレクトします。
  - 移行期間が必要な場合、または新しい Iceberg テーブルがテスト用である場合は、このセクションの後半の「インプレース移行後の Iceberg テーブルの同期の維持」を参照してください。
- snapshot プロシージャを使用すると、作成された Iceberg テーブルのテーブルプロパティfalseで プロパティが にgc.enabled設定されます。この設定ではexpire_snapshots、、remove_orphan_files、 などのアクションを PURGEオプションDROP TABLEで禁止します。これにより、データファイルが物理的に削除されます。ソースファイルに直接影響しない Iceberg 削除またはマージオペレーションは引き続き許可されます。
- データファイルを物理的に削除するアクションに制限なしで新しい Iceberg テーブルを完全に機能させるには、gc.enabledテーブルプロパティを に変更できますtrue。ただし、この設定により、ソースデータファイルに影響を与えるアクションが許可され、元のテーブルへのアクセスが破損する可能性があります。したがって、元のテーブルの機能を維持する必要がなくなった場合にのみ、 gc.enabledプロパティを変更します。例えば、次のようになります。

```
spark.sql(f"""
ALTER TABLE mydb.products_iceberg
SET TBLPROPERTIES ('gc.enabled' = 'true');
""")
```

オプション 2: 移行手順

migrate この手順では、ソーステーブルと同じ名前、スキーマ、パーティショニングを持つ新しい Iceberg テーブルを作成します。この手順を実行すると、ソーステーブルがロックされ、名前が <table_name>_BACKUP_ (またはbackup_table_nameプロシージャパラメータで指定されたカスタム名) に変更されます。

> **Note**
>
> drop_backup プロシージャパラメータを に設定した場合true、元のテーブルはバックアップとして保持されません。

したがって、migrateテーブルプロシージャでは、アクションが実行される前にソーステーブルに影響するすべての変更を停止する必要があります。migrate 手順を実行する前に、以下を実行します。

- ソーステーブルを操作するすべてのライターを停止します。
- Iceberg をネイティブにサポートしていないリーダーとライターを変更して、Iceberg サポートを有効にします。

例えば、次のようになります。

- Athena は変更なしで動作し続けます。
- Spark には以下が必要です。
  - クラスパスに含める Iceberg Java Archive (JAR) ファイル (このガイドの前半のセクションの「Amazon EMR での Iceberg の使用」および「Iceberg の使用 AWS Glue」を参照してください)。
  - 次の Spark セッションカタログ設定 (Iceberg 以外のテーブルの組み込みカタログ機能を維持しながら Iceberg サポートを追加SparkSessionCatalogするために使用します)。
    - "spark.sql.extensions":"org.apache.iceberg.spark.extensions.IcebergSparkSes
    - "spark.sql.catalog.spark_catalog":"org.apache.iceberg.spark.SparkSessionCat
    - "spark.sql.catalog.spark_catalog.type":"glue"
    - "spark.hadoop.hive.metastore.client.factory.class":"com.amazonaws.glue.cata

プロシージャを実行したら、新しい Iceberg 設定でライターを再起動できます。

現在、Data Catalog は RENAMEオペレーションをサポートしていないため AWS Glue Data Catalog、migrateこの手順は と互換性がありません。したがって、Hive Metastore を使用している場合にのみ、この手順を使用することをお勧めします。データカタログを使用している場合は、次のセクションで代替アプローチを参照してください。

migrate この手順は、すべての Amazon EMR デプロイモデル (EC2 上の Amazon EMR、EKS 上の Amazon EMR、EMR Serverless) および で実行できますが AWS Glue、Hive メタストアへの接続を設定する必要があります。Amazon EMR on EC2 は、セットアップの複雑さを最小限に抑える Hive メタストア設定が組み込まれているため、推奨される選択肢です。

Hive Metastore migrate で設定された Amazon EMR on EC2 クラスターから Spark プロシージャを使用してインプレース移行をテストするには、次の手順に従います。

1. Spark アプリケーションを起動し、Iceberg Hive カタログ実装を使用するように Spark セッションを設定します。たとえば、 CLI pyspark を使用している場合:

```
pyspark --conf
 spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions
 --conf spark.sql.catalog.spark_catalog=org.apache.iceberg.spark.SparkSessionCatalog
 --conf spark.sql.catalog.spark_catalog.type=hive
```

2. Hive メタストアにproductsテーブルを作成します。これはソーステーブルであり、一般的な移行に既に存在します。
   a. Hive Metastore でproducts外部 Hive テーブルを作成し、Amazon S3 の既存のデータを参照します。

```
spark.sql(f"""
CREATE EXTERNAL TABLE products (
    product_id INT,
    product_name STRING
)
PARTITIONED BY (category STRING)
STORED AS PARQUET
LOCATION 's3://amzn-s3-demo-bucket/products/';
"""
)
```

   b. MSCK REPAIR TABLE コマンドを使用して既存のパーティションを追加します。

```
spark.sql(f"""
MSCK REPAIR TABLE products
```

```
"""
)
```

c. SELECT クエリを実行して、テーブルにデータが含まれていることを確認します。

```
spark.sql(f"""
SELECT * FROM products
"""
).show(truncate=False)
```

サンプル出力:

```
>>> spark.sql(f"""
... SELECT * FROM products
... """
... ).show(truncate=False)
+----------+------------+-----------+
|product_id|product_name|category   |
+----------+------------+-----------+
|1001      |Smartphone  |electronics|
|1002      |Laptop      |electronics|
|2001      |T-Shirt     |clothing   |
|2002      |Jeans       |clothing   |
+----------+------------+-----------+
```

3. Iceberg `migrate`の手順を使用します。

```
df_res=spark.sql(f"""
CALL system.migrate(
table => 'default.products'
)
"""
)

df_res.show()
```

出力データフレームには 、 `migrated_files_count` (Iceberg テーブルに追加されたファイルの数) が含まれます。

```
>>> df_res.show()
+--------------------+
|migrated_files_count|
+--------------------+
|                   2|
+--------------------+
```

4. バックアップテーブルが作成されたことを確認します。

```
spark.sql("show tables").show()
```

サンプル出力:

```
>>> spark.sql("show tables").show()
+---------+----------------+-----------+
|namespace|       tableName|isTemporary|
+---------+----------------+-----------+
|  default|        products|      false|
|  default|products_backup_|      false|
+---------+----------------+-----------+
```

5. Iceberg テーブルをクエリしてオペレーションを検証します。

```
spark.sql(f"""
SELECT * FROM products
"""
).show(truncate=False)
```

注意事項

- プロシージャを実行すると、Iceberg サポートで適切に設定されていない場合、ソーステーブルにクエリまたは書き込む現在のプロセスがすべて影響を受けます。したがって、以下のステップに従うことをお勧めします。
  1. 移行前にソーステーブルを使用してすべてのプロセスを停止します。
  2. 移行を実行します。
  3. 適切な Iceberg 設定を使用してプロセスを再アクティブ化します。
- 移行プロセス中にデータファイルの変更が発生した場合 (新しいファイルが追加されるか、ファイルが削除された場合) 、生成されたテーブルは同期しなくなります。同期オプ

ションについては、このセクションの後半の「インプレース移行後に Iceberg テーブルを同期させる」を参照してください。

でのテーブル移行手順のレプリケート AWS Glue Data Catalog

以下の手順に従って AWS Glue Data Catalog 、移行手順の結果を (元のテーブルをバックアップして Iceberg テーブルに置き換える) でレプリケートできます。

1. スナップショットプロシージャを使用して、元のテーブルのデータファイルを指す新しい Iceberg テーブルを作成します。
2. データカタログで元のテーブルメタデータをバックアップします。
   a. GetTable API を使用して、ソーステーブル定義を取得します。
   b. GetPartitions API を使用して、ソーステーブルのパーティション定義を取得します。
   c. CreateTable API を使用して、データカタログにバックアップテーブルを作成します。
   d. CreatePartition または BatchCreatePartition API を使用して、データカタログのバックアップテーブルにパーティションを登録します。
3. `gc.enabled` Iceberg テーブルプロパティを に変更`false`して、完全なテーブルオペレーションを有効にします。
4. 元のテーブルを削除 (Drop) します。
5. テーブルのルートロケーションのメタデータフォルダで Iceberg テーブルメタデータ JSON ファイルを見つけます。
6. 元のテーブル名とプロシージャによって作成された`metadata.json`ファイルの場所を指定して register_table `snapshot`プロシージャを使用して、データカタログに新しいテーブルを登録します。

```
spark.sql(f"""
CALL system.register_table(
    table => 'mydb.products',
    metadata_file => '{iceberg_metadata_file}'
)
"""
).show(truncate=False)
```

インプレース移行後の Iceberg テーブルの同期の維持

add_files この手順では、既存のデータを Iceberg テーブルに柔軟に組み込むことができます。具体的には、Iceberg のメタデータレイヤーで絶対パスを参照することで、既存のデータファイル (Parquet ファイルなど) を登録します。デフォルトでは、プロシージャはすべてのテーブルパーティションから Iceberg テーブルにファイルを追加しますが、特定のパーティションからファイルを選択的に追加できます。この選択的なアプローチは、いくつかのシナリオで特に役立ちます。

- 最初の移行後に新しいパーティションがソーステーブルに追加された場合。
- 最初の移行後にデータファイルが既存のパーティションに追加または削除された場合。ただし、変更されたパーティションを再追加するには、最初にパーティションを削除する必要があります。詳細については、このセクションの後半で説明します。

インプレース移行 (snapshot または migrate) の実行後に add_fileプロシージャを使用して、新しい Iceberg テーブルをソースデータファイルと同期させる際の考慮事項を次に示します。

- ソーステーブルの新しいパーティションに新しいデータが追加されたら、 オプション で partition_filter add_filesプロシージャを使用して、これらの追加を Iceberg テーブルに選択的に組み込みます。

```
spark.sql(f"""
CALL system.add_files(
source_table => 'mydb.products',
table => 'mydb.products_iceberg',
partition_filter => map('category', 'electronics')
).show(truncate=False)
```

または

```
spark.sql(f"""
CALL system.add_files(
source_table => '`parquet`.`s3://amzn-s3-demo-bucket/products/`',
table => 'mydb.products_iceberg',
partition_filter => map('category', 'electronics')
).show(truncate=False)
```

- add_files プロシージャは、 partition_filterオプションを指定すると、ソーステーブル全体または特定のパーティション内のファイルをスキャンし、見つかったすべてのファイルを Iceberg テーブルに追加しようとします。デフォルトでは、check_duplicate_filesプロシー

ジャプロパティは に設定されます。これによりtrue、ファイルが Iceberg テーブルに既に存在する場合、プロシージャは実行されません。これは、以前に追加されたファイルをスキップする組み込みオプションがなく、無効にするとcheck_duplicate_filesファイルが 2 回追加され、重複が生じるため、重要です。ソーステーブルに新しいファイルが追加されたら、次の手順に従います。

1. 新しいパーティションの場合は、 add_filesで を使用してpartition_filter、新しいパーティションからファイルのみをインポートします。
2. 既存のパーティションの場合、まず Iceberg テーブルからパーティションを削除し、次にそのパーティションadd_filesに対して を指定して再実行しますpartition_filter。例えば、次のようになります。

```
# We initially perform in-place migration with snapshot
spark.sql(f"""
CALL system.snapshot(
source_table => 'mydb.products',
table => 'mydb.products_iceberg',
location => 's3://amzn-s3-demo-bucket/products_iceberg/'
)
"""
).show(truncate=False)

# Then on the source table, some new files were generated under the
 category='electronics' partition. Example:
spark.sql("""
INSERT INTO mydb.products
VALUES (1003, 'Tablet', 'electronics')
""")

# We delete the modified partition from the Iceberg table. Note this is a metadata
 operation only
spark.sql("""
DELETE FROM mydb.products_iceberg WHERE category = 'electronics'
""")

# We add_files from the modified partition
spark.sql("""
CALL system.add_files(
  source_table => 'mydb.products',
  table => 'mydb.products_iceberg',
  partition_filter => map('category', 'electronics')
)
```

```
""").show(truncate=False)
```

> **Note**
>
> add_files オペレーションごとに、追加されたデータを含む新しい Iceberg テーブルスナップショットが生成されます。

適切なインプレース移行戦略の選択

最適なインプレース移行戦略を選択するには、次の表の質問を検討してください。

質問	推奨事項	説明
Hive テーブルと Iceberg テーブルの両方をテストや段階的な移行のためにアクセスできるようにしながら、データを書き換えずにすばやく移行したいですか?	snapshot プロシージャとそれに続くadd_files プロシージャ	snapshot この手順を使用して、ソーステーブルを変更せずにスキーマをクローンし、データファイルを参照することで、新しい Iceberg テーブルを作成します。移行後に追加または変更されたパーティションを組み込むには、add_files プロシージャを使用します。変更されたパーティションを再追加するには、最初にパーティションを削除する必要があることに注意してください。
Hive メタストアを使用していて、データを書き換えることなく、すぐに Hive テーブルを Iceberg テーブルに置き換えたいですか?	migrate プロシージャに続く add_files プロシージャ	migrate 手順に従って Iceberg テーブルを作成し、ソーステーブルをバックアップし、元のテーブルを Iceberg バージョンに置き換えます。 注: このオプションは Hive メタストアと互換性があります

質問	推奨事項	説明
		が、 と互換性がありません AWS Glue Data Catalog。 移行後に追加または変更されたパーティションを組み込むには、 `add_files` プロシージャを使用します。変更されたパーティションを再追加するには、最初にパーティションを削除する必要があることに注意してください。

質問	推奨事項	説明
を使用していて AWS Glue Data Catalog 、データを書き換えることなく、Hive テーブルを Iceberg テーブルにすぐに置き換えたいですか?	`migrate` 手順の適応とそれに続く`add_files` 手順	`migrate` プロシージャの動作をレプリケートします。 1. `snapshot` を使用して Iceberg テーブルを作成します。 2. APIs を使用して AWS Glue 元のテーブルメタデータをバックアップします。 3. Iceberg テーブルプロパティ`gc.enabled` で を有効にします。 4. 元のテーブルを削除 (Drop) します。 5. `register_table` を使用して、元の名前で新しいテーブルエントリを作成します。 注: このオプションでは、メタデータバックアップの AWS Glue API コールを手動で処理する必要があります。 移行後に追加または変更されたパーティションを組み込むには、 `add_files` プロシージャを使用します。変更されたパーティションを再追加するには、最初にパーティションを削除する必要があることに注意してください。

フルデータ移行

フルデータ移行では、データファイルとメタデータが再作成されます。このアプローチには時間がかかり、インプレース移行と比較して追加のコンピューティングリソースが必要です。ただし、フルデータ移行は、テーブルの品質を向上させ、データストレージとアクセスパターンを最適化する大きな機会を提供します。

フルデータ移行中、整合性と正確性を確保するためのデータ検証、現在の要件をより適切に満たすためのスキーマの変更、クエリパフォーマンスを向上させるためのパーティション戦略の調整など、いくつかの有益なオペレーションを実行できます。また、データを再ソートして一般的なアクセスパターンを最適化し、Iceberg の隠しパーティショニングを実装してクエリ効率を高め、必要に応じてファイル形式変換 (CSV から Parquet など) を実行することもできます。

これらの機能により、フルデータ移行は Iceberg 形式への移行や、データストレージ戦略の包括的な調整と最適化に最適です。フルデータ移行にはより多くの時間とリソースが事前に必要ですが、結果的にデータ品質、組織、クエリのパフォーマンスが改善されると、長期的な利点が得られます。フルデータ移行を実装するには、次のいずれかのオプションを使用します。

- Spark (Amazon EMR または) または Athena で `CREATE TABLE ... AS SELECT` (CTAS AWS Glue) ステートメントを使用します。および 句を使用して `PARTITIONED BY`、新しい Iceberg テーブルのパーティション仕様と `TBLPROPERTIES`テーブルプロパティを設定できます。ソーステーブルから継承するのではなく、必要に応じて新しいテーブルのスキーマとパーティショニングを変更できます。
- Amazon EMR または で Spark を使用して、ソーステーブルから読み取り、データを新しい Iceberg テーブルとして書き込みます AWS Glue。詳細については、Iceberg ドキュメントの「テーブルの作成」を参照してください。

移行戦略の選択

Iceberg 形式に移行する場合、インプレース移行とフル移行の選択が不可欠です。特定のニーズに最適なアプローチを決定するには、次の質問と推奨事項を検討してください。

質問	推奨事項
データファイル形式 (CSV や Apache Parquet など) は何ですか?	• テーブルファイル形式が Parquet、ORC、または Avro の場合は、インプレース移行を検討してください。

質問	推奨事項
	• CSV、JSON などの他の形式の場合は、フルデータ移行を使用します。
テーブルスキーマを更新または統合しますか?	• Iceberg ネイティブ機能を使用してテーブルスキーマを進化させる場合は、インプレース移行を検討してください。たとえば、移行後に列の名前を変更できます。(スキーマは Iceberg メタデータレイヤーで変更できます) 。 • 不要になった列全体を削除する場合は、フルデータ移行を使用することをお勧めします。
テーブルはパーティション戦略を変更するとメリットがありますか?	• Iceberg のパーティショニングアプローチが要件を満たしている場合 (たとえば、新しいデータは新しいパーティションレイアウトを使用して保存され、既存のパーティションはそのままになる) 、インプレース移行を検討してください。 • テーブルで非表示のパーティションを使用する場合は、完全なデータ移行を検討してください。非表示パーティションの詳細については、「ベストプラクティス」セクションを参照してください。
テーブルはソート順序戦略を追加または変更することでメリットがありますか?	• データのソート順序を追加または変更するには、データセットを書き換える必要があります。この場合、フルデータ移行の使用を検討してください。 • すべてのテーブルパーティションを書き換えるコストが非常に高い大きなテーブルの場合は、インプレース移行の使用を検討し、最も頻繁にアクセスされるパーティションに対して圧縮 (ソートが有効) を実行してください。
テーブルには小さなファイルが多数ありますか?	• 小さなファイルを大きなファイルにマージするには、データセットを書き換える必要があります。この場合、フルデータ移行の使用を検討してください。 • すべてのテーブルパーティションを書き換えるコストが非常に高い大きなテーブルの場合は、インプレース移行の使用を検討し、最も頻繁にアクセスされるパーティションに対して圧縮 (ソートが有効) を実行してください。

移行オプションの概要

この表は、各移行オプションの主な特徴と考慮事項をまとめたものです。

機能	インプレース移行 スナップショット	インプレース移行 migrate	フルデータ移行 CTAS または (CREATE TABLE + INSERT)
移行プロセスの一環としてのデータレイアウトの改善			
• データの再ソート	⊗いいえ	⊗いいえ	⊘はい
• パーティショニングを変更する (Icebe の非表示パーティシ	⊗いいえ	⊗いいえ	⊘はい

機能	インプレース移行 スナップショット	インプレース移行 migrate	フルデータ移行 CTAS または (CREATE TABLE + INSERT)
ョニングを使用するなど)			
• テーブルスキーマを変更する	⊗いいえ	⊗いいえ	⊘はい
• ファイルサイズの最適化	⊗いいえ	⊗いいえ	⊘はい

機能	インプレース移行 スナップショット	インプレース移行 migrate	フルデータ移行 CTAS または (CREATE TABLE + INSERT)
• データを追加する前に、既存のデータのスキーマを検証する	⊗いいえ	⊗いいえ	⊘はい
サポートされているファイル形式	Parquet、Avro、ORC	Parquet、Avro、ORC	Parquet、Avro、ORC、JSON、CSV
Iceberg テーブルによるソーステーブルの置換	⊗いいえ (新しいテーブルを作成しますが、追加の手順でソーステーブルを置き換えることができます)	⊘はい (バックアップテーブルを作成し、ソーステーブルを Iceberg テーブルに置き換えます)	⊗いいえ (新しいテーブルを作成します)

機能	インプレース移行 スナップショット	インプレース移行 migrate	フルデータ移行 CTAS または (CREATE TABLE + INSERT)
ソーステーブルの影響			
• Iceber テーブルのファイル削除オペレーション (expi apsho オペレーション、パージを含むテーブルの削除)	ソーステーブルの破損	バックアップテーブルの破損	安全でソースに影響なし

機能	インプレース移行 スナップショット	インプレース移行 migrate	フルデータ移行 CTAS または (CREATE TABLE + INSERT)
Iceberg テーブルの影響			
• ソーステーブルファイルが削除された場合の影響	Iceberg テーブルの破損	Iceberg テーブルの破損	Iceberg テーブルに影響を与えない

機能	インプレース移行 スナップショット	インプレース移行 migrate	フルデータ移行 CTAS または (CREATE TABLE + INSERT)
• ソーステーブルの場所に新しいファイルが追加された場合の影響	新しいテーブルには表示されません (にパーティションを組み込む必要がありますadd_files)	新しいテーブルには表示されません (にパーティションを組み込む必要がありますadd_files)	新しいテーブルには表示されません (INSERT INTO新しいテーブルに必要)
コスト	低	低	高 (フルデータ書き換え)
移行速度	高速	高速	低速
Amazon S3 Tables への移行に使用できます	⊗いいえ	⊗いいえ	⊘はい

機能	インプレース移行 スナップショット	インプレース移行 migrate	フルデータ移行 CTAS または (CREATE TABLE + INSERT)
手動 DDL が必要	⊗いいえ (スキーマとパーティションはソーステーブルからコピーされます)	⊗いいえ (スキーマとパーティションはソーステーブルからコピーされます)	CTAS を使用する場合、 ではパーティショニングのみを指定する必要があります
最適な使用法	データを書き換えることなく迅速に移行できるため、テストや段階的な移行に Hive と Iceberg をside-by-side して使用できます。	即時スイッチオーバーが許容される場合に、データを書き換えずに Hive テーブルを置き換えます。	データ書き換えによる Iceberg 最適化。パーティションやスキーマを再設計する場合や、レイアウトとパフォーマンスを向上させる場合に最適です。可能な場合は、常に推奨されます。

