

JAPAN | 2024

aws SUMMIT



AWS-12

データ基盤のコストを最適化する ベストプラクティス

関山 宜孝

アマゾン ウェブ サービス ジャパン合同会社

Principal Big Data Architect, AWS Glue

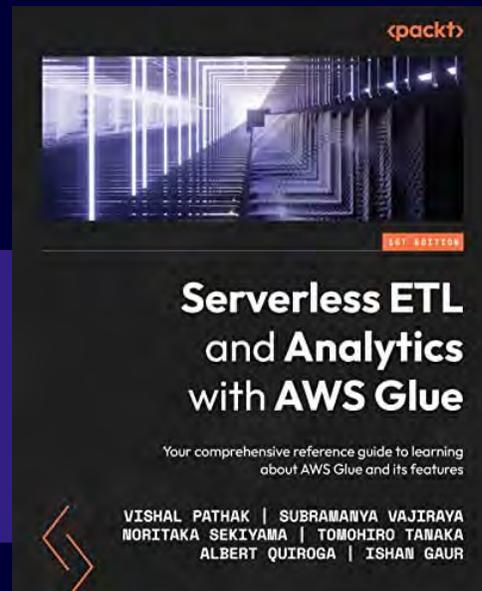
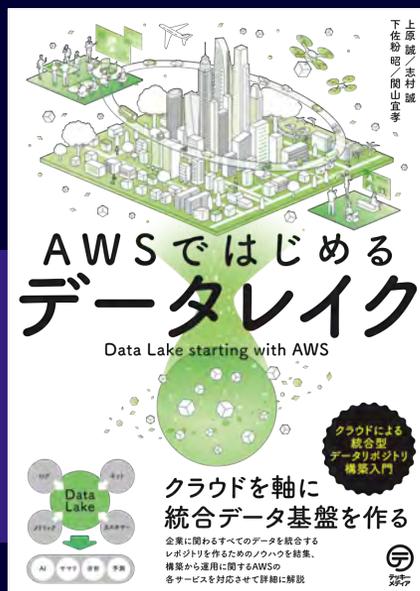


自己紹介

関山 宜孝

Principal Big Data Architect, AWS Glue

- 5年間 AWS サポートにて技術支援を担当
- 2019年にAWS Glue 開発チームにジョイン



@moomindani



moomindani



本セッションについて

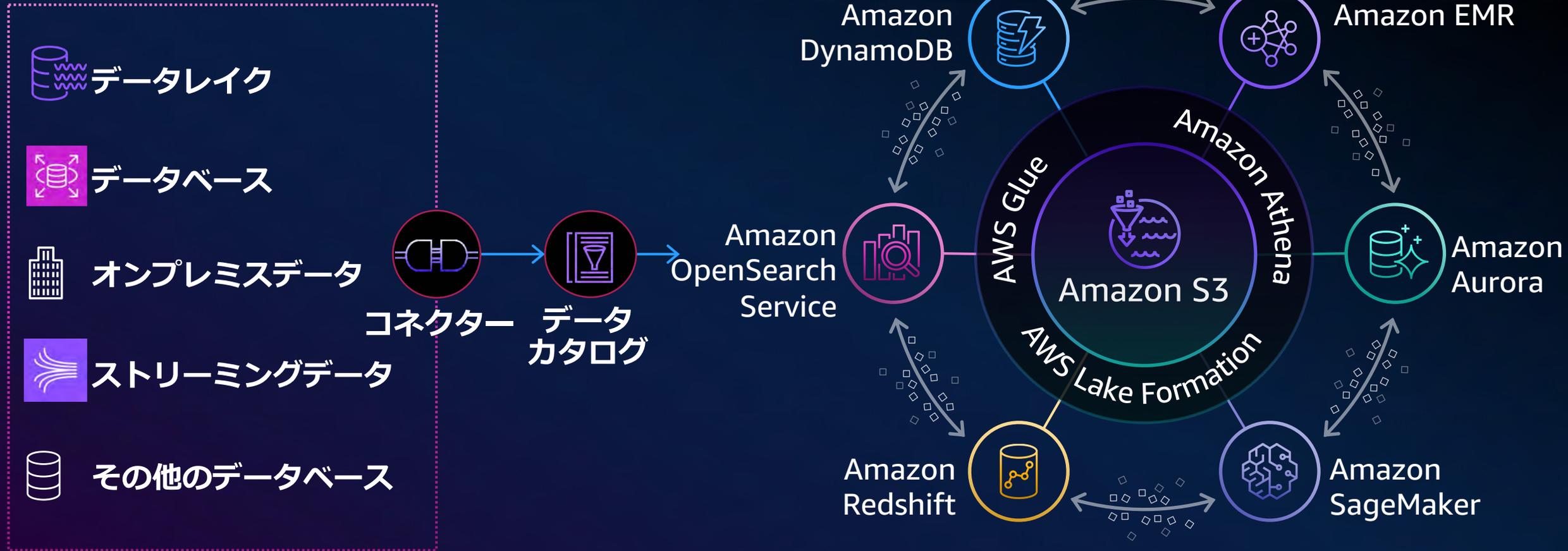
本セッションでは、AWS で費用対効果の高いデータ基盤を構築・運用するベストプラクティスを解説します。

