aws

ユーザーガイド

Amazon Neptune

Amazon Neptune: ユーザーガイド

Table of Contents

Amazon Neptune とは

Amazon Neptune は、高速で信頼性に優れたフルマネージド型のグラフデータベースサービスで、高度に接続されたデータセットを使用するアプリケーションの構築と実行を容易にします。Neptune の中核は、専用のハイパフォーマンスなグラフデータベース エンジンです。このエンジンは、数十億の関係を保存し、ミリ秒単位のレイテンシーでグラフをクエリできるよう最適化されています。Neptune は、人気の高いプロパティグラフクエリ言語 Apache TinkerPop Gremlin と Neo4j の OpenCypher、および W3C の RDF クエリ言語 SPARQL をサポートしています。これにより、高度に接続されたデータセットを効率的にナビゲートするクエリを構築できます。Neptune は、推奨エンジン、不正検出、知識グラフ、創薬、ネットワークセキュリティなどのグラフのユースケースを強化します。

Neptune データベースは、可用性が高く、リードレプリカ、ポイントインタイムリカバリ、Amazon S3 への継続的なバックアップ、およびアベイラビリティーゾーン間のレプリケーションに対応しています。Neptune はデータセキュリティ機能を提供し、保管時および伝送中の暗号化をサポートします。Neptune はフルマネージド型であるため、ハードウェアプロビジョニング、ソフトウェアパッチ適用、セットアップ、構成、バックアップなどのデータベース管理タスクについて頭を悩ます必要はありません。

Neptune Analytics は、Neptune データベースを補完する分析データベースエンジンであり、メモリ内の大量のグラフデータをすばやく分析して、インサイトを得たり傾向を確認したりできます。Neptune Analytics は、データレイクに保存されている既存のグラフデータベースやグラフデータセットを迅速に分析するためのソリューションです。一般的なグラフ分析アルゴリズムと低レイテンシーの分析クエリを使用します。

Amazon Neptune の詳細については、まず以下のセクションを参照することをお勧めします。

- Amazon Neptune の開始方法
- Amazon Neptune の機能の概要

グラフを初めて使用する場合、または Neptune の完全本番環境に投資する準備がまだできていない場合は、費用をかけずに学習と開発に Neptune Jupyter ノートブックを使用する方法がわかる はじめに トピックをご覧ください。

また、データベースの設計を開始する前に、GitHub リポジトリ グラフデータベースを使用するための AWS リファレンスアーキテクチャ を参照することをお勧めします。ここでは、グラフデータモデルとクエリ言語の選択内容を知らせたり、参照配置アーキテクチャの例を参照したりできます。

主なサービスコンポーネント

- プライマリ DB インスタンス - 読み書きオペレーションをサポートし、クラスターボリュームに対するすべてのデータ変更を実行します。各 Neptune DB クラスターには、グラフデータベースの内容の書き込み (つまりロードまたは変更) を担当する 1 つのプライマリ DB インスタンスがあります。
- Neptune レプリカ – プライマリ DB インスタンスと同じストレージボリュームに接続し、読み取りオペレーションのみをサポートします。各 Neptune DB クラスターは、プライマリ DB インスタンスに加えて 15 Neptune までのレプリカを持つことができます。これにより、Neptune レプリカを別々のアベイラビリティーゾーンに配置し、読み込みクライアントからの負荷を分散することで高可用性を実現します。
- クラスターボリューム – Neptune データはクラスターボリュームに保存されます。このボリュームは、信頼性と高可用性を重視して設計されています。クラスターボリュームは、単一 AWS リージョンの複数のアベイラビリティーゾーン間のデータのコピーで構成されます。データはアベイラビリティーゾーン間で自動的にレプリケートされるため、データ損失の可能性は非常に低く、耐久性は非常に高くなります。

オープングラフ API をサポート

Amazon Neptune では、プロパティグラフ (Gremlin と openCypher) と RDF グラフ (SPARQL) の両方に対してオープングラフ API をサポートしています。そのグラフモデルとクエリ言語両方に優れたパフォーマンスを提供します。プロパティグラフ (PG) モデルを選択し、openCypher クエリ言語と Gremlin クエリ言語の両方で同じグラフにアクセスできます。W3C 標準のリソース記述フレームワーク (RDF) モデルを使用する場合は、標準 SPARQL クエリ言語を使用してグラフにアクセスできます。

高い安全性

Neptune では、データベースのために複数のレベルのセキュリティが用意されています。セキュリティ機能には、Amazon VPC を使用したネットワーク分離、および AWS Key Management Service (AWS KMS) で作成して管理するキーを使用した保管時の暗号化があります。暗号化された Neptune インスタンスでは、基盤となるストレージのデータが暗号化されます。さらに、同じクラスター内にある自動化バックアップ、スナップショット、レプリカも暗号化されます。

完全マネージド型

Amazon Neptune では、ハードウェアのプロビジョニング、ソフトウェアのパッチ適用、セットアップ、設定またはバックアップなどのデータベース管理タスクについて頭を悩ます必要はなくなります。

Neptune を使用して、数十億の関係に対してミリ秒単位でクエリを実行する、洗練されたインタラクティブなグラフアプリケーションを作成できます。高度に接続されたデータの SQL クエリは複雑で、パフォーマンスの調整は困難です。Neptune では、人気のあるグラフクエリ言語である Gremlin、openCypher、および SPARQL を使用して、書き込みが容易で、接続されたデータを高パフォーマンスで処理する強力なクエリを実行できます。この機能により、コードの複雑さが大幅に軽減されるため、関係を処理するアプリケーションを迅速に作成できます。

Neptune は、99.99 パーセントの可能性を提供するように設計されています。データベースワークロード用に構築された SSD ベースの仮想化ストレージレイヤーとデータベースエンジンを完全に統合することで、データベースのパフォーマンスおよび可用性を向上します。耐障害性と自己修復機能を備えた Neptune ストレージ ディスク障害が発生した場合は、データベースの可用性を低下させることなくバックグラウンドで修復されます。Neptune は、データベースのクラッシュを自動的に検出して再起動します。クラッシュ復旧やデータベースキャッシュの再構築は必要ありません。インスタンス全体に障害が発生した場合、Neptune は最大 15 個のリードレプリカの 1 つに自動的にフェイルオーバーします。

Amazon Neptune の変更と更新

以下のテーブルは、Amazon Neptune の重要な変更点をまとめたものです。

変更	説明	日付
エンジンバージョン 1.3.2.1	2024-06-20 の時点で、エンジンバージョン 1.3.2.1 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.3.2.1」を参照してください。	2024 年 6 月 20 日
エンジンバージョン 1.3.2.0	2024-06-10 の時点で、エンジンバージョン 1.3.2.0 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.3.2.0」を参照してください。	2024 年 6 月 10 日
エンジンバージョン 1.2.1.1	2024-03-11 の時点で、エンジンバージョン 1.2.1.1 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリ	2024 年 3 月 11 日

	リース 1.2.1.1」を参照してください。	
エンジンバージョン 1.3.1.0	2024-03-06 の時点で、エンジンバージョン 1.3.1.0 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.3.1.0」を参照してください。	2024 年 3 月 6 日
AWS マネージドポリシーのアクセス許可の更新	`NeptuneReadOnlyAccess` および `NeptuneFullAccess` マネージドポリシーでは、ポリシーステートメントに識別子として `Sid` (ステートメント ID) が含まれるようになりました。	2024 年 1 月 22 日
Neptune が I/O 最適化ストレージを提供開始	I/O 最適化ストレージの場合は、使用したストレージとインスタンスに対して料金を支払います。ストレージコストは標準ストレージよりも高くなりますが、使用した I/O に対する料金は発生しません。	2023 年 12 月 13 日

Neptune の IAM マネージドポリシーの変更	NeptuneConsoleFullAccess IAM 管理ポリシーが更新され、Neptune Analytics グラフを操作するために必要なアクセス許可が付与されました。新しいNeptuneGraphReadOnlyアクセスポリシーが追加され、Neptune Analytics グラフリソースへの読み取り専用アクセスが可能になりました。また、新しい`AWSServiceRoleForNeptuneGraphPolicy` ポリシーが追加され、Neptune Analytics グラフが CloudWatch 運用および使用状況のメトリクスとログを公開できるようになりました。	2023 年 11 月 29 日
エンジンバージョン 1.3.0.0	2023 年 11 月 15 日現在、エンジンバージョン 1.3.0.0 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.3.0.0」を参照してください。	2023 年 11 月 15 日
Neptune がイスラエル (テルアビブ) リージョンで利用可能に	Amazon Neptune がイスラエル (テルアビブ) (`il-central-1`) リージョンで利用可能になりました。	2023 年 11 月 13 日

Neptune プロパティグラフ time-to-live での の実装に関するブログ記事	Melissa Kwok、Mike Havey、Kevin Phillips による「Amazon Neptune での有効期限の実装、パート 1: プロパティグラフ」を参照してください。	2023 年 10 月 27 日
エンジンバージョン 1.2.1.0.R7	2023 年 10 月 6 日現在、エンジンバージョン 1.2.1.0.R7 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.2.1.0.R7」を参照してください。	2023 年 10 月 6 日
エンジンバージョン 1.2.0.0.R4	2023 年 9 月 29 日現在、エンジンバージョン1.2.0.0.R4 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.2.0.0.R4」を参照してください。	2023 年 9 月 29 日

エンジンバージョン 1.2.0.1.R3	2023 年 9 月 27 日現在、エンジンバージョン 1.2.0.1.R3 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.2.0.1.R3」を参照してください。	2023 年 9 月 27 日
エンジンバージョン 1.2.1.0.R6	2023 年 9 月 12 日現在、エンジンバージョン 1.2.1.0.R6 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.2.1.0.R6」を参照してください。	2023 年 9 月 12 日
エンジンバージョン 1.2.0.2.R6	2023 年 9 月 12 日現在、エンジンバージョン 1.2.0.2.R6 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.2.0.2.R6」を参照してください。	2023 年 9 月 12 日

Neptune エンジンのアップグレードにおけるブルー/グリーンデプロイ戦略の使用に関するブログ記事	Ankit Gupta と Abhishek Mishra による「ブルー/グリーンデプロイを使用したエンジンアップグレード時の Amazon Neptune の可用性の向上」を参照してください。	2023 年 9 月 11 日
エンジンバージョン 1.2.1.0.R5	2023 年 9 月 2 日現在、エンジンバージョン 1.2.1.0.R5 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.2.1.0.R5」を参照してください。	2023 年 9 月 2 日
エンジンバージョン 1.2.0.2.R5	2023 年 8 月 16 日現在、エンジンバージョン 1.2.0.2.R5 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.2.0.2.R5」を参照してください。	2023 年 8 月 16 日

エンジンバージョン 1.2.1.0.R4	2023 年 8 月 10 日現在、エンジンバージョン 1.2.1.0.R4 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.2.1.0.R4」を参照してください。	2023 年 8 月 10 日
Neptune エンジンリリース 1.2.1.0 に関するブログ記事	Joy Wang、Kevin Phillips、Andrea Nassisi、Navtanay Sinha による「Amazon Neptune の機能満載 1.2.1.0 リリースを探る」を参照してください。	2023 年 8 月 4 日
Neptune を使用したマルチモーダルデータベースソリューションの作成に関するブログ記事	Mike Havey による「Amazon Neptune を使用したユースケース主導型で拡張性の高いマルチモデルデータベースソリューションの設計」を参照してください。	2023 年 7 月 18 日
Neptune サーバーレスのユースケースとベストプラクティスに関するブログ記事	Kevin Phillips と Ankit Gupta による「Amazon Neptune サーバーレスでコストとパフォーマンスを最適化するためのユースケースとベストプラクティス」を参照してください。	2023 年 6 月 28 日

エンジンバージョン 1.2.1.0.R3	2023 年 6 月 13 日現在、エンジンバージョン 1.2.1.0.R3 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.2.1.0.R3」を参照してください。	2023 年 6 月 13 日
レジャー提案のリアルタイムでの生成に関するブログ記事	Michael Meidlinger と Nils Müller による「Amazon Neptune を使用してレジャー活動の提案をリアルタイムで生成する」を参照してください。	2023 年 6 月 6 日
Neptune と RDKit による分子モデリングに関するブログ記事	Graham Kutchek による「Amazon Neptune と RDkit による分子 SMILES データのモデル化」を参照してください。	2023 年 6 月 1 日
Neptune のパフォーマンスを向上させるためのキャッシュの使用に関するブログ記事 (パート 3)	Taylor Riggan、Abhishek Mishra、Melissa Kwok、および Kelvin Lawrence による「Amazon Neptune でのキャッシュによるグラフクエリパフォーマンスの高速化」、パート 3: Amazon を使用した Neptune クラスター全体のキャッシュアーキテクチャ ElastiCache」を参照してください。	2023 年 5 月 26 日

Neptune のパフォーマンスを向上させるためのキャッシュの使用に関するブログ記事 (パート 2)	Taylor Riggan、Abhishek Mishra、Melissa Kwok、Kelvin Lawrence による「Amazon Neptune のキャッシュによるグラフクエリのパフォーマンスの向上、パート 2: その他の Neptune キャッシュ機能」を参照してください。	2023 年 5 月 26 日
Neptune のパフォーマンスを向上させるためのキャッシュの使用に関するブログ記事 (パート 1)	Taylor Riggan、Abhishek Mishra、Melissa Kwok、Kelvin Lawrence による「Amazon Neptune のキャッシュによるグラフクエリのパフォーマンスの向上、パート 1: クエリとバッファプールキャッシュ」を参照してください。	2023 年 5 月 26 日
Neptune を使用したサプライチェーン分析に関するブログ記事	Dhiraj Thakur と Rajdip Chaudhur による「Amazon Neptune と Neptune ワークベンチをを使用したサプライチェーンデータの分析と視覚化」を参照してください。	2023 年 5 月 10 日
エンジンバージョン 1.2.0.2.R4	2023 年 5 月 8 日現在、エンジンバージョン 1.2.0.2.R4 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.2.0.2.R4」を参照してください。	2023 年 5 月 8 日

Neptune が中東 (UAE) リージョンで発売開始	Amazon Neptune が、中東 (UAE) (`me-central-1`) リージョンで利用可能になりました。	2023 年 5 月 2 日
エンジンバージョン 1.2.1.0.R2	2023 年 5 月 2 日現在、エンジンバージョン 1.2.1.0.R2 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.2.1.0.R2」を参照してください。	2023 年 5 月 2 日
Media2Cloud を使用し、AI によるビデオ分析を使用した Neptune でのナレッジグラフの構築に関するブログ記事	Mike Havey による「Media2Cloud を使用し、AI によるビデオ分析を使用した Amazon Neptune でのナレッジグラフの作成」を参照してください。	2023 年 5 月 2 日
が Neptune を使用して脆弱性修復プラットフォーム DevOcean を構築した方法に関するブログ記事	Gil Makmel と チャールズ・イビエによるAmazon Neptune を使用してクラウドネイティブアプリケーション用の脆弱性修復管理プラットフォーム DevOcean を構築した方法」を参照してください。	2023 年 4 月 25 日

Getir が Neptune を使用して不正検知システムを構築する方法に関するブログ記事	Berkay Berkman、Mahmut Turan、Mutlu Polatcan、Umut Cemal Kıraç、Yağız Yanıkoğlu、Esra Kayabali による「Getir が Amazon Neptune と Amazon DynamoDB を使用して包括的な不正検出システムを構築する方法」を参照してください。	2023 年 4 月 6 日
Wiz が Neptune を使用してクラウドセキュリティを再考する方法に関するブログ記事	Ami Luttwak と Brad Bebee による「世界はグラフ: Wiz が Amazon Neptune のグラフを使用してクラウドセキュリティを再考する方法」をご覧ください。	2023 年 3 月 31 日
エンジンバージョン 1.2.0.2.R3	2023 年 3 月 27 日現在、エンジンバージョン 1.2.0.2.R3 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.2.0.2.R3」を参照してください。	2023 年 3 月 27 日
CSC Generation が Neptune を使用して製品発見を促進する方法に関するブログ記事	Bobber Cheng、Ronit Rudra、Melissa Kwok による「CSC Generation が Amazon Neptune を使用してナレッジグラフで製品発見を促進する方法」をご覧ください。	2023 年 3 月 21 日

エンジンバージョン 1.2.1.0	2023 年 3 月 8 日現在、エンジンバージョン 1.2.1.0 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.2.1.0」を参照してください。	2023 年 3 月 8 日
Neptune での TinkerPop 3.6.x の新機能の探索に関するブログ記事	Stephen Mallette による「Amazon Neptune での Apache TinkerPop 3.6.x の新機能の探索」を参照してください。	2023 年 3 月 8 日
セマンティック推論を使用して RDF グラフから新しい事実を推測することについてのブログ記事	Charles Ivie と Diana Marks による「RDFox を Amazon Neptune に統合することで、セマンティック推論を使用して RDF グラフから新しい事実を推測する」を参照してください。	2023 年 2 月 20 日
Neptune による医療 FHIR データの分析に関するブログ記事	Alena Schmickl による「Amazon Neptune による医療 FHIR データの分析」を参照してください。	2023 年 2 月 13 日
Neptune ML を使用したリアルタイム不正検知ソリューションの構築に関するブログ記事	Hua Shu と Soji AdeshinaHua による「Amazon Neptune ML を使用したリアルタイム不正検知ソリューションの構築」を参照してください。	2023 年 2 月 8 日

エンジンバージョン 1.1.1.0.R7	2023 年 1 月 23 日現在、エンジンバージョン1.1.1.0.R7は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.1.1.0.R7」を参照してください。	2023 年 1 月 23 日
グラフエクスプローラーがリリースされました	グラフエクスプローラーは、グラフデータを可視化するためのオープンソースのフロントエンド Web アプリケーションツールです。https://github.com/aws/graph-explorer を参照してください。	2023 年 1 月 3 日
メンテナンスリリースバージョン 1.1.0.0.R3	2022 年 12 月 23 日現在、エンジンバージョン 1.1.0.0.R3 メンテナンスリリースは一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.1.0.0.R3」を参照してください。	2022 年 12 月 23 日

Neptune ワークベンチが Amazon Linux 2 および JupyterLab 3 で動作するようになりました。	Neptune グラフノートブックは、 JupyterLab 3 の Amazon Linux 2 環境で実行されるようになりました。この新しい環境に移行する方法については、「Neptune ノートブックを Jupyter から JupyterLab 3 に移行する」を参照してください。	2022 年 12 月 21 日
IMDb ナレッジグラフを使用した電力の推奨と検索に関するブログ記事 (パート 3)	Divya Bhargavi、Soji Adeshina、Gaurav Rele、Karan Sindwani、Vidya Sagar Ravipati、Matthew Rhodes による「IMDb ナレッジグラフを使用した電力の推奨と検索 — パート 3」を参照してください。	2022 年 12 月 20 日
IMDb ナレッジグラフを使用した電力の推奨と検索に関するブログ記事 (パート 2)	Matthew Rhodes、Soji Adeshina、Divya Bhargavi、Gaurav Rele、Karan Sindwani、Vidya Sagar Ravipati による「IMDb ナレッジグラフを使用した電力の推奨と検索 — パート 2」を参照してください。	2022 年 12 月 20 日
IMDb ナレッジグラフを使用した電力の推奨と検索に関するブログ記事 (パート 1)	Gaurav Rele、Soji Adeshina、Divya Bhargavi、Karan Sindwani、Vidya Sagar Ravipati、Matthew Rhodes による「IMDb ナレッジグラフを使用した電力の推奨と検索 — パート 1」を参照してください。	2022 年 12 月 20 日

Neptune サーバーレスが新しい AWS リージョンで利用可能になりました	2022 年 12 月 16 日現在、Neptune サーバーレスは、新しい AWS リージョン、すなわち、カナダ (中部)、欧州 (ストックホルム)、欧州 (フランクフルト)、アジアパシフィック (シンガポール)、アジアパシフィック (シドニー)で使用可能になりました。Neptune サーバーレスが利用可能なすべてのリージョンについては、「Amazon Neptune サーバーレスの制約」を参照してください。	2022 年 12 月 16 日
エンジンバージョン 1.2.0.2.R2	2022 年 12 月 15 日現在、エンジンバージョン 1.2.0.2.R2 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.2.0.2.R2」を参照してください。	2022 年 12 月 15 日
エンジンバージョン 1.2.0.0.R3	2022 年 12 月 15 日現在、エンジンバージョン 1.2.0.0.R3 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.2.0.0.R3」を参照してください。	2022 年 12 月 15 日

エンジンバージョン 1.2.0.1.R2	2022 年 12 月 13 日現在、エンジンバージョン 1.2.0.1.R2 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.2.0.1.R2」を参照してください。	2022 年 12 月 13 日
を使用した教育用ビッグデータ分析アーキテクチャの設計に関するブログ記事 AWS	Lavanya Sood による「AWS を使用した教育用ビッグデータ分析アーキテクチャの設計」を参照してください。	2022 年 12 月 13 日
グラフデータベースによる学習の強化に関するブログ記事	Lavanya Sood による「グラフデータベースによる学習の強化」を参照してください。	2022 年 12 月 8 日
AWS Glue を使用した Neptune への RDF データのロードに関するブログ記事	Mike Havey と Fabrizio Napolitano による「 AWS Glue を使用して RDF データを Amazon Neptune にロードする」を参照してください。	2022 年 11 月 23 日
エンジンバージョン 1.2.0.2	2022 年 11 月 20 日現在、エンジンバージョン 1.2.0.2 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.2.0.2」を参照してください。	2022 年 11 月 20 日

エンジンバージョン 1.2.0.1	2022 年 10 月 26 日現在、エンジンバージョン1.2.0.1. は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.2.0.1」を参照してください。	2022 年 10 月 26 日
Neptune を使用した不正検知に関するブログ記事	Wilson Tang、Amr Elnaggar、Matias Pons、Mohammad Azzam、Saurabh Deshpande、Luis Rodrigues Soares による「Amazon Neptune での Delivery Hero における不正検知機能の強化」を参照してください。	2022 年 10 月 26 日
Neptune サーバーレスに関するブログ記事	Danilo Poccia による「Amazon Neptune サーバーレスの紹介 — ワークロードに合わせて容量を調整するフルマネージド型のグラフデータベース」を参照してください。	2022 年 10 月 26 日
Lambda と SPARQL UPDATE LOAD を使用した Neptune へのイベント駆動型 RDF インポートに関するブログ記事	John Walker、Onno Buijs、チャールズ・イビエ、Javy de Koning による「NXP が AWS Lambda と SPARQL UPDATE LOAD を使用して Amazon Neptune にイベント駆動型の RDF インポートを実行する方法」を参照してください。	2022 年 10 月 20 日

エンジンバージョン 1.2.0.0.R2	2022 年 10 月 14 日現在、エンジンバージョン 1.2.0.0.R2 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.2.0.0.R2」を参照してください。	2022 年 10 月 14 日
Neptune での多言語テキストプロパティのエンコードに関するブログ記事	Jiani Zhang による「Amazon Neptune での多言語テキストプロパティのエンコーディングによる予測モデルのトレーニング」を参照してください。	2022 年 10 月 14 日
Neptune データアクセスの自動テストに関するブログ記事	Greg Biegel による「Apache TinkerPop Gremlin による Amazon Neptune データアクセスの自動テスト」を参照してください。	2022 年 9 月 28 日
エンジンバージョン 1.1.1.0.R6	2022 年 9 月 23 日現在、エンジンバージョン1.1.1.0.R6 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.1.1.0.R6」を参照してください。	2022 年 9 月 23 日

Informatica® による Neptune の使用方法に関するブログ記事	Tiju Titus John、Deepak Ram、Farooq Ashraf による「ナレッジグラフについて、Informatica® クラウドデータガバナンスとカタログは Amazon Neptune をどのように使用しているか」をご覧ください。	2022 年 9 月 20 日
Neptune と Tom Sawyer Perspectives を使用して金融詐欺を暴くことについてのブログ記事	Tom Sawyer Software のシニアプロダクトマネージャー、Janet M. Six による「Amazon Neptune と Tom Sawyer Perspectives で金融詐欺を暴く」を参照してください。	2022 年 8 月 30 日
、Neptune SageMaker、DGL を使用した GNN ベースのリアルタイム不正検出ソリューションの構築に関するブログ記事	Jian Zhang、Haozhu Wang SageMaker、Mengxin Zhu による「Amazon Amazon Neptune、 Deep Graph Library を使用して GNN ベースのリアルタイム不正検出ソリューションを構築する」を参照してください。	2022 年 8 月 11 日
リソースタグを使用して Neptune 環境リソースを停止および起動することについてのブログ記事	Kevin Phillips による「リソースタグを使用して Amazon Neptune 環境リソースの停止と起動を自動化する」を参照してください。	2022 年 8 月 1 日

Apache に貢献しているアーティストに関するブログ記事 TinkerPop	スティーブン・マレットとケトリーナ・トンプソンによる「Beyond Code: The Artist Who Contributes to Apache TinkerPop」を参照してください。	2022 年 8 月 1 日
Neptune データプレーンアクションのきめ細かいアクセスコントロールに関するブログ記事	Abhishek Mishra と Ankit Gupta による「Amazon Neptune データプレーンアクションのきめ細かいアクセスコントロール」をご覧ください。	2022 年 7 月 29 日
Neptune グローバルデータベースに関するブログ記事	Navtanay Sinha による「Amazon Neptune グローバルデータベースの紹介」を参照してください。	2022 年 7 月 27 日
エンジンバージョン 1.2.0.0	2022 年 7 月 21 日現在、エンジンバージョン 1.2.0.0 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.2.0.0」を参照してください。	2022 年 7 月 21 日

エンジンバージョン 1.1.1.0.R5	2022 年 7 月 21 日現在、エンジンバージョン1.1.1.0.R5は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.1.1.0.R5」を参照してください。	2022 年 7 月 21 日
エンジンバージョン 1.1.1.0.R4	2022 年 6 月 23 日現在、エンジンバージョン1.1.1.0.R4は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.1.1.0.R4」を参照してください。	2022 年 6 月 23 日
Python 統合によるグラフ分析と機械学習のワークフローの簡素化	データサイエンスと ML ワークフローを簡素化するオープンソースの Python 統合を使用して、Amazon Neptune に保存されているグラフデータに対してグラフ分析と機械学習タスクを実行できるようになりました。「Neptune のAWS Data Wrangler ドキュメント」を参照してください。	2022 年 6 月 7 日

エンジンバージョン 1.1.1.0.R3	2022 年 6 月 7 日現在、エンジンバージョン1.1.1.0.R3 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.1.1.0.R3」を参照してください。	2022 年 6 月 7 日
フェイクニュースの検出に関するブログ記事	Hasan Shojaei と Sarita Joshi による「Amazon Neptune ML によるグラフ機械学習を使用したソーシャルメディアのフェイクニュースの検出」を参照してください。	2022 年 5 月 19 日
Neptune での SQL Server Integration Services (SSIS) の使用に関するブログ記事	Mesgana Gormley と Melissa Kwok による「SQL Server Integration Services (SSIS) と Amazon Neptune を使用してデータから新しいインサイトを発見する」を参照してください。	2022 年 5 月 18 日
メンテナンスリリースバージョン 1.1.0.0.R2	2022 年 5 月 16 日現在、エンジンバージョン 1.1.1.0.R2 メンテナンスリリースは一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.1.1.0.R2」を参照してください。	2022 年 5 月 16 日

メンテナンスリリースバージョン 1.1.0.0.R2	2022 年 5 月 16 日現在、エンジンバージョン 1.1.0.0.R2 メンテナンスリリースは一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.1.0.0.R2」を参照してください。	2022 年 5 月 16 日
メンテナンスリリースバージョン 1.0.5.1.R4	2022 年 5 月 16 日現在、エンジンバージョン 1.0.5.1.R4 メンテナンスリリースは一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.0.5.1.R4」を参照してください。	2022 年 5 月 16 日
メンテナンスリリースバージョン 1.0.5.0.R5	2022 年 5 月 16 日現在、エンジンバージョン 1.0.5.0.R5 メンテナンスリリースは一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.0.5.0.R5」を参照してください。	2022 年 5 月 16 日

Neptune での openCypher の一般提供に関するブログ記事	Navtanay Sinha と Dave Bechberger による「Amazon Neptune の openCypher サポートの一般提供の発表」を参照してください。	2022 年 4 月 22 日
エンジンバージョン 1.1.1.0	2022 年 4 月 19 日現在、エンジンバージョン 1.1.1.0 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.1.1.0」を参照してください。	2022 年 4 月 19 日
データ系統に関するブログ記事	Khoa Nguyen、Krithivasan Balasubramaniyan、Rahul Shaurya による「 AWS Glue、Amazon Neptune、Spline を使用してデータレイクのデータ系統を構築する」を参照してください。	2022 年 4 月 1 日
エンジンリリース 1.1.0.0 へのアップグレードが再び有効になりました	2022 年 2 月 21 日現在、エンジンリリース 1.1.0.0 へのアップグレードは一時的に無効になりました。現在は再び有効になっています。	2022 年 3 月 22 日
エンジニアリングユーティリティの信頼性に関するブログ記事	Abhineet Parchure による「データに基づくクラウドベースの電力システムモデルを使用したユーティリティの信頼性設計」を参照してください。	2022 年 3 月 22 日

Neptune がアフリカ (ケープタウン) で提供開始	Amazon Neptune が アフリカ (ケープタウン)(af-south-1) で利用可能になりました。ただし、そのリージョンでは、Neptune ワークベンチノートブックのサポートは Neptune コンソールでは一時的に無効になっています。	2022 年 2 月 24 日
OWL を使ったモデル駆動型グラフに関するブログ記事	Mike Havey による 「Amazon Neptune での OWL を使用したモデル駆動型グラフ」を参照してください。	2022 年 2 月 23 日
Rhizomer によるセマンティックナレッジグラフの探索に関するブログ記事	Roberto Garcia による 「Amazon Neptune と Rhizomer を使用して SPARQL を使用せずにセマンティックナレッジグラフを探索する」を参照してください。	2022 年 2 月 22 日
ユーティリティグリッドのグラフ化に関するブログ記事	Bobby Wilson と Joseph Beer による「 のユーティリティグリッドのグラフ AWS化」を参照してください。	2022 年 2 月 18 日
新しい Neptune ML テキスト機能のエンコーディングオプション	Neptune は、トレーニング用に FastText および Sentence BERT テキストエンコーディングをサポートするようになりました。FastText Neptune ML の機能および Neptune ML の Sentence BERT の機能を参照してください。	2022 年 2 月 15 日

Neptune OpenSearch で を使用する地理空間クエリに関するブログ記事	Ross Gabay と Abhilash Vinod による地理空間クエリについては、Amazon Neptune と Amazon OpenSearch Service を組み合わせる」を参照してください。	2022 年 2 月 1 日
Amazon EKS と Neptune を使用した金融犯罪発見に関するブログ記事	Severin Gassauer-Fleissner と Zahi Ben Shabat による「Amazon EKS とグラフデータベースを使用した金融犯罪の発見」を参照してください。	2022 年 2 月 1 日
Neptune クラスターボリュームは、最大 128 テビバイト (TiB) まで増やすことができるようになりました	中国と を除くサポートされているすべてのリージョンで GovCloud、Neptune クラスターボリュームのサイズ制限が 64 TiB から 128 TB に増加しました。これは、リリース 1.0.2.2 から始まるすべてのエンジンリリースに適用されます。「Amazon Neptune ストレージ」のページを参照してください。	2022 年 2 月 1 日
Neptune 全文検索が のすべてのバージョンと統合されるようになりました OpenSearch。	「Amazon Service を使用した Amazon Neptune でのフルテキスト検索」を参照してください OpenSearch 。	2022 年 1 月 28 日
Docker コンテナを使用してグラフノートブックをデプロイすることに関するブログ記事	Ganesh Sawhney と Qiang Zhang による「Docker コンテナを使用して にグラフノートブックをデプロイ AWSする」を参照してください。	2022 年 1 月 22 日

エンジンバージョン 1.0.5.1.R3	2022 年 1 月 13 日現在、エンジンバージョン 1.0.5.1.R3 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、「Neptune エンジンリリース 1.0.5.1.R3」を参照してください。	2022 年 1 月 13 日
Neptune ML を使用したグラフベースのレコメンデーションシステムに関するブログ記事	Yanwei Cui と Will Badr による「Neptune ML によるグラフベースのレコメンデーションシステム: ソーシャルネットワークのリンク予測の課題に関する図」を参照してください。	2022 年 1 月 12 日
Neptune の自動スケーリングに関するブログ記事	Navtanay Sinha と Sudhanshu Gupta による「ワークロードの需要を満たすための Amazon Neptune データベースの自動スケーリング」を参照してください。	2021 年 11 月 29 日
インタラクティブなグラフデータ分析と可視化に関するブログ記事	Sandeep Veldi と Abhishek Mishra による Amazon Neptune 、Amazon Athena 横串検索、Amazon を使用してインタラクティブなグラフデータ分析とビジュ QuickSight アライゼーションを構築する」を参照してください。	2021 年 11 月 24 日

グラフベースの深層学習を使用した ID 詐欺の検出に関するブログ記事	Kevin O'Brien、Kamran Habib、Will Badr による「Careem がグラフベースの深層学習と Amazon Neptune を使用して ID 詐欺を検出する方法」を参照してください。	2021 年 11 月 23 日
エンジンバージョン 1.1.0.0	2021 年 11 月 19 日現在、エンジンバージョン 1.1.0.0. は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、Neptune エンジンリリース 1.1.0.0を参照してください。	2021 年 11 月 19 日
データ保護とコンプライアンスの一元化に関するブログ記事	Brian O'Keefe によるAWS「バックアップ による Amazon Neptune でのデータ保護とコンプライアンスの一元化」を参照してください。	2021 年 11 月 8 日
詐欺や不適切な支払いとの闘いに関するブログ記事	Vladi Royzman と Spencer Smith による「連邦支出規模での詐欺や不適切な支払いとのリアルタイムの戦い」を参照してください。	2021 年 11 月 2 日
偽アカウント登録の防止に関するブログ記事	Anjan Biswas による「Amazon Fraud Detector を使用した AI による偽アカウント登録のリアルタイム防止」を参照してください。	2021 年 10 月 29 日

エンジンバージョン 1.0.5.1.R2	2021 年 10 月 26 日現在、エンジンバージョン 1.0.5.1.R2 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、Neptune エンジンリリース 1.0.5.1.R2を参照してください。	2021 年 10 月 26 日
Deep Graph Library を使用した船舶リスクの HawkEye 予測に関するブログ記事	Tim Pavlick、Ian Avilez、Dan Ford、および Gaurav Rele によるHawkEye 「360 が Deep Graph Library と Amazon Neptune を使用して住宅リスクを予測する」を参照してください。	2021 年 10 月 15 日
エンジンバージョン 1.0.5.1	2021 年 10 月 1 日現在、エンジンバージョン1.0.5.1. は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、Neptune エンジンリリース 1.0.5.1を参照してください。	2021 年 10 月 1 日

デベロッパーが を好む理由に関するブログ記事 TinkerPop	ブラッド・ビー TinkerPop、ケルビン・ローレンス、スティーブン・マレットによるグラフコンピューティングのオープンソースフレームワークである Apache が開発者に好まれる理由を参照してください。	2021 年 9 月 27 日
エンジンバージョン 1.0.5.0.R3	2021 年 9 月 15 日現在、エンジンバージョン 1.0.5.0.R3 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、Neptune エンジンリリース 1.0.5.0.R3を参照してください。	2021 年 9 月 15 日
Amundsen と Neptune を使用したデータディスカバリソリューションの作成に関するブログ記事	ピーター・ハンセンとドン・シンプソンによるAmundsen と Amazon Neptune でデータディスカバリーソリューションを構築するを参照してください。	2021 年 9 月 8 日
Neptune は、非文字列フルテキスト検索クエリをサポートするようにストリームポーラーを更新しました	このリリースには、文字列ではないプロパティ値のインデックス作成のサポートなどのフルテキスト検索が改善されています。Amazon Amazon Neptune の「文字列以外の OpenSearch インデックス作成」を参照してください。	2021 年 8 月 23 日

エンジンバージョン 1.0.5.0.R2	2021 年 8 月 16 日現在、エンジンバージョン 1.0.5.0.R2 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、Neptune エンジンリリース 1.0.5.0.R2を参照してください。	2021 年 8 月 16 日
エンジンバージョン 1.0.4.2.R5	2021 年 8 月 16 日現在、エンジンバージョン 1.0.4.2.R5 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、Neptune エンジンリリース 1.0.4.2.R5を参照してください。	2021 年 8 月 16 日
Neptune でのグラフストアプロトコルのサポートに関するブログ投稿	クリス・スミスによるAmazon Neptune のグラフストアプロトコルサポートの紹介を参照してください。	2021 年 8 月 2 日
Neptune ML の新機能に関するブログ投稿	ソウジ・アデシナによるAmazon Neptune ML の新機能で、グラフでより多くの知見を得ようを参照してください。	2021 年 7 月 30 日
Neptune ML で予測を高速化することに関するブログ投稿	ソウジ・アデシナによるAmazon Neptune ML でグラフデータの進化の予測を迅速に取得を参照してください。	2021 年 7 月 30 日

Neptune ML を使用したグラフ機械学習の簡単で高速なブログ投稿	ソウジ・アデシナによるAmazon Neptune ML で機械学習を簡単かつ迅速にグラフ化を参照してください。	2021 年 7 月 30 日
Neptune openCypher のサポートに関するブログ投稿	ブラッド・ベビーによるAmazon Neptune 用 openCypher の発表。openCypher と Gremlin を併用し、より良いグラフアプリケーションを構築するを参照してください。	2021 年 7 月 29 日
エンジンバージョン 1.0.5.0	2021 年 7 月 27 日現在、エンジンバージョン1.0.5.0. は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、Neptune エンジンリリース 1.0.5.0を参照してください。	2021 年 7 月 27 日
エンジンバージョン 1.0.4.2.R4	2021 年 7 月 23 日現在、エンジンバージョン1.0.4.2.R4 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、Neptune エンジンリリース 1.0.4.2.R4を参照してください。	2021 年 7 月 23 日

Neptune が中国 (北京) で利用可能に	Amazon Neptune が中国 (北京) で利用可能になりました (`cn-north-1`)。	2021 年 7 月 21 日
エンジンバージョン 1.0.4.2.R3	2021 年 6 月 28 日現在、エンジンバージョン1.0.4.2.R3 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、Neptune エンジンリリース 1.0.4.2.R3を参照してください。	2021 年 6 月 28 日
Dream11 が Neptune を使ってソーシャルネットワークをどのように拡張したかについてのブログ投稿	「世界最大規模のファンタジースポーツプラットフォームである Dream11 が Amazon Neptune と Amazon でソーシャルネットワークを拡張する方法について説明します。」を参照してください ElastiCache。	2021 年 6 月 25 日
Neptune と PoolParty Semantic Suite を使用したデータからナレッジへの変換に関するブログ記事	Ioanna Lytra と Albin Ahmeti による PoolParty 「Semantic Suite と Amazon Neptune を使用してデータを知識に変換する」を参照してください。	2021年6月16日
Neptune を使用して UniProt ナレッジベースを詳しく調べるためのブログ記事	Eric Greene、Rafa Xu、Yuan Shi によるAWS 「Open Data と Amazon Neptune を使用した UniProt おい知識ベースの探索」を参照してください。	2021 年 6 月 10 日