Apache Iceberg ワークロードを最適化するためのベストプラクティス

Iceberg は、データレイク管理を簡素化し、ワークロードのパフォーマンスを向上させるように設計されたテーブル形式です。ユースケースが異なると、コスト、読み取りパフォーマンス、書き込みパフォーマンス、データ保持などのさまざまな側面が優先される可能性があるため、Iceberg はこれらのトレードオフを管理するための設定オプションを提供します。このセクションでは、要件を満たすために Iceberg ワークロードを最適化および微調整するためのインサイトを提供します。

トピック

一般的なベストプラクティス

ユースケースにかかわらず、 で Apache Iceberg を使用する場合は AWS、以下の一般的なベストプラクティスに従うことをお勧めします。

- Iceberg 形式バージョン 2 を使用します。

  Athena は、デフォルトで Iceberg 形式バージョン 2 を使用します。

  Amazon EMR で Spark または を使用して Iceberg テーブル AWS Glue を作成する場合は、Iceberg ドキュメントの説明に従って形式バージョンを指定します。
- をデータカタログ AWS Glue Data Catalog として使用します。

  Athena は AWS Glue Data Catalog デフォルトで を使用します。

  Amazon EMR で Spark を使用する場合、または Iceberg AWS Glue を操作する場合は、Spark セッションに次の設定を追加して を使用します AWS Glue Data Catalog。詳細については、このガイドの前半の「Iceberg の Spark 設定 AWS Glue」セクションを参照してください。

```
"spark.sql.catalog.<your_catalog_name>.type": "glue"
```

- を ロックマネージャー AWS Glue Data Catalog として使用します。

  Athena は、デフォルトで Iceberg テーブルのロックマネージャー AWS Glue Data Catalog としてを使用します。

  Amazon EMR で Spark を使用する場合、または Iceberg AWS Glue を操作する場合は、ロックマネージャー AWS Glue Data Catalog として を使用するように Spark セッション設定を必ず設定してください。詳細については、Iceberg ドキュメントの「オプティミスティックロック」を参照してください。

- Zstandard (ZSTD) 圧縮を使用します。

  Iceberg のデフォルトの圧縮コーデックは gzip であり、テーブルプロパティ を使用して変更できますwrite.<file_type>.compression-codec。Athena はすでに Iceberg テーブルのデフォルトの圧縮コーデックとして ZSTD を使用しています。

  一般的に、ZSTD 圧縮コーデックを使用することをお勧めします。これは、GZIP と Snappy のバランスが取れており、圧縮率を損なうことなく優れた読み取り/書き込みパフォーマンスを実現するためです。さらに、必要に応じて圧縮レベルを調整できます。詳細については、Athena ドキュメントの「Athena の ZSTD 圧縮レベル」を参照してください。

  Snappy は全体的な読み取りおよび書き込みパフォーマンスが最適ですが、GZIP および ZSTD よりも圧縮率が低くなります。パフォーマンスを優先する場合、Amazon S3 により大きなデータボリュームを保存する場合でも、Snappy が最適な選択肢である可能性があります。

読み取りパフォーマンスの最適化

このセクションでは、エンジンに関係なく読み取りパフォーマンスを最適化するために調整できるテーブルプロパティについて説明します。

パーティション

Hive テーブルと同様に、Iceberg はパーティションをインデックス作成のプライマリレイヤーとして使用して、不要なメタデータファイルやデータファイルの読み取りを回避します。列統計は、クエリ計画をさらに改善するためのインデックス作成のセカンダリレイヤーとしても考慮されるため、全体的な実行時間が長くなります。

データのパーティション化

Iceberg テーブルのクエリ時にスキャンされるデータの量を減らすには、予想される読み取りパターンに沿ったバランスの取れたパーティション戦略を選択します。

- クエリで頻繁に使用される列を特定します。これらは理想的なパーティショニング候補です。たとえば、通常、特定の日のデータをクエリする場合、パーティション列の自然な例は日付列になります。
- パーティションの数が過剰にならないように、低基数パーティション列を選択します。パーティションが多すぎると、テーブル内のファイル数が増え、クエリのパフォーマンスに悪影響を及ぼす可能性があります。経験則として、「パーティションが多すぎます」は、パーティションの大部分のデータサイズが によって設定された値の 2～5 倍未満であるシナリオとして定義できますtarget-file-size-bytes。

> **Note**
>
> 通常、高基数列 (数千の値を持つことができる id 列など) でフィルターを使用してクエリを実行する場合は、次のセクションで説明するように、バケット変換で Iceberg の非表示パーティショニング機能を使用します。

非表示パーティショニングを使用する

クエリが一般的にテーブル列の派生をフィルタリングする場合は、パーティションとして機能する新しい列を明示的に作成する代わりに、非表示のパーティションを使用します。この機能の詳細については、Iceberg のドキュメントを参照してください。

たとえば、タイムスタンプ列 (など2023-01-01 09:00:00) を持つデータセットでは、解析された日付 (など2023-01-01) で新しい列を作成する代わりに、パーティション変換を使用してタイムスタンプから日付部分を抽出し、これらのパーティションをその場で作成します。

非表示パーティショニングの最も一般的なユースケースは次のとおりです。

- データにタイムスタンプ列がある場合の、日付または時刻のパーティション分割。Iceberg は、タイムスタンプの日付または時刻部分を抽出するための複数の変換を提供します。
- パーティショニング列のカーディナリティが高く、パーティションが多すぎる場合に、列のハッシュ関数でパーティショニングします。Iceberg のバケット変換は、パーティショニング列でハッ

シュ関数を使用して、複数のパーティション値を少数の非表示 (バケット) パーティションにグループ化します。

使用可能なすべてのパーティション変換の概要については、Iceberg ドキュメントの「パーティション変換」を参照してください。

隠しパーティショニングに使用される列は、 `year()`や などの通常の SQL 関数を使用することで、クエリ述語の一部になる可能性がありますmonth()。述語は、 BETWEENや などの演算子と組み合わせることもできますAND。

> **Note**
>
> Iceberg は、 など、異なるデータ型を生成する関数に対してパーティションプルーニングを実行できませんsubstring(event_time, 1, 10) = '2022-01-01'。

パーティションの進化を使用する

既存のパーティション戦略が最適でない場合は、Iceberg のパーティション進化を使用します。たとえば、小さすぎる (それぞれわずか数メガバイト) 時間単位のパーティションを選択した場合は、日単位または月単位のパーティションへの移行を検討してください。

このアプローチは、テーブルに最適なパーティション戦略が最初は不明確で、より多くのインサイトを得るためにパーティション戦略を絞り込む場合に使用できます。パーティション進化のもう 1 つの効果的な用途は、データボリュームが変更され、現在のパーティショニング戦略が時間の経過とともに効果が低下する場合です。

パーティションを進化させる方法については、Iceberg ドキュメントの「ALTER TABLE SQL extensions」を参照してください。

ファイルサイズの調整

クエリのパフォーマンスを最適化するには、テーブル内の小さなファイルの数を最小限に抑える必要があります。クエリのパフォーマンスを向上させるには、通常、Parquet ファイルと ORC ファイルのサイズを 100 MB 以上にすることをお勧めします。

ファイルサイズは、Iceberg テーブルのクエリ計画にも影響します。テーブル内のファイル数が増えると、 はメタデータファイルのサイズも増加します。メタデータファイルが大きいほど、クエリ計

画が遅くなる可能性があります。したがって、テーブルサイズが大きくなったら、ファイルサイズを増や してメタデータの指数関数的な拡張を軽減します。

次のベストプラクティスを使用して、Iceberg テーブルに適切なサイズのファイルを作成します。

ターゲットファイルと行グループのサイズを設定する

Iceberg には、データファイルレイアウトを調整するための以下のキー設定パラメータが用意されています。これらのパラメータを使用して、ターゲットファイルサイズと行グループまたはストライクサイズを設定することをお勧めします。

パラメータ	デフォルト値:	[Comment] (コメント)
`write.target-file-size-bytes`	512 MB	このパラメータは、Iceberg が作成する最大ファイルサイズを指定します。ただし、特定のファイルは、この制限よりも小さいサイズで書き込まれる場合があります。
`write.parquet.row-group-size-bytes`	128 MB	Parquet と ORC はどちらもデータをチャンクに保存するため、エンジンは一部のオペレーションでファイル全体の読み取りを回避できます。
`write.orc.stripe-size-bytes`	64 MB	
`write.distribution-mode`	なし、Iceberg バージョン 1.1 以前の場合 Iceberg バージョン 1.2 から始まるハッシュ	Iceberg は、ストレージに書き込む前にタスク間でデータをソートするように Spark にリクエストします。

- 予想されるテーブルサイズに基づいて、次の一般的なガイドラインに従ってください。
  - スモールテーブル (最大数ギガバイト) — ターゲットファイルサイズを 128 MB に減らします。また、行グループまたはストライプサイズを小さくします (8 MB または 16 MB など) 。

  - 中 ~ 大きなテーブル (数ギガバイト ~ 数百ギガバイト) – デフォルト値は、これらのテーブルの出発点として最適です。クエリが非常に選択的である場合は、行グループまたはストライプサイズ (16 MB など) を調整します。
  - 非常に大きなテーブル (数百ギガバイトまたはテラバイト) – ターゲットファイルサイズを 1,024 MB 以上に増やし、クエリが通常大量のデータセットをプルする場合は、行グループまたはストライプサイズを増やすことを検討してください。

- Iceberg テーブルに書き込む Spark アプリケーションが適切なサイズのファイルを作成するようにするには、 `write.distribution-mode`プロパティを `hash`または に設定します`range`。これらのモードの違いの詳細については、Iceberg ドキュメントの「Writing Distribution Modes」を参照してください。

これらは一般的なガイドラインです。テストを実行して、特定のテーブルとワークロードに最適な値を特定することをお勧めします。

通常の圧縮を実行する

前の表の設定では、書き込みタスクが作成できる最大ファイルサイズが設定されていますが、ファイルのサイズがそのことを保証するものではありません。適切なファイルサイズを確保するには、圧縮を定期的に実行して、小さなファイルを大きなファイルにまとめます。圧縮の実行に関する詳細なガイダンスについては、このガイドの後半にある「Iceberg 圧縮」を参照してください。

列統計の最適化

Iceberg は列統計を使用してファイルプルーニングを実行し、クエリによってスキャンされるデータの量を減らすことでクエリのパフォーマンスを向上させます。列統計を活用するには、Iceberg がクエリフィルターで頻繁に使用されるすべての列の統計を収集していることを確認してください。

デフォルトでは、Iceberg はテーブルプロパティ で定義されているように、各テーブルの最初の 100 列の統計のみを収集します`write.metadata.metrics.max-inferred-column-defaults`。テーブルに 100 を超える列があり、クエリが最初の 100 列以外の列を頻繁に参照している場合 (たとえば、列 132 でフィルタリングする クエリがある場合) 、Iceberg がそれらの列の統計を収集していることを確認します。これを実現するには、次の 2 つのオプションがあります。

- Iceberg テーブルを作成するときは、クエリに必要な列が で設定された列範囲内に収まるように列の順序を変更します `write.metadata.metrics.max-inferred-column-defaults` (デフォルトは 100)。

注: 100 列の統計が必要ない場合は、`write.metadata.metrics.max-inferred-column-defaults`設定を目的の値 (20 など) に調整し、列の順序を変更して、読み取りおよび書き込みクエリが必要な列がデータセットの左側にある最初の 20 列に収まるようにできます。

- クエリフィルターで少数の列のみを使用する場合は、メトリクス収集の全体的なプロパティを無効にし、次の例に示すように、統計を収集する個々の列を選択的に選択できます。

```
.tableProperty("write.metadata.metrics.default", "none")
.tableProperty("write.metadata.metrics.column.my_col_a", "full")
.tableProperty("write.metadata.metrics.column.my_col_b", "full")
```

注: 列統計は、データがそれらの列でソートされるときに最も効果的です。詳細については、このガイドの後半にある「ソート順の設定」セクションを参照してください。

適切な更新戦略を選択する

遅い書き込みオペレーションがユースケースで許容できる場合、copy-on-writeライト戦略を使用して読み取りパフォーマンスを最適化します。これは Iceberg で使用されるデフォルトの戦略です。

ファイルが読み取り最適化方式でストレージに直接書き込まれるため、Copy-on-writeにより読み取りパフォーマンスが向上します。ただし、merge-on-readと比較して、各書き込みオペレーションには時間がかかり、より多くのコンピューティングリソースを消費します。これは、読み取りレイテンシーと書き込みレイテンシーの従来のトレードオフを示します。通常、copy-on-writeは、ほとんどの更新が同じテーブルパーティションにコロケーションされるユースケース (毎日のバッチロードなど) に最適です。

Copy-on-write 設定 (`write.update.mode`、、および `write.merge.mode`) は、テーブルレベルで設定することも`write.delete.mode`、アプリケーション側で個別に設定することもできます。

ZSTD 圧縮を使用する

Iceberg で使用される圧縮コーデックは、テーブルプロパティ を使用して変更できます`write.<file_type>.compression-codec`。テーブルの全体的なパフォーマンスを向上させるには、ZSTD 圧縮コーデックを使用することをお勧めします。

デフォルトでは、Iceberg バージョン 1.3 以前では GZIP 圧縮が使用されており、ZSTD と比較して読み取り/書き込みのパフォーマンスが遅くなります。

注: エンジンによっては、異なるデフォルト値を使用する場合があります。これは、Athena または Amazon EMR バージョン 7.x で作成された Iceberg テーブルの場合です。

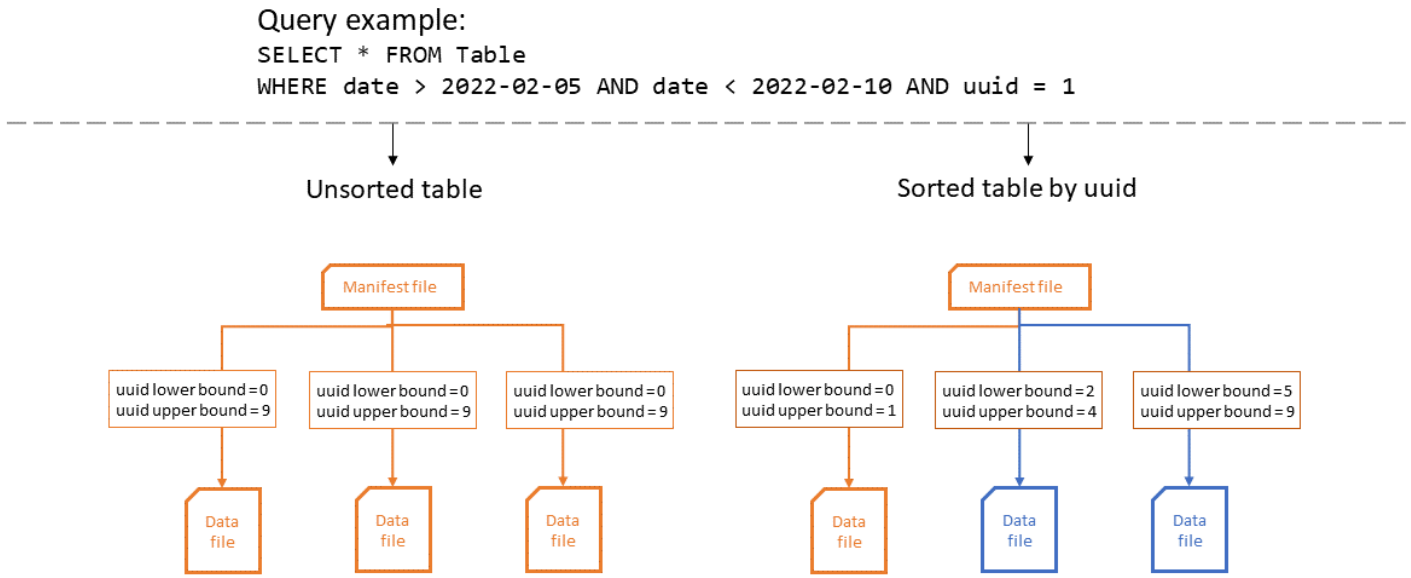
ソート順序を設定する

Iceberg テーブルの読み取りパフォーマンスを向上させるには、クエリフィルターで頻繁に使用される 1 つ以上の列に基づいてテーブルをソートすることをお勧めします。ソートと Iceberg の列統計を組み合わせると、ファイルプルーニングが大幅に効率化され、読み取り操作が高速化されます。また、ソートすると、クエリフィルターでソート列を使用するクエリの Amazon S3 リクエストの数も減ります。

Spark でデータ定義言語 (DDL) ステートメントを実行することで、テーブルレベルで階層ソート順序を設定できます。使用可能なオプションについては、Iceberg のドキュメントを参照してください。ソート順序を設定すると、ライターはこのソートを Iceberg テーブルの後続のデータ書き込みオペレーションに適用します。

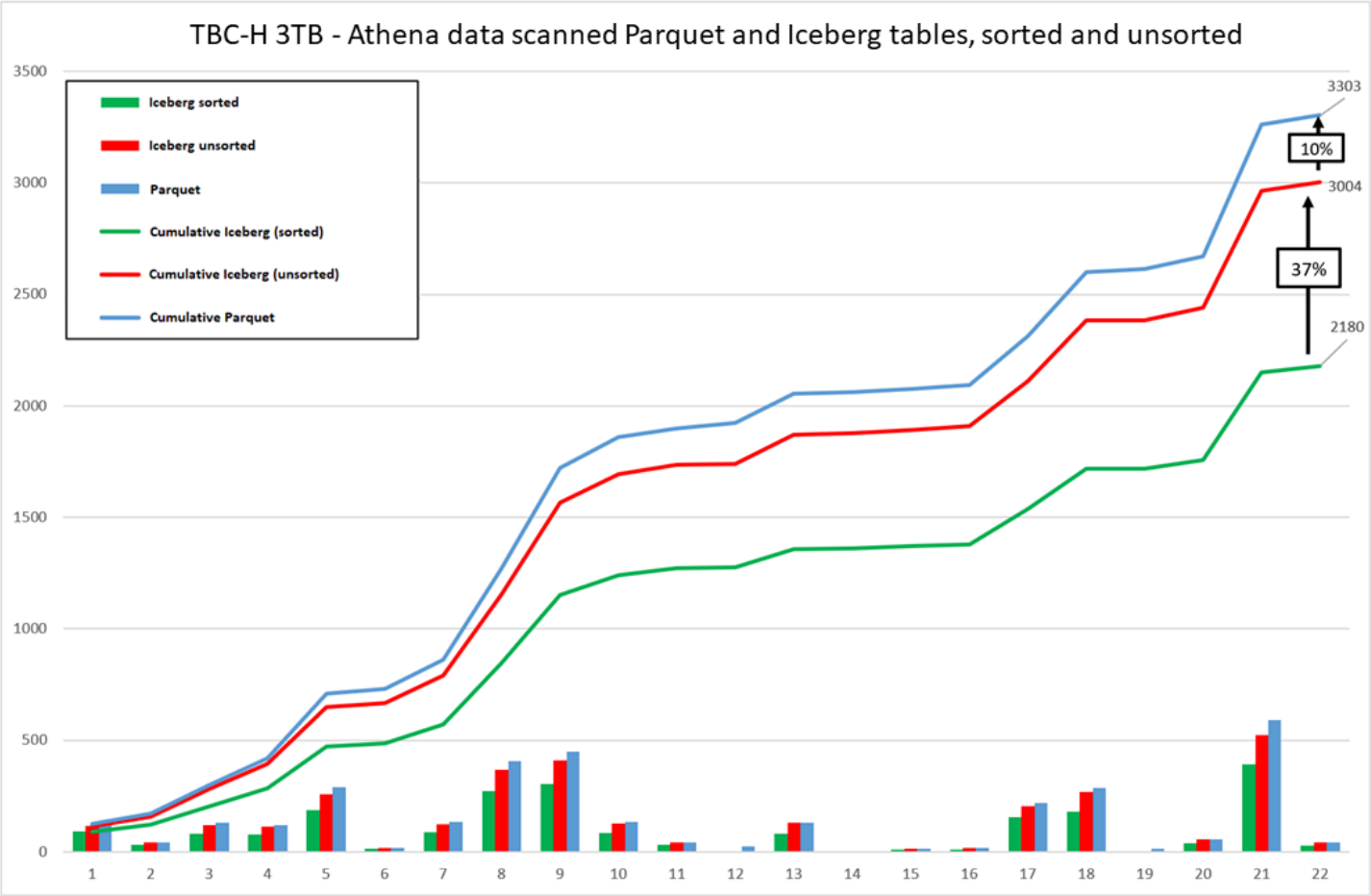
たとえば、ほとんどのクエリが でフィルタリングされる日付 (yyyy-mm-dd) でパーティション分割されたテーブルではuuid、DDL オプションを使用して、Spark が重複しない範囲のファイルを書き込むWrite Distributed By Partition Locally Orderedようにできます。

次の図は、テーブルをソートしたときに列統計の効率がどのように向上するかを示しています。この例では、ソートされたテーブルは 1 つのファイルのみを開く必要があり、Iceberg のパーティションとファイルを最大限に活用できます。ソートされていないテーブルでは、任意の uuidが任意のデータファイルに存在する可能性があるため、クエリはすべてのデータファイルを開く必要があります。