ワークロードに適したサービスやオプションを選択し、コストを犠牲にすることなくスケラブルでパフォーマンスの高いアーキテクチャを設計します。



**昨今、多くのお客様のデータ活用において
コストへの関心が高まっている**

AWS を用いたデータ活用



AWS を用いたデータアーキテクチャの例



データ基盤のコストの課題



サービス横断で
コストが発生する



データ量と処理に
強く依存する

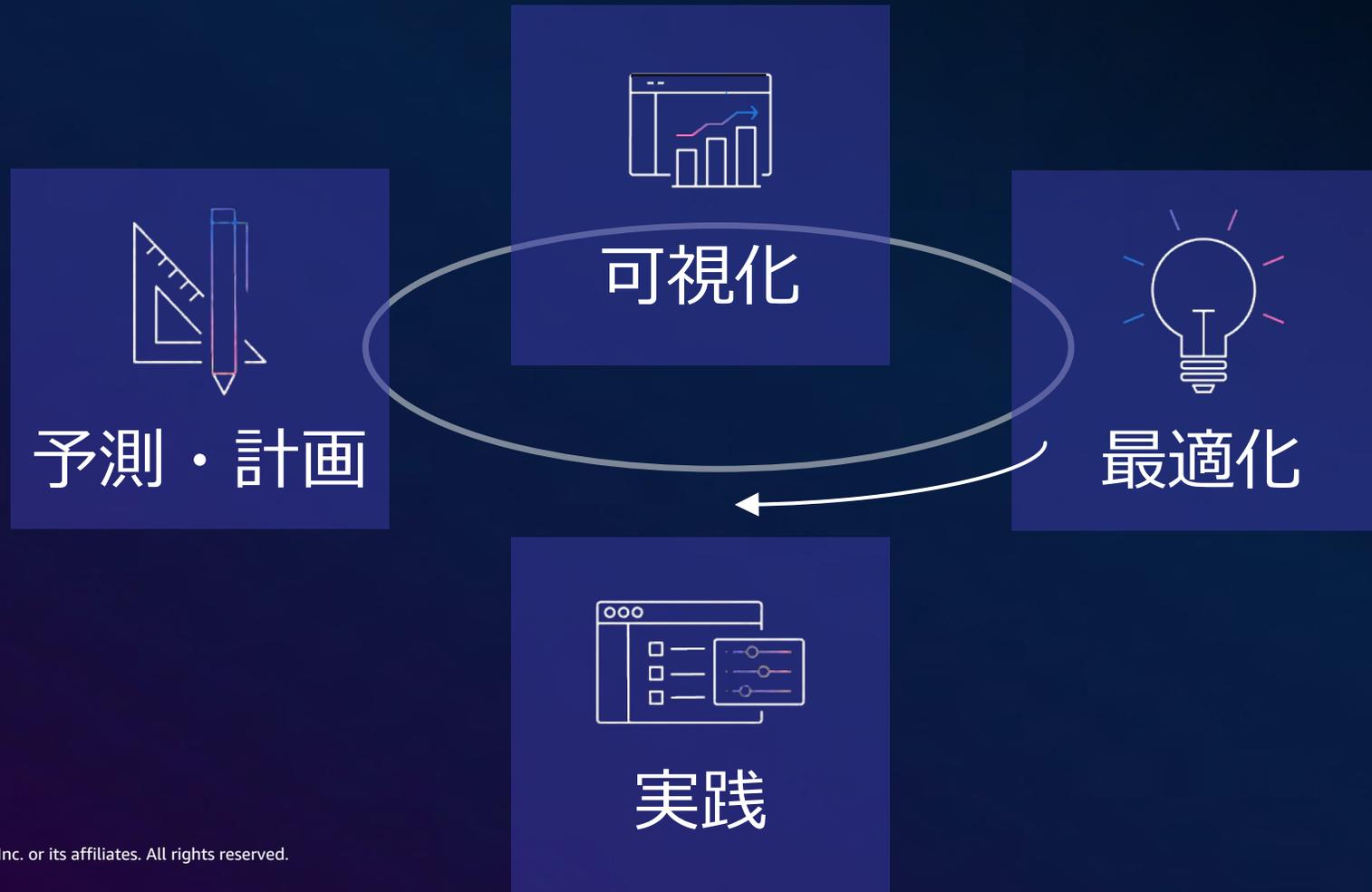


予測が難しい

適切な設計により大幅なコストダウンが可能

Cloud Financial Management (CFM) フレームワーク

AWS Well-Architected Framework で定義されたコスト最適化プロセス

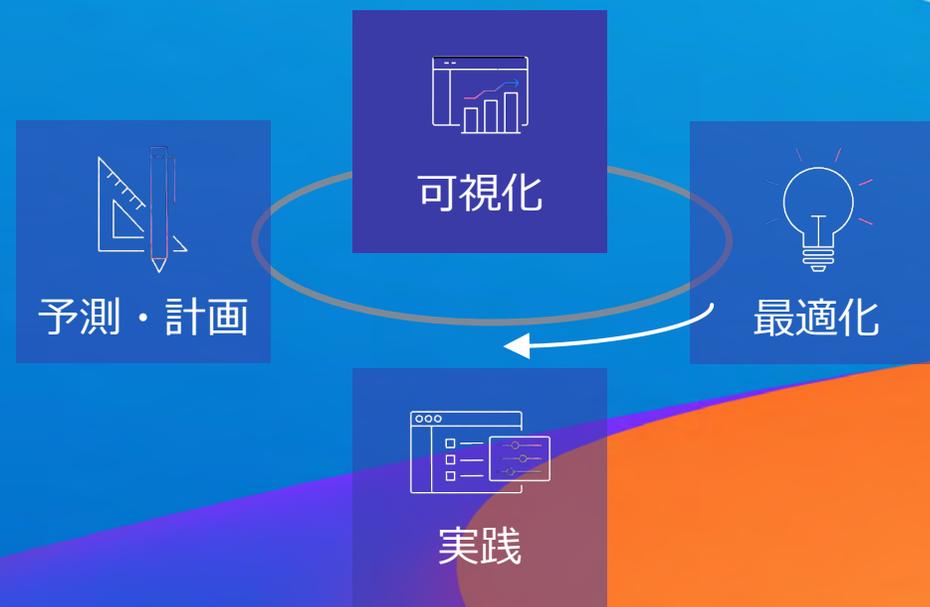


Cloud Financial Management (CFM) フレームワーク

AWS Well-Architected Framework で定義されたコスト最適化プロセス



コストの可視化





If you can't see it,
you can't change it

コストの可視化



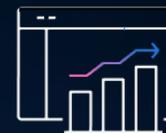
ストレージ



データ転送



データ
パイプライン



クエリ

コストの可視化



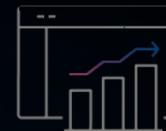
ストレージ



データ転送



データ
パイプライン



クエリ



データソース

データ転送

生データ



加工済みデータ



データマート



Amazon S3

データレイク



Amazon Athena

アドホッククエリ



ダッシュボード



Amazon EMR



AWS Glue

コストの可視化



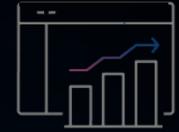
ストレージ



データ転送



データ
パイプライン



クエリ



コストの可視化



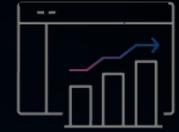
ストレージ



データ転送



データ
パイプライン



クエリ



Amazon S3



Amazon Athena



データソース

データ転送

生データ



加工済みデータ



データマート



Amazon EMR



AWS Glue



アドホッククエリ



ダッシュボード



コストの可視化



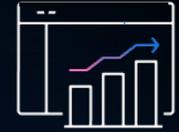
ストレージ



データ転送



データ
パイプライン



クエリ



AWS Cost Explorer



利用量に基づく
AWSコストを確認
する無料ツール



AWS費用を監視・
最適化

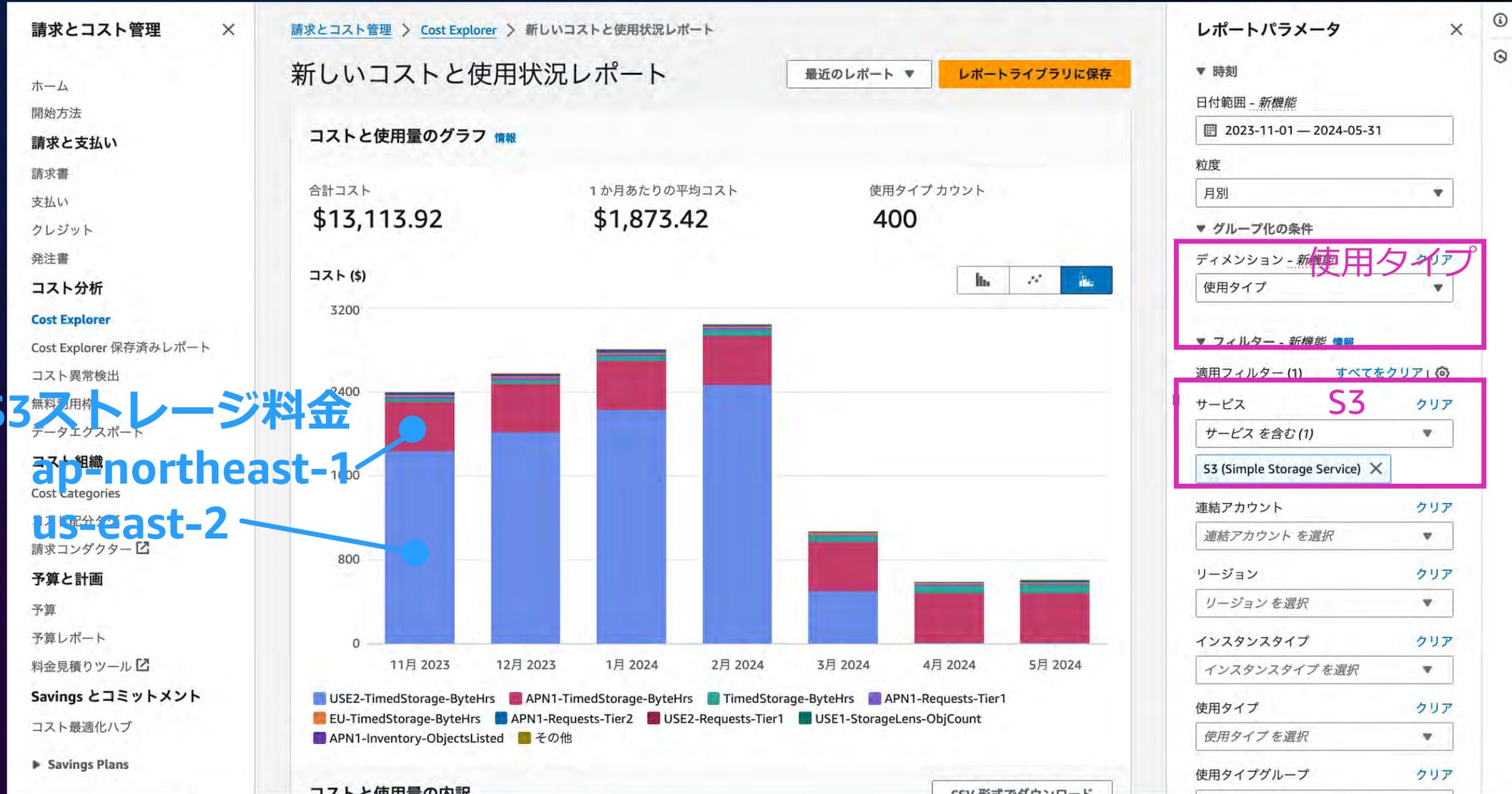


デフォルトで直近
14ヶ月分のコスト
を遡って分析



将来の12ヶ月分の
コストを予測

AWS Cost Explorer : ストレージ

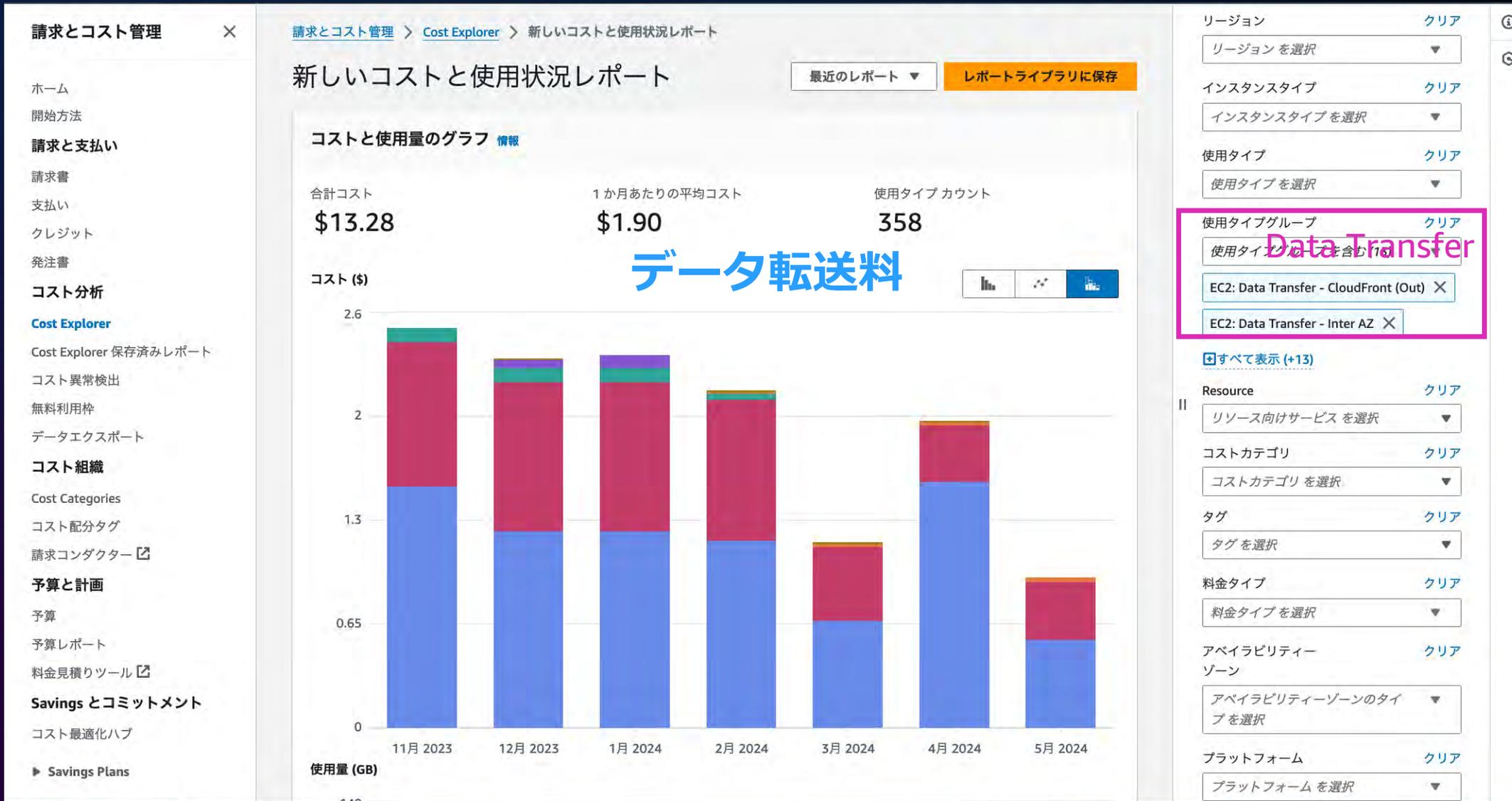


- ストレージ
- データ転送
- データパイプライン
- クエリ

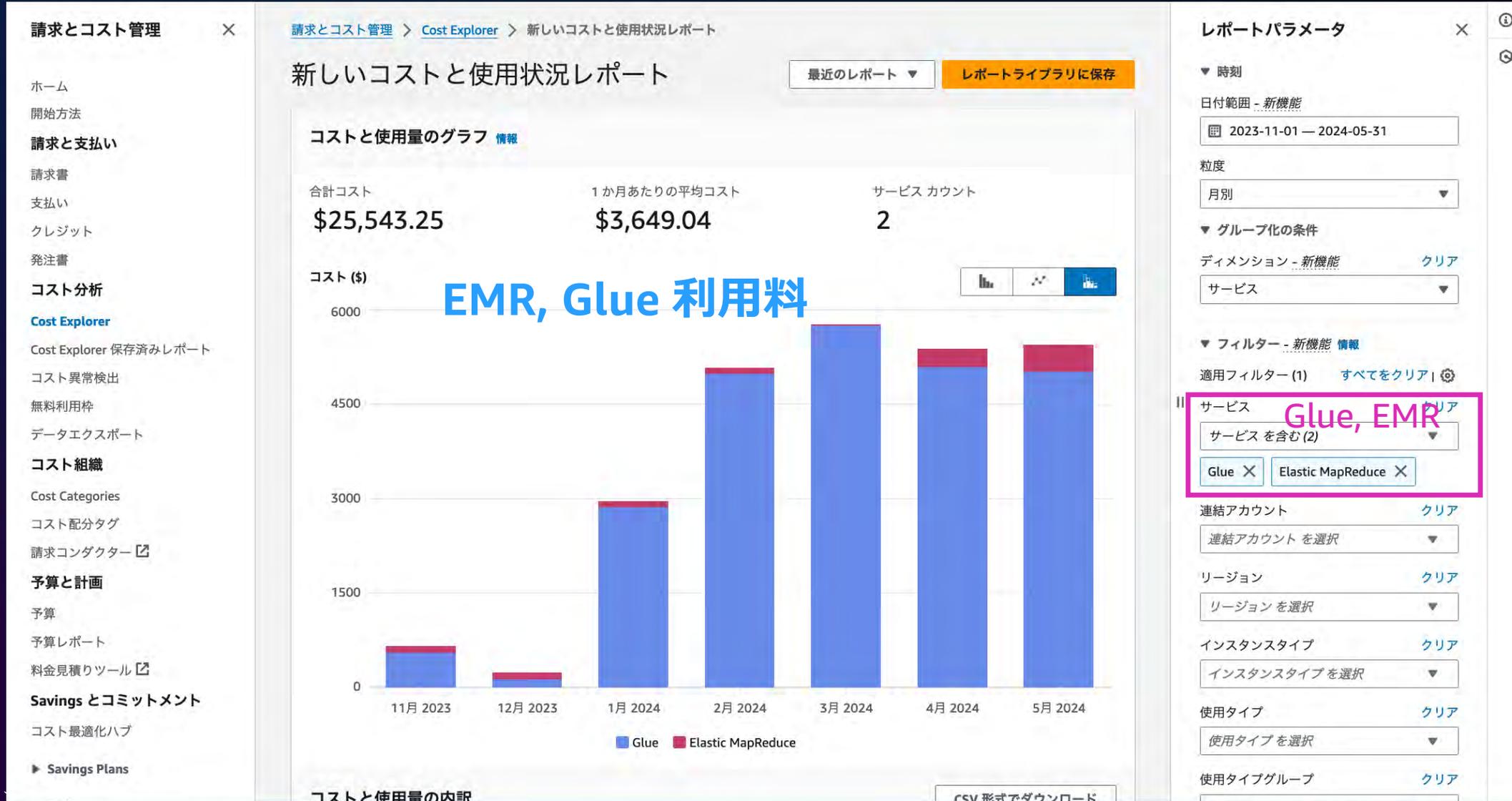
S3 ストレージ料金

- ap-northeast-1
- us-east-2

AWS Cost Explorer : データ転送



AWS Cost Explorer : データパイプライン



AWS Cost Explorer : クエリ



ストレージ



データ転送



データパイプライン



クエリ

コスト配分タグ

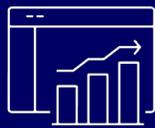
AWS リソースに設定されたタグごとにコスト配分を管理可能

The screenshot shows the AWS Cost Allocation Tags console. The left sidebar contains navigation options such as '請求とコスト管理', 'ホーム', '開始方法', '請求と支払い', '請求書', '支払い', 'クレジット', '発注書', 'コスト分析', 'Cost Explorer', 'Cost Explorer 保存済みレポート', 'コスト異常検出', '無料利用枠', 'データエクスポート', 'コスト組織', 'Cost Categories', 'コスト配分タグ', '請求コンダクター', '予算と計画', '予算', '予算レポート', and '料金見積りツール'. The main content area is titled 'コスト配分タグ' and shows '5個のコスト配分タグがアクティブ化されました'. There are buttons for 'バックフィルのタグ' and 'CSVのダウンロード'. Below, there are tabs for 'ユーザー定義のコスト配分タグ' and 'AWS生成コスト配分タグ'. The 'ユーザー定義のコスト配分タグ (90)' tab is active, showing a search bar with 'コスト配分タグを検索' and '3件の一致'. A filter 'ステータス = アクティブ' is applied. The table below lists the tags:

<input type="checkbox"/>	タグキー	▲	ステータス	▼	最終更新日	▼	最終使用月	▼
<input type="checkbox"/>	creator		🟢 アクティブ		June 11, 2024, 17:35 (UTC+09:00)		June 2024	
