



での Apache Iceberg の使用 AWS

AWS 規範ガイド



AWS 規範ガイド: での Apache Iceberg の使用 AWS

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon の商標およびトレードドレスは、Amazon 以外の製品およびサービスに使用することはできません。また、お客様に誤解を与える可能性がある形式で、または Amazon の信用を損なう形式で使用することもできません。Amazon が所有していないその他のすべての商標は、Amazon との提携、関連、支援関係の有無にかかわらず、それら該当する所有者の資産です。

Table of Contents

序章	1
最新のデータレイク	2
最新のデータレイクにおける高度なユースケース	2
Apache Iceberg の概要	3
AWS Apache Iceberg のサポート	4
Athena SQL での Iceberg テーブルの開始方法	6
パーティション分割されていないテーブルの作成	6
パーティションテーブルの作成	7
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	7
データの挿入、更新、削除	8
Iceberg テーブルのクエリ	9
Iceberg テーブルの構造	9
Amazon EMR での Iceberg の使用	12
バージョンと機能の互換性	12
Iceberg を使用した Amazon EMR クラスターの作成	12
Amazon EMR での Iceberg アプリケーションの開発	13
Amazon EMR Studio ノートブックの使用	13
Amazon EMR での Iceberg ジョブの実行	14
Amazon EMR のベストプラクティス	19
での Iceberg の使用 AWS Glue	21
ネイティブ Iceberg 統合の使用	21
カスタム Iceberg バージョンの使用	22
カスタムコネクタの使用	22
独自の JAR ファイルを持ち込む	24
での Iceberg の Spark 設定 AWS Glue	24
AWS Glue ジョブのベストプラクティス	26
Spark を使用した Iceberg テーブルの操作	27
Iceberg テーブルの作成と書き込み	27
Spark SQL の使用	27
DataFrames API の使用	28
Iceberg テーブルのデータの更新	29
Iceberg テーブルのデータの更新	30
Iceberg テーブル内のデータの削除	30
データの読み込み	31

タイムトラベルの使用	31
増分クエリの使用	32
メタデータへのアクセス	33
Athena SQL を使用した Iceberg テーブルの操作	35
バージョンと機能の互換性	35
Iceberg テーブル仕様のサポート	35
Iceberg 機能のサポート	35
Iceberg テーブルの操作	36
既存のテーブルを Iceberg に移行する	37
インプレース移行	37
フルデータ移行	42
移行戦略の選択	43
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	45
一般的なベストプラクティス	45
読み取りパフォーマンスの最適化	46
パーティション	46
ファイルサイズの調整	48
列統計の最適化	50
適切な更新戦略を選択する	51
ZSTD 圧縮を使用する	51
ソート順序を設定する	52
書き込みパフォーマンスの最適化	54
テーブル分散モードを設定する	54
適切な更新戦略を選択する	55
適切なファイル形式を選択する	55
ストレージの最適化	56
S3 Intelligent-Tiering を有効にする	57
履歴スナップショットのアーカイブまたは削除	57
孤立したファイルを削除する	60
圧縮を使用したテーブルのメンテナンス	61
Iceberg 圧縮	61
圧縮動作の調整	62
Amazon EMR または で Spark を使用して圧縮を実行する AWS Glue	64
Amazon Athena による圧縮の実行	65
圧縮を実行するための推奨事項	65
Amazon S3 での Iceberg ワークロードの使用	66

ホットパーティショニングの防止 (HTTP 503 エラー)	66
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	67
間でデータをレプリケートする AWS リージョン	67
Iceberg ワークロードのモニタリング	70
テーブルレベルのモニタリング	70
データベースレベルのモニタリング	72
予防メンテナンス	73
ガバナンスとアクセスコントロール	75
リファレンスアーキテクチャ	76
夜間のバッチ取り込み	76
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	77
リソース	78
寄稿者	79
ドキュメント履歴	81
用語集	82
#	82
A	83
B	86
C	88
D	91
E	95
F	97
G	98
H	99
I	100
L	102
M	103
O	107
P	110
Q	112
R	113
S	115
T	119
U	120
V	121
W	121

Z	122
.....	CXXIV

AWS での Apache Iceberg の使用

Amazon Web Services ([寄稿者](#))

2024 年 4 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 AWS Glue、Amazon Athena、Amazon Redshift などの AWS 分析サービスには Apache Iceberg のネイティブサポートが含まれているため、で 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。

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

このガイドは AWS、 で Apache Iceberg を使用しているすべてのユーザーに適用されます。Apache Iceberg をすぐに使い始める初心者ユーザーから、 で既存の Apache Iceberg ワークロードを最適化および調整したい上級ユーザーまで、対象です AWS。

このガイドの内容

- [最新のデータレイク](#)
- [Athena SQL での Iceberg テーブルの開始方法](#)
- [Amazon EMR での Iceberg の使用](#)
- [での Iceberg の使用 AWS Glue](#)
- [Spark を使用した Iceberg テーブルの操作](#)
- [Athena SQL を使用した Iceberg テーブルの操作](#)
- [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 ワークロードのモニタリング](#)
- [ガバナンスとアクセスコントロール](#)
- [リファレンスアーキテクチャ](#)
- [リソース???](#)
- [寄稿者](#)

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、より良い意思決定を導き出すことができます。

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、一貫性、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。

- ビジネスにおけるデータ変更に伴う効率的なレコードレベルの更新と削除の実行
- テーブルが数百万のファイルと数十万のパーティションに拡大するにつれてクエリパフォーマンスを管理する
- 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の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にな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。

これらの課題を克服するために、Apache Iceberg は、Amazon [Simple Storage Service \(Amazon S3\)](#) などの費用対効果の高いシステムで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を追加で提供します。

Apache Iceberg の概要

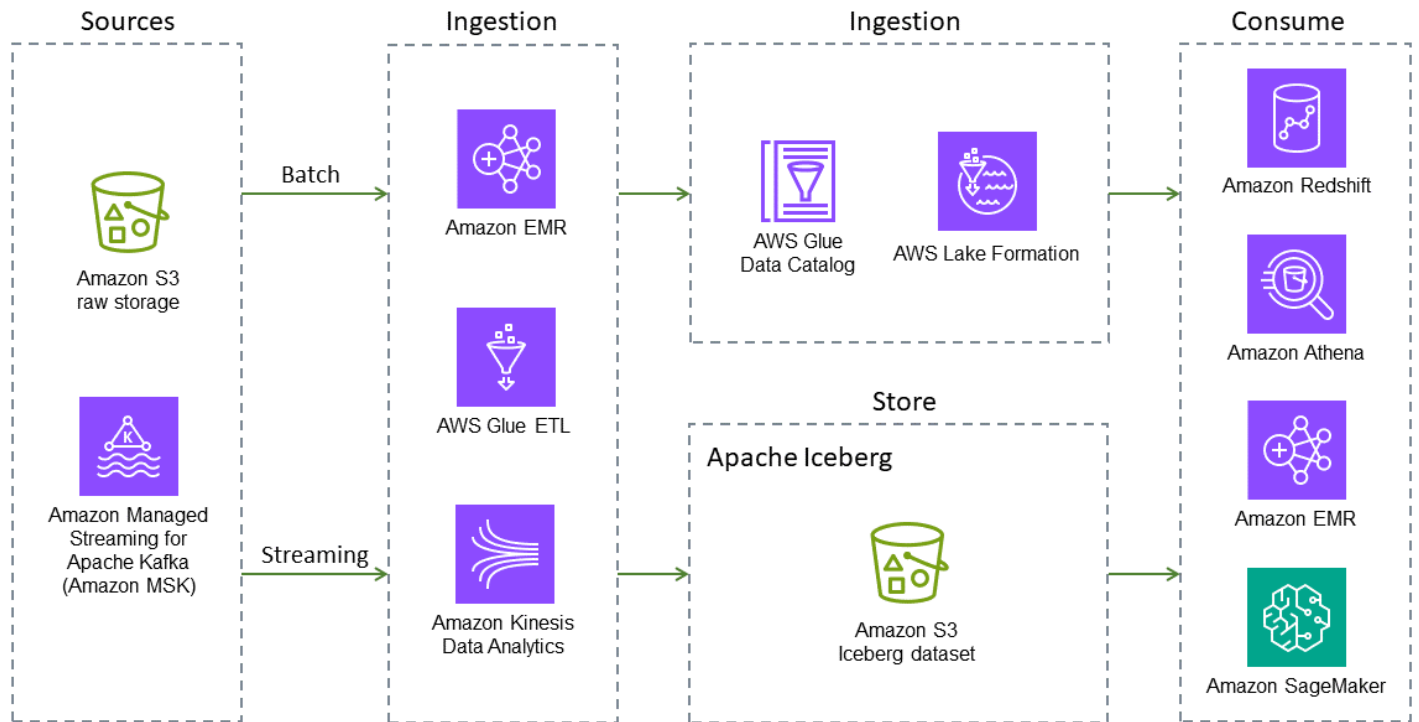
Apache Iceberg は、これまでデータベースまたはデータウェアハウスでのみ使用されていたデータレイクテーブルの機能を提供するオープンソースのテーブル形式です。スケールとパフォーマンスを考慮して設計されており、数百ギガバイトを超えるテーブルの管理に適しています。Iceberg テーブルの主な機能は次のとおりです。

- 削除、更新、マージ。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、エンジンがクエリの計画と実行を高速化するために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、名前の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 では、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がサポートされています。
- 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こと防止します。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。
- バージョンのロールバック。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。
- 直列化可能な分離。テーブルの変更はアトミックであるため、閲覧者には部分的な変更やコミットされていない変更が見られません。
- 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。これらの機能と他の Iceberg 機能の詳細については、[Apache Iceberg 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、一般的なオープンソースデータ処理フレームワーク、および [Amazon EMR](#)、[Amazon Athena](#)、[Amazon Redshift](#)、AWS サービス などの でサポートされています [AWS Glue](#)。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。



ネイティブ Iceberg 統合を以下 AWS サービス に示します。間接的 AWS サービス に、または Iceberg ライブラリをパッケージ化することで、Iceberg とやり取りできる 他にもあります。

- 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。
- 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

ト単位のデータが存在する場所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の検出、準備、移動、統合を容易にするサーバーレスデータ統合サービスです。AWS Glue 3.0 以降のバージョンでは、データレイクの Iceberg フレームワークがサポートされています。を使用して AWS Glue、Amazon S3 の Iceberg テーブルに対して読み取りおよび書き込みオペレーションを実行したり、を使用して Iceberg テーブルを操作したりできます AWS Glue Data Catalog。挿入、更新、Spark クエリ、Spark 書き込みなどの追加のオペレーションもサポートされています。
- AWS Glue Data Catalog Iceberg テーブルをサポートする Hive メタストア互換データカタログサービスを提供します。
- AWS Glue クローラーは、に登録 Iceberg テーブルするための自動化を提供します AWS Glue Data Catalog。
- Amazon は、Iceberg 形式を使用して Amazon Feature Store の機能セットのストレージ SageMaker をサポートしています。SageMaker
- 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 をサポートするさまざまな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) と AWS Glue Amazon 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 サポートを簡単に説明します。

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

Amazon Athena は、Apache 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場合を除き、追加の手順や設定なしで Iceberg を使用できます。このセクションでは、Athena でのテーブルの作成について簡単に説明します。詳細については、このガイドで後述する「[Athena SQL を使用した Apache Iceberg テーブルの操作](#)」を参照してください。

異なるエンジンを使用して、で Iceberg AWS テーブルを作成できます。これらのテーブルは全体でシームレスに動作します AWS 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](#)」の「[Iceberg 構文の詳細](#)」を参照してください。