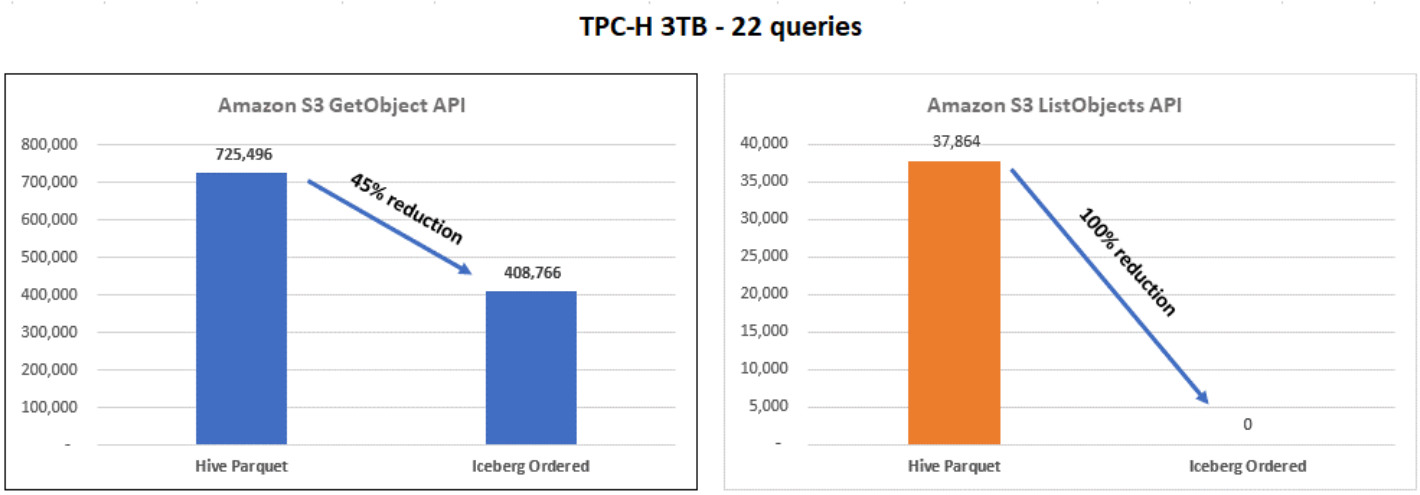


ソート順序を変更しても、既存のデータファイルには影響しません。Iceberg 圧縮を使用して、ソート順を適用できます。

次のグラフに示すように、Iceberg ソートテーブルを使用すると、ワークロードのコストを削減できます。



これらのグラフは、Iceberg ソートテーブルと比較した Hive (Parquet) テーブルの TPC-H ベンチマークを実行した結果をまとめたものです。ただし、結果は他のデータセットやワークロードで異なる場合があります。



書き込みパフォーマンスの最適化

このセクションでは、エンジンに関係なく Iceberg テーブルの書き込みパフォーマンスを最適化するために調整できるテーブルプロパティについて説明します。

テーブル分散モードを設定する

Iceberg には、Spark タスク間での書き込みデータの分散方法を定義する複数の書き込み分散モードが用意されています。使用可能なモードの概要については、Iceberg ドキュメントの「Writing Distribution Modes」を参照してください。

特にストリーミングワークロードで書き込み速度を優先するユースケースの場合は、`write.distribution-mode`を に設定しますnone。これにより、Iceberg は追加の Spark シャッフルをリクエストせず、Spark タスクで使用できるようになるとデータが書き込まれます。このモードは、Spark 構造化ストリーミングアプリケーションに特に適しています。

> **Note**
>
> 書き込み分散モードを に設定すると、多数の小さなファイルが生成されるnone傾向があり、読み取りパフォーマンスが低下します。これらの小さなファイルをクエリパフォーマンスのために適切なサイズのファイルに統合するには、定期的に圧縮することをお勧めします。

適切な更新戦略を選択する

読み取りmerge-on-read 戦略を使用して、最新のデータに対する読み取りオペレーションの遅延がユースケースで許容できる場合に、書き込みパフォーマンスを最適化します。

merge-on-readを使用すると、Iceberg は更新を書き込み、個別の小さなファイルとしてストレージを削除します。テーブルが読み取られると、リーダーはこれらの変更をベースファイルとマージして、データの最新のビューを返す必要があります。これにより、読み取りオペレーションのパフォーマンスが低下しますが、更新と削除の書き込みが高速化されます。通常、merge-on-readは、更新を含むストリーミングワークロードや、多くのテーブルパーティションに分散される更新が少ないジョブに最適です。

merge-on-read設定 (`write.update.mode`、、および `write.merge.mode`) は、テーブルレベルで設定することも`write.delete.mode`、アプリケーション側で個別に設定することもできます。

merge-on-readを使用するには、読み込みパフォーマンスが時間の経過とともに低下するのを防ぐために、定期的な圧縮を実行する必要があります。圧縮は、更新と削除を既存のデータファイルと照合して新しいデータファイルセットを作成するため、読み取り側で発生するパフォーマンスのペナルティを排除します。デフォルトでは、プロパティのデフォルトをより`delete-file-threshold`小さい値に変更しない限り、Iceberg の圧縮は削除ファイルをマージしません (Iceberg ドキュメントを参照)。圧縮の詳細については、このガイドの後半にある「Iceberg 圧縮」セクションを参照してください。

適切なファイル形式を選択する

Iceberg は、Parquet、ORC、および Avro 形式のデータの書き込みをサポートしています。Parquet はデフォルトの形式です。Parquet と ORC は、優れた読み取りパフォーマンスを提供する列指向形式ですが、一般的に書き込みが遅くなります。これは、読み取りパフォーマンスと書き込みパフォーマンスの一般的なトレードオフを表します。

ストリーミングワークロードなど、ユースケースで書き込み速度が重要な場合は、ライターのオプション`Avro`で `write-format`を に設定して Avro 形式で書き込むことを検討してください。Avro は行ベースの形式であるため、書き込み時間が短縮され、読み取りパフォーマンスが低下します。

読み取りパフォーマンスを向上させるには、通常の圧縮を実行して小さな Avro ファイルをマージし、より大きな Parquet ファイルに変換します。圧縮プロセスの結果は、`write.format.default`テーブル設定によって管理されます。Iceberg のデフォルト形式は Parquet であるため、Avro で書き込み、圧縮を実行すると、Iceberg は Avro ファイルを Parquet ファイルに変換します。例を示します。

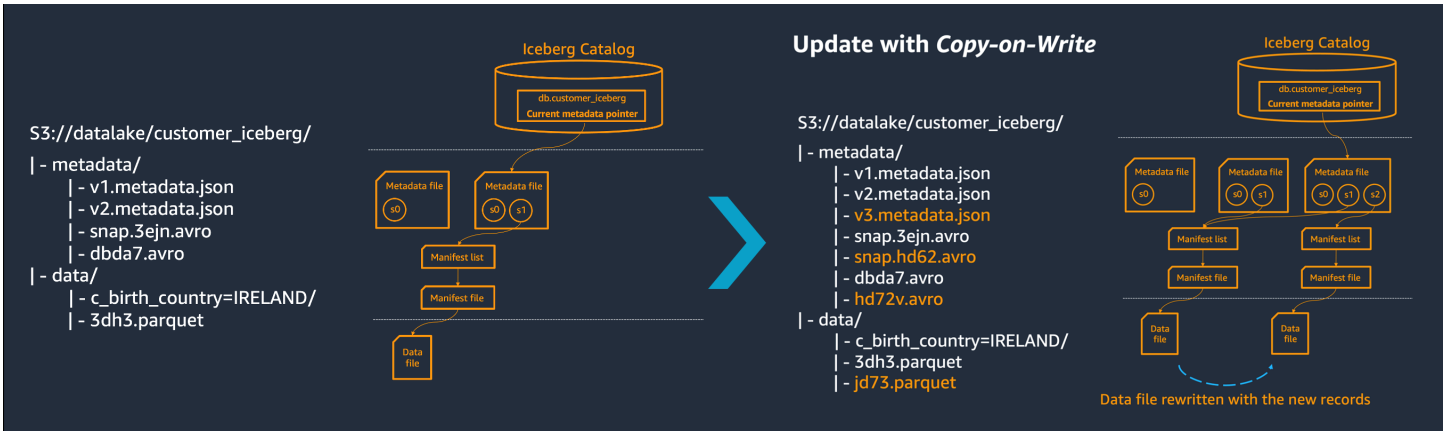
```
spark.sql(f"""
    CREATE TABLE IF NOT EXISTS glue_catalog.{DB_NAME}.{TABLE_NAME} (
        Col_1 float,
        <<<…other columns…>>
        ts timestamp)
    USING iceberg
    PARTITIONED BY (days(ts))
    OPTIONS (
      'format-version'='2',
      write.format.default'=parquet)
""")

query = df \
    .writeStream \
    .format("iceberg") \
    .option("write-format", "avro") \
    .outputMode("append") \
    .trigger(processingTime='60 seconds') \
    .option("path", f"glue_catalog.{DB_NAME}.{TABLE_NAME}") \
    .option("checkpointLocation", f"s3://{BUCKET_NAME}/checkpoints/iceberg/")

    .start()
```

ストレージの最適化

Iceberg テーブルのデータを更新または削除すると、次の図に示すように、データのコピー数が増加します。圧縮を実行する場合も同じです。Amazon S3 のデータコピーの数が増えます。これは、Iceberg がすべてのテーブルの基盤となるファイルをイミュータブルとして扱うためです。



このセクションのベストプラクティスに従って、ストレージコストを管理します。

S3 Intelligent-Tiering を有効にする

Amazon S3 Intelligent-Tiering ストレージクラスを使用して、アクセスパターンが変化したときに最も費用対効果の高いアクセス階層にデータを自動的に移動します。このオプションは、運用上のオーバーヘッドやパフォーマンスへの影響はありません。

注: Iceberg テーブルで S3 Intelligent-Tiering のオプション階層 (アーカイブアクセスやディープアーカイブアクセスなど) を使用しないでください。データをアーカイブするには、次のセクションのガイドラインを参照してください。

Amazon S3 ライフサイクルルールを使用して、S3 Standard-IA や Amazon S3 S3 ストレージクラスにオブジェクトを移動するための独自のルールを設定することもできます (Amazon S3 ドキュメントの「サポートされている移行と関連する制約」を参照) 。

履歴スナップショットのアーカイブまたは削除

Iceberg テーブルへのコミットされたトランザクション (挿入、更新、マージ、圧縮) ごとに、テーブルの新しいバージョンまたはスナップショットが作成されます。時間の経過とともに、Amazon S3 のバージョン数とメタデータファイル数が累積されます。

スナップショットの分離、テーブルのロールバック、タイムトラベルクエリなどの機能には、テーブルのスナップショットを保持する必要があります。ただし、ストレージコストは、保持するバージョンの数に応じて増加します。

次の表は、データ保持要件に基づいてコストを管理するために実装できる設計パターンを示しています。

設計パターン	解決策	ユースケース
古いスナップショットを削除する	• Athena の VACUUM ステートメントを使用して、古いスナップショットを削除します。このオペレーションでは、コンピューティングコストは発生しません。 • または、Amazon EMR で Spark または AWS Glue を	このアプローチでは、ストレージコストを削減するために不要になったスナップショットを削除します。データ保持要件に基づいて、保持するスナップショットの数または保持期間を設定できます。

設計パターン	解決策	ユースケース
	使用してスナップショットを削除することもできます。詳細については、Iceberg ドキュメントの expire_snapshots を参照してください。	このオプションは、スナップショットのハード削除を実行します。期限切れのスナップショットにロールバックしたり、タイムトラベルしたりすることはできません。
特定のスナップショットの保持ポリシーを設定する	1. タグを使用して特定のスナップショットをマークし、Iceberg で保持ポリシーを定義します。詳細については、Iceberg ドキュメントの「履歴タグ」を参照してください。 たとえば、Amazon EMR の Spark で次の SQL ステートメントを使用して、1 年間、1 か月あたり 1 つのスナップショットを保持できます。 `ALTER TABLE glue_catalog.db.table CREATE TAG 'EOM-01' AS OF VERSION 30 RETAIN 365 DAYS` 2. Amazon EMR の Spark または AWS Glue を使用して、タグ付けされていない残りの中間スナップショットを削除します。	このパターンは、過去の特定の時点でテーブルの状態を表示する必要があるビジネス要件または法的要件への準拠に役立ちます。特定のタグ付けされたスナップショットに保持ポリシーを配置することで、作成された他の (タグ付けされていない) スナップショットを削除できます。これにより、作成されたすべてのスナップショットを保持することなく、データ保持要件を満たすことができます。

設計パターン	解決策	ユースケース
古いスナップショットのアーカイブ	1. Amazon S3 タグを使用して、Spark でオブジェクトをマークします。(Amazon S3 タグは Iceberg タグとは異なります。詳細については、Iceberg ドキュメントを参照してください)。例えば、次のようになります。 `spark.sql.catalog.my_catalog.s3.delete-enabled=false and \ spark.sql.catalog.my_catalog.s3.delete.tags.my_key=to_archive` 2. Amazon EMR で Spark または AWS Glue を使用してスナップショットを削除します。 https://iceberg.apache.org/docs/latest/spark-procedures/#expire_snapshotsこの例の設定を使用すると、この手順では、Amazon S3 から削除するのではなく、オブジェクトにタグを付け、Iceberg テーブルメタデータからデタッチします。 3. S3 ライフサイクルルールを使用して、 としてタグ付けされたオブジェクトto_archive を S3	このパターンにより、すべてのテーブルバージョンとスナップショットを低コストで維持できます。 アーカイブされたスナップショットをタイムトラベルまたはロールバックするには、まずそれらのバージョンを新しいテーブルとして復元する必要があります。これは通常、監査目的で許容されます。 このアプローチを以前の設計パターンと組み合わせて、特定のスナップショットの保持ポリシーを設定できます。

設計パターン	解決策	ユースケース
	Glacier ストレージクラスの 1 つに移行します。 4. アーカイブされたデータをクエリするには: • アーカイブされたオブジェクトを復元します (オブジェクトが Amazon Glacier Instant Retrieval ストレージクラスに移行された場合、このステップは必要ありません)。 • Iceberg の register_table プロシージャを使用して、スナップショットをカタログのテーブルとして登録します。 詳細な手順については、 AWS ブログ記事「Amazon S3 データレイク上に構築された Apache Iceberg テーブルの運用効率の向上」を参照してください。	

孤立ファイルを削除する

状況によっては、トランザクションをコミットする前に Iceberg アプリケーションが失敗することがあります。これにより、データファイルは Amazon S3 に残ります。コミットがないため、これらのファイルはテーブルに関連付けられないため、非同期的にクリーンアップする必要がある場合があります。

これらの削除を処理するには、Amazon Athena で VACUUM ステートメントを使用できます。このステートメントはスナップショットを削除し、孤立したファイルも削除します。Athena はこのオペレーションのコンピューティングコストを請求しないため、これは非常にコスト効率的です。また、VACUUMステートメントを使用する場合、追加のオペレーションをスケジュールする必要はありません。

または、Amazon EMR または で Spark AWS Glue を使用してremove_orphan_filesプロシージャを実行できます。このオペレーションにはコンピューティングコストがかかり、個別にスケジュールする必要があります。詳細については、Iceberg ドキュメントを参照してください。

圧縮を使用したテーブルのメンテナンス

Iceberg には、テーブルにデータを書き込んだ後にテーブルのメンテナンス操作を実行できるようにする機能が含まれています。一部のメンテナンスオペレーションではメタデータファイルの合理化に重点を置いていますが、他のメンテナンスオペレーションでは、クエリエンジンがユーザーリクエストに応答するために必要な情報を効率的に見つけられるように、データのファイルのクラスター化方法を強化しています。このセクションでは、圧縮関連の最適化に焦点を当てます。

Iceberg 圧縮

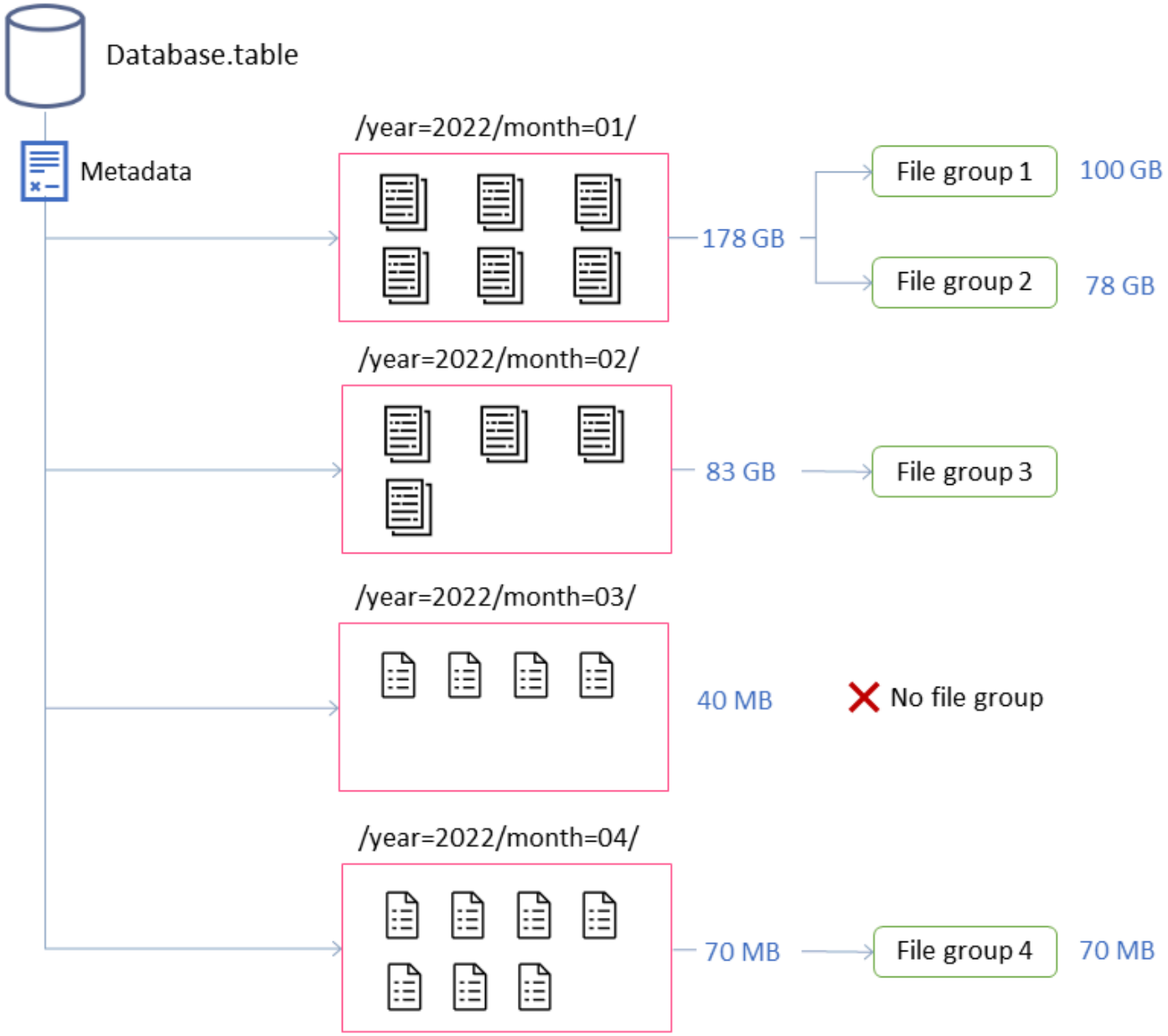
Iceberg では、圧縮を使用して 4 つのタスクを実行できます。

- 小さなファイルを、通常 100 MB を超えるサイズの大きなファイルに結合します。この手法はビンパッキングと呼ばれます。
- 削除ファイルをデータファイルとマージします。削除ファイルは、merge-on-readアプローチを使用する更新または削除によって生成されます。
- (再) クエリパターンに従ってデータをソートします。データは、ソート順なしで、または書き込みと更新に適したソート順で書き込むことができます。
- スペースフィル曲線を使用してデータをクラスター化し、個別のクエリパターン、特に z 順序ソートを最適化します。

では AWS、Amazon Athena を介して、または Amazon EMR または で Spark を使用して、Iceberg のテーブル圧縮およびメンテナンスオペレーションを実行できます AWS Glue。

rewrite_data_files プロシージャを使用して圧縮を実行する場合、いくつかのつまみを調整して圧縮動作を制御できます。次の図は、ビンパッキングのデフォルトの動作を示しています。ビンパッキング圧縮を理解することは、階層ソートと Z 順序ソートの実装を理解する上で重要です。これらはビン

パッキングインターフェイスの拡張であり、同様の方法で動作するためです。主な違いは、データのソートまたはクラスター化に必要な追加のステップです。



この例では、Iceberg テーブルは 4 つのパーティションで構成されています。各パーティションのサイズとファイルの数は異なります。Spark アプリケーションを起動して圧縮を実行すると、アプリケーションは合計 4 つのファイルグループを作成して処理します。ファイルグループは、単一の Spark ジョブによって処理されるファイルのコレクションを表す Iceberg 抽象化です。つまり、圧縮を実行する Spark アプリケーションは、データを処理するための 4 つの Spark ジョブを作成します。

圧縮動作の調整

次のキープロパティは、圧縮のためにデータファイルを選択する方法を制御します。

- MAX_FILE_GROUP_SIZE_BYTES は、デフォルトで単一のファイルグループ (Spark ジョブ) のデータ制限を 100 GB に設定します。このプロパティは、パーティションのないテーブルや数百ギガバイトにまたがるパーティションを持つテーブルにとって特に重要です。この制限を設定することで、クラスターでのリソースの枯渇を防ぎながら、オペレーションを分割して作業を計画し、進行させることができます。