Neptune を使用したデータ駆動型リスク分析の使用に関するブログ記事	Adriaan de Jonge と Rohit Satyanarayana による「フィールドノート: Amazon Neptune と Amazon OpenSearch Service によるデータ駆動型リスク分析」を参照してください。	2021 年 6 月 10 日
エンジンバージョン 1.0.4.2.R2	2021 年 6 月 1 日現在、エンジンバージョン1.0.4.2.R2 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、Neptune エンジンリリース 1.0.4.2.R2を参照してください。	2021 年 6 月 1 日
Neptune を使用した AWS インフラストラクチャの視覚化に関するブログ記事	Rohan Raizada と Amey Dhavle によるAmazon Neptune と AWS Config で AWS インフラストラクチャを視覚化する」を参照してください。	2021 年 5 月 25 日
Neptune で Data Lens を使用するための構成に関するブログ投稿	Russell Waterson の「Configure AWS services to build a knowledge graph in Amazon Neptune using Data Lens」を参照してください。	2021 年 5 月 5 日
Data Lens を使用した Neptune でのナレッジグラフの作成に関するブログ記事	ラッセル・ウォーターソンによるData Lens を使用して Amazon Neptune でナレッジグラフを構築するを参照してください。	2021 年 5 月 5 日

エンジンバージョン 1.0.1.0、1.0.1.1、および 1.0.1.2 が廃止されました	これ以降、これらのエンジンバージョンのいずれか、または関連するパッチを使用して、新しい DB インスタンスは作成されません。	2021 年 4 月 26 日
Neptune を使った日本の経済産業省に関する導入事例の英訳	日本の経済産業省が gBizINFO 企業情報検索データベースを AWSで強化を参照してください。	2021 年 3 月 31 日
Amazon Comprehend と Lex で Neptune を使用することに関するブログ投稿	デイヴ・ベッベルガーによるAmazon Neptune、Amazon Comprehend、Amazon Lex を使用して知識グラフを強化するを参照してください。	2021 年 3 月 31 日
Neptune での Lambda 関数の使用に関するブログ投稿	Ian Robinson によるAmazon Neptune で AWS Lambda 関数を使用する」を参照してください。	2021 年 3 月 26 日
エンジンバージョン 1.0.4.1.R1.1	2021 年 3 月 22 日現在、エンジンバージョン 1.0.4.1.R1.1 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、Neptune エンジンリリース 1.0.4.1.R1.1を参照してください。	2021 年 3 月 22 日

エンジンバージョン 1.0.4.1.R2.1	2021 年 3 月 11 日現在、エンジンバージョン 1.0.4.1.R2.1 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、Neptune エンジンリリース 1.0.4.1.R2.1を参照してください。	2021 年 3 月 11 日
グラフの視覚化に Neptune のオープンソースのグラフノートブックを使用することに関するブログ投稿	ジョイ・ワン、オラ・ラッシラ、スティーヴン・マレットによるグラフ可視化のためのオープンソースのグラフノートブックの使用開始を参照してください。	2021 年 3 月 10 日
Neptune と Amundsen のデータディスカバリおよびメタデータエンジンとの統合に関するチュートリアル	アンドリュー・チャムブローンによるAmazon Neptune と Amundsenの使い方を参照してください。	2021 年 3 月 2 日
エンジンバージョン 1.0.4.1.R2	2021 年 2 月 24 日現在、エンジンバージョン 1.0.4.1.R2 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、Neptune エンジンリリース 1.0.4.1.R2を参照してください。	2021 年 2 月 24 日

エンジンバージョン 1.0.4.0.R2	2021 年 2 月 24 日現在、エンジンバージョン 1.0.4.0.R2 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、Neptune エンジンリリース 1.0.4.0.R2を参照してください。	2021 年 2 月 24 日
エンジンバージョン 1.0.3.0.R3	2021 年 2 月 19 日現在、エンジンバージョン 1.0.3.0.R3 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、Neptune エンジンリリース 1.0.3.0.R3を参照してください。	2021 年 2 月 19 日
エンジンバージョン 1.0.2.2.R6	2021 年 2 月 19 日現在、エンジンバージョン1.0.2.2.R6 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、Neptune エンジンリリース 1.0.2.2.R6を参照してください。	2021 年 2 月 19 日

Amazon Comprehend イベントを使用したナレッジグラフの作成に関するブログ記事	ブライアン・オキーフ、グラハム・ホーウッド、ナブタナイ・シンハによるAmazon Comprehend イベントを使用して Amazon Neptune でナレッジグラフを作成するを参照してください。	2021 年 1 月 19 日
ローコードグラフデータアプリの有効化に関するブログ投稿	レオ・メイエロヴィッチ、デイブ・ベヒバーガー、テイラー・リガンによるAmazon Neptune と Graphistry でローコードグラフデータアプリを有効にするを参照してください。	2021 年 1 月 18 日
グラフデータの入門用ノートブックドキュメントが追加されました。	Neptune ワークベンチとの統合セクションを追加しました。これにより、準備が整うまで Neptune クラスターをスピンアップしなくても、グラフデータの作成とグラフアプリケーションの開発を開始できます。	2021 年 1 月 15 日
Neptune グラフデータを数秒でリセットするためのブログ投稿	ニラジ・ジェトリーとナブタナイ・シンハによるAmazon Neptune のグラフデータを数秒でリセットするを参照してください。	2020 年 12 月 17 日

BERT での と SageMaker Neptune の使用方法に関するブログ記事	Othmane Hamzaoui、Fatema Alkhanaizi、Viktor Malsevic SageMaker Amazon Neptune による「Crymane Hamzaoui、Fatema Alkhanaizi、Viktor Malsevic」の「」を参照してください。	2020 年 12 月 14 日
トピックネットワークを使用したナレッジグラフの作成に関するブログ記事	AI プロジェクト責任者のエドワード・ブラウン、建築家のマルシア・オリヴェイラ、リードデータサイエンティストである Deeper Insights の最高経営責任者（CEO）のジャック・ハンプソンによる Amazon Neptune でトピックネットワークを使用したナレッジグラフの作成を参照してください。	2020 年 12 月 14 日
エンジンバージョン 1.0.4.1	2020 年 12 月 8 日現在、エンジンバージョン1.0.4.1. は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、Neptune エンジンリリース 1.0.4.1を参照してください。	2020 年 12 月 8 日
Neptune ML で開始することに関するブログ投稿	ジョージ・カリピス、デイヴ・ベカーバーガー、カルティック・バラティによる Neptune ML の使用を開始する方法を参照してください。	2020 年 12 月 8 日

Neptune に高速リセット API が備わりました	高速リセット API を使用すると、DB クラスター内のすべてのデータをすばやく簡単に削除できます。高速リセット APIを参照してください。	2020 年 12 月 4 日
Pendulum での生物学的ナレッジグラフの作成に関するブログ記事	コナー・スケナートンによるAmazon Neptune を使用して Pendulum で生物学的知識グラフを作成するを参照してください。	2020 年 11 月 26 日
Neptune の TinkerPop 3.4.8 の新機能に関するブログ記事	Stephen Mallette によるAmazon Neptune での Apache TinkerPop 3.4.8 の新機能の探索」を参照してください。	2020 年 11 月 18 日
Neptune での Amazon Kendra 検索サービスの使用に関するブログ投稿	ヤツダン・シルヴァニ、モヒット・メタ、ディプト・チャクラヴァルティによるエンタープライズナレッジグラフを Amazon Kendra に組み込むを参照してください。、。	2020 年 11 月 17 日
イベント通知が利用可能になりました	Neptune は DB クラスターをより簡単に監視するために使用できるイベント通知をサポートするようになりました。Neptune イベント通知の使用を参照してください。	2020 年 10 月 29 日

カスタムエンドポイントが利用可能になりました	Neptune は DB インスタンスへの接続をより詳細に制御するためのカスタムエンドポイントをサポートするようになりました。Amazon Neptune エンドポイントに接続するを参照してください。	2020 年 10 月 29 日
AWS Database Migration Service (DMS) を使用した Neptune グラフへの入力に関するブログ記事	Database AWS Migration Service (DMS) を使用したリレーショナルデータベースからの Amazon Neptune でのグラフの入力 – パート 4: すべてをまとめる - Chris Smith による を参照してください。	2020 年 10 月 22 日
AWS Database Migration Service (DMS) を使用した Neptune グラフへの入力に関するブログ記事	Database AWS Migration Service (DMS) を使用したリレーショナルデータベースからの Amazon Neptune でのグラフの入力 – パート 3: Chris Smith による RDF モデル の設計を参照してください。	2020 年 10 月 22 日
AWS Database Migration Service (DMS) を使用した Neptune グラフへの入力に関するブログ記事	Database AWS Migration Service (DMS) を使用したリレーショナルデータベースからの Amazon Neptune でのグラフの入力 – パート 2: プロパティグラフモデル の設計 を参照してください。	2020 年 10 月 22 日

AWS Database Migration Service (DMS) を使用した Neptune グラフへの入力に関するブログ記事	Database AWS Migration Service (DMS) を使用したリレーショナルデータベースからの Amazon Neptune でのグラフの入力 – パート 1: ステージ の設定 を参照してください。	2020 年 10 月 22 日
エンジンバージョン 1.0.4.0	2020 年 10 月 12 日現在、エンジンバージョン1.0.4.0. は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、Neptune エンジンリリース 1.0.4.0を参照してください。	2020 年 10 月 12 日
エンジンバージョン 1.0.3.0.R2	2020 年 10 月 12 日現在、エンジンバージョン 1.0.3.0.R2 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、Neptune エンジンリリース 1.0.3.0.R2を参照してください。	2020 年 10 月 12 日

エンジンバージョン 1.0.2.2.R5	2020 年 10 月 12 日現在、エンジンバージョン1.0.2.2.R5は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、Neptune エンジンリリース 1.0.2.2.R5を参照してください。	2020 年 10 月 12 日
SPARQL フェデレーションクエリ用の VPC の設定に関するブログ投稿	チャールズ・イヴィーによるAmazon Neptune を使用した SPARQL 1.1 フェデレーションクエリ用に Amazon VPC を設定するを参照してください。	2020 年 10 月 12 日
SPARQL カスケード削除の記述に関するブログ投稿	オラ・ラッシーラによるSPARQL でカスケード削除を書くを参照してください。	2020 年 10 月 5 日
Neptune を使用した AWS リソースのグラフ化に関するブログ記事	Dave Bechberger によるAmazon Neptune で AWS リソースをグラフ化する」を参照してください。	2020 年 9 月 28 日
Neptune を使用したファーマコビジランスと有害事象報告のための MedDRA 用語マッピングの構築に関するブログ投稿	ヴァイジャヤンティ・ジョシ、デヴェン・アントゥール、スダンシュ・マルホトラ博士によるAmazon Neptune ベースの MedDRA 用語マッピングを構築して、ファーマコビジランスと有害事象レポートを作成するを参照してください。	2020 年 9 月 24 日

Neptune を使用してデータウェアハウスからナレッジグラフを構築し、コマーシャルインテリジェンスを補完することについてのブログ投稿	シャフリア・ホセインとミカエル・グレインドルゲによるAmazon Neptune でデータウェアハウスからナレッジグラフを構築して商業インテリジェンスを補完を参照してください。	2020 年 9 月 23 日
Neptune Gremlin クライアントを使用した負荷分散に関するブログ投稿	イアン・ロビンソンによるAmazon Neptune Gremlin クライアントを使用した負荷分散グラフクエリを参照してください。	2020 年 9 月 16 日
Cox Automotive でのアイデンティティグラフを使用したデジタルパーソナライゼーションに関するブログ投稿	カルロス・レンドンとニラジ・ジェトリーによるCox Automotive は Amazon Neptune を搭載したアイデンティティグラフを使用して、デジタルパーソナライゼーションをスケーリングするを参照してください。	2020 年 9 月 16 日
Yelp データの共同フィルタリングに関するブログ投稿	チャド・ティンデルによるYelp データで共同フィルタリングを使用して Amazon Neptune でレコメンデーションシステムを構築するを参照してください。	2020 年 9 月 8 日
Amazon Neptune でのクエリ結果の可視化に関するブログ投稿	ケルビン・ローレンスによるAmazon Neptune ワークベンチを使用してクエリ結果を可視化するを参照してください。	2020 年 9 月 2 日

Neptune がグラフ可視化をリリースしました	Amazon Neptune は、Neptune ワークベンチの Jupyter ノートブックに広範なグラフ可視化機能を提供し、ノートブックがより使いやすくなる多くの新機能を備えています。グラフビジュアライゼーションを参照してください。	2020 年 8 月 12 日
Neptune が南米 (サンパウロ) で利用可能に	Amazon Neptune が南米 (サンパウロ) (sa-east-1) リージョンで利用可能になりました。	2020 年 8 月 6 日
Neptune がアジアパシフィック (香港) で利用可能に	Amazon Neptune がアジアパシフィック (香港) (ap-east-1) リージョンで利用可能になりました。	2020 年 8 月 6 日
エンジンバージョン 1.0.3.0	2020 年 8 月 3 日現在、エンジンバージョン1.0.3.0. は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、Neptune エンジンリリース 1.0.3.0を参照してください。	2020 年 8 月 3 日

エンジンバージョン 1.0.2.2.R4	2020 年 7 月 23 日現在、エンジンバージョン1.0.2.2.R4 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、Neptune エンジンリリース 1.0.2.2.R4を参照してください。	2020 年 7 月 23 日
Amazon Neptune を使用した Zerobase の自動コンタクトトレースに関するブログ投稿	デヴィッド・ハリスとアロン・サントによるZerobase は Amazon Neptune を使用して、プライベートの、安全な、自動化された連絡先トレースを作成するを参照してください。	2020 年 7 月 13 日
Neptune が米国西部 (北カリフォルニア) で利用可能に	Amazon Neptune が米国西部 (北カリフォルニア) で利用可能になりました (`us-west-1`)。	2020 年 7 月 9 日
Amazon Neptune はタグベースのアクセスコントロールをサポートしています	IAM ポリシーの AWS タグを使用して、Neptune データベースへのアクセスを制御できるようになりました。Amazon Neptune におけるタグベースのアクセスコントロールを参照してください。	2020 年 7 月 7 日

Java ストリームポーラーが使用可能になりました	Amazon Neptune は、Python と同様に Neptune ストリーム用の Java バージョンのラムダストリームポーラーをサポートするようになりました。作成中の Neptune Streams コンシューマースタックの詳細の追加を参照してください。	2020 年 7 月 6 日
AWS COVID-19 ナレッジグラフに関するブログ記事	Ninad Kulkarni、Collby Wise、ジョージ・プライス、Miguel Romero による「Building and querying the AWS COVID-19 knowledge graph 」を参照してください。	2020 年 7 月 1 日
エンジンバージョン 1.0.1.1	2020 年 6 月 26 日 の時点で、エンジンのバージョン 1.0.1.1 は一般的にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、Neptune エンジンリリース 1.0.1.1を参照してください。	2020 年 6 月 26 日
Blazegraph から Amazon Neptune への移行に関するブログ記事	Dave Bechberger 氏による「Moving to the cloud: Migrating Blazegraph to Amazon Neptune」を参照してください。	2020 年 6 月 25 日

Neo4j から Amazon Neptune へのデータキャプチャの変更に関するブログ記事	Sanjeet Sahay 氏による「Change data capture from Neo4j to Amazon Neptune using Amazon Managed Streaming for Apache Kafka」を参照してください。	2020年6月22日
Waves における Amazon Neptune の使用に関するブログ記事	Pavel Vasilyev 氏による「How Waves runs user queries and recommendations at scale with Amazon Neptune」を参照してください。	2020 年 6 月 16 日
エンジンバージョン 1.0.1.2	2020 年 6 月 10 日の時点で、エンジンのバージョン 1.0.1.2 は一般的にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、Neptune エンジンリリース 1.0.1.2を参照してください。	2020 年 6 月 10 日
顧客ナレッジリポジトリの構築に関するブログ記事	Ram Bhandarkar 氏による「Building a customer 360 knowledge repository with Amazon Neptune and Amazon Redshift」を参照してください。	2020 年 6 月 9 日

Gunosy における Amazon Neptune の使用に関するブログ記事	Yosuke Uchiyama 氏による「How Gunosy built a comment feature in News Pass using Amazon Neptune」を参照してください。	2020 年 6 月 8 日
AWS COVID-19 ナレッジグラフに関するブログ記事	Ninad Kulkarni、Collby Wise、ジョージ・プライス、Miguel Romero による「Building and querying the AWS COVID-19 knowledge graph」を参照してください。	2020 年 6 月 2 日
Amazon Neptune を使用して COVID-19 の研究を探るブログ記事	George Price、Colby Wise、Miguel Romero、Ninad Kulkarni による、Exploring scientific research on COVID-19 with Amazon Neptune, Amazon Comprehend Medical, and the Tom Sawyer Graph Database Browser を参照してください。	2020 年 6 月 2 日
を使用して Neptune にデータをロードできるようになりました。 AWS DMS	AWS 「データベース移行サービスを使用して、別のデータストア から Amazon Neptune にデータをロードする」を参照してください。	2020 年 6 月 1 日

エンジンバージョン 1.0.2.0 は非推奨です	Amazon Neptune エンジンのバージョン 1.0.2.0 は非推奨です。このエンジンバージョンで実行されているクラスターは、2020 年 6 月 1 日以降の最初のメンテナンスウィンドウに、自動的にバージョン 1.0.2.1 にアップグレードされます。	2020 年 5 月 19 日
Neptune を使用したカスタマー ID グラフの作成に関するブログ記事	Rajesh Wunnava 氏および Taylor Riggan 氏による「Building a customer identity graph with Amazon Neptune」を参照してください。	2020 年 5 月 12 日
エンジンバージョン 1.0.2.0.R3	2020 年 5 月 5 日現在、エンジンバージョン1.0.2.0.R3 が一般展開されています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、Neptune エンジンリリース 1.0.2.0.R3を参照してください。	2020 年 5 月 5 日
エンジンバージョン 1.0.2.1.R6	2020 年 4 月 22 日現在、エンジンバージョン1.0.2.1.R6 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、Neptune エンジンリリース 1.0.2.1.R6を参照してください。	2020 年 4 月 22 日

Neo4j から Neptune へのデータの移行に関するブログ記事	Sanjeet Sahay 氏による「Migrating a Neo4j graph database to Amazon Neptune with a fully automated utility」するを参照してください。	2020 年 4 月 13 日
Neptune でのグラフアプリケーションの構築コストの削減に関するブログ記事	Karthik Bharathy 氏および Brad Bebee 氏「Lower the cost of building graph apps by up to 76% with Amazon Neptune T3 instances」を参照してください。	2020 年 4 月 9 日
Neptune には、T3 バースト可能インスタンスクラスが用意されています。	コスト効率の高い開発およびテストのために、Amazon Neptune T3 バースト可能インスタンスを作成できるようになりました。「Neptune T3 バースト可能インスタンスクラス」を参照してください。	2020 年 4 月 8 日
エンジンバージョン 1.0.2.2.R2	2020 年 4 月 2 日現在、エンジンバージョン1.0.2.2.R2 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、Neptune エンジンリリース 1.0.2.2.R2を参照してください。	2020 年 4 月 2 日
EDGAR での投資依存性のグラフ化に関するブログ記事	Lawrence Verdi 氏による「Graphing investment dependency with Amazon Neptune」を参照してください。	2020 年 3 月 17 日

Neptune が欧州 (パリ) で利用可能に	Amazon Neptune がヨーロッパ (パリ) (eu-west-3) で利用可能になりました。	2020 年 3 月 11 日
エンジンバージョン 1.0.2.2	2020 年 3 月 9 日現在、エンジンバージョン1.0.2.2. は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。このエンジンバージョンの詳細については、Neptune エンジンリリース 1.0.2.2を参照してください。	2020 年 3 月 9 日
DB クラスターの停止と再起動	Neptune コンソールを使用して DB クラスターを 7 日間停止し、後で再び必要なときに再起動できるようになりました。DB クラスターの停止中は、DB インスタンスの時間ではなく、クラスターストレージ、手動のスナップショット、自動化されたバックアップストレージに対してのみ課金されます。詳しくは「Amazon Neptune DB クラスターの停止と開始」を参照してください。	2020 年 2 月 19 日

Nike のソーシャルグラフに関するビデオ	Todd Escalona of AWS talks with Marc Wangenheim, Senior Engineering Manager at , about the company powers a number of applications via a social graph built on Amazon Neptune 。「Nike: A Social Graph at Scale with Amazon Neptune」を参照してください。	2020 年 2 月 11 日
Neptune クラスターが SSL 接続を要求するように設定できるようになりました。	HTTP 接続をサポートしているリージョンでは、すべての新しいパラメータグループで SSL がデフォルトでオンになるようになりました。既存のパラメータグループには変更はありませんが、`neptune_enforce_ssl` パラメータを 1 に変更することで、クライアントに SSL を使用するように強制できます。HTTP 接続をサポートしているリージョンでクラスターの HTTP 接続を有効にする方法については、「転送中の暗号化: SSL/HTTPS を使用して Neptune に接続する」を参照してください。クラスターおよびインスタンスのパラメータの説明については、「Amazon Neptune の設定に使用可能なパラメータ」を参照してください。	2020 年 2 月 10 日

Neptune の CloudFormation テンプレートでエンジンバージョンと削除保護を指定できるようになりました	Amazon Neptune は、新しい DB クラスターに特定のエンジンバージョンを指定できる `AWS::Neptune::DBCluster.EngineVersion` パラメータと、その削除保護をオンにできる `AWS::Neptune::DBCluster.DeletionProtection` パラメータを含むように CloudFormation テンプレートを更新しました。	2020 年 2 月 9 日
削除保護	Amazon Neptune は、DB クラスターとインスタンスの削除保護を公開しました。DB クラスターまたはインスタンスで削除保護が有効になっている限り、削除することはできません。「削除保護が有効になっている場合、DB インスタンスを削除できません」を参照してください。	2020 年 1 月 20 日
Neptune が中国 (寧夏) で利用可能に	Amazon Neptune が中国 (寧夏) (cn-northwest-1) で利用可能になりました。	2020 年 1 月 15 日
エンジンバージョン 1.0.2.1.R4	エンジンバージョン 1.0.2.1 のパッチ R4 が一般公開されました。詳細については、Neptune エンジンリリース 1.0.2.1.R4を参照してください。	2019 年 12 月 20 日

エンジンバージョン 1.0.2.1.R3	エンジンバージョン 1.0.2.1 のパッチ R3 が一般公開されました。詳細については、Neptune エンジンリリース 1.0.2.1.R3を参照してください。	2019 年 12 月 12 日
Neptune を使ったソーシャルメディアフィードの分析についてのブログ記事	「Analyzing social media feeds using Amazon Neptune」を参照してください。	2019 年 11 月 27 日
エンジンバージョン 1.0.2.1.R2	エンジンバージョン 1.0.2.1 のパッチ R2 が一般公開されました。詳細については、Neptune エンジンリリース 1.0.2.1.R2を参照してください。	2019 年 11 月 25 日
エンジンバージョン 1.0.2.1.R1	Amazon Neptune エンジンバージョン 1.0.2.1.R1 は一般にご利用いただけます。詳細については、Neptune エンジンリリース 1.0.2.1を参照してください。	2019 年 11 月 22 日
エンジンバージョン 1.0.2.0.R2	エンジンバージョン 1.0.2.0 のパッチ R2 が一般公開されました。詳細については、Neptune エンジンリリース 1.0.2.0.R2を参照してください。	2019 年 11 月 21 日

re:Invent 2019 での Neptune セッションとワークショップに関するブログ記事	reAWS :Invent 2019 の Amazon Neptune セッション、ワークショップ、およびチャットトークのガイド」を参照してください。	2019 年 11 月 20 日
エンジンバージョン 1.0.2.0.R1	Amazon Neptune エンジンバージョン1.0.2.0.R1 が一般展開されています。詳細については、Neptune エンジンリリース 1.0.2.0を参照してください。	2019年11月8日
Neptune Streams を使用してグラフの変化をキャプチャすることに関するブログ記事	「Neptune Streams を使用したグラフの変化の取り込み」を参照してください。	2019 年 11 月 6 日
エンジンバージョン 1.0.1.0.200502.0	Amazon Neptune エンジンのバージョン 1.0.1.0.200502.0 が一般に入手可能です。詳細については、1.0.1.0.200502.0 のアップデートを参照してください。	2019 年 10 月 31 日
Neptune が中東 (バーレーン) で利用可能に	Amazon Neptune が中東 (バーレーン) (me-south-1) で利用可能になりました。	2019 年 10 月 30 日
Neptune がカナダ (中部) で利用可能に	Amazon Neptune が、カナダ (中部) (ca-central-1) で利用可能になりました。	2019 年 10 月 30 日
Neptune の新しい SPARQL ストリーム機能と SPARQL フェデレーテッドクエリのサポートに関するブログ記事	Amazon Neptune で Streams、グラフ用 SPARQL フェデレ–テッドクエリなどの新機能がリリースを参照してください。	2019 年 10 月 17 日

エンジンバージョン 1.0.1.0.200463.0	Amazon Neptune エンジンバージョン 1.0.1.0.200463.0 が一般展開されています。詳細については、1.0.1.0.200463.0 のアップデートを参照してください。	2019 年 10 月 15 日
エンジンバージョン 1.0.1.0.200457.0	Amazon Neptune エンジンバージョン 1.0.1.0.200457.0 は一般にご利用いただけます。詳細については、1.0.1.0.200457.0 のアップデートを参照してください。	2019 年 9 月 19 日
Neptune の新しい SPARQL 説明機能に関するブログっ記事	「SPARQL explain を使用して Amazon Neptune のクエリ実行を理解する」を参照してください。	2019 年 9 月 17 日
Neptune TinkerPop 3.4 のサポートに関するブログ記事	Amazon Neptune は TinkerPop 3.4 機能をサポートするようになりました」を参照してください。	2019 年 9 月 6 日
Amazon PyTorch での での Neptune の使用に関するブログ記事 SageMaker	Amazon SageMaker および Amazon Neptune PyTorch でを使用したパーソナライズされた「スタイルごとのショッピング」エクスペリエンスを参照してください。	2019年8月22日
AWS AppSync および Amazon Elasticache での Neptune の使用に関するブログ記事	「代替データソースと の統合 AWS AppSync: Amazon Neptune および Amazon ElastiCache」を参照してください。	2019年8月22日

Neptune が AWS GovCloud (米国東部) で利用可能に	Amazon Neptune が AWS GovCloud (米国東部) (us-gov-east-1) で利用可能になりました。	2019 年 8 月 21 日
Neptune が AWS GovCloud (米国西部) で利用可能に	Amazon Neptune が AWS GovCloud (米国西部) (us-gov-west-1) で利用可能になりました。	2019 年 8 月 14 日
エンジンバージョン 1.0.1.0.200369.0	Amazon Neptune エンジンバージョン 1.0.1.0.200369.0 は一般にご利用いただけます。詳細については、1.0.1.0.200369.0 のアップデートを参照してください。	2019 年 8 月 13 日
エンジンバージョン 1.0.1.0.200366.0	Amazon Neptune エンジンバージョン 1.0.1.0.200366.0 は一般にご利用いただけます。詳細については、1.0.1.0.200366.0 のアップデートを参照してください。	2019 年 7 月 26 日
Amazon PyTorch での での Neptune の使用に関するブログ記事 SageMaker	Amazon SageMaker および Amazon Neptune PyTorch でを使用したパーソナライズされた「スタイルごとのショッピング」エクスペリエンスを参照してください。	2019 年 7 月 3 日

エンジンバージョン 1.0.1.0.200348.0	Amazon Neptune エンジンバージョン 1.0.1.0.200348.0 は一般にご利用いただけます。詳細については、1.0.1.0.200348.0 のアップデートを参照してください。	2019 年 7 月 2 日
Neptune が 欧州 (ストックホルム) で利用可能に	Amazon Neptune がヨーロッパ (ストックホルム) (eu-north-1) で利用可能になりました。	2019 年 6 月 27 日
Neptune が監査ログを CloudWatch Logs に発行できるようになりました	詳細については、「Amazon Logs への Neptune CloudWatch ログの発行」を参照してください。	2019 年 6 月 18 日
エンジンバージョン 1.0.1.0.200310.0	Amazon Neptune エンジンバージョン 1.0.1.0.200310.0 は一般にご利用いただけます。詳細については、1.0.1.0.200310.0 のアップデートを参照してください。	2019 年 6 月 12 日
LifeOmicに関するブログ記事 JupiterOne	LifeOmic「 が Amazon Neptune でセキュリティおよびコンプライアンスオペレーション JupiterOne を簡素化する方法」を参照してください。	2019 年 5 月 2 日
Neptune がアジアパシフィック (ソウル) で利用可能に	Amazon Neptune が、アジアパシフィック (ソウル) (ap-northeast-2) で利用可能になりました。	2019 年 5 月 1 日

エンジンバージョン 1.0.1.0.200296.0	Amazon Neptune エンジンバージョン 1.0.1.0.200296.0 は一般にご利用いただけます。詳細については、1.0.1.0.200296.0 のアップデートを参照してください。	2019 年 5 月 1 日
Neptune がアジアパシフィック (ムンバイ) で利用可能に	Amazon Neptune が、アジアパシフィック (ムンバイ) (ap-south-1) で利用可能になりました。	2019 年 3 月 6 日
Gremlin クエリのヒントに関するブログ記事	「Amazon Neptune 用の Gremlin クエリヒントの紹介」を参照してください。	2019 年 2 月 26 日
Neptune がアジアパシフィック (東京) で利用可能に	Amazon Neptune が、アジアパシフィック (東京) (ap-northeast-1) リージョンで利用可能になりました。	2019 年 1 月 23 日
AWS CloudFormation Neptune にアクセスするための AWS Lambda 関数を作成するための テンプレート	「開始方法」セクションを更新し、Neptune で使用する Lambda 関数を作成するための AWS CloudFormation テンプレートを追加しました。詳細については、Neptune の開始方法を参照してください。	2019 年 1 月 23 日
エンジンバージョン 1.0.1.0.200267.0	Amazon Neptune エンジンバージョン 1.0.1.0.200267.0 は一般にご利用いただけます。詳細については、1.0.1.0.200267.0 のアップデートを参照してください。	2019 年 1 月 21 日

Neptune がアジアパシフィック (シドニー) で利用可能に	Amazon Neptune が、アジアパシフィック (シドニー)(ap-southeast-2) で利用可能になりました。	2019 年 1 月 9 日
Metaphactory の使用に関するブログ記事	「Metaphactory を使った Amazon Neptune 上のナレッジグラフの探索」を参照してください。	2019 年 1 月 9 日
Neptune がアジアパシフィック (シンガポール) で利用可能に	Amazon Neptune が、アジアパシフィック (シンガポール) (ap-southeast-1) で利用可能になりました。	2018 年 12 月 13 日
エンジンバージョン 1.0.1.0.200264.0	Amazon Neptune エンジンバージョン 1.0.1.0.200264.0 は一般にご利用いただけます。詳細については、1.0.1.0.200264.0 のアップデートを参照してください。	2018 年 11 月 19 日
Amazon Neptune SSL Support	Neptune が SSL 接続をサポートするようになりました。	2018 年 11 月 19 日
一括エラートピック	すべてのエラーメッセージとコード情報は、1 つのトピックになりました。	2018 年 11 月 15 日
「開始方法」のトピックが更新されました	追加リンクとともに「開始方法」のトピックが更新され、ドキュメントが再編されました。	2018 年 11 月 14 日

エンジンバージョン 1.0.1.0.200258.0	Amazon Neptune エンジンバージョン 1.0.1.0.200258.0 が一般展開されています。詳細については、1.0.1.0.200258.0 のアップデートを参照してください。	2018 年 11 月 8 日
Neptune が欧州 (フランクフルト) で利用可能に	Amazon Neptune がヨーロッパ (フランクフルト) (eu-central-1) で利用可能になりました。	2018 年 11 月 7 日
シリーズのブログ記事 #1	「Let Me Graph That For You – Part 1 – Air Routes」を参照してください。	2018 年 11 月 7 日
Amazon SageMaker Jupyter Notebooks の使用に関するブログ記事	「Amazon Jupyter Notebooks を使用して Amazon Neptune グラフを分析する」を参照してください SageMaker 。	2018 年 11 月 1 日
エンジンバージョン 1.0.1.0.200255.0	Amazon Neptune エンジンバージョン 1.0.1.0.200255.0 が一般展開されています。詳細については、1.0.1.0.200255.0 のアップデートを参照してください。	2018 年 10 月 29 日
Neptune が欧州 (ロンドン) で利用可能に	Amazon Neptune がヨーロッパ (ロンドン) (eu-west-2) で利用可能になりました。	2018 年 10 月 3 日

エンジンバージョン 1.0.1.0.200237.0	Amazon Neptune エンジンバージョン 1.0.1.0.200237.0 は一般にご利用いただけます。詳細については、1.0.1.0.200237.0 のアップデートを参照してください。	2018 年 9 月 6 日
エンジンバージョン 1.0.1.0.200236.0	Amazon Neptune エンジンバージョン 1.0.1.0.200236.0 は一般にご利用いただけます。詳細については、1.0.1.0.200236.0 のアップデートを参照してください。	2018 年 7 月 24 日
エンジンバージョン 1.0.1.0.200233.0	Amazon Neptune エンジンバージョン 1.0.1.0.200233.0 は一般にご利用いただけます。詳細については、1.0.1.0.200233.0 のアップデートを参照してください。	2018 年 6 月 22 日
新しい Neptune クイックスタート	AWS CloudFormation と Gremlin コンソールのチュートリアルでクイックスタートを更新しました。詳細については、「 を使用した Amazon Neptune クイックスタート AWS CloudFormation」を参照してください。	2018 年 6 月 19 日

Amazon Neptune 初回リリース	これは、『Neptune ユーザーガイド』の最初のリリースです。リリースのブログ投稿「Amazon Neptune の一般公開を開始」も参照してください。	2018 年 5 月 30 日
Neptune ブログでの紹介の投稿	「Amazon Neptune – 完全マネージド型のグラフデータベースサービス」を参照してください。	2017 年 11 月 29 日

Amazon Neptune の開始方法

Amazon Neptune はフルマネージド型のグラフデータベースサービスで、数十億のリレーションシップを処理するように拡張でき、ミリ秒のレイテンシーでクエリを実行でき、容量の規模の割には低コストでクエリを実行できます。

Neptune に関するより詳細な情報については、「Amazon Neptune の機能の概要」を参照してください。

グラフについて既にわかっている場合は、グラフノートブックを使用する に進んでください。または、Neptune データベースをすぐに作成する場合は、「AWS CloudFormation スタックを使用して Neptune DB クラスターを作成する」を参照してください。

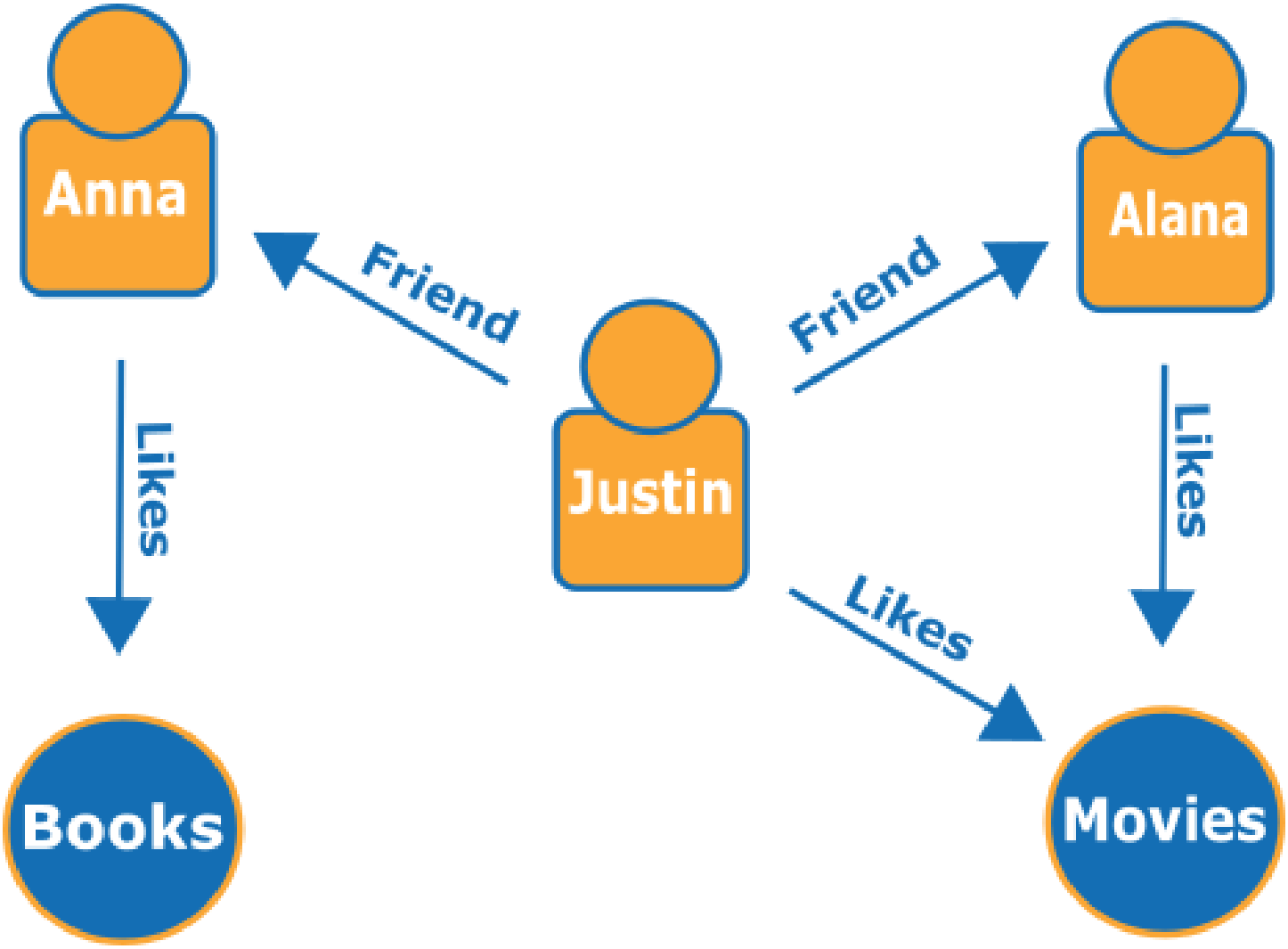
それ以外の場合は、始める前にグラフデータベースについてもう少し詳しく知りたいと思うかもしれません。

グラフデータベースとは正確には何か?