<input type="checkbox"/>	name		🟢 アクティブ		June 11, 2024, 17:34 (UTC+09:00)		June 2024	
<input type="checkbox"/>	owner		🟢 アクティブ		June 11, 2024, 17:34 (UTC+09:00)		June 2024	



Amazon S3 Storage Lens



Amazon S3 コンソール上のインタラクティブなダッシュボード



オブジェクトストレージの利用状況を組織全体で可視化



リージョンや、ストレージクラス、バケット、プレフィックスでドリルダウン



利用量とアクティビティに関する詳細なメトリクス

S3 Storage Lens

default-account-dashboard 情報

ダッシュボード設定を表示

削除

無効にする

2024/05/06

フィルター

このダッシュボードの範囲をさらに制限するには、一時フィルターを適用します。

- 概要
- アカウント
- AWS リージョン
- ストレージクラス
- バケット
- プレフィックス
- ストレージレンズグループ

2024/05/06 のスナップショット 情報

スナップショットは、頻繁に使用されるメトリクスの厳選されたリストです。ダッシュボードのグラフやテーブルで追加のメトリクスを表示できます。メトリクスの用語集が用意されています。

21.1 TB

ストレージの合計

87.5 M

オブジェクト数

259.3 KB

オブジェクトサイズの平均

126

アクティブなバケット

1

アカウント

-

すべてのリクエスト

メトリクスのカテゴリ

%(変更比較)

メトリクスのカテゴリを選択する

日/日

週/週

月/月

概要 X

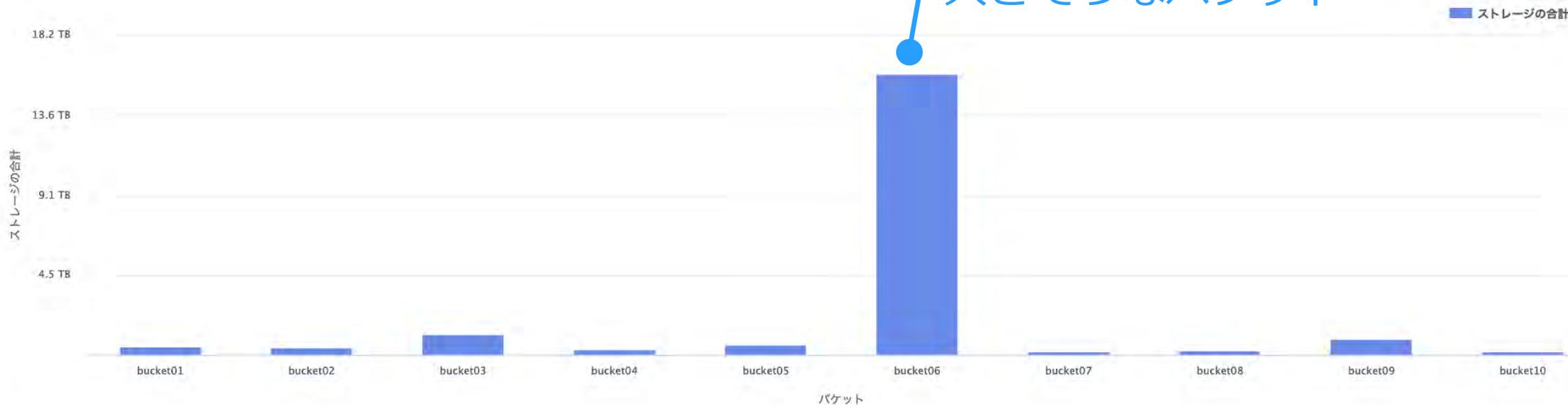
メトリクス名	メトリクスのカテゴリ	2024/05/06 の合計	%(変更)	30 日間の傾向	推奨事項
ストレージの合計	概要	21.1 TB	0.01%		コールアウト...
オブジェクト数	概要	87.5 M	0.04%		コールアウト...
オブジェクトサイズの平均	概要	259.3 KB	-0.02%		-
アクティブなバケット	概要	126	0%		-
アカウント	概要	1	0%		-

データサイズの大いバケットを特定

2024/05/06 のバケット別のディストリビューション

メトリクス

ストレージの合計



2024/05/06 のバケットによるバブル分析

平均オブジェクトサイズの小さいバケットの特定

2024/05/06 のバケットによるバブル分析



バケット (10)

メトリクスのエクスポートを設定することで、このダッシュボードでデータのエクスポートを表示できます。メトリクスの用語集が利用可能です。 [詳細はこちら](#)

メトリクスのカテゴリ

メトリクスのカテゴリを選択する

概要 X

🔍 バケットを検索

アクセス頻度の低いデータの特定



バケット (10)

[メトリクスのエクスポートを設定する](#)ことで、このダッシュボードでデータのエクスポートを表示できます。メトリクスの用語集が利用可能です。 [詳細はこちら](#)

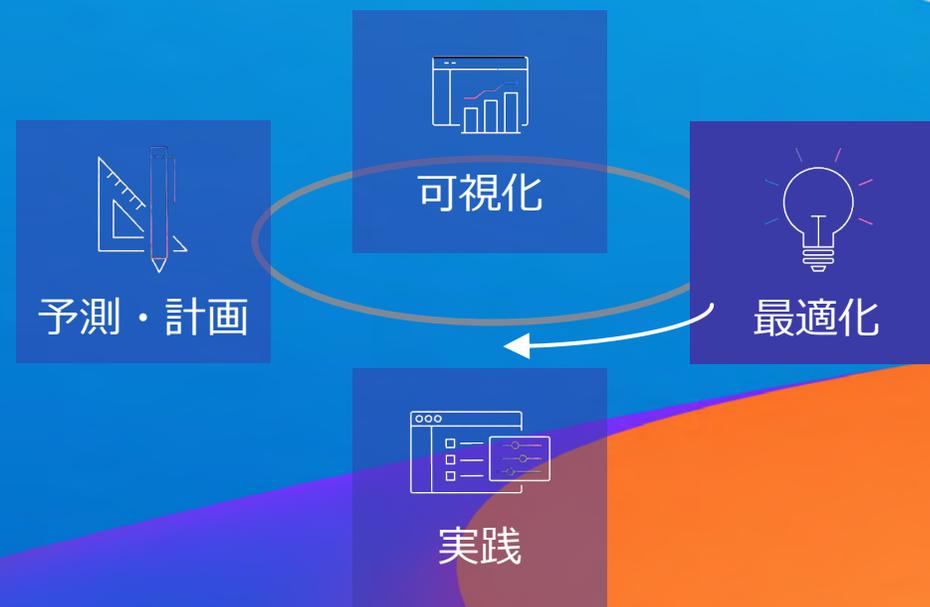
メトリクスのカテゴリ

メトリクスのカテゴリを選択する

概要 X

バケットを検索

コストの最適化



ストレージコストの最適化



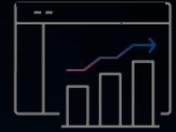
ストレージ



データ転送



データ
パイプライン



クエリ



データソース



データ転送

生データ



加工済みデータ



データマート



Amazon S3

データレイク



Amazon Athena



アドホッククエリ



Amazon EMR



AWS Glue



ダッシュボード

ストレージコストの最適化

課題

- ✗ 不適切なストレージクラス
- ✗ ライフサイクル管理の欠如
- ✗ 効率の悪いストレージ利用
- ✗ 不要な/一時的なデータの保持



最適化

- ✓ S3 ストレージクラスの活用
- ✓ S3 ライフサイクルの活用
- ✓ データの圧縮
- ✓ 不要なデータの自動削除



ストレージ



データ転送



データ
パイプライン



クエリ

ストレージコストの最適化

課題

- ✗ 不適切なストレージクラス
- ✗ ライフサイクル管理の欠如
- ✗ 効率の悪いストレージ利用
- ✗ 不要な/一時的なデータの保持



最適化

- ✓ S3 ストレージクラスの活用
- ✓ S3 ライフサイクルの活用
- ✓ データの圧縮
- ✓ 不要なデータの自動削除



ストレージ



データ転送



データ
パイプライン



クエリ

データ要件に適した S3 ストレージクラスの選択



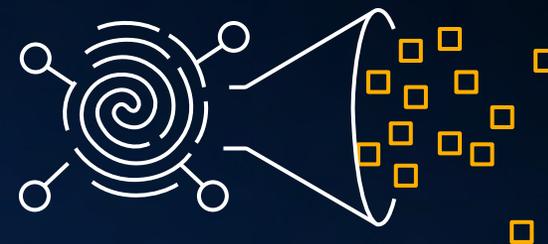
アクセス頻度

例：書き込みは一度
読み取りは多数
読み書きともに直近のデータ
へのアクセスが多い



保存期間

例：データレイクで直近5年分
のデータを保管・分析



性能

例：直近のデータはすぐに取り
得できる必要あり
1年以上前のデータの取得には
時間がかかってもよい

S3 ストレージクラスの選択

S3 Standard



頻繁にアクセス
するデータ

S3 Standard-IA



たまにアクセス
するデータ

S3 Glacier
Instant Retrieval



ごく稀にアクセス
するデータ

S3 Glacier
Flexible Retrieval



アーカイブする
データ

S3 Glacier
Deep Archive



長期間アーカイブ
するデータ

..... アクセス時間 : ミリ秒

..... アクセス時間 : 分・時間

ホット

コールド

Intelligent-Tiering – ストレージクラスの自動選択



ミリ秒アクセス(自動)