  注: 各ファイルグループは個別にソートされます。したがって、パーティションレベルのソートを実行する場合は、パーティションサイズに合わせてこの制限を調整する必要があります。
- MIN_FILE_SIZE_BYTES または MIN_FILE_SIZE_DEFAULT_RATIO は、デフォルトでテーブルレベルで設定されたターゲットファイルサイズの 75% に設定されます。たとえば、テーブルのターゲットサイズが 512 MB の場合、384 MB 未満のファイルは圧縮されるファイルのセットに含まれます。
- MAX_FILE_SIZE_BYTES または MAX_FILE_SIZE_DEFAULT_RATIO のデフォルトは、ターゲットファイルサイズの 180% です。最小ファイルサイズを設定する 2 つのプロパティと同様に、これらのプロパティは圧縮ジョブの候補ファイルを識別するために使用されます。
- MIN_INPUT_FILES は、テーブルパーティションサイズがターゲットファイルサイズより小さい場合に圧縮するファイルの最小数を指定します。このプロパティの値は、ファイル数 (デフォルトは 5) に基づいてファイルを圧縮する価値があるかどうかを判断するために使用されます。
- DELETE_FILE_THRESHOLD は、圧縮に含まれる前のファイルの削除オペレーションの最小数を指定します。特に指定しない限り、圧縮は削除ファイルとデータファイルを組み合わせません。この機能を有効にするには、このプロパティを使用してしきい値を設定する必要があります。このしきい値は個々のデータファイルに固有のため、3 に設定すると、それを参照する削除ファイルが 3 つ以上ある場合にのみ、データファイルが書き直されます。

これらのプロパティは、前の図のファイルグループの形成に関するインサイトを提供します。

たとえば、ラベルが付けられたパーティションには、100 GB の最大サイズ制約を超えているため、2 つのファイルグループ`month=01`が含まれています。対照的に、`month=02`パーティションには 100 GB 未満の単一のファイルグループが含まれています。`month=03` パーティションは、5 つのファイルのデフォルトの最小入力ファイル要件を満たしていません。その結果、圧縮されません。最後に、`month=04`パーティションには目的のサイズの単一のファイルを形成するのに十分なデータが含まれていませんが、パーティションには 5 つ以上の小さなファイルが含まれているため、ファイルは圧縮されます。

Amazon EMR または で実行されている Spark に対してこれらのパラメータを設定できます AWS Glue。Amazon Athena では、プレフィックス で始まるテーブルプロパティを使用して、同様のプロパティを管理できますoptimize_)。

Amazon EMR または で Spark を使用して圧縮を実行する AWS Glue

このセクションでは、Iceberg の圧縮ユーティリティを実行するために Spark クラスターを適切にサイズ設定する方法について説明します。次の例では Amazon EMR Serverless を使用していますが、EC2 または EKS 上の Amazon EMR、または で同じ方法を使用できます AWS Glue。

ファイルグループと Spark ジョブの相関関係を活用して、クラスターリソースを計画できます。ファイルグループを順番に処理するには、ファイルグループあたりの最大サイズ 100 GB を考慮して、次の Spark プロパティを設定できます。

- `spark.dynamicAllocation.enabled = FALSE`
- `spark.executor.memory = 20 GB`
- `spark.executor.instances = 5`

圧縮を高速化する場合は、並列に圧縮されるファイルグループの数を増やすことで、水平方向にスケールできます。手動または動的スケーリングを使用して Amazon EMR をスケーリングすることもできます。

- 手動スケーリング (4 倍など)
  - `MAX_CONCURRENT_FILE_GROUP_REWRITES = 4` (係数)
  - `spark.executor.instances = 5` (例で使用されている値) x `4` (係数) = `20`
  - `spark.dynamicAllocation.enabled = FALSE`
- 動的スケーリング
  - `spark.dynamicAllocation.enabled = TRUE` (デフォルト、アクション不要)
  - MAX_CONCURRENT_FILE_GROUP_REWRITES = `N` (この値をデフォルトで 100 `spark.dynamicAllocation.maxExecutors`の に揃えます。例の工グゼキュター設定に基づいて、 `N` を 20 に設定できます)

  これらは、クラスターのサイズ設定に役立つガイドラインです。ただし、Spark ジョブのパフォーマンスをモニタリングして、ワークロードに最適な設定を見つける必要もあります。

Amazon Athena で圧縮を実行する

Athena は、OPTIMIZE ステートメントを通じて、マネージド機能として Iceberg の圧縮ユーティリティの実装を提供します。このステートメントを使用して、インフラストラクチャを評価することなく圧縮を実行できます。

このステートメントは、ビンパッキングアルゴリズムを使用して小さなファイルを大きなファイルにグループ化し、削除ファイルを既存のデータファイルとマージします。階層ソートまたは z 順序ソートを使用してデータをクラスター化するには、Amazon EMR または で Spark を使用します AWS Glue。

テーブル作成時の OPTIMIZEステートメントのデフォルト動作は、 ステートメントでテーブルプロパティを渡すか、 CREATE TABLEステートメントを使用してテーブル作成後に変更できますALTER TABLE。デフォルト値については、Athena のドキュメントを参照してください。

圧縮を実行するための推奨事項

ユースケース	レコメンデーション
スケジュールに基づいてビンパッキング圧縮を実行する	• テーブルに含まれる小さなファイル数がわからない場合は、Athena の OPTIMIZEステートメントを使用します。Athena の料金モデルはスキャンされたデータに基づいているため、圧縮するファイルがない場合、これらのオペレーションに関連するコストは発生しません。Athena テーブルでタイムアウトが発生しないようにするには、OPTIMIZEper-table-partitionを実行します。 • 大量の小さなファイルが圧縮されると予想される場合は、Amazon EMR または を動的スケーリング AWS Glue で使用します。
イベントに基づいてビンパッキング圧縮を実行する	• 大量の小さなファイルが圧縮されると予想される場合は、Amazon EMR または を動的スケーリング AWS Glue で使用します。
圧縮を実行してデータをソートする	• ソートは高価な操作であり AWS Glue、データをディスクにスピルする必要がある可能性

ユースケース	レコメンデーション
	があるため、Amazon EMR または を使用します。
圧縮を実行して z 順序ソートを使用してデータをクラスター化する	• Amazon EMR または を使用します。z 順序ソートは非常に高価な操作であり AWS Glue、データをディスクにスピルする必要がある可能性があるためです。
遅延受信データが原因で他のアプリケーションによって更新される可能性のあるパーティションで圧縮を実行する	• Amazon EMR または を使用します AWS Glue。Iceberg PARTIAL_PROGRESS_ENABLED プロパティを有効にします。このオプションを使用すると、Iceberg は圧縮出力を複数のコミットに分割します。衝突がある場合 (つまり、圧縮の実行中にデータファイルが更新された場合) 、この設定は、影響を受けるファイルを含むコミットに制限することで、再試行のコストを削減します。それ以外の場合は、すべてのファイルを再圧縮する必要がある場合があります。
コールドパーティション (アクティブな書き込みを受信しなくなったデータパーティション) で圧縮を実行する	• Amazon EMR または を使用します AWS Glue。`rewrite_data_files` プロシージャで、アクティブに書き込まれたパーティションを除外するwhere述語を指定します。この戦略は、ライターと圧縮ジョブ間のデータ競合を防ぎ、Iceberg が自動的に解決できるメタデータ競合のみを残します。

Amazon S3 での Iceberg ワークロードの使用

このセクションでは、Iceberg の Amazon S3 とのやり取りを最適化するために使用できる Iceberg プロパティについて説明します。

ホットパーティショニングの防止 (HTTP 503 エラー)

Amazon S3 で実行される一部のデータレイクアプリケーションは、数百万または数十億のオブジェクトを処理し、ペタバイトのデータを処理します。これにより、大量のトラフィックを受信するプレフィックスが発生する可能性があります。これは通常、HTTP 503 (サービス利用不可) エラーによって検出されます。この問題を回避するには、次の Iceberg プロパティを使用します。

- Iceberg が大きなファイルを書き込むrangeように hashまたは write.distribution-modeに設定すると、Amazon S3 リクエストが少なくなります。これは推奨される設定であり、ほとんどのケースに対応する必要があります。
- ワークロード内の大量のデータが原因で 503 エラーが続く場合は、Iceberg trueで write.object-storage.enabledを に設定できます。これにより、オブジェクト名をハッシュし、ランダム化された複数の Amazon S3 プレフィックスに負荷を分散するように Iceberg に指示します。

これらのプロパティの詳細については、Iceberg ドキュメントの「プロパティの書き込み」を参照してください。

Iceberg メンテナンスオペレーションを使用して未使用のデータをリリースする

Iceberg テーブルを管理するには、Iceberg コア API、Iceberg クライアント (Spark など) 、または Amazon Athena などのマネージドサービスを使用できます。Amazon S3 から古いファイルまたは未使用のファイルを削除するには、Iceberg ネイティブ APIs のみを使用して、スナップショットの削除、古いメタデータファイルの削除、孤立ファイルの削除を行うことをお勧めします。

Boto3、Amazon S3 SDK、または AWS Command Line Interface (AWS CLI) を介して Amazon S3 APIs を使用するか、Iceberg 以外の他のメソッドを使用して Iceberg テーブルの Amazon S3 ファイルを上書きまたは削除すると、テーブルの破損やクエリの失敗が発生します。

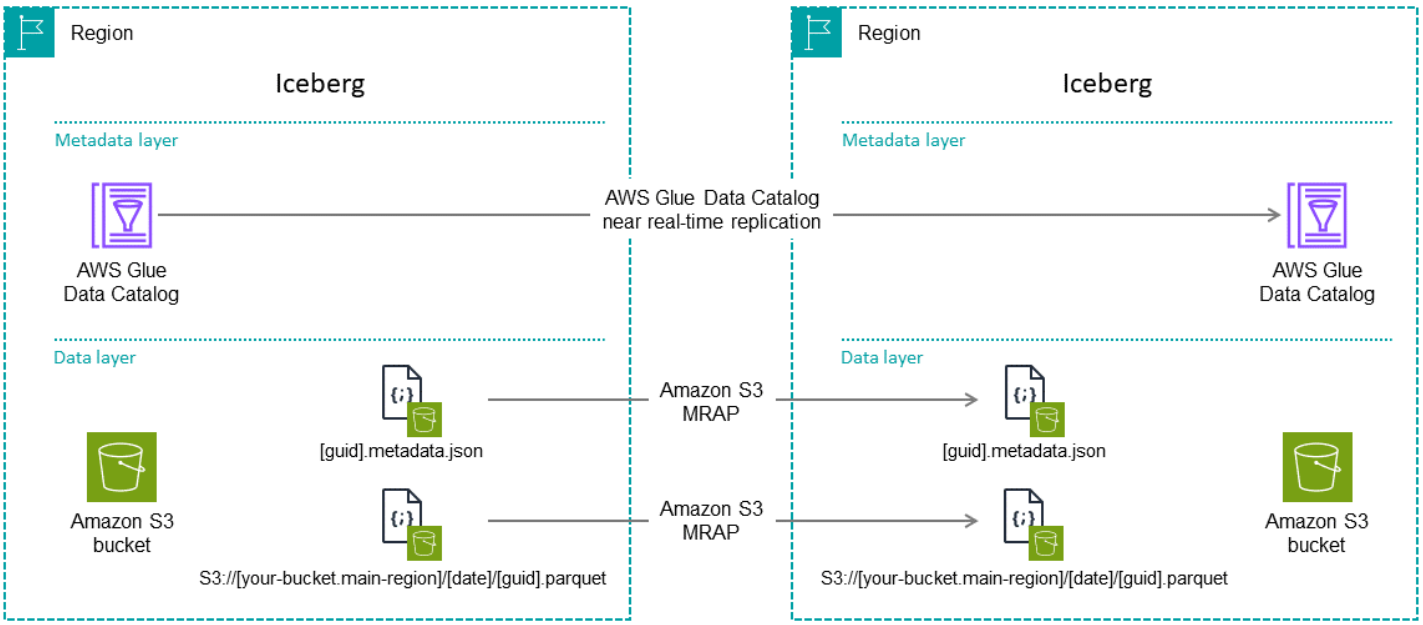
間でデータをレプリケートする AWS リージョン

Iceberg テーブルを Amazon S3 に保存する場合、クロスリージョンレプリケーション (CRR) やマルチリージョンアクセスポイント (MRAP) などの Amazon S3 の組み込み機能を使用して、複数の にデータをレプリケートできます AWS リージョン。MRAP は、アプリケーションが複数の にある S3 バケットにアクセスするためのグローバルエンドポイントを提供します AWS リージョン。Iceberg は相対パスをサポートしていませんが、MRAP を使用してバケットをアクセスポイントにマッピン

グすることで Amazon S3 オペレーションを実行できます。MRAP は Amazon S3 クロスリージョンレプリケーションプロセスともシームレスに統合されるため、最大 15 分の遅延が発生します。データファイルとメタデータファイルの両方をレプリケートする必要があります。

> **Important**
>
> 現在、Iceberg と MRAP の統合は Apache Spark でのみ機能します。セカンダリにフェイルオーバーする必要がある場合は AWS リージョン、フェイルオーバーリージョンの Spark SQL 環境 (Amazon EMR など) にユーザークエリをリダイレクトする計画を立てる必要があります。

CRR および MRAP 機能は、次の図に示すように、Iceberg テーブル用のクロスリージョンレプリケーションソリューションを構築するのに役立ちます。



このクロスリージョンレプリケーションアーキテクチャを設定するには:

1. MRAP の場所を使用してテーブルを作成します。これにより、Iceberg メタデータファイルは物理バケットの場所ではなく MRAP の場所を指します。
2. Amazon S3 MRAP を使用して Iceberg ファイルをレプリケートします。 MRAP は、15 分のサービスレベルアグリーメント (SLA) でデータレプリケーションをサポートします。Iceberg は、レプリケーション中に読み取りオペレーションに不整合が生じるのを防ぎます。
3. テーブルをセカンダリリージョンの AWS Glue Data Catalog で使用可能にします。2 つのオプションから選択できます。

- AWS Glue Data Catalog レプリケーションを使用して Iceberg テーブルメタデータをレプリケートするためのパイプラインを設定します。このユーティリティは、GitHub Glue Catalog および Lake Formation Permissions レプリケーションリポジトリで使用できます。このイベント駆動型メカニズムは、イベントログに基づいてターゲットリージョンのテーブルをレプリケートします。
- フェイルオーバーする必要がある場合は、セカンダリリージョンにテーブルを登録します。このオプションでは、前のユーティリティまたは Iceberg register_table プロシージャを使用して、最新の`metadata.json`ファイルを参照できます。

Apache Iceberg ワークロードのモニタリング

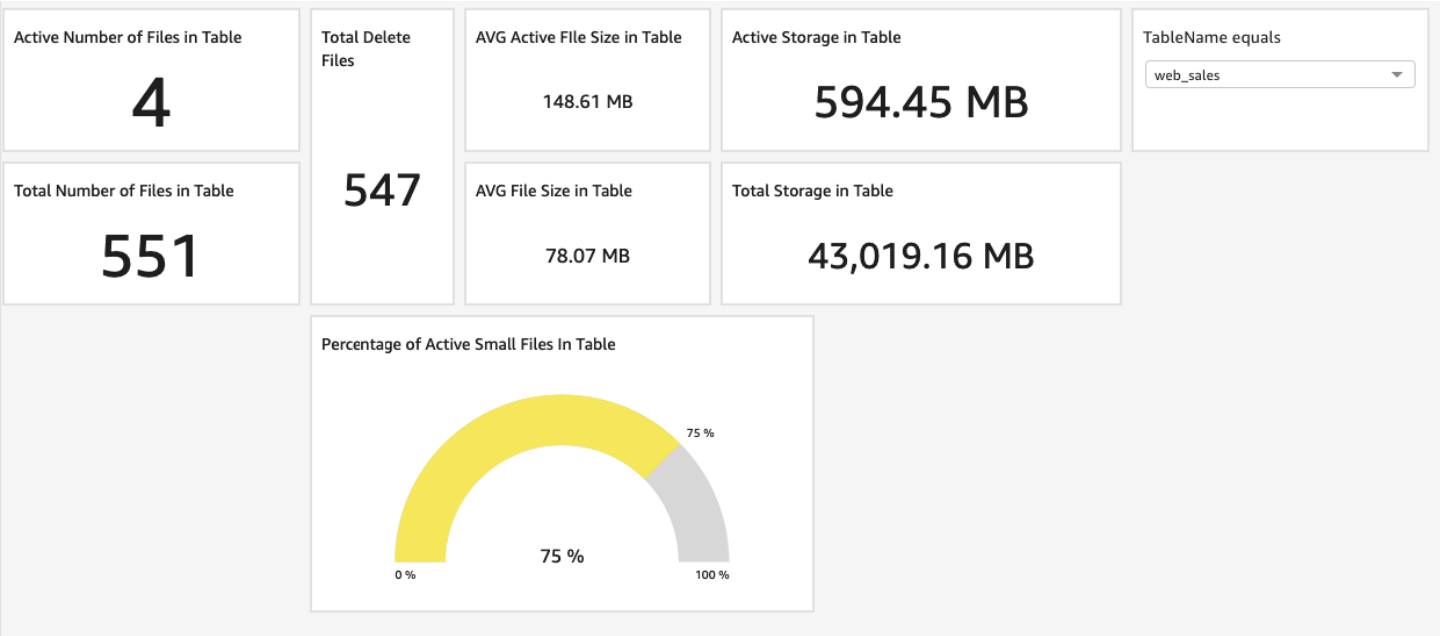
Iceberg ワークロードをモニタリングするには、メタデータテーブルの分析とメトリクスレポーターの使用の 2 つのオプションがあります。メトリクスレポーターは Iceberg バージョン 1.2 で導入され、REST カタログと JDBC カタログでのみ使用できます。

を使用している場合は AWS Glue Data Catalog、Iceberg が公開するメタデータテーブルの上にモニタリングを設定することで、Iceberg テーブルの状態に関するインサイトを得ることができます。

モニタリングは、パフォーマンス管理とトラブルシューティングに不可欠です。例えば、Iceberg テーブルのパーティションが小さなファイルの特定の割合に達すると、ワークロードは圧縮ジョブを開始してファイルをより大きなファイルに統合できます。これにより、クエリが許容レベルを超えて遅くなります。

テーブルレベルのモニタリング

次の画面は、Amazon Quick Sight で作成されたテーブルモニタリングダッシュボードを示しています。このダッシュボードは、Spark SQL を使用して Iceberg メタデータテーブルをクエリし、アクティブなファイル数や合計ストレージ数などの詳細なメトリクスをキャプチャします。この情報は、運用上の目的で AWS Glue テーブルに保存されます。最後に、次の図に示すように、Amazon Athena を使用して Quick Sight ダッシュボードが作成されます。この情報は、システム内の特定の問題を特定して対処するのに役立ちます。



Quick Sight ダッシュボードの例は、Iceberg テーブルの次の主要業績評価指標 (KPIs) を収集します。

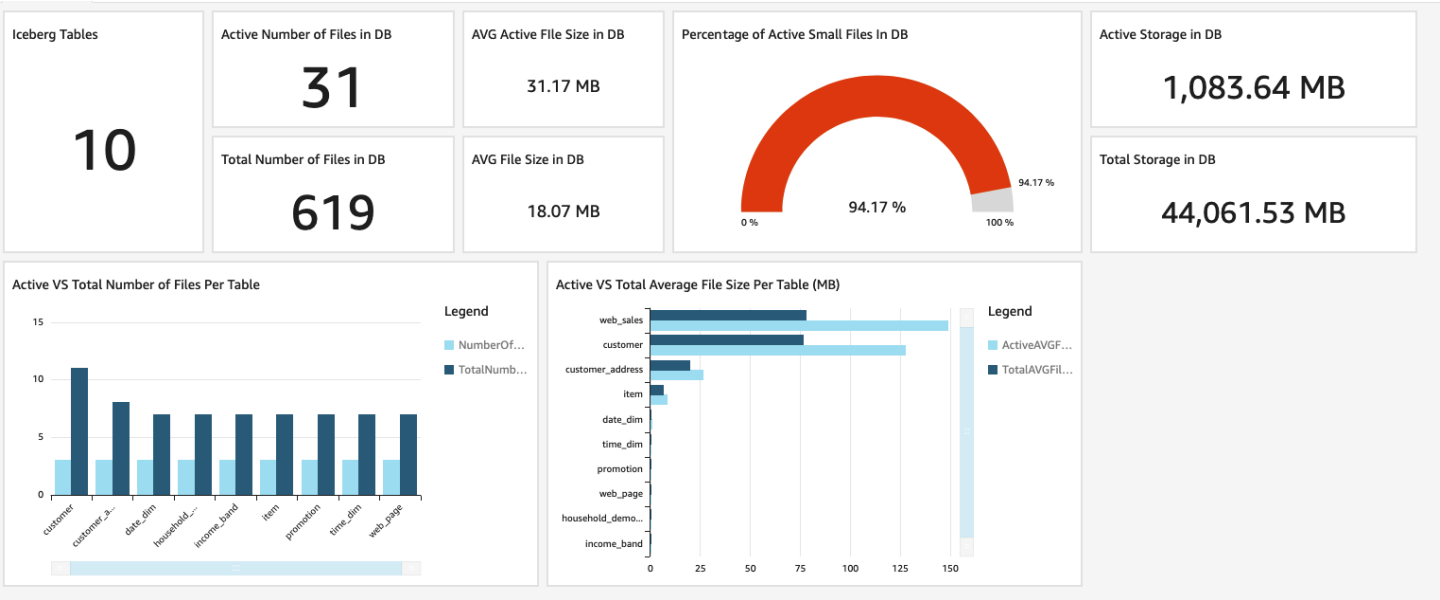
KPI	説明	クエリ
ファイルの数	Iceberg テーブル内のファイルの数 (すべてのスナップショット用)	`select count(*) from <catalog.database.table_name>.all_files`
アクティブなファイルの数	Iceberg テーブルの最後のスナップショットにあるアクティブなファイルの数	`select count(*) from <catalog.database.table_name>.files`
平均ファイルサイズ	Iceberg テーブル内のすべてのファイルの平均ファイルサイズ、メガバイト単位	`select avg(file_size_in_bytes)/1000000 from <catalog.database.table_name>.all_files`
平均アクティブファイルサイズ	Iceberg テーブル内のアクティブなファイルの平均ファイルサイズ、メガバイト単位	`select avg(file_size_in_bytes)/1000000 from <catalog.database.table_name>.files`
小さいファイルの割合	100 MB 未満のアクティブなファイルの割合	`select cast(sum( case when file_size_in_bytes < 100000000 then 1 else 0 end)*100/count(*) as decimal(10,2)) from <catalog.database.table_name>.files`
合計ストレージサイズ	孤立したファイルと Amazon S3 オブジェクトバージョンを除く、テーブル内のすべての	`select sum(file_size_in_bytes)/1000000`