グラフデータベースは、データ項目間の関係を格納し照会するために最適化されています。

データ項目自体をグラフの頂点として、それらの間の関係はエッジとして格納します。各エッジにはタイプがあり、ある頂点 (始点) から別の頂点 (終点) に向けられます。関係はエッジだけでなく述語とも呼ばれ、頂点はノードと呼ばれることもあります。いわゆるプロパティグラフでは、頂点とエッジの両方がそれらに関連付けられた追加のプロパティを持つことができます。

ソーシャルネットワーク内の友達や趣味を表す小さなグラフを次に示します。



エッジは名前の付いた矢印で表示され、頂点はつながりのある特定の人物や趣味を表します。

このグラフの単純な探索により、Justin の友だちについて知ることができます。

グラフデータベースを使用する理由

モデル化するデータの核となるのがエンティティ同士のつながりまたは関係性である場合は、グラフデータベースが適しています。

1 つは、データ相互接続をグラフとしてモデル化し、グラフから実世界の情報を抽出する複雑なクエリを簡単に作成すできます。

リレーショナルデータベースを使用して同等のアプリケーションを構築するには、複数の外部キーを持つテーブルを多数作成し、ネストされた SQL クエリと複雑な結合を記述する必要があります。こ

のアプローチは、コーディングの観点からすぐに扱いにくくなるだけでなく、データ量が増えるにつれてパフォーマンスが急速に低下します。

対照的に、Neptune のようなグラフデータベースは、非常に多くの頂点間の関係を乱すことなく照会できます。

グラフデータベースでできること

グラフは、アクション、所有権、親子関係、購入の選択、個人的なつながり、家族の関係などの観点から、現実世界のエンティティの相互関係をさまざまな方法で表すことができます。

グラフデータベースが使用される最も一般的な領域を次に示します。

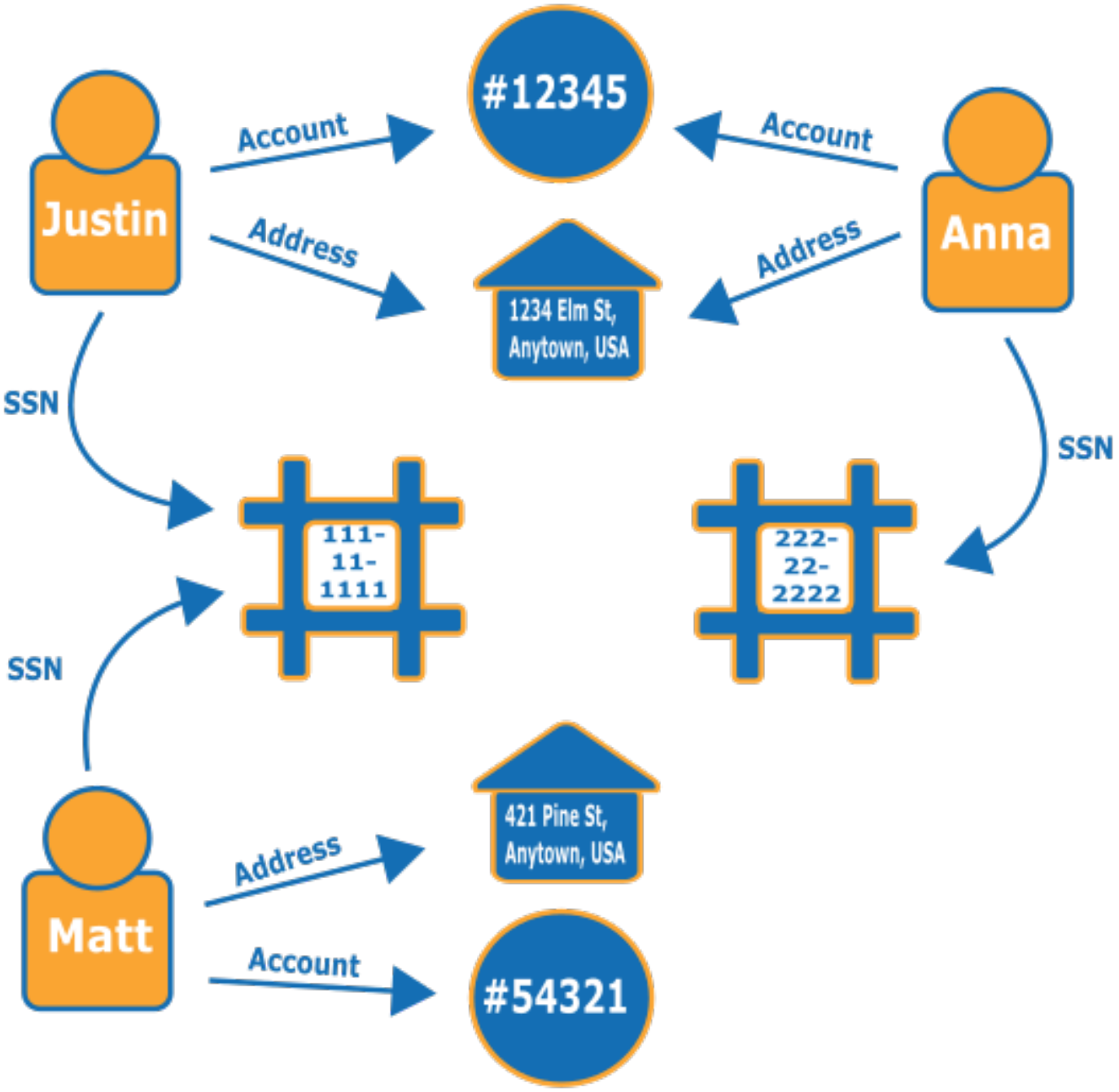
- ナレッジグラフ - ナレッジグラフを使用すると、接続されているあらゆる種類の情報を整理して照会し、一般的な質問に答えることができます。ナレッジグラフを使用して、トピック情報を製品カタログに追加したり、Wikidata に含まれるのような多様な情報をモデル化したりできます。

  ナレッジグラフがどのように機能し、使用されているかについては、AWS についてのナレッジグラフを参照してください。
- アイデンティティグラフ - グラフデータベースでは、顧客の興味、友人、購入履歴のような情報カテゴリの間の関係をグラフに保存し、データを照会して、パーソナライズされた関連性のある推奨事項を作成できます。

  たとえば、高可用性のグラフデータベースを使用して、同じスポーツをフォローしている他のユーザーや、購入履歴が似ている他のユーザーが購入した製品に基づいて、ユーザーに製品の推奨を行うことができます。または、共通の友人がいて、お互いはまだ知り合っていない人物を特定し、友人関係の推奨を行うことができます。

  この種のグラフはアイデンティティグラフと呼ばれ、ユーザーとのインタラクションをパーソナライズするために広く使用されています。詳細については、AWS についてのアイデンティティグラフを参照してください。独自のアイデンティティグラフの作成を開始するには、まず Amazon Neptune を使用したアイデンティティグラフサンプルから始められます。
- 不正グラフ — これは、グラフデータベースの一般的な使用法です。これらは、クレジットカードの購入と購入場所を追跡して、特徴的でない使用を検出したり、購入者が既知の不正行為で使用したのと同じメールアドレスとクレジットカードを使用しようとしていることを検出するのに役立ちます。これにより、1 つの個人用 E メールアドレスに関連付けられている複数の人物、同じ IP アドレスを共有する異なる物理的な場所にいる複数の人物を確認することができます。

次のグラフについて考慮します。3 名の人物と ID 関連の情報の関係が示されています。それぞれの人物に、住所、銀行口座、および社会保障番号があります。ただし、Matt および Justin は同じ社会保障番号を共有していることがわかります。これは異常なことで、このうち1 人が不正を行っている可能性を示唆しています。不正グラフへのクエリでは、この種のつながりを明らかにし、レビューできます。



不正グラフがどのように機能し、使用されているかについては、AWS についての不正グラフを参照してください。

- ソーシャルネットワーキング — グラフデータベースが使用される最初の最も一般的な領域の 1 つは、ソーシャルネットワーキングアプリケーションです。

  たとえば、ウェブサイトにソーシャルフィードを構築するとします。バックエンドのグラフデータベースを使用して、家族、友人、アップデートに「いいね」した人、近くに住む人々からの最新のアップデートの反映結果をユーザーに提供できます。

- ルート案内 — グラフは、現在の道路状況と一般的な混雑パターンを考慮して、始点から目的地までの最適なルートを見つけるのに役立ちます。

- ロジスティクス – グラフは、お客様の要件を満たすために利用可能な出荷および配送リソースを使用する最も効率的な方法を特定するのに役立ちます。

- 診断機能 — グラフは、観測された問題や障害の原因を特定するために照会できる複雑な診断ツリーを表すことができます。

- 科学研究 – グラフデータベースでは、保存時の暗号化を使用して、科学データと機密医療情報を保存およびナビゲートするアプリケーションを構築できます。たとえば、疾患と遺伝子の相互関係モデルを保存できます。タンパク質の経路内でグラフパターンを検索して、疾患と関連する可能性がある他の遺伝子を見つけることができます。化学物質をグラフとしてモデル化し、分子構造のパターンに対してクエリを実行できます。さまざまなシステムで医療記録の患者データを関連付けることができます。題目別に研究出版物を整理して関連する情報をすばやく見つけることができます。

- 規制 - 複雑な規制要件をグラフとして保存し、それらを照会して、日常業務に適用される可能性のある状況を検出できます。

- ネットワークトポロジとイベント – グラフデータベースは、IT ネットワークの管理と保護に役立ちます。ネットワークトポロジをグラフとして保存すると、ネットワーク上でさまざまな種類のイベントを保存して処理することもできます。特定のアプリケーションを実行しているホストの数などの質問に答えることができます。特定のホストが悪意のあるプログラムによって侵害された可能性を示すパターンを照会し、そのホストをダウンロードした元のホストのプログラムのトレースに役立つ接続データを照会できます。

グラフのクエリはどのように行いますか?

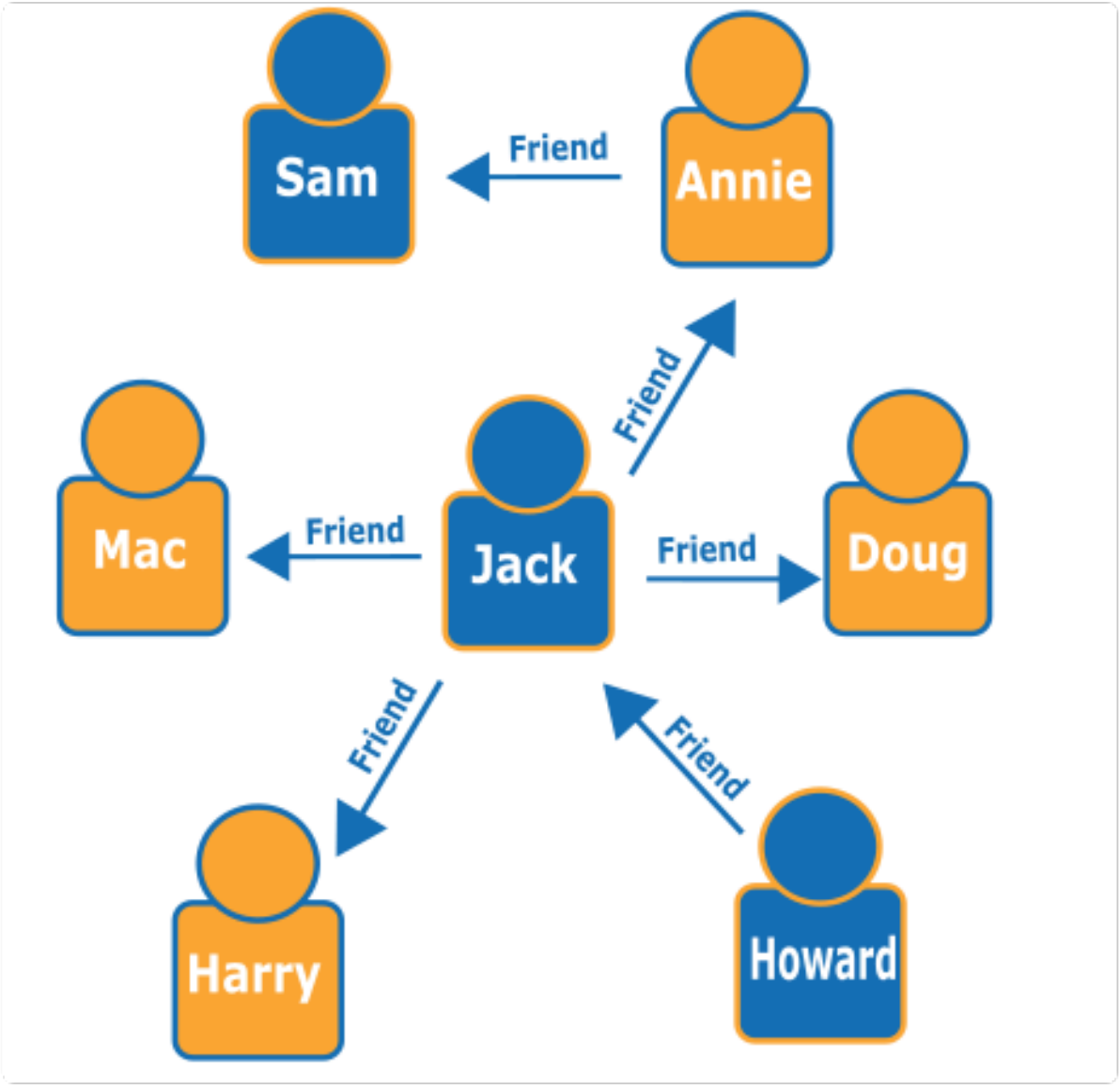
Neptune は、さまざまな種類のグラフデータをクエリするために設計された 3 つの特殊用途のクエリ言語をサポートしています。これらの言語を使用して、Neptune グラフデータベース内のデータを追加、変更、削除、およびクエリできます。

- Gremlin は、プロパティグラフのグラフトラバーサル言語です。Gremlin のクエリは個別のステップで構成されたトラバーサルで、各ステップはエッジからノードに従います。詳細については、Apache TinkerPop3 の Gremlin ドキュメントを参照してください。

  Gremlin の Neptune 実装は、特に Gremlin-Groovy (シリアル化されたテキストとして送信される Gremlin クエリ) を使用している場合、他の実装とはいくつかの相違点があります。詳細については、「Amazon Neptune の Gremlin 標準への準拠」を参照してください。
- openCypher – openCypher は、プロパティグラフの宣言型クエリ言語です。当初は Neo4jが開発し、その後2015年にオープンソース化され、Apache 2 オープンソースライセンスの下で openCypher プロジェクトで活用されました。言語仕様については、Cypher クエリ言語リファレンス (バージョン 9)を、さらに詳細については、Cypher スタイルガイドをご覧ください。
- SPARQL は、RDF データ用の宣言型クエリ言語です。World Wide Web Consortium (W3C) により標準化され、「SPARQL 1.1 概要」と SPARQL 1.1 クエリ言語」仕様で記述されているグラフパターンマッチングに基づいています。SPARQL の Neptune 実装に関する具体的な詳細については、「Amazon Neptune の SPARQL 標準準拠」を参照してください。

合致する Gremlin および SPARQL クエリの例

人物 (ノード) とその関係 (エッジ) の以下のグラフが与えられると、特定の人物の「友達の友達」(Howard の友達の友達など) が誰かを調べることができます。



グラフを見ると、Howard には Jack という 1 人の友達がいて、Jack には Annie、Harry、Doug および Mac という 4 人の友達がいることがわかります。これは簡単なグラフを使用した単純な例ですが、これらのタイプのクエリは複雑性、データセット、および結果のサイズにおいてスケーリングできます。

以下は、Howard の友達の友達の名前を返す Gremlin トラバーサルクエリです。

```
g.V().has('name', 'Howard').out('friend').out('friend').values('name')
```

以下は、Howard の友達の友達の名前を返す SPARQL クエリです。

```
prefix : <#>

  select ?names where {
    ?howard :name "Howard" .
    ?howard :friend/:friend/:name ?names .
  }
```

> **Note**
>
> リソース記述フレームワーク (RDF) トリプルの各要素には、それに関連付けられた URI があります。↑の例では、URI プレフィックスは意図的に短くしています。

Amazon Neptune の使用に関するオンラインコースを受講する

動画で学習したい人のために、AWS は、AWS Online Tech Talks でオンラインコースを提供しています。グラフデータベースを紹介するのは以下のとおりです。

Amazon Neptune によるグラフデータベースの紹介、詳細説明、デモ

グラフのリファレンスアーキテクチャを深く掘り下げる

グラフデータベースが解決できる問題と、それにどのようにアプローチするかについて考えるとき、まず最初に取り掛かるものの 1 つは、Neptune グラフリファレンスアーキテクチャ GitHub プロジェクトです。

ここでは、グラフワークロードタイプの詳細な説明と、効果的なグラフデータベースの設計に役立つ 3 つのセクションがあります。

- データモデルとクエリ言語 — このセクションでは、Gremlin と SPARQL の違いと、どちらを選ぶかについて説明します。
- グラフデータモデリング — これは、Gremlin を使用したプロパティグラフモデリングや、SPARQL を使用したRDFモデリングの詳細なウォークスルーなど、グラフデータモデリングの決定方法に関して徹底的に説明しています。
- 他のデータモデルをグラフモデルに変換する — ここでは、リレーショナルデータモデルをグラフモデルに変換する方法について説明します。

また、Neptune を使用するための具体的な手順を説明する 3 つのセクションがあります。

- Neptune VPC 外のクライアントから Amazon Neptune に接続する — このセクションでは、DB クラスターがある VPC の外部から Neptune に接続するためのいくつかのオプションを示します。
- AWSLambda 関数から Amazon Neptune へのアクセス — ここでは、Lambda 関数から Neptune に確実に接続する方法を説明します。
- Amazon Kinesis データストリームから Amazon Neptune に書き込む — このセクションは、Neptune で高い書き込みスループットシナリオを処理するのに役立ちます。

Neptune グラフノートブックを使用して、すぐに使用を開始する

Neptune グラフを操作するために Neptune グラフノートブックを使用する必要はなく、必要な場合は、AWS CloudFormation テンプレートを使用して新しい Neptune データベースをすぐに作成できます。

同時に、グラフを初めて使用する場合や、学習して試してみたい場合、または、経験豊富でクエリを改良したい場合にも、Neptune ワークベンチは、グラフアプリケーションを構築する際の生産性を向上させるインタラクティブ開発環境 (IDE) を提供します。

Neptune は、 のオープンソース Neptune グラフJupyterLabノートブックプロジェクト GitHubと Neptune ワークベンチで Jupyter とノートブックを提供します。 https://github.com/aws/graph-notebookこれらのノートブックには、グラフテクノロジーと Neptune について学ぶことができるインタラクティブなコーディング環境でサンプルアプリケーションチュートリアルとコードスニペットが用意されています。これらを使用して、異なるクエリ言語、異なるデータセット、およびバックエンド上の異なるデータベースを使用して、グラフの設定、構成、入力、およびクエリを実行できます。

これらのノートブックはいくつかの方法でホストできます。

- Neptune ワークベンチを使用すると、Amazon でホストされているフルマネージド環境で Jupyter Notebook を実行し SageMaker、Neptune グラフノートブックプロジェクトの最新リリースを自動的にロードできます。新しい Neptune データベースを作成するときに、Neptune コンソールでワークベンチを簡単にセットアップできます。

> **Note**
>
> Neptune ノートブックインスタンスを作成する場合、ネットワークアクセスには、Amazon 経由の直接アクセス SageMaker（ デフォルト) と VPC 経由のアクセスの 2

つのオプションが用意されています。どちらのオプションでも、ノートブックは Neptune ワークベンチをインストールするためのパッケージの依存関係を取得するためにインターネットにアクセスする必要があります。インターネットにアクセスできないと、Neptune ノートブックインスタンスの作成が失敗します。

- Jupyter をローカルにインストールすることもできます。これにより、Neptune またはオープンソースのグラフデータベースのローカルインスタンスに接続されたラップトップからノートブックを実行することもできます。後者の場合、無料でグラフテクノロジーを好きなだけ試してみることができます。準備ができたら、Neptune が提供するマネージドプロダクション環境にスムーズに移行できます。

Neptune ワークベンチを使用して Neptune ノートブックをホストする

Neptune は、1 時間あたり 0.10 ドル未満で開始できる T3 および T4g インスタンスタイプを提供します。Neptune の請求とは別に SageMaker、Amazon を通じてワークベンチリソースの料金が請求されます。Neptune の料金ページをご覧ください。Neptune ワークベンチで作成された Jupyter と JupyterLab ノートブックはすべて Amazon Linux 2 および JupyterLab 3 環境を使用します。JupyterLab ノートブックのサポートの詳細については、「Amazon SageMaker ドキュメント」を参照してください。

Jupyter または JupyterLab ノートブックは、次の 2 つの方法のいずれか AWS Management Console で の Neptune ワークベンチを使用して作成できます。

- 新しい Neptune DB クラスターを作成するときは、[ノートブック設定] メニューを使用してください。このためには、「AWS Management Console を使用して Neptune DB クラスターを起動する」のステップに従います。
- DB クラスターが作成された後、左側のナビゲーションペインの [ノートブック] メニューを使用してください。そのためには、以下の手順を実行します。

JupyterLab ノートブックメニューを使用して Jupyter またはノートブックを作成するには

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/neptune/home で Amazon Neptune コンソールを開きます。
2. 左側のナビゲーションペインで、[Notebooks (ノートブック)] を選択します。
3. [Create notebook (ノートブックの作成)] を選択します。

4. [Cluster] (クラスター) リストで、Neptune DB クラスターを選択します。DB クラスターをまだ作成していない場合は、[Create cluster (クラスターの作成)] を選択して作成します。
5. [ノートブックインスタンスタイプ] を選択します。
6. ノートブックに名前を付け、必要に応じて説明を入力します。
7. ノートブック用の AWS Identity and Access Management (IAM) ロールを既に作成していない限り、IAM ロール の作成 を選択し、IAM ロール名を入力します。

   > **Note**
   >
   > 以前のノートブック用に作成した IAM ロールを再利用する場合、ロールポリシーに、使用している Neptune DB クラスターにアクセスするための正しいアクセス許可が含まれている必要があります。これを確認するには、neptune-db:* アクションのリソース ARN 内のコンポーネントがそのクラスターと一致することを確認します。アクセス許可の設定を誤ると、notebook magic コマンドを実行しようとすると接続エラーが発生します。

8. [Create notebook (ノートブックの作成)] を選択します。作成プロセスでは、すべての準備が整うまでに 5 ~ 10 分かかる場合があります。
9. ノートブックを作成したら、ノートブックを選択し、Jupyter を開く または を開く JupyterLabを選択します。

コンソールでは、ノートブックの AWS Identity and Access Management (IAM) ロールを作成することも、自分でロールを作成することもできます。このロールのポリシーには、次のものを含める必要があります。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::aws-neptune-notebook-(AWS region)",
        "arn:aws:s3:::aws-neptune-notebook-(AWS region)/*"
      ]
```

```
    },
    {
      "Effect": "Allow",
      "Action": "neptune-db:*",
      "Resource": [
        "arn:aws:neptune-db:(AWS region):(AWS account ID):(Neptune resource ID)/*"
      ]
    }
  ]
}
```

上記のポリシーの 2 番目のステートメントには、1 つ以上の Neptune クラスターリソース ID がリストされていることに注意してください。

また、ロールは次の信頼関係を確立する必要があります。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sagemaker.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

繰り返しますが、すべての準備ができるまで、5 ~ 10 分かかることがあります。

「Neptune ML 用の Neptune ノートブックの手動設定」で説明されているように、新しいノートブックを Neptune ML と連携するように設定できます。

Python を使用して汎用 SageMaker ノートブックを Neptune に接続する

Neptune マジックをインストールしている場合、ノートブックを Neptune に接続するのは簡単ですが、Neptune SageMaker ノートブックを使用していない場合でも、Python を使用してノートブックを Neptune に接続することもできます。

SageMaker ノートブックセルで Neptune に接続する手順

1. Gremlin Python クライアントをインストールします。

```
!pip install gremlinpython
```

Neptune ノートブックは Gremlin Python クライアントをインストールするため、このステップはプレーン SageMaker ノートブックを使用している場合にのみ必要です。

2. Gremlin クエリを接続して発行するには、次のようなコードを記述します。

```
from gremlin_python import statics
from gremlin_python.structure.graph import Graph
from gremlin_python.process.graph_traversal import __
from gremlin_python.process.strategies import *
from gremlin_python.driver.driver_remote_connection import DriverRemoteConnection
from gremlin_python.driver.aiohttp.transport import AiohttpTransport
from gremlin_python.process.traversal import *
import os

port = 8182
server = '(your server endpoint)'

endpoint = f'wss://{server}:{port}/gremlin'

graph=Graph()

connection = DriverRemoteConnection(endpoint,'g',

 transport_factory=lambda:AiohttpTransport(call_from_event_loop=True))

g = graph.traversal().withRemote(connection)

results = (g.V().hasLabel('airport')
                .sample(10)
                .order()
                .by('code')
                .local(__.values('code','city').fold())
                .toList())

# Print the results in a tabular form with a row index
for i,c in enumerate(results,1):
    print("%3d %4s %s" % (i,c[0],c[1]))

connection.close()
```

> **Note**
>
> 3.5.0 より古いバージョンの Gremlin Python クライアントを使用している場合は、次の行を使用します。
>
> ```
> connection = DriverRemoteConnection(endpoint,'g',
>
>  transport_factory=lambda:AiohttpTransport(call_from_event_loop=True))
> ```
>
> ちょうど次のようになります。
>
> ```
> connection = DriverRemoteConnection(endpoint,'g')
> ```

Neptune ノートブックでの CloudWatch ログの有効化

CloudWatch Neptune Notebooks の ログがデフォルトで有効になりました。 CloudWatch ログを生成していない古いノートブックがある場合は、次のステップに従って手動で有効にします。

1. にサインイン AWS Management Console し、SageMaker コンソール を開きます。
2. 左側のナビゲーションペインで [ノートブック] を選択し、[ノートブックインスタンス] を選択します。ログを有効にする Neptune ノートブックの名前を探します。
3. そのノートブックインスタンスの名前を選択して、詳細ページに移動します。
4. ノートブックインスタンスが実行中の場合は、ノートブックの詳細ページの右上にある [停止] ボタンを選択します。
5. [アクセス許可と暗号化] に、IAM ロール ARN のフィールドがあります。このフィールドのリンクを選択して、このノートブックインスタンスが実行される IAM ロールに移動します。
6. 以下のポリシーを作成します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogDelivery",
        "logs:CreateLogGroup",
```

```
        "logs:CreateLogStream",
        "logs:DeleteLogDelivery",
        "logs:Describe*",
        "logs:GetLogDelivery",
        "logs:GetLogEvents",
        "logs:ListLogDeliveries",
        "logs:PutLogEvents",
        "logs:PutResourcePolicy",
        "logs:UpdateLogDelivery"
      ],
      "Resource": "*"
    }
  ]
}
```

7. この新しいポリシーを保存し、ステップ 4 で確認した IAM ロールにアタッチします。
8. SageMaker ノートブックインスタンスの詳細ページの右上にある開始をクリックします。
9. ログが流れ始めると、詳細ページの [ノートブックインスタンス設定] セクションの左下にある [ライフサイクル設定] というラベルの付いたフィールドの下に [ログを表示] リンクが表示されます。

ノートブックの起動に失敗すると、 SageMaker コンソールのノートブックの詳細ページの から、ノートブックインスタンスが起動するまでに 5 分以上かかったことを示すメッセージが表示されます。この問題に関連する CloudWatch ログは、次の名前で確認できます。

```
(your-notebook-name)/LifecycleConfigOnStart
```

ローカルマシンでのグラフノートブックの設定

グラフノートブックプロジェクトには、ローカルマシンで Neptune ノートブックを設定する手順が記載されています。

- 前提条件
- Jupyter と JupyterLab インストール
- グラフデータベースへの接続

ローカルノートブックは、Neptune DB クラスター、またはオープンソースのグラフデータベースのローカルインスタンスまたはリモートインスタンスに接続できます。

Neptune ノートブックを Neptune クラスターで使用する

バックエンドで Neptune クラスターに接続する場合は、Amazon でノートブックを実行することをお勧めします SageMaker。から Neptune に接続すると、ノートブックのローカルインストールよりも SageMaker 便利になり、Neptune ML をより簡単に操作できるようになります。

でノートブックを設定する方法については SageMaker、「Amazon を使用したグラフノートブックの起動 SageMaker」を参照してください。

Neptune 自体の設定およびセットアップ方法については、Neptune の設定 を参照してください。

Neptune ノートブックのローカルインストールを Neptune DB クラスターに接続することもできます。Amazon Neptune DB クラスターは、外部から隔離された Amazon Virtual Private Cloud (VPC) でのみ作成できるため、これはやや複雑になる可能性があります。VPC を外部から VPC に接続するには、さまざまな方法があります。1 つは、ロードバランサーを使用することです。もう 1 つは VPC ピアリングを使用する方法です (Amazon Virtual Private Cloud ピアリングガイドを参照)。

ただし、ほとんどの人にとって最も便利な方法は、接続して VPC 内に Amazon EC2 プロキシサーバーをセットアップし、SSH トンネリング (ポートフォワーディングとも呼ばれます) を使い、接続します。設定方法については、グラフノートブックプロジェクトの フォルダにある「グラフノートブックを Amazon Neptune にローカル接続する」を参照してください。 `additional-databases/neptune` https://github.com/aws/graph-notebook/ GitHub

Neptune ノートブックをオープンソースのグラフデータベースで使用する

グラフテクノロジーを無償で開始するには、バックエンドでさまざまなオープンソースデータベースを備えた Neptune ノートブックを使用することもできます。例としては、 TinkerPop Gremlin サーバー や Blazegraph データベースがあります。

Gremlin Server をバックエンドデータベースとして使用するには、次の手順に従います。

- グラフノートブックを Gremlin Server フォルダに接続する GitHub 。
- グラフノートブックの Gremlin 設定 GitHub フォルダ。

バックエンドデータベースとして Blazegraph のローカルインスタンスを使用するには、次の手順に従ってください。

- Blazegraph クイックスタート指示
- グラフノートブック Blazegraph 設定 GitHub フォルダ。

Neptune ノートブックを Jupyter から JupyterLab 3 に移行する

2022 年 12 月 21 日より前に作成された Neptune ノートブックは Amazon Linux 1 環境を使用しています。この AWS ブログ記事「Amazon Linux 2 で Amazon Notebook インスタンスに作業を移行する」で説明されているステップを実行することで、その日付より前に作成された古い Jupyter Notebooks を JupyterLab 3 で新しい Amazon Linux 2 環境に移行できます。 SageMaker

さらに、Neptune ノートブックを新しい環境に移行する場合に特に適用される手順が他にもいくつかあります。

Neptune 固有の前提条件

ソース Neptune ノートブックの IAM ロールに、以下のすべてのアクセス許可を追加します。

```
{
  "Effect": "Allow",
  "Action": [
    "s3:GetObject",
    "s3:ListBucket",
    "s3:CreateBucket",
    "s3:PutObject"
  ],
  "Resource": [
    "arn:aws:s3:::(your ebs backup bucket name)",
    "arn:aws:s3:::(your ebs backup bucket name)/*"
  ]
},
{
  "Effect": "Allow",
  "Action": [
    "sagemaker:ListTags"
  ],
  "Resource": [
    "*"
  ]
}
```

バックアップに使用する S3 バケットの正しい ARN を必ず指定してください。

Neptune 固有のライフサイクル設定

ブログ記事で説明されているように、(on-create.sh から) バックアップを復元するための 2 つ目のライフサイクル設定スクリプトを作成する場合、ライフサイクル名は aws-neptune-* 形式に従う必要があります (aws-neptune-sync-from-s3 など)。これにより、Neptune コンソールでノートブックを作成する際に LCC を選択できるようになります。

スナップショットから新しいインスタンスへの Neptune 固有の同期

スナップショットから新しいインスタンスへの同期に関するブログ記事で説明されている手順には、Neptune 固有の変更点があります。

- ステップ 4 で、notebook-al2-v2 を選択します。
- ステップ 5 で、ソース Neptune ノートブックの IAM ロールを再利用します。
- ステップ 7 と 8 の間:
  - [ノートブックインスタンス設定] で、aws-neptune-* 形式を使用する名前を設定します。
  - [ネットワーク] 設定アコーディオンを開き、ソースノートブックと同じ VPC、サブネット、およびセキュリティグループを選択します。

新しいノートブックが作成された後の Neptune 固有のステップ

1. ノートブックの [Jupyter を開く] ボタンを選択します。メインディレクトリに SYNC_COMPLETE ファイルが表示されたら、次のステップに進みます。
2. SageMaker コンソールのノートブックインスタンスページに移動します。
3. ノートブックを停止します。
4. [Edit] (編集) を選択します。
5. ノートブックインスタンス設定で、ソース Neptune ノートブックの元のライフサイクルを選択して、[ライフサイクル設定] フィールドを編集します。これは EBS バックアップライフサイクルではないことに注意してください。
6. [ノートブック設定の更新] を選択します。
7. ノートブックを再起動します。

ブログ記事で説明されている手順にここで説明する変更を加えると、グラフノートブックを Amazon Linux 2 および JupyterLab 3 環境を使用する新しい Neptune ノートブックインスタンスに移行できるようになります。の Neptune ページでアクセスと管理のために表示され AWS Management

Console、オープン Jupyter またはオープン を選択して、中断した場所から作業を続行できるようになりました JupyterLab。

ノートブックで Neptune ワークベンチマジックを使う

Neptune ワークベンチは、いわゆるノートブックのマジックコマンドをいくつか提供し、これを使用すると、時間と労力を大幅に節約できます。これらは、ラインマジックとセルマジックの 2 つに分類されます。

ラインマジックは、コマンドの前にパーセント記号 (%) が 1 つ付いています。行入力のみを取得し、セル本体の残りの部分からの入力は取得しません。Neptune ワークベンチは、次のラインマジックを提供します。

- %seed
- %load
- %load_ids
- %load_status
- %cancel_load
- %status
- %gremlin_status
- %opencypher_status、または %oc_status
- %stream_viewer
- %sparql_status
- %graph_notebook_config
- %graph_notebook_host
- %graph_notebook_version
- %graph_notebook_vis_options
- %statistics
- %summary

セルマジックは、コマンドの前にパーセント記号 (%%) が 1 つではなく、2 つ付いています。セルの内容を入力として取得しますが、行の内容を入力として取得することもできます。Neptune ワークベンチは、次のセルマジックを提供します。

- %%sparql
- %%gremlin
- %%opencypher、または %%oc
- %%graph_notebook_config
- %%graph_notebook_vis_options

また、ラインマジックとセルマジックの 2 つのマジックは、Neptune の機械学習 と一緒に使用できます。

- %neptune_ml
- %%neptune_ml

> **Note**
>
> Neptune のマジックを使って作業する場合、通常、ヘルプテキストは `--help` または `-h` パラメータを使用しています。セルマジックでは、本文を空にすることはできません。そのため、ヘルプを得るときは、フィラーテキスト、つまり 1 文字でも本文に入れてください。例:
>
> ```
> %%gremlin --help
>   x
> ```

セルまたはラインマジックへの変数のインジェクション

ノートブックで定義された変数は、`${VAR_NAME}` の形式を使用して、ノートブック内の任意のセルまたはラインマジック内で参照できます。

例えば、以下の変数を定義するとします。

```
c = 'code'
my_edge_labels = '{"route":"dist"}'
```

その場合、セルマジック内の次の Gremlin クエリは次のようになります。

```
%%gremlin -de $my_edge_labels
g.V().has('${c}','SAF').out('route').values('${c}')
```

これは次に相当します。

```
%%gremlin -de {"route":"dist"}
g.V().has('code','SAF').out('route').values('code')
```

すべてのクエリ言語で機能するクエリ引数

以下のクエリ引数は、Neptune ワークベンチの `%%gremlin`、`%%opencypher`、および `%%sparql` マジックで動作します。

一般的なクエリ引数

- **`--store-to`** (または **`-s`**) – クエリ結果を格納する変数の名前を指定します。
- **`--silent`** — 存在する場合、クエリの完了後に出力は表示されません。

- **`--group-by`** (または **`-g`**) - ノードのグループ化に使用されるプロパティを指定します (`code` または `T.region` など)。頂点は割り当てられたグループに基づいて色分けされます。
- **`--ignore-groups`** - 存在する場合、すべてのグループ化オプションは無視されます。
- **`--display-property`** (または **`-d`**) - 各頂点に値が表示されるプロパティを指定します。

  各クエリ言語のデフォルト値は次のとおりです。

  - Gremlin の場合: `T.label`。
  - openCypher の場合: `~labels`。
  - SPARQL の場合: `type`。
- **`--edge-display-property`** (または **`-t`**) – 各エッジに値が表示されるプロパティを指定します。

  各クエリ言語のデフォルト値は次のとおりです。

  - Gremlin の場合: `T.label`。
  - openCypher の場合: `~labels`。
  - SPARQL の場合: `type`。
- **`--tooltip-property`**(または **`-de`**) — 各ノードのツールチップとして値が表示されるプロパティを指定します。

  各クエリ言語のデフォルト値は次のとおりです。

  - Gremlin の場合: `T.label`。

  - openCypher の場合: `~labels`。
  - SPARQL の場合: `type`。
- **`--edge-tooltip-property`** (または **`-te`**) – 各エッジのツールチップとして値が表示されるプロパティを指定します。

  各クエリ言語のデフォルト値は次のとおりです。

  - Gremlin の場合: `T.label`。
  - openCypher の場合: `~labels`。
  - SPARQL の場合: `type`。
- **`--label-max-length`** (または **`-l`**) — 任意の頂点ラベルの最大文字長を指定します。デフォルトは 10 です。
- **`--edge-label-max-length`** (または **`-le`**) — 任意のエッジラベルの最大文字長を指定します。デフォルトは 10 です。

  openCypher の場合のみ、これは `--rel-label-max-length` または `-rel` です。
- **`--simulation-duration`** (または **`-sd`**) — ビジュアライゼーション物理シミュレーションの最大時間を指定します。デフォルトは 1500 ms です。
- **`--stop-physics`** (または **`-sp`**) — 最初のシミュレーションが安定したら、ビジュアライゼーション物理を無効にします。

これらの引数のプロパティ値は、1 つのプロパティキーで構成されることも、ラベルタイプごとに異なるプロパティを指定できる JSON 文字列で構成されることもあります。JSON 文字列は、変数インジェクションを使用してのみ指定できます。

%seed ラインマジック

`%seed` ラインマジックは、Gremlin、openCypher、または SPARQL クエリを探索して実験するために使用できる Neptune エンドポイントにデータを追加する便利な方法です。探索するデータモデル (プロパティグラフまたは RDF) を選択し、Neptune が提供する多数の異なるサンプルデータセットから選択できるフォームを提供します。