パーティション分割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 [Athenaconsole](#) 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 手順については、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では、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、その値はその場で算出されます。

```
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で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、UPDATEMERGE INTO および DELETE FROM ステートメントを使用して Iceberg 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。

注: UPDATE、MERGE INTO、および DELETE FROM は、位置削除で merge-on-read アプローチを使用します。この copy-on-write アプローチは現在、Athena SQL ではサポートされていません。

例えば、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:

#	color	date	name	price	product	ts
1	red	222022-07-19T03:47:29	PersonNew	178	Tuna	2023-10-11 11:35:01.298000 UTC

詳細については、[「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。

Athena は、通常のクエリに加えて、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 に基づいて古いバージョンのテーブルをさかのぼるのが便利です。

例えば、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:

#	color	date	name	price	product	ts
1	cyan	2022-07-19T03:47:29	Person5	353	Keyboard	2023-01-03 10:15:52.268000 UTC
2	lime	2022-07-19T03:47:29	Person1	833	Towels	2023-01-03 10:15:52.268000 UTC
3	turquoise	2022-07-19T03:47:29	Person1	1319	Shirt	2023-01-03 10:15:52.268000 UTC
4	blue	2022-07-19T03:47:29	Person3	163	Sausages	2023-01-03 10:15:52.268000 UTC

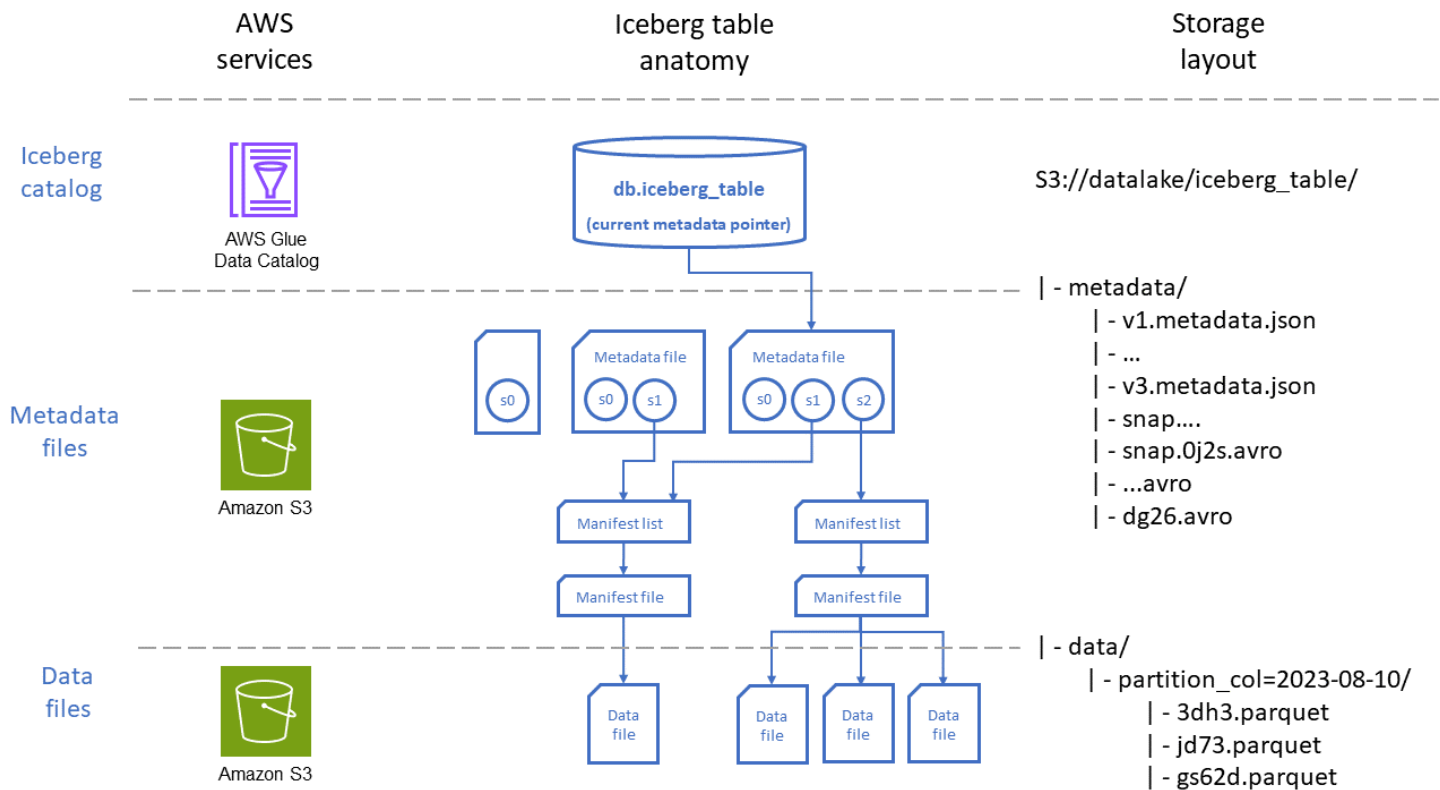
タイムトラベルクエリの構文とその他の例については、[Athena ドキュメント](#) を参照してください。

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 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 つの主要なレイヤーで構成されています。

- **Iceberg カタログ:** は Iceberg と AWS Glue Data Catalog ネイティブに統合されており、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、データの読み取りまたは書き込みのために、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ています。
- **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。
- ファイルを削除すると、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がエンコードされます。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 Apache 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、例に Apache Spark を使用します。

Amazon EMR は、Amazon EC2 の Amazon EMR、Amazon EKS の Amazon EMR、Amazon EMR Serverless、の 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、[Amazon EMR で Iceberg を使用する際の考慮事項と制限事項](#) を確認して、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 バージョンを利用するには、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-6.9.0 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 6.6.0 以降、Amazon EMR Serverless または Amazon EMR on Amazon EKS を 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。これは、Amazon EMR クラスターで実行されるフルマネージド Jupyter Notebook 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 Amazon EC2 クラスターの Amazon EMR または Amazon EKS マネージドエンドポイントの Amazon EMR に接続できます。Amazon [Amazon EC2 で Amazon EMR 用の EMR Studio](#) を、Amazon [EKS で Amazon EMR](#) をセットアップする手順については、AWS 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 Installed でクラスターを使用する」の説明に従って、[Iceberg を有効にして 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>.catalog-impl":
"org.apache.iceberg.aws.glue.GlueCatalog",
    "spark.sql.catalog.<catalog_name>.io-impl":
"org.apache.iceberg.aws.s3.S3FileIO",
```

```
"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。カタログの名前に置き換え、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で Iceberg テーブルを参照する必要があります。

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

- `<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、`<catalog_name>.catalog-impl` をに設定します `org.apache.iceberg.aws.glue.GlueCatalog`。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。
 - Amazon S3 マルチパートアップロードを利用して高並列処理を実現する `<catalog_name>.io-impl`には、`org.apache.iceberg.aws.s3.S3FileIO`として を使用します。 Amazon S3
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 EC2 クラスター上の Amazon EMR 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。

- Amazon 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 Amazon EKS については、「[で Spark ジョブを実行する StartJobRun](#)」を参照してください。
- Amazon 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 EC2 での Amazon EMR

Iceberg Spark ジョブを送信するには、次の手順に従います。

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>.catalog-
impl=org.apache.iceberg.aws.glue.GlueCatalog",
    "--conf",
    "spark.sql.catalog.<catalog_name>.warehouse=s3://YOUR-BUCKET-NAME/YOUR-
FOLDER-NAME/",
    "--conf",
    "spark.sql.catalog.<catalog_name>.io-
impl=org.apache.iceberg.aws.s3.S3FileIO",
    "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 Amazon 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>",
  "jobDriver": {
    "sparkSubmit": {
      "entryPoint": "s3://YOUR-BUCKET-NAME/code/iceberg-job.py",
      "entryPointArguments": [],
      "sparkSubmitParameters": "--jars /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>.catalog-impl":
"org.apache.iceberg.aws.glue.GlueCatalog",
        "spark.sql.catalog.<catalog_name>.warehouse": "s3://YOUR-BUCKET-NAME/YOUR-FOLDER-NAME/",
        "spark.sql.catalog.<catalog_name>.io-impl":
"org.apache.iceberg.aws.s3.S3FileIO",
        "spark.jars": "/usr/share/aws/iceberg/lib/iceberg-spark3-runtime.jar",
```

```
"spark.hadoop.hive.metastore.client.factory.class":"com.amazonaws.glue.catalog.metastore.AWSGlueMetastoreClientFactory",
  },
  },
  "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 Amazon EMR Serverless に送信するには :

1. [「Amazon 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 EKS での Amazon EMR

を使用して Iceberg Spark ジョブを Amazon EKS 上の Amazon EMR 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>.catalog-impl":
"org.apache.iceberg.aws.glue.GlueCatalog",
        "spark.sql.catalog.<catalog_name>.warehouse": "s3://YOUR-BUCKET-NAME/
YOUR-FOLDER-NAME/",
        "spark.sql.catalog.<catalog_name>.io-impl":
"org.apache.iceberg.aws.s3.S3FileIO",
        "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示します。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。

- 最新バージョンの Amazon EMR 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に対応するために、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 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 CloudWatch 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) を特定し、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に、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、パーティションの配置が保証されます。
- マネージドスケールリングを有効にする – ほぼすべてのユースケースで、マネージドスケール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、自動スケール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。マネージドスケールリングが有効になっている場合は、スポットインスタンスを使用してコストを削減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 – ブロードキャスト (マップサイド) 結合は、最小ノードのメモリに収まるのに十分な大きさ (MBs) のテーブルの 1 つで、等号 (=) 結合を実行している限り、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。GC 時間は、タスクランタイムの合計の 1% 以下であることが理想的です。

での Apache 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の検出、準備、移動、統合を容易にするサーバーレスデータ統合サービスです。の中核的な機能の 1 つは、シンプルで費用対効果の高い方法で抽出、変換、ロード (ETL) オペレーションを実行する機能 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 Spark](#) または 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。このスクリプトは、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 および 4.0 は、AWS Glue for Spark で 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 ジョブで 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までスクロールし、キーを `--datalake-formats` に、その値を `iceberg` に設定します。

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"
}
```

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 バージョンの制御を保持し、自分のペースでアップグレードしたい場合があります。例えば、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。特定の Iceberg バージョンを使用するには AWS Glue、カスタムコネクタまたは独自の JAR ファイルを使用できます。

カスタムコネクタの使用

AWS Glue はコネクタをサポートしています。コネクタは、のデータストアへのアクセスを支援するオプションのコードパッケージです AWS Glue Studio。で [コネクタをサブスクライブ](#) することも AWS Marketplace、カスタムコネクタを作成することもできます。

Note

AWS Marketplace は、 [用の Apache Iceberg コネクタを提供します AWS Glue](#)。ただし、Iceberg バージョンに対する制御を保持するために、代わりにカスタムコネクタを使用することをお勧めします。

例えば、Iceberg バージョン 0.13.1 のカスタムコネクタを作成するには、次の手順に従います。

1. ファイル `iceberg-spark-runtime-3.1_2.12-0.13.1.jar`、`bundle-2.17.161.jar`、および `url-connection-client-2.17.161.jar` を Amazon S3 バケットにアップロードします。これらのファイルは、それぞれの Apache Maven リポジトリからダウンロードできます。
2. [AWS Glue Studio コンソール](#) で、カスタム Spark コネクタを作成します。
 - a. ナビゲーションペインで、データ接続 を選択します。(古いナビゲーションを使用している場合は、「コネクタ」、「カスタムコネクタの作成」を選択します。)
 - b. コネクタボックスで、カスタムコネクタの作成 を選択します。
 - c. カスタムコネクタの作成ページで、次の操作を行います。
 - Amazon S3 の JAR ファイルへのパスを指定します。
 - コネクタの名前を入力します。
 - コネクタタイプとして Spark を選択します。
 - クラス名 には、`format` 演算子を使用して Spark データソースをロードするときに使用する完全修飾データソースクラス名 (またはそのエイリアス) を指定します。
 - (オプション) コネクタの説明を入力します。

3. [コネクタを作成] をクリックします。

でコネクタを使用する場合は AWS Glue、コネクタの接続を作成する必要があります。接続には、特定のデータストアへの接続に必要なプロパティが含まれます。ETL ジョブでは、データソースおよびデータターゲットとの接続を使用します。コネクタと接続は、データストアへのアクセスを容易にするために連携して動作します。

作成したカスタム Iceberg コネクタを使用して接続を作成するには：

1. [AWS Glue Studio コンソール](#) で、カスタム Iceberg コネクタを選択します。
2. プロンプトに従って、VPC やジョブに必要なその他のネットワーク設定などの詳細を指定し、接続の作成 を選択します。

AWS Glue ETL ジョブで接続を使用できるようになりました。ジョブの作成方法に応じて、接続をジョブにアタッチする方法は異なります。

- を使用してビジュアルジョブを作成する場合は AWS Glue Studio、データソースプロパティ – コネクタタブの接続リストから接続を選択できます。
- ノートブックでジョブを開発する場合は、`%connections`マジックを使用して接続名を設定します。

```
%glue_version 3.0

%connections <name-of-the iceberg-connection>

%%configure
{
  "--conf" : "job-specific Spark configurations, to be discussed later",
  "--datalake-formats" : "iceberg"
}
```