KPI	説明	クエリ
	ファイルの合計サイズ (有効になっている場合)	`from <catalog.database.table_name>.all_files`
アクティブなストレージの合計サイズ	特定のテーブルの現在のスナップショット内のすべてのファイルの合計サイズ	`select sum(file_size_in_bytes)/1000000 from <catalog.database.table_name>.files`

ダッシュボードの作成の詳細については、Quick Sight ドキュメントを参照してください。

データベースレベルのモニタリング

次の例は、Quick Sight で作成されたモニタリングダッシュボードを示し、Iceberg テーブルのコレクションに関するデータベースレベルの KPIs の概要を提供します。



このダッシュボードは、次の KPIs。

KPI	説明	クエリ
ファイル数	Iceberg データベース内のファイルの数 (すべてのスナップショット用)	このダッシュボードは、前のセクションで提供されたテーブルレベルのクエリを使用し、結果を統合します。
アクティブなファイルの数	Iceberg データベース内のアクティブなファイルの数 (Iceberg テーブルの最後のスナップショットに基づく)	
平均ファイルサイズ	Iceberg データベース内のすべてのファイルの平均ファイルサイズ、メガバイト単位	
平均アクティブファイルサイズ	Iceberg データベース内のすべてのアクティブなファイルの平均ファイルサイズ、メガバイト単位	
小さなファイルの割合	Iceberg データベースで 100 MB 未満のアクティブなファイルの割合	
合計ストレージサイズ	孤立したファイルと Amazon S3 オブジェクトバージョンを除く、データベース内のすべてのファイルの合計サイズ (有効の場合)	
アクティブなストレージの合計サイズ	データベース内のすべてのテーブルの現在のスナップショット内のすべてのファイルの合計サイズ	

予防メンテナンス

前のセクションで説明したモニタリング機能を設定することで、事後的な角度ではなく予防的なテーブルメンテナンスにアプローチできます。たとえば、テーブルレベルとデータベースレベルのメトリクスを使用して、次のようなアクションをスケジュールできます。

- テーブルが N 個の小さなファイルに達したときに小さなファイルをグループ化するには、ビンパッキング圧縮を使用します。
- テーブルが特定のパーティション内の N 個の削除ファイルに達したときに、ビンパッキング圧縮を使用して削除ファイルをマージします。
- 合計ストレージがアクティブストレージの X 倍になったら、スナップショットを削除して、圧縮済みの小さなファイルを削除します。

での Apache Iceberg のガバナンスとアクセスコントロール AWS

Apache Iceberg は と統合 AWS Lake Formation して、データガバナンスを簡素化します。この統合により、データレイク管理者は Iceberg テーブルにセルレベルのアクセス許可を割り当てることができます。Amazon Athena と を使用して Iceberg テーブルをクエリする例については AWS Lake Formation、ブログ記事「Amazon Athena を使用して Amazon Athena」および「 を使用してアカウント間のきめ細かなアクセス許可 AWS Lake Formation」を参照してください AWS 。

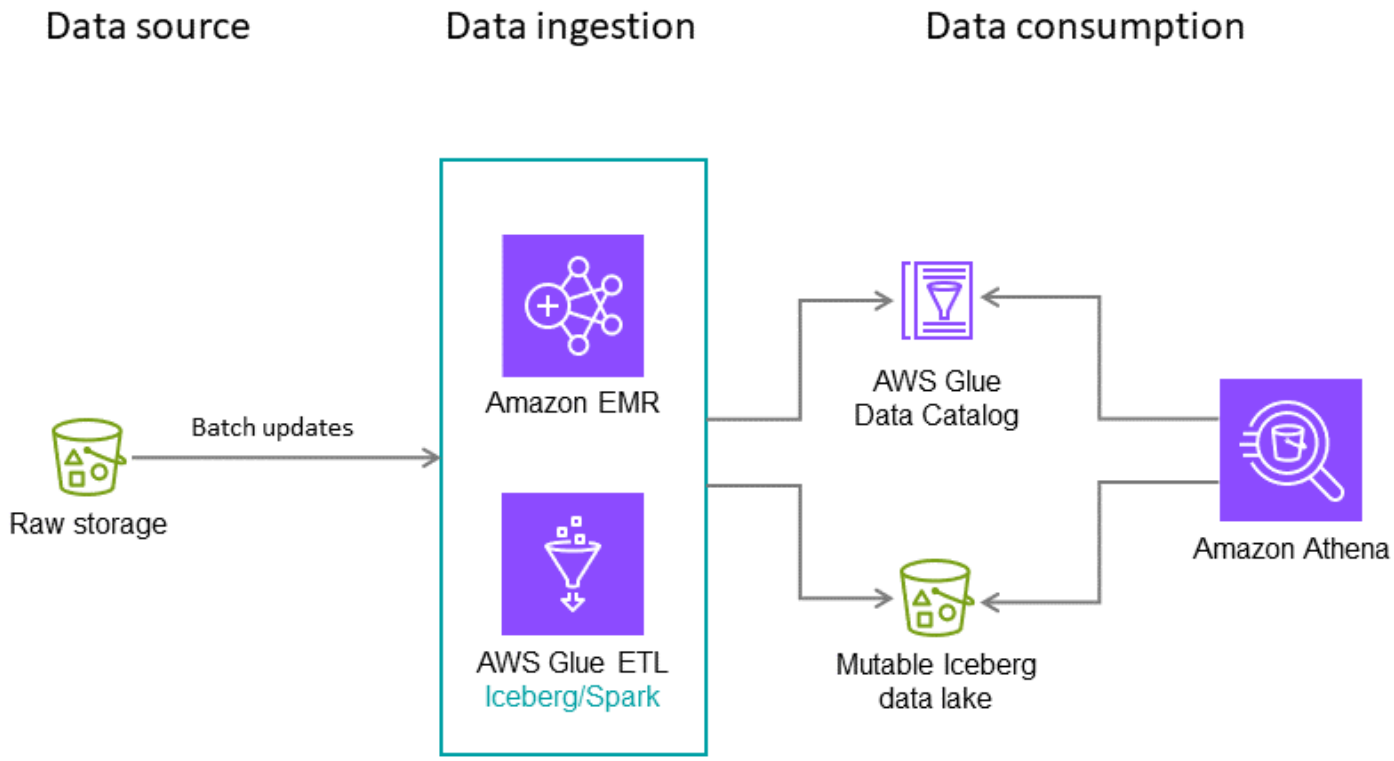
での Apache Iceberg のリファレンスアーキテクチャ AWS

このセクションでは、バッチ取り込みや、バッチデータ取り込みとストリーミングデータ取り込みを組み合わせたデータレイクなど、さまざまなユースケースでベストプラクティスを適用する方法の例を示します。

毎晩のバッチ取り込み

この架空のユースケースでは、Iceberg テーブルがクレジットカード取引を毎晩取り込むとします。各バッチには増分更新のみが含まれており、ターゲットテーブルにマージする必要があります。1 年に数回、完全な履歴データを受信します。このシナリオでは、次のアーキテクチャと設定をお勧めします。

注: これは単なる例です。最適な設定は、データと要件によって異なります。



推奨事項:

- ファイルサイズ: Apache Spark タスクは 128 MB のチャンクでデータを処理するため、128 MB。
- 書き込みタイプ: copy-on-write。このガイドの前半で説明したように、このアプローチは、データが読み取りに最適化された方法で書き込まれるようにするのに役立ちます。

- パーティション変数: year/month/day。架空のユースケースでは、最近のデータを最も頻繁にクエリしますが、過去 2 年間のデータに対して完全なテーブルスキャンを実行することがあります。パーティショニングの目的は、ユースケースの要件に基づいて高速読み取りオペレーションを促進することです。
- ソート順: タイムスタンプ
- データカタログ: AWS Glue Data Catalog

バッチ取り込みとほぼリアルタイムの取り込みを組み合わせたデータレイク

アカウントとリージョン間でバッチデータとストリーミングデータを共有するデータレイクを Amazon S3 にプロビジョニングできます。アーキテクチャ図と詳細については、 AWS ブログ記事「Build a transactional data lake using Apache Iceberg AWS Glue」および「 と AWS Lake Formation Amazon Athena を使用したクロスアカウントデータ共有」を参照してください。

リソース

- での Iceberg フレームワークの使用 AWS Glue (AWS Glue ドキュメント)
- Iceberg (Amazon EMR ドキュメント)
- Apache Iceberg テーブルの使用 (Amazon Athena ドキュメント)
- 「Amazon S3 ドキュメント」
- クイックドキュメント
- Glue Catalog と Lake Formation のアクセス許可レプリケーション (GitHub リポジトリ)
- Apache Iceberg ドキュメント
- Apache Spark ドキュメント

寄稿者

以下の の担当者は、このガイド AWS を作成、共同作成、レビューしました。

寄稿者

- Stefano Sandona、ビッグデータ、ソリューションアーキテクト
- Imtiaz (Taz) Sayed、ソリューションアーキテクトテクニカルリーダー、分析
- Shana Schipers、ソリューションアーキテクト、ビッグデータ
- Amazon EMR、ソフトウェア開発エンジニア、Prashant Singh
- Arun A K、ソリューションアーキテクト、ビッグデータ、ETL
- ストリーミング、ソリューションアーキテクト、Francisco Morillo
- Amazon EMR、分析アーキテクト、Suthan Phillips
- ソリューションアーキテクト、Sercan Karaoglu
- Yonatan Dolan、分析スペシャリスト
- Guy Bachar、ソリューションアーキテクト
- ストリーミング、ソリューションアーキテクト、Sofia Zilberman
- Dan Stair、スペシャリストソリューションアーキテクト
- ソリューションアーキテクト、Sakti Mishra
- Amazon S3、プリンシパル製品マネージャー、Ron Ortloff
- 分析、ソリューションアーキテクト、David Zhang

レビュアー

- Amazon EMR、GM、Rick Sears
- Amazon EMR スペシャリスト、Linda OConnor
- Amazon EMR、ディレクター、Ian Meyers
- Amazon EMR Product Management、ディレクター、Vinita Ananth
- Jason Berkowitz、Product Manager、 AWS Lake Formation
- Amazon Redshift、Product Manager、Mahesh Mishra
- ビッグデータ、ソリューションアーキテクチャ、マネージャー、Vladimir Zlatkin
- Amazon EMR、分析アーキテクト、Karthik Prabhakar

- Vijay Jain、プロダクトマネージャー
- Amazon S3、Product Manager、Anupriti Warade
- データ、ソリューションアーキテクト、Ajit Tandale
- 製品マーケティングマネージャー、Gwen Chen
- Noritaka Seki Yama、プリンシパルアーキテクト、ビッグデータ

ドキュメント履歴

以下の表は、本ガイドの重要な変更点について説明したものです。今後の更新に関する通知を受け取る場合は、RSS フィード をサブスクライブできます。

変更	説明	日付
新規セクション	Iceberg テーブル形式仕様バージョン 3 の操作に関する情報を追加しました。	2025 年 11 月 26 日
修正	スナップショット手順セクションgc.enabled の設定に関する情報を修正しました。	2025 年 10 月 24 日
追加	Trino と PyIceberg を使用した Iceberg テーブルの操作に関する新しいセクションを追加し、テーブルの Iceberg への移行に関するセクションを拡張しました。	2025 年 8 月 12 日
更新	、Amazon EMR AWS Glue、Apache Iceberg の最新バージョンを反映するために、ガイド全体で情報を強化し、明確にしました。	2025 年 7 月 14 日
追加	Amazon Data Firehose を使用した Iceberg テーブルの操作に関する新しいセクションを追加しました。	2025 年 2 月 20 日
初版発行	—	2024 年 4 月 30 日

AWS 規範ガイダンスの用語集

以下は、 AWS 規範ガイダンスによって提供される戦略、ガイド、パターンで一般的に使用される用語です。エントリを提案するには、用語集の最後のフィードバックの提供リンクを使用します。

数字

7 Rs

アプリケーションをクラウドに移行するための 7 つの一般的な移行戦略。これらの戦略は、ガートナーが 2011 年に特定した 5 Rs に基づいて構築され、以下で構成されています。

- リファクタリング/アーキテクチャの再設計 — クラウドネイティブ特徴を最大限に活用して、俊敏性、パフォーマンス、スケーラビリティを向上させ、アプリケーションを移動させ、アーキテクチャを変更します。これには、通常、オペレーティングシステムとデータベースの移植が含まれます。例: オンプレミスの Oracle データベースを Amazon Aurora PostgreSQL 互換エディションに移行する。
- リプラットフォーム (リフトアンドリシェイプ) — アプリケーションをクラウドに移行し、クラウド機能を活用するための最適化レベルを導入します。例: お客様のオンプレミスの Oracle データベースを AWS クラウドの Oracle 用の Amazon Relational Database Service (Amazon RDS) に移行する。
- 再購入 (ドロップアンドショップ) — 通常、従来のライセンスから SaaS モデルに移行して、別の製品に切り替えます。例: 顧客関係管理 (CRM) システムを Salesforce.com に移行する。
- リホスト (リフトアンドシフト) — クラウド機能を活用するための変更を加えずに、アプリケーションをクラウドに移行します。例: お客様のオンプレミスの Oracle データベースを AWS クラウドの EC2 インスタンス上の Oracle に移行する。
- 再配置 (ハイパーバイザーレベルのリフトアンドシフト) – 新しいハードウェアを購入したり、アプリケーションを書き換えたり、既存の運用を変更したりすることなく、インフラストラクチャをクラウドに移行できます。オンプレミスプラットフォームから同じプラットフォームのクラウドサービスにサーバーを移行します。例： Microsoft Hyper-Vアプリケーションを に移行します AWS。
- 保持 (再アクセス) — アプリケーションをお客様のソース環境で保持します。これには、主要なリファクタリングを必要とするアプリケーションや、お客様がその作業を後日まで延期したいアプリケーション、およびそれらを移行するためのビジネス上の正当性がないため、お客様が保持するレガシーアプリケーションなどがあります。
- 廃止 — お客様のソース環境で不要になったアプリケーションを停止または削除します。

A

ABAC

「属性ベースのアクセス制御」をご覧ください。

抽象化されたサービス

「マネージドユーザー」をご覧ください。

ACID

「原子性、一貫性、分離性、耐久性 (ACID)」をご覧ください。

アクティブ/アクティブ移行

(双方向レプリケーションツールまたは二重書き込み操作を使用して) ソースデータベースとターゲットデータベースを同期させ、移行中に両方のデータベースが接続アプリケーションからのトランザクションを処理するデータベース移行方法。この方法では、1 回限りのカットオーバーの必要がなく、管理された小規模なバッチで移行できます。アクティブ/パッシブ移行よりも柔軟な方法ですが、さらに多くの作業が必要となります。

アクティブ/パッシブ移行

ソースデータベースとターゲットデータベースを同期させながら、データがターゲットデータベースにレプリケートされている間、接続しているアプリケーションからのトランザクションをソースデータベースのみで処理するデータベース移行方法。移行中、ターゲットデータベースはトランザクションを受け付けません。

集計関数

複数行に処理を行い、グループ全体を対象に単一の戻り値を計算する SQL 関数。集計関数の例としては、SUM や MAX などがあります。

AI

「人工知能」をご覧ください。

AIOps

「AI オペレーション」をご覧ください。

匿名化

データセット内の個人情報を完全に削除するプロセス。匿名化は個人のプライバシー保護に役立ちます。匿名化されたデータは、もはや個人データとは見なされません。

アンチパターン

繰り返し起こる問題に対して頻繁に用いられる解決策で、その解決策が逆効果であったり、効果がなかったり、代替案よりも効果が低かったりするもの。

アプリケーション制御

マルウェアからシステムを保護するために、承認されたアプリケーションのみを使用できるようにするセキュリティアプローチ。

アプリケーションポートフォリオ

アプリケーションの構築と維持にかかるコスト、およびそのビジネス価値を含む、組織が使用する各アプリケーションに関する詳細情報の集まり。この情報は、ポートフォリオの検出と分析プロセス の重要な要素であり、移行、モダナイズ、最適化するアプリケーションを特定し、優先順位を付けるのに役立ちます。

人工知能 (AI)

コンピューティングテクノロジーを使用し、学習、問題の解決、パターンの認識など、通常は人間に関連づけられる認知機能の実行に特化したコンピュータサイエンスの分野。詳細については、「人工知能 (AI) とは何ですか?」をご覧ください。

AI オペレーション (AIOps)

機械学習技術を使用して運用上の問題を解決し、運用上のインシデントと人の介入を減らし、サービス品質を向上させるプロセス。 AWS 移行戦略での AIOps の使用方法については、オペレーション統合ガイド を参照してください。

非対称暗号化

暗号化用のパブリックキーと復号用のプライベートキーから成る 1 組のキーを使用した、暗号化のアルゴリズム。パブリックキーは復号には使用されないため共有しても問題ありませんが、プライベートキーの利用は厳しく制限する必要があります。

原子性、一貫性、分離性、耐久性 (ACID)

エラー、停電、その他の問題が発生した場合でも、データベースのデータ有効性と運用上の信頼性を保証する一連のソフトウェアプロパティ。

属性ベースのアクセス制御 (ABAC)

部署、役職、チーム名など、ユーザーの属性に基づいてアクセス許可をきめ細かく設定する方法。詳細については、 AWS Identity and Access Management (IAM) ドキュメントの「 の ABAC AWS」を参照してください。

信頼できるデータソース

最も信頼性のある情報源とされるデータのプライマリーバージョンを保存する場所。匿名化、編集、仮名化など、データを処理または変更する目的で、信頼できるデータソースから他の場所にデータをコピーすることができます。

アベイラビリティーゾーン (AZ)

他のアベイラビリティーゾーンの障害から AWS リージョン 隔離され、同じリージョン内の他のアベイラビリティーゾーンへの低コストで低レイテンシーのネットワーク接続を提供する 内の別の場所。

AWS クラウド導入フレームワーク (AWS CAF)

組織がクラウドへの移行を成功させるための効率的で効果的な計画を立て AWS るための、 のガイドラインとベストプラクティスのフレームワークです。 AWS CAF は、ビジネス、人材、ガバナンス、プラットフォーム、セキュリティ、運用という 6 つの重点分野にガイダンスを整理しています。ビジネス、人材、ガバナンスの観点では、ビジネススキルとプロセスに重点を置き、プラットフォーム、セキュリティ、オペレーションの視点は技術的なスキルとプロセスに焦点を当てています。例えば、人材の観点では、人事 (HR)、人材派遣機能、および人材管理を扱うステークホルダーを対象としています。この観点から、 AWS CAF は人材開発、トレーニング、コミュニケーションに関するガイダンスを提供し、組織がクラウド導入を成功させるための準備を支援します。詳細については、AWS CAF ウェブサイトと AWS CAF のホワイトペーパー を参照してください。

AWS ワークロード認定フレームワーク (AWS WQF)

データベース移行ワークロードを評価し、移行戦略を推奨し、作業見積もりを提供するツール。AWS WQF は AWS Schema Conversion Tool (AWS SCT) に含まれています。データベーススキーマとコードオブジェクト、アプリケーションコード、依存関係、およびパフォーマンス特性を分析し、評価レポートを提供します。

B

不正なボット

個人や組織に混乱や損害を与えることを目的としたボット。

BCP

「ビジネス継続性計画 (BCP)」をご覧ください。

動作グラフ

リソースの動作とインタラクションを経時的に示した、一元的なインタラクティブビュー。Amazon Detective の動作グラフを使用すると、失敗したログオンの試行、不審な API 呼び出し、その他同様のアクションを調べることができます。詳細については、Detective ドキュメントの「動作グラフのデータ」を参照してください。

ビッグエンディアンシステム

最上位バイトを最初に格納するシステム。「エンディアン性」もご覧ください。

二項分類

バイナリ結果 (2 つの可能なクラスのうちの 1 つ) を予測するプロセス。例えば、お客様の機械学習モデルで「この E メールはスパムですか、それともスパムではありませんか」などの問題を予測する必要があるかもしれません。または「この製品は書籍ですか、車ですか」などの問題を予測する必要があるかもしれません。

ブルームフィルター

要素がセットのメンバーであるかどうかをテストするために使用される、確率的でメモリ効率の高いデータ構造。

ブルー/グリーンデプロイ

それぞれが独立しているが、同一の環境を 2 つ作成するデプロイ戦略。現在のアプリケーションバージョンを 1 つの環境 (ブルー) で実行し、新しいアプリケーションバージョンを別の環境 (グリーン) で実行します。この戦略は、最小限の影響で迅速にロールバックするのに役立ちます。

ボット

インターネット経由で自動タスクを実行し、人間のアクティビティややり取りをシミュレートするソフトウェアアプリケーション。インターネット上の情報のインデックスを作成するウェブクローラーなど、一部のボットは有用または有益です。悪質なボットと呼ばれる他のボットの中には、個人や組織を混乱させたり、損害を与えたりすることを意図したものもあります。

ボットネット

マルウェアに感染しており、ボットハーダーまたはボットオペレーターと呼ばれる単一の当事者によって制御されているボットのネットワーク。ボットネットは、ボットとその影響力を拡大する仕組みとして、非常によく知られています。