%load ラインマジック

`%load` ラインマジックは Neptune に一括ロードリクエストを送信するために使用できるフォームを生成します (Neptune ローダーコマンド を参照)。ソースは、Neptune クラスターと同じリージョンにある Amazon S3 パスである必要があります。

%load_ids ラインマジック

%load_ids ラインマジックは、ノートブックのホストエンドポイントに送信されたロード ID を取得します (Neptune Loader Get-Status リクエストパラメータ を参照)。リクエストは以下のような形式です。

```
GET https://your-neptune-endpoint:port/loader
```

%load_status ラインマジック

%load_status ラインマジックは、ノートブックのホストエンドポイントに送信された特定のロードジョブのロードステータスを取得します。これは、行入力によって指定されます (Neptune Loader Get-Status リクエストパラメータ を参照)。リクエストは以下のような形式です。

```
GET https://your-neptune-endpoint:port/loader?loadId=loadId
```

ラインマジックは次のようになります。

```
%load_status load id
```

%cancel_load ラインマジック

%cancel_load ラインマジックは特定のロードジョブをキャンセルします (Neptune Loader Cancel Job を参照)。リクエストは以下のような形式です。

```
DELETE https://your-neptune-endpoint:port/loader?loadId=loadId
```

ラインマジックは次のようになります。

```
%cancel_load load id
```

%status ラインマジック

ノートブックのホストエンドポイントからステータス情報を取得します (%graph_notebook_config はホストエンドポイントを表示します)。

%gremlin_status ラインマジック

Gremlin クエリステータス情報を取得します。

%opencypher_status ラインマジック (%oc_status)

opencypher クエリのクエリステータスを取得します。コマンドは次のオプションの引数を取得します。

- **--queryId** または **-q** - ステータスを表示する特定の実行中のクエリの ID を指定します。
- **--cancel_query** または **-c** - 実行中のクエリをキャンセルします。値を取得しません。
- **--silent** または **-s** - クエリをキャンセルする際に `--silent` が `true` に設定される場合、実行中のクエリは、HTTP レスポンスコード `200` でキャンセルされます。それ以外の場合、HTTP レスポンスコードは `500` になります。
- **--store-to** - クエリ結果を保存する変数の名前を指定します。

%sparql_status ラインマジック

SPARQL クエリのステータス情報を取得します。

%stream_viewer ラインマジック

`%stream_viewer` ラインマジックは、Neptune クラスターでストリームが有効になっている場合、Neptune ストリームに記録されたエントリをインタラクティブに探索できるインターフェースを表示します。これは、次のオプション引数を受け入れます。

- **language** — ストリームデータのクエリ言語 (`gremlin` または `sparql`)。この引数を指定しなかった場合のデフォルトは `gremlin` です。
- **--limit** — ページあたりに表示するストリームエントリの最大数を指定します。この引数を指定しなかった場合のデフォルト値は `10` です。

> **Note**
>
> `%stream_viewer` ラインマジックは、エンジンバージョン 1.0.5.1 以前でのみ完全にサポートされています。

%graph_notebook_config ラインマジック

このラインマジックは、ノートブックが Neptune との通信に使用している設定を含む JSON オブジェクトを表示します。以下の設定ファイルがあります。

- host: コマンドを接続して発行するエンドポイント。
- port: Neptune にコマンドを発行するときに使用するポート。デフォルト値は 8182 です。
- auth_mode: Neptune にコマンドを発行するときに使用する認証のモード。IAM 認証が有効になっているクラスターに接続する場合は IAM、そうでない場合は DEFAULT の必要があります。
- load_from_s3_arn: 使用する %load マジックに対して Amazon S3 ARN を指定します。この値が空の場合、ARN は %load コマンドで指定する必要があります。
- ssl: TLS を使用して Neptune に接続するかどうかを示すブール値。デフォルト値は、trueです。
- aws_region: このノートブックが展開されているリージョン。この情報は IAM 認証と、%load リクエストに関して使用されます。

設定を変更するには、新しいセルに %graph_notebook_config 出力をコピーし、そこに変更を加えます。次に、新しいセルで %%graph_notebook_config セルマジックを実行する場合、それに応じて設定が変更されます。

%graph_notebook_host ラインマジック

行入力をノートブックのホストとして設定します。

%graph_notebook_version ラインマジック

%graph_notebook_version ラインマジックは Neptune ワークベンチノートブックのリリース番号を返します。例えば、グラフのビジュアライゼーションがバージョン 1.27 で導入されました。

%graph_notebook_vis_options ラインマジック

%graph_notebook_vis_options ラインマジックは、ノートブックが使用している現在のビジュアライゼーション設定を表示します。これらのオプションについては、vis.js ドキュメント内で説明されています。

これらの設定を変更するには、出力を新しいセルにコピーし、必要な変更を行い、そのセル上で %%graph_notebook_vis_options セルマジックを実行します。

ビジュアライゼーション設定をデフォルト値に戻すには、%graph_notebook_vis_options ラインマジックを reset パラメータとともに実行します。これにより、すべてのビジュアライゼーション設定がリセットされます。

```
%graph_notebook_vis_options reset
```

%statistics ラインマジック

%statistics ラインマジックは、DFE エンジン統計の取得や管理に使用されます (「Neptune DFE が使用する統計情報の管理」を参照)。このマジックはグラフのサマリーを取得するのにも使用できます。

次のパラメータを受け入れます。

- **--language** — 統計エンドポイントのクエリ言語。propertygraph (またはpg) または rdf。

  指定しなかった場合のデフォルトは propertygraph です。
- **--mode** (または **-m**) — 送信するリクエストまたはアクションのタイプを指定します (status、disableAutoCompute、enableAutoCompute、refresh、delete、detailed、または basic のいずれか)。

  指定しなかった場合のデフォルトは status です。ただし、--summary が指定された場合のデフォルトは basic です。
- **--summary** — 選択さえた言語の統計サマリーエンドポイントからグラフのサマリーを取得します。
- **--silent** — 存在する場合、クエリの完了後に出力は表示されません。
- **--store-to** - クエリ結果の保存先となる変数を指定するために使用されます。

%summary ラインマジック

%summary ラインマジックは、グラフのサマリー情報を取得するために使用されます。Neptune エンジンバージョン 1.2.1.0 以降で利用可能です。

次のパラメータを受け入れます。

- **--language** — 統計エンドポイントのクエリ言語。propertygraph (またはpg) または rdf。

  指定しなかった場合のデフォルトは propertygraph です。

- **--detailed** — 出力での構造体フィールドの表示のオンとオフを切り替えます。

  指定しなかった場合のデフォルトは、basic サマリー表示モードです。
- **--silent** — 存在する場合、クエリの完了後に出力は表示されません。
- **--store-to** - クエリ結果の保存先となる変数を指定するために使用されます。

%%graph_notebook_config セルマジック

%%graph_notebook_config セルマジックは、可能であれば、ノートブックが Neptune との通信に使用している設定を変更するために、構成情報を含む JSON オブジェクトを使用します。構成は %graph_notebook_config ラインマジックによって返される形式と同じ形式になります。

例:

```
%%graph_notebook_config
{
  "host": "my-new-cluster-endpoint.amazon.com",
  "port": 8182,
  "auth_mode": "DEFAULT",
  "load_from_s3_arn": "",
  "ssl": true,
  "aws_region": "us-east-1"
}
```

%%sparql セルマジック

%%sparql セルマジックは、Neptune エンドポイントに SPARQL クエリを発行します。次のオプションの行入力を受け付けます。

- **-h** または **--help** - これらのパラメータに関するヘルプテキストを返します。
- **--path** - SPARQL エンドポイントへのパスをプレフィックスします。例えば、--path "abc/def" と指定した場合、呼び出されるエンドポイントは *host*:*port*/abc/def になります。
- **--expand-all** - これは、バインドの種類に関係なく、グラフダイアグラムに ?s ?p ?o 結果のすべてを含めるようにビジュアライザに指示するクエリビジュアライゼーションのヒントです。

  デフォルトでは、SPARQL の視覚化には、o? は uri または bnode (空白ノード) であるトリプルパターンのみを含みます。その他のリテラル文字列や整数などの ?o バインディング型は、すべてグラフタブ内の詳細ペインを使用して表示できる ?s ノードのプロパティとして処理されます。

ビジュアライゼーションに頂点などのリテラル値を含める必要がある場合の、代わりに `--expand-all` クエリヒントを使用します。

explain クエリは視覚化されないため、このビジュアライゼーションヒントを explain パラメータと組み合わせないでください。

- **`--explain-type`** - 使用する explain モードを指定するために使用します (`dynamic`、`static`、または `details` のいずれか)。
- **`--explain-format`** - explain クエリ (`text/csv` または `text/html` のいずれか 1 つ) の応答形式を指定するために使用します。
- --store-to - クエリ結果の保存先となる変数を指定するために使用します。

`explain` クエリの例は次のとおりです。

```
%%sparql explain

SELECT * WHERE {?s ?p ?o} LIMIT 10
```

`--expand-all` ビジュアライゼーションヒントパラメータを使用するビジュアライゼーションクエリの例 (SPARQL の視覚化 を参照)。

```
%%sparql --expand-all

SELECT * WHERE {?s ?p ?o} LIMIT 10
```

%%gremlin セルマジック

`%%gremlin` セルマジックは、 を使用して Neptune エンドポイントに Gremlin クエリを発行します WebSocket。これは、Gremlin `explain` /> モードまたは Gremlin `profile` API へトグルするオプションの行入力を受け入れます。また、ビジュアライゼーションの出力動作を変更するための個別のオプションのビジュアライゼーションヒント入力を受け入れます (Gremlin の視覚化 を参照)。

`explain` クエリの例は次のとおりです。

```
%%gremlin explain

g.V().limit(10)
```

profile クエリの例は次のとおりです。

```
%%gremlin profile

g.V().limit(10)
```

ビジュアライゼーションクエリヒントを使用するビジュアライゼーションクエリの例。

```
%%gremlin -p v,outv

g.V().out().limit(10)
```

%%gremlin profile クエリのオプションパラメータ

- **--chop** — プロファイル結果文字列の最大長を指定します。この引数を指定しなかった場合のデフォルト値は 250 です。
- **--serializer** — 結果に使用するシリアライザーを指定します。使用できる値は、有効な MIME タイプまたは TinkerPop ドライバーの「シリアライザー」列挙値のいずれかです。この引数を使用しない場合、デフォルト値は、application.json です。
- **--no-results** — 結果数のみを表示します。使用されなかった場合、デフォルトでは、すべてのクエリ結果がプロファイルレポートに表示されます。
- **--indexOps** — すべてのインデックス操作の詳細レポートを表示します。

%%opencypher セルマジック (%%oc)

%%opencypher セルマジック (%%oc フォームという略称あり) は、Neptune エンドポイントに openCypher クエリを発行します。次のオプションの行入力引かずを受け付けます。

- mode - query または bolt いずれかのクエリモード。この引数を使用しない場合、デフォルト値は、query です。
- **--group-by** または **-g** - ノードのグループ化に使用するプロパティを指定します。例えば、code, ~id です。この引数を使用しない場合、デフォルト値は、~labels です。
- **--ignore-groups** - 存在する場合、すべてのグループ化オプションは無視されます。
- **--display-propery** または **-d** - 各頂点の値を表示するプロパティを指定します。この引数を使用しない場合、デフォルト値は、~labels です。
- **--edge-display-propery** または **-de** – 各エッジの値を表示するプロパティを指定します。この引数を使用しない場合、デフォルト値は、~labels です。

- **--label-max-length** または **-l** - 表示する頂点ラベルの最大文字数を指定します。この引数を使用しない場合、デフォルト値は、10 です。
- **--store-to** または **-s** — クエリ結果を保存する変数の名前を指定します。
- **--plan-cache** または **-pc** — 使用するプランキャッシュモードを指定します。デフォルト値はです auto(*plan-cache は Neptune Analytics でのみ使用できます) 。
- **--query-timeout** または **-qt** — 最大クエリタイムアウトをミリ秒単位で指定します。デフォルト値は、1800000です。
- **--query-parameters** または **qp** — クエリに適用するパラメータ定義。このオプションでは、単一の変数名またはマップの文字列表現を使用できます。

  --query-parameters の使用例

  1. openCypher パラメータのマップをノートブックの 1 つのノートブックセルに定義します。

  ```
  params = '''{
   "name":"john",
   "age": 20,
  }'''
  ```

  2. %%oc を使用してパラメータを別のセルの --query-parameters に渡します。

  ```
  %%oc --query-parameters params

  MATCH (n {name: $name, age: $age})
  RETURN n
  ```

- --explain-type – 使用する説明モード (動的、静的、詳細のいずれか) を指定するために使用されます。

%%graph_notebook_vis_options セルマジック

%%graph_notebook_vis_options セルマジックでは、ノートブックのビジュアライゼーションオプションを設定できます。%graph-notebook-vis-options ラインマジックによって返された設定をコピーして新しいセルに並べ、それらに変更を加えて、%%graph_notebook_vis_options セルマジックで新しい値を設定します。

これらのオプションについては、vis.js ドキュメント内で説明されています。

ビジュアライゼーション設定をデフォルト値に戻すには、%graph_notebook_vis_options ラインマジックを reset パラメータとともに実行します。これにより、すべてのビジュアライゼーション設定がリセットされます。

```
%graph_notebook_vis_options reset
```

%neptune_ml ラインマジック

%neptune_ml ラインマジックで Neptune MLのさまざまな操作を開始および管理します。

> **Note**
>
> %%neptune_ml セルマジックでも Neptune MLのさまざまな操作を開始および管理できます。

- **%neptune_ml export start** - 新しいエクスポートジョブを開始します。

  パラメータ

  - **--export-url** *exporter-endpoint* - (オプション) エクスポータを呼び出すことができる Amazon API Gateway エンドポイント。
  - **--export-iam** - (オプション) エクスポート URL へのリクエストに SigV4 を使用して署名する必要があることを示すフラグ。
  - **--export-no-ssl** - (オプション) エクスポータに接続するときに SSL を使用しないことを示すフラグ。
  - **--wait** - (オプション) エクスポートが完了するまで操作を待機する必要があることを示すフラグ。
  - **--wait-interval** *interval-to-wait* – (オプション) エクスポートステータスチェックの間隔を秒単位で設定します (デフォルト: 60)。
  - **--wait-timeout** *timeout-seconds* - (オプション) 最新のステータスを返す前に、エクスポートジョブの完了を待機する時間 (秒単位) を設定します (デフォルト値: 3,600)。
  - **--store-to** *location-to-store-result* – (オプション) エクスポート結果を保存する変数。--wait を指定した場合、最終ステータスがそこに保存されます。
- **%neptune_ml export status** - エクスポートジョブのステータスを取得します。

パラメータ

- **`--job-id`** *`export job ID`* - ステータスを取得するエクスポートジョブの ID。
- **`--export-url`** *`exporter-endpoint`* - (オプション) エクスポータを呼び出すことができる Amazon API Gateway エンドポイント。
- **`--export-iam`** - (オプション) エクスポート URL へのリクエストに SigV4 を使用して署名する必要があることを示すフラグ。
- **`--export-no-ssl`** - (オプション) エクスポータに接続するときに SSL を使用しないことを示すフラグ。
- **`--wait`** - (オプション) エクスポートが完了するまで操作を待機する必要があることを示すフラグ。
- **`--wait-interval`** *`interval-to-wait`* – (オプション) エクスポートステータスチェックの間隔を秒単位で設定します (デフォルト: 60)。
- **`--wait-timeout`** *`timeout-seconds`* - (オプション) 最新のステータスを返す前に、エクスポートジョブの完了を待機する時間 (秒単位) を設定します (デフォルト値: 3,600)。
- **`--store-to`** *`location-to-store-result`* – (オプション) エクスポート結果を保存する変数。`--wait` を指定した場合、最終ステータスがそこに保存されます。

- **`%neptune_ml dataprocessing start`** - Neptune ML データ処理ステップを開始します。

  パラメータ

  - **`--job-id`** *`ID for this job`* - (オプション) このジョブに割り当てる ID。
  - **`--s3-input-uri`** *`S3 URI`* - (オプション) このデータ処理ジョブの入力を検索する S3 URI。
  - **`--config-file-name`** *`file name`* - (オプション) このデータ処理ジョブの構成ファイルの名前。
  - **`--store-to`** *`location-to-store-result`* – (オプション) データ処理結果を保存する変数。
  - **`--instance-type`** *`(#########)`* — (オプション) このデータ処理ジョブに使用するインスタンスサイズ。
  - **`--wait`** – (オプショナル) データ処理が完了するまで操作を待機する必要があることを示すフラグ。
  - **`--wait-interval`** *`interval-to-wait`* – (オプション) データ処理ステータスチェック間の時間を秒単位で設定します (デフォルト: 60)。
  - **`--wait-timeout`** *`timeout-seconds`* – (オプショナル) 最新のステータスを返す前に、データ処理ジョブの完了を待機する時間 (秒単位) を設定します (デフォルト値: 3,600)。

- **`%neptune_ml dataprocessing status`** - データ処理ジョブのステータスを取得します。

  パラメータ

  - **`--job-id`** *`ID of the job`* – ステータスを取得するジョブの ID。
  - **`--store-to`** *`instance type`* - (オプション) モデルトレーニング結果を保存する変数。
  - **`--wait`** – (オプショナル) モデルトレーニングが完了するまで操作を待機する必要があることを示すフラグ。
  - **`--wait-interval`** *`interval-to-wait`* – (オプション) モデルトレーニングステータスチェック間の時間を秒単位で設定します (デフォルト: 60)。
  - **`--wait-timeout`** *`timeout-seconds`* - (オプション) 最新のステータスを返す前に、データ処理ジョブの完了を待機する時間 (秒単位) を設定します (デフォルト値: 3,600)。
- **`%neptune_ml training start`** - Neptune ML モデルトレーニングプロセスを開始します。

  パラメータ

  - **`--job-id`** *`ID for this job`* - (オプション) このジョブに割り当てる ID。
  - **`--data-processing-id`** *`dataprocessing job ID`* - (オプション) トレーニングに使用するアーティファクトを作成したデータ処理ジョブの ID。
  - **`--s3-output-uri`** *`S3 URI`* - (オプション) このモデルトレーニングジョブからの出力を保存する S3 URI。
  - **`--instance-type`** *`(#########)`* - (オプション) このモデルトレーニングジョブに使用するインスタンスサイズ。
  - **`--store-to`** *`location-to-store-result`* – (オプション) モデルトレーニング結果を保存する変数。
  - **`--wait`** – (オプショナル) モデルトレーニングが完了するまで操作を待機する必要があることを示すフラグ。
  - **`--wait-interval`** *`interval-to-wait`* – (オプション) モデルトレーニングステータスチェック間の時間を秒単位で設定します (デフォルト: 60)。
  - **`--wait-timeout`** *`timeout-seconds`* - (オプション) 最新のステータスを返す前に、モデルトレーニングジョブの完了を待機する時間 (秒単位) を設定します。デフォルト値: 3,600)。
- **`%neptune_ml training status`** - Neptune ML モデルトレーニングジョブのステータスを取得します。

  パラメータ

  - **`--job-id`** *`ID of the job`* - ステータスを取得するジョブの ID。

- **`--store-to`** *`instance type`* - (オプション) ステータス結果を保存する変数。
- **`--wait`** - (オプション) モデルトレーニングが完了するまで操作を待機する必要があることを示すフラグ。
- **`--wait-interval`** *`interval-to-wait`* – (オプション) モデルトレーニングステータスチェック間の時間を秒単位で設定します (デフォルト: 60)。
- **`--wait-timeout`** *`timeout-seconds`* - (オプション) 最新のステータスを返す前に、データ処理ジョブの完了を待機する時間 (秒単位) を設定します (デフォルト値: 3,600)。

- **`%neptune_ml endpoint create`** - Neptune ML モデルのクエリエンドポイントを作成します。

  パラメータ

  - **`--job-id`** *`ID for this job`* - (オプション) このジョブに割り当てる ID。
  - **`--model-job-id`** *`model-training job ID`* - (オプション) クエリエンドポイントを作成するモデルトレーニングジョブの ID。
  - **`--instance-type`** *`(#########)`* - (オプション) クエリエンドポイントに使用するインスタンスサイズ。
  - **`--store-to`** *`location-to-store-result`* – (オプション) エンドポイント作成の結果を保存する変数。
  - **`--wait`** – (オプショナル) エンドポイント作成が完了するまで操作を待機する必要があることを示すフラグ。
  - **`--wait-interval`** *`interval-to-wait`* – (オプション) ステータスチェック間の時間を秒単位で設定します (デフォルト: 60)。
  - **`--wait-timeout`** *`timeout-seconds`* – (オプショナル) 最新のステータスを返す前に、エンドポイント作成ジョブの完了を待機する時間 (秒単位) を設定します (デフォルト値: 3,600)。

- **`%neptune_ml endpoint status`** - Neptune ML クエリエンドポイントのステータスを取得します。

  パラメータ

  - **`--job-id`** *`endpoint creation ID`* - (オプション) ステータスをレポートするエンドポイント作成ジョブの ID。
  - **`--store-to`** *`location-to-store-result`* – (オプション) ステータス結果を保存する変数。
  - **`--wait`** – (オプショナル) エンドポイント作成が完了するまで操作を待機する必要があることを示すフラグ。

- **`--wait-interval`** *`interval-to-wait`* – (オプション) ステータスチェック間の時間を秒単位で設定します (デフォルト: 60)。
- **`--wait-timeout`** *`timeout-seconds`* - (オプション) 最新のステータスを返す前に、エンドポイント作成ジョブの完了を待機する時間 (秒単位) を設定します (デフォルト値: 3,600)。

%%neptune_ml セルマジック

%%neptune_ml セルマジックは、--job-id または --export-url のような行入力を無視します。代わりに、これらの入力とセル本文内の他の入力を指定できます。

このような入力を Jupyter 変数に割り当てた別のセルに保存し、その変数を使用してセル本文に挿入することもできます。そうすれば、毎回再入力しなくても、この入力内容を何度も使用できます。

これは、注入変数がセルの唯一の内容である場合にのみ機能します。1 つのセルに複数の変数を使用したり、テキストと変数の組み合わせを使用したりすることはできません。

例えば、%%neptune_ml export start セルマジックは、Neptune エクスポートプロセスを制御するために使用されるパラメーター で説明されているすべてのパラメータを含む JSON ドキュメントをセル本文で消費できます。

Export the data and model configuration (データとモデルの設定をエクスポートする) セクションの機能の設定の下にある Neptune-ML-01-Introduction-to-Node-Classification-Gremlin ノートブックでは、次のセルが、export-params という名前の Jupyter 変数に割り当てられたドキュメント内のエクスポートパラメータをどのように保持しているかが説明されています。

```
export_params = {
  "command": "export-pg",
  "params": {
    "endpoint": neptune_ml.get_host(),
    "profile": "neptune_ml",
    "useIamAuth": neptune_ml.get_iam(),
    "cloneCluster": False
   },
  "outputS3Path": f'{s3_bucket_uri}/neptune-export',
  "additionalParams": {
    "neptune_ml": {
      "targets": [
        {
          "node": "movie",
          "property": "genre"
```

```
        }
      ],
      "features": [
        {
          "node": "movie",
          "property": "title",
          "type": "word2vec"
        },
        {
          "node": "user",
          "property": "age",
          "type": "bucket_numerical",
          "range" : [1, 100],
          "num_buckets": 10
        }
      ]
    }
  },
  "jobSize": "medium"}
```

このセルを実行すると、Jupyter はその名前でパラメータドキュメントを保存します。次に `${export_params}` を使用して、次のように JSON ドキュメントを `%%neptune_ml export start cell` の本文に挿入できます。

```
%%neptune_ml export start --export-url {neptune_ml.get_export_service_host()} --export-
iam --wait --store-to export_results

${export_params}
```

利用可能な **%%neptune_ml** セルマジックの形式

`%%neptune_ml` セルマジックは以下のような形式で使用できます。

- **`%%neptune_ml export start`** - Neptune ML エクスポートプロセスを開始します。
- **`%%neptune_ml dataprocessing start`** - Neptune ML データ処理ジョブを開始します。
- **`%%neptune_ml training start`** - Neptune ML モデルトレーニングジョブを開始します。
- **`%%neptune_ml endpoint create`** - Neptune ML モデルのクエリエンドポイントを作成します。

Neptune ワークベンチでのグラフの視覚化

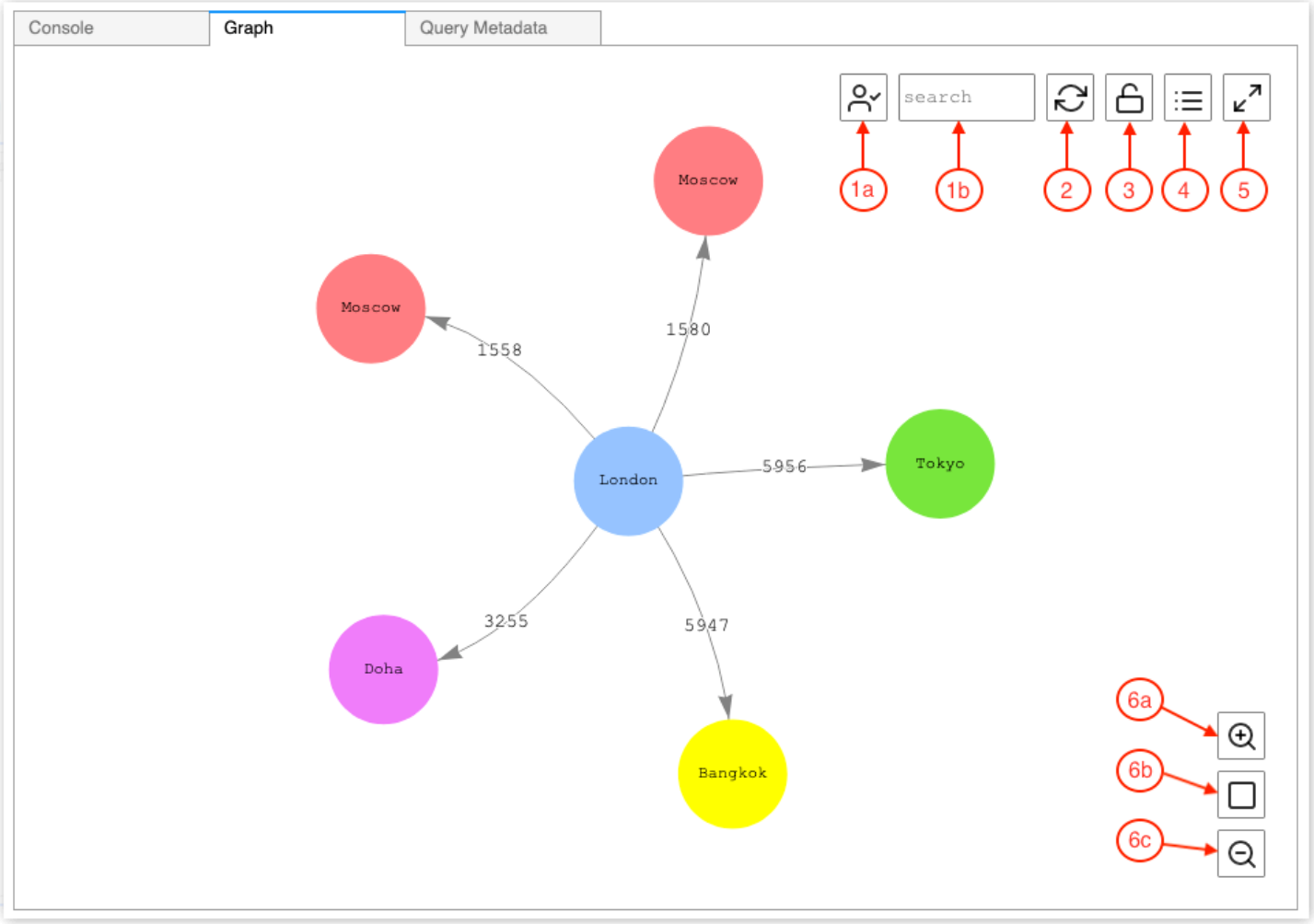
多くの場合、Neptune ワークベンチは、クエリ結果の視覚図を作成し、また表形式にして返すことができます。グラフの視覚化は、視覚化が可能な場合はいつでもクエリ結果内のグラフタブに表示されます。

ここで説明する組み込みの視覚化機能に加えて、Neptune グラフノートブックでは[より高度な視覚化ツール]を使用することもできます。

> **Note**
>
> すでに使用しているノートブックに最近追加された機能や修正プログラムにアクセスするには、まずノートブックインスタンスを停止してから再起動します。

[グラフ] タブインターフェイスの概要

この図は、[グラフ] タブにあるユーザーインターフェイス要素を示しています。



1. グラフ検索

   a. UUID トグル: グラフ検索に ID プロパティ値を含めるかどうかを切り替えます。デフォルトでは、ID の含有は有効です。無効にすると、ノード ID を参照するエッジプロパティを含む ID プロパティが一致しても、要素は強調表示されません。

   b. 検索テキストフィールド: ここで指定するテキスト文字列を含むすべての頂点とエッジのプロパティ値を強調表示します。

2. グラフリセット — グラフ物理シミュレーションを再実行し、ウィンドウ内のグラフに合わせてズームを設定します。

3. グラフ物理の切り替え — グラフ物理シミュレーションの実行を切り替えます。物理演算はデフォルトで有効であり、グラフは動的に変化します。無効にすると、他の頂点を動かしても頂点はその位置に固定されたままになります。

4. 詳細ビュー — ノードまたはエッジを選択すると、要素のプロパティキーと値 (クエリ結果に存在する場合) のリストが表示されます。

5. 全画面表示 — グラフタブウィンドウを画面に合わせて拡大します。もう一度クリックすると、グラフタブが最小化されます。
6. ズームオプション
   a. ズームイン
   b. ズームリセット: すべての頂点がグラフタブウィンドウに収まるようにズームを設定します。
   c. ズームアウト

Gremlin クエリ結果の視覚化

Neptune ワークベンチは、path を返す Gremlin クエリのクエリ結果の視覚化を作成します。視覚化を表示するには、クエリを実行した後のクエリの下にあるコンソールタブの右側にあるグラフタブを選択します。

クエリの視覚化のヒントを使用して、ビジュアライザがクエリ出力を図示化する方法を制御できます。これらのヒントは、%%gremlin セルマジックに従うもので、--path-pattern (またはその短い形式、-p) パラメータ名に先行します。

```
%%gremlin -p comma-separated hints
```

また、--group-by (または -g) フラグを使用して、頂点のグループ化に使用する頂点のプロパティを指定します。これにより、さまざまな頂点グループの色またはアイコンを指定できます。

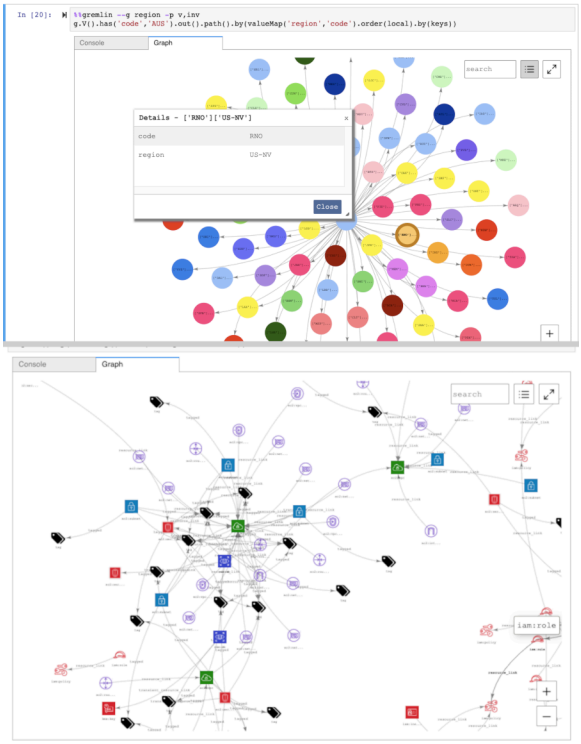
ヒントの名前は、頂点間を移動するときに一般的に使用される Gremlin のステップを反映し、それに応じて動作します。複数のヒントをカンマで区切って (間にスペースは不要) 組み合わせて使用できます。使用されるヒントは、視覚化するクエリの対応する Gremlin ステップと一致する必要があります。以下がその例です。

```
%%gremlin -p v,oute,inv
g.V().hasLabel('airport').outE().inV().path().by('code').by('dist').limit(5)
```

使用可能な視覚化のヒントは次のとおりです。

```
v
inv
outv
e
ine
oute
```

グループを使用したグラフの視覚化の例をいくつか示します。



クエリ結果の視覚化

Neptune ワークベンチは、次のいずれかの形式をとる SPARQL クエリのクエリ結果の視覚化を作成します。

- `SELECT ?subject ?predicate ?object`
- `SELECT ?s ?p ?o`

視覚化を表示するには、クエリを実行した後のクエリの下にある表タブの右側にあるグラフタブを選択します。

デフォルトでは、SPARQL の視覚化には、o? は `uri` または bnode (空白ノード) であるトリプルパターンのみを含みます。その他のリテラル文字列や整数などの ?o バインディング型は、すべてグラフタブ内の詳細ペインを使用して表示できる ?s ノードのプロパティとして処理されます。

ただし、視覚化には頂点などのリテラル値を含める場合があります。これを行うには、`%%sparql` セルマジック後の `--expand-all` クエリヒントを使用します。

```
%%sparql --expand-all
```

これは、ビジュアライザーにバインドの種類に関係なく、グラフ図にあるすべての ?s ?p ?o 結果を含めるように指示します。

このヒントは、Air-Routes-SPARQL.ipynb ノートブックで使用されており、ヒントの有無にかかわらずクエリを実行して、ビジュアライゼーションでどのような違いがあるかを確認できます。

Neptune ワークベンチでビジュアライゼーションチュートリアルノートブックにアクセスする

Neptune ワークベンチに付属する 2 つのビジュアライゼーションチュートリアルノートブックには、Gremlin と SPARQL でグラフデータを効果的にクエリして結果を視覚化する方法の例が豊富にあります。

Visualization ノートブックに移動します

1. 左側のナビゲーションペインで、右側にあるノートブックを開くボタンを選択します。
2. Neptune ワークベンチが開き、Jupyter を実行すると、トップレベルにNeptuneフォルダが表示されます。フォルダを開くには、それを選択します。
3. 次のレベルには、02-Visualization という名前のフォルダがあります。このフォルダを開きます。内部には、Gremlin と SPARQL でグラフデータをクエリするさまざまな方法、およびクエリ結果を視覚化する方法を説明するいくつかのノートブックがあります。

   - Air-Routes-Gremlin
   - Air-Routes-SPARQL
   - ワークベンチの視覚化ブログ
   - EPL-Gremlin
   - EPL-SPARQL

   ノートブックを選択して、そのノートブックに含まれるクエリを試します。

Neptune の設定

Amazon Neptune へようこそ。このセクションは、新しい Neptune DB クラスターを作成し、Neptune ドキュメントで探しているものを見つけるのに役立ちます。

> **Note**
>
> AWS グラフデータベースリファレンスアーキテクチャとリファレンスデプロイアーキテクチャについては、Amazon Neptune リソース」を参照してください。これらのリソースは、グラフデータモデルとクエリ言語に関する選択を通知し、開発プロセスを加速するのに役立ちます。

トピック

適切な Neptune DB インスタンスタイプを選択する

Amazon Neptune には、さまざまなグラフワークロードに適したさまざまな機能を提供するさまざまなインスタンスサイズとファミリーが用意されています。このセクションでは、ニーズに最適なインスタンスタイプを選択する方法について説明します。

これらのファミリーの各インスタンスタイプの料金については、Neptune の料金表ページをご覧ください。

インスタンスリソース割り当ての概要

Neptune で使用されている各 Amazon EC2 インスタンスタイプとサイズは、定義された量のコンピューティング (vCPU) とシステムメモリを提供します。Neptune のプライマリストレージはクラスター内の DB インスタンスの外部にあり、コンピューティングとストレージの容量を互いに独立してスケーリングできます。

このセクションでは、コンピューティングリソースをスケーリングする方法と、さまざまなインスタンスファミリーのそれぞれの違いに焦点を当てます。

すべてのインスタンスファミリーにおいて、vCPU リソースは vCPU あたり 2 つのクエリ実行スレッドをサポートするように割り当てられます。このサポートはインスタンスサイズによって決まります。特定の Neptune DB インスタンスの適切なサイズを決定する際には、アプリケーションの同時実行可能性とクエリの平均レイテンシーを考慮する必要があります。必要な vCPUs の数は、次のように見積もることができます。レイテンシーはクエリの平均レイテンシー (秒単位) として測定され、同時実行性は 1 秒あたりのクエリの目標数として測定されます。

$$vCPUs = \frac{latency \times concurrency}{2}$$

> **Note**
>
> DFE クエリエンジンを使用する SPARQL クエリ、openCypher クエリ、および Gremlin 読み取りクエリは、特定の状況下では、クエリ 1 つにつき複数の実行スレッドを使用する場合があります。DB クラスターのサイズを最初に決定するときは、各クエリが実行ごとに 1 つの実行スレッドを消費するという前提から始め、クエリキューへのバックプレッシャーが確認されたらスケールアップします。これは、、`/gremlin/status`、`/oc/status`または `/sparql/status` APIs、 `MainRequestsPendingRequestsQueue` CloudWatch メトリクスを使用して観察することもできます。

各インスタンスのシステムメモリは、バッファプールキャッシュとクエリ実行スレッドメモリという 2 つの主要な割り当てに分かれています。

インスタンスで使用可能なメモリの約 3 分の 2 がバッファープールキャッシュに割り当てられます。バッファープールキャッシュは、グラフの最も最近使用されたコンポーネントをキャッシュして、それらのコンポーネントに繰り返しアクセスするクエリのアクセスを高速化するために使用され

ます。システムメモリの容量が大きいインスタンスは、バッファープールキャッシュも大きく、グラフをより多くローカルに保存できます。ユーザーは、バッファキャッシュヒットと で利用可能なメトリクスの欠落をモニタリングすることで、適切な量のバッファプールキャッシュを調整できます CloudWatch。

一定期間キャッシュヒット率が 99.9% を下回った場合は、インスタンスのサイズを増やしたほうがいいかもしれません。これは、バッファープールの容量が十分ではなく、エンジンが基盤となるストレージボリュームからデータを効率的よりも頻繁に取得しなければならないことを示しています。

システムメモリの残りの 3 分の 1 は、オペレーティングシステム用のメモリと、必要に応じてスレッドが使用できる小さな動的プールを残して、クエリ実行スレッドに均等に分散されます。各スレッドで使用できるメモリは、インスタンスサイズごとにわずかに増加し、8xl インスタンスタイプでは、スレッドごとに割り当てられるメモリが最大サイズに達します。