ストレージコストの最適化

課題

- ✗ 不適切なストレージクラス
- ✗ ライフサイクル管理の欠如
- ✗ 効率の悪いストレージ利用
- ✗ 不要な/一時的なデータの保持



最適化

- ✓ S3 ストレージクラスの活用
- ✓ S3 ライフサイクルの活用
- ✓ データの圧縮
- ✓ 不要なデータの自動削除



ストレージ



データ転送



データ
パイプライン



クエリ

データ要件に適したライフサイクル設定の例



S3 Standard

1年



S3 Glacier
Instant Retrieval

2年



S3 Glacier
Deep Archive

ストレージコストの最適化

課題

- ✗ 不適切なストレージクラス
- ✗ ライフサイクル管理の欠如
- ✗ 効率の悪いストレージ利用
- ✗ 不要な/一時的なデータの保持



最適化

- ✓ S3 ストレージクラスの活用
- ✓ S3 ライフサイクルの活用
- ✓ データの圧縮
- ✓ 不要なデータの自動削除



ストレージ



データ転送



データ
パイプライン



クエリ

ストレージコストの最適化

課題

- ✗ 不適切なストレージクラス
- ✗ ライフサイクル管理の欠如
- ✗ 効率の悪いストレージ利用
- ✗ 不要な/一時的なデータの保持



最適化

- ✓ S3 ストレージクラスの活用
- ✓ S3 ライフサイクルの活用
- ✓ データの圧縮
- ✓ 不要なデータの自動削除



ストレージ



データ転送



データ
パイプライン



クエリ

ストレージコストの最適化

課題

- ✗ 不適切なストレージクラス
- ✗ ライフサイクル管理の欠如
- ✗ 効率の悪いストレージ利用
- ✗ 不要な/一時的なデータの保持



最適化

- ✓ S3 ストレージクラスの活用
- ✓ S3 ライフサイクルの活用
- ✓ データの圧縮
- ✓ 不要なデータの自動削除



ストレージ



データ転送



データ
パイプライン



クエリ

データ転送コストの最適化



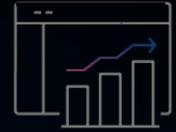
ストレージ



データ転送



データ
パイプライン



クエリ



データ転送コストの最適化

課題

- ✗ 非効率なデータ転送経路
- ✗ ローカリティの考慮不足
- ✗ 転送効率の悪いデータ転送
- ✗ 不要なデータレプリケーション



最適化

- ✓ データ転送経路の最適化
- ✓ ローカリティを考慮した設計
- ✓ 圧縮による転送効率の向上
- ✓ 必要最小限のレプリケーション



ストレージ



データ転送



データ
パイプライン



クエリ

データ転送コストの最適化

課題

- ✗ 非効率なデータ転送経路
- ✗ ローカリティの考慮不足
- ✗ 転送効率の悪いデータ転送
- ✗ 不要なデータレプリケーション



最適化

- ✓ データ転送経路の最適化
- ✓ ローカリティを考慮した設計
- ✓ 圧縮による転送効率の向上
- ✓ 必要最小限のレプリケーション



ストレージ



データ転送

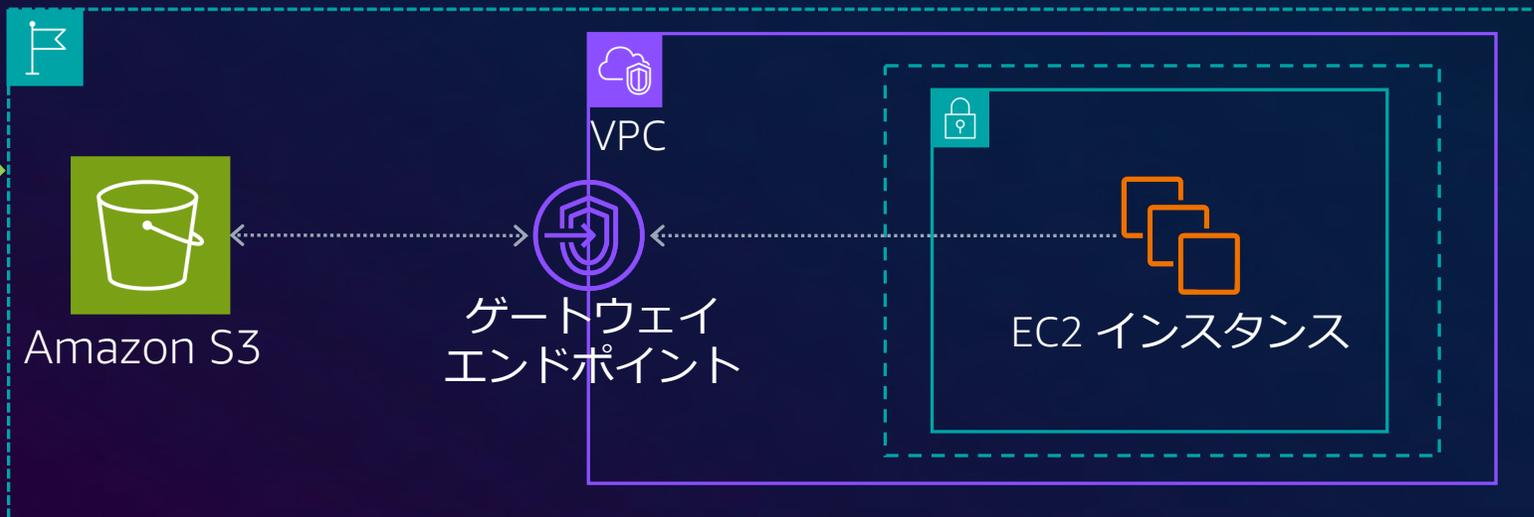
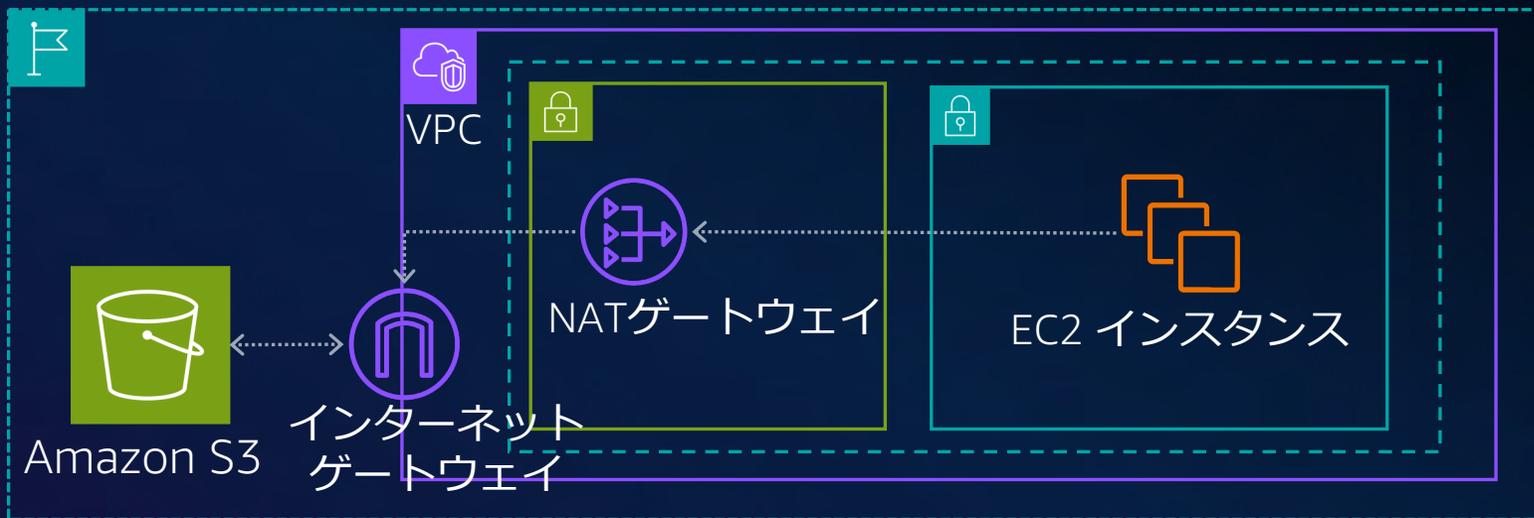


データ
パイプライン



クエリ

データ転送経路の最適化



不要なデータ転送を廃止

NATゲートウェイを削減

データ転送コストの最適化

課題

- ✗ 非効率なデータ転送経路
- ✗ ローカリティの考慮不足
- ✗ 転送効率の悪いデータ転送
- ✗ 不要なデータレプリケーション



最適化

- ✓ データ転送経路の最適化
- ✓ ローカリティを考慮した設計
- ✓ 圧縮による転送効率の向上
- ✓ 必要最小限のレプリケーション



ストレージ



データ転送



データ
パイプライン



クエリ

データ転送コストの最適化

課題

- ✗ 非効率なデータ転送経路
- ✗ ローカリティの考慮不足
- ✗ 転送効率の悪いデータ転送
- ✗ 不要なデータレプリケーション



最適化

- ✓ データ転送経路の最適化
- ✓ ローカリティを考慮した設計
- ✓ 圧縮による転送効率の向上
- ✓ 必要最小限のレプリケーション



ストレージ



データ転送



データ
パイプライン



クエリ

データ転送コストの最適化

課題

- ✗ 非効率なデータ転送経路
- ✗ ローカリティの考慮不足
- ✗ 転送効率の悪いデータ転送
- ✗ 不要なデータレプリケーション



最適化

- ✓ データ転送経路の最適化
- ✓ ローカリティを考慮した設計
- ✓ 圧縮による転送効率の向上
- ✓ 必要最小限のレプリケーション



ストレージ



データ転送



データ
パイプライン



クエリ

データ転送コストの最適化

課題

- ✗ 非効率なデータ転送経路
- ✗ ローカリティの考慮不足
- ✗ 転送効率の悪いデータ転送
- ✗ 不要なデータレプリケーション



最適化

- ✓ データ転送経路の最適化
- ✓ ローカリティを考慮した設計
- ✓ 圧縮による転送効率の向上
- ✓ 必要最小限のレプリケーション



ストレージ



データ転送



データ
パイプライン



クエリ

データパイプラインコストの最適化



ストレージ



データ転送



データ
パイプライン



クエリ



データソース

データ転送

生データ



加工済みデータ



データマート



データレイク



アドホッククエリ



ダッシュボード

データパイプラインコストの最適化

課題

- ✗ デフォルトのままの料金プラン
- ✗ 過剰な/過小なリソース
- ✗ アイドルリソース
- ✗ ボトルネックのあるジョブ
- ✗ 必要以上のデータの処理
- ✗ 重い変換や不要な集合演算
- ✗ 不正なデータへの無駄な処理