- スクリプトエディタを使用してジョブを作成する場合は、「ジョブの詳細」タブの「高度なプロパティ」、「追加のネットワーク接続」で接続を指定します。

このセクションの手順の詳細については、AWS Glue ドキュメントの [「でのコネクタと接続 AWS Glue Studioの使用」](#) を参照してください。

独自の JAR ファイルを持ち込む

では AWS Glue、コネクタを使用せずに Iceberg を使用することもできます。このアプローチは、Iceberg バージョンに対する制御を保持し、すばやく更新する場合に役立ちます。このオプションを使用するには、必要な Iceberg JAR ファイルを任意の S3 バケットにアップロードし、AWS Glue ジョブ内のファイルを参照します。例えば、Iceberg 1.0.0 を使用している場合、必要な JAR ファイルは `iceberg-spark-runtime-3.0_2.12-1.0.0.jar`、`url-connection-client-2.15.40.jar`、および `bundle-2.15.40.jar` です。ジョブの `--user-jars-first` パラメータを `true` に設定することで、クラスパス内の追加の JAR ファイルの優先順位 `true` を付けることもできます。

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 キーとすべての `--conf` 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 3.0

%connections <name-of-the-iceberg-connection>

%%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です。カタログの名前に置き換え、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で Iceberg テーブルを参照する必要があります。 `<catalog_name>.<database_name>.<table_name>`
- `<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 `org.apache.iceberg.aws.glue.GlueCatalog` にするには AWS Glue Data Catalog、`<catalog_name>.catalog-impl` を `org.apache.iceberg.aws.glue.GlueCatalog` に設定します。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。

- Amazon S3 マルチパートアップロードを利用して高並列処理を実現する<catalog_name>.io-implには、org.apache.iceberg.aws.s3.S3FileIOとして を使用します。

例えば、 というカタログがある場合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.catalog-impl=org.apache.iceberg.aws.glue.GlueCatalog ",
  "--conf" : " spark.sql.catalog.glue_iceberg.io-impl=org.apache.iceberg.aws.s3.S3FileIO
}
```

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.catalog-impl",
"org.apache.iceberg.aws.glue.GlueCatalog") \
    .config("spark.sql.catalog.glue_iceberg.io-impl",
"org.apache.iceberg.aws.s3.S3FileIO") \
    .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、 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。

- 可能な限り最新バージョンの AWS Glue を使用してアップグレードする – の新バージョン 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、「」の [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 – AWS ブログ記事「 [のメモリ管理の最適化](#) 」の推奨事項に従ってください [AWS Glue](#) 。
- 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、ジョブをスケールリングできるワーカー AWS Glue の最大数を指定します。詳細については、ドキュメントの「 [の Auto Scaling AWS Glue の使用 AWS Glue](#) 」を参照してください。
- カスタムコネクタを使用するか、ライブラリの依存関係を追加する - 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、カスタムコンテナを使用するか、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、ジョブの最新の Iceberg 機能とパフォーマンスの向上のメリットを享受するのに役立ちます AWS Glue 。
- モニタリングとデバッグのために Spark UI を有効にする – [で 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 Apache 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主な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 を使用して、データ DataFrames を作成して Iceberg テーブルに追加できます。

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')
""")
```

パーティション分割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 で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、グローバルソートを実行してからデータを書き込みます。

```
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  
  ORDER BY c_birth_country  
""")
```

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 テーブルを作成して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()`。次の例を使用します。これは `.createOrReplace()` に相当する のバリエーション `.create()` 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、ローカルソートを使用してデータを取り込むことができます。

```
input_data.sortWithinPartitions("c_birth_country") \  
    .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、グローバルソートを使用してデータを取り込むことができます。

```
input_data.orderBy("c_birth_country") \  
    .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 つのトランザクションに更新することです。データを Iceberg テーブルに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 *
""")
```

- 既存の顧客レコードが同じ {TABLE_NAME} に {UPSERT_TABLE_NAME} すでに存在する場合 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)。
- 既存の顧客レコードが {TABLE_NAME} に存在し {UPSERT_TABLE_NAME} 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 テーブル内のデータの削除

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 はメタデータのみを削除を実行し、データファイルをそのまま残します。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。例えば、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 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 DataFrames。

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 テーブル内の書き込みオペレーション (挿入、更新、アップサート、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。

snapshot-id と 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 `snapshot-id` に基づいてテーブルのステータスを表示します。

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。

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、追加されたデータを増分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。、、または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。replaceoverwritedelete。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。特定のテーブル (<table_name>) のメタデータにアクセスするには、 という名前空間をクエリします <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

Working with Apache Iceberg tables by using Amazon Athena SQL

Amazon Athena は Apache Iceberg の組み込みサポートを提供しており、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、Athena を使用して Iceberg テーブルを操作するための、サポートされている機能の詳細と大まかなガイダンスを提供します。

バージョンと機能の互換性

Note

以下のセクションでは、[Athena エンジンバージョン 3](#) を使用していることを前提としています。

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 場合、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。でデータを更新または削除するには copy-on-write、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
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。

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 で公開されないプロパティを含めることができません。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内のランダムな 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、 <code>write.object-storage.enabled</code> 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。

既存のテーブルを Iceberg に移行する

現在の Athena または AWS Glue テーブル (Hive テーブルとも呼ばれる) 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。
- フルデータ移行では、Iceberg メタデータレイヤーが作成され、既存のデータファイルが元のテーブルから新しい Iceberg テーブルに書き換えられます。

以下のセクションでは、テーブルの移行に使用できる APIs の概要と、移行戦略を選択するためのガイドランスについて説明します。これら 2 つの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

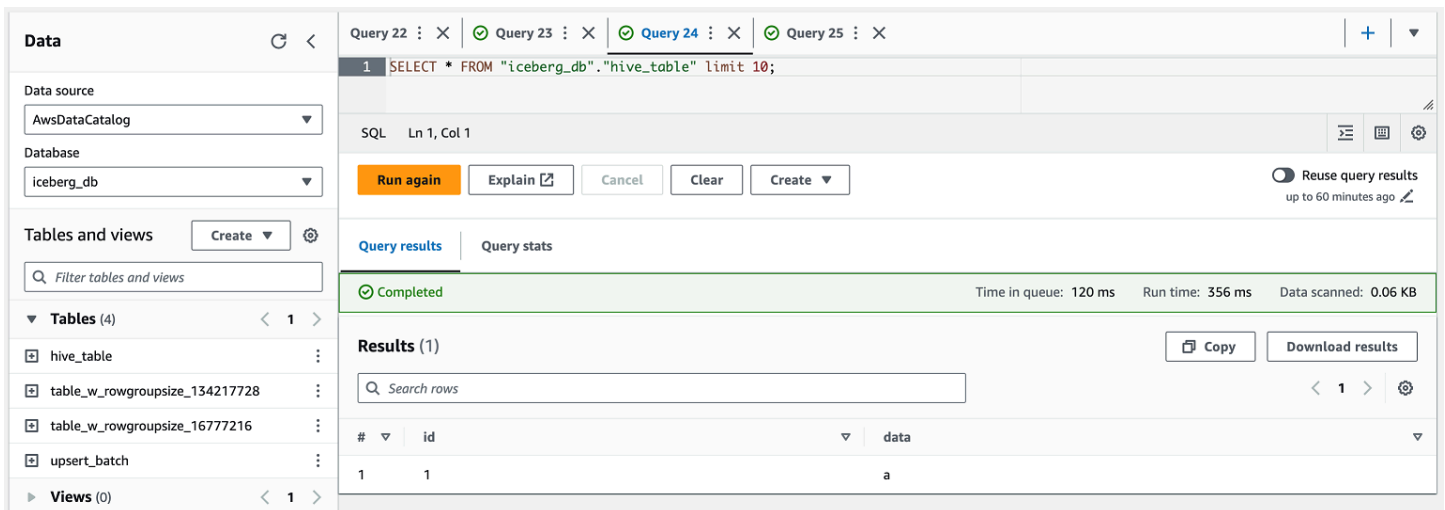
インプレース移行により、すべてのデータファイルを書き換える必要がなくなります。代わりに、Iceberg メタデータファイルが生成され、既存のデータファイルにリンクされます。Iceberg には、インプレース移行を実装するための 3 つのオプションがあります。

- Iceberg ドキュメントの [「スナップショットテーブルと Spark プロシージャ: スナップショット」](#) セクションで説明 snapshot されている手順を使用します。

- Iceberg ドキュメントの「[Add Files and Spark procedure: add_files](#)」セクションで説明 add_files されている手順を使用します。
- migrate 「Iceberg ドキュメント」の「[テーブルと Spark の移行](#)」の手順を使用します。

現在、移行手順はと直接連携しません AWS Glue Data Catalog。Hive メタストアでのみ動作します。snapshot または の代わりに migrate プロシージャを使用する必要がある場合は add_files、Hive メタストア (HMS) で一時的な Amazon EMR クラスターを使用できます。このアプローチには Iceberg バージョン 1.2 以降が必要です。

次の Hive テーブルを作成するとします。



The screenshot shows the AWS Athena console interface. On the left, the 'Data' sidebar is visible, showing the 'Data source' as 'AwsDataCatalog', the 'Database' as 'iceberg_db', and a list of tables including 'hive_table'. The main area displays the SQL query: 'SELECT * FROM "iceberg_db"."hive_table" limit 10;'. Below the query, there are buttons for 'Run again', 'Explain', 'Cancel', 'Clear', and 'Create'. The 'Query results' section shows a 'Completed' status with 'Time in queue: 120 ms', 'Run time: 356 ms', and 'Data scanned: 0.06 KB'. The 'Results (1)' section shows a single row with columns '#', 'id', and 'data', containing the value '1' for 'id' and 'a' for 'data'.

この Hive テーブルを作成するには、Athena コンソールでこのコードを実行します。

```
CREATE EXTERNAL TABLE 'hive_table'(  
  'id' bigint,  
  'data' string)  
USING parquet  
LOCATION  
  's3://datalake-xxxx/aws_workshop/iceberg_db/hive_table'  
  
INSERT INTO iceberg_db.hive_table VALUES (1, 'a')
```

Hive テーブルがパーティション分割されている場合は、パーティションステートメントを含めて、Hive の要件に従ってパーティションを追加します。

```
ALTER TABLE default.placeholder_table_for_migration ADD  
  PARTITION (date = '2023-10-10')
```

ステップ:

1. AWS Glue Data Catalog 統合を有効にせずに Amazon EMR クラスターを作成します。つまり、Hive または Spark テーブルメタデータのチェックボックスをオンにしないでください。これは、この回避策のためにクラスターで利用可能なネイティブ Hive メタストア (HMS) を使用するためです。

AWS Glue Data Catalogue settings

Use the AWS Glue Data Catalog to provide an external metastore for your application.

Use for Hive table metadata

Use for Spark table metadata

2. 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。

```
"spark.sql.extensions": "org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions",  
"spark.sql.catalog.spark_catalog": "org.apache.iceberg.spark.SparkSessionCatalog",  
"spark.sql.catalog.spark_catalog.type": "hive",
```

3. `show databases` または AWS Glue Data Catalog を実行して、Amazon EMR クラスターがに接続されていないことを確認します `show tables`。

```
[2]: %%sql  
show databases  
Last executed at 2023-07-05 12:24:26 in 35.03s  
Starting Spark application
```

ID	YARN Application ID	Kind	State	Spark UI	Driver log	User	Current session?
1	application_1686667730124_0002	pyspark	idle	Link	Link	None	✓

SparkSession available as 'spark'.

Type: Table Pie

namespace

default

4. Amazon EMR クラスターの Hive メタストアに Hive テーブルを登録し、Iceberg migrate プロシージャを使用します。

Register the Hive table in this local HMS catalog pointing to the S3 location where the files from the original Hive tables exist

```
%%sql -q
CREATE TABLE default.placeholder_hive_table (id bigint NOT NULL, data string)
USING parquet
LOCATION 's3://datalake-743490154766/aws_workshop/iceberg_db/hive_table/'
```

Last executed at 2023-07-05 12:55:19 in 3.25s

```
%%sql
select * from default.placeholder_hive_table limit 5
```

Last executed at 2023-07-05 12:57:13 in 7.43s

Type: Table Pie Scatter Line Area Bar

id	data
1	a

Once the Hive table is registered in this local HMS catalog, you can use Iceberg's migrate procedure.