スレッドメモリを追加するタイミングは、`OutOfMemoryException` (OOM) が発生したときです。OOM 例外は、1 つのスレッドに割り当てられた最大メモリを超えるメモリが必要になった場合に発生します (これは、インスタンス全体がメモリ不足になることと同じではありません)。

t3 および t4g インスタンスタイプ

`t3` および `t4g` ファミリーのインスタンスは、グラフデータベースを使い始めるときだけでなく、初期の開発とテストにも低コストのオプションを提供します。これらのインスタンスは、Neptune 無料利用枠オファー の対象となります。これにより、新規のお客様は、スタンドアロン AWS アカウント内で使用される、または一括請求 (支払いアカウント) で AWS 組織の下にロールアップされる最初の 750 インスタンス時間を無料で Neptune を使用できます。

`t3` および `t4g` インスタンスは中規模構成 (`t3.medium` と `t4g.medium`) でのみ提供されます。

これらは本番環境での使用を目的としていません。

これらのインスタンスはリソースが非常に限られているため、クエリの実行時間やデータベース全体のパフォーマンスのテストにはお勧めできません。クエリのパフォーマンスを評価するには、他のインスタンスファミリーのいずれかにアップグレードしてください。

r4 ファミリーのインスタンスタイプ

非推奨 — `r4` ファミリーは 2018 年に Neptune が発売されたときに提供されていましたが、新しいインスタンスタイプでは価格/パフォーマンスが大幅に向上しています。エンジンバージョン 1.1.0.0 では、Neptune は `r4` インスタンスタイプをサポートしなくなりました。

r5 ファミリーのインスタンスタイプ

r5 ファミリーには、ほとんどのグラフユースケースに適したメモリ最適化インスタンスタイプが含まれています。r5 ファミリーには、r5.large ~ r5.24xlarge までのインスタンスタイプが含まれます。サイズが大きくなるにつれて、コンピューティングパフォーマンスは直線的にスケーリングします。例えば、r5.xlarge (4 つの vCPU と 32 GiB のメモリ) は r5.large (2 つの vCPU と 16 GiB のメモリ) の 2 倍の vCPU とメモリを搭載し、r5.2xlarge (8 つの vCPU と 64 GiB のメモリ) は、r5.xlarge の 2 倍の vCPU とメモリを搭載しています。クエリのパフォーマンスは、r5.12xlarge インスタンスタイプまで、コンピューティング容量に直接影響すると期待できます。

r5 インスタンスファミリーは 2 ソケットの Intel CPU アーキテクチャを採用しています。r5.12xlarge 以下のタイプでは、1 つのソケットと、そのシングルソケットプロセッサが所有するシステムメモリを使用します。r5.16xlarge および r5.24xlarge タイプは、ソケットと使用可能なメモリの両方を使用します。2 ソケットアーキテクチャでは 2 つの物理プロセッサ間にいくらかのメモリ管理オーバーヘッドが必要なため、r5.12xlarge から r5.16xlarge または r5.24xlarge インスタンスタイプへのスケールアップで得られるメリットは、小さいサイズでのスケールアップほど直線的ではありません。

r5d ファミリーのインスタンスタイプ

Neptune にはルックアップキャッシュ機能があり、大量のプロパティ値やリテラルを取得して返す必要があるクエリのパフォーマンスを向上させることができます。この機能は主に、多数の属性を返す必要があるクエリを行うお客様が使用します。ルックアップキャッシュは、Neptune のインデックスストレージで属性値を何度も検索するのではなく、これらの属性値をローカルで取得することで、これらのクエリのパフォーマンスを向上させます。

ルックアップキャッシュは、r5d インスタンスタイプの NVMe にアタッチされた EBS ボリュームを使用して実装されます。クラスターのパラメータグループを使用して有効化されます。Neptune インデックス化ストレージからデータが取得されると、プロパティ値と RDF リテラルがこの NVMe ボリューム内にキャッシュされます。

ルックアップキャッシュ機能が必要ない場合は、r5d のコストが高くなるのを避けるため、r5d ではなく標準の r5 インスタンスタイプを使用してください。

r5d ファミリーには、r5 ファミリーと同じサイズ (r5d.large から r5d.24xlarge まで) のインスタンスタイプがあります。

r6g ファミリーのインスタンスタイプ

AWS は Graviton という独自の ARM ベースのプロセッサを開発し、Intel および AMD 同等品よりも優れた価格/パフォーマンスを提供します。r6g ファミリーは Graviton2 プロセッサーを使用しています。今回のテストでは、Graviton2 プロセッサは OLTP スタイルの (制約のある) グラフクエリのパフォーマンスが 10 ~ 20% 向上することを確認しました。ただし、大きな OLAP スタイルのクエリの場合、Graviton2 プロセッサの方がメモリページングのパフォーマンスがわずかに低いため、Intel プロセッサよりもパフォーマンスがわずかに低下する可能性があります。

また、r6g ファミリーはシングルソケットアーキテクチャを採用しているため、パフォーマンスは r6g.large から r6g.16xlarge (ファミリーの最大タイプ) へのコンピューティング容量に比例して増加するという点にも注意してください。

r6i ファミリーのインスタンスタイプ

Amazon R6i インスタンスは、第 3 世代の Intel Xeon スケーラブルプロセッサー (コードネーム Ice Lake) を搭載しており、メモリ集約的なワークロードに最適です。原則として、同等の R5 インスタンスタイプよりもコンピューティングコストパフォーマンスが最大 15% 高く、vCPU あたりのメモリ帯域幅が最大 20% 高くなります。

x2g ファミリーのインスタンスタイプ

グラフのユースケースの中には、インスタンスのバッファープールキャッシュが大きいほどパフォーマンスが向上するものがあります。x2g ファミリーは、こうしたユースケースをより良くサポートするために立ち上げられました。x2g ファミリーの memory-to-vCPU 比率は、 r5または r6gファミリーよりも大きくなります。x2g インスタンスは Graviton2 プロセッサも使用し、r6g インスタンスタイプと同じパフォーマンス特性の多くを備えているほか、バッファプールキャッシュも大きくなっています。

CPU 使用率が低く、バッファプールのキャッシュミス率が高い r5 または r6g インスタンスタイプを使用している場合は、代わりに x2g ファミリーを使用してみてください。そうすれば、CPU 容量を増やさずに、必要な追加メモリを確保できます。

serverless インスタンスタイプ

Neptune サーバーレス機能は、ワークロードのリソースニーズに基づいてインスタンスサイズを動的にスケーリングできます。Neptune サーバーレスでは、アプリケーションに必要な vCPU の数を計算する代わりに、DB クラスター内のインスタンスのコンピューティング容量 (Neptune キャパシ

ティユニットで測定) の上限と下限を設定できます。使用率が変動するワークロードは、プロビジョニングされたインスタンスではなくサーバーレスを使用することでコストを最適化できます。

プロビジョニングされたインスタンスとサーバーレスインスタンスの両方を同じ DB クラスターにセットアップして、最適なコストパフォーマンスを実現できます。

Neptune DB クラスターに適したストレージタイプの選択

Neptune では、料金モデルが異なる次の 2 種類のストレージを提供しています。

- 標準ストレージ — 標準ストレージは、I/O 使用率が中程度から低いアプリケーション向けの費用対効果の高いデータベースストレージです。
- I/O 最適化ストレージ – エンジンバージョン 1.3.0.0 から利用可能な I/O 最適化ストレージでは、使用しているストレージとインスタンスに対してのみ料金が発生します。ストレージコストは標準ストレージよりも高く、使用した I/O に対する料金は発生しません。I/O 使用率が高い場合は、プロビジョンド IOPS ストレージを使用すると、コストを大幅に削減できます。

  I/O 最適化ストレージは、コストが予測可能で、I/O レイテンシーが低く、I/O スループットが一貫している、I/O 負荷の高いグラフワークロードのニーズを満たすように設計されています。I/O 最適化ストレージタイプと標準ストレージタイプを切り替えることができるのは、30 日に 1 回のみです。

  I/O 最適化ストレージの料金情報については、Neptune の料金ページを参照してください。次のセクションでは、Neptune DB クラスターに I/O 最適化ストレージを設定する方法について説明します。

Neptune DB クラスター用の I/O 最適化ストレージの選択

デフォルトでは、Neptune DB クラスターは標準ストレージを使用します。I/O 最適化ストレージは、次のように、DB クラスターの作成時に有効にすることができます。

AWS CLIを使用してクラスターの作成時に I/O 最適化ストレージを有効にする方法の例を次に示します。

```
aws neptune create-db-cluster \
  --database-name (name for the new database) \
  --db-cluster-identifier (an ID for the cluster) \
  --engine neptune \
```

```
  --engine-version 1.3.0.0 \
  --storage-type iopt1
```

これにより、作成したすべてのインスタンスで、I/O 最適化ストレージが自動的に有効になります。

```
aws neptune create-db-instance \
  --db-cluster-identifier (the ID of the new cluster) \
  --db-instance-identifier (an ID for the new instance) \
  --engine neptune \
  --db-instance-class db.r5.large
```

既存の DB クラスターを変更して、次のように I/O 最適化ストレージを有効にすることもできます。

```
aws neptune modify-db-cluster \
  --db-cluster-identifier (the ID of a cluster without I/O-Optimized storage) \
  --storage-type iopt1 \
  --apply-immediately
```

I/O 最適化ストレージが有効になっている DB クラスターにバックアップスナップショットを復元できます。

```
aws neptune restore-db-cluster-from-snapshot \
  --db-cluster-identifier (an ID for the restored cluster) \
  --snapshot-identifier (the ID of the snapshot to restore from) \
  --engine neptune \
  --engine-version 1.3.0.0 \
  --storage-type iopt1
```

クラスターが I/O 最適化ストレージを使用しているかどうかは、`describe-` を呼び出して確認できます。I/O 最適化ストレージが有効になっている場合、呼び出しは `iop1` に設定されたストレージタイプフィールドを返します。

新しい Neptune DB クラスターの作成

新しい Amazon Neptune DB クラスターを作成する最も簡単な方法は、必要なリソースをすべて作成する AWS CloudFormation テンプレートを使用することです。手動で行う必要はありません。 AWS CloudFormation テンプレートは、Amazon Elastic Compute Cloud (Amazon EC2) インスタンスの作成など、セットアップの多くを実行します。

AWS CloudFormation テンプレートを使用して新しい Neptune DB クラスターを起動するには

1. 「IAM ユーザー許可」で説明されているように、Neptune DB クラスターを操作するために必要なアクセス許可を持つ新しい IAM ユーザーを作成します。
2. 「」で説明されているように、 AWS CloudFormation テンプレートを使用するために必要な追加の前提条件を設定しますAWS CloudFormation を使用して Neptune をセットアップするための前提条件。
3. の説明に従って AWS CloudFormation 、スタックを呼び出しますAWS CloudFormation スタックを使用して Neptune DB クラスターを作成する。

また、複数の にまたがる Neptune グローバルデータベースを作成することもできます。これにより AWS リージョン、低レイテンシーのグローバル読み取りが可能になり、停止が 全体に影響を与えるまれなケースで高速リカバリが可能になります AWS リージョン。

を使用して Amazon Neptune クラスターを手動で作成する方法については AWS Management Console、「」を参照してくださいAWS Management Console を使用して Neptune DB クラスターを起動する。

AWS CloudFormation テンプレートを使用して、Neptune で使用する Lambda 関数を作成することもできます (「」を参照AWS CloudFormation を使用して Neptune で使用する Lambda 関数を作成するには)。

Neptune のクラスターとインスタンスの管理の詳細については、「Amazon Neptune データベースの管理」を参照してください。

AWS CloudFormation を使用して Neptune をセットアップするための前提条件

AWS CloudFormation テンプレートを使用して Amazon Neptune クラスターを作成する前に、以下が必要です。

- Amazon EC2 の [キーペア]
- を使用するために必要なアクセス許可 AWS CloudFormation。

を使用して Neptune クラスターの起動に使用する Amazon EC2 キーペアを作成する AWS CloudFormation

AWS CloudFormation テンプレートを使用して Neptune DB クラスターを起動するには、 AWS CloudFormation スタックを作成するリージョンで Amazon EC2keyペア (および関連する PEM ファイル) が使用可能である必要があります。

キーペアを作成する必要がある場合は、Amazon EC2 ユーザーガイド」の「Amazon EC2 を使用したキーペアの作成」またはAmazon EC2 ユーザーガイド」の「Amazon EC2 を使用したキーペアの作成」を参照してください。 Amazon EC2 Amazon EC2

AWS CloudFormation テンプレートを使用するために必要なアクセス許可を付与する IAM ポリシーを追加する

まず、「Neptune のアクセス許可を持つ IAM ユーザーの作成」で説明されているように、Neptune での作業に必要なアクセス許可を持つ IAM ユーザーを設定する必要があります。

次に、 AWS 管理ポリシー `AWSCloudFormationReadOnlyAccess`をそのユーザーに追加する必要があります。

最後に、以下のカスタマー管理ポリシーを作成して、そのユーザーに追加する必要があります。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
          "rds:CreateDBCluster",
          "rds:CreateDBInstance"
      ],
```

```
      "Resource": [
          "arn:aws:rds:*:*:*"
      ],
      "Condition": {
          "StringEquals": {
              "rds:DatabaseEngine": ["graphdb","neptune"]
          }
      }
    },
    {
      "Action": [
        "rds:AddRoleToDBCluster",
        "rds:AddSourceIdentifierToSubscription",
        "rds:AddTagsToResource",
        "rds:ApplyPendingMaintenanceAction",
        "rds:CopyDBClusterParameterGroup",
        "rds:CopyDBClusterSnapshot",
        "rds:CopyDBParameterGroup",
        "rds:CreateDBClusterParameterGroup",
        "rds:CreateDBClusterSnapshot",
        "rds:CreateDBParameterGroup",
        "rds:CreateDBSubnetGroup",
        "rds:CreateEventSubscription",
        "rds:DeleteDBCluster",
        "rds:DeleteDBClusterParameterGroup",
        "rds:DeleteDBClusterSnapshot",
        "rds:DeleteDBInstance",
        "rds:DeleteDBParameterGroup",
        "rds:DeleteDBSubnetGroup",
        "rds:DeleteEventSubscription",
        "rds:DescribeAccountAttributes",
        "rds:DescribeCertificates",
        "rds:DescribeDBClusterParameterGroups",
        "rds:DescribeDBClusterParameters",
        "rds:DescribeDBClusterSnapshotAttributes",
        "rds:DescribeDBClusterSnapshots",
        "rds:DescribeDBClusters",
        "rds:DescribeDBEngineVersions",
        "rds:DescribeDBInstances",
        "rds:DescribeDBLogFiles",
        "rds:DescribeDBParameterGroups",
        "rds:DescribeDBParameters",
        "rds:DescribeDBSecurityGroups",
        "rds:DescribeDBSubnetGroups",
```

```
        "rds:DescribeEngineDefaultClusterParameters",
        "rds:DescribeEngineDefaultParameters",
        "rds:DescribeEventCategories",
        "rds:DescribeEventSubscriptions",
        "rds:DescribeEvents",
        "rds:DescribeOptionGroups",
        "rds:DescribeOrderableDBInstanceOptions",
        "rds:DescribePendingMaintenanceActions",
        "rds:DescribeValidDBInstanceModifications",
        "rds:DownloadDBLogFilePortion",
        "rds:FailoverDBCluster",
        "rds:ListTagsForResource",
        "rds:ModifyDBCluster",
        "rds:ModifyDBClusterParameterGroup",
        "rds:ModifyDBClusterSnapshotAttribute",
        "rds:ModifyDBInstance",
        "rds:ModifyDBParameterGroup",
        "rds:ModifyDBSubnetGroup",
        "rds:ModifyEventSubscription",
        "rds:PromoteReadReplicaDBCluster",
        "rds:RebootDBInstance",
        "rds:RemoveRoleFromDBCluster",
        "rds:RemoveSourceIdentifierFromSubscription",
        "rds:RemoveTagsFromResource",
        "rds:ResetDBClusterParameterGroup",
        "rds:ResetDBParameterGroup",
        "rds:RestoreDBClusterFromSnapshot",
        "rds:RestoreDBClusterToPointInTime"
      ],
      "Effect": "Allow",
      "Resource": [
          "*"
      ]
    },
    {
      "Action": [
        "cloudwatch:GetMetricStatistics",
        "cloudwatch:ListMetrics",
        "ec2:DescribeAccountAttributes",
        "ec2:DescribeAvailabilityZones",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcAttribute",
        "ec2:DescribeVpcs",
```

```
        "kms:ListAliases",
        "kms:ListKeyPolicies",
        "kms:ListKeys",
        "kms:ListRetirableGrants",
        "logs:DescribeLogStreams",
        "logs:GetLogEvents",
        "sns:ListSubscriptions",
        "sns:ListTopics",
        "sns:Publish"
      ],
      "Effect": "Allow",
      "Resource": [
          "*"
      ]
    },
    {
      "Action": "iam:PassRole",
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
            "iam:passedToService": "rds.amazonaws.com"
        }
      }
    },
    {
      "Action": "iam:CreateServiceLinkedRole",
      "Effect": "Allow",
      "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
AWSServiceRoleForRDS",
      "Condition": {
        "StringLike": {
            "iam:AWSServiceName": "rds.amazonaws.com"
        }
      }
    }
  ]
}
```

> **Note**
>
> 次のアクセス権限は、スタックの削除にのみ必要で
> す。iam:DeleteRole、iam:RemoveRoleFromInstanceProfile、iam:DeleteRolePolicy、i
> および ec2:DeleteVpcEndpoints。
> また、ec2:*Vpc は、ec2:DeleteVpc アクセス許可を付与します。

AWS CloudFormation スタックを使用して Neptune DB クラスターを作成する

AWS CloudFormation テンプレートを使用して Neptune DB クラスターをセットアップできます。

1. AWS CloudFormation コンソールで AWS CloudFormation スタックを起動するには、次の表の「スタックの起動」ボタンのいずれかを選択します。

リージョン	ビュー	デザイナーで表示	起動する
米国東部 (バージニア北部)	表示	デザイナーで表示	
米国東部 (オハイオ)	表示	デザイナーで表示	
米国西部 (北カリフォルニア)	表示	デザイナーで表示	
米国西部 (オレゴン)	表示	デザイナーで表示	
カナダ (中部)	表示	デザイナーで表示	
南米 (サンパウロ)	表示	デザイナーで表示	
欧州 (ストックホルム)	表示	デザイナーで表示	
欧州 (アイルランド)	表示	デザイナーで表示	
欧州 (ロンドン)	表示	デザイナーで表示	
欧州 (パリ)	表示	デザイナーで表示	

リージョン	ビュー	デザイナーで表示	起動する
欧州 (フランクフルト)	表示	デザイナーで表示	
中東 (バーレーン)	表示	デザイナーで表示	
中東 (アラブ首長国連邦)	表示	デザイナーで表示	
イスラエル (テルアビブ)	表示	デザイナーで表示	
アフリカ (ケープタウン)	表示	デザイナーで表示	
アジアパシフィック (香港)	表示	デザイナーで表示	
アジアパシフィック (東京)	表示	デザイナーで表示	
アジアパシフィック (ソウル)	表示	デザイナーで表示	
アジアパシフィック (シンガポール)	表示	デザイナーで表示	
アジアパシフィック (シドニー)	表示	デザイナーで表示	
アジアパシフィック (ムンバイ)	表示	デザイナーで表示	
中国 (北京)	表示	デザイナーで表示	
中国 (寧夏)	表示	デザイナーで表示	

リージョン	ビュー	デザイナーで表示	起動する
AWS GovCloud（米国西部）	表示	デザイナーで表示	Launch Stack
AWS GovCloud（米国東部）	表示	デザイナーで表示	Launch Stack

2. [Select Template] ページで、[Next] を選択します。
3. 詳細の指定ページで、EC2SSHKeyPair 名 のキーペアを選択します。

   このキーペアは、EC2 インスタンスにアクセスするために必要です。選択したキーペアの PEM ファイルがあることを確認します。
4. [次へ] をクリックします。
5. [Options(オプション)] ページで、[Next(次へ)] を選択します。
6. [確認] ページで、 AWS CloudFormation によって IAM リソースが作成されることを確認する最初のチェックボックスをオンにします。新しいスタックの `CAPABILITY_AUTO_EXPAND` を確認する 2 つ目のチェックボックスをオンにします。

   > **Note**
   >
   > `CAPABILITY_AUTO_EXPAND` は、スタックの作成時に事前の確認なしにマクロが展開されることを明示的に確認します。ユーザーは、処理されたテンプレートから変更セットを作成することが多いため、実際にスタックを作成する前にマクロによって行われた変更を確認できます。詳細については、 AWS CloudFormation CreateStack API を参照してください。

   次に [作成] を選択します。

> **Note**
>
> AWS CloudFormation テンプレートを使用して DB クラスターのエンジンバージョンをアップグレードすることもできます。

Amazon Neptune DB クラスターが配置される Amazon VPC をセットアップする

Amazon Neptune DB クラスターは、Amazon Virtual Private Cloud (Amazon VPC) でのみ作成できます。そのエンドポイントには、その VPC 内でアクセスできます。

VPC の設定には、DB クラスターへのアクセス方法に応じてさまざまな方法があります。

Neptune DB クラスターが配置される VPC を設定するときの注意事項をいくつか次に示します。

- VPC には少なくとも 2 つのサブネットが必要です。これらのサブネットは、2 つの異なるアベイラビリティーゾーン (AZ) になければなりません。クラスターインスタンスを少なくとも 2 つの AZ に分散することにより、万一、アベイラビリティーゾーンに障害が発生した場合でも、Neptune では、DB クラスターに使用可能なインスタンスが常に存在します。Neptune DB クラスターのクラスターボリュームは、データ損失の可能性が少ない耐久性のあるストレージを提供するために、常に 3 つのアベイラビリティーゾーンにまたがっています。
- 各サブネットの CIDR ブロックは、メンテナンス作業、フェイルオーバー、およびスケーリング時に Neptune が必要とする可能性のある IP アドレスに十分対応できる大きさが必要です。
- VPC には、作成したサブネットを含む DB サブネットグループが必要です。Neptune は、サブネットグループから 1 つのサブネットと、そのサブネット内の IP アドレスを選んで、DB クラスター内の各 DB インスタンスに関連付けます。これで DB インスタンスはサブネットと同じ AZ に配置されます。
- VPC は DNS が有効になっている必要があります (DNS ホスト名と DNS 解決の両方)。
- VPC には、DB クラスターへのアクセスを許可する VPC セキュリティグループが必要です。
- Neptune VPC のテナンシーはデフォルトに設定する必要があります。

Neptune DB クラスターが配置されている VPC にサブネットを追加する

サブネットは、VPC の IP アドレスの範囲です。Neptune DB クラスターや EC2 インスタンスなどのリソースを特定のサブネットに起動できます。サブネットを作成する際、VPC CIDR ブロックのサブセットである、サブネットの CIDR ブロックを指定します。各サブネットは、全体が 1 つのアベイラビリティーゾーン (AZ) 内に含まれている必要があり、複数のゾーンにまたがることはできません。個別のアベイラビリティーゾーンでインスタンスを起動することにより、1 つのゾーンで発生した障害からアプリケーションを保護できます。詳細については、VPC サブネットのドキュメントを参照してください。

Neptune DB クラスターには少なくとも 2 つの VPC サブネットが必要です。

VPC にサブネットを追加するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/vpc/ で Amazon VPC コンソールを開きます。
2. ナビゲーションペインで、[Subnets (サブネット)] を選択します。
3. [VPC ダッシュボード] で [サブネット] を選択し、[サブネットの作成] を選択します。
4. [サブネットの作成] ページで、サブネットを作成する VPC を選択します。
5. [サブネット設定] で、次の選択を行います。
   a. [サブネット名] に、新しいサブネットの名前を入力します。
   b. サブネットのアベイラビリティーゾーン (AZ) を選択するか、[指定なし] のままにします。
   c. [IPv4 CIDR ブロック] にサブネットの IP アドレスブロックを入力します。
   d. 必要に応じてサブネットにタグを追加します。
   e. 選択します。
6. 同時に別のサブネットを作成する場合は、[新しいサブネットの追加] を選択します。
7. [サブネットの作成] を選択して、新しいサブネットを作成します。

VPC 内にサブネットを作成する

サブネットグループを作成します。

Neptune サブネットグループを作成するには

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/neptune/home で Amazon Neptune コンソールを開きます。
2. [サブネットグループ] を選択してから、[DB サブネットグループの作成] を選択します。
3. 新しいサブネットグループの名前と説明を入力します (説明は必須です)。
4. [VPC] で、このサブネットグループを配置する VPC を選択します。
5. [アベイラビリティーゾーン] で、このサブネットグループを配置する AZ を選択します。
6. [サブネット] で、この AZ 内の 1 つ以上のサブネットをこのサブネットグループに追加します。
7. [作成] をクリックして、新しいサブネットグループを作成します。

VPC コンソールを使用してセキュリティグループを作成する

セキュリティグループは、VPC 内の Neptune DB クラスターへのアクセスを提供します。セキュリティグループは、関連付けられた DB クラスターのファイアウォールとして機能し、インバウンドトラフィックとアウトバウンドトラフィックの両方をインスタンスレベルで制御します。デフォルトでは、DB インスタンスは、アクセスを防ぐファイアウォールとデフォルトのセキュリティグループとともに作成されます。アクセスを有効にするには、追加のルールを含む VPC セキュリティグループが必要です。

次の手順は、Amazon EC2 インスタンスが Neptune DB クラスターにアクセスするために使用するポート範囲と IP アドレスを指定するカスタム TCP ルールを追加する方法を示しています。IP アドレスの代わりに、EC2 インスタンスに割り当てられた VPC セキュリティグループを使用できます。

コンソールで Neptune の VPC セキュリティグループを作成するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/vpc/ で Amazon VPC コンソールを開きます。
2. コンソールの右上で、Neptune の VPC セキュリティグループを作成する AWS リージョンを選択します。そのリージョンの Amazon VPC リソースのリストには、少なくとも 1 つの VPC と複数のサブネットがあることが示されていなければなりません。表示されない場合、そのリージョンにはデフォルトの VPC はありません。
3. ナビゲーションペインの [セキュリティ] で、[セキュリティグループ] を選択します。
4. [Create Security Group] を選択します。[セキュリティグループの作成] ウィンドウで、[セキュリティグループ名]、[説明]、および Neptune DB クラスターを配置する VPC の識別子を入力します。
5. Neptune DB クラスターに接続したい Amazon EC2 インスタンスのセキュリティグループのインバウンドルールを追加します。

   a. [インバウンドルール] エリアで、[ルールの追加] を選択します。

   b. [タイプ] リストで、[カスタム TCP] を選択したままにします。

   c. [ポート範囲] ボックスに、Neptune のデフォルトのポート値である 8182 を入力します。

   d. [ソース] に、Neptune にアクセスする IP アドレスの範囲 (CIDR 値) を入力するか、既存のセキュリティグループ名を選択します。

   e. IP アドレスをさらに追加するか、別のポート範囲を追加する必要がある場合は、[ルールの追加] を再び選択します。

6. [アウトバウンドルール] エリアで、必要に応じて 1 つ以上のアウトバウンドルールを追加することもできます。
7. 完了したら、[Create security group] を選択します。

新しい Neptune DB クラスターを作成するときに、この新しい VPC セキュリティグループを使用できます。

デフォルトの VPC を使用する場合、すべての VPC のサブネットにまたがるデフォルトのサブネットグループがすでに作成されています。Neptune コンソールで [データベースの作成] を選択すると、別の VPC を指定しない限り、デフォルトの VPC が使用されます。

VPC に DNS サポートがあることを確認してください。

ドメインネームシステム (DNS) は、インターネットで使用する名前を対応する IP アドレスに解決するための標準です。DNS ホスト名は、ホスト名とドメイン名で構成され、コンピュータに一意の名前を付けます。DNS サーバーは DNS ホスト名を対応する IP アドレスに解決します。

DNS ホスト名と DNS 解決の両方が VPC で確実に有効になっていることを確認します。VPC ネットワーク属性の `enableDnsHostnames` と `enableDnsSupport` を `true` に設定する必要があります。これらの属性を表示および変更するには、VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。

詳細については、「Using DNS with Your VPC」を参照してください。

> **Note**
>
> Route 53 を使用している場合は、設定によって VPC 内の DNS ネットワーク属性が上書きされないことを確認します。

Amazon Neptune ングラフへの接続

Neptune DB クラスターを作成したら、次のステップはクラスターへの接続をセットアップすることです。

Neptune エンドポイントと通信するために curl または awscurl をセットアップする

このドキュメントの多くの例に示されるように、Neptune DB クラスターにクエリを送信するためのコマンドラインツールがあると非常に便利です。curl コマンドラインツールは、IAM 認証が有効になっていない場合に Neptune エンドポイントと通信するための優れたオプションです。7.75.0 以降のバージョンでは、IAM `--aws-sigv4` 認証が有効になっている場合にリクエストに署名するオプションがサポートされています。

IAM 認証が有効になっているエンドポイントでは、awscurl を使用することもできます。これは、`curl` とほぼ同じ構文を使用しますが、IAM 認証に必要な署名リクエストをサポートします。IAM 認証はセキュリティを強化するため、一般的には有効にすることをおすすめします。

`curl` (または `awscurl`) の使用法については、「curl man page」および書籍「Everything curl」を参照してください。

(Neptune に必要な) HTTPS を使用して接続するには、`curl` に適切な証明書へのアクセスが必要です。`curl` が適切な証明書を見つけられる限り、これによって、別のパラメータを必要とすることなく、HTTP 接続のように HTTPS 接続が処理されます。同じことが `awscurl` にも当てはまります。このドキュメントの例はこのシナリオに基づいています。

そのような証明書を取得する方法と、`curl` が使える証明書を認証局 (CA) 証明書ストアに適切にフォーマットする方法については、`curl` ドキュメント内の「SSL 証明書の検証」を参照してください。

次に、`CURL_CA_BUNDLE` 環境変数を使用してこの CA 証明書ストアの場所を指定できます。Windows では、`curl` は自動的に `curl-ca-bundle.crt` という名前のファイルを検索します。まず `curl.exe` と同じディレクトリで curl.exe を検索し、次にこのパスの他の場所を検索します。詳細については、「SSL Certificate Verification」を参照してください。

Neptune DB クラスターに接続するさまざまな方法

Amazon Neptune DB クラスターは、Amazon Virtual Private Cloud (Amazon VPC) でのみ作成できます。DB クラスターの Neptune パブリックエンドポイントを有効にして設定しない限り、そのエンドポイントにはその VPC 内でのみアクセスできます。

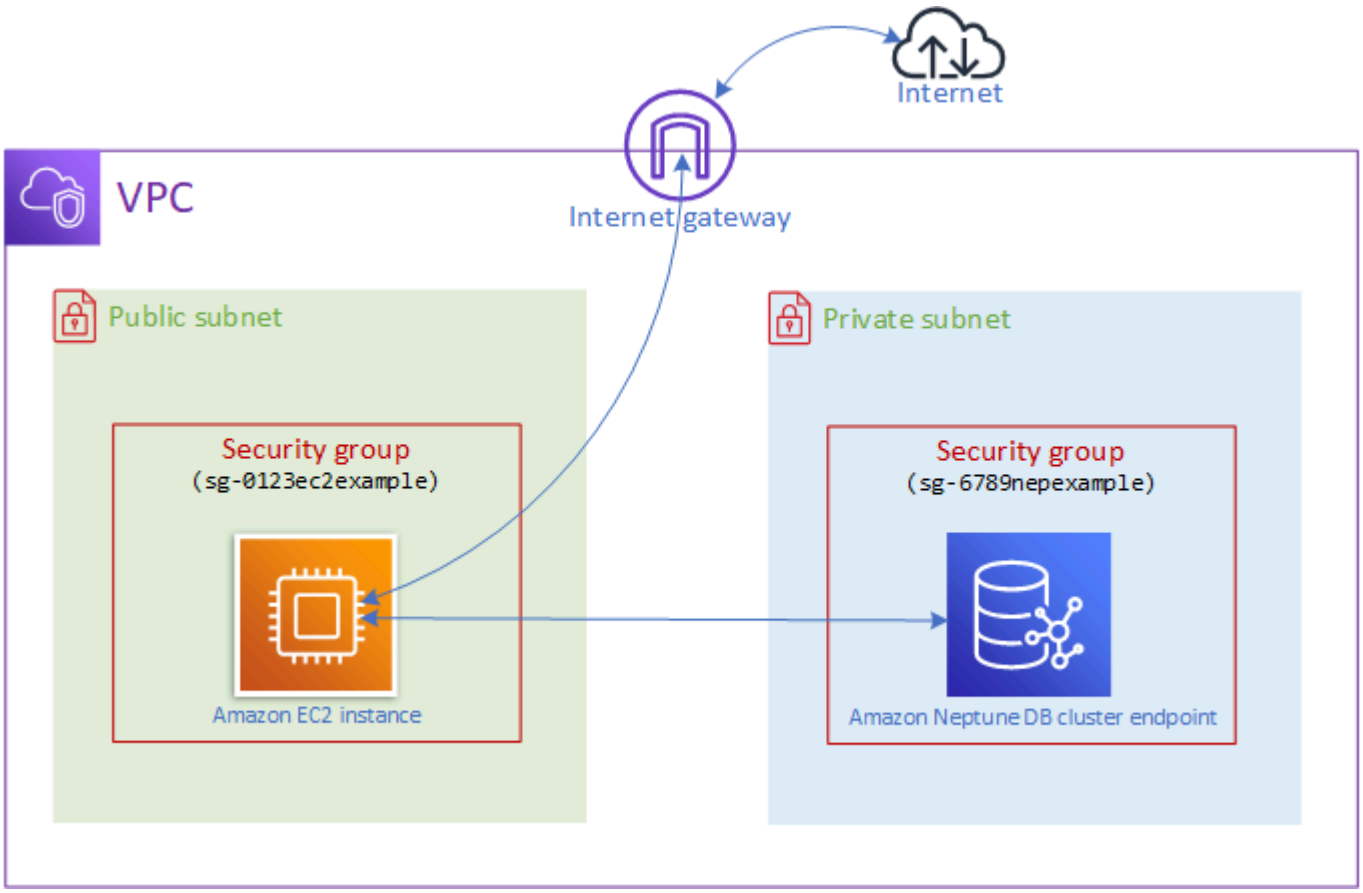
VPC 内の Neptune DB クラスターへのアクセスをセットアップするには、さまざまな方法があります。

- 同じ VPC 内の Amazon EC2 インスタンスからの接続

- 別の VPC 内の Amazon EC2 インスタンスからの接続
- プライベートネットワークからの接続

同じ VPC 内の Amazon EC2 インスタンスから Neptune DB クラスターへの接続

Neptune データベースに接続する最も一般的な方法の 1 つは、Neptune DB クラスターと同じ VPC の Amazon EC2 インスタンスから接続することです。例えば、EC2 インスタンスはインターネットに接続するウェブサーバーを実行している場合があります。この場合、EC2 インスタンスのみが Neptune DB クラスターにアクセスでき、インターネットは EC2 インスタンスにのみアクセスできます。



この設定を有効にするには、適切な VPC セキュリティグループとサブネットグループを設定する必要があります。ウェブサーバーはパブリックインターネットにアクセスできるようにパブリックサブネットでホストされ、Neptune クラスターインスタンスはプライベートサブネットでホストされ、安全な状態に保たれます。Amazon Neptune DB クラスターが配置される Amazon VPC をセットアップする を参照してください。

Amazon EC2 インスタンスがポート 8182 などの Neptune エンドポイントに接続するには、セキュリティグループを設定する必要があります。Amazon EC2 インスタンスで、たとえば `ec2-sg1` という名前のセキュリティグループを使用している場合、ポート 8182 のインバウンドルールを持ち、ソースとして `ec2-sg1` を持つ別の Amazon EC2 セキュリティグループ (例:`db-sg1`) を作成する必要があります。次に、`db-sg1` を Neptune クラスターに追加して接続を許可します。

Amazon EC2 インスタンスを作成したら、SSH を使用してログインし、Neptune DB クラスターに接続できます。SSH を使用して EC2 インスタンスに接続する方法については、「Amazon EC2 ユーザーガイド」の「Linux インスタンスに接続する」を参照してください。 Amazon EC2

Linux または macOS コマンドラインを使用して EC2 インスタンスに接続する場合は、 AWS CloudFormation スタックの Outputs セクションの SSHAccess コマンドを貼り付けることができます。現在のディレクトリに PEM ファイルがあり、PEM ファイルのアクセス権限が 400 (`chmod 400 keypair.pem`) に設定されていることが必要です。

プライベートサブネットとパブリックサブネットの両方を持つ VPC を作成するには

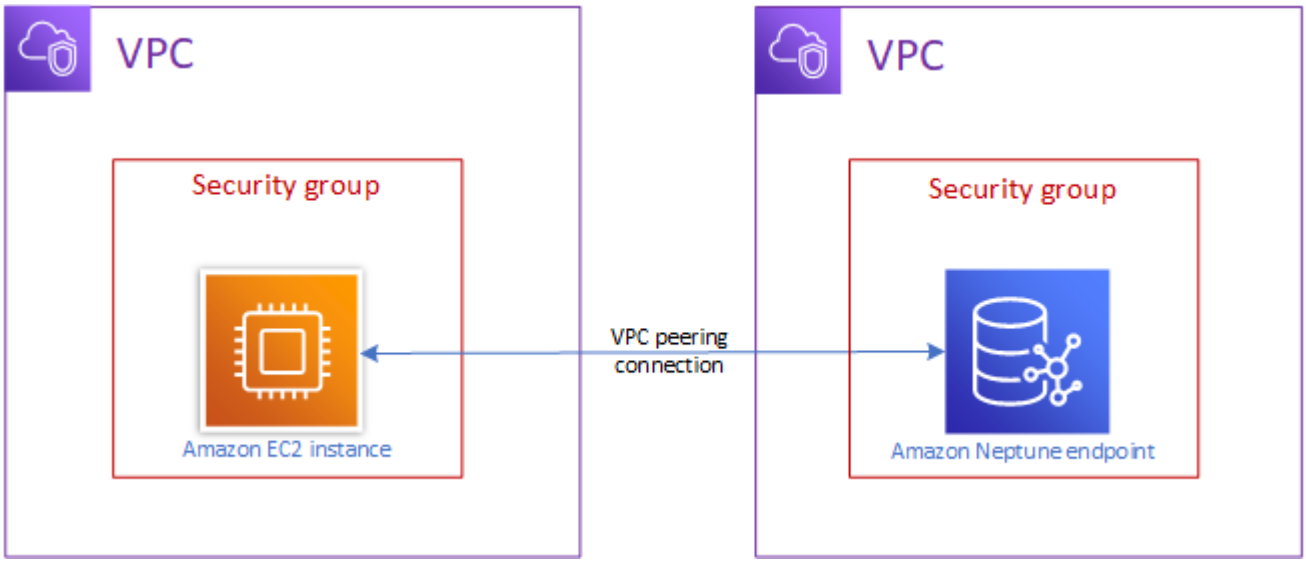
1. にサインイン AWS Management Console し、https://console.aws.amazon.com/vpc/ で Amazon VPC コンソールを開きます。
2. の右上隅で AWS Management Console、VPC を作成するリージョンを選択します。
3. [VPC ダッシュボード] で、[VPC ウィザードの起動] を選択します。
4. [VPC の作成] ページの [VPC 設定] エリアに入力します。
    a. [Resources to create] (作成するリソース) で、[VPC, subnets, etc.] (VPC、サブネットなど) を選択します。
    b. デフォルトの名前タグをそのまま使用するか、任意の名前を入力するか、[自動生成] チェックボックスをオフにして名前タグの生成を無効にします。
    c. IPv4 CIDR ブロックの値は `10.0.0.0/16` のままにしておきます。
    d. [IPv6 CIDR ブロック] の値は、[IPv6 CIDR ブロックなし] のままにしておきます。
    e. [テナンシー] は [デフォルト] のままにしておきます。
    f. [アベイラビリティーゾーン (AZ)] の数は [2] のままにしておきます。
    g. 1 つ以上の NAT ゲートウェイが必要になる場合を除いて、[NAT ゲートウェイ ($)] は [なし] のままにしておきます。
    h. Amazon S3 を使用する場合を除き、[VPC エンドポイント] を [なし] に設定します。
    i. [DNS ホスト名を有効化] と [DNS 解決を有効化] の両方がオンになっていることを確認します。

5. [Create VPC (VPC の作成)] を選択します。

別の VPC の Amazon EC2 インスタンスから DB クラスターにアクセスする

Amazon Neptune DB クラスターは、Amazon Virtual Private Cloud (Amazon VPC) にのみ作成でき、そのエンドポイントはその VPC 内でのみアクセス可能であり、通常、その VPC で実行している Amazon Elastic Compute Cloud (Amazon EC2) インスタンスからです。

DB クラスターが、それへのアクセスに使用している EC2 インスタンスとは異なる VPC にある場合、VPC ピアリング接続を使用して接続できます。



VPC ピアリング接続は、2 つの VPC 間でプライベートなトラフィックのルーティングを行うネットワーク接続であり、2 つの VPC のインスタンスが同じネットワーク内に存在しているかのように通信できます。アカウントの VPC 間、アカウントの VPCsと別のアカウントの AWS VPC 間 AWS 、または別の AWS リージョンの VPC との間で VPC ピアリング接続を作成できます。

AWS は VPC の既存のインフラストラクチャを使用して VPC ピアリング接続を作成します。ゲートウェイでも AWS Site-to-Site VPN 接続でもなく、個別の物理ハードウェアに依存しません。通信の単一障害点や帯域幅のボトルネックは存在しません。

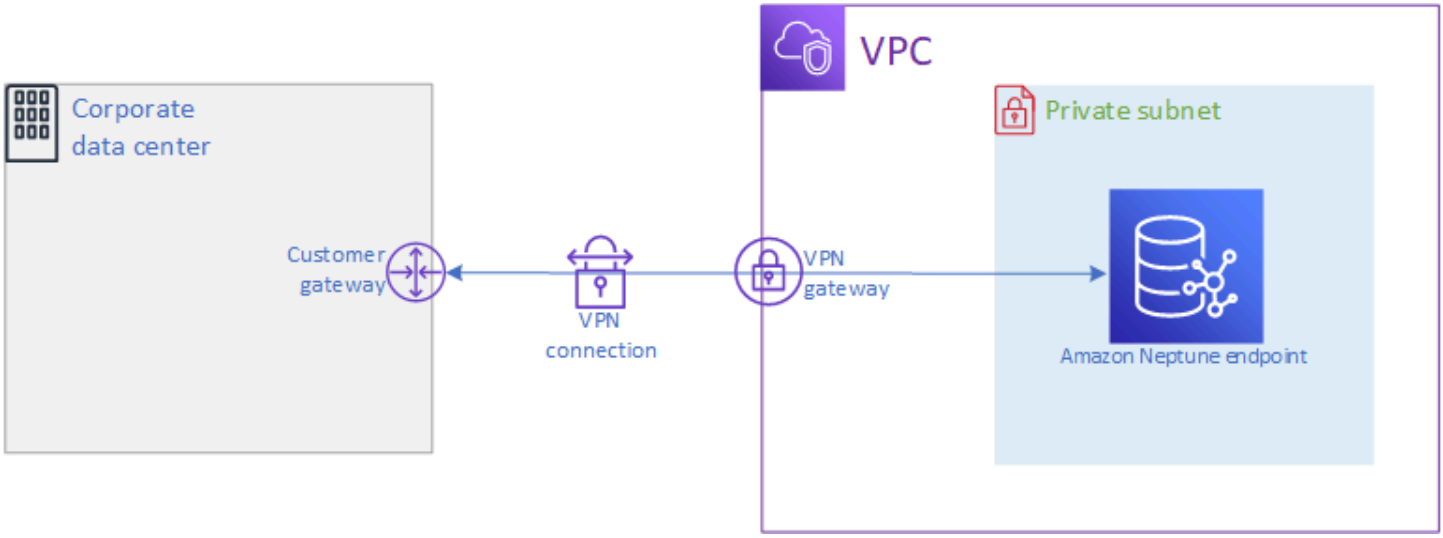
VPC ピアリングの使用法については、「Amazon VPC ピアリングガイド」を参照してください。

プライベートネットワークから DB クラスターにアクセスする

Neptune DB クラスターには、プライベートネットワークから 2 つの方法でアクセスできます。

- AWS サイト間 VPN 接続の使用。
- AWS Direct Connect 接続の使用。

上記のリンクには、これらの接続方法とその設定方法に関する情報が記載されています。 AWS Site-to-Site 接続の設定は次のようになります。



Amazon Neptune でのデータの保護

Amazon Neptune クラスターを保護する方法は複数あります。

IAM ポリシーを使用して Neptune DB クラスターへのアクセスを制限する

Neptune DB クラスターと DB インスタンスで Neptune 管理アクションを実行できるユーザーを制御するには、 AWS Identity and Access Management (IAM) を使用します。

IAM アカウントを使用して Neptune コンソールにアクセスする場合、https://console.aws.amazon.com/neptune/home で Neptune コンソールを開く前に、まず IAM アカウント AWS Management Console を使用して にサインインする必要があります。

IAM 認証情報 AWS を使用して に接続する場合、IAM アカウントには、Neptune 管理オペレーションの実行に必要なアクセス許可を付与する IAM ポリシーが必要です。詳細については、「Neptune へのアクセスを制御するためのさまざまな種類の IAM ポリシーの使用」を参照してください。

VPC セキュリティグループを使用して Neptune DB クラスターへのアクセスを制限する

Neptune DB クラスターは Amazon Virtual Private Cloud (Amazon VPC) に作成する必要があります。VPC 内の Neptune DB クラスター用の DB インスタンスのエンドポイントとポートに対して接続を開くことができるデバイスと EC2 インスタンスを制御するには、VPC セキュリティグループを使用します。VPCs の詳細については、「VPC コンソールを使用してセキュリティグループを作成する」を参照してください。

IAM 認証を使用して Neptune DB クラスターへのアクセスを制限する

Neptune DB クラスターで AWS Identity and Access Management (IAM) 認証を有効にする場合、DB クラスターにアクセスするすべてのユーザーを最初に認証する必要があります。IAM 認証の設定の詳細については、「Amazon Neptune での AWS Identity and Access Management (IAM) の概要」を参照してください。

、、Amazon EC2 の例など AWS CLI、一時的な認証情報を使用して認証する方法については AWS Lambda、「」を参照してくださいthe section called “一時認証情報”。

以下のリンクでは、個別のクエリ言語で IAM 認証を使用して Neptune へ接続する方法について詳細な情報を提供します。

IAM 認証で Gremlin を使用する

- the section called “Gremlin コンソール”
- the section called “Gremlin Java”
- the section called “Python の例”

> **Note**
>
> この例は、Gremlin と SPARQL の両方に適用されます。

IAM 認証で openCypher を使用する

- the section called “Gremlin コンソール”
- the section called “Gremlin Java”
- the section called “Python の例”

> **Note**
>
> この例は、Gremlin と SPARQL の両方に適用されます。

IAM 認証で SPARQL を使用する

- the section called “SPARQL Java (RDF4J および Jena)”
- the section called “Python の例”

> **Note**
>
> この例は、Gremlin と SPARQL の両方に適用されます。

Neptune グラフへのアクセスの開始方法

Neptune DB クラスターを作成し、クラスターへの接続を設定したら、次のステップは、クラスターと通信して、データのロードやクエリの実行などを行うことです。そのためには、ほとんどの人が `curl` または `awscurl` コマンドラインツールを使用します。

Neptune エンドポイントと通信するために **`curl`** をセットアップする

このドキュメントの多くの例に示されるように、curl コマンドラインツールは エンドポイントとの通信に役立つオプションです。このツールの詳細については、「curl man page」およびブック「Everything curl」を参照してください。

HTTPS を使用して接続するには、`curl` にアクセスするための適切な証明書が必要となります (これは推奨事項ですが、ほとんどのリージョンでは Neptune での必須事項です)。これらの証明書を取得する方法と、`curl` が使える証明書を認証局 (CA) 証明書ストアに適切にフォーマットする方法については、`curl` ドキュメント内のSSL 証明書の検証 を参照してください。

次に、`CURL_CA_BUNDLE` 環境変数を使用してこの CA 証明書ストアの場所を指定できます。Windows では、`curl` は自動的に `curl-ca-bundle.crt` という名前のファイルを検索します。まず `curl.exe` と同じディレクトリで curl.exe を検索し、次にこのパスの他の場所を検索します。詳細については、「SSL Certificate Verification」を参照してください。

curl が適切な証明書を見つけられる限り、これによって、別のパラメータを必要とすることなく、HTTP 接続のように HTTPS 接続が処理されます。このドキュメントの例はこのシナリオに基づいています。

クエリ言語を使用して Neptune DB クラスター内のグラフデータにアクセスする

接続したら、Gremlin および openCypher クエリ言語を使用して、プロパティグラフを作成してクエリしたり、SPARQL クエリ言語を使用して RDF データを含むグラフを作成してクエリしたりできます。

Neptune がサポートするグラフクエリ言語

- Gremlin は、プロパティグラフのグラフトラバーサル言語です。Gremlin のクエリは個別のステップで構成されたトラバーサルで、各ステップはエッジからノードに従います。詳細については、Apache TinkerPop3 の「Gremlin ドキュメント」を参照してください。

  Gremlin の Neptune 実装は、特に Gremlin-Groovy (シリアル化されたテキストとして送信される Gremlin クエリ) を使用している場合、他の実装とはいくつかの相違点があります。詳細については、「Amazon Neptune の Gremlin 標準への準拠」を参照してください。
- openCypher は、プロパティグラフの宣言型クエリ言語です。当初は Neo4j が 開発し、その後 2015 年にオープンソース化され、Apache 2 オープンソースライセンスの下で openCypher プロジェクトで活用されました。構文は Cypher クエリ言語リファレンス、バージョン 9 で説明されています。
- SPARQL は、RDF データ用の宣言型クエリ言語です。World Wide Web Consortium (W3C) によって標準化され、「SPARQL 1.1 概要」と SPARQL 1.1 クエリ言語」仕様で記述されているグラフパターンマッチングに基づいています。

> **Note**
>
> Neptune のプロパティグラフデータには Gremlin と openCypher の両方を使用してアクセスできますが、SPARQL は使用できません。同様に、RDF データには SPARQL を使用してのみアクセスでき、Gremlin や openCypher ではアクセスできません。

Gremlin を使用して Amazon Neptune のグラフにアクセスする

Gremlin コンソールを使用して、REPL (read-eval-print ループ) 環境で TinkerPop グラフとクエリを試すことができます。

以下のチュートリアルでは、Gremlin コンソールを使用して頂点、エッジ、プロパティなどを Neptune グラフに追加する方法を取り上げ、Neptune 固有の Gremlin 実装のいくつかの違いに着目します。

> **Note**
>
> この例では、次の操作が完了していることを前提としています。
>
> - SSH を使用して Amazon EC2 インスタンスに接続する。
> - DB クラスターを作成する で説明されているように Neptune クラスターを作成する。
> - 「Gremlin コンソールのインストール」で説明されているように Gremlin コンソールをインストールする。

Gremlin コンソールの使用

1. ディレクトリを Gremlin コンソールファイルが解凍されたフォルダに変更します。