最適化

- ✓ 適切な料金プランの選択
- ✓ オートスケーリングの活用
- ✓ サーバーレスの活用
- ✓ ジョブのチューニング
- ✓ 処理対象データの削減
- ✓ 差分データ反映の最適化
- ✓ データクオリティ管理の導入



ストレージ



データ転送



データ
パイプライン



クエリ

データパイプラインコストの最適化

課題

- ✗ デフォルトのままの料金プラン
- ✗ 過剰な/過小なリソース
- ✗ アイドルリソース
- ✗ ボトルネックのあるジョブ
- ✗ 必要以上のデータの処理
- ✗ 重い変換や不要な集合演算
- ✗ 不正なデータへの無駄な処理



最適化

- ✓ 適切な料金プランの選択
- ✓ オートスケーリングの活用
- ✓ サーバーレスの活用
- ✓ ジョブのチューニング
- ✓ 処理対象データの削減
- ✓ 差分データ反映の最適化
- ✓ データクオリティ管理の導入



ストレージ



データ転送



データ
パイプライン

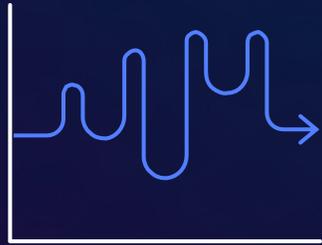


クエリ

適切な料金プランの選択

オンデマンド

長期コミットメントなしに
使った分だけの支払い



ステートフル・スパイク
のあるワークロード

Savings plans

1年間または3年間の
コミットメントにより
最大 **72%** のコスト削減



予測可能で安定した
ワークロード

スポット

余剰キャパシティにより
オンデマンドに比べて
最大 **90%** のコスト削減



ステートレスなワークロード

AWS Glue Flex Jobs



Standard

実行クラス

1分以内で起動
予測可能なレイテンシー

マイクロバッチや
低レイテンシワークロード



Flex

実行クラス

最大 **34%** コスト削減

一度きりのデータロード

データパイプラインコストの最適化

課題

- ✗ デフォルトのままの料金プラン
- ✗ 過剰な/過小なリソース
- ✗ アイドルリソース
- ✗ ボトルネックのあるジョブ
- ✗ 必要以上のデータの処理
- ✗ 重い変換や不要な集合演算
- ✗ 不正なデータへの無駄な処理

最適化

- ✓ 適切な料金プランの選択
- ✓ オートスケーリングの活用
- ✓ サーバーレスの活用
- ✓ ジョブのチューニング
- ✓ 処理対象データの削減
- ✓ 差分データ反映の最適化
- ✓ データクオリティ管理の導入



ストレージ



データ転送



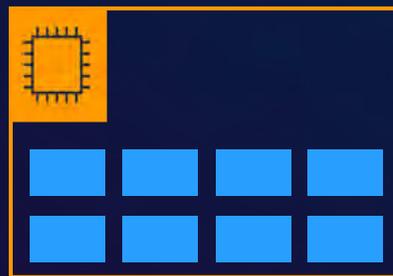
データ
パイプライン



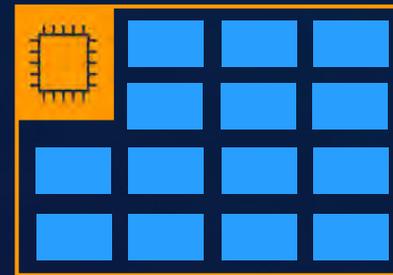
クエリ

Amazon EMR Managed Scaling

- クラスタ内のインスタンス数をワークロードに合わせて自動調整
- スポットインスタンスと併用
- 20%–60% のコスト削減

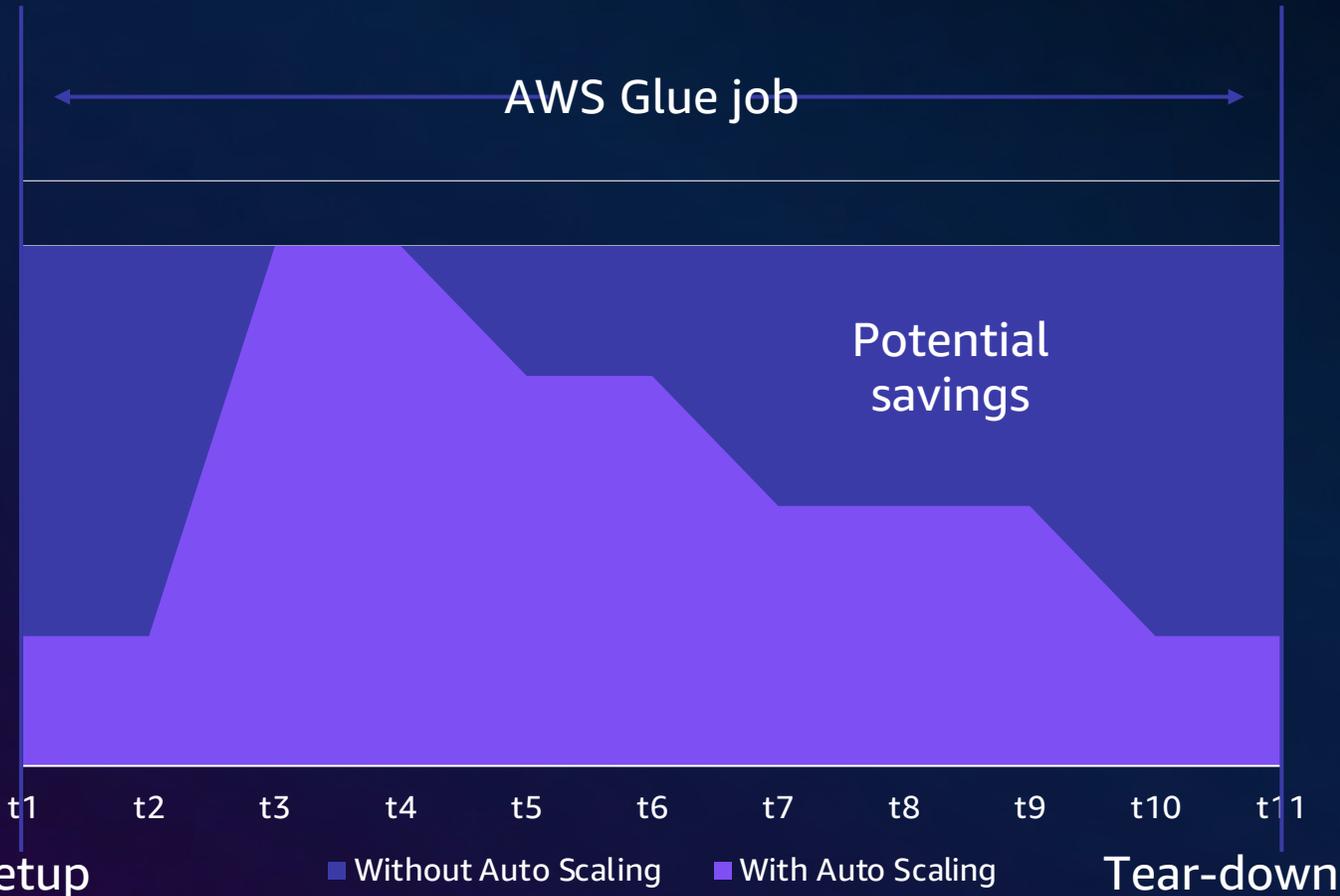


Cluster



Expanded cluster

AWS Glue Auto Scaling



コストを削減



自動で動作



シンプルな
キャパシティ計画

Cost = f(compute)

タイムライン

AWS Glue usage profiles

Glue リソースのコストをより柔軟に管理

- 開発者用、テスター用、分析者用など、ユーザーやロールに設定したプロファイルごとに Glue リソースのキャパシティを管理
- ワーカー数、ワーカータイプ、タイムアウト値のデフォルト値、最小値、最大値を設定
- Glue ジョブ、Glue Interactive Sessions に対応

データパイプラインコストの最適化

課題

- ✗ デフォルトのままの料金プラン
- ✗ 過剰な/過小なリソース
- ✗ アイドルリソース
- ✗ ボトルネックのあるジョブ
- ✗ 必要以上のデータの処理
- ✗ 重い変換や不要な集合演算
- ✗ 不正なデータへの無駄な処理

最適化

- ✓ 適切な料金プランの選択
- ✓ オートスケーリングの活用
- ✓ サーバーレスの活用
- ✓ ジョブのチューニング
- ✓ 処理対象データの削減
- ✓ 差分データ反映の最適化
- ✓ データクオリティ管理の導入



ストレージ



データ転送



データ
パイプライン



クエリ

データパイプラインコストの最適化

課題

- ✗ デフォルトのままの料金プラン
- ✗ 過剰な/過小なリソース
- ✗ アイドルリソース
- ✗ ボトルネックのあるジョブ
- ✗ 必要以上のデータの処理
- ✗ 重い変換や不要な集合演算
- ✗ 不正なデータへの無駄な処理

最適化

- ✓ 適切な料金プランの選択
- ✓ オートスケーリングの活用
- ✓ サーバーレスの活用
- ✓ ジョブのチューニング
- ✓ 処理対象データの削減
- ✓ 差分データ反映の最適化
- ✓ データクオリティ管理の導入



ストレージ



データ転送



データ
パイプライン



クエリ

Best practices for performance tuning AWS Glue for Apache Spark jobs

<https://docs.aws.amazon.com/prescriptive-guidance/latest/tuning-aws-glue-for-apache-spark/introduction.html>

AWS > Documentation > AWS Prescriptive Guidance > Best practices for performance tuning AWS Glue for Apache Spark jobs

Feedback Preferences

Best practices for performance tuning AWS Glue for Apache Spark jobs

Introduction

Key topics

Investigate performance issues

► Strategies for tuning performance

Resources

Document history

Glossary

Best practices for performance tuning AWS Glue for Apache Spark jobs

[PDF](#) | [RSS](#)

Roman Myers, Takashi Onikura, and Noritaka Sekiyama, Amazon Web Services (AWS)

December 2023 ([document history](#))

AWS Glue provides different options for tuning performance. This guide defines key topics for tuning AWS Glue for Apache Spark. It then provides a baseline strategy for you to follow when tuning these AWS Glue for Apache Spark jobs. Use this guide to learn how to identify performance problems by interpreting metrics available in AWS Glue. Then incorporate strategies to address these problems, maximizing performance and minimizing costs.