```
spark.sql("CALL spark_catalog.system.migrate('default.placeholder_hive_table')")
```

Last executed at 2023-07-05 13:00:06 in 3.27s

▶ Spark Job Progress

DataFrame[migrated_files_count: bigint]

```
%%sql
show tables from default
```

Last executed at 2023-07-05 13:00:49 in 7.42s

Type: Table Pie Scatter Line Area Bar

namespace	tableName	isTemporary
default	placeholder_hive_table	False
default	placeholder_hive_table_backup_	False

この手順では、Hive テーブルと同じ場所に Iceberg メタデータファイルを作成します。

5. 移行した Iceberg テーブルを に登録します AWS Glue Data Catalog。
6. AWS Glue Data Catalog 統合が有効になっている Amazon EMR クラスターに切り替えます。

AWS Glue Data Catalogue settings

Use the AWS Glue Data Catalog to provide an external metastore for your application.

Use for Hive table metadata

Use for Spark table metadata

7. Spark セッションで次の Iceberg 設定を使用します。

```
"spark.sql.extensions": "org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions",
  "spark.sql.catalog.glue_catalog": "org.apache.iceberg.spark.SparkCatalog",
  "spark.sql.catalog.glue_catalog.warehouse": "s3://datalake-xxxx/
aws_workshop",
  "spark.sql.catalog.glue_catalog.catalog-impl":
"org.apache.iceberg.aws.glue.GlueCatalog",
  "spark.sql.catalog.glue_catalog.io-impl":
"org.apache.iceberg.aws.s3.S3FileIO",
```

このテーブルを Amazon EMR、AWS Glue または Athena からクエリできるようになりました。

```

%%sql
show tables from iceberg_db
Last executed at 2023-07-05 13:10:50 in 7.44s

Type: Table Pie Scatter Line Area Bar

namespace      tableName      isTemporary
iceberg_db     hive_table     False
iceberg_db     table_w_rowgroupsize_134217728  False
iceberg_db     table_w_rowgroupsize_16777216   False
iceberg_db     upsert_batch   False
iceberg_db     ws_webpage_pk_partitioned_140gb_trino  False

%%bash
aws s3 ls s3://datalake-743490154766/aws_workshop/iceberg_db/hive_table/metadata/
Last executed at 2023-07-05 13:10:20 in 488ms
2023-07-05 12:00:07      2239 00000-12a20051-6a3f-4b46-bae1-985f6df254db.metadata.json
2023-07-05 12:00:07      5802 b3d40480-0cb9-4cea-a4af-94c40a123689-m0.avro
2023-07-05 12:00:07      3781 snap-6104693268717769849-1-b3d40480-0cb9-4cea-a4af-94c40a123689.avro

metadata_file = "s3://datalake-743490154766/aws_workshop/iceberg_db/hive_table/metadata/00000-12a20051-6a3f-4b46-bae1-985f6df254db.metadata.json"
Last executed at 2023-07-05 13:11:46 in 49ms

spark.sql(f"CALL glue_catalog.system.register_table('iceberg_db.migrated_iceberg_table',{metadata_file})")
Last executed at 2023-07-05 13:12:27 in 3.32s

▶ Spark Job Progress

DataFrame[current_snapshot_id: bigint, total_records_count: bigint, total_data_files_count: bigint]

%%sql -q
alter table glue_catalog.iceberg_db.migrated_iceberg_table SET TBLPROPERTIES('format-version='2')
Last executed at 2023-07-05 13:12:33 in 2.24s

%%sql
select * from glue_catalog.iceberg_db.migrated_iceberg_table limit 5
Last executed at 2023-07-05 13:12:44 in 7.42s

Type: Table Pie Scatter Line Area Bar

id data
1 a

```

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、このオプションはテーブル品質の向上に役立ちます。データの検証、スキーマとパーティションの変更、データのリソートなどを行うことができます。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。

- Amazon EMR の Spark、または Athena で AWS Glue (CREATE TABLE ... AS SELECT [CTAS](#)) ステートメントを使用します。 および TBLPROPERTIES 句を使用して、新しい Iceberg テーブルの PARTITIONED BY パーティション仕様と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ンを微調整できます。

- 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 (Iceberg [ドキュメントの「テーブルの作成」](#) を参照)。

移行戦略の選択

最適な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	レコメンデーション
データファイル形式 (CSV や Apache Parquet など) は何ですか？	<ul style="list-style-type: none">• 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。• 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？	<ul style="list-style-type: none">• 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。例えば、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せん。)• データファイルから列全体を削除する場合は、完全な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？	<ul style="list-style-type: none">• 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 (例えば、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残る)、インプレース移行を検討してください。

質問

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？

テーブルには小さなファイルが多数ありますか？

レコメンデーション

- 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。非表示の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。
- 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。
- 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。

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。ユースケースによって、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、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](#)
- [読み取りパフォーマンスの最適化](#)
- [書き込みパフォーマンスの最適化](#)
- [ストレージの最適化](#)
- [圧縮を使用したテーブルのメンテナンス](#)
- [Amazon S3 での Iceberg ワークロードの使用](#)

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 または を使用して Iceberg AWS Glue を操作する場合は、AWS Glue データカタログを使用するように Spark セッションに次の設定を追加します。詳細については、このガイドの前半の[AWS Glue での Iceberg の Spark 設定](#) セクションを参照してください。

```
"spark.sql.catalog.<your_catalog_name>.catalog-impl":  
"org.apache.iceberg.aws.glue.GlueCatalog"
```

- 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 または を使用して 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「[Optimistic Locking](#)」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、テーブルプロパティ を使用して変更できません `write.<file_type>.compression-codec`。Athena は、Iceberg テーブルのデフォルトの圧縮コーデックとして ZSTD を既に使用しています。

一般に、GZIP と Snappy のバランスをとり、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、ZSTD 圧縮コーデックを使用することをお勧めします。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は最適ですが、GZIP および ZSTD よりも圧縮率が低くなります。Amazon S3 により大きなデータボリュームを保存することを意味する場合でも、パフォーマンスを優先する場合、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 は不要なメタデータファイルやデータファイルを読み取らないように、インデックス作成のプライマリレイヤーとしてパーティションを使用します。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、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合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。例えば、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多すぎないように、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 (例えば、数千の値を持つことができる `id` 列) 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。

非表示のパーティショニングを使用する

クエリがテーブル列の派生をフィルタリングすることが一般的である場合は、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のではなく、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](#) ドキュメントを参照してください。

例えば、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、日付または時刻でのパーティション化。Iceberg には、タイムスタンプの日付または時刻の部分から抽出するための複数の変換が用意されています。
- パーティション化列の基数が高く、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ン化します。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。

非表示の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てください。例えば、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、テーブルの最適なパーティション戦略が当初は明確ではなく、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改良したい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 拡張機能](#)」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 ファイルを 100 MB 以上にしておく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、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)
<code>write.target-file-size-bytes</code>	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。
<code>write.parquet.row-group-size-bytes</code>	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を読み取らないようにできます。
<code>write.orc.stripe-size-bytes</code>	64 MB	
<code>write.distribution-mode</code>	なし、Iceberg バージョン 1.1 以前の場合 Iceberg バージョン 1.2 以降のハッシュ	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。
 - 小さなテーブル (最大数ギガバイト) – ターゲットファイルのサイズを 128 MB に減らします。また、行グループまたはストライプのサイズを小さくします (例えば、8 MB または 16 MB)。
 - 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 1024 MB 以上に増やし、クエリで通常大量のデータセットがプルされる場合は、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には、`write.distribution-mode` プロパティを `hash` または `range` に設定します。これらのモードの違いの詳細な説明については、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。

通常のコマンドを実行する

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、ファイルがそのサイズである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、圧縮を定期的に行っ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ドラインについては、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ます。これにより、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が向上します。列統計のメリットを享受するには、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する場合 (例えば、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 `write.metadata.metrics.max-inferred-column-defaults` で設定された列の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 列の統計情報が必要ない場合は、`write.metadata.metrics.max-inferred-column-defaults`設定を希望の値 (20 など) に調整し、列の順序を変更して、クエリの読み取りと書き込みを行う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、次の例に示すように、メトリクス収集の全体的なプロパティを無効にし、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場合に最も効果的です。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は、`copy-on-write` 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の方法でストレージに直接書き込まれるため、`copy-on-write` では読み取りパフォーマンスが向上します。ただし、と比較すると `merge-on-read`、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に最適です (例えば、毎日のバッチロードなど)。

`copy-on-write` 設定 (`write.update.mode`、`write.delete.mode`、および `write.merge.mode`) は、テーブルレベルで設定することも、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が使用されている場合があります。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。Iceberg の列統計と組み合わせたソートにより、ファイルプルーニングの効率が大幅に向上し、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。

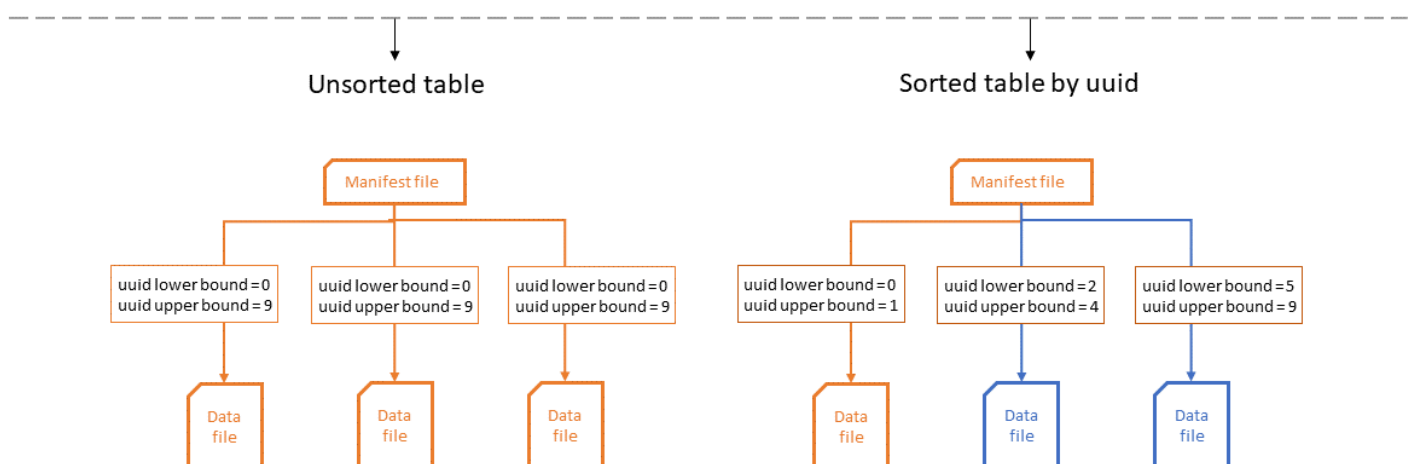
Spark でデータ定義言語 (DDL) ステートメントを実行することで、テーブルレベルで階層ソート順序を設定できます。使用可能なオプションについては、[Iceberg](#) ドキュメントを参照してください。ソート順序を設定すると、ライターはこのソートを Iceberg テーブルの後続のデータ書き込みオペレーションに適用します。

例えば、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、任意のデータファイルに存在する uuid 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。

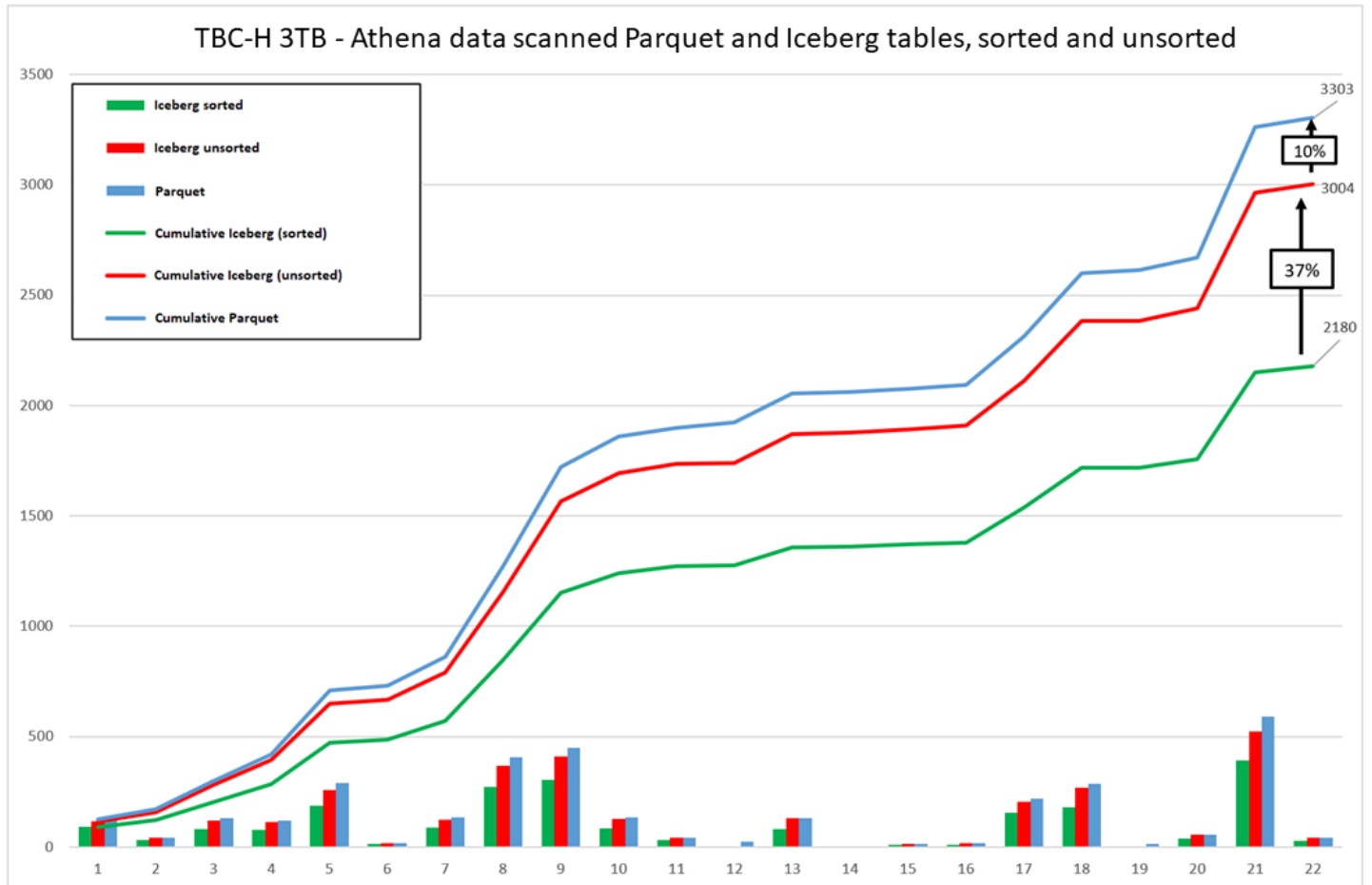
Query example:

```
SELECT * FROM Table
WHERE date > 2022-02-05 AND date < 2022-02-10 AND uuid = 1
```



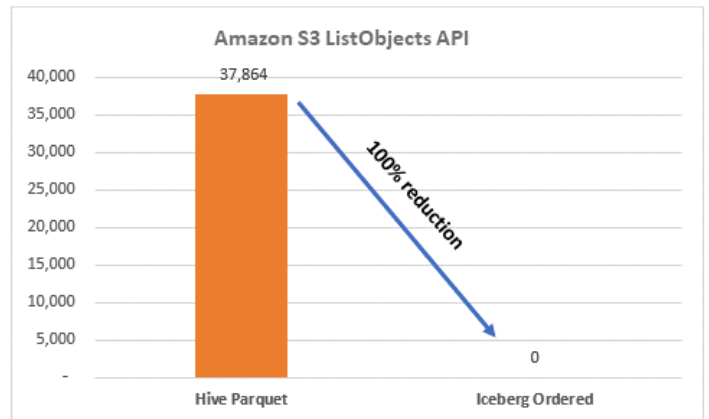
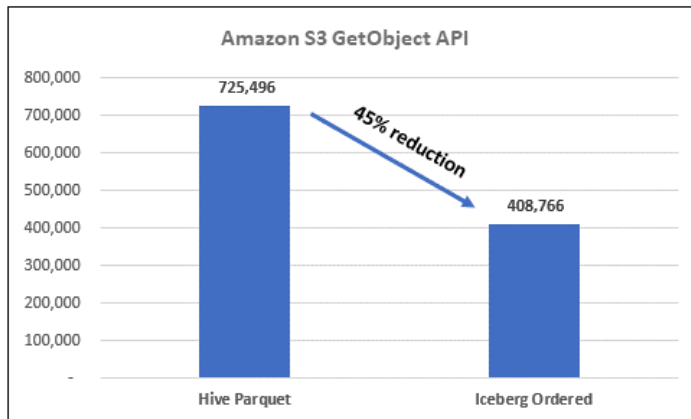
ソート順序を変更しても、既存のデータファイルには影響しません。Iceberg 圧縮を使用して、ソート順を適用できます。

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る可能性があります。



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、他のデータセットやワークロードでは結果が異なる場合があります。

TPC-H 3TB - 22 queries



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では、`write.distribution-mode` を `none` に設定します。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 タスクで利用可能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

`write.distribution-mode` を `none` に設定すると、多数の小さなファイルが生成される傾向があり、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適切なサイズのファイルに統合してクエリのパフォーマンスを向上させるには、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

最新のデータの読み込みオペレーションが遅いことがユースケースで許容できる場合は、merge-on-read 戦略を使用して書き込みパフォーマンスを最適化します。

を使用すると merge-on-read、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損なわれますが、更新と削除の書き込みが高速化されます。通常、merge-on-read は、多数のテーブルパーティションに分散された更新プログラムや少数の更新ジョブを含むストリーミングワークロードに最適です。

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

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 merge-on-read 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上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 形式で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、通常は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で `write-format` を に設定して Avro 形式で Avro 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、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 のデフォルトの形式

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。

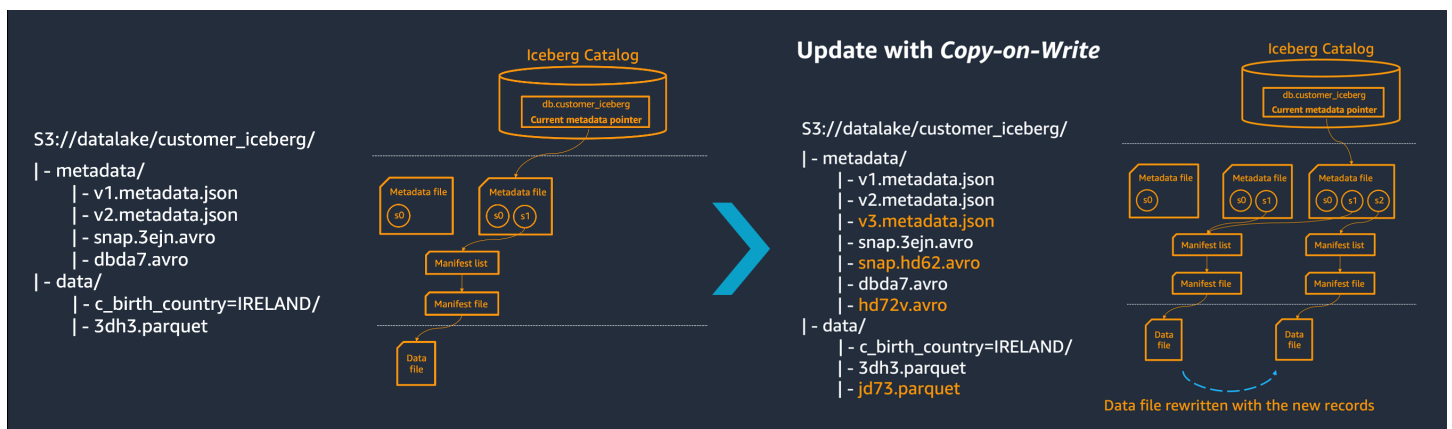
```
spark.sql(f"""
CREATE TABLE IF NOT EXISTS glue_catalog.{DB_NAME}.{TABLE_NAME} (
  Col_1 float,
  <<<...other columns...>>
  ts timestamp)
USING iceberg
PARTITIONED BY (days(ts))
OPTIONS (
  'format-version'='2',
  write.format.default='parquet')
""")

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

次の図に示すように、Iceberg テーブルのデータを更新または削除すると、データのコピー数が増加します。圧縮の実行にも同じことが当てはまります。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すると、アクセスパターンが変更されたときに、最もコスト効率の高いアクセス階層にデータを自動的に移動できます。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)。S3

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	<ul style="list-style-type: none"> 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。 	この方法では、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が削除されます。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期間を設定できます。

設計パターン

特定のスナップショットの保持ポリシーを設定する

解決策

- または、Amazon EMR で Spark または を使用してスナップショット AWS Glue 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「[endover_snapshots](#)」を参照してください。

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。

例えば、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 または で Spark 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。

ユースケース

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。失効した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。