ブランチ

コードリポジトリに含まれる領域。リポジトリに最初に作成するブランチは、メインブランチといいます。既存のブランチから新しいブランチを作成し、その新しいブランチで機能を開発した

り、バグを修正したりできます。機能を構築するために作成するブランチは、通常、機能ブランチと呼ばれます。機能をリリースする準備ができたら、機能ブランチをメインブランチに統合します。詳細については、「ブランチの概要」(GitHub ドキュメント) を参照してください。

ブレークグラスアクセス

例外的な状況では、承認されたプロセスを通じて、ユーザーが AWS アカウント 通常アクセス許可を持たない にすばやくアクセスできるようにします。詳細については、 AWS Well-Architected ガイダンスの「ブレークグラス手順の実装」インジケータを参照してください。

ブラウンフィールド戦略

環境の既存インフラストラクチャ。システムアーキテクチャにブラウンフィールド戦略を導入する場合、現在のシステムとインフラストラクチャの制約に基づいてアーキテクチャを設計します。既存のインフラストラクチャを拡張している場合は、ブラウンフィールド戦略とグリーンフィールド戦略を融合させることもできます。

バッファキャッシュ

アクセス頻度が最も高いデータが保存されるメモリ領域。

ビジネス能力

価値を生み出すためにビジネスが行うこと (営業、カスタマーサービス、マーケティングなど)。マイクロサービスのアーキテクチャと開発の決定は、ビジネス能力によって推進できます。詳細については、AWSでのコンテナ化されたマイクロサービスの実行ホワイトペーパーの「ビジネス機能を中心に組織化」セクションを参照してください。

ビジネス継続性計画 (BCP)

大規模移行など、中断を伴うイベントが運用に与える潜在的な影響に対処し、ビジネスを迅速に再開できるようにする計画。

C

CAF

「AWS クラウド導入フレームワーク」を参照してください

カナリアデプロイ

エンドユーザーへのバージョンリリースを、時間をかけて段階的に行うこと。確信が持てたら新規バージョンをデプロイして、現在のバージョン全体を置き換えます。

CCoE

「Cloud Center of Excellence」を参照してください。

CDC

「変更データキャプチャ」を参照してください。

変更データキャプチャ (CDC)

データソース (データベーステーブルなど) の変更を追跡し、その変更に関するメタデータを記録するプロセス。CDC は、ターゲットシステムでの変更を監査またはレプリケートして同期を維持するなど、さまざまな目的に使用できます。

カオスエンジニアリング

障害や破壊的なイベントを意図的に導入して、システムの耐障害性をテストすること。AWS Fault Injection Service (AWS FIS) を使用して、 AWS ワークロードにストレスを与え、その応答を評価する実験を実行できます。

CI/CD

「継続的インテグレーションと継続的デリバリー」を参照してください。

分類

予測を生成するのに役立つ分類プロセス。分類問題の機械学習モデルは、離散値を予測します。離散値は、常に互いに区別されます。例えば、モデルがイメージ内に車があるかどうかを評価する必要がある場合があります。

クライアント側の暗号化

ターゲットがデータ AWS のサービス を受信する前のローカルでのデータの暗号化。

Cloud Center of Excellence (CCoE)

クラウドのベストプラクティスの作成、リソースの移動、移行のタイムラインの確立、大規模変革を通じて組織をリードするなど、組織全体のクラウド導入の取り組みを推進する学際的なチーム。詳細については、 AWS クラウド エンタープライズ戦略ブログの CCoE 投稿を参照してください。

クラウドコンピューティング

リモートデータストレージと IoT デバイス管理に通常使用されるクラウドテクノロジー。クラウドコンピューティングは、一般的に、エッジコンピューティングに接続されています。

クラウド運用モデル

IT 組織において、1 つ以上のクラウド環境を構築、成熟、最適化するために使用される運用モデル。詳細については、「クラウド運用モデルの構築」を参照してください。

導入のクラウドステージ

組織が、 AWS クラウドへの移行時に通常実行する 4 つの段階。

- プロジェクト — 概念実証と学習を目的として、クラウド関連のプロジェクトをいくつか実行する
- 基礎固め — お客様のクラウドの導入を拡大するための基礎的な投資 (ランディングゾーンの作成、CCoE の定義、運用モデルの確立など)
- 移行 — 個々のアプリケーションの移行
- 再発明 — 製品とサービスの最適化、クラウドでのイノベーション

これらのステージは、 AWS クラウド エンタープライズ戦略ブログのブログ記事「クラウドファーストへのジャーニー」と「導入のステージ」で Stephen Orban によって定義されました。AWS 移行戦略との関連性については、「移行準備ガイド」を参照してください。

CMDB

「構成管理データベース (CMDB)」を参照してください。

コードリポジトリ

ソースコードやその他の資産 (ドキュメント、サンプル、スクリプトなど) が保存され、バージョン管理プロセスを通じて更新される場所。一般的なクラウドリポジトリには、GitHub や Bitbucket Cloud があります。コードの各バージョンはブランチと呼ばれます。マイクロサービスの構造では、各リポジトリは 1 つの機能専用です。1 つの CI/CD パイプラインで複数のリポジトリを使用できます。

コールドキャッシュ

空である、または、かなり空きがある、もしくは、古いデータや無関係なデータが含まれているバッファキャッシュ。データベースインスタンスはメインメモリまたはディスクから読み取る必要があり、バッファキャッシュから読み取るよりも時間がかかるため、パフォーマンスに影響します。

コールドデータ

めったにアクセスされず、通常は過去のデータです。この種類のデータをクエリする場合、通常は低速なクエリでも問題ありません。このデータを低パフォーマンスで安価なストレージ階層またはクラスに移動すると、コストを削減することができます。

コンピュータビジョン (CV)

機械学習を使用してデジタルイメージやビデオといった、ビジュアル形式の情報を分析および抽出する AI の分野。例えば、Amazon SageMaker AI では、CV 用の画像処理アルゴリズムを利用できます。

設定ドリフト

ワークロードにおいて、設定が想定した状態から変化すること。これによって、ワークロードが非準拠になる可能性がありますが、この状態は、徐々に生じ、意図的なものではありません。

構成管理データベース (CMDB)

データベースとその IT 環境 (ハードウェアとソフトウェアの両方のコンポーネントとその設定を含む) に関する情報を保存、管理するリポジトリ。通常、CMDB のデータは、移行のポートフォリオの検出と分析の段階で使用します。

コンフォーマンスパック

コンプライアンスチェックとセキュリティチェックをカスタマイズするためにアセンブルできる AWS Config ルールと修復アクションのコレクション。YAML テンプレートを使用して、コンフォーマンスパックを AWS アカウント および リージョンの単一のエンティティとしてデプロイすることも、組織全体にデプロイすることもできます。詳細については、 AWS Config ドキュメントの「コンフォーマンスパック」を参照してください。

継続的インテグレーションと継続的デリバリー (CI/CD)

ソフトウェアリリースプロセスのソース、ビルド、テスト、ステージング、本番の各ステージを自動化するプロセス。CI/CD は一般的にパイプラインと呼ばれます。プロセスの自動化、生産性の向上、コード品質の向上、配信の加速化を可能にします。詳細については、「継続的デリバリーの利点」を参照してください。CD は継続的デプロイ (Continuous Deployment) の略語でもあります。詳細については「継続的デリバリーと継続的なデプロイ」を参照してください。

CV

「コンピュータビジョン」を参照してください。

D

保管中のデータ

ストレージ内にあるデータなど、常に自社のネットワーク内にあるデータ。

データ分類

ネットワーク内のデータを重要度と機密性に基づいて識別、分類するプロセス。データに適した保護および保持のコントロールを判断する際に役立つため、あらゆるサイバーセキュリティのリスク管理戦略において重要な要素です。データ分類は、 AWS Well-Architected フレームワークのセキュリティの柱のコンポーネントです。詳細については、「データ分類」を参照してください。

データドリフト

実稼働データと ML モデルのトレーニングに使用されたデータとの間に有意な差異が生じたり、入力データが時間の経過と共に有意に変化したりすることです。データドリフトは、ML モデル予測の全体的な品質、精度、公平性を低下させる可能性があります。

転送中のデータ

ネットワーク内 (ネットワークリソース間など) を活発に移動するデータ。

データメッシュ

非一元的で分散型のデータ所有権を持つとともに、一元的な管理およびガバナンスを行えるアーキテクチャフレームワーク。

データ最小化

厳密に必要なデータのみを収集し、処理するという原則。でデータ最小化を実践 AWS クラウドすることで、プライバシーリスク、コスト、分析のカーボンフットプリントを削減できます。

データ境界

AWS 環境内の一連の予防ガードレール。信頼できる ID のみが、期待されるネットワークから信頼できるリソースにアクセスできるようにします。詳細については、「 でのデータ境界の構築 AWS」を参照してください。

データの前処理

raw データをお客様の機械学習モデルで簡単に解析できる形式に変換すること。データの前処理とは、特定の列または行を削除して、欠落している、矛盾している、または重複する値に対処することを意味します。

データ出所

データの生成、送信、保存の方法など、データのライフサイクル全体を通じてデータの出所と履歴を追跡するプロセス。

データ件名

データを収集、処理している個人。

データウェアハウス

分析などのビジネスインテリジェンスをサポートするデータ管理システム。データウェアハウスには、一般的に、大量の履歴データが含まれており、多くの場合、それらはクエリや分析に使用されます。

データベース定義言語 (DDL)

データベース内のテーブルやオブジェクトの構造を作成または変更するためのステートメントまたはコマンド。

データベース操作言語 (DML)

データベース内の情報を変更 (挿入、更新、削除) するためのステートメントまたはコマンド。

DDL

「データベース定義言語」を参照してください。

ディープアンサンブル

予測のために複数の深層学習モデルを組み合わせます。ディープアンサンブルを使用して、より正確な予測を取得したり、予測の不確実性を推定したりできます。

深層学習

人工ニューラルネットワークの複数層を使用して、入力データと対象のターゲット変数の間のマッピングを識別する機械学習サブフィールド。

多層防御

一連のセキュリティメカニズムとコントロールをコンピュータネットワーク全体に層状に重ねて、ネットワークとその内部にあるデータの機密性、整合性、可用性を保護する情報セキュリティの手法。この戦略を採用するときは AWS、リソースの保護に役立つように、 AWS Organizations 構造の異なるレイヤーに複数のコントロールを追加します。たとえば、多層防御アプローチでは、多要素認証、ネットワークセグメンテーション、暗号化を組み合わせることができます。

委任管理者

では AWS Organizations、互換性のあるサービスが AWS メンバーアカウントを登録して組織のアカウントを管理し、そのサービスのアクセス許可を管理できます。このアカウントを、そのサービスの委任管理者と呼びます。詳細、および互換性のあるサービスの一覧は、 AWS

Organizations ドキュメントの「AWS Organizationsで使用できるサービス」を参照してください。

トラブルシューティング

アプリケーション、新機能、コードの修正をターゲットの環境で利用できるようにするプロセス。デプロイでは、コードベースに変更を施した後、アプリケーションの環境でそのコードベースを構築して実行します。

開発環境

「環境」を参照してください。

検出管理

イベントが発生したときに、検出、ログ記録、警告を行うように設計されたセキュリティコントロール。これらのコントロールは副次的な防衛手段であり、実行中の予防的コントロールをすり抜けたセキュリティイベントをユーザーに警告します。詳細については、「 AWSでのセキュリティコントロールの実装」の「検出的コントロール」を参照してください。

開発バリューストリームマッピング (DVSM)

ソフトウェア開発ライフサイクルのスピードと品質に悪影響を及ぼす制約を特定し、優先順位を付けるために使用されるプロセス。DVSM は、もともとリーンマニュファクチャリング・プラクティスのために設計されたバリューストリームマッピング・プロセスを拡張したものです。ソフトウェア開発プロセスを通じて価値を創造し、動かすために必要なステップとチームに焦点を当てています。

デジタルツイン

建物、工場、産業機器、生産ラインなど、現実世界のシステムを仮想的に表現したものです。デジタルツインは、予知保全、リモートモニタリング、生産最適化をサポートします。

ディメンションテーブル

スタースキーマにおいて、ファクトテーブルの定量データに関するデータ属性が含まれる小さいテーブル。ディメンションテーブルの属性は、通常、テキストフィールド、またはテキストのように扱える個別の数値で示されます。これらの属性は、一般的に、クエリの制約、フィルタリング、結果セットのラベル付けに使用されます。

ディザスタ

ワークロードまたはシステムが、導入されている主要な場所でのビジネス目標の達成を妨げるイベント。これらのイベントは、自然災害、技術的障害、または意図しない設定ミスやマルウェア攻撃などの人間の行動の結果である場合があります。

ディザスタリカバリ (DR)

ディザスタによるダウンタイムとデータ損失を最小限に抑えるための戦略とプロセス。詳細については、 AWS Well-Architected フレームワークの「 でのワークロードのディザスタリカバリ AWS: クラウドでのリカバリ」を参照してください。

DML

「データベース操作言語」を参照してください。

ドメイン駆動型設計

各コンポーネントが提供している変化を続けるドメイン、またはコアビジネス目標にコンポーネントを接続して、複雑なソフトウェアシステムを開発するアプローチ。この概念は、エリック・エヴァンスの著書、Domain-Driven Design: Tackling Complexity in the Heart of Software (ドメイン駆動設計:ソフトウェアの中心における複雑さへの取り組み) で紹介されています (ボストン: Addison-Wesley Professional、2003)。strangler fig パターンでドメイン駆動型設計を使用する方法の詳細については、「コンテナと Amazon API Gateway を使用して、従来の Microsoft ASP.NET (ASMX) ウェブサービスを段階的にモダナイズ」を参照してください。

DR

「ディザスタリカバリ」を参照してください。

ドリフト検出

ベースライン設定からの偏差を追跡します。たとえば、 AWS CloudFormation を使用してシステムリソースのドリフトを検出したり、 を使用して AWS Control Tower 、ガバナンス要件のコンプライアンスに影響を与える可能性のあるランディングゾーンの変更を検出したりできます。

DVSM

「開発バリューストリームマッピング」を参照してください。

E

EDA

「探索的データ分析」を参照してください。

EDI

「電子データ交換」を参照してください。

エッジコンピューティング

IoT ネットワークのエッジにあるスマートデバイスの計算能力を高めるテクノロジー。クラウドコンピューティングと比較すると、エッジコンピューティングは通信レイテンシーを短縮し、応答時間を改善できます。

電子データ交換 (EDI)

組織間で行う、ビジネスドキュメントの自動交換。詳細については、「電子データ交換とは」を参照してください。

暗号化

人間が読み取り可能なプレーンテキストデータを暗号文に変換するコンピューティング処理。

暗号化キー

暗号化アルゴリズムが生成した、ランダム化されたビットからなる暗号文字列。キーの長さは決まっておらず、各キーは予測できないように、一意になるように設計されています。

エンディアン

コンピュータメモリにバイトが格納される順序。ビッグエンディアンシステムでは、最上位バイトが最初に格納されます。リトルエンディアンシステムでは、最下位バイトが最初に格納されます。

エンドポイント

「サービスエンドポイント」を参照してください。

エンドポイントサービス

仮想プライベートクラウド (VPC) 内でホストして、他のユーザーと共有できるサービス。を使用してエンドポイントサービスを作成し AWS PrivateLink 、他の AWS アカウント または AWS Identity and Access Management (IAM) プリンシパルにアクセス許可を付与できます。これらのアカウントまたはプリンシパルは、インターフェイス VPC エンドポイントを作成することで、エンドポイントサービスにプライベートに接続できます。詳細については、Amazon Virtual Private Cloud (Amazon VPC) ドキュメントの「エンドポイントサービスを作成する」を参照してください。

エンタープライズリソースプランニング (ERP)

エンタープライズの主要なビジネスプロセス (会計、MES、プロジェクト管理など) を自動化および管理するシステム。

エンベロープ暗号化

暗号化キーを、別の暗号化キーを使用して暗号化するプロセス。詳細については、 AWS Key Management Service (AWS KMS) ドキュメントの「エンベロープ暗号化」を参照してください。

環境

実行中のアプリケーションのインスタンス。クラウドコンピューティングにおける一般的な環境の種類は以下のとおりです。

- 開発環境 — アプリケーションのメンテナンスを担当するコアチームのみが使用できる、実行中のアプリケーションのインスタンス。開発環境は、上位の環境に昇格させる変更をテストするときに使用します。このタイプの環境は、テスト環境と呼ばれることもあります。
- 下位環境 — 初期ビルドやテストに使用される環境など、アプリケーションのすべての開発環境。
- 本番環境 — エンドユーザーがアクセスできる、実行中のアプリケーションのインスタンス。CI/CD パイプラインでは、本番環境が最後のデプロイ環境になります。
- 上位環境 — コア開発チーム以外のユーザーがアクセスできるすべての環境。これには、本番環境、本番前環境、ユーザー承認テスト環境などが含まれます。

エピック

アジャイル方法論で、お客様の作業の整理と優先順位付けに役立つ機能カテゴリ。エピックでは、要件と実装タスクの概要についてハイレベルな説明を提供します。例えば、 AWS CAF セキュリティエピックには、ID とアクセスの管理、検出コントロール、インフラストラクチャセキュリティ、データ保護、インシデント対応が含まれます。 AWS 移行戦略のエピックの詳細については、プログラム実装ガイドを参照してください。

ERP

「エンタープライズリソース計画」を参照してください。

探索的データ分析 (EDA)

データセットを分析してその主な特性を理解するプロセス。お客様は、データを収集または集計してから、パターンの検出、異常の検出、および前提条件のチェックのための初期調査を実行します。EDA は、統計の概要を計算し、データの可視化を作成することによって実行されます。

F

ファクトテーブル

スタースキーマの中央にあるテーブル。ビジネスオペレーションに関する定量的データが保存されます。一般的に、ファクトテーブルは、2 種類の列で構成されます。1 つは測定値が含まれる列、もう 1 つはディメンションテーブルへの外部キーが含まれる列です。

フェイルファスト

開発ライフサイクルを短縮するために、頻繁かつ段階的にテストを行う哲学であり、アジャイルアプローチでは、この考え方がきわめて重要です。

障害分離境界

では AWS クラウド、障害の影響を制限し、ワークロードの耐障害性を高めるのに役立つアベイラビリティーゾーン AWS リージョン、コントロールプレーン、データプレーンなどの境界。詳細については、「AWS 障害分離境界」を参照してください。

機能ブランチ

「ブランチ」を参照してください。

特徴量

お客様が予測に使用する入力データ。例えば、製造コンテキストでは、特徴量は製造ラインから定期的にキャプチャされるイメージの可能性もあります。

特徴量重要度

モデルの予測に対する特徴量の重要性。これは通常、Shapley Additive Deskonations (SHAP) や積分勾配など、さまざまな手法で計算できる数値スコアで表されます。詳細については、「 を使用した機械学習モデルの解釈可能性 AWS」を参照してください。

機能変換

追加のソースによるデータのエンリッチ化、値のスケーリング、単一のデータフィールドからの複数の情報セットの抽出など、機械学習プロセスのデータを最適化すること。これにより、機械学習モデルはデータの恩恵を受けることができます。例えば、「2021-05-27 00:15:37」の日付を「2021 年」、「5 月」、「木」、「15」に分解すると、学習アルゴリズムがさまざまなデータコンポーネントに関連する微妙に異なるパターンを学習するのに役立ちます。

数ショットプロンプト

LLM に、タスクと望ましい出力を示す例を少数提示した後に、類似のタスクを実行させること。この手法は、プロンプトに記述された例 (ショット) からモデルが学習する「インコンテキスト学

習」の一種です。数ショットプロンプトは、特定のフォーマット、推論、専門知識が必要なタスクに効果的です。「ゼロショットプロンプト」も参照してください。

FGAC

「きめ細かなアクセス制御」を参照してください。

きめ細かなアクセス制御 (FGAC)

複数の条件を使用してアクセス要求を許可または拒否すること。

フラッシュカット移行

変更データのキャプチャによる継続的なデータ複製を利用して、段階的なアプローチではなく、可能な限り短時間でデータを移行するデータベース移行方法。目的はダウンタイムを最小限に抑えることです。

FM

「基盤モデル」を参照してください。

基盤モデル (FM)

大規模な深層学習ニューラルネットワークであり、一般化およびラベル付けされていないデータからなる大規模データセットでトレーニングされています。FM により、言語理解、テキストおよび画像生成、自然言語での会話といった、一般的な各種タスクを実行できます。詳細については、「基盤モデルとは何ですか?」を参照してください。

G

生成 AI

AI モデルのサブセット。大量のデータでトレーニングされており、シンプルなテキストプロンプトを使用して、画像、動画、テキスト、オーディオなどの新しいコンテンツやアーティファクトを作成できます。詳細については、「生成 AI とは何ですか?」を参照してください。

ジオブロッキング

「地理的制限」を参照してください。

地理的制限 (ジオブロッキング)

特定の国のユーザーがコンテンツ配信にアクセスできないようにするための、Amazon CloudFront のオプション。アクセスを許可する国と禁止する国は、許可リストまたは禁止リスト

を使って指定します。詳細については、CloudFront ドキュメントの「コンテンツの地理的ディストリビューションの制限」を参照してください。

Gitflow ワークフロー

下位環境と上位環境が、ソースコードリポジトリでそれぞれ異なるブランチを使用する方法。Gitflow ワークフローは古いと見なされている方法であり、トランクベースのワークフローは推奨されている新しい方法です。

ゴールデンイメージ

システムまたはソフトウェアのスナップショットであり、システムまたはソフトウェアの新規インスタンスをデプロイするテンプレートとして使用されます。製造の例で言えば、ゴールデンイメージを使用すると、複数のデバイスにソフトウェアをプロビジョニングして、デバイス製造オペレーションの速度、スケーラビリティ、生産性を向上させることができます。