This guide covers the following tuning practices:

- [Scale cluster capacity](#)
- [Use the latest AWS Glue version](#)
- [Reduce the amount of data scan](#)
- [Parallelize tasks](#)
- [Minimize planning overhead](#)
- [Optimize shuffles](#)
- [Optimize user-defined functions](#)

Feedback

Share

Feedback

データパイプラインコストの最適化

課題

- ✗ デフォルトのままの料金プラン
- ✗ 過剰な/過小なリソース
- ✗ アイドルリソース
- ✗ ボトルネックのあるジョブ
- ✗ 必要以上のデータの処理
- ✗ 重い変換や不要な集合演算
- ✗ 不正なデータへの無駄な処理

最適化

- ✓ 適切な料金プランの選択
- ✓ オートスケーリングの活用
- ✓ サーバーレスの活用
- ✓ ジョブのチューニング
- ✓ 処理対象データの削減
- ✓ 差分データ反映の最適化
- ✓ データクオリティ管理の導入



ストレージ



データ転送



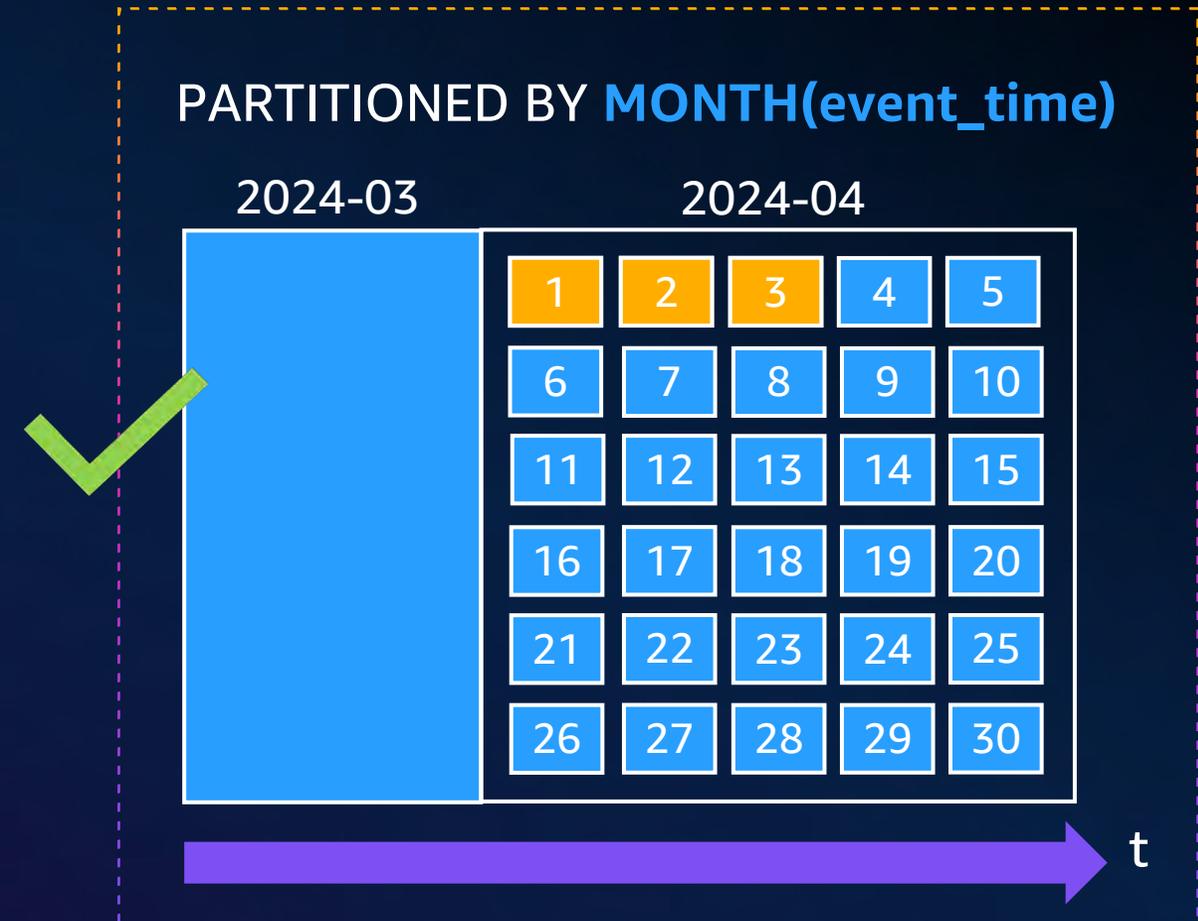
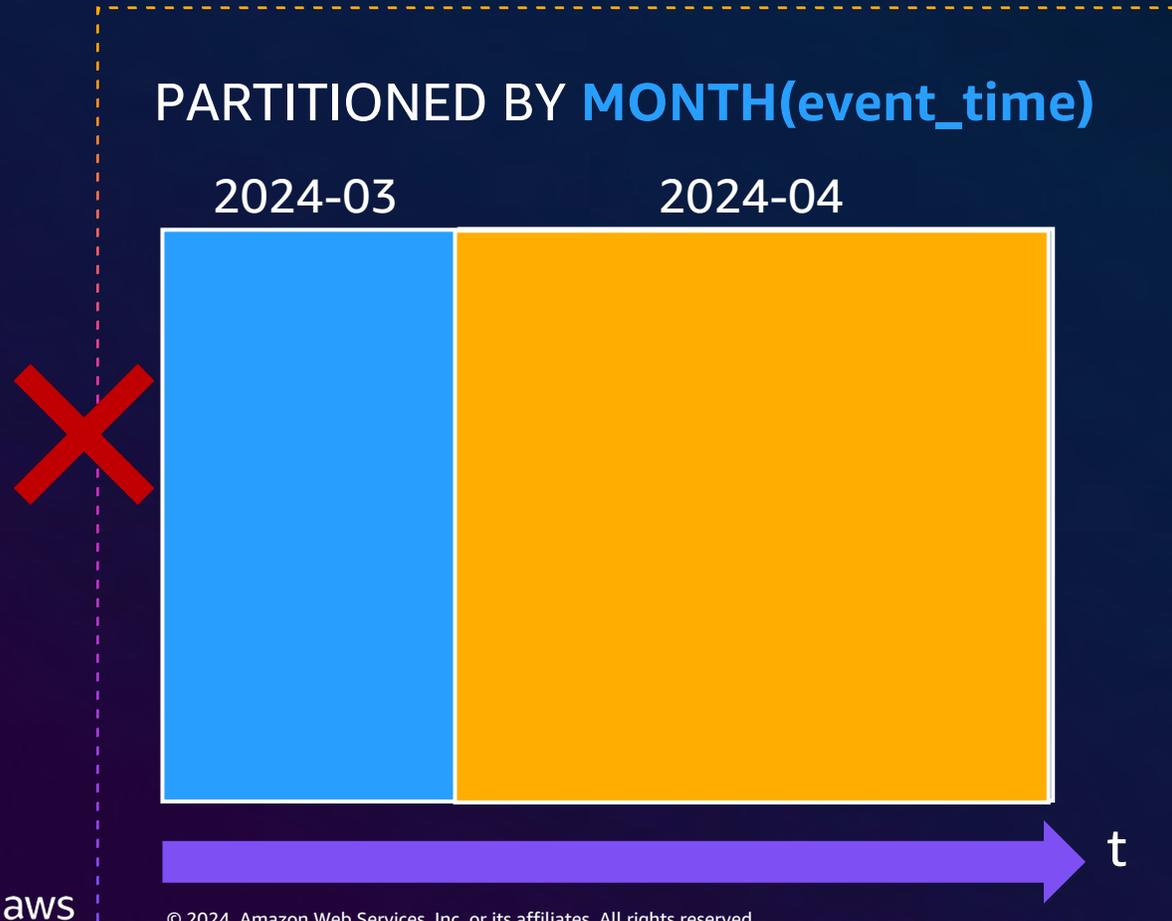
データ
パイプライン



クエリ

処理対象データの削減

- データの洗い替えによる更新対象範囲を必要最小限に



データパイプラインコストの最適化

課題

- ✗ デフォルトのままの料金プラン
- ✗ 過剰な/過小なリソース
- ✗ アイドルリソース
- ✗ ボトルネックのあるジョブ
- ✗ 必要以上のデータの処理
- ✗ 重い変換や不要な集合演算
- ✗ 不正なデータへの無駄な処理

最適化

- ✓ 適切な料金プランの選択
- ✓ オートスケーリングの活用
- ✓ サーバーレスの活用
- ✓ ジョブのチューニング
- ✓ 処理対象データの削減
- ✓ 差分データ反映の最適化
- ✓ データクオリティ管理の導入



ストレージ



データ転送



データ
パイプライン



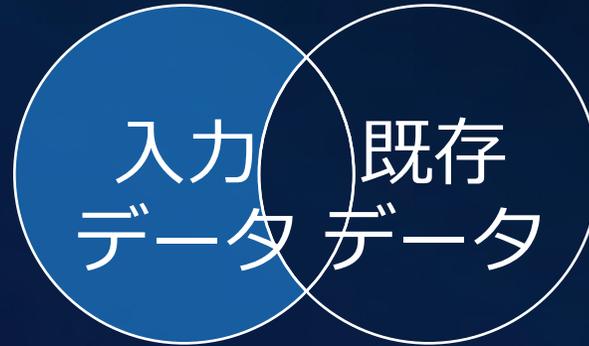
クエリ

差分データ反映の最適化：旧来の手法

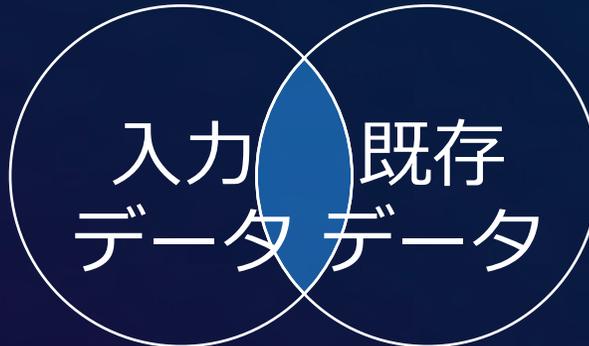
0. 初期状態



1. INSERT



2. UPDATE



差分データ反映の最適化：レコード単位の更新

データレイク



Apache Iceberg



Apache Hudi



Delta Lake

データウェアハウス



Amazon Redshift



差分データ反映の最適化 : MERGE INTO クエリ

Apache Spark / Apache Iceberg

```
MERGE INTO {catalog_name}.{database_name}.{table_name} AS t
USING (SELECT * FROM tmp_{table_name}_updates) AS u
ON t.product_id = u.product_id
```

```
WHEN MATCHED THEN UPDATE SET
```

```
    t.product_name = u.product_name,
```

```
    t.price = u.price,
```

```
    t.category = u.category,
```

```
    t.updated_at = u.updated_at
```

UPDATE

```
WHEN NOT MATCHED THEN INSERT *
```

INSERT



データパイプラインコストの最適化

課題

- ✗ デフォルトのままの料金プラン
- ✗ 過剰な/過小なリソース
- ✗ アイドルリソース
- ✗ ボトルネックのあるジョブ
- ✗ 必要以上のデータの処理
- ✗ 重い変換や不要な集合演算
- ✗ 不正なデータへの無駄な処理

最適化

- ✓ 適切な料金プランの選択
- ✓ オートスケーリングの活用
- ✓ サーバーレスの活用
- ✓ ジョブのチューニング
- ✓ 処理対象データの削減
- ✓ 差分データ反映の最適化
- ✓ データクオリティ管理の導入