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

古いスナップショットのアーカイブ

解決策

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:

```
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 または Spark AWS Glue を使用して、[スナップショットを削除します](#)。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て Iceberg テーブルメタデータからデータタッチします。
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 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 :

ユースケース

このパターンでは、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。

最初にそれらのバージョンを新しいテーブルとして復元しない限り、アーカイブされたスナップショットにタイムトラベルまたはロールバックすることはできません。これは通常、監査目的で許容されません。

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
	<ul style="list-style-type: none">• アーカイブされたオブジェクト を復元します。• 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。 <p>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。</p>	

孤立した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どのテーブルにも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 の [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 で Spark または を使用して `remove_orphan_files` プロシージャ AWS Glue 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、データのファイルへ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。

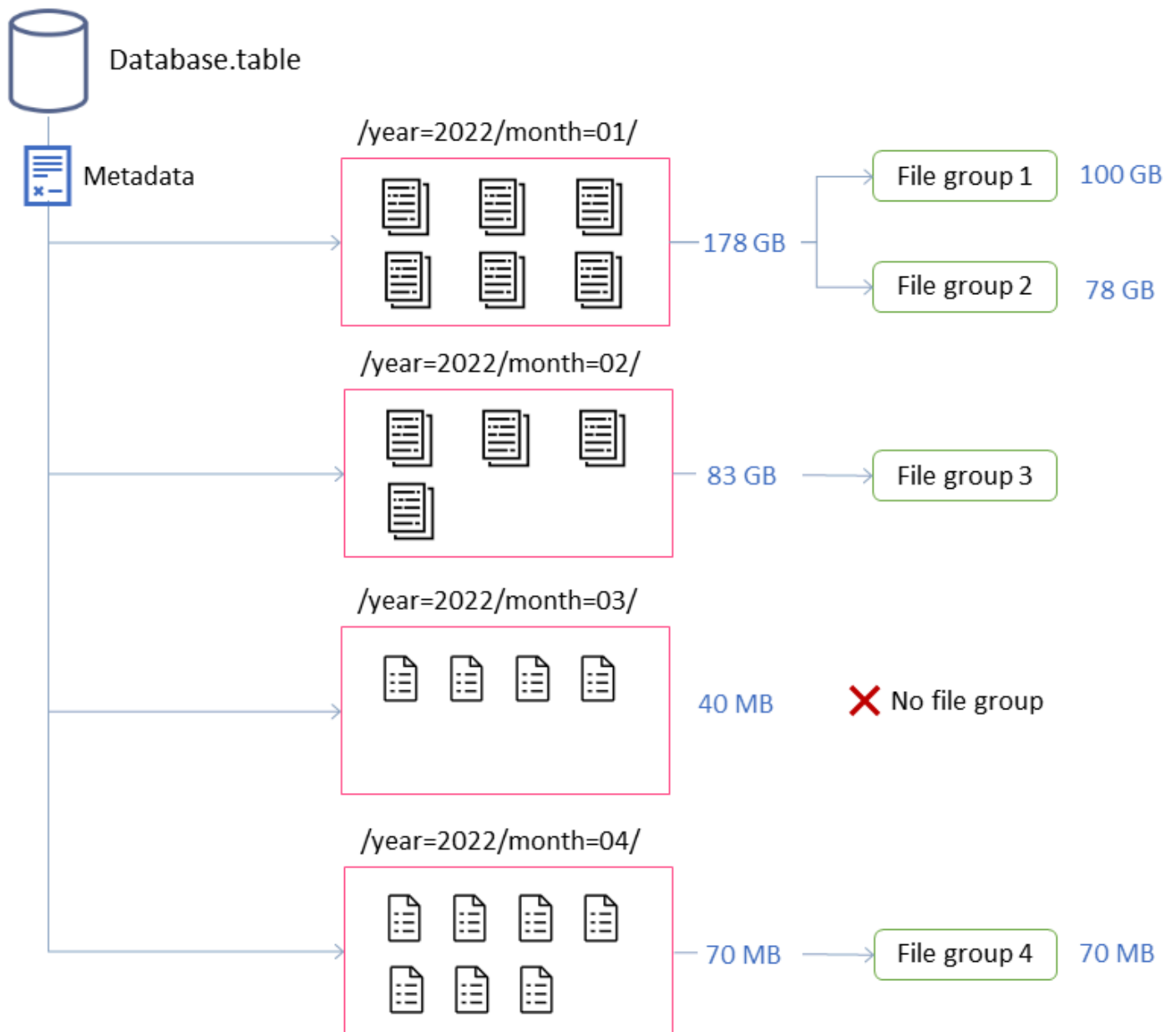
Iceberg 圧縮

Iceberg では、圧縮を使用して次の 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、または書き込みや更新に適したソート順で書き込むことができます。
- スペース塗りつぶし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 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と、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 1 つ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、作業を計画し、進行するためのオペレーションを分割できます。

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。例えば、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も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決定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であるため、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。

例えば、ラベルが付けられたパーティションには、最大サイズの制約である 100 GB 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、デフォルトの最小入力ファイル要件である 5 つのファイル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。

これらのパラメータは、Amazon EMR または で実行されている Spark に対して設定できます 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、Amazon EC2 または Amazon EKS の Amazon EMR、または でも同じ方法を使用できます AWS Glue。

ファイルグループと Spark ジョブ間の相関関係を利用して、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を 100 GB にすることを考慮して、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 をスケールすることもできます。

- 手動スケーリング (例えば、4 倍)
 - `MAX_CONCURRENT_FILE_GROUP_REWRITES = 4` (係数)
 - `spark.executor.instances = 5` (この例で使用されている値) \times 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、アクションは不要)
 - `MAX_CONCURRENT_FILE_GROUP_REWRITES = N` (この値をデフォルトで 100 `spark.dynamicAllocation.maxExecutors` のに調整します。この例のエグゼキューター設定に基づいて、`spark.executor.instances` を $20N$ 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 による圧縮の実行

Athena は、[OPTIMIZE ステートメント](#) を通じて Iceberg の圧縮ユーティリティをマネージド機能として実装します。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 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、ステートメントに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

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

イベントに基づいてビンパッキング圧縮を実行する

レコメンデーション

- 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、`per-table-partition` ベース OPTIMIZE として実行します。
- 大量の小さなファイルが圧縮される可能性がある場合は、Amazon EMR または Spark を動的スケールリング AWS Glue で使用します。
- 大量の小さなファイルが圧縮される可能性がある場合は、Amazon EMR または Spark を動的スケールリング AWS Glue で使用します。

ユースケース

圧縮を実行してデータをソートする

z 順序ソートを使用して圧縮を実行してデータをクラスター化する

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があるパーティションで圧縮を実行する

レコメンデーション

- ソートはコストがかかり 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、Amazon EMR または を使用します。
- Amazon EMR または を使用します。z 順序ソートは非常にコストがかかり 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。
- 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。

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ラットフォームが発生する可能性があり、通常は 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 rangeが大きなファイルを書き込むように 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「書き込みプロパティ」](#)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[スナップショットを削除](#)し、[古いメタデータファイルを削除](#)し、[孤立ファイルを削除](#)する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。

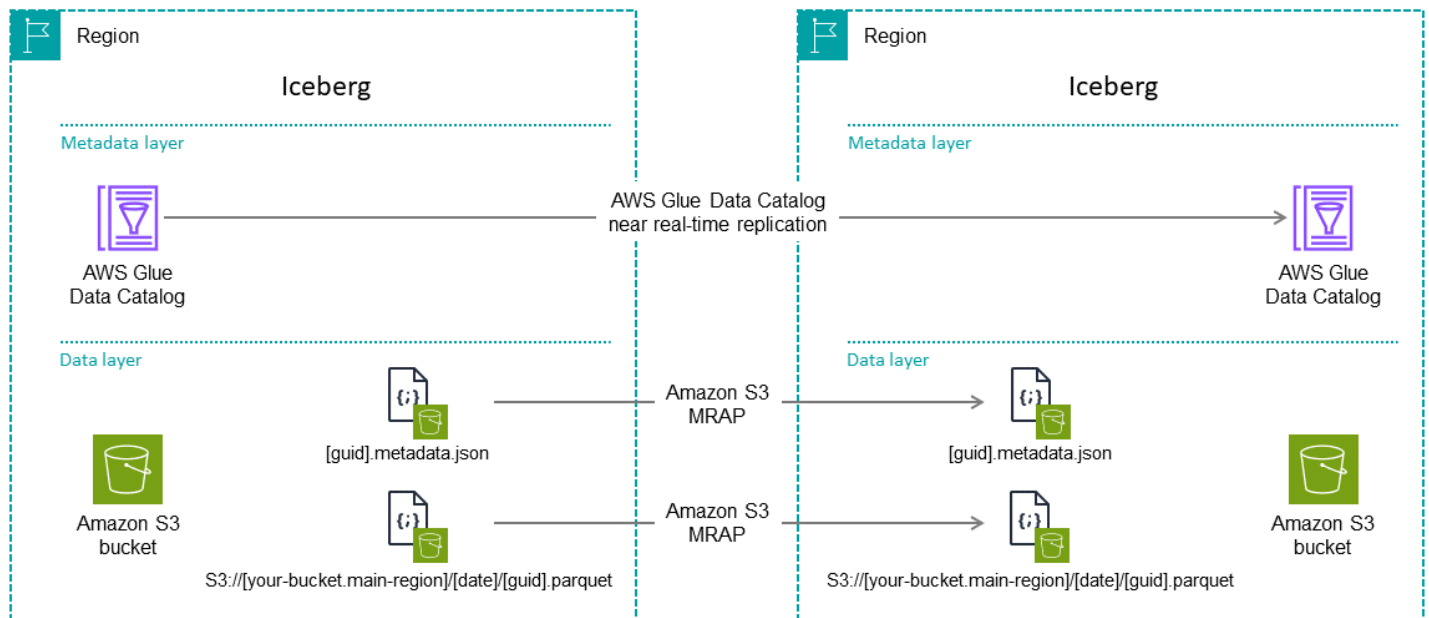
間でデータをレプリケートする AWS リージョン

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 AWS リージョンにデータをレプリケートできます。 <https://docs.aws.amazon.com/AWSAmazonS3/latest/userguide/MultiRegionAccessPoints.html>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) によるデータレプリケーションをサポートしています。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 で使用できるようにします。2 つのオプションから選択できます。
 - レプリケーションを使用して Iceberg テーブルメタデータを AWS Glue Data Catalog 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、Glue Catalog GitHub および Lake Formation のアクセス許可レプリケーションリポジトリで使用できます。 <https://>

github.com/aws-samples/lake-formation-permissions-sync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。

- フェイルオーバーが必要なときに、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

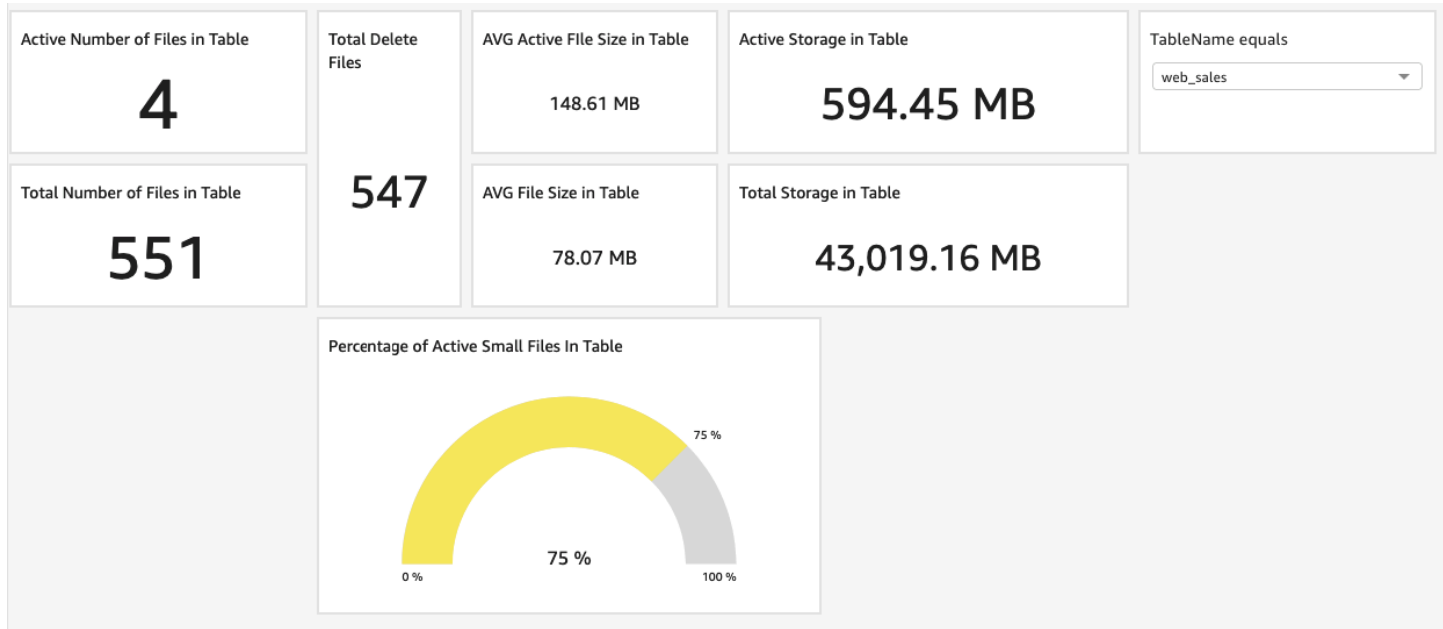
Iceberg ワークロードをモニタリングするには、[メタデータテーブルの分析](#)と[メトリクスレポーターの使用の 2](#)つのオプションがあります。メトリクスレポーターは Iceberg バージョン 1.2 で導入され、REST カタログと JDBC カタログでのみ使用できます。

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、クエリが許容可能なレベルを超えて遅くなるのを防ぐことができます。

テーブルレベルのモニタリング

次の画面は、Amazon 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 QuickSight。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の数やストレージの合計などの詳細なメトリクスをキャプチャします。その後、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 QuickSight 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。



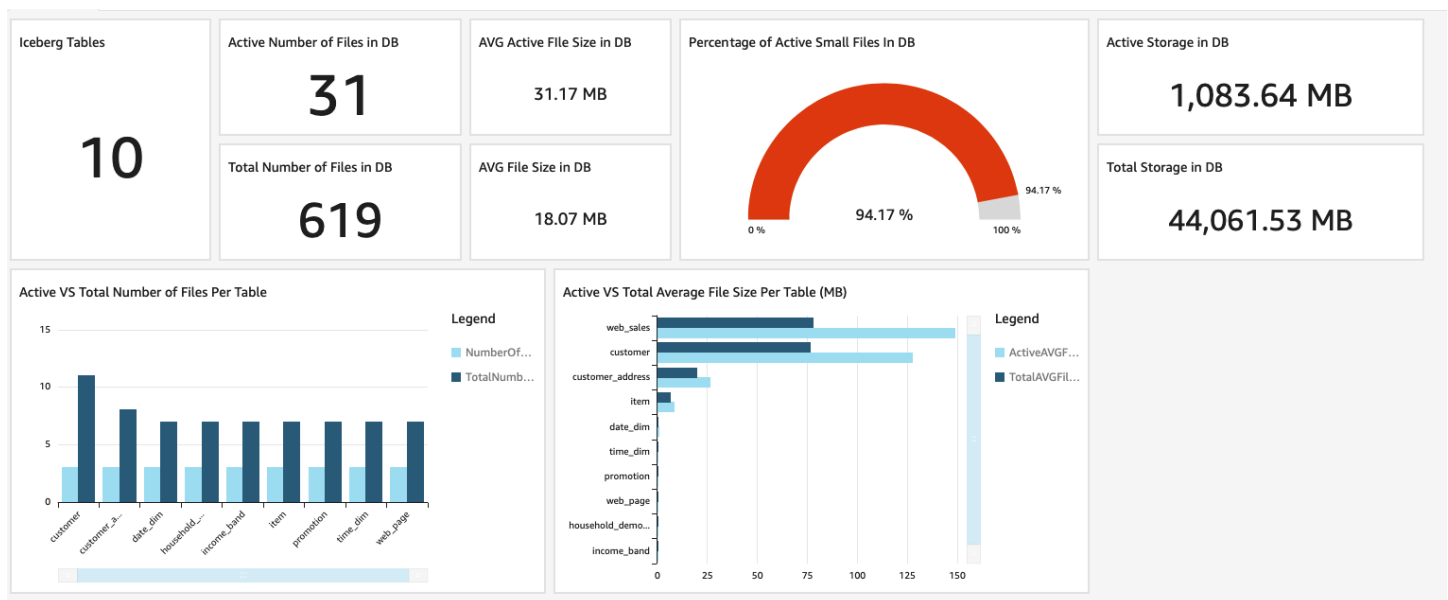
QuickSight このダッシュボードの例では、Iceberg テーブルについて以下の主要業績評価指標 (KPIs) を収集します。

KPI	説明	Query
ファイル数	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)	<pre>select count(*) from <catalog.database. table_name>.all_files</pre>
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	<pre>select count(*) from <catalog.database. table_name>.files</pre>
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をメガバイト単位で表します。	<pre>select avg(file_ size_in_bytes)/100 0000 from <catalog.database. table_name>.all_files</pre>
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をメガバイト単位で表します。	<pre>select avg(file_ size_in_bytes)/100 0000 from <catalog.database. table_name>.files</pre>
小さなファイルの割合	100 MB 未満のアクティブなファイルの割合	<pre>select cast(sum(case when file_size _in_bytes < 100000000 then 1 else 0 end)*100/ count(*) as decimal(1 0,2)) from <catalog.database. table_name>.files</pre>
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	<pre>select sum(file_ size_in_bytes)/100 0000</pre>

KPI	説明	Query
	ファイルの合計サイズ (有効な場合)	<pre>from <catalog.database.table_name>.all_files</pre>
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	<pre>select sum(file_size_in_bytes)/1000000 from <catalog.database.table_name>.files</pre>

データベースレベルのモニタリング

次の例は、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 QuickSight を提供するために作成された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、次の KPIs。

KPI	説明	Query
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)	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。

KPI	説明	Query
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をメガバイト単位で表します。	
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をメガバイト単位で表します。	
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 (有効な場合)	
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、事後対応型ではなく予防型からテーブルのメンテナンスにアプローチできます。例えば、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削除ファイルをマージするには、ビンパッキング圧縮を使用します。
- 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、既に圧縮された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。

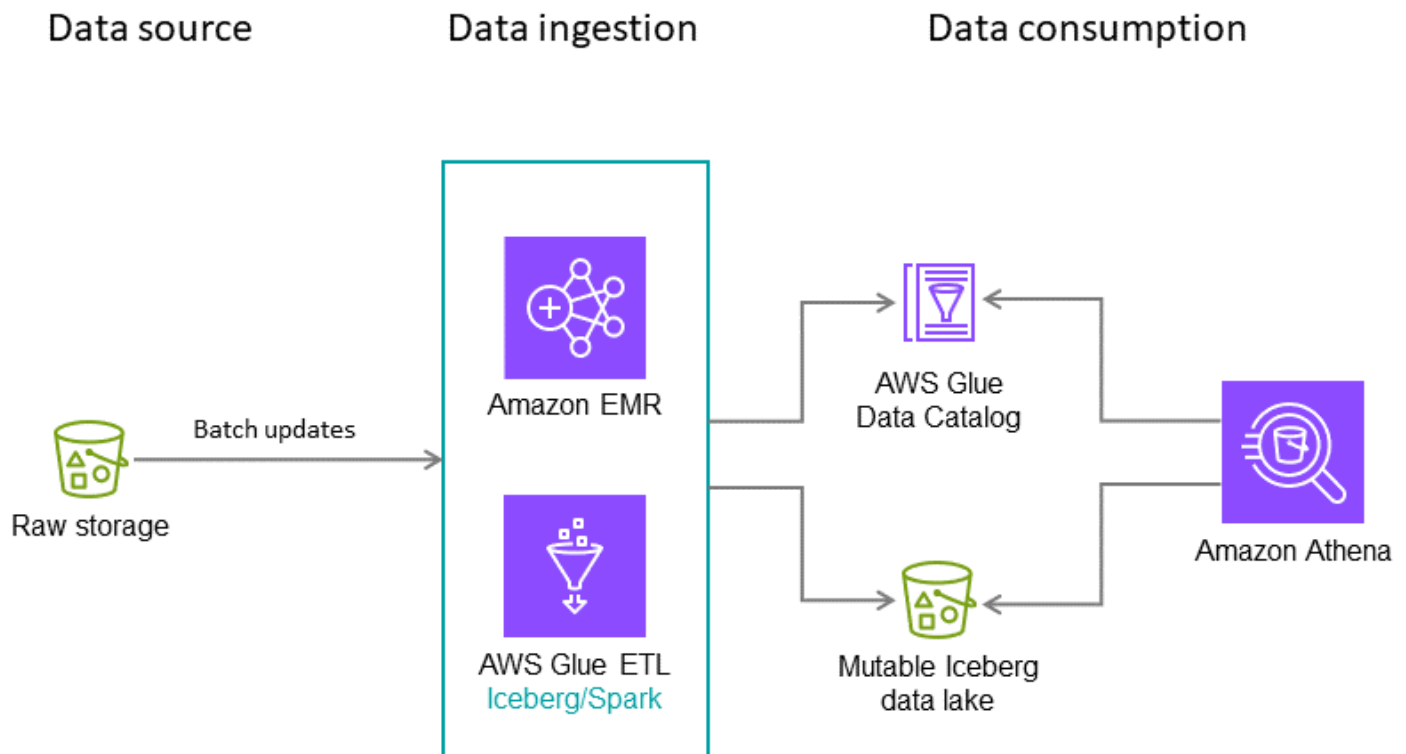
での Apache Iceberg のリファレンスアーキテクチャ AWS

このセクションでは、バッチ取り込みや、バッチとストリーミングの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。

夜間のバッチ取り込み

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: これは一例にすぎません。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:

- Apache Spark タスクは 128 MB のチャンクでデータを処理するため、ファイルサイズ: 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、データが読み取り最適化方式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: 年/月/日。仮定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「[Apache Iceberg を使用してトランザクションデータレイクを構築する AWS Glue](#)」と「[と AWS Lake Formation Amazon Athena を使用して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](#)」
- [Amazon QuickSight ドキュメント](#)
- [Glue Catalog と Lake Formation のアクセス許可のレプリケーション](#) (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

このガイドは、以下の人が AWS 作成、共同作成、レビューしました。

寄稿者

- Carlos Rodrigues、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
- Stefano Sandona、ビッグデータ、ソリューションアーキテクト
- Arun A K、ソリューションアーキテクト、ビッグデータ、ETL
- フランシスコ・モリロ、ソリューションアーキテクト、ストリーミング
- Amazon EMR、分析アーキテクト、Suthan Phillips
- Sercan Karaoglu、ソリューションアーキテクト
- 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
- Sofia Zilberman、ソリューションアーキテクト、ストリーミング
- Ismail Makhoul、ソリューションアーキテクト、分析
- 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

レビュー担当者

- Rick Sears、Amazon EMR 担当 総マネージャー
- Linda OConnor Amazon EMR スペシャリスト
- Ian Meyers、Amazon EMR ディレクター