グリーンフィールド戦略

新しい環境に既存のインフラストラクチャが存在しないこと。システムアーキテクチャにグリーンフィールド戦略を導入する場合、既存のインフラストラクチャ (別名ブラウンフィールド) との互換性の制約を受けることなく、あらゆる新しいテクノロジーを選択できます。既存のインフラストラクチャを拡張している場合は、ブラウンフィールド戦略とグリーンフィールド戦略を融合させることもできます。

ガードレール

組織単位 (OU) 全般のリソース、ポリシー、コンプライアンスを管理するのに役立つ概略的なルール。予防ガードレールは、コンプライアンス基準に一致するようにポリシーを実施します。これらは、サービスコントロールポリシーと IAM アクセス許可の境界を使用して実装されます。検出ガードレールは、ポリシー違反やコンプライアンス上の問題を検出し、修復のためのアラートを発信します。これらは AWS Config、、 AWS Security Hub CSPM、Amazon GuardDuty、、 AWS Trusted Advisor Amazon Inspector、およびカスタム AWS Lambda チェックを使用して実装されます。

H

HA

「高可用性」を参照してください。

異種混在データベースの移行

別のデータベースエンジンを使用するターゲットデータベースへお客様の出典データベースの移行 (例えば、Oracle から Amazon Aurora)。異種間移行は通常、アーキテクチャの再設計作業の一部であり、スキーマの変換は複雑なタスクになる可能性があります。AWS は、スキーマの変換に役立つ AWS SCTを提供します。

高可用性 (HA)

課題や災害が発生した場合に、介入なしにワークロードを継続的に運用できること。HA システムは、自動的にフェイルオーバーし、一貫して高品質のパフォーマンスを提供し、パフォーマンスへの影響を最小限に抑えながらさまざまな負荷や障害を処理するように設計されています。

ヒストリアンのモダナイゼーション

製造業のニーズによりよく応えるために、オペレーションテクノロジー (OT) システムをモダナイズし、アップグレードするためのアプローチ。ヒストリアンは、工場内のさまざまなソースからデータを収集して保存するために使用されるデータベースの一種です。

ホールドアウトデータ

機械学習モデルのトレーニング用データセットから保留される、ラベル付き履歴データの一部。ホールドアウトデータを使用すると、モデル予測をホールドアウトデータと比較して、モデルのパフォーマンスを評価できます。

同種データベースの移行

お客様の出典データベースを、同じデータベースエンジンを共有するターゲットデータベース (Microsoft SQL Server から Amazon RDS for SQL Server など) に移行する。同種間移行は、通常、リホストまたはリプラットフォーム化の作業の一部です。ネイティブデータベースユーティリティを使用して、スキーマを移行できます。

ホットデータ

リアルタイムデータや最近の翻訳データなど、頻繁にアクセスされるデータ。通常、このデータには高速なクエリ応答を提供する高性能なストレージ階層またはクラスが必要です。

ホットフィックス

本番環境の重大な問題を修正するために緊急で配布されるプログラム。緊急性が高いため、通常の DevOps のリリースワークフローからは外れた形で実施されます。

ハイパーケア期間

カットオーバー直後、移行したアプリケーションを移行チームがクラウドで管理、監視して問題に対処する期間。通常、この期間は 1～4 日です。ハイパーケア期間が終了すると、アプリケーションに対する責任は一般的に移行チームからクラウドオペレーションチームに移ります。

I

IaC

「Infrastructure as Code」を参照してください。

ID ベースのポリシー

AWS クラウド 環境内のアクセス許可を定義する 1 つ以上の IAM プリンシパルにアタッチされたポリシー。

アイドル状態のアプリケーション

90 日間の平均的な CPU およびメモリ使用率が 5～20% のアプリケーション。移行プロジェクトでは、これらのアプリケーションを廃止するか、オンプレミスに保持するのが一般的です。

IIoT

「インダストリアル IoT」を参照してください。

イミュータブルインフラストラクチャ

既存インフラストラクチャの更新、パッチ適用、変更などを行わずに、本番環境ワークロードに使用する新規インフラストラクチャをデプロイするモデル。本質的に、イミュータブルインフラストラクチャは、ミュータブルインフラストラクチャよりも一貫性、信頼性、予測性に優れています。詳細については、 AWS Well-Architected フレームワークにある「イミュータブルインフラストラクチャを使用してデプロイする」のベストプラクティスを参照してください。

インバウンド (受信) VPC

AWS マルチアカウントアーキテクチャでは、アプリケーションの外部からネットワーク接続を受け入れ、検査し、ルーティングする VPC。AWS Security Reference Architecture では、アプリケーションとより広範なインターネット間の双方向のインターフェイスを保護するために、インバウンド、アウトバウンド、インスペクションの各 VPC を使用してネットワークアカウントを設定することを推奨しています。

増分移行

アプリケーションを 1 回ですべてカットオーバーするのではなく、小さい要素に分けて移行するカットオーバー戦略。例えば、最初は少数のマイクロサービスまたはユーザーのみを新しいシステムに移行する場合があります。すべてが正常に機能することを確認できたら、残りのマイクロサービスやユーザーを段階的に移行し、レガシーシステムを廃止できるようにします。この戦略により、大規模な移行に伴うリスクが軽減されます。

インダストリー 4.0

2016 年に Klaus Schwab 氏が提唱した用語で、接続、リアルタイムデータ、オートメーション、分析、AI/ML の進歩による、ビジネスプロセスのモダナイズを意味します。

インフラストラクチャ

アプリケーションの環境に含まれるすべてのリソースとアセット。

Infrastructure as Code (IaC)

アプリケーションのインフラストラクチャを一連の設定ファイルを使用してプロビジョニングし、管理するプロセス。IaC は、新しい環境を再現可能で信頼性が高く、一貫性のあるものにするため、インフラストラクチャを一元的に管理し、リソースを標準化し、スケールを迅速に行えるように設計されています。

インダストリアル IoT (IIoT)

製造、エネルギー、自動車、ヘルスケア、ライフサイエンス、農業などの産業部門におけるインターネットに接続されたセンサーやデバイスの使用。詳細については、「インダストリアル IoT (IIoT) デジタルトランスフォーメーション戦略の構築」を参照してください。

インスペクション VPC

AWS マルチアカウントアーキテクチャでは、VPC (同一または異なる 内 AWS リージョン)、インターネット、オンプレミスネットワーク間のネットワークトラフィックの検査を管理する一元化された VPCs。AWS Security Reference Architecture では、アプリケーションとより広範なインターネット間の双方向のインターフェイスを保護するために、インバウンド、アウトバウンド、インスペクションの各 VPC を使用してネットワークアカウントを設定することを推奨しています。

IoT

インターネットまたはローカル通信ネットワークを介して他のデバイスやシステムと通信する、センサーまたはプロセッサが組み込まれた接続済み物理オブジェクトのネットワーク。詳細については、「IoT とは」を参照してください。

解釈可能性

機械学習モデルの特性で、モデルの予測がその入力にどのように依存するかを人間が理解できる度合いを表します。詳細については、「 を使用した機械学習モデルの解釈可能性 AWS」を参照してください。

IoT

「IoT」を参照してください。

IT 情報ライブラリ (ITIL)

IT サービスを提供し、これらのサービスをビジネス要件に合わせるための一連のベストプラクティス。ITIL は ITSM の基盤を提供します。

IT サービス管理 (ITSM)

組織の IT サービスの設計、実装、管理、およびサポートに関連する活動。クラウドオペレーションと ITSM ツールの統合については、オペレーション統合ガイドを参照してください。

ITIL

「IT 情報ライブラリ」を参照してください。

ITSM

「IT サービス管理」を参照してください。

L

ラベルベースアクセス制御 (LBAC)

強制アクセス制御 (MAC) の実装で、ユーザーとデータ自体にそれぞれセキュリティラベル値が明示的に割り当てられます。ユーザーセキュリティラベルとデータセキュリティラベルが交差する部分によって、ユーザーに表示される行と列が決まります。

ランディングゾーン

ランディングゾーンは、スケーラブルで安全な、適切に設計されたマルチアカウント AWS 環境です。これは、組織がセキュリティおよびインフラストラクチャ環境に自信を持ってワークロードとアプリケーションを迅速に起動してデプロイできる出発点です。ランディングゾーンの詳細については、「安全でスケーラブルなマルチアカウント AWS 環境のセットアップ」を参照してください。

大規模言語モデル (LLM)

大量のデータで事前トレーニングされた深層学習 AI モデル。LLM では、質問への回答、ドキュメントの要約、他言語へのテキスト翻訳、文を完成させるなど、さまざまなタスクを実行できます。詳細については、「大規模言語モデル (LLM) とは何ですか?」を参照してください。

大規模な移行

300 台以上のサーバの移行。

LBAC

「ラベルベースアクセス制御」を参照してください。

最小特権

タスクの実行には必要最低限の権限を付与するという、セキュリティのベストプラクティス。詳細については、IAM ドキュメントの「最小特権アクセス許可を適用する」を参照してください。

リフトアンドシフト

「7 Rs」を参照してください。

リトルエンディアンシステム

最下位バイトを最初に格納するシステム。「エンディアン性」もご覧ください。

LLM

「大規模言語モデル」を参照してください。

下位環境

「環境」を参照してください。

M

機械学習 (ML)

パターン認識と学習にアルゴリズムと手法を使用する人工知能の一種。ML は、モノのインターネット (IoT) データなどの記録されたデータを分析して学習し、パターンに基づく統計モデルを生成します。詳細については、「機械学習」を参照してください。

メインブランチ

「ブランチ」を参照してください。

マルウェア

コンピュータのセキュリティやプライバシーを侵害するように設計されたソフトウェア。マルウェアは、コンピュータシステムの中断、機密情報の漏洩、不正アクセスを招く可能性があります。マルウェアの例には、ウイルス、ワーム、ランサムウェア、トロイの木馬、スパイウェア、キーロガーなどがあります。

マネージドサービス

AWS のサービス はインフラストラクチャレイヤー、オペレーティングシステム、プラットフォーム AWS を運用し、エンドポイントにアクセスしてデータを保存および取得します。マネージドサービスの例として、Amazon Simple Storage Service (Amazon S3) と Amazon DynamoDB が挙げられます。このサービスは、抽象化されたサービスとも呼ばれます。

製造実行システム (MES)

生産プロセスを追跡、モニタリング、文書化、制御するソフトウェアシステムであり、工場では、これによって、原材料から製品を完成させます。

MAP

「Migration Acceleration Program」を参照してください。

メカニズム

ツールを作成してその導入を推進し、導入結果を調べて調整を行うための包括的なプロセス。メカニズムとは、運用中にそれ自体を強化し改善するサイクルを意味します。詳細については、AWS 「 Well-Architected フレームワーク」の「メカニズムの構築」を参照してください。

メンバーアカウント

組織の一部である管理アカウント AWS アカウント 以外のすべて AWS Organizations。 アカウントが 組織のメンバーになることができるのは、一度に 1 つのみです。

MES

「製造実行システム」を参照してください。

Message Queuing Telemetry Transport (MQTT)

発行/サブスクライブのパターンに基づく、軽量のマシンツーマシン (M2M) 通信プロトコルであり、リソースに限りのある IoT デバイスに使用されます。

マイクロサービス

明確に定義された API を介して通信し、通常は小規模な自己完結型のチームが所有する、小規模で独立したサービスです。例えば、保険システムには、販売やマーケティングなどのビジネス

機能、または購買、請求、分析などのサブドメインにマッピングするマイクロサービスが含まれる場合があります。マイクロサービスの利点には、俊敏性、柔軟なスケーリング、容易なデプロイ、再利用可能なコード、回復力などがあります。詳細については、AWS「サーバーレスサービスを使用したマイクロサービスの統合」を参照してください。

マイクロサービスアーキテクチャ

各アプリケーションプロセスをマイクロサービスとして実行する独立したコンポーネントを使用してアプリケーションを構築するアプローチ。これらのマイクロサービスは、軽量 API を使用して、明確に定義されたインターフェイスを介して通信します。このアーキテクチャの各マイクロサービスは、アプリケーションの特定の機能に対する需要を満たすように更新、デプロイ、およびスケーリングできます。詳細については、「 でのマイクロサービスの実装 AWS」を参照してください。

Migration Acceleration Program (MAP)

組織がクラウドに移行するための強力な運用基盤を構築し、移行の初期コストを相殺するのに役立つコンサルティングサポート、トレーニング、サービスを提供する AWS プログラム。MAP には、組織的な方法でレガシー移行を実行するための移行方法論と、一般的な移行シナリオを自動化および高速化する一連のツールが含まれています。

大規模な移行

アプリケーションポートフォリオの大部分を次々にクラウドに移行し、各ウェーブでより多くのアプリケーションを高速に移動させるプロセス。この段階では、以前の段階から学んだベストプラクティスと教訓を使用して、移行ファクトリー チーム、ツール、プロセスのうち、オートメーションとアジャイルデリバリーによってワークロードの移行を合理化します。これは、AWS 移行戦略 の第 3 段階です。

移行ファクトリー

自動化された俊敏性のあるアプローチにより、ワークロードの移行を合理化する部門横断的なチーム。移行ファクトリーチームには、通常、運用、ビジネスアナリストおよび所有者、移行エンジニア、デベロッパー、およびスプリントで作業する DevOps プロフェッショナルが含まれます。エンタープライズアプリケーションポートフォリオの 20～50% は、ファクトリーのアプローチによって最適化できる反復パターンで構成されています。詳細については、このコンテンツセットの移行ファクトリーに関する解説とCloud Migration Factory ガイドを参照してください。

移行メタデータ

移行を完了するために必要なアプリケーションおよびサーバーに関する情報。移行パターンごとに、異なる一連の移行メタデータが必要です。移行メタデータの例としては、ターゲットサブネット、セキュリティグループ、 AWS アカウントなどがあります。

移行パターン

移行戦略、移行先、および使用する移行アプリケーションまたはサービスを詳述する、反復可能な移行タスク。例: AWS Application Migration Service を使用して Amazon EC2 への移行をリホストします。

Migration Portfolio Assessment (MPA)

オンラインツール。これによって、 AWS クラウドに移行するビジネスケースの検証に必要な情報を得られます。MPA は、詳細なポートフォリオ評価 (サーバーの適切なサイジング、価格設定、TCO 比較、移行コスト分析) および移行プラン (アプリケーションデータの分析とデータ収集、アプリケーションのグループ化、移行の優先順位付け、およびウェーブプランニング) を提供します。MPA ツール (ログインが必要) は、すべての AWS コンサルタントと APN パートナーコンサルタントが無料で利用できます。

移行準備状況評価 (MRA)

AWS CAF を使用して、組織のクラウド準備状況に関するインサイトを取得し、長所と短所を特定し、特定されたギャップを埋めるためのアクションプランを構築するプロセス。詳細については、移行準備状況ガイドを参照してください。MRA は、AWS 移行戦略の第一段階です。

移行戦略

ワークロードを AWS クラウドに移行するために使用するアプローチ。詳細については、この用語集の 7 Rs エントリと、「組織を動員して大規模な移行を加速する」を参照してください。

ML

「機械学習」を参照してください。

モダナイゼーション

古い (レガシーまたはモノリシック) アプリケーションとそのインフラストラクチャをクラウド内の俊敏で弾力性のある高可用性システムに変換して、コストを削減し、効率を高め、イノベーションを活用します。詳細については、「AWS クラウドでのアプリケーションのモダナイズ戦略」を参照してください。

モダナイゼーション準備状況評価

組織のアプリケーションのモダナイゼーションの準備状況を判断し、利点、リスク、依存関係を特定し、組織がこれらのアプリケーションの将来の状態をどの程度適切にサポートできるかを決定するのに役立つ評価。評価の結果として、ターゲットアーキテクチャのブループリント、モダナイゼーションプロセスの開発段階とマイルストーンを詳述したロードマップ、特定されたギャップに対処するためのアクションプランが得られます。詳細については、「AWS クラウドでのアプリケーションのモダナイゼーションの準備状況を評価する」を参照してください。

モノリシックアプリケーション (モノリス)

緊密に結合されたプロセスを持つ単一のサービスとして実行されるアプリケーション。モノリシックアプリケーションにはいくつかの欠点があります。1 つのアプリケーション機能エクスペリエンスの需要が急増する場合は、アーキテクチャ全体をスケーリングする必要があります。モノリシックアプリケーションの特徴を追加または改善することは、コードベースが大きくなると複雑になります。これらの問題に対処するには、マイクロサービスアーキテクチャを使用できます。詳細については、「モノリスをマイクロサービスに分解する」を参照してください。

MPA

「Migration Portfolio Assessment」を参照してください。

MQTT

「Message Queuing Telemetry Transport」を参照してください。

多クラス分類

複数のクラスの予測を生成するプロセス (2 つ以上の結果の 1 つを予測します)。例えば、機械学習モデルが、「この製品は書籍、自動車、電話のいずれですか?」 または、「このお客様にとって最も関心のある商品のカテゴリはどれですか?」と聞くかもしれません。

ミュータブルなインフラストラクチャ

本番ワークロードに使用する既存のインフラストラクチャを更新および変更するためのモデル。Well-Architected AWS フレームワークでは、一貫性、信頼性、予測可能性を向上させるために、イミュータブルインフラストラクチャの使用をベストプラクティスとして推奨しています。

O

OAC

「オリジンアクセス制御」を参照してください。

OAI

「オリジンアクセスアイデンティティ」を参照してください。

OCM

「組織変更管理」を参照してください。

オフライン移行

移行プロセス中にソースワークロードを停止させる移行方法。この方法はダウンタイムが長くなるため、通常は重要ではない小規模なワークロードに使用されます。

OI

「オペレーション統合」を参照してください。

Ola

「オペレーショナルレベルアグリーメント」を参照してください。

オンライン移行

ソースワークロードをオフラインにせずにターゲットシステムにコピーする移行方法。ワークロードに接続されているアプリケーションは、移行中も動作し続けることができます。この方法はダウンタイムがゼロから最小限で済むため、通常は重要な本番稼働環境のワークロードに使用されます。

OPC-UA

「Open Process Communications - Unified Architecture」を参照してください。

Open Process Communications - Unified Architecture (OPC-UA)

産業オートメーション用のマシンツーマシン (M2M) 通信プロトコル。OPC-UA により、相互運用の際に、データ暗号化、認証、認可の各スキームを標準化できます。

オペレーショナルレベルアグリーメント (OLA)

サービスレベルアグリーメント (SLA) をサポートするために、どの機能的 IT グループが互いに提供することを約束するかを明確にする契約。

運用準備状況レビュー (ORR)

質問と関連するベストプラクティスのチェックリスト。インシデントや起こり得る障害を理解、評価、防止したり、その範囲を縮小したりする際に役立ちます。詳細については、 AWS Well-Architected フレームワークの「Operational Readiness Reviews (ORR)」を参照してください。

運用テクノロジー (OT)

産業オペレーション、機器、インフラストラクチャを制御するために物理環境と連携させるハードウェアおよびソフトウェアシステム。製造分野では、Industry 4.0 への変革を進める上で、OT と情報技術 (IT) システムの統合に焦点が当てられています。

オペレーション統合 (OI)

クラウドでオペレーションをモダナイズするプロセスには、準備計画、オートメーション、統合が含まれます。詳細については、オペレーション統合ガイドを参照してください。

組織の証跡

組織 AWS アカウント 内のすべての のすべてのイベント AWS CloudTrail を ログに記録する によって作成された証跡 AWS Organizations。証跡は、組織に含まれている各 AWS アカウント に作成され、各アカウントのアクティビティを追跡します。詳細については、CloudTrail ドキュメントの「組織の証跡の作成」を参照してください。

組織変更管理 (OCM)

人材、文化、リーダーシップの観点から、主要な破壊的なビジネス変革を管理するためのフレームワーク。OCM は、変化の導入を加速し、移行問題に対処し、文化や組織の変化を推進することで、組織が新しいシステムと戦略の準備と移行するのを支援します。 AWS 移行戦略では、クラウド導入プロジェクトに必要な変化のスピードにより、このフレームワークは人材アクセラレーションと呼ばれます。詳細については、OCM ガイドを参照してください。

オリジンアクセス制御 (OAC)

Amazon Simple Storage Service (Amazon S3) コンテンツを保護するための、CloudFront のアクセス制限の強化オプション。OAC は AWS リージョン、すべての S3 バケット、 AWS KMS (SSE-KMS) によるサーバー側の暗号化、S3 バケットへの動的 PUT および DELETEリクエストをサポートします。

オリジンアクセスアイデンティティ (OAI)

CloudFront の、Amazon S3 コンテンツを保護するためのアクセス制限オプション。OAI を使用すると、CloudFront が、Amazon S3 に認証可能なプリンシパルを作成します。認証されたプリンシパルは、S3 バケット内のコンテンツに、特定の CloudFront ディストリビューションを介してのみアクセスできます。OAC も併せて参照してください。OAC では、より詳細な、強化されたアクセス制御が可能です。

ORR

「運用準備状況レビュー」を参照してください。

OT

「運用テクノロジー」を参照してください。

アウトバウンド (送信) VPC

AWS マルチアカウントアーキテクチャでは、アプリケーション内から開始されたネットワーク接続を処理する VPC。AWS Security Reference Architecture では、アプリケーションとより広範なインターネット間の双方向のインターフェイスを保護するために、インバウンド、アウトバウンド、インスペクションの各 VPC を使用してネットワークアカウントを設定することを推奨しています。

P

アクセス許可の境界

ユーザーまたはロールが使用できるアクセス許可の上限を設定する、IAM プリンシパルにアタッチされる IAM 管理ポリシー。詳細については、IAM ドキュメントのアクセス許可の境界を参照してください。

個人を特定できる情報 (PII)

直接閲覧した場合、または他の関連データと組み合わせた場合に、個人の身元を合理的に推測するために使用できる情報。PII の例には、氏名、住所、連絡先情報などがあります。