ストレージ



データ転送



データ
パイプライン



クエリ

不正なデータへの無駄な処理

誤ったデータ、欠損したデータ等が混入



データソース

データ転送

生データ



加工済みデータ



データマート



Amazon S3

データレイク

Amazon EMR

AWS Glue

Amazon Athena

アドホッククエリ

ダッシュボード

無駄な処理

データクオリティ管理の導入

データ伝送路上で
データクオリティを検査



AWS Glue Data Quality

データレイクとパイプライン全体で
高品質なデータを保つ仕組み

- サーバーレス、スケーラブル、高速
- データ格納前（伝送路上）、データ格納後のデータ品質を検査
- 機械学習ベースで、データ品質の隠れた問題や以上を検知
- 様々なスキルセットのユーザーから利用可能

データパイプラインコストの最適化

課題

- ✗ デフォルトのままの料金プラン
- ✗ 過剰な/過小なリソース
- ✗ アイドルリソース
- ✗ ボトルネックのあるジョブ
- ✗ 必要以上のデータの処理
- ✗ 重い変換や不要な集合演算
- ✗ 不正なデータへの無駄な処理



最適化

- ✓ 適切な料金プランの選択
- ✓ オートスケーリングの活用
- ✓ サーバーレスの活用
- ✓ ジョブのチューニング
- ✓ 処理対象データの削減
- ✓ 差分データ反映の最適化
- ✓ データクオリティ管理の導入



ストレージ



データ転送



データ
パイプライン



クエリ

クエリコストの可視化



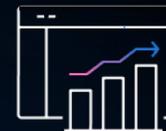
ストレージ



データ転送



データ
パイプライン



クエリ



クエリコストの最適化

課題

- ✗ 不要なデータスキャン
- ✗ 非効率的なデータ形式
- ✗ クエリ最適化の欠如
- ✗ 同一クエリの繰り返し実行
- ✗ 不要なデータ移動



最適化

- ✓ データレイアウトの最適化
- ✓ データ形式や圧縮形式の最適化
- ✓ クエリのチューニング
- ✓ クエリ結果の再利用
- ✓ フェデレーテッドクエリの活用



ストレージ



データ転送



データ
パイプライン



クエリ

Athena 料金プラン

Pay per query

\$5/TB

- 簡単に利用可能
- 実行したクエリのみに対して課金

Pay for compute

\$0.30/DPU-Hour

- 事前にキャパシティを予約
- 予約されたキャパシティに対して課金

1DPU: 4 vCPU & 16 GB RAM

i 両方の料金プランを同時に、同一アカウント内で併用可能

Athena キャパシティ予約

関連付けたワークグループに対して
コンピューティング能力を確保・利用

- スケールと同時実行数を管理
- 高優先度のクエリを予約されたコンピューティングで実行
- スキャン量ベースではなく、フラットレートで予測可能な料金

キャパシティ予約の作成

aws サービス 検索 [オプション+S]

Amazon Athena > キャパシティ予約 > 作成 キャパシティ予約

作成 キャパシティ予約 情報

キャパシティ予約の詳細

キャパシティ予約の名前
キャパシティ予約の一意の名前を入力します。予約の作成後に名前を変更することはできません。

キャパシティ予約の名前を入力

1~128文字を使用します。有効な文字は、a~z、A~Z、0~9、_(アンダースコア)、.(ピリオド)、-(ハイフン)です。

分析エンジン
キャパシティ予約は、Athena engine version 3 でサポートされています

Athena engine version 3

DPU 情報
リクエストする DPU の数を 4 個単位で入力します。

48

最小: 24 DPU

▶ タグ - オプション 情報
タグのキーと値は編集が可能で、いつでもキャパシティ予約からタグを削除できます。タグのキーと値では、大文字と小文字が区別されます。タグごとにタグキーは必須ですが、タグ値はオプションです。重複したタグキーを使用することはできません。

キャンセル レビュー

DPU 数を設定

ワークグループを選択

キャパシティを割り当て

The screenshot shows the Amazon Athena console interface. The left sidebar contains navigation options for Amazon Athena, including Query Editor, Notebooks, Jobs, Management, and Data Sources. The main content area displays the details for a capacity reservation named 'my_capacity_reservation'. Below this, there is a section for selecting workgroups, showing a list of available workgroups: 'AnalystGroup' and 'primary'. The 'AnalystGroup' workgroup is selected. The bottom of the console shows a metrics section and a navigation bar with timestamps and a 'ダッシュボードに追加' button.

aws サービス 検索 [オプション+S] パージニア北部 Username @ 1234-567

Amazon Athena X

クエリエディタ
ノートブックエディタ [新規](#)
ノートブックエクスプローラー [新規](#)

▼ ジョブ
ワークフロー
Step Functions を利用

▼ 管理
ワークグループ
キャパシティ予約 [新規](#)
データソース

最新情報 **9+**

コンパクトモードをオンにする

Amazon Athena > [キャパシティ予約](#) > my_capacity_reservation

my_capacity_reservation 削除する キャパシティ予約をキャンセル 編集

キャパシティ予約の詳細

キャパシティ予約の名前 my_capacity_reservation	ステータス 🟢 アクティブ	作成日 2024-06-11T16:02:55.776+09:00
アクティブなデータ処理ユニット (DPU) 48	ターゲット DPU 48	キャパシティ予約 ARN arn:aws:athena:us-east-1:970154250367:capacity-reservation/my_capacity_reservation

ワークグループ (2) ワークグループを削除 ワークグループを追加

クエリでキャパシティを使用できるワークグループを選択します。ワークグループはアクティブで、Athena engine version 3 を使用し、1つの予約にのみ存在する必要があります。

Q ワークグループの検索

<input type="checkbox"/>	名前	説明
<input type="checkbox"/>	AnalystGroup	A workgroup for analysts
<input type="checkbox"/>	primary	-

メトリクス タグ

メトリクス

以下のダッシュボードには、Amazon CloudWatch にプッシュされたキャパシティ予約割り当てと消費メトリクスが表示されます。ダッシュボードまたは CloudWatch コンソールを直接使用すると、予約に関する詳細なインサイトを得ることができます。

2024-06-11T07:02:55 > 2024-06-11T07:03:11 UTC タイムゾーン ダッシュボードに追加

クエリコストの最適化

課題

- ✗ 不要なデータスキャン
- ✗ 非効率的なデータ形式
- ✗ クエリ最適化の欠如
- ✗ 同一クエリの繰り返し実行
- ✗ 不要なデータ移動



最適化

- ✓ データレイアウトの最適化
- ✓ データ形式や圧縮形式の最適化
- ✓ クエリのチューニング
- ✓ クエリ結果の再利用
- ✓ フェデレーテッドクエリの活用



ストレージ



データ転送



データ
パイプライン



クエリ

クエリコストの最適化

課題

- ✗ 不要なデータスキャン
- ✗ 非効率的なデータ形式
- ✗ クエリ最適化の欠如
- ✗ 同一クエリの繰り返し実行
- ✗ 不要なデータ移動



最適化

- ✓ データレイアウトの最適化
- ✓ データ形式や圧縮形式の最適化
- ✓ クエリのチューニング
- ✓ クエリ結果の再利用
- ✓ フェデレーテッドクエリの活用



ストレージ



データ転送



データ
パイプライン



クエリ

クエリコストの最適化

課題

- ✗ 不要なデータスキャン
- ✗ 非効率的なデータ形式
- ✗ クエリ最適化の欠如
- ✗ 同一クエリの繰り返し実行
- ✗ 不要なデータ移動



最適化

- ✓ データレイアウトの最適化
- ✓ データ形式や圧縮形式の最適化
- ✓ クエリのチューニング
- ✓ クエリ結果の再利用
- ✓ フェデレーテッドクエリの活用



ストレージ



データ転送



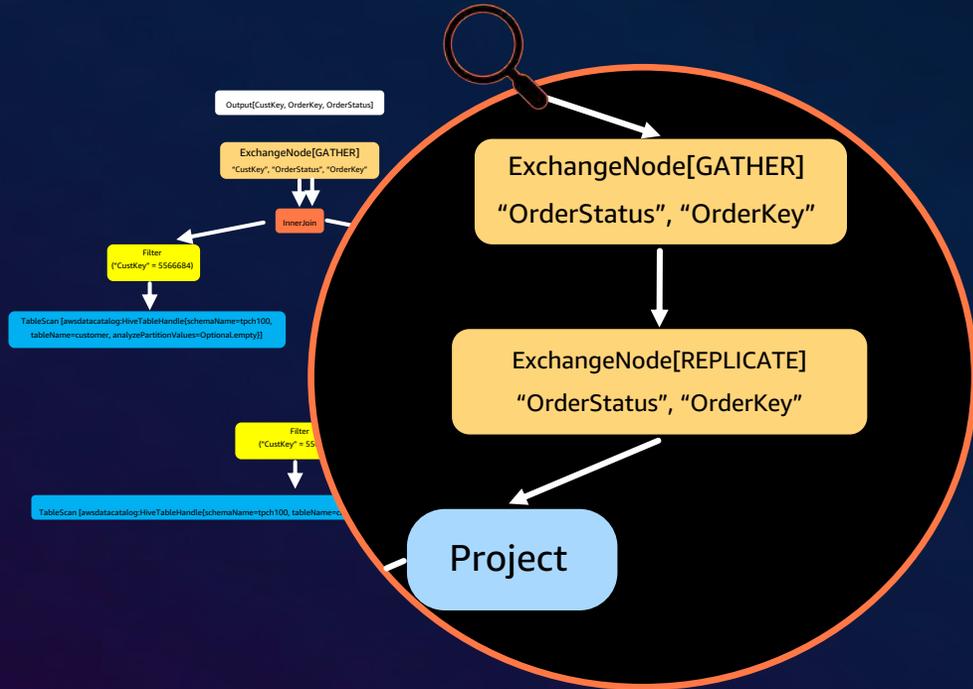
データ
パイプライン



クエリ

Athena クエリプラン

EXPLAIN [(option [, ...])] statement



EXPLAIN ANALYZE [(option [, ...])] statement

Fragment 1

```
CPU: 24.60ms, Input: 10 rows (1.48kB); per task: std.dev.: 0.00%
Output layout: [date, time, location, bytes, requestip, method]
- Limit[10] => [[date, time, location, bytes, requestip, method]]
  CPU: 1.00ms (0.03%), Output: 10 rows (1.48kB)
  Input avg.: 10.00 rows, Input std.dev.: 0.00%
- LocalExchange[SINGLE] () => [[date, time, location, bytes, requestip, method]]
  CPU: 0.00ns (0.00%), output: 10 rows (1.48kB)
  Input avg.: 0.63 rows, Input std.dev.: 387.30%
- RemoteSource[2] => [[date, time, location, bytes, requestip, method]]
  CPU: 1.00ms (0.03%), Output: 10 rows (1.48kB)
  Input avg.: 0.63 rows, Input std.dev.: 387.30%
```

JSON、テキスト、ヴィジュアルによるクエリプラン

JSON またはテキストによるクエリ詳細

Athena コストベース オプティマイザ

クエリプランを最適化しデータクエリを
ブースト

- AWS Glue Data Catalog のテーブル・
カラム統計情報を活用
- クエリ高速化のために最も効率の良い
クエリプランを選択
- デフォルトで有効

Top 10 Performance Tuning Tips for Amazon Athena

<https://aws.amazon.com/blogs/big-data/top-10-performance-tuning-tips-for-amazon-athena/>

AWS Big Data Blog

Top 10 Performance Tuning Tips for Amazon Athena

by Mert Hocanin and Pathik Shah | on 24 MAR 2017 | in [Amazon Athena](#), [AWS Big Data](#) | [Permalink](#) | [Comments](#) |

[Share](#)

February 2024: This post was reviewed and updated to reflect changes in Amazon Athena engine version 3, including cost-based optimization and query result reuse.