```
cd apache-tinkerpop-gremlin-console-3.6.5
```

2. 以下のコマンドを入力して、Gremlin コンソールを実行します。

```
bin/gremlin.sh
```

以下の出力が表示されます。

```
         \,,,/
         (o o)
-----oOOo-(3)-oOOo-----
plugin activated: tinkerpop.server
plugin activated: tinkerpop.utilities
plugin activated: tinkerpop.tinkergraph
gremlin>
```

gremlin> プロンプトが表示されます。このプロンプトで残りのステップを入力します。

3. gremlin> プロンプトで、次のように入力して Neptune DB インスタンスに接続します。

```
:remote connect tinkerpop.server conf/neptune-remote.yaml
```

4. gremlin> プロンプトで、次のように入力してリモートモードに切り替えます。これにより、すべての Gremlin クエリがリモート接続に送信されます。

```
:remote console
```

5. 頂点を追加して、ラベルとプロパティを割り当てます。

```
g.addV('person').property('name', 'justin')
```

頂点には、GUID を含む string ID が割り当てられます。Neptune ではすべての頂点 ID は文字列です。

6. 頂点を追加して、カスタム ID を割り当てます。

```
g.addV('person').property(id, '1').property('name', 'martin')
```

id プロパティは引用符で囲みません。これは、頂点 ID のキーワードです。ここでの頂点 ID は、数字 1 を含む文字列です。

通常のプロパティ名は引用符で囲む必要があります。

7. プロパティを変更するか、プロパティを追加します (まだ追加してない場合)。

```
g.V('1').property(single, 'name', 'marko')
```

ここでは、前のステップからの頂点の name プロパティを変更します。これにより、name プロパティの既存の値はすべて削除されます。

single を指定しなかった場合は、代わりに name プロパティに値が付加されます (まだ付加されていない場合)。

8. プロパティを追加しますが、プロパティにすでに値がある場合は値を付加します。

```
g.V('1').property('age', 29)
```

Neptuneはデフォルトのアクションとしてカーディナリティ (cardinality) を使用します。

このコマンドは age プロパティを追加して値 29 を指定しますが、既存の値は置き換えません。

age プロパティにすでに値がある場合、このコマンドはプロパティに 29 を付加します。たとえば、age プロパティの値が 27 である場合、新しい値は [27, 29] となります。

9. 複数の頂点を追加します。

```
g.addV('person').property(id, '2').property('name', 'vadas').property('age',
 27).iterate()
g.addV('software').property(id, '3').property('name', 'lop').property('lang',
 'java').iterate()
g.addV('person').property(id, '4').property('name', 'josh').property('age',
 32).iterate()
g.addV('software').property(id, '5').property('name', 'ripple').property('lang',
 'java').iterate()
g.addV('person').property(id, '6').property('name', 'peter').property('age', 35)
```

Neptune に複数のステートメントを同時に送信できます。

ステートメントは改行 ('\n')、スペース (' ')、またはセミコロン ('; ') で区切ることができます。区切り記号はなしでもかまいません (たとえば g.addV('person').iterate()g.V() は有効です)。

> **Note**
>
> Gremlin コンソールは改行 ('\n') ごとに個別のコマンドを送信するため、その場合は各コマンドが個別のトランザクションになります。この例では、すべてのコマンドが読みやすくなるように個別の行にあります。Gremlin コンソールから 1 つのコマンドとして送信するには、改行 ('\n') 文字を削除してください。

最後のステートメント以外のステートメントは .next() や .iterate() などの終了ステップで終わる必要があります。そうしないとステートメントは実行されません。Gremlin コンソールでは、これらの終了ステップは不要です。結果をシリアル化する必要がないときはいつでも .iterate を使用します。

一緒に送信されたすべてのステートメントは、1 つのトランザクションに含まれ、まとめて成功または失敗となります。

10. エッジを追加します。

```
g.V('1').addE('knows').to(__.V('2')).property('weight', 0.5).iterate()
g.addE('knows').from(__.V('1')).to(__.V('4')).property('weight', 1.0)
```

ここでは、2 つの異なる方法でエッジを追加しています。

11. Modern グラフの残りの部分を追加します。

```
g.V('1').addE('created').to(__.V('3')).property('weight', 0.4).iterate()
g.V('4').addE('created').to(__.V('5')).property('weight', 1.0).iterate()
g.V('4').addE('knows').to(__.V('3')).property('weight', 0.4).iterate()
g.V('6').addE('created').to(__.V('3')).property('weight', 0.2)
```

12. 頂点を削除します。

```
g.V().has('name', 'justin').drop()
```

name プロパティが justin である頂点を削除します。

> **Important**
>
> ここで停止すると、完全な Apache TinkerPop Modern グラフが表示されます。TinkerPop ドキュメントの「トラバーサル」セクションの例では、最新グラフを使用しています。

13. トラバーサルを実行します。

```
g.V().hasLabel('person')
```

すべての person 頂点を返します。

14. 値付きのトラバーサルを実行します (valueMap())。

```
g.V().has('name', 'marko').out('knows').valueMap()
```

marko "knows" に該当するすべての頂点のキーと値のペアを返します。

15. 複数のラベルを指定します。

```
g.addV("Label1::Label2::Label3")
```

Neptune は、頂点の複数のラベルをサポートしています。ラベルを作成する際、`::` で区切ることで複数のラベルを指定できます。

この例では、1 つの頂点に 3 つの異なるラベルを追加します。

`hasLabel` ステップでは、この頂点を `hasLabel("Label1")`、`hasLabel("Label2")`、および `hasLabel("Label3")` の 3 つのラベルのいずれかと一致させます。

`::` 区切り記号は、この使用のみに予約されます。

`hasLabel` ステップで複数のラベルを指定することはできません。たとえば、`hasLabel("Label1::Label2")` はいずれにも一致しません。

16. 日付 / 時刻を指定します。

```
g.V().property(single, 'lastUpdate', datetime('2018-01-01T00:00:00'))
```

Neptune は Java Date をサポートしていません。代わりに、 `datetime()` 関数を使用します。`datetime()` は ISO8061 準拠 `datetime` 文字列を受け入れます。

サポートされている形式は、`YYYY-MM-DD`, `YYYY-MM-DDTHH:mm`、`YYYY-MM-DDTHH:mm:SS`、`YYYY-MM-DDTHH:mm:SSZ` です。

17. 頂点、プロパティ、またはエッジを削除します。

```
g.V().hasLabel('person').properties('age').drop().iterate()
g.V('1').drop().iterate()
g.V().outE().hasLabel('created').drop()
```

ここでは、いくつかの削除例を示しています。

> **Note**
>
> `.next()` ステップは `.drop()` では機能しません。代わりに `.iterate()` を使用します。

18. 完了したら、次のように入力して Gremlin コンソールを終了します。

```
:exit
```

> **Note**
>
> 各ステートメントを区切るには、セミコロン (;) または改行文字 (\n) を使用します。
> 最終的なトラバーサルに先行する各トラバーサルは、`iterate()` を実行して終わる必要があります。最終的なトラバーサルからのデータのみが返されます。

openCypher を使用して Amazon Neptune のグラフにアクセスする

openCypher の使用を開始するには、「」を参照するかopenCypher、Neptune グラフノートブックリポジトリ の openCypher ノートブックを使用します。 GitHub https://github.com/aws/graph-notebook/tree/main/src/graph_notebook/notebooks

RDF および SPARQL を使用して、Amazon Neptune のグラフにアクセスする

SPARQL は、ウェブ用に設計されたグラフデータ形式であるリソース記述フレームワーク (RDF) のためのクエリ言語です。Amazon Neptune は、SPARQL 1.1 と互換性があります。つまり、Neptune DB インスタンス に接続して、SPARQL 1.1 クエリ言語仕様で記述されたクエリ言語を使用してグラフにクエリを実行できるということです。

SPARQL のクエリは、返す変数を指定する `SELECT` 句と、グラフで一致するデータを指定する `WHERE` 句で構成されます。SPARQL クエリに慣れていない場合は、SPARQL 1.1 クエリ言語にある「シンプルなクエリの書き込み」を参照してください。

Neptune DB インスタンスへの SPARQL クエリ用の HTTP エンドポイントは `https://`*`your-neptune-endpoint`*`:`*`port`*`/sparql` です。

SPARQL に接続するには

1. Neptune クラスターの SPARQL エンドポイントは、 AWS CloudFormation スタックの出力セクションのSparqlEndpoint項目から取得できます。
2. 以下のように入力して、HTTP `POST` と curl コマンドを使用して SPARQL **`UPDATE`** を送信します。

```
curl -X POST --data-binary 'update=INSERT DATA { <https://test.com/s> <https://
test.com/p> <https://test.com/o> . }' https://your-neptune-endpoint:port/sparql
```

前述の例では、次のトリプルを SPARQL デフォルトのグラフに挿入します。<https://test.com/s> <https://test.com/p> <https://test.com/o>

3. 以下のように入力して、HTTP POST と curl コマンドを使用して SPARQL **QUERY** を送信します。

```
curl -X POST --data-binary 'query=select ?s ?p ?o where {?s ?p ?o} limit 10'
 https://your-neptune-endpoint:port/sparql
```

前述の例では、10 の制限がある ?s ?p ?o クエリを使用して、グラフのトリプル (主語 - 述語 - 目的語) のうち最大 10 個を返します。その他の対象にクエリを実行するには、別の SPARQL クエリで置き換えます。

> **Note**
>
> SELECT および ASK クエリの場合、レスポンスのデフォルトの MIME タイプは application/sparql-results+json です。
> CONSTRUCT および DESCRIBE クエリの場合、レスポンスのデフォルトの MIME タイプは application/n-quads です。
> すべての使用可能な MIME タイプのリストについては、「SPARQL HTTP API」を参照してください。

Neptune にデータをロードする

Amazon Neptune は外部ファイルから Neptune DB インスタンスに直接データをロードするプロセスを提供します。多数の INSERT ステートメント、addV および addE ステップ、または、API 呼び出しを実行する代わりに、このプロセスを使用できます。

追加のロード情報へのリンクを次に示します。

- データをロードする方法 - データのロード
- バルクローダーによってサポートされるデータ形式 — the section called “データ形式”
- ロードの例 – the section called “ロードの例”

Amazon Neptune のモニタリング

Amazon Neptune では、次のモニタリング方法がサポートされています。

- Amazon CloudWatch – Amazon Neptune は にメトリクスを自動的に送信 CloudWatch し、CloudWatch アラームもサポートします。詳細については、「the section called “の使用 CloudWatch”」を参照してください。
- AWS CloudTrail — Amazon Neptune は、 を使用した API ログ記録をサポートしています CloudTrail。詳細については、「the section called “を使用した Neptune API コールのログ記録 AWS CloudTrail”」を参照してください。
- タグ付け – タグを使用してメタデータを Neptune リソースに追加し、タグに基づいて使用量を追跡します。詳細については、「the section called “Neptune リソースにタグを付ける”」を参照してください。
- 監査ログファイル – Neptune コンソールを使用して、データベースログファイルを表示、ダウンロード、調査します。詳細については、「the section called “Neptune による監査ログ ”」を参照してください。
- インスタンスステータス – Neptune のインスタンスのグラフデータベースエンジンの状態をチェックし 、インストールされているエンジンのバージョンを確認して、インスタンスステータス API を使用して、他のエンジンステータス情報を取得します。

Neptune におけるトラブルシューティングとベストプラクティス

以下のリンクは、Amazon Neptune に関する問題を解決するのに役立ちます。

- ベストプラクティス – 一般的な問題に対する解決策およびパフォーマンスに関する提案については、ベストプラクティス を参照してください。
- サービスエラー – 管理 API とグラフデータベース接続の両方のエラーのリストについては、Neptune エラー を参照してください。
- サービスの制限 – Neptune での制限については、Neptune の制限 を参照してください。
- エンジンリリース – リリースノートを含むグラフエンジンリリースの詳細については、エンジンリリース を参照してください。
- サポートフォーラム – Neptune についてのディスカッションに参加するには、Amazon Neptune フォーラムを参照してください。

- 料金　— Amazon Neptune の使用料金の詳細については、Amazon Neptune 料金表を参照してください。
- AWS サポート — エキスパートからのヘルプとガイダンスについては、「」を参照してくださいAWS Support。

Neptune グローバルデータベースの作成

Amazon Neptune グローバルデータベースは複数の AWS リージョン にまたがるため、低レイテンシーのグローバル読み取りが実現し、AWS リージョン 全体に影響が及ぶ可能性のある停止がまれに起きても、すばやい復旧が可能です。

Neptune グローバルデータベースは、1 つのリージョンにあるプライマリ DB クラスターと、異なるリージョンに最大 5 つのセカンダリ DB クラスターがあります。

書き込みはプライマリリージョンでのみ可能です。セカンダリリージョンは読み取りのみをサポートします。各セカンダリリージョンは最大 16 個のリーダーインスタンスを持つことができます。

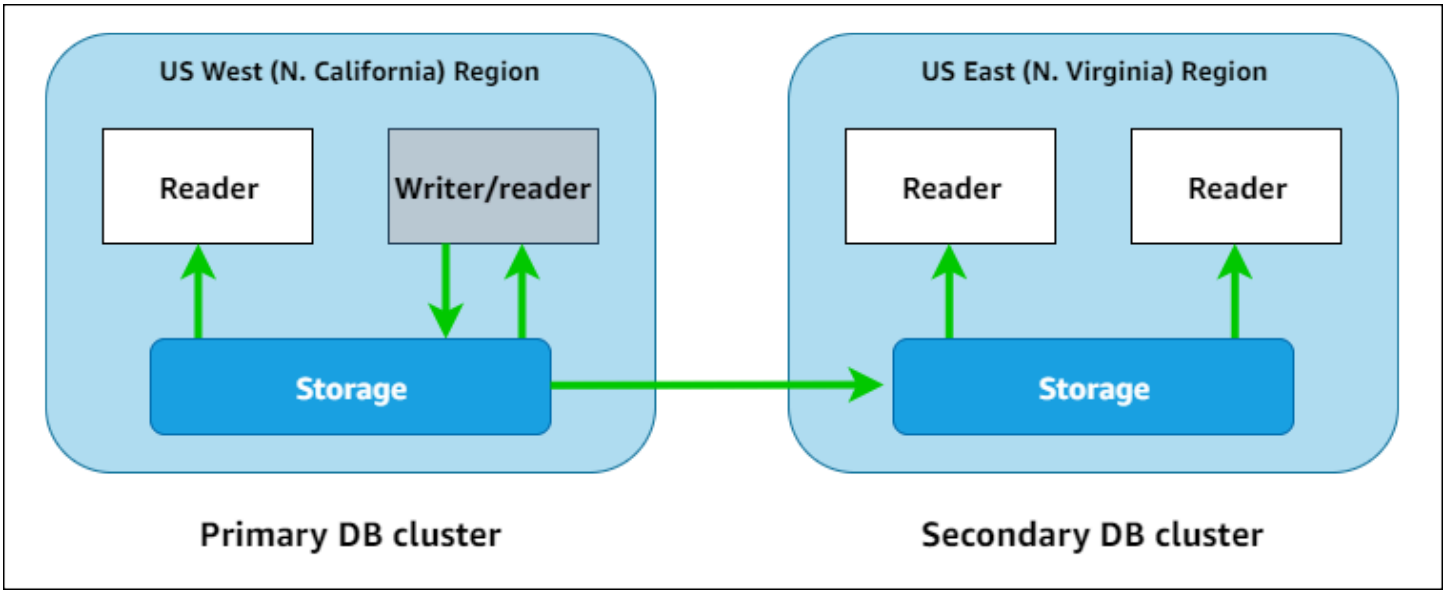
トピック

Amazon Neptune のグローバルデータベースの概要

Neptune グローバルデータベースを使用することで、複数の AWS リージョンにまたがる単一のデータベースで、グローバルに分散したアプリケーションを実行することができます。

Neptune グローバルデータベースは、プライマリ AWS リージョン の 1 つの DB クラスター (データが書き込まれる) と、セカンダリ AWS リージョン の最大 5 つの読み取り専用 DB クラスターで構成されます。プライマリ DB クラスターで書き込み操作を実行すると、Neptune は、書き込まれたデータを専用のインフラストラクチャを使用して、すべてのセカンダリ DB クラスターにレプリケートします。レイテンシーは通常 1 秒未満です。

次の図は、2 つの AWS リージョン にまたがるサンプルのグローバルデータベースを示しています。



各セカンダリクラスターは、読み取り専用のワークロードを処理するために 1 つまたは複数のリードレプリカインスタンスを追加することで、個別にスケールアップすることができます。

書き込み操作を実行するには、プライマリ DB クラスターの DB クラスターエンドポイントに接続する必要があります。書き込み操作はプライマリクラスターのみが実行できます。次に、上の図に示すように、レプリケーションはデータベースエンジンではなく、クラスターストレージボリュームによって実行されます。

Neptune グローバルデータベースは、ワールドワイドなフットプリントを持つアプリケーション向けに設計されています。読み取り専用セカンダリ DB クラスターは、アプリケーションユーザーのさらに近くで読み取り操作をサポートします。

Neptune グローバルデータベースは、フェイルオーバーに対して 2 つの異なるアプローチをサポートします。

- プライマリリージョンの障害から回復するには、手動による計画外のデタッチとプロモートプロセスを使用します。このプロセスでは、セカンダリクラスタの 1 つをデタッチしてスタンドアロンクラスタにしてから、新しいプライマリクラスタに昇格させます。
- メンテナンスなどの計画された運用手順には、管理された計画されたフェイルオーバーを使用します。この方法では、データを損失することなく、プライマリクラスターをセカンダリリージョンの 1 つに再配置します。

Amazon Neptune でグローバルデータベースを使用するメリット

グローバルデータベースを使用すると、次の利点を得ることができます。

- ローカルレイテンシーによるグローバルな読み取り — 世界中にオフィスを持つ企業は、グローバルデータベースにより、セカンダリリージョンにあるオフィスがローカルレイテンシーで自分のリージョンにあるデータにアクセスできます。
- スケーラブルなセカンダリ Neptune DB クラスター - リードレプリカ DB インスタンスを追加することで、セカンダリクラスターをスケールできます。セカンダリクラスターは読み取り専用なので、それぞれが、通常の 15 ではなく最大 16 のリードレプリカをサポートできます。
- セカンダリ DB クラスターへの迅速なレプリケーション — プライマリからセカンダリ DB クラスターへのレプリケーションは高速であり、レイテンシーは通常 1 秒未満で、プライマリ DB クラスターのパフォーマンスにほとんど影響しません。レプリケーションはストレージレベルで実行されるため、DB インスタンスのリソースはアプリケーションの読み取り/書き込みワークロードに完全に利用できます。
- リージョン全体の停止からの回復 - セカンダリ DB クラスターを使用すると、従来のレプリケーションソリューションよりも迅速に (低い RTO)、少ないデータ損失 (低い RPO) でプライマリクラスタを新しいリージョンに迅速に移動できます。

Amazon Neptune のグローバルデータベースのの制限

現在、 グローバルデータベースには以下の制限があります。

- Neptune グローバルデータベースは、次の AWS リージョン でのみ使用できます。
  - 米国東部 (バージニア北部): `us-east-1`
  - 米国東部 (オハイオ): `us-east-2`
  - 米国西部 (北カリフォルニア): `us-west-1`
  - 米国西部 (オレゴン): `us-west-2`
  - 欧州 (アイルランド): `eu-west-1`
  - 欧州 (ロンドン): `eu-west-2`
  - アジアパシフィック (東京): `ap-northeast-1`
- Neptune グローバルデータベースは、現在、セカンダリ DB クラスターの自動スケーリングをサポートしていません。

- グローバルデータベースのメジャーバージョンアップグレードを実行している間、グローバルデータベースクラスターにカスタムパラメータグループを適用できません。代わりに、グローバルクラスターの各リージョンにカスタムパラメータグループを作成してから、アップグレード後に手動でリージョンクラスターに適用します。
- グローバルデータベースの DB クラスターを個別に停止または開始することはできません。
- セカンダリ DB クラスターのリードレプリカインスタンスは、特定の場合に再起動することが可能です。プライマリクラスターのライター DB インスタンスが再起動またはフェイルオーバーすると、セカンダリリージョン内のすべてのインスタンスも再起動します。セカンダリクラスターは、その後、すべてのインスタンスがプライマリ DB クラスターのライターインスタンスと再び同期するまで使用できません。

Amazon Neptune でのグローバルデータベースのセットアップ

Neptune グローバルデータベースは、次のいずれかの方法で作成できます。

- すべて新しい DB クラスターとインスタンスを持つグローバルデータベースを作成します。
- 既存の Neptune DB クラスターをプライマリクラスターとして使用してグローバルデータベースを作成します。

トピック

- Amazon Neptune のグローバルデータベースの設定要件
- AWS CLI を使用して Amazon Neptune にグローバルデータベースを作成する
- 既存の DB クラスターをグローバルデータベースに変える
- Amazon Neptune のプライマリリージョンにセカンダリグローバルデータベースリージョンを追加する
- Neptune グローバルデータベースへの接続

Amazon Neptune のグローバルデータベースの設定要件

Neptune グローバルデータベースは少なくとも 2 つの AWS リージョン にまたがります。プライマリ AWS リージョン には、1 つのライターインスタンスを持つ Neptune DB クラスターが含まれます。1 つから 5 つのセカンダリ AWS リージョン のそれぞれに、リードレプリカインスタンス全体で構成される読み取り専用の Neptune DB クラスターが含まれます。少なくとも 1 つのセカンダリ AWS リージョン が必要です。

グローバルデータベースを構成する Neptune DB クラスターには、以下の固有の要件があります。

- DB インスタンスクラスの要件 — グローバルデータベースには、`db.r5.large` インスタンスタイプなど、メモリ集約的ワークロードに最適化された `r5` または `r6g` DB インスタンスクラスが必要です。
- AWS リージョン の要件 - グローバルデータベースには、1 つの AWS リージョン に 1 つのプライマリ Neptune DB クラスターと、別のリージョンに少なくとも 1 つのセカンダリ Neptune DB クラスターが必要です。最大 5 つのセカンダリ読み取り専用 Neptune DB クラスターを作成でき、それぞれが異なるリージョンになければなりません。つまり、Neptune グローバルデータベースの 2 つの Neptune DB クラスターを同じ AWS リージョン に置くことはできません。
- エンジンバージョン要件 — グローバルデータベース内のすべての DB クラスターで使用される Neptune エンジンのバージョンは同じでなければならず、かつ、`1.2.0.0` 以上である必要があります。新しいグローバルデータベース、クラスター、またはインスタンスを作成するときにエンジンバージョンを指定しなかった場合、最新のエンジンバージョンが使用されます。

> **Important**
>
> DB クラスターパラメータグループはグローバルデータベースの DB クラスターごとに個別に設定できますが、セカンダリクラスターをプライマリに昇格させる必要がある場合に予期しない動作の変化を避けるため、クラスター間で設定を一定に保つことが最善です。たとえば、すべての DB クラスターでオブジェクトインデックス、ストリームなどに同じ設定を使用します。

AWS CLI を使用して Amazon Neptune にグローバルデータベースを作成する

> **Note**
>
> このセクションの例は、行延長文字としてバックスラッシュ (\) を使用するという UNIX の規則に従っています。Windows の場合は、バックスラッシュをキャレット (^) に置き換えてください。

AWS CLI を使用してグローバルデータベースを作成するには

1. まず、create-global-cluster AWS CLI コマンド (CreateGlobalCluster API をラップする) を使用して、空のグローバルデータベースを作成します。プライマリにする AWS リージョンの名前を指定し、データベースエンジンとして Neptune を設定し、オプションで、使用するエンジンのバージョンを指定します (これはバージョン 1.2.0.0 以上である必要があります)。

```
aws neptune create-global-cluster
  --region (primary region, such as us-east-1) \
  --global-cluster-identifier (ID for the global database) \
  --engine neptune \
  --engine-version (engine version; this is optional)
```

2. グローバルデータベースが使用可能になるまで数分かかることがあるため、次のステップに進む前に、describe-global-clusters CLI コマンド (DescribeGlobalClusters API をラップする) を使用して、グローバルデータベースが使用可能であることを確認します。

```
aws neptune describe-global-clusters \
  --region (primary region) \
  --global-cluster-identifier (global database ID)
```

3. Neptune グローバルデータベースが使用可能になったら、新しい Neptune DB クラスターを作成してプライマリクラスターにすることができます。

```
aws neptune create-db-cluster \
  --region (primary region) \
  --db-cluster-identifier (ID for the primary DB cluster) \
  --engine neptune \
  --engine-version (engine version; must be >= 1.2.0.0) \
  --global-cluster-identifier (global database ID)
```

4. describe-db-clusters AWS CLI コマンドを使用して、新しい DB クラスターにプライマリ DB インスタンスを追加する準備ができていることを確認します。

```
aws neptune describe-db-clusters \
  --region (primary region) \
  --db-cluster-identifier (primary DB cluster ID)
```

結果に "Status": "available" のステータスが表示されたら、次のステップに進みます。

5. create-db-instance AWS CLI コマンドを使用して、プライマリクラスターのプライマリ DB インスタンスを作成します。db.r5.large など、メモリ最適化された r5 または r6g インスタンスタイプのいずれかを使用する必要があります。

```
aws neptune create-db-instance \
  --region (primary region) \
  --db-cluster-identifier (primary cluster ID) \
  --db-instance-class (instance class) \
  --db-instance-identifier (ID for the DB instance) \
  --engine neptune \
  --engine-version (optional: engine version)
```

> **Note**
>
> Neptune バルクローダーを使用して新しいプライマリ DB クラスターにデータを追加する予定がある場合は、セカンダリリージョンを追加する前に追加してください。その方が、グローバルデータベースが完全にセットアップされた後にバルクロードを実行するよりも速く、コスト効率も高くなります。

次に、AWS CLI を使用してセカンダリリージョンを追加する で説明されているように、新しいグローバルデータベースに 1 つ以上のセカンダリリージョンを追加します。

既存の DB クラスターをグローバルデータベースに変える

既存の DB クラスターをグローバルデータベースに変換するには、create-global-cluster AWS CLI コマンドを使用して、既存の DB クラスターと同じ AWS リージョン に新しいグローバルデータベースを作成し、その --source-db-cluster-identifier パラメータをそこにある既存のクラスターの Amazon リソースネーム (ARN) に設定します。