PII

「個人を特定できる情報」を参照してください。

プレイブック

クラウドでのコアオペレーション機能の提供など、移行に関連する作業を取り込む、事前定義された一連のステップ。プレイブックは、スクリプト、自動ランブック、またはお客様のモダナイズされた環境を運用するために必要なプロセスや手順の要約などの形式をとることができます。

PLC

「プログラマブルロジックコントローラー」を参照してください。

PLM

「製品ライフサイクル管理」を参照してください。

ポリシー

次の操作を可能にするオブジェクト: アクセス許可を定義する (ID ベースのポリシーを参照)。アクセス条件を指定する (リソースベースのポリシーを参照)。 AWS Organizations の組織における全アカウントにアクセス許可の上限を定義する (サービスコントロールポリシーを参照)。

多言語の永続性

データアクセスパターンやその他の要件に基づいて、マイクロサービスのデータストレージテクノロジーを個別に選択します。マイクロサービスが同じデータストレージテクノロジーを使用している場合、実装上の問題が発生したり、パフォーマンスが低下する可能性があります。マイクロサービスは、要件に最も適合したデータストアを使用すると、より簡単に実装でき、パフォーマンスとスケーラビリティが向上します。

ポートフォリオ評価

移行を計画するために、アプリケーションポートフォリオの検出、分析、優先順位付けを行うプロセス。詳細については、「移行の準備状況の評価」を参照してください。

述語

`true` または `false` を返すためのクエリ条件。一般的に、`WHERE` 句に記述されます。

述語プッシュダウン

データベースクエリを最適化する手法。これによって、転送前にクエリ内のデータをフィルタリングします。この手法を取ると、リレーショナルデータベースから取得し処理する必要のあるデータの量が減少するため、クエリのパフォーマンスが向上します。

予防的コントロール

イベントの発生を防ぐように設計されたセキュリティコントロール。このコントロールは、ネットワークへの不正アクセスや好ましくない変更を防ぐ最前線の防御です。詳細については、「AWSでのセキュリティコントロールの実装」の「予防的コントロール」を参照してください。

プリンシパル

アクションを実行し AWS 、リソースにアクセスできる のエンティティ。このエンティティは通常、、IAM AWS アカウントロール、または ユーザーのルートユーザーです。詳細については、IAM ドキュメントの「ロールに関する用語と概念」にあるプリンシパルを参照してください。

プライバシーバイデザイン

開発プロセス全体を通してプライバシーが考慮されているシステムエンジニアリングのアプローチ。

プライベートホストゾーン

1 つ以上の VPC 内のドメインとそのサブドメインへの DNS クエリに対し、Amazon Route 53 がどのように応答するかに関する情報を保持するコンテナ。詳細については、Route 53 ドキュメントの「プライベートホストゾーンの使用」を参照してください。

プロアクティブコントロール

非準拠リソースのデプロイ防止を目的としたセキュリティコントロール。このコントロールにより、プロビジョニング前にリソースをスキャンします。コントロールに準拠していないリソースは、プロビジョニングされません。詳細については、 AWS Control Tower ドキュメントの「コントロールリファレンスガイド」および「 セキュリティコントロールの実装」の「プロアクティブコントロール」を参照してください。 AWS

製品ライフサイクル管理 (PLM)

製品の設計、開発、発売から、成長、成熟、衰退、廃棄に至る、製品のライフサイクル全体を通してデータとプロセスを管理すること。

本番環境

「環境」を参照してください。

プログラマブルロジックコントローラー (PLC)

製造分野で使用される、信頼性と適応性に優れたコンピュータであり、これによって、マシンをモニタリングするとともに、製造プロセスを自動化します。

プロンプトチェイニング

1 つの LLM プロンプトによる出力を次のプロンプトの入力に使用して、より良いレスポンスを生成します。この手法を使用すると、複雑なタスクをサブタスクに分割したり、事前レスポンスを繰り返し改良または拡張したりできます。これによって、モデルのレスポンスの精度と関連性が向上し、粒度の高いパーソナライズされた結果を得られます。

仮名化

データセット内の個人識別子をプレースホルダー値に置き換えるプロセス。仮名化は個人のプライバシー保護に役立ちます。仮名化されたデータは、依然として個人データとみなされます。

発行/サブスクライブ (pub/sub)

マイクロサービス間の非同期通信を可能にするパターン。これにより、スケーラビリティと応答性を向上させます。例えば、マイクロサービスベースの MES の場合、マイクロサービスは、他のマイクロサービスがサブスクライブ可能なチャネルにイベントメッセージを発行できます。このシステムでは、発行サービスの変更なしに、新規マイクロサービスを追加できます。

Q

クエリプラン

手順などの一連のステップであり、SQL リレーショナルデータベースシステムのデータにアクセスするために使用されます。

クエリプランのリグレッション

データベースサービスのオプティマイザーが、データベース環境に特定の変更が加えられる前に選択されたプランよりも最適性の低いプランを選択すること。これは、統計、制限事項、環境設定、クエリパラメータのバインディングの変更、およびデータベースエンジンの更新などが原因である可能性があります。

R

RACI マトリックス

「実行責任者、説明責任者、協業先、報告先 (RACI)」を参照してください。

RAG

「検索拡張生成」を参照してください。

ランサムウェア

決済が完了するまでコンピュータシステムまたはデータへのアクセスをブロックするように設計された、悪意のあるソフトウェア。

RASCI マトリックス

「実行責任者、説明責任者、協業先、報告先 (RACI)」を参照してください。

RCAC

「行と列のアクセス制御」を参照してください。

リードレプリカ

読み取り専用に使用されるデータベースのコピー。クエリをリードレプリカにルーティングして、プライマリデータベースへの負荷を軽減できます。

リアーキテクト

「7 Rs」を参照してください。

目標復旧時点 (RPO)

最後のデータリカバリポイントからの最大許容時間です。これにより、最後の回復時点からサービスが中断されるまでの間に許容できるデータ損失の程度が決まります。

目標復旧時間 (RTO)

サービスの中断から復旧までの最大許容遅延時間。

リファクタリング

「7 Rs」を参照してください。

リージョン

地理的エリア内の AWS リソースのコレクション。各 AWS リージョン は、耐障害性、安定性、耐障害性を提供するために、他の から分離され、独立しています。詳細については、「アカウントが使用できる AWS リージョン を指定する」を参照してください。

リグレッション

数値を予測する機械学習手法。例えば、「この家はどれくらいの値段で売れるでしょうか?」という問題を解決するために、機械学習モデルは、線形回帰モデルを使用して、この家に関する既知の事実 (平方フィートなど) に基づいて家の販売価格を予測できます。

リホスト

「7 Rs」を参照してください。

リリース

デプロイプロセスで、変更を本番環境に昇格させること。

再配置

「7 Rs」を参照してください。

リプラットフォーム

「7 Rs」を参照してください。

再購入

「7 Rs」を参照してください。

回復性

中断に抵抗または中断から回復するアプリケーションの機能。 AWS クラウドでの回復力を計画する際には、一般的に、高可用性とディザスタリカバリが考慮されます。詳細については、「AWS クラウド の耐障害性」を参照してください。

リソースベースのポリシー

Amazon S3 バケット、エンドポイント、暗号化キーなどのリソースにアタッチされたポリシー。このタイプのポリシーは、アクセスが許可されているプリンシパル、サポートされているアクション、その他の満たすべき条件を指定します。

実行責任者、説明責任者、協業先、報告先 (RACI) に基づくマトリックス

移行活動とクラウド運用に関わるすべての関係者の役割と責任を定義したマトリックス。マトリックスの名前は、マトリックスで定義されている責任の種類、すなわち責任 (R) 、説明責任 (A) 、協議 (C) 、情報提供 (I) に由来します。サポート (S) タイプはオプションです。サポートが含まれる場合は RASCI マトリックスと呼ばれ、含まれない場合は RACI マトリックスと呼ばれます。

レスポンシブコントロール

有害事象やセキュリティベースラインからの逸脱について、修復を促すように設計されたセキュリティコントロール。詳細については、「 AWSでのセキュリティコントロールの実装」の「レスポンシブコントロール」を参照してください。

保持

「7 Rs」を参照してください。

廃止

「7 Rs」を参照してください。

検索拡張生成 (RAG)

生成 AI の技術。これにより、LLM では、レスポンスの生成前に、トレーニングデータソースの外部にある信頼できるデータソースが参照されます。例えば、RAG モデルによって、組織のナレッジベースまたはカスタムデータのセマンティック検索を実行できる場合があります。細については、「RAG (検索拡張生成) とは何ですか?」を参照してください。

ローテーション

定期的にシークレット情報を更新して、攻撃者が認証情報にアクセスするのをより困難にするプロセス。

行と列のアクセス制御 (RCAC)

アクセスルールが定義された、基本的で柔軟な SQL 表現の使用。RCAC は行権限と列マスクで構成されています。

RPO

「目標復旧時点」を参照してください。

RTO

「目標復旧時間」を参照してください。

ランブック

特定のタスクを実行するために必要な手動または自動化された一連の手順。これらは通常、エラー率の高い反復操作や手順を合理化するために構築されています。

S

SAML 2.0

多くの ID プロバイダー (IdP) が使用しているオープンスタンダード。この機能を使用すると、フェデレーティッドシングルサインオン (SSO) が有効になるため、ユーザーは組織内のすべてのユーザーを IAM で作成しなくても、 AWS マネジメントコンソール にログインしたり AWS 、API オペレーションを呼び出すことができます。SAML 2.0 ベースのフェデレーションの詳細については、IAM ドキュメントの「SAML 2.0 ベースのフェデレーションについて」を参照してください。

SCADA

「監視制御とデータ取得」を参照してください。

SCP

「サービスコントロールポリシー」を参照してください。

シークレット

暗号化された形式で保存する AWS Secrets Managerパスワードやユーザー認証情報などの機密情報または制限付き情報。シークレット値とそのメタデータで構成されます。シークレット値には、バイナリ、1 つの文字列、複数の文字列を指定できます。詳細については、Secrets Manager ドキュメントの「Secrets Manager シークレットの概要」を参照してください。

セキュリティバイデザイン

開発プロセス全体を通してセキュリティが考慮されているシステムエンジニアリングのアプローチ。

セキュリティコントロール

脅威アクターによるセキュリティ脆弱性の悪用を防止、検出、軽減するための、技術上または管理上のガードレール。セキュリティコントロールには、主に 4 つの種類があります。4 つとは、予防、検出、レスポンシブ、プロアクティブです。

セキュリティ強化

アタックサーフェスを狭めて攻撃への耐性を高めるプロセス。このプロセスには、不要になったリソースの削除、最小特権を付与するセキュリティのベストプラクティスの実装、設定ファイル内の不要な機能の無効化、といったアクションが含まれています。

Security Information and Event Management (SIEM) システム

セキュリティ情報管理 (SIM) とセキュリティイベント管理 (SEM) のシステムを組み合わせたツールとサービス。SIEM システムは、サーバー、ネットワーク、デバイス、その他ソースからデータを収集、モニタリング、分析して、脅威やセキュリティ違反を検出し、アラートを発信します。

セキュリティレスポンスの自動化

セキュリティイベントへの自動レスポンスまたは自動修復を目的として、事前定義およびプログラムされたアクション。これらの自動化は、セキュリティのベストプラクティスを実装するのに役立つ検出的または応答的な AWS セキュリティコントロールとして機能します。自動レスポンスアクションの例には、VPC セキュリティグループの変更、Amazon EC2 インスタンスへのパッチ適用、認証情報の更新などがあります。

サーバー側の暗号化

送信先で、それ AWS のサービス を受け取る によるデータの暗号化。

サービスコントロールポリシー (SCP)

AWS Organizationsの組織内の、すべてのアカウントのアクセス許可を一元的に管理するポリシー。SCP は、管理者がユーザーまたはロールに委任するアクションに、ガードレールを定義したり、アクションの制限を設定したりします。SCP は、許可リストまたは拒否リストとして、許可または禁止するサービスやアクションを指定する際に使用できます。詳細については、 AWS Organizations ドキュメントの「サービスコントロールポリシー」を参照してください。

サービスエンドポイント

のエントリポイントの URL AWS のサービス。ターゲットサービスにプログラムで接続するには、エンドポイントを使用します。詳細については、「AWS 全般のリファレンス」の「AWS のサービス エンドポイント」を参照してください。

サービスレベルアグリーメント (SLA)

サービスのアップタイムやパフォーマンスなど、IT チームがお客様に提供すると約束したものを明示した合意書。

サービスレベルインジケータ (SLI)

エラー率、可用性、スループットといった、サービスパフォーマンス面の指標。

サービスレベル目標 (SLO)

サービスレベルインジケータによって測定され、サービスの状態を表すターゲットメトリクス。

責任共有モデル

クラウドのセキュリティとコンプライアンス AWS について と共有する責任を説明するモデル。AWS はクラウドのセキュリティを担当しますが、 はクラウドのセキュリティを担当します。詳細については、「責任共有モデル」を参照してください。

SIEM

「Security Information and Event Management システム」を参照してください。

単一障害点 (SPOF)

特定のアプリケーションを構成する単一の重要なコンポーネントで発生し、システム稼働に支障をきたす可能性のある障害。

SLA

「サービスレベルアグリーメント」を参照してください。

SLI

「サービスレベルインジケータ」を参照してください。

SLO

「サービスレベルの目標」を参照してください。

スプリットアンドシードモデル

モダナイゼーションプロジェクトのスケーリングと加速のためのパターン。新機能と製品リリースが定義されると、コアチームは解放されて新しい製品チームを作成します。これにより、お客様の組織の能力とサービスの拡張、デベロッパーの生産性の向上、迅速なイノベーションのサポートに役立ちます。詳細については、「AWS クラウドでのアプリケーションをモダナイズするための段階的アプローチ」を参照してください。

SPOF

「単一障害点」を参照してください。

スタースキーマ

データベースの編成構造を意味し、1 つの大きいファクトテーブルにトランザクションデータまたは測定データが保存され、1 つ以上の小さいディメンションテーブルにデータ属性が保存されます。この構造は、データウェアハウスやビジネスインテリジェンスを用途とするように設計されています。

strangler fig パターン

レガシーシステムが廃止されるまで、システム機能を段階的に書き換えて置き換えることにより、モノリシックシステムをモダナイズするアプローチ。このパターンは、宿主の樹木から根を成長させ、最終的にその宿主を包み込み、宿主に取って代わるイチジクのつるを例えています。そのパターンは、モノリシックシステムを書き換えるときのリスクを管理する方法として Martin Fowler により提唱されました。このパターンの適用方法の例については、「コンテナと Amazon API Gateway を使用して、従来の Microsoft ASP.NET (ASMX) ウェブサービスを段階的にモダナイズ」を参照してください。

サブネット

VPC 内の IP アドレスの範囲。サブネットは、1 つのアベイラビリティーゾーンに存在する必要があります。

監視制御とデータ取得 (SCADA)

製造分野において、ハードウェアとソフトウェアを使用して物理アセットと本番運用をモニタリングするシステム。

対称暗号化

データの暗号化と復号に同じキーを使用する暗号化のアルゴリズム。

合成テスト

ユーザーとのやり取りをシミュレートして、起こり得る問題を検出したり、パフォーマンスをモニタリングしたりすることで、システムをテストします。Amazon CloudWatch Synthetics を使用すると、こうしたテストを作成できます。

システムプロンプト

コンテキスト、指示、ガイドラインなどを提示して、LLM に動作を指示する手法。システムプロンプトは、コンテキストを設定して、ユーザーとやり取りするルールを確立するのに有用です。

T

タグ

AWS リソースを整理するためのメタデータとして機能するキーと値のペア。タグは、 リソースの管理、識別、整理、検索、フィルタリングに役立ちます。詳細については、「AWS リソースのタグ付け」を参照してください。

ターゲット変数

監督された機械学習でお客様が予測しようとしている値。これは、結果変数のことも指します。例えば、製造設定では、ターゲット変数が製品の欠陥である可能性があります。

タスクリスト

ランブックの進行状況を追跡するために使用されるツール。タスクリストには、ランブックの概要と完了する必要のある一般的なタスクのリストが含まれています。各一般的なタスクには、推定所要時間、所有者、進捗状況が含まれています。

テスト環境

「環境」を参照してください。

トレーニング

お客様の機械学習モデルに学習するデータを提供すること。トレーニングデータには正しい答えが含まれている必要があります。学習アルゴリズムは入力データ属性をターゲット (お客様が予測したい答え) にマッピングするトレーニングデータのパターンを検出します。これらのパターンをキャプチャする機械学習モデルを出力します。そして、お客様が機械学習モデルを使用して、ターゲットがわからない新しいデータでターゲットを予測できます。

トランジットゲートウェイ

VPC とオンプレミスネットワークを相互接続するために使用できる、ネットワークの中継ハブ。詳細については、 AWS Transit Gateway ドキュメントの「トランジットゲートウェイとは」を参照してください。

トランクベースのワークフロー

デベロッパーが機能ブランチで機能をローカルにビルドしてテストし、その変更をメインブランチにマージするアプローチ。メインブランチはその後、開発環境、本番前環境、本番環境に合わせて順次構築されます。

信頼されたアクセス

ユーザーに代わって AWS Organizations およびそのアカウントで組織内でタスクを実行するために指定したサービスにアクセス許可を付与します。信頼されたサービスは、サービスにリンクされたロールを必要なときに各アカウントに作成し、ユーザーに代わって管理タスクを実行します。詳細については、 ドキュメントの「Using AWS Organizations with other AWS services AWS Organizations 」を参照してください。

チューニング

機械学習モデルの精度を向上させるために、お客様のトレーニングプロセスの側面を変更する。例えば、お客様が機械学習モデルをトレーニングするには、ラベル付けセットを生成し、ラベルを追加します。これらのステップを、異なる設定で複数回繰り返して、モデルを最適化します。

ツーピザチーム

2 枚のピザを分け合えることができるくらい小さな DevOps チーム。ツーピザチームの規模では、ソフトウェア開発におけるコラボレーションに最適な機会が確保されます。

U

不確実性

予測機械学習モデルの信頼性を損なう可能性がある、不正確、不完全、または未知の情報を指す概念。不確実性には、次の 2 つのタイプがあります。認識論的不確実性は、限られた、不完全なデータによって引き起こされ、弁論的不確実性 は、データに固有のノイズとランダム性によって引き起こされます。詳細については、深層学習システムにおける不確実性の定量化ガイドを参照してください。

未分化なタスク

ヘビーリフティングとも呼ばれ、アプリケーションの作成と運用には必要だが、エンドユーザーに直接的な価値をもたらさなかったり、競争上の優位性をもたらしたりしない作業です。未分化なタスクの例としては、調達、メンテナンス、キャパシティプランニングなどがあります。

上位環境

「環境」を参照してください。

V

バキューミング

ストレージを再利用してパフォーマンスを向上させるために、増分更新後にクリーンアップを行うデータベースのメンテナンス操作。

バージョンコントロール

リポジトリ内のソースコードへの変更など、変更を追跡するプロセスとツール。

VPC ピアリング

プライベート IP アドレスを使用してトラフィックをルーティングできる、2 つの VPC 間の接続。詳細については、Amazon VPC ドキュメントの「VPC ピア機能とは」を参照してください。

脆弱性

システムのセキュリティを脅かすソフトウェアまたはハードウェアの欠陥。

W

ウォームキャッシュ

頻繁にアクセスされる最新の関連データを含むバッファキャッシュ。データベースインスタンスはバッファキャッシュから、メインメモリまたはディスクからよりも短い時間で読み取りを行うことができます。

ウォームデータ

アクセス頻度の低いデータ。この種類のデータをクエリする場合、通常は適度に遅いクエリでも問題ありません。

ウィンドウ関数

現在のレコードに何らかの形で関連している行のグループに計算を実行する SQL 関数。ウィンドウ関数は、移動平均を計算したり、現在の行の相対位置に基づいて他の行の値にアクセスするといったタスクの処理に役立ちます。

ワークロード

ビジネス価値をもたらすリソースとコード (顧客向けアプリケーションやバックエンドプロセスなど) の総称。

ワークストリーム

特定のタスクセットを担当する移行プロジェクト内の機能グループ。各ワークストリームは独立していますが、プロジェクト内の他のワークストリームをサポートしています。たとえば、ポートフォリオワークストリームは、アプリケーションの優先順位付け、ウェーブ計画、および移行メタデータの収集を担当します。ポートフォリオワークストリームは、これらの設備を移行ワークストリームで実現し、サーバーとアプリケーションを移行します。

WORM

「Write-Once-Read-Many」を参照してください。

WQF

「AWS ワークロード資格フレームワーク」を参照してください

Write-Once-Read-Many (WORM)

データを 1 回のみ書き込むことで、データの削除や変更を防ぐストレージモデル。承認済みユーザーは、必要な回数だけデータを読み取ることができますが、変更することはできません。このデータストレージインフラストラクチャは、イミュータブルと見なされます。

Z

ゼロデイエクスプロイト

ゼロデイ脆弱性を悪用した攻撃 (一般的にマルウェアによる)。

ゼロデイ脆弱性

実稼働システムにおける未解決の欠陥または脆弱性。脅威アクターは、このような脆弱性を利用してシステムを攻撃する可能性があります。開発者は、よく攻撃の結果で脆弱性に気付きます。

ゼロショットプロンプト

LLM にタスク実行の手順は提示するが、実行のガイドとして役立つ例 (ショット) は提示しない方法。LLM は、事前トレーニング済みの知識を使用してタスクを処理する必要があります。ゼロショットプロンプトの有効性は、タスクの複雑さとプロンプトの品質によって異なります。「数ショットプロンプト」も参照してください。

ゾンビアプリケーション

平均 CPU およびメモリ使用率が 5% 未満のアプリケーション。移行プロジェクトでは、これらのアプリケーションを廃止するのが一般的です。

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。