- Vinita Ananth、Amazon EMR 製品管理担当ディレクター
- Jason Berkowitz、プロダクトマネージャー、AWS Lake Formation
- Amazon Redshift、プロダクトマネージャー、Mahesh Mishra
- Vladimir Zlatkin、マネージャー、ソリューションアーキテクト、ビッグデータ
- Karthik Prabhakar、Amazon EMR、分析アーキテクト

- Amazon EMR、ソフトウェア開発エンジニア、Jack Ye
- Vijay Jain、プロダクトマネージャー
- Anupriti Warade、Amazon S3 製品マネージャー
- モリー・ブラウン、全般マネージャー、AWS Lake Formation
- 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

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
初版発行	—	2024 年 4 月 30 日

AWS 規範的ガイドの用語集

以下は、AWS 規範的ガイドが提供する戦略、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します。
- 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: オンプレミスの Oracle データベースをの Oracle 用 Amazon Relational Database Service (Amazon RDS) に移行します AWS クラウド。
- 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: カスタマーリレーションシップ管理 (CRM) システムを Salesforce.com に移行します。
- 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: オンプレミスの Oracle データベースをの EC2 インスタンス上の Oracle に移行します AWS クラウド。
- 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。サーバーを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、およびそれらを行き移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。

- 使用停止 — 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コントロール」](#)を参照してください。

抽象化されたサービス

[「マネージドサービス」](#)を参照してください。

ACID

[「原子性、一貫性、分離性、耐久性」](#)を参照してください。

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](#)よりも柔軟ですが、より多くの作業が必要です。

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の方法。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

行のグループを操作し、グループの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」](#)を参照してください。

AIOps

[「人工知能オペレーション」](#)を参照してください。

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。

アプリケーションコントロール

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](#)の需要要素であり、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?](#)」を参照してください。

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。

アベイラビリティゾーン

他のアベイラビリティゾーンの障害から 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)

組織がクラウドに正常に移行 AWS するための効率的で効果的な計画を立てるのに役立つ、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編成します。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、組織がクラウド導入を成功させるための準備に役立つ、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

[「事業継続計画」](#)を参照してください。

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[Data in a behavior graph](#)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。[エンディアンネス](#)も参照してください。

二項分類

バイナリ結果 (2 つの可能なクラスのうちの一つ) 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

2 つの異なる同一の環境を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は 1 つの環境 (青) で実行し、新しいアプリケーションバージョンは他の環境 (緑) 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インタラクション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

トの中には、個人や組織に混乱を与えたり、損害を与えたりすることを意図しているものがあります。

ポットネット

[マルウェア](#)に感染し、[ポット](#)のヘルダーまたはポットオペレーターと呼ばれる、単一関係者の管理下にあるポットのネットワーク。ポットは、ポットとその影響をスケールするための最もよく知られているメカニズムです。

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、「[ブランチについて](#) (GitHub ドキュメント)」を参照してください。

ブレイクグラスアクセス

例外的な状況や承認されたプロセスを通じて、ユーザーが通常アクセス許可を持たない AWS アカウント 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、Well-Architected [ガイド](#)の「[ブレイクグラス手順の実装](#)」インジケータ AWS を参照してください。

ブラウフィールド戦略

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、ホワイトペーパー [AWSでのコンテナ化されたマイクロサービスの実行](#) 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。](#)

Canary デプロイ

エンドユーザーへのバージョンの低速かつ増分的なリリース。確信できたら、新しいバージョンをデプロイし、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

[「データキャプチャの変更」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します。[AWS Fault Injection Service \(AWS FIS \)](#) を使用して、AWS ワークロードに負荷をかけ、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 サービス を受信する前に、ローカルでデータを暗号化します。

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 の投稿](#)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](#)テクノロジー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

組織が移行するときに通常実行する 4 つのフェーズ AWS クラウド：

- 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。移行戦略とどのように関連しているかについては、AWS [「移行準備ガイド」](#)を参照してください。

CMDB

[「設定管理データベース」](#)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 またはが含まれます AWS CodeCommit。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ます。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などのビジュアル形式から情報を分析および抽出する [AI](#) の分野。例えば、はオンプレミスのカメラネットワークに CV を追加するデバイス AWS Panorama を提供し、Amazon SageMaker は CV の画像処理アルゴリズムを提供します。

設定ドリフト

ワークロードの場合、設定は想定した状態から変化します。これにより、ワークロードが非標準になる可能性があり、通常は段階的かつ意図的ではありません。

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

一元化された管理とガバナンスにより、分散型の分散型データ所有権を提供す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していることを確認できます。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通常、大量の履歴データが含まれており、クエリや分析によく使用されます。

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る。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。

ディープラーニング

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。

defense-in-depth

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。この戦略をに採用するときは AWS、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て、リソースの安全性を確保します。例えば、defense-in-depth 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で利用できるサービス](#)を参照してください。

デプロイメント

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、Implementing security controls on AWSの[Detective controls](#)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ユファクチャリング・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](#) では、ファクトテーブル内の量的データに関するデータ属性を含む小さなテーブル。ディメンションテーブル属性は通常、テキストフィールドまたはテキストのように動作する離散数値です。これらの属性は、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一般的に使用されます。

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)

[災害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に使用する戦略とプロセス](#)。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の追跡。例えば、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 ガバナンス要件への準拠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

[「開発値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。

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、応答時間を短縮できます。

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プロセス。

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 (アカウンティング、[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利用できる、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](#) の中央テーブル。事業運営に関する定量的データを保存します。通常、ファクトテーブルには、メジャーを含む列とディメンションテーブルへの外部キーを含む列の 2 種類の列が含まれます。

フェイルファスト

頻繁で段階的なテストを使用して開発ライフサイクルを短縮する哲学。これはアジャイルアプローチの重要な部分です。

障害分離境界

では AWS クラウド、障害の影響を制限し AWS リージョン、ワークロードの耐障害性を向上させるアベイラビリティゾーン、コントロールプレーン、データプレーンなどの境界です。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。

FGAC

[「きめ細かなアクセスコントロール」](#) 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

段階的なアプローチを使用するのではなく、[変更データキャプチャ](#)による継続的なデータレプリケーションを使用して、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。

G

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)

Amazon では CloudFront、特定の国のユーザーがコンテンツディストリビューションにアクセスできないようにするオプションです。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 ワークフローはレガシーと見なされ、[トランクベースのワークフロー](#)はモダンで推奨されるアプローチです。

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、Amazon AWS Security Hub、GuardDuty、Amazon Inspector AWS Trusted Advisor、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。](#)

ハイアベイラビリティ (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。

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。緊急性のため、通常、修正は一般的な DevOps リリースワークフローの外で行われます。

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

laC

[「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

[「産業モノのインターネット」](#) 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

既存のインフラストラクチャを更新、パッチ適用、または変更する代わりに、本番ワークロード用の新しいインフラストラクチャをデプロイするモデル。イミュータブルなインフラストラクチャは、[本質的にミュータブルなインフラストラクチャ](#) よりも一貫性、信頼性、予測性が高くなります。詳細については、AWS Well-Architected フレームワークの[「変更不可能なインフラストラクチャを使用したデプロイ」](#) 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

接続、リアルタイムデータ、自動化、分析、AI/ML の進歩を通じて、のビジネスプロセスのモダナイゼーションを指すために 2016 年に [Klaus Schwab](#) によって導入された用語。

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。

産業分野における 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、「[Building an industrial Internet of Things \(IIoT\) digital transformation strategy](#)」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、「[AWS を使用した機械学習モデルの解釈](#)」を参照してください。

IoT

「[モノのインターネット](#)」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。

大規模な移行

300 台以上のサーバの移行。

LBAC

[「ラベルベースのアクセスコントロール」](#) 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

[「7R」](#) 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。[エンディアンネス](#) も参照してください。

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またはプライバシーを侵害するように設計されているソフトウェア。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、不正アクセスにつながる

可能性があります。マルウェアの例としては、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 サービスが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、ユーザーが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。Amazon Simple Storage Service (Amazon S3) と Amazon DynamoDB は、マネージドサービスの例です。これら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、制御するためのソフトウェアシステム。これにより、加工品を現場の完成製品に変換します。

MAP

[「移行促進プログラム」](#)を参照してください。

メカニズム