```
aws neptune create-global-cluster \
  --region (region where the existing cluster is located) \
  --global-cluster-identifier (provide an ID for the new global database) \
  --source-db-cluster-identifier (the ARN of the existing DB cluster) \
  --engine neptune \
  --engine-version (engine version; this is optional)
```

次に、AWS CLI を使用してセカンダリリージョンを追加する で説明されているように、新しいグローバルデータベースに 1 つ以上のセカンダリリージョンを追加します。

スナップショットから復元された DB クラスターをプライマリクラスターとして使用する

スナップショットから復元された DB クラスターを Neptune グローバルデータベースに変換できます。復元が完了したら、作成した DB クラスターを上記のように新しいグローバルデータベースのプライマリクラスターにします。

Amazon Neptune のプライマリリージョンにセカンダリグローバルデータベースリージョンを追加する

Neptune グローバルデータベースには、プライマリ DB クラスターのほかに、異なる AWS リージョン に少なくとも 1 つのセカンダリ Neptune DB クラスターが必要です。プライマリ DB クラスターには、最大 5 つのセカンダリ DB クラスターをアタッチできます。

セカンダリ DB クラスターを追加するたびに、プライマリクラスターに配置できるリードレプリカインスタンスの最大数が 1 つ減ります。例えば、セカンダリクラスターが 4 つある場合、プライマリクラスターに配置できるリードレプリカインスタンスの最大数は 15 - 4 = 11 です。つまり、プライマリ DB クラスターにリーダーインスタンスが既に 14 あり、セカンダリクラスターが 1 つある場合、これ以上セカンダリクラスターを追加できません。

AWS CLI を使用して Neptune のグローバルデータベースにセカンダリリージョンを追加する

AWS CLI を使用して Neptune グローバルデータベースにセカンダリ AWS リージョン を追加するには

1. create-db-cluster AWS CLI コマンドを使用して、プライマリクラスターとは異なるリージョンに新しい DB クラスターを作成し、その `--global-cluster-identifier` パラメータを設定して、グローバルデータベースの ID を指定します。

   ```
   aws neptune create-db-cluster \
     --region (the secondary region) \
     --db-cluster-identifier (ID for the new secondary DB cluster) \
     --global-cluster-identifier (global database ID)
     --engine neptune \
     --engine-version (optional: engine version)
   ```

2. describe-db-clusters AWS CLI コマンドを使用して、新しい DB クラスターにプライマリ DB インスタンスを追加する準備ができていることを確認します。

```
aws neptune describe-db-clusters \
  --region (primary region) \
  --db-cluster-identifier (primary DB cluster ID)
```

結果に "Status": "available" のステータスが表示されたら、次のステップに進みます。

3. create-db-instance AWS CLI コマンドを使用し、r5 または r6g インスタンスクラスのインスタンスタイプを使用して、プライマリクラスターのプライマリ DB インスタンスを作成します。

```
aws neptune create-db-instance \
  --region (secondary region) \
  --db-cluster-identifier (secondary cluster ID) \
  --db-instance-class (instance class) \
  --db-instance-identifier (ID for the DB instance) \
  --engine neptune \
  --engine-version (optional: engine version)
```

> **Note**
>
> セカンダリリージョンで大量の読み取りリクエストを処理する予定がなく、データを確実にバックアップしておくことが主な目的であれば、DB インスタンスのないセカンダリ DB クラスターを作成できます。これにより、セカンダリクラスターのストレージに対してのみ支払うことになり、Neptune はプライマリ DB クラスターのストレージと同期し続けるため、コストを節約できます。

Neptune グローバルデータベースへの接続

Neptune グローバルデータベースへの接続方法は、データベースへの書き込みと、データベースからの読み取りの、いずれを必要とするのかに応じて異なります。

- 読み取り専用のリクエストまたはクエリの場合、AWS リージョン にある Neptune クラスターのリーダーエンドポイントに接続します。
- ミューテーションクエリを実行するには、プライマリ DB クラスターのクラスターエンドポイントに接続します。このエンドポイントはアプリケーションとは異なる AWS リージョン にある場合があります。

Amazon Neptune グローバルデータベースの管理

管理された計画済みのフェイルオーバープロセスを除いて、ほとんどの管理操作は、Neptune グローバルデータベースを構成する個々のクラスターに対して実行します。管理された計画済みのフェイルオーバープロセスは、個々の Neptune DB クラスターではなく、Neptune グローバルデータベースでのみ使用できます。詳細については、「Neptune グローバルデータベースの管理された計画的なフェイルオーバーを実行する」を参照してください。

プライマリリージョンの予期しない停止から Neptune グローバルデータベースを復元するには、計画外の停止に備えて Neptune グローバルデータベースをデタッチして昇格する を参照してください。

DB クラスターパラメータグループはグローバルデータベースの Neptune クラスターごとに個別に設定できますが、セカンダリクラスターをプライマリに昇格させる必要がある場合に予期しない動作の変化を避けるため、すべてのクラスター間で設定を一定に保つことが最善です。例えば、すべての DB クラスターでオブジェクトインデックス、ストリームなどに同じ設定を使用します。

Neptune グローバルデータベースからのクラスターの削除

グローバルデータベースから DB クラスターを削除するには、いくつかの理由があります。例:

- プライマリクラスターが機能しなくなったり分離されたりした場合は、グローバルデータベースから削除して、新しいグローバルデータベースの作成に使用できるスタンドアロンのプロビジョニング済みクラスターにすることができます (「計画外の停止に備えて Neptune グローバルデータベースをデタッチして昇格する」を参照)。
- グローバルデータベースを削除するには、まず、プライマリが最後に残るように、関連付けられているすべてのクラスターを削除 (デタッチ) する必要があります (「Neptune グローバルデータベースの削除」を参照)。

remove-from-global-cluster CLI コマンド (RemoveFromGlobalCluster API をラップする) を使用して、Neptune DB クラスターをグローバルデータベースからデタッチできます。

```
aws neptune remove-from-global-cluster \
  --region (region of the cluster to remove) \
  --global-cluster-identifier (global database ID) \
  --db-cluster-identifier (ARN of the cluster to remove)
```

その後、デタッチされた DB クラスターは、スタンドアロン DB クラスターになります。

Neptune グローバルデータベースの削除

グローバルデータベースおよび関連付けられているクラスターを 1 回のステップで削除することはできません。代わりに、そのコンポーネントを 1 つずつ削除する必要があります。

1. クラスターの削除 で説明されているように、グローバルデータベースからすべてのセカンダリ DB クラスターをデタッチします。必要に応じて、個別に削除できるようになりました。
2. グローバルデータベースからプライマリ DB クラスターをデタッチします。
3. プライマリクラスターからすべてのリードレプリカ DB インスタンスを削除します。
4. プライマリクラスターからプライマリ (ライター) DB インスタンスを削除します。コンソールでこれを行うと、DB クラスターも削除されます。
5. グローバルデータベース自体を削除します。AWS CLI を使用してこれを実行するには、次のように delete-global-cluster CLI コマンド (DeleteGlobalCluster API をラップする) を使用します。

```
aws neptune delete-global-cluster \
  --region (region of the DB cluster to delete) \
  --global-cluster-identifier (global database ID)
```

Neptune グローバルデータベースの変更

DB クラスターパラメータグループはグローバルデータベースの DB クラスターごとに個別に設定できますが、セカンダリクラスターをプライマリに昇格させる必要がある場合に予期しない動作の変化を避けるため、クラスター間で設定を一定に保つことが最善です。

modify-global-cluster CLI コマンド (ModifyGlobalCluster API をラップする) を使用して、グローバルデータベース自体の設定を変更できます。例えば、次のようにグローバルデータベース ID を変更すると同時に削除保護を無効にできます。

```
aws neptune modify-global-cluster \
  --region (region of the DB cluster to modify) \
  --global-cluster-identifier (current global database ID) \
  --new-global-cluster-identifier (new global database ID to assign) \
  --deletion-protection false
```

Neptune グローバルデータベースでフェイルオーバーを使用する

Neptune グローバルデータベースは、スタンドアロン Neptune DB クラスターよりも包括的なフェイルオーバー機能を提供します。グローバルデータベースを使用することにより、災害に対する計画と復旧をかなり迅速に実行できます。ディザスタリカバリは、一般に、目標復旧時間 (RTO) と目標復旧時点 (RPO) の評価を使用して評価されます。

- 目標復旧時間 (RTO) — 災害後にシステムが稼働状態に戻るまでの時間。つまり、RTO はダウンタイムを測定します。Neptune グローバルデータベースの場合、RTO は分単位で行えます。
- 目標復旧時点 (RPO) — データが失われる時間の長さ。Neptune グローバルデータベースの場合、RPO は通常、秒単位で測定されます (「Neptune グローバルデータベースの管理された計画的なフェイルオーバーを実行する」を参照)。

Neptune グローバルデータベースの場合、フェイルオーバーに対して 2 つの異なるアプローチがあります。

- デタッチと昇格 (手動の計画外の回復) — 計画外の停止から回復したり、ディザスタリカバリテスト (DR テスト) を行うには、グローバルデータベース内のセカンダリ DB クラスターの 1 つでクロスリージョンデタッチと昇格を実行します。この手動プロセスの RTO (目標復旧時間) は、デタッチと昇格 に示すタスクの実行速度によって異なります。RPO は、通常、秒単位ですが、これは障害発生時のネットワーク経由でのストレージレプリケーションの遅延によって異なります。
- 管理された計画的フェイルオーバー — このアプローチは、グローバルデータベースのプライマリ DB クラスターをセカンダリリージョンのいずれかに移転するなど、運用保守やその他の計画的な運用手順を目的としています。このプロセスは、他の変更を行う前にセカンダリ DB クラスターとプライマリクラスターを同期するため、RPO は事実上 0 です (すなわち、データ損失はありません)。「Neptune グローバルデータベースの管理された計画的なフェイルオーバーを実行する」を参照してください。

計画外の停止に備えて Neptune グローバルデータベースをデタッチして昇格する

非常にまれに、Neptune グローバルデータベースのプライマリ AWS リージョン で予期しない停止が発生した場合、プライマリ Neptune DB クラスターとそのライターノードが使用できなくなり、プライマリクラスターとセカンダリクラスター間のレプリケーションが停止します。ダウンタイム (RTO) とデータ損失 (RPO) の両方を最小限に抑えるには、リージョン間のデタッチと昇格を迅速に行って、グローバルデータベースを再構築します。

> **Tip**
>
> 使用する前に、このプロセスを理解し、リージョン全体の問題が最初に発生したらすぐに処理を進めるための計画を立てておくとよいでしょう。
>
> - フェイルオーバーが必要な場合にラグタイムが最も少ないセカンダリリージョンを特定できるように、Amazon CloudWatch を定期的に使用して、セカンダリクラスターのラグタイムを追跡します。
> - プランのテストをして、手順が完全かつ正確であることを確認します。
> - シミュレートされた環境を使用して、スタッフがトレーニングを受け、必要になった場合に DR フェイルオーバーを迅速に実行できるよう準備されていることを確認します。

プライマリリージョンで予期しない停止が発生した後にセカンダリクラスターにフェイルオーバーするには

1. プライマリ DB クラスターでのミューテーションクエリやその他の書き込み操作の発行を停止します。
2. グローバルデータベースの新しいプライマリ DB クラスターとして使用するセカンダリ AWS リージョン の DB クラスターを特定します。グローバルデータベースに 2 つ以上のセカンダリ AWS リージョン がある場合、遅延時間が最も少ないセカンダリクラスターを選択します。
3. 選択したセカンダリ DB クラスターを Neptune グローバルデータベースからデタッチします。

   Neptune グローバルデータベースからセカンダリ DB クラスターを削除すると、プライマリからこのセカンダリへのレプリケーションが直ちに停止し、完全な読み取り/書き込み機能を備えたスタンドアロンの DB クラスターに昇格します。グローバルデータベース内の他のセカンダリクラスターは引き続き使用可能で、アプリケーションからの読み取り呼び出しを受け入れることができます。

   Neptune グローバルデータベースを再作成する前に、クラスター間のデータの不整合を避けるために、他のセカンダリクラスターもデタッチする必要があります (「クラスターの削除」を参照)。
4. 新しいプライマリクラスターとして選択したスタンドアロン Neptune DB クラスターに、新しいエンドポイントを使用して、すべての書き込み操作を送信するように、アプリケーションを再設定します。Neptune グローバルデータベースの作成時にデフォルトの名前を受け入れた場合は、アプリケーションでクラスターのエンドポイント文字列から `-ro` を削除することで、エンドポイントを変更できます。

例えば、セカンダリクラスターのエンドポイント `my-global.cluster-ro-aaaaaabbbbbb.us-west-1.neptune.amazonaws.com` は、そのクラスターがグローバルデータベースからデタッチされたときに `my-global.cluster-aaaaaabbbbbb.us-west-1.neptune.amazonaws.com` になります。

この Neptune DB クラスターは、次の手順でリージョンの追加を開始すると、新しい Neptune グローバルデータベースのプライマリクラスターになります。

5. DB クラスターに AWS リージョン を追加します。これを行うと、プライマリからセカンダリへのレプリケーションプロセスがスタートされます。「Amazon Neptune のプライマリリージョンにセカンダリグローバルデータベースリージョンを追加する」を参照してください。
6. 必要に応じて、AWS リージョン を追加して、アプリケーションのサポートに必要なトポロジを再作成します。

これらの変更を行う前、変更中、および変更後に、アプリケーションの書き込みが正しい Neptune DB クラスターに送信されていることを確認してください。これにより、Neptune グローバルデータベース内の DB クラスター間のデータの不整合 (スプリットブレイン問題) が回避されます。

Neptune グローバルデータベースの管理された計画的なフェイルオーバーを実行する

管理された計画的フェイルオーバーを使用すると、必要に応じて、Neptune グローバルデータベースのプライマリクラスターを別の AWS リージョン に再配置できます。組織によっては、プライマリクラスターの位置を定期的にローテーションしたい場合があります。

> **Note**
>
> ここで説明されている管理された計画的フェイルオーバープロセスは、正常な Neptune グローバルデータベースでの使用を目的としています。予期しない停止から復旧したり、ディザスタリカバリ (DR) のテストをしたりするには、代わりにデタッチと昇格プロセスに従ってください。

管理された計画的フェイルオーバー中、プライマリクラスターは選択したセカンダリリージョンにフェイルオーバーされ、グローバルデータベースの既存のレプリケーショントポロジが維持されます。管理された計画的フェイルオーバープロセスを開始する前に、グローバルデータベースはすべてのセカンダリクラスターをプライマリクラスターと同期します。すべてのクラスターが同期したこと

を確認すると、管理された計画的なフェイルオーバーがスタートします。プライマリリージョンの DB クラスターは読み取り専用になり、選択したセカンダリクラスターは、読み取り専用インスタンスの 1 つを完全なライターステータスに昇格して、クラスターがプライマリクラスターのロールを引き受けることができます。プロセスの開始時にすべてのセカンダリクラスターがプライマリと同期されているため、新しいプライマリは、データを失うことなく、 グローバルデータベースの操作を続行します。プライマリクラスターと選択したセカンダリクラスターが新しいロールを引き受ける間、短時間だけ、データベースは使用できなくなります。

アプリケーションの可用性を最適化するには、ピーク時以外の、プライマリ DB クラスターへの書き込みが最小限のときに、フェイルオーバーを実行します。また、フェイルオーバーの開始前に次のステップを実行します。

- プライマリクラスターへの書き込みを減らすために、できる限りアプリケーションをオフラインにします。
- グローバルデータベース内のすべてのセカンダリ Neptune DB クラスターのラグタイムを確認し、全体のラグタイムが最も短いセカンダリを選択してプライマリにします。Amazon CloudWatch を使用して、すべてのセカンダリに対する `NeptuneGlobalDBProgressLag` メトリクスを表示します。このメトリクスは、セカンダリがプライマリ DB クラスターからどのくらい遅れているか (ミリ秒単位) を示します。その値は、Neptune がフェイルオーバーの完了までにかかる時間に正比例します。つまり、ラグ値が大きいほどフェイルオーバー停止時間が長くなるため、ラグの少ないセカンダリを選択してください。詳細については、「Neptune CloudWatch メトリクス」を参照してください。

管理された計画的フェイルオーバー中、選択したセカンダリ DB クラスターは、プライマリとしての新しいロールに昇格されますが、プライマリ DB クラスターの完全な設定を継承するわけではありません。構成の不一致は、パフォーマンスの問題、ワークロードの非互換性、およびその他の異常な動作につながる可能性があります。このような問題を回避するには、フェイルオーバーの前に、グローバルデータベースクラスター間の次のような設定の違いを解決してください。

- 新しいプライマリのパラメータを、現在のプライマリのパラメータと一致するように設定します。
- 監視ツール、オプション、アラームの設定 — 新しいプライマリとなる DB クラスターに、現在のプライマリと同じロギング機能、アラームなどを設定します。
- 他の AWS サービスとの統合を設定する - Neptune グローバルデータベースを AWS Identity and Access Management (IAM)、Amazon S3、または AWS Lambda など、AWS サービスと統合する場合は、必要に応じて、これらが新しいプライマリ DB クラスターと統合するように設定されていることを確認してください。

フェイルオーバープロセスが完了し、昇格した DB クラスターがグローバルデータベースの書き込み操作を処理できるようになったら、新しいプライマリの新しいエンドポイントを使用するようにアプリケーションを変更してください。

AWS CLI を使用して、管理された計画的フェイルオーバーを開始する

failover-global-cluster CLI コマンド (FailoverGlobalCluster API をラップする) を使用して、Neptune グローバルデータベースをフェイルオーバーします。

```
aws neptune failover-global-cluster \
  --region (the region where the primary cluster is located) \
  --global-cluster-identifier (global database ID) \
  --target-db-cluster-identifier (the ARN of the secondary DB cluster to promote)
```

> **Note**
>
> failover-global-cluster API は、プレビューでは使用できません。GA リリースの一部となる予定です。

CloudWatch メトリクスを使用した Neptune グローバルデータベースの監視

Neptune は、Neptune グローバルデータベースの監視に使用できる次の CloudWatch メトリクスをサポートしています。

- **GlobalDbDataTransferBytes** — Neptune グローバルデータベース内のプライマリ AWS リージョン からセカンダリ AWS リージョン に転送された REDO ログデータのバイト数。
- **GlobalDbReplicatedWriteIO** - グローバルデータベースのプライマリ AWS リージョン からセカンダリ AWS リージョンのクラスターボリュームに複製された書き込み I/O 操作の数。

  Neptune グローバルデータベース内の各 DB クラスターの課金計算では、そのクラスター内で実行された書き込みを説明するために VolumeWriteIOPS メトリクスが使用されます。プライマリ DB クラスターの場合、課金計算では、NeptuneGlobalDbReplicatedWriteIO が使用され、セカンダリ DB クラスターへのクロスリージョンレプリケーションが考慮されます。
- **GlobalDbProgressLag** — ユーザートランザクションとシステムトランザクションのどちらでも、セカンダリクラスターはプライマリクラスターより数ミリ秒遅れます。

メトリクス	ディメンション	で発行	単位
GlobalDbDataTransferBytes	SourceRegion、DBClusterIdentifier	Secondary	バイト
GlobalDbReplicatedWriteIO	SourceRegion、DBClusterIdentifier	Secondary	Count (カウント)
GlobalDbProgressLag	DBClusterIdentifier、SecondaryRegion : プライマリで DBClusterIdentifier、SourceRegion : セカンダリで	プライマリ、セカンダリ	ミリ秒

Amazon Neptune の機能の概要

このセクションでは、以下を含む Neptune 機能の概要を示します。

- Neptune のクエリ言語標準への準拠。
- Neptune のグラフデータモデル。
- Neptune トランザクションセマンティクスの説明。
- Neptune クラスターとインスタンスの概要。
- Neptune のストレージ、信頼性、可用性。
- Neptune のエンドポイントの説明。
- Neptune のカスタムクエリ ID でクエリのステータスを確認する方法。
- Neptune ラボモードの使用で実験的な機能を有効にするには。
- Neptune の DFE エンジンの説明。
- Neptune の JDBC 接続。
- Neptune エンジンのリリースとエンジンの更新方法のリスト。

> **Note**
>
> このセクションでは、Neptune グラフのデータにアクセスするために使用できるクエリ言語の使用については説明しません。
> Gremlin を使用して実行中の Neptune DB クラスターに接続する方法については、Gremlin を使用した Neptune グラフへのアクセス を参照してください。
> openCypher を使用して実行中の Neptune DB クラスターに接続する方法については、openCypher でNeptune グラフにアクセスする を参照してください。
> SPARQL を使用して実行中の Neptune DB クラスターに接続する方法については、SPARQL を使用した Neptune グラフへのアクセス を参照してください。

トピック

- Amazon Neptune 標準への準拠に関する注意事項
- Neptune のグラフデータモデル。
- Neptune ルックアップキャッシュは読み取りクエリを高速化できます。
- Neptune でのトランザクションセマンティクス

- Amazon Neptune DB クラスターとインスタンス
- Amazon Neptune ストレージ、信頼性、可用性
- Amazon Neptune エンドポイントに接続する
- Neptune Gremlin または SPARQL クエリにカスタム ID を挿入する
- Neptune ラボモード
- Amazon Neptune 代替クエリエンジン (DFE)
- Neptune DFE が使用する統計情報の管理
- グラフに関する簡単なサマリーレポートを取得する
- Amazon Neptune JDBC 接続
- Amazon Neptune エンジンの更新

Amazon Neptune 標準への準拠に関する注意事項

Amazon Neptune は、ほとんどの場合、Gremlin および SPARQL グラフクエリ言語を実装する際に該当する標準に準拠します。

以下のセクションでは標準について説明します。さらに Neptune が標準を超えたり逸脱したりする領域についても説明します。

トピック

- Amazon Neptune の Gremlin 標準への準拠
- Amazon Neptune の SPARQL 標準準拠
- Amazon Neptune での openCypher 仕様コンプライアンス Amazon Neptune

Amazon Neptune の Gremlin 標準への準拠

以下のセクションでは、Gremlin の Neptune 実装の概要と、Apache TinkerPop 実装との違いについて説明します。

Neptune は、いくつかの Gremlin ステップをエンジンにネイティブに実装し、Apache TinkerPop Gremlin 実装を使用して他のステップを処理します (「」を参照Amazon Neptune でのネイティブ Gremlin ステップサポート)。

> **Note**
>
> Gremlin コンソールと Amazon Neptune における実装の違いの具体的な例については、クイックスタートの the section called “Gremlin を使用する” セクションを参照してください。

トピック

- Gremlin に適用される標準
- スクリプト内の変数とパラメータ
- TinkerPop 列挙
- Java コード
- 要素のプロパティ
- スクリプトの実行
- セッション

- トランザクション
- 頂点および エッジ ID
- ユーザーによって指定された ID
- 頂点プロパティ ID
- 頂点プロパティのカーディナリティ
- 頂点プロパティの更新
- ラベル
- エスケープ文字
- Groovy の制限
- シリアル化
- Lambda ステップ
- サポートされていない Gremlin メソッド
- サポートされていない Gremlin ステップ
- Neptune での Gremlin グラフ機能

Gremlin に適用される標準

- Gremlin 言語は、正式な仕様ではなく、Apache TinkerPop ドキュメントと Gremlin の Apache TinkerPop 実装によって定義されます。
- 数値の形式については、Gremlin は IEEE 754 標準に従っています (IEEE 754-2019 - IEEE Standard for Floating-Point Arithmetic/Wikipedia IEEE 754 ページも参照してください)。

スクリプト内の変数とパラメータ

事前にバインドされた変数に関しては、トラバーサルオブジェクト g はNeptune では事前にバインドされており、`graph` オブジェクトはサポートされていません。

Neptune は Gremlin 変数やスクリプト内のパラメータ化をサポートしていませんが、インターネット上では、次のような変数宣言を含む Gremlin Server 用のサンプルスクリプトをよく目にすることがあります。

```
String query = "x = 1; g.V(x)";
List<Result> results = client.submit(query).all().get();
```

また、クエリの送信時にパラメータ化 (またはバインディング) を利用する例も多数あります。例えば、次のようになります。

```
Map<String,Object> params = new HashMap<>();
params.put("x",1);
String query = "g.V(x)";
List<Result> results = client.submit(query).all().get();
```

パラメータの例は通常、可能であればパラメータ化しないことによるパフォーマンス上のペナルティに関する警告と関連付けられます。このような例は多数ある TinkerPop ため、パラメータ化の必要性について非常に説得力があるように聞こえます。

ただし、変数宣言機能とパラメータ化機能 (警告とともに) はどちらも、 を使用している TinkerPopGremlin Server にのみ適用されます`GremlinGroovyScriptEngine`。Gremlin Server が Gremlin の `gremlin-language` ANTLR 文法を使用してクエリを解析する場合には適用されません。ANTLR 文法は変数宣言もパラメータ化もサポートしていないため、ANTLR を使用するときにはパラメータ化の失敗を心配する必要はありません。ANTLR 文法は の新しいコンポーネントであるため TinkerPop、インターネットで遭遇する可能性のある古いコンテンツは、通常、この区別を反映していません。

Neptune は、クエリ処理エンジンでは `GremlinGroovyScriptEngine` ではなく ANTLR 文法を使用するため、変数、パラメータ化、または `bindings` プロパティをサポートしません。その結果、パラメータ化の失敗に関連する問題は、Neptune には当てはまりません。Neptune を使用すると、通常はパラメータ化するようなクエリをそのまま送信しても完全に安全です。そのため、前の例を次のように簡略化しても、パフォーマンスを低下させずに済みます。

```
String query = "g.V(1)";
List<Result> results = client.submit(query).all().get();
```

TinkerPop 列挙

Neptune では、列挙値の完全修飾クラス名はサポートしていません。たとえば、Groovy リクエストでは `org.apache.tinkerpop.gremlin.structure.VertexProperty.Cardinality.single` ではなく `single` を使用する必要があります。

列挙型は、パラメータタイプによって決定されます。

次の表は、許可される列挙値と関連する TinkerPop 完全修飾名を示しています。

許可された値	Class
`id`, `key`, `label`, `value`	org.apache.tinkerpop.gremlin.structure.T
`T.id`, `T.key`, `T.label`, `T.value`	org.apache.tinkerpop.gremlin.structure.T
`set`, `single`	org.apache.tinkerpop.gremlin.structureVertexProperty。カーディナリティ
`asc`, `desc`, `shuffle`	org.apache.tinkerpop.gremlin.process.traversal.Order
`Order.asc` , `Order.desc` , `Order.shuffle`	org.apache.tinkerpop.gremlin.process.traversal.Order
`global`, `local`	org.apache.tinkerpop.gremlin.process.traversal.Scope
`Scope.global` , `Scope.local`	org.apache.tinkerpop.gremlin.process.traversal.Scope
`all`, `first`, `last`, `mixed`	org.apache.tinkerpop.gremlin.process.traversal.Pop
`normSack`	org.apache.tinkerpop.gremlin.process.traversalSackFunctions。バリエ
`addAll`, `and`, `assign`, `div`, `max`, `min`, `minus`, `mult`, `or`, `sum`, `sumLong`	org.apache.tinkerpop.gremlin.process.traversal.Operator
`keys`, `values`	org.apache.tinkerpop.gremlin.structure.Column
`BOTH`, `IN`, `OUT`	org.apache.tinkerpop.gremlin.structure.Direction
`any`, `none`	org.apache.tinkerpop.gremlin.process.traversal.step.TraversalOptionParent.Pick

Java コード

Neptune は、サポートされている Gremlin API 以外の、任意の Java または Java ライブラリ呼び出しにより定義されるメソッドへの呼び出しをサポートしていません。たとえば、`java.lang.*`、`Date()`、および `g.V().tryNext().orElseGet()` は許可されていません。

要素のプロパティ

Neptune は、 要素のプロパティを返すために TinkerPop 3.7.0 で導入された `materializeProperties`フラグをサポートしていません。その結果、Neptune は頂点またはエッジのみを、その `id`と `label`のみを含むリファレンスとして返します。

スクリプトの実行

すべてのクエリは、トラバーサルオブジェクト `g` で始まる必要があります。

文字列クエリの送信で、複数のトラバーサルは、セミコロン (`;`) または改行文字 (\n) で区切って発行することができます。実行されるように、最終以外の各ステートメントは、`.iterate()` ステップで終わる必要があります。最終的なトラバーサルデータのみが返されます。これは GLV ByteCode クエリの送信には適用されないことに注意してください。

セッション

Neptune のセッションは、長さが 10 分のみに制限されます。詳細については、Gremlin スクリプトベースのセッション「」およびTinkerPop「セッションリファレンス」を参照してください。

トランザクション

Neptune は、各 Gremlin トラバーサルの開始時に新しいトランザクションを開き、トラバーサルが正常に完了したときにトランザクションを閉じます。エラーが発生すると、トランザクションはロールバックされます。

セミコロン (`;`) または改行文字 (\n) で区切られた複数のステートメントは、単一のトランザクションに含まれています。最後のもの以外の各ステートメントは、`next()` ステップの実行で終わる必要があります。最終的なトラバーサルデータのみが返されます。

`tx.commit()` および `tx.rollback()` を使用した手動トランザクションロジックはサポートされていません。

> **Important**
>
> これは Gremlin クエリをテキスト文字列として送信するメソッドのみに当てはまります (「Gremlin トランザクション」を参照)。

頂点およびエッジ ID

Neptune Gremlin 頂点およびエッジ ID は、タイプ `String` である必要があります。これらの ID 文字列は Unicode 文字をサポートし、サイズは 55 MB を超えることはできません。

ユーザーが指定した ID はサポートされますが、通常の使用ではオプションとなります。頂点やエッジを追加するときに ID を指定しなかった場合、Neptune は UUID を生成し、次のような形式の文字列に変換します。これらの UUID は RFC 標準に準拠していないため、標準 UUID が必要な場合は、外部で生成し、頂点やエッジを追加するときに指定する必要があります。

> **Note**
>
> Neptune Load コマンドでは、~id フィールドを使用して、Neptune CSV 形式で ID を指定する必要があります。

ユーザーによって指定された ID

ユーザーにより提供される ID は、以下の規定により Neptune Gremlin で許可されます。

- 指定 ID はオプションです。
- 頂点とエッジのみがサポートされています。
- タイプ `String` のみがサポートされます。

カスタム ID で新しい頂点を作成するには、`id` キーワードで `property` ステップを使用します。`g.addV().property(id, 'customid')`。

> **Note**
>
> `id` キーワードを引用符で囲むことはできません。`T.id` を指します。

すべての頂点 ID およびすべてのエッジ ID は、一意である必要があります。ただし、Neptune では、頂点とエッジで同じ ID を持つことができます。

`g.addV()` を使用して新しい頂点を作成しようとする場合、すでにその ID を持つ頂点が存在すると、オペレーションは失敗します。この例外として、頂点に新しいラベルを指定するとオペレーションは成功しますが、新しいラベルおよび既存の頂点に指定されたすべての追加のプロパティが追加されます。Nothing は上書きされます。新しい頂点は作成されません。頂点 ID は変更せず、一意のままになります。

たとえば、次の Gremlin コンソールコマンドは成功します。

```
gremlin> g.addV('label1').property(id, 'customid')
gremlin> g.addV('label2').property(id, 'customid')
gremlin> g.V('customid').label()
==>label1::label2
```

頂点プロパティ ID

頂点プロパティ ID は自動的に生成され、クエリを実行したときに、正または負の数値で表示されます。

頂点プロパティのカーディナリティ

Neptune は、セット濃度と単一濃度をサポートしています。指定されていない場合は、セット濃度が選択されます。つまり、プロパティ値を設定した場合、値のセットにすでに表示されていない場合にのみ、プロパティに新しい値が追加されます。これは、セットの Gremlin 列挙の値です。

`List` はサポートされていません。プロパティカーディナリティの詳細については、Gremlin の「頂点」トピックを参照してください JavaDoc。

頂点プロパティの更新

値のセットに追加の値を追加せずにプロパティ値を更新するには、`property` ステップで `single` 濃度を指定します。

```
g.V('exampleid01').property(single, 'age', 25)
```

これにより、既存のプロパティの値はすべて削除されます。

ラベル

Neptune は、頂点の複数のラベルをサポートしています。ラベルを作成する際、`::` で区切ることで複数のラベルを指定できます。たとえば、`g.addV("Label1::Label2::Label3")` は頂点に 3 つの異なるラベルを追加します。`hasLabel` ステップでは、この頂点を `hasLabel("Label1")`、`hasLabel("Label2")`、および `hasLabel("Label3")` の 3 つのラベルのいずれかと一致させます。

> **Important**
>
> `::` 区切り記号は、この使用のみに予約されます。`hasLabel` ステップで複数のラベルを指定することはできません。たとえば、`hasLabel("Label1::Label2")` はいずれにも一致しません。

エスケープ文字

Neptune は、Apache Groovy 言語ドキュメントの 特殊文字のエスケープ セクションで説明されているすべての[エスケープ文字]を解決します。

Groovy の制限

Neptune は、g 以外で始まる Groovy コマンドをサポートしていません。これには、算術 (1+1 など)、システム呼び出し (`System.nanoTime()` など)、および変数の定義 (1+1 など) が含まれます。

> **Important**
>
> Neptune では、完全修飾クラス名はサポートしていません。たとえば、Groovy リクエストでは `org.apache.tinkerpop.gremlin.structure.VertexProperty.Cardinality.single` ではなく `single` を使用する必要があります。

シリアル化

Neptune は、リクエストされた MIME タイプに基づいて、以下のシリアル化をサポートしています。

MIME タイプ	シリアル化	構成
application/vnd.gremlin-v1.0+gryo	GryoMessageSerializerV1	ioRegistries: [org.apache.tinkerpop.gremlin.tinkergraph.structure.TinkerIoRegistryV1]
application/vnd.gremlin-v1.0+gryo-stringd	GryoMessageSerializerV1	serializeResultToString: true}
application/vnd.gremlin-v3.0+gryo	GryoMessageSerializerV3	ioRegistries: [org.apache.tinkerpop.gremlin.tinkergraph.structure.TinkerIoRegistryV3]
application/vnd.gremlin-v3.0+gryo-stringd	GryoMessageSerializerV3	serializeResultToString: true
application/vnd.gremlin-v1.0+json	GraphSONMessageSerializerGremlinV1	ioRegistries: [org.apache.tinkerpop.gremlin.tinkergraph.structure.TinkerIoRegistryV1]
application/vnd.gremlin-v2.0+json	GraphSONMessageSerializerV2 （でのみ動作 WebSockets ）	ioRegistries: [org.apache.tinkerpop.gremlin.tinkergraph.structure.TinkerIoRegistryV2]
application/vnd.gremlin-v3.0+json	GraphSONMessageSerializerV3	

application/json	GraphSONMessageSerializerV3	ioRegistries: [org.apache.tinkerpop.gremlin.tinkergraph.structure.TinkerIoRegistryV3]
application/vnd.graphbinary-v1.0	GraphBinaryMessageSerializerV1	

Neptune はこれらのさまざまなシリアライザータイプをサポートしていますが、その使用に関するガイダンスはかなり簡単です。HTTP 経由で Neptune に接続する場合は、GraphSON 3 の代替バージョンの埋め込みタイプ`application/vnd.gremlin-v3.0+json;types=false`としてを使用することを優先し、操作を複雑にします。Apache TinkerPop ドライバーを使用している場合は、デフォルトの を使用するため、選択を行う必要がない可能性がありますapplication/vnd.graphbinary-v1.0。レガシーの理由により、残りの形式は残ります。

> **Note**
>
> ここに示すシリアライザテーブルは、 TinkerPop 3.7.0 以降の命名を参照しています。この変更の詳細については、TinkerPop アップグレードドキュメント を参照してください。Gryo シリアル化のサポートは 3.4.3 では非推奨となり、3.6.0 では正式に削除されました。Gryo または をデフォルトで使用しているドライバーバージョンで明示的に使用している場合は、ドライバーに切り替える GraphBinary か、アップグレードする必要があります。

Lambda ステップ

Neptune では、Lambda ステップはサポートされていません。

サポートされていない Gremlin メソッド

Neptune は、以下の Gremlin メソッドをサポートしていません。

- `org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.GraphTraversal.progra`
- `org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.GraphTraversal.sideEf`
- `org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.GraphTraversal.from(o`
- `org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.GraphTraversal.to(org`

たとえば、以下のトラバーサルは許可されません。`g.V().addE('something').from(__.V().next()).to(__.V().next())`。

> **Important**
>
> これは Gremlin クエリをテキスト文字列として送信するメソッドのみに当てはまります。

サポートされていない Gremlin ステップ

Neptune は、以下の Gremlin ステップをサポートしていません。

- Gremlin io( ) ステップは、Neptune では部分的にしかサポートされていません。`g.io(`*`url`*`).read()` のように読み取りコンテキストでは使用できますが、書き込みには使用できません。

Neptune での Gremlin グラフ機能

Gremlin の Neptune 実装には、`graph` オブジェクトは表示されません。次の表は、Gremlin の機能と、Neptune がそれらをサポートしているかどうかを示しています。

Neptune の **`graph`** 機能のサポート

Neptune グラフ機能は、サポートされている場合、`graph.features()` コマンドによって返されるものと同じです。

グラフ機能	有効?
`Transactions`	true
`ThreadedTransactions`	false
`Computer`	false
`Persistence`	true
`ConcurrentAccess`	true

Neptune の変数機能のサポート

変数機能	有効?
Variables	false
SerializableValues	false
UniformListValues	false
BooleanArrayValues	false
DoubleArrayValues	false
IntegerArrayValues	false
StringArrayValues	false
BooleanValues	false
ByteValues	false
DoubleValues	false
FloatValues	false
IntegerValues	false
LongValues	false
MapValues	false
MixedListValues	false
StringValues	false
ByteArrayValues	false
FloatArrayValues	false
LongArrayValues	false

Neptune の頂点機能のサポート

頂点機能	有効?
MetaProperties	false
DuplicateMultiProperties	false
AddVertices	true
RemoveVertices	true
MultiProperties	true
UserSuppliedIds	true
AddProperty	true
RemoveProperty	true
NumericIds	false
StringIds	true
UuidIds	false
CustomIds	false
AnyIds	false

Neptune の頂点プロパティ機能のサポート

頂点プロパティ機能	有効?
UserSuppliedIds	false
AddProperty	true
RemoveProperty	true
NumericIds	true

StringIds	true
UuidIds	false
CustomIds	false
AnyIds	false
Properties	true
SerializableValues	false
UniformListValues	false
BooleanArrayValues	false
DoubleArrayValues	false
IntegerArrayValues	false
StringArrayValues	false
BooleanValues	true
ByteValues	true
DoubleValues	true
FloatValues	true
IntegerValues	true
LongValues	true
MapValues	false
MixedListValues	false
StringValues	true
ByteArrayValues	false
FloatArrayValues	false

LongArrayValues	false

Neptune のエッジ機能のサポート

エッジ機能	有効?
AddEdges	true
RemoveEdges	true
UserSuppliedIds	true
AddProperty	true
RemoveProperty	true
NumericIds	false
StringIds	true
UuidIds	false
CustomIds	false
AnyIds	false

Neptune のエッジプロパティ機能のサポート

エッジプロパティ機能	有効?
Properties	true
SerializableValues	false
UniformListValues	false
BooleanArrayValues	false
DoubleArrayValues	false

IntegerArrayValues	false
StringArrayValues	false
BooleanValues	true
ByteValues	true
DoubleValues	true
FloatValues	true
IntegerValues	true
LongValues	true
MapValues	false
MixedListValues	false
StringValues	true
ByteArrayValues	false
FloatArrayValues	false
LongArrayValues	false

Amazon Neptune の SPARQL 標準準拠

適用可能な SPARQL 標準を列挙したあと、以下のセクションでは、Neptune の SPARQL 実装がそれらの標準からどのように拡張し、または異なるかについて具体的な詳細を説明します。

トピック

- SPARQL に適用される標準
- Neptune SPARQL のデフォルトの名前空間プレフィックス
- SPARQL デフォルトグラフと名前が付いたグラフ
- Neptune でサポートされている SPARQL XPath コンストラクタ関数
- クエリと更新のためのデフォルトベース IRI

- Neptune での xsd:dateTime の値
- Neptune による特殊な浮動小数点値の処理
- Neptune での任意長の値の制限
- Neptune で拡張される SPARQL の不等比較
- Neptune SPARQL での範囲外リテラルの処理

Amazon Neptune は、SPARQL グラフクエリ言語を実装する際に次の標準に準拠しています。

SPARQL に適用される標準

- SPARQL は、2013 年 3 月 21 日の W3C SPARQL 1.1 クエリ言語勧告に基づいて定義されています。
- SPARQL 更新プロトコルとクエリ言語は、W3C SPARQL 1.1 更新仕様で定義されています。
- 数値形式の場合、SPARQL は W3C XML スキーマ定義言語 (XSD) 1.1 パート 2: データ型仕様に従っています。この仕様は IEEE 754 仕様を準拠しています (IEEE 754-2019 - IEEE Standard for Floating-Point Arithmetic/Wikipedia IEEE 754 ページも参照してください)。ただし、IEEE 754-1985 バージョン以降に導入された機能は、仕様には含まれません。

Neptune SPARQL のデフォルトの名前空間プレフィックス

Neptune は、SPARQL クエリで使用するためにデフォルトで以下のプレフィックスを定義します。詳細については、SPARQL 仕様の「接頭辞付き名」を参照してください。

- `rdf – http://www.w3.org/1999/02/22-rdf-syntax-ns#`
- `rdfs – http://www.w3.org/2000/01/rdf-schema#`
- `owl – http://www.w3.org/2002/07/owl#`
- `xsd – http://www.w3.org/2001/XMLSchema#`

SPARQL デフォルトグラフと名前が付いたグラフ

Amazon Neptune はすべてのトリプルを名前が付いたグラフに関連付けます。デフォルトグラフはすべての名前が付いたグラフの総合として定義されます。

クエリ用のデフォルトグラフ

GRAPH キーワードや構成 (FROM NAMED など) を使用してグラフを明示的に指定せずに SPARQL クエリを送信すると、Neptune は常に DB インスタンスのすべてのトリプルを考慮します。たとえば、次のクエリはすべてのトリプルを Neptune SPARQL エンドポイントから返します。