[Amazon Athena](#) is an interactive analytics service built on open source frameworks that make it straightforward to analyze data stored using open table and file formats in [Amazon Simple Storage Service](#) (Amazon S3) using standard SQL. Athena is serverless, so there is no infrastructure to manage, and you pay only for the queries that you run. Athena is easy to use, simply point to your data in Amazon S3, define the schema, and start querying using standard SQL.

In this post, we review the top 10 tips that can improve query performance. We focus on aspects related to storing data in Amazon S3 and tuning specific to queries.

Storage

This section discusses how to structure your data so that you can get the most out of Athena. You can apply the same practices to [Amazon EMR](#) data processing applications such as Spark, Trino, Presto, and Hive when your data is stored in Amazon S3. We discuss the following best practices:

1. Partition your data
2. Bucket your data
3. Use compression
4. Optimize file size
5. Use columnar file formats

1. Partition your data

Partitioning divides your table into parts and keeps the related data together based on column values such as date, country, and region. Partitions act as virtual columns. You define them at table creation, and they can help reduce the amount of data scanned per query, thereby improving performance. You can restrict the amount of data scanned by a

Query tuning

The Athena SQL engine is built on the open source distributed query engines [Trino](#) and [Presto](#). Understanding how it works provides insight into how you can optimize queries when running them. This section details the following best practices:

1. Optimize `ORDER BY`
2. Optimize joins
3. Optimize `GROUP BY`
4. Use approximate functions
5. Only include the columns that you need

6. Optimize ORDER BY

The `ORDER BY` clause returns the results of a query in sort order. Athena uses distributed sort to run the sort operation in parallel on multiple nodes. If you're using the `ORDER BY` clause to look at the top or bottom N values, use a `LIMIT` clause to reduce the cost of the sort, which results in a faster query runtime.

For example, the following table summarizes the runtime for a dataset with a 7.25 GB table, uncompressed in text format, with approximately 60 million rows.

Query	Runtime
<pre>SELECT * FROM lineitem ORDER BY l_shipdate</pre>	274 seconds
<pre>SELECT * FROM lineitem ORDER BY l_shipdate LIMIT 10000</pre>	4.6 seconds

クエリコストの最適化

課題

- ✗ 不要なデータスキャン
- ✗ 非効率的なデータ形式
- ✗ クエリ最適化の欠如
- ✗ 同一クエリの繰り返し実行
- ✗ 不要なデータ移動



最適化

- ✓ データレイアウトの最適化
- ✓ データ形式や圧縮形式の最適化
- ✓ クエリのチューニング
- ✓ クエリ結果の再利用
- ✓ フェデレーテッドクエリの活用



ストレージ



データ転送



データ
パイプライン



クエリ

Athena クエリ結果の再利用

同一クエリの処理時間の改善

データスキャンコストの削減

ワークグループ単位の結果共有

5x

繰り返し実行されるクエリ
を最大5倍高速に実行

\$0

データスキャンなし
で無料

Amazon Athena > Query editor

Editor | Recent queries | Saved queries | Settings | Workgroup: primary

Query 1 :
1 select dayofweek, sum(cancelled) as cancelled
2 from flight_delay_parquet
3 group by 1 order by 1

SQL Ln 1, Col 1

Run again | Explain | Cancel | Clear | Create

Reuse query results
up to 60 minutes ago

Query results | Query stats

Completed Time in queue: 296 ms Run time: 3.261 sec Data scanned: 51.72 MB

Results (7) Copy Download results

Search rows

dayofweek	cancelled
1	462645
2	496085
3	479074
4	465589
5	444138
6	332599
7	352093

クエリ結果の再利用

Athena クエリ結果の再利用

The screenshot displays the Amazon Athena console interface. At the top, there's a navigation bar with the AWS logo, service menu, search bar, and user information. Below this, the 'Amazon Athena' console is open to the 'クエリエディタ' (Query Editor) page. The page has tabs for 'エディタ' (Editor), '最近のクエリ' (Recent Queries), '保存したクエリ' (Saved Queries), and '設定' (Settings). The 'エディタ' tab is active, showing a SQL query in a text area. Below the query, there are buttons for 'もう一度実行する' (Run Again), 'Explain', 'キャンセル' (Cancel), 'クリア' (Clear), and '作成' (Create). A checkbox labeled 'クエリ結果を再利用する' (Reuse Query Results) is checked, with a note '60分前まで' (Up to 60 minutes ago). Below the query editor, there are tabs for 'クエリ結果' (Query Results) and 'クエリの統計' (Query Statistics). The 'クエリ結果' tab is active, showing a green status bar '完了済み' (Completed) and a summary box containing 'キュー内の時間: 110 ms', '実行時間: 12.482 sec', and 'スキャンしたデータ: 26.66 GB'. Below this, there are buttons for 'コピー' (Copy) and '結果をダウンロード' (Download Results). A search bar '行を検索' (Search rows) is present. The main area shows a table with 137 results, with the first two rows visible:

#	avg_duration	avg_passenger_count	avg_trip_distance	avg_total_amount	year	month	type
1	0 00:00:15.593	1.6818984440891576	11.829075784983798	0.2997212561384697	2015	6	yellow
2	0 00:00:12.819	1.7033087704049275	2.9627206074935546	14.687306263606944	2012	9	yellow

クエリ結果
再利用
の有効化

Athena クエリ結果の再利用

aws サービス 検索 [オプション+S] バージニア北部 Username @ 1234-5678-9012

Amazon Athena > クエリエディタ

エディタ 最近のクエリ 保存したクエリ 設定

ワークグループ DataZone 環境 primary

```
1 SELECT avg(dropoff_datetime - pickup_datetime )/60 as avg_duration
2     , avg(passenger_count) as avg_passenger_count
3     , avg(trip_distance) as avg_trip_distance
4     , avg(total_amount) as avg_total_amount
5     , year
6     , month
7     , type
8 FROM "nyctaxi"."records"
9 WHERE dropoff_datetime is not null
10 GROUP BY year, month, type
```

SQL Ln 10, Col 27

もう一度実行する Explain キャンセル クリア 作成

クエリ結果を再利用する
60分前まで

クエリ結果 クエリの統計

完了済み - 再利用されたクエリ結果の使用

キュー内の時間: 81 ms 実行時間: 181 ms スキャンしたデータ: -

結果 (100+) コピー 結果をダウンロード

行を検索

#	avg_duration	avg_passenger_count	avg_trip_distance	avg_total_amount	year	month	type
1	0 00:00:15.593	1.6818984440891576	11.829075784983798	0.2997212561384697	2015	6	yellow
2	0 00:00:12.819	1.7033087704049275	2.9627206074935546	14.687306263606944	2012	9	yellow

Athena クエリ結果の再利用

aws サービス 検索 [オプション+S] バージニア北部 Username @ 1234-5678-9012

Amazon Athena > クエリエディタ

エディタ 最近のクエリ 保存したクエリ 設定

ワークグループ DataZone 環境 primary

最近のクエリ (500+)

最近のクエリを検索

実行 ID	クエリ	開始時刻	ステータス	実行時間	キャッシュ
fc0f70de-3043-47db-8a0f-110a6c0734c4	SELECT avg(dropoff_datetime - pickup_datetime)/60 as avg_...	2024-06-11T15:58:25.292+09...	完了済み	181 ms	結果の再利用
45f9fa35-d0af-428d-8c9a-8824720f2cc1	SELECT avg(dropoff_datetime - pickup_datetime)/60 as avg_...	2024-06-11T15:54:59.656+09...	完了済み	12.482 sec	-
c54228be-32ce-4f6d-9ea2-df020aa2c89c	SELECT avg(dropoff_datetime - pickup_datetime)/60 as avg_...	2024-06-11T15:54:00.952+09...	完了済み	10.599 sec	-
e148f124-9141-4f11-b423-7573b7f290b6	SELECT avg((CAST(dropoff_datetime as BIGINT) - CAST(pickup_...	2024-06-11T15:53:23.663+09...	失敗	257 ms	-
26a1a8ff-4d59-435c-be97-54ff7e2aa3bd	SELECT avg((CAST(dropoff_datetime as LONG) - CAST(pickup_...	2024-06-11T15:52:58.792+09...	失敗	188 ms	-
be731991-3138-4c2e-aa7c-a2c14b4c0fb1	SELECT avg((CAST(dropoff_datetime as LONG) - CAST(pickup_...	2024-06-11T15:52:28.048+09...	失敗	175 ms	-
de99e2b8-f3a3-4f51-8db9-14cbecf82722	SELECT type, count(*) FROM "nyctaxi"."records" GROUP BY type	2024-06-11T15:50:56.461+09...	完了済み	3.312 sec	-
1abf035c-c200-4844-b472-8fda0b5bd556	SELECT * FROM "nyctaxi"."records" limit 10	2024-06-11T15:50:25.607+09...	完了済み	1.513 sec	-
2309a451-960f-44da-bf59-ae6e55fc9709	SELECT * FROM "dbt_glue_demo_nyc_metrics"."gold_passenge...	2024-06-11T15:48:37.219+09...	完了済み	1.495 sec	-
4ebd96a6-0310-4498-9c6d-2c1f4eccb58e	/* QuickSight 5802d365-a240-4fee-96ac-98c70d1da3ee */ S...	2024-06-11T15:14:14.365+09...	完了済み	4.655 sec	-
f5014293-1fad-49c8-8589-6cf71bd899f9	/* QuickSight 6d7fea96-c858-489e-b4af-10a93c85a3b4 */ SE...	2024-06-11T14:14:14.692+09...	完了済み	5.058 sec	-
06cefdd26-3021-42be-88a3-a6294c5f5648	/* QuickSight 44ac233f-da75-4887-a076-9f2f547245c4 */ SE...	2024-06-11T13:14:14.722+09...	完了済み	5.623 sec	-
cee5c97b-b8ff-4e7a-b6a1-db39b11535cd	/* QuickSight 651b7891-8f02-40b9-a09d-87d568d1050e */ ...	2024-06-11T12:14:15.629+09...	完了済み	5.894 sec	-
c55ea7dc-d0d4-4d6e-8d74-d55d20900cdc	/* QuickSight a48d1192-0833-49f6-b35c-65c471ebc54f */ S...	2024-06-11T11:14:15.699+09...	完了済み	5.102 sec	-

キャッシュ
状態

クエリコストの最適化

課題

- ✗ 不要なデータスキャン
- ✗ 非効率的なデータ形式
- ✗ クエリ最適化の欠如
- ✗ 同一クエリの繰り返し実行
- ✗ 不要なデータ移動



最適化

- ✓ データレイアウトの最適化
- ✓ データ形式や圧縮形式の最適化
- ✓ クエリのチューニング
- ✓ クエリ結果の再利用
- ✓ フェデレーテッドクエリの活用



ストレージ



データ転送



データ
パイプライン



クエリ

クエリコストの最適化

課題

- ✗ 不要なデータスキャン
- ✗ 非効率的なデータ形式
- ✗ クエリ最適化の欠如
- ✗ 同一クエリの繰り返し実行
- ✗ 不要なデータ移動



最適化

- ✓ データレイアウトの最適化
- ✓ データ形式や圧縮形式の最適化
- ✓ クエリのチューニング
- ✓ クエリ結果の再利用
- ✓ フェデレーテッドクエリの活用



ストレージ



データ転送



データ
パイプライン



クエリ

おわりに



おわりに



ストレージ



データ転送



データ
パイプライン



クエリ

コスト最適化のジャーニー



Thank you!

関山宜孝

sekiyama@amazon.com