ツールを作成し、ツールの導入を推進し、調整のために結果を検査する完全なプロセス。メカニズムとは、動作中にそれ自体を強化して改善するサイクルです。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

の組織の一部である管理アカウント AWS アカウントを除くすべての 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。

メッセージキューイングテレメトリトランスポート (MQTT)

リソースに制約のある IoT デバイス用の、[パブリッシュ/サブスクライブ](#)パターンに基づく軽量の machine-to-machine (M2M) 通信プロトコル。

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、およびスケールできます。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)

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。組織がクラウドへの移行のための強固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ちます。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、オペレーション、ビジネスアナリストと所有者、移行エンジニア、デベロッパー、スプリントに取り組む 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には、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: Application Migration Service を使用して Amazon EC2 AWS への移行をリホストします。

Migration Portfolio Assessment (MPA)

に移行するためのビジネスケースを検証するための情報を提供するオンラインツール AWS クラウド。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 に移行するために使用されるアプローチ AWS クラウド。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、「[アプリケーションをモダナイズするための戦略 AWS クラウド](#)」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、[「」の「アプリケーションのモダナイゼーション準備状況の評価 AWS クラウド」](#)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

[「移行ポートフォリオ評価」](#)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。

変更可能なインフラストラクチャ

本番ワークロードの既存のインフラストラクチャを更新および変更するモデル。Well-Architected AWS Framework では、一貫性、信頼性、予測可能性を向上させるために、[イミュータブルな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

[「オリジンアクセスコントロール」](#)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

「[運用レベルの契約](#)」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。

オープンプロセス通信 - 統合アーキテクチャ (OPC-UA)

産業用オートメーション用の machine-to-machine (M2M) 通信プロトコル。OPC-UA は、データの暗号化、認証、認可スキームを備えた相互運用性標準を提供します。

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)

インシデントや潜在的な障害の理解、評価、防止、または範囲の縮小に役立つ質問とそれに関連するベストプラクティスのチェックリスト。詳細については、AWS Well-Architected フレームワークの「[運用準備状況レビュー \(ORR\)](#)」を参照してください。

運用テクノロジー (OT)

産業運用、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するハードウェアおよびソフトウェアシステム。製造では、OT と情報技術 (IT) システムの統合が、[Industry 4.0](#) トランスフォーメーションの主要な焦点です。

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

の組織 AWS アカウント 内のすべての のすべてのイベントをログ AWS CloudTrail 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、ドキュメントの「[組織の証跡の作成](#)」を参照してください。CloudTrail

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から、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。

オリジンアクセスコントロール (OAC)

では CloudFront、Amazon Simple Storage Service (Amazon S3) コンテンツを保護するためのアクセスを制限するための拡張オプションです。OAC は、すべての のすべての S3 バケット AWS リージョン、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)

では CloudFront、Amazon S3 コンテンツを保護するためのアクセスを制限するオプションです。OAI を使用すると、は Amazon S3 が認証できるプリンシパル CloudFront を作成します。認証されたプリンシパルは、特定の CloudFront ディストリビューションを介してのみ S3 バケット内のコンテンツにアクセスできます。[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コントロールが可能です。

ORR

[「運用準備状況レビュー」](#)を参照してください。

OT

[「運用技術」](#)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る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

[「プログラム可能な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

アクセス許可の定義 ([アイデンティティベースのポリシー](#) を参照)、アクセス条件の指定 ([リソースベースのポリシー](#) を参照)、または の組織内のすべてのアカウントに対する最大アクセス許可の定義 AWS Organizations ([サービスコントロールポリシー](#) を参照) が可能なオブジェクト。

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。詳細については、[マイクロサービスでのデータ永続性の有効化](#) を参照してください。

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、「[移行準備状況ガイド](#)」を参照してください。

述語

true または を返すクエリ条件。false 通常は WHERE 句にあります。

述語プッシュダウン

転送前にクエリ内のデータをフィルタリングするデータベースクエリ最適化手法。これにより、リレーショナルデータベースから取得して処理する必要があるデータの量が減少し、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、Implementing security controls on AWS の [Preventative controls](#) 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 ロール AWS アカウント、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](#) 内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

エンジニアリングプロセス全体を通してプライバシーを考慮に入れた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

非準拠のリソースのデプロイを防止するように設計された[セキュリティコントロール](#)。これらのコントロールは、プロビジョニング前にリソースをスキャンします。リソースがコントロールに準拠していない場合、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「[での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)

設計、開発、発売から成長と成熟まで、製品のデータとプロセスのライフサイクル全体にわたる管理。

本番環境

[「環境」](#)を参照してください。

プログラミング可能ロジックコントローラー (NAL)

製造では、マシンをモニタリングし、承認プロセスを自動化する、信頼性が高く適応可能なコンピュータです。

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。

パブリッシュ/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。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](#)では、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できるチャンネルにイベントメッセージを発行できます。システムは、公開サービスを変更せずに新しいマイクロサービスを追加できます。

Q

クエリプラン

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る手順などの一連のステップ。

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

[責任、説明責任、相談、通知 \(RACI\)](#) 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

[責任、説明責任、相談、通知 \(RACI\)](#) を参照してください。

RCAC

[「行と列のアクセスコントロール」](#) を参照してください。

リードレプリカ

読み取り専用で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。

再構築

[「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、他のとは分離され、独立しています。詳細については、[AWS リージョン「を使用できるアカウントを指定する」](#)を参照してください。

回帰

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HOST

[「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 R」を参照してください。](#)

プラットフォーム変更

[「7 R」を参照してください。](#)

再購入

[「7 R」を参照してください。](#)

回復性

中断に耐えたり、中断から回復したりするアプリケーションの機能。で障害耐性を計画する場合、[高可用性](#)と[ディザスタリカバリ](#)が一般的な考慮事項です AWS クラウド。詳細については、[AWS クラウド「レジリエンス」](#)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。サポートを含めると、そのマトリックスは RASCI マトリックスと呼ばれ、サポートを除外すると RACI マトリックスと呼ばれます。

レスポンスコントロール

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、Implementing security controls on AWSの[Responsive controls](#)を参照してください。

保持

[「7R」](#)を参照してください。

廃止

[「7Rs」](#)を参照してください。

ローテーション

攻撃者が認証情報にアクセスすることをより困難にするために、[シークレット](#)を定期的に更新するプロセス。

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 (IdPs) が使用するオープンスタンダード。この機能により、フェデレーテ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 [AWS](#)

Management Console したり、組織内のすべてのユーザーを IAM で作成しなくても 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

[「監視コントロールとデータ収集」](#)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

では AWS Secrets Manager、暗号化された形式で保存する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。シークレット値は、バイナリ、1つの文字列、または複数の文字列にすることができます。詳細については、[Secrets Manager ドキュメントの「Secrets Manager シークレットの内容」](#)を参照してください。

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、[予防的](#)、[検出的](#)、[???応答的](#)、[プロアクティブ](#)の4つの主なタイプがあります。

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に自動的に応答または修正するように設計された、事前定義された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の実装に役立つ[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

クシヨンの例としては、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、認証情報のローテーションなどがあります。

サーバー側の暗号化

送信先にあるデータの、それを受け取る AWS サービス による暗号化。

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

のエンドポイントの URL AWS 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 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、スループットなど、サービスのパフォーマンス側面の測定。

サービスレベルの目標 (SLO)

サービスレベルのインジケータによって測定される、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、お客様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

[「セキュリティ情報とイベント管理システム」](#)を参照してください。

単一障害点 (SPOF)

システムを中断させる可能性のあるアプリケーションの単一の重要なコンポーネントの障害。

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。

split-and-seed モデル

モダナイゼーションプロジェクトのスケールアップ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、[「」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 AWS クラウド」](#)を参照してください。

SPOF

[単一障害点](#)を参照してください。

star スキーマ

トランザクションデータまたは測定データを保存するために 1 つの大きなファクトテーブルを使用し、データ属性を保存するために 1 つ以上の小さなディメンションテーブルを使用するデータベースの組織構造。この構造は、[データウェアハウス](#)またはビジネスインテリジェンスの目的で使用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、宿主にと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ゾーンに存在する必要があります。

監視統制とデータ収集 (SCADA)

製造では、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生産オペレーション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

ユーザーインタラクションをシミュレートして潜在的な問題を検出したり、パフォーマンスをモニタリングしたりする方法でシステムをテストします。[Amazon CloudWatch Synthetics](#) を使用してこれらのテストを作成できます。

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が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するサービスへの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と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「[AWS Organizations を他の AWS のサービスで使用する 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つのピザを食べることができる小さな 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

現在のレコードに関連する行のグループに対して計算を実行する SQL 関数。ウィンドウ関数は、移動平均の計算や、現在の行の相対位置に基づく行の値へのアクセスなどの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

[「書き込み 1 回」](#)を参照し、[多くの](#)を読み取ります。

WQF

[「AWS ワークロード認定フレームワーク」](#)を参照してください。

Write Once, Read Many (WORM)

データを 1 回書き込み、データの削除や変更を防ぐストレージモデル。承認されたユーザーは、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 [イミュータブルな](#) と見なされます。

Z

ゼロデイ 익스プロイト

[ゼロデイ脆弱性](#) を利用する攻撃、通常はマルウェア。

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。

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。