```
SELECT * WHERE { ?s ?p ?o }
```

1 つ以上のグラフに表されるトリプルは、1 度だけ返されます。

デフォルトグラフ仕様についての詳細は、SPARQL 1.1 クエリ言語仕様の「RDF データセット」セクションを参照してください。

ロード、挿入や更新用に名前付きのグラフを指定する

トリプルのロード、挿入あるいは更新時に名前が付いたグラフを指定しない場合、Neptune は URI `http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph` が定義するフォールバックの名前付きグラフを使用します。

Neptune がトリプルベースの形式を使用して Load リクエストを発行する場合、`parserConfiguration: namedGraphUri` パラメータを使用して、すべてのトリプルを使用するように名前付きのグラフを指定できます。Load コマンド構文の詳細については、「the section called “ローダーコマンド”」を参照してください。

> **Important**
>
> このパラメータを使用せず、名前付きのグラフを指定しない場合、フォールバック URI が使用されます。`http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph`。

このフォールバックの名前付きグラフは、名前付きグラフターゲットを明示的に提供しないで、SPARQL UPDATE を介してトリプルをロードする場合にも使用されます。

クワッドベース形式の N-Quads を使用して、データベースの各トリプルに名前付きグラフを指定できます。

> **Note**
>
> N-Quads では名前付きグラフを空にできます。この場合、`http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph` が使用されます。`namedGraphUri` パーサー設定オプションを使用して、N-Quads のデフォルトの名前付きグラフを上書きできます。

Neptune でサポートされている SPARQL XPath コンストラクタ関数

SPARQL 標準により、SPARQL エンジンは一連の拡張可能な XPath コンストラクタ関数をサポートできます。Neptune は現在、以下のコンストラクター関数をサポートしています。ここで xsd プレフィックスは `http://www.w3.org/2001/XMLSchema#` と定義されます。

- `xsd:boolean`
- `xsd:integer`
- `xsd:double`
- `xsd:float`
- `xsd:decimal`
- `xsd:long`
- `xsd:unsignedLong`

クエリと更新のためのデフォルトベース IRI

Neptune クラスターには複数の異なるエンドポイントがあるため、クエリまたは更新のリクエスト URL をベース IRI として使用すると、相対 IRI を解決する際に予期しない結果が生じる可能性があります。

エンジンリリース 1.2.1.0 では、明示的なベース IRI がリクエストに含まれていない場合、Neptune は `http://aws.amazon.com/neptune/default/` をベース IRI として使用します。

次のリクエストでは、ベース IRI はリクエストの一部です。

```
BASE <http://example.org/default/>
INSERT DATA { <node1> <id> "n1" }

BASE <http://example.org/default/>
SELECT * { <node1> ?p ?o }
```

結果は次のようになります。

```
?p                                                  ?o
http://example.org/default/id                       n1
```

ただし、このリクエストにはベース IRI は含まれていません。

```
INSERT DATA { <node1> <id> "n1" }

SELECT * { <node1> ?p ?o }
```

その場合、結果は以下のようになります。

```
?p                                                  ?o
http://aws.amazon.com/neptune/default/id            n1
```

Neptune での xsd:dateTime の値

パフォーマンス上の理由から、Neptune は 常に協定世界時 (UTC) として日付/時刻値を保存します。これにより、直接比較が非常に効率的になります。

つまり、特定のタイムゾーンを指定する `dateTime` 値を入力すると、Neptune は値を UTC に変換し、そのタイムゾーン情報を破棄します。その後、後で `dateTime` 値を取得すると、元のタイムゾーンの時間ではなく UTC で表され、元のタイムゾーンが何であったかを知ることができなくなります。

Neptune による特殊な浮動小数点値の処理

Neptune は、SPARQL の特殊な浮動小数点値を次のように処理します。

SPARQL NaN Neptune での処理

Neptune では、SPARQL はクエリで値として `NaN` を使用できます。シグナルおよびクワイエットの `NaN` 値は区別されません。Neptune は、すべての `NaN` 値をクワイエットとして扱います。

`NaN` より大きい、より小さい、または等しいものは何もないため、`NaN` の比較はできません。つまり、比較の一方の `NaN` の値は、比較の他方のいずれの値とも一致しません。

しかし、XSD仕様は 2 つの`xsd:double`または`xsd:float` `NaN`等しいものとして扱います。Neptune は、`IN` フィルター、フィルター式の等号演算子、完全一致セマンティクス (トリプルパターンのオブジェクト位置の `NaN` にある) についてはこれに従います。

Neptune での SPARQL 無限値の処理

Neptune では、SPARQL はクエリで `INF` または`-INF` の値を使用できます。`INF` は他のどの数値よりも大きい値、`-INF` は他のどの数値よりも小さい値と見なされます。

符号が一致する 2 つの INF 値は、型に関係なく互いに等しいと比較されます (たとえば、float -INF は double -INF と等しいと比較されます)。

NaN より大きい、より小さい、等しいものは何もないため、当然ながら NaN の比較はできません。

Neptune での SPARQL 負のゼロ処理

Neptune は、負のゼロ値を符号なしゼロに正規化します。負のゼロ値はクエリで使用できますが、データベースにはそのように記録されず、符号なしゼロと等しいと見なされます。

Neptune での任意長の値の制限

Neptune は、SPARQL の XSD 整数、浮動小数点、10 進値のストレージサイズを 64 ビットに制限します。より大きい値を使用すると、InvalidNumericDataException エラーが発生します。

Neptune で拡張される SPARQL の不等比較

SPARQL 標準は、値式の 3 項ロジックを定義します。値式は、true、false、または error のいずれかに評価されます。SPARQL 1.1 仕様) で定義される項等価のデフォルトのセマンティクスは、FILTER 条件で = および != の比較に当てはまり、で仕様で演算子テーブルにおいて明示的に比較できないデータ型を比較する場合、error を生成します。

この動作は、次の例のように、直感的でない結果につながる可能性があります。

データ:

```
<http://example.com/Server/1> <http://example.com/ip> "127.0.0.1"^^<http://example.com/
datatype/IPAddress>
```

クエリ 1:

```
SELECT * WHERE {
    <http://example.com/Server/1> <http://example.com/ip> ?o .
    FILTER(?o = "127.0.0.2"^^<http://example.com/datatype/IPAddress>)
}
```

クエリ 2:

```
SELECT * WHERE {
    <http://example.com/Server/1> <http://example.com/ip> ?o .
```

```
    FILTER(?o != "127.0.0.2"^^<http://example.com/datatype/IPAddress>)
}
```

Neptune でリリース 1.0.2.1 より前に使用していたデフォルトの SPARQL セマンティクスでは、どちらのクエリからも空の結果が返されます。その理由として、`?o = "127.0.0.2"^^<http://example.com/IPAddress>` は、`?o := "127.0.0.1"^^<http://example.com/IPAddress>` に評価された場合に、`false` ではなく `error` を生成するためです。これは、カスタムデータ型 `<http://example.com/IPAddress>` に対して明示的に指定された比較ルールが存在しないことが原因です。その結果、2 番目のクエリの否定バージョンも `error` を生成します。どちらのクエリでも、`error` により、候補ソリューションが除外されます。

リリース 1.0.2.1 以降では、Neptune は仕様どおりに SPARQL の非等値演算子を拡張しています。演算子の拡張性に関する SPARQL 1.1 セクションを参照してください。この拡張により、エンジンは、ユーザー定義や比較不可能な組み込みデータ型を比較する方法に関する追加のルールを定義できます。

このオプションを利用することで、Neptune は、演算子マッピングテーブルで明示的に定義されていない任意の 2 つのデータ型を比較した場合に、リテラル値とデータ型が構文的に等しいときは `true` として、そうでないときは false として評価するようになります。いずれの場合も、`error` は生成されません。

これらの新しいセマンティクスを使用すると、上の 2 番目のクエリからは空の結果の代わりに `"127.0.0.1"^^<http://example.com/IPAddress>` が返されます。

Neptune SPARQL での範囲外リテラルの処理

XSD セマンティクスは、`integer` および `decimal` を除く各数値型を値空間で定義します。これらの定義は、各型を値の範囲に制限します。たとえば、`xsd:byte` 範囲の範囲は -128 から +127 までです。この範囲外の値は無効とみなされます。

タイプの値空間の外部でリテラル値を割り当てようとした場合 (例えば、 `xsd:byte`をリテラル値 999 に設定しようとする場合) 、Neptune は out-of-range 値を丸めたり切り捨てたりすることなくそのまま受け入れます。しかし、与えられた型はそれを表すことができないので、数値として持続しません。

つまり、定義された `xsd:byte` 値の範囲外の値であっても Neptune は `"999"^^xsd:byte` を受け入れます。ただし、値がデータベースで永続化されると、3 重パターンのオブジェクト位置で、完全一致セマンティクスでのみ使用できます。 out-of-range リテラルは数値として扱われないため、範囲フィルターは実行できません。

SPARQL 1.1 仕様では、numeric-operatornumeric、string-operatorstring、literal-operatorliteralなどの範囲演算子フォームに定義します。Neptune はinvalid-literal-operator-numeric-valueなどの範囲比較演算子を実行できません。

Amazon Neptune での openCypher 仕様コンプライアンス Amazon Neptune

openCypher の Amazon Neptune リリースは、一般に、現在の openCypher 仕様 (「Cypher Query Language Reference Version 9」) で定義されている句、演算子、式、関数、および構文をサポートしています。openCypher の Neptune サポートの制限と相違点を以下に説明します。

> **Note**
>
> 現在の Neo4j の Cypher 実装には、上記の openCypher 仕様には含まれていない機能が含まれています。現在の Cypher コードを Neptune に移行する場合、詳細については,「Neptune の Neo4j との互換性」と「Neptune 上の openCypherで実行するように Cypher クエリを書き直す」を参照してください。

Neptune における openCypher 句のサポート

Neptune は、特に注記がなければ、以下の句をサポートしています。

- MATCH — サポートされています。ただし、*shortestPath()* と *allShortestPaths()* は、現在はサポートされていません。
- OPTIONAL MATCH
- *MANDATORY MATCH* — Neptune では現在サポートされていません。ただし、Neptune は MATCH クエリでの カスタム ID 値をサポートしています。
- RETURN — サポートされています。ただし、SKIP または LIMIT として非静的な値が使用される場合を除きます。例えば、以下は現時点では機能しません。

```
MATCH (n)
RETURN n LIMIT toInteger(rand())    // Does NOT work!
```

- WITH — サポートされています。ただし、SKIP または LIMIT として非静的な値が使用される場合を除きます。例えば、以下は現時点では機能しません。

```
MATCH (n)
```

```
WITH n SKIP toInteger(rand())
WITH count() AS count
RETURN count > 0 AS nonEmpty    // Does NOT work!
```

- UNWIND
- WHERE
- ORDER BY
- SKIP
- LIMIT
- CREATE — Neptune では、CREATE クエリでカスタム ID 値を作成できます。
- DELETE
- SET
- REMOVE
- MERGE - Neptune は MERGE クエリでのカスタム ID 値をサポートしています。
- *CALL[YIELD...]* — Neptune では現在サポートされていません。
- UNION, UNION ALL — 読み取り専用クエリはサポートされていますが、ミューテーションクエリは現在サポートされていません。

Neptune における openCypher 演算子のサポート

Neptune は、特に注記がなければ、以下の演算子をサポートしています。

一般的な演算子

- DISTINCT
- ネストされたリテラルマップのプロパティにアクセスするための **.** 演算子。

算術演算子

- **+** 加算演算子。
- **-** 減算演算子。
- ***** 乗算演算子。
- **/** 除算演算子。
- **%** 剰余除算演算子。

- **^** 指数演算子は#######です。

比較演算子

- **=** 加算演算子。
- **<>** 不等式演算子。
- **<** 小なり演算子は、引数のいずれかが Path、List、または Map の場合を除いてサポートされます。
- **>** 大なり演算子は、引数のいずれかが Path、List、または Map の場合を除いてサポートされます。
- **<=** less-than-or-equal-to 演算子は、引数がパス、リスト、マップのいずれかである場合を除き、サポートされます。
- **>=** greater-than-or-equal-to 演算子は、引数がパス、リスト、マップのいずれかである場合を除き、サポートされます。
- `IS NULL`
- `IS NOT NULL`
- `STARTS WITH` は、検索するデータが文字列の場合にサポートされます。
- `ENDS WITH` は、検索するデータが文字列の場合にサポートされます。
- `CONTAINS` は、検索するデータが文字列の場合にサポートされます。

ブール演算子

- `AND`
- `OR`
- `XOR`
- `NOT`

文字列演算子

- **+** 連結演算子。

演算子一覧

- **+** 連結演算子。

- IN (リスト内の項目の存在を確認します)

Neptune での openCypher 式のサポート

Neptune は、特に注記がなければ、以下の式をサポートしています。

- CASE
- ノード、リレーションシップ、またはマップ内で動的に計算されたプロパティキーにアクセスするための **[]** 式は、現在、Neptune ではサポートされていません。例えば、以下は機能しません。

```
MATCH (n)
WITH [5, n, {key: 'value'}] AS list
RETURN list[1].name
```

Neptune における openCypher 関数のサポート

Neptune は、特に注記がなければ、以下の関数をサポートしています。

述語関数

- exists()

スカラー関数

- coalesce()
- endNode()
- epochmillis()
- head()
- id()
- last()
- length()
- randomUUID()
- properties()
- removeKeyFromMap

- size() — このオーバーロードされたメソッドは、現在のところ、パターン表現、リスト、文字列に対してのみ機能します。
- startNode()
- timestamp()
- toBoolean()
- toFloat()
- toInteger()
- type()

集計関数

- avg()
- collect()
- count()
- max()
- min()
- percentileDisc()
- stDev()
- percentileCont()
- stDevP()
- sum()

関数の一覧表示

- join() (リスト内の文字列を 1 つの文字列に連結する)
- keys()
- labels()
- nodes()
- range()
- relationships()
- reverse()

- tail()

数学関数 — 数値

- abs()
- ceil()
- floor()
- rand()
- round()
- sign()

数学関数 — 対数

- e()
- exp()
- log()
- log10()
- sqrt()

数学関数 — 三角関数

- acos()
- asin()
- atan()
- atan2()
- cos()
- cot()
- degrees()
- pi()
- radians()
- sin()

- tan()

文字列関数

- join() (リスト内の文字列を 1 つの文字列に連結する)
- left()
- lTrim()
- replace()
- reverse()
- right()
- rTrim()
- split()
- substring()
- toLower()
- toString()
- toUpper()
- trim()

ユーザー定義関数

########は、Neptune では現在サポートされていません。

Neptune 固有の openCypher 実装の詳細

以下のセクションでは、openCypher の Neptune 実装が openCypher 仕様と異なる場合や、それを超える場合があることについて説明します。

Neptune における可変長パス (VLP) 評価

可変長パス (VLP) 評価は、グラフ内のノード間のパスを検出します。クエリではパスの長さに制限はありません。サイクルを防ぐため、openCypher 仕様は、各エッジはソリューションごとに最大 1 回トラバースする必要があると規定しています。

VLP の場合、Neptune 実装は、プロパティ等価フィルターの定数値のみをサポートするという点で openCypher 仕様とは異なります。次のようなクエリがあるとします。

```
MATCH (x)-[:route*1..2 {dist:33, code:x.name}]->(y) return x,y
```

x.name プロパティ等価フィルターの値は定数ではないため、このクエリの結果は UnsupportedOperationException になり、Property predicate over variable-length relationships with non-constant expression is not supported in this release. というメッセージが表示されます。

Neptune openCypher 実装での一時的なサポート (Neptune データベース 1.3.1.0 以下)

Neptune は現在、openCypher の時間関数を限定的にサポートしています。時間型の DateTime データ型をサポートしています。

datetime() 関数を使用して、以下のように現在の UTC の日付と時刻を取得できます。

```
RETURN  datetime() as res
```

日付と時刻の値は、次のように、Neptune に保存されているデータから変換できます。

```
MATCH (n) RETURN datetime(n.createdDate)
```

日付と時刻の値は、"dateTtime" 形式で文字列から解析できます。ここで、date と time は、どちらも以下のサポートされている形式のいずれかで表されます。

サポートされている日付形式

- yyyy-MM-dd
- yyyyMMdd
- yyyy-MM
- yyyy-DDD
- yyyyDDD
- yyyy

サポートされている時刻形式

- HH:mm:ssZ
- HHmmssZ

- HH:mm:ssZ
- HH:mmZ
- HHmmZ
- HHZ
- HHmmss
- HH:mm:ss
- HH:mm
- HHmm
- HH

例:

```
RETURN datetime('2022-01-01T00:01')      // or another example:
RETURN datetime('2022T0001')
```

Neptune openCypher のすべての日付/時刻値は UTC 値として保存および取得されることに注意してください。

Neptune openCypher は statement クロックを使用します。つまり、クエリの実行中、時間的に同じインスタントが使用されます。同じトランザクション内の別のクエリでは、時間的に異なるインスタントが使用される場合があります。

Neptune は datetime() の呼び出し内での関数の使用をサポートしていません。例えば、以下は機能しません。

```
CREATE (:n {date:datetime(tostring(2021))})  // ---> NOT ALLOWED!
```

Neptune は datetime を epochmillis に変換する epochmillis() 関数をサポートしています。例:

```
MATCH (n) RETURN epochMillis(n.someDateTime)
1698972364782
```

Neptune は現在、DateTime オブジェクトに対する他の関数や演算 (加算や減算など) をサポートしていません。

Neptune openCypher 実装 (Neptune Analytics および Neptune データベース 1.3.2.0 以降) での一時的なサポート

の次の日時機能は、Neptune Analytics OpenCypher に適用されます。または、 ラボモードパラメータを使用して、Neptune エンジンリリースバージョン 1.3.2.0 以降で次の日時機能DatetimeMillisecond=enabledを有効にすることもできます。ラボモードでこの機能を使用する方法の詳細については、「」を参照してください日時サポートの延長。

- ミリ秒のサポート。日時リテラルは、ミリ秒が 0 であっても、常にミリ秒単位で返されます。(以前の動作はミリ秒を切り捨てることでした)。

```
CREATE (:event {time: datetime('2024-04-01T23:59:59Z')})

# Returning the date returns with 000 suffixed representing milliseconds
MATCH(n:event)
RETURN n.time as datetime

{
  "results" : [ {
    "n" : {
      "~id" : "0fe88f7f-a9d9-470a-bbf2-fd6dd5bf1a7d",
      "~entityType" : "node",
      "~labels" : [ "event" ],
      "~properties" : {
        "time" : "2024-04-01T23:59:59.000Z"
      }
    }
  } ]
}
```

- 保存されたプロパティまたは中間結果に対する datetime() 関数の呼び出しのサポート。例えば、この機能より前には、次のクエリを実行できませんでした。

  プロパティの日時 ():

```
// Create node with property 'time' stored as string
CREATE (:event {time: '2024-04-01T23:59:59Z'})

// Match and return this property as datetime
MATCH(n:event)
RETURN datetime(n.time) as datetime
```

中間結果に対する日時 ():

```
// Parse datetime from parameter
UNWIND $list as myDate
RETURN datetime(myDate) as d
```

- 上記の場合、 で作成された日時のアクセス許可を保存できるようになりました。

  あるプロパティの文字列プロパティから別のプロパティへの日時の保存：

```
// Create node with property 'time' stored as string
CREATE (:event {time: '2024-04-01T23:59:59Z', name: 'crash'})

// Match and update the same property to datetime type
MATCH(n:event {name: 'crash'})
SET n.time = datetime(n.time)

// Match and update another node's property
MATCH(e:event {name: 'crash'})
MATCH(n:server {name: e.servername})
SET n.time = datetime(e.time)
```

  日時プロパティを使用してパラメータからノードをバッチ作成します。

```
// Batch create from parameter
UNWIND $list as events
CREATE (n:crash) {time: datetime(events.time)}
// Parameter value
{
  "x":[
    {"time":"2024-01-01T23:59:29", "name":"crash1"},
    {"time":"2023-01-01T00:00:00Z", "name":"crash2"}
  ]
}
```

- ISO8601 日時形式のより大きなサブセットのサポート。以下の「」を参照してください。

サポートされる形式

日時値の形式は [Date]T[Time][Timezone] で、T は区切り文字です。明示的なタイムゾーンが指定されていない場合、UTC (Z) がデフォルトと見なされます。

タイムゾーン

サポートされているタイムゾーン形式は次のとおりです。

- +/-HH:mm
- +/-HHmm
- +/-HH

日時文字列にタイムゾーンが存在するかどうかはオプションです。タイムゾーンオフセットが 0 の場合、上記のタイムゾーンプレフィックスの代わりに Z を使用して UTC 時間を指定できます。サポートされているタイムゾーンの範囲は -14:00 から +14:00 です。

日付

タイムゾーンが存在しない場合、またはタイムゾーンが UTC (Z) の場合、サポートされている日付形式は次のとおりです。

> **Note**
>
> DDD は序数の日付を指し、001 から 365 (うるう年の 366) までの年を表す。例えば、2024 年 002 日は 2024 年 1 月 2 日を表します。

- `yyyy-MM-dd`
- `yyyyMMdd`
- `yyyy-MM`
- `yyyyMM`
- `yyyy-DDD`
- `yyyyDDD`
- `yyyy`

Z 以外のタイムゾーンを選択した場合、サポートされている日付形式は次のように制限されます。

- `yyyy-MM-dd`
- `yyyy-DDD`

- yyyyDDD

サポートされている日付の範囲は 1400-01-01 から 9999-12-31 です。

時間

タイムゾーンが存在しない場合、またはタイムゾーンが UTC (Z) の場合、サポートされている時間形式は次のとおりです。

- HH:mm:ss.SSS
- HH:mm:ss
- HHmmss.SSS
- HHmmss
- HH:mm
- HHmm
- HH

Z 以外のタイムゾーンを選択した場合、サポートされている時間形式は次のように制限されます。

- HH:mm:ss
- HH:mm:ss.SSS

Neptune openCypher 言語セマンティクスの違い

Neptune は、ノードとリレーションシップ ID を整数ではなく文字列で表します。ID は、データローダーによって指定された ID と等しくなります。カラムに名前空間がある場合、名前空間に ID を加えたもの。よって、id 関数は、整数の代わりに文字列を返します。

INTEGER データ型は 64 ビットに制限されています。より大きな浮動小数点値または文字列値を整数に変換する場合TOINTEGER 関数、負の値は LLONG_MIN に切り捨てられ、正の値は LLONG_MAX に切り捨てられます。

例:

```
RETURN TOINTEGER(2^100)
>  9223372036854775807
```

```
RETURN TOINTEGER(-1 * 2^100)
>  -9223372036854775808
```

Neptune 固有の **join()** 関数

Neptune は openCypher 仕様にはない `join()` 関数を実装しています。文字列リテラルと文字列区切り文字のリストから文字列リテラルを作成します。2 つの引数を取ります。

- 1 番目の引数は文字列リテラルのリストです。
- 2 番目の引数は区切り文字列であり、0、1、または複数の文字で構成できます。

例：

```
join(["abc", "def", "ghi"], ", ")    // Returns "abc, def, ghi"
```

Neptune 固有の **removeKeyFromMap()** 関数

Neptune は openCypher 仕様にはない `removeKeyFromMap()` 関数を実装しています。指定されたキーをマップから削除し、生成された新しいマップを返します。

関数は 2 つの引数を取ります。

- 最初の引数は、キーを削除するマップです。
- 2 番目の引数は、マップから削除するキーです。

`removeKeyFromMap()` 関数は、マップのリストを巻き戻して、ノードやリレーションシップの値を設定したい場合に特に便利です。例:

```
UNWIND [{`~id`: 'id1', name: 'john'}, {`~id`: 'id2', name: 'jim'}] as val
CREATE (n {`~id`: val.`~id`})
SET n = removeKeyFromMap(val, '~id')
```

ノードとリレーションシッププロパティのカスタム ID 値

エンジンリリース 1.2.0.2 以降、Neptune は openCypher 仕様を拡張し、`CREATE`, `MERGE`, および `MATCH` 句でノードとリレーションシップの `id` 値を指定できるようになりました。これにより、システムによって生成された UUID の代わりに、ユーザーフレンドリーな文字列を割り当てて、ノードやリレーションシップを識別できるようになりました。

> **Warning**
>
> ~id は、現在、予約済みのプロパティ名と見なされているため、openCypher 仕様に対するこの拡張は下位互換性がありません。~id をデータやクエリで既にプロパティとして使用している場合、既存のプロパティを新しいプロパティキーに移行して、古いものを削除する必要があります。現在、~id をプロパティとして使用している場合の対処方法 を参照してください。

カスタム ID を持つノードとリレーションシップを作成する方法の例を次に示します。

```
CREATE (n {`~id`: 'fromNode', name: 'john'})
  -[:knows {`~id`: 'john-knows->jim', since: 2020}]
  ->(m {`~id`: 'toNode', name: 'jim'})
```

既に使用されているカスタム ID を作成しようとすると、Neptune は DuplicateDataException エラーをスローします。

MATCH 句でカスタム ID を使用する例を次に示します。

```
MATCH (n {`~id`: 'id1'})
RETURN n
```

MERGE 句でカスタム ID を使用する例を次に示します。

```
MATCH (n {name: 'john'}), (m {name: 'jim'})
MERGE (n)-[r {`~id`: 'john->jim'}]->(m)
RETURN r
```

現在、**~id** をプロパティとして使用している場合の対処方法

エンジンリリース 1.2.0.2 では、openCypher 句の ~id キーは、プロパティではなく id として扱われるようになりました。つまり、~id という名前のプロパティがある場合、アクセスできなくなるということです。

~id プロパティを使用している場合、エンジンリリース 1.2.0.2 以降にアップグレードする前にしなければならないことは、まず、既存の ~id プロパティを新しいプロパティキーに移行し、~id プロパティを削除することです。例えば、次のクエリは、

- すべてのノードに「newId」という名前の新しいプロパティを作成します。
- 「~id」プロパティの値を「newId」プロパティにコピーし、
- データから「~id」プロパティを削除します。

```
MATCH (n)
WHERE exists(n.`~id`)
SET n.newId = n.`~id`
REMOVE n.`~id`
```

データ内の、~id プロパティのあるリレーションシップについても同じことを行う必要があります。

また、~id プロパティを参照するクエリを使用している場合は、すべて変更する必要があります。例えば、次のクエリは、

```
MATCH (n)
WHERE n.`~id` = 'some-value'
RETURN n
```

次のように変わります。

```
MATCH (n)
WHERE n.newId = 'some-value'
RETURN n
```

Neptune openCypher と Cypher のその他の違い

- Neptune は Bolt プロトコルの TCP 接続のみをサポートしています。 WebSockets Bolt 用の 接続はサポートされていません。
- Neptune openCypher は、`trim()`、`ltrim()` および `rtrim()` 関数で Unicode で定義された空白を削除します。
- Neptune openCypher では `tostring(` ダブル `)` は、倍精度の高い値に対して E 表記に自動的に切り替わりません。
- openCypher CREATE では多値プロパティは作成されませんが、Gremlin を使用して作成されたデータに存在できます。Neptune openCypher が複数値プロパティを検出すると、いずれかの値が任意に選択され、非決定的な結果が作成されます。

Neptune のグラフデータモデル。

Amazon Neptune グラフデータの基本単位は 4 つの位置 (四角型) の要素で、リソース記述フレームワーク (RDF) の四角形に似ています。Neptune 四角形の 4 つの位置は次のとおりです。

- `subject  (S)`
- `predicate  (P)`
- `object  (O)`
- `graph  (G)`

各 quad は、1 つ以上のリソースについてアサーションを実行するステートメントです。ステートメントでは、2 つのリソース間の関係の有無に対してアサーションを行ったり、リソースにプロパティ (キーと値のペア) を付加したりすることができます。通常、四角形の述語値はステートメントの動詞と考えることができます。定義される関係またはプロパティのタイプについて説明します。オブジェクトは、関係のターゲット、またはプロパティの値を表します。次に例を示します。

- 2 つの頂点間の関係は、ソース頂点識別子を `S` の位置、ターゲット頂点識別子を `O` の位置、エッジラベルを `P` の位置に保存することによって表すことができます。
- プロパティは、要素識別子を `S` の位置、プロパティキーを `P` の位置、プロパティ値を `O` の位置に保存することで表すことができます。

`G` グラフの位置は、スタックごとに異なる方法で使用されます。Neptune の RDF データの場合、 `G` 位置には名前付きグラフ識別子が含まれます。Gremlin のプロパティグラフでは、エッジの場合にエッジ ID 値を保存するために使用されます。それ以外の場合、デフォルトは固定値です。

共有リソース識別子を持つ一連の quad ステートメントではグラフを作成します。

ユーザー向け値のディクショナリ

Neptune は、ほとんどのユーザー向け値を、管理するさまざまなインデックスに直接保存しません。代わりに、それらを個別にディクショナリに保存し、インデックス内の値を 8 バイトの識別子に置き換えます。

- `S`、`P`、または `G` インデックスに入れられるすべてのユーザー向け値は、このようにしてディクショナリに保存されます。
- `O` インデックスでは、数値はインデックスに直接格納されます (インライン化)。これには、`date` および `datetime` の値 (エポックからのミリ秒で表される) が含まれます。

- 0 インデックスに入れられるその他すべてのユーザー向け値は、ディクショナリに格納され、インデックス内で ID によって表されます。

ディクショナリには、ユーザー向け値から value_to_id インデックス内の 8 バイト ID へのフォワードマッピングが含まれています。

8 バイト ID から値への逆マッピングは、値のサイズに応じて 2 つのインデックスのうちの 1 つに格納されます。

- id_to_valueインデックスは、内部エンコーディング後に ID を 767 バイト未満のユーザー向け値にマッピングします。
- id_to_blob インデックスは、ID をより大きいユーザー向け値にマッピングします。

Neptune でステートメントのインデックスを作成する方法

quad のグラフをクエリする場合は、quad の位置ごとに値制約を指定するかどうかを選択できます。このクエリでは、指定した値制約に一致するすべての quad が返ります。

Neptune ではインデックスを使用してクエリを解決します。Andreas Harth 氏と Stefan Decker 氏が 2005 年の論文 (Optimized Index Structures for Querying RDF from the Web) で考察したように、4 つの quad 位置に対して 16 (2^4) のアクセスパターンがあります。6 つのクアッドステートメントインデックスを使用して、スキャンやフィルタリングを行うことなく、16 のパターンすべてに対して効率的にクエリを実行できます。各 quad ステートメントインデックスは、異なる順序で連結された 4 つの位置の値で構成されるキーを使用します。

```
     Access Pattern                                        Index key order
-----------------------------------------------------    ---------------
 1.  ????  (No constraints; returns every quad)              SPOG
 2.  SPOG  (Every position is constrained)                   SPOG
 3.  SPO?  (S, P, and O are constrained; G is not)           SPOG
 4.  SP??  (S and P are constrained; O and G are not)        SPOG
 5.  S???  (S is constrained; P, O, and G are not)           SPOG
 6.  S??G  (S and G are constrained; P and O are not)        SPOG

 7.  ?POG  (P, O, and G are constrained; S is not)           POGS
 8.  ?PO?  (P and O are constrained; S and G are not)        POGS
 9.  ?P??  (P is constrained; S, O, and G are not)           POGS
```

```
10.  ?P?G  (P and G are constrained; S and O are not)       GPSO
11.  SP?G  (S, P, and G are constrained; O is not)          GPSO
12.  ???G  (G is constrained; S, P, and O are not)          GPSO

13.  S?OG  (S, O, and G are constrained; P is not)          OGSP
14.  ??OG  (O and G are constrained; S and P are not)       OGSP
15.  ??O?  (O is constrained; S, P, and G are not)          OGSP

16.  S?O?  (S and O are constrained; P and G are not)       OSGP
```

Neptune は、デフォルトでは、これらの 6 つのインデックスのうち 3 つのみを作成および維持します。

- SPOG – では、Subject + Predicate + Object + Graph から構成されるキーを使用します。
- POGS – では、Predicate + Object + Graph + Subject から構成されるキーを使用します。
- GPSO – では、Graph + Predicate + Subject + Object から構成されるキーを使用します。

これら 3 つのインデックスは、最も一般的なアクセスパターンの多くを処理します。ステートメントのインデックスを 6 つではなく 3 つだけ維持することで、スキャンやフィルタリングを行わずに高速アクセスをサポートするために必要なリソースが大幅に削減されます。たとえば、SPOG インデックスでは、位置のプレフィックス (頂点または頂点とプロパティ識別子など) がバインドされるたびに効率的にルックアップできます。POGS インデックスでは、 P 位置に保存されている エッジまたはプロパティラベルのみがバインドされている場合に、効率的にアクセスできます。

ステートメントを検出するための低レベル API では、いくつかの位置が分かっており、残りの位置はインデックス検索による検出用に残されているステートメントパターンを使用します。ステートメントインデックスのいずれかのインデックスキーの順序に従って、既知の位置をキープレフィックスに構成することによって、Neptune は、既知の位置に一致するすべてのステートメントを検索するために範囲スキャンを実行します。

ただし、Neptune がデフォルトで作成しないステートメントインデックスの 1 つは、リバーストラバーサル OSGP インデックスです。このインデックスは、複数のオブジェクトやサブジェクトにまたがって述語を収集できます。代わりに、Neptune はデフォルトで {all P x POGS} の結合スキャンを行うために使用する別のインデックスで、個別の述語を追跡します。Gremlin を使用すると、述語はプロパティまたはエッジラベルに対応します。

グラフ内の個別の述語の数が多くなると、Neptune のデフォルトのアクセス方式は効率が悪くなる場合があります。たとえば、Gremlin の場合、エッジラベルが指定されていない `in()` ステップや、`in()` を内部で使用するステップ (`both()` や `drop()` など) は非常に効率が悪くなる場合があります。

ラボモードを使用した OSGP インデックス作成の有効化

データモデルで個別の述語を多数作成する場合、パフォーマンスが低下し、運用コストが高くなることがありますが、Neptune がデフォルトで維持する 3 つのインデックスに加えて、ラボモードを使用して OSPG インデックスを有効にすることで、これを大幅に改善できます。

> **Note**
>
> この機能は、Neptune エンジンリリース 1.0.1.0.200463.0 からアクセスできます。

OSPG インデックスの有効化には、次の欠点が伴う場合があります。

- 挿入速度が最大 23% 遅くなる場合がある。
- ストレージが最大 20% 増加する。
- すべてのインデックスに等しく接する読み取りクエリ (非常にまれです) のレイテンシーが増す場合がある。

ただし、一般的に、多数の個別の述語がある DB クラスターに対しては OSGP インデックスを有効にすることは価値があります。オブジェクトベースの検索 (頂点へのすべての着信エッジの検索や、特定のオブジェクトに接続されたすべてのサブジェクトの検索など) が非常に効率化され、その結果として頂点の削除も効率化されます。

> **Important**
>
> OSGP インデックスは、データを読み込む前の空の DB クラスターでのみ有効にできます。

Neptune データモデルにおける Gremlin ステートメント

Gremlin プロパティグラフデータは、次の 3 つのクラスのステートメントを使用して SPOG モデルで表されます。

- 頂点ラベルステートメント
- エッジステートメント
- プロパティステートメント

Gremlin クエリでこれらがどのように使用されるかについては、Neptune での Gremlin クエリの動作を理解する を参照してください。

Neptune ルックアップキャッシュは読み取りクエリを高速化できます。

Amazon Neptune は、R5d インスタンスの NVME ベースの SSD を使うルックアップキャッシュを実装しており、プロパティ値または RDF リテラルを頻繁に繰り返し検索するクエリの読み取りパフォーマンスを向上させます。ルックアップキャッシュは、これらの値をすばやくアクセスできる NVMe SSD ボリュームに一時的に保存します。

この機能は、Amazon Neptune エンジンバージョン 1.0.4.2.R2 (2021-06-01) で始めることで使用できます。

多数の頂点とエッジ、または多くの RDF トリプルのプロパティを返す読み取りクエリは、プロパティ値またはリテラルをメモリではなくクラスターストレージボリュームから取得する必要がある場合に、レイテンシーが高くなる可能性があります。たとえば、アイデンティティグラフから多数のフルネームを返す、または不正検出グラフから IP アドレスを返す、実行時間が長い読み取りクエリなどがあります。クエリによって返されるプロパティ値または RDF リテラルの数が増加すると、使用可能なメモリが減少し、クエリの実行が大幅に低下する可能性があります。

Neptune ルックアップキャッシュのユースケース

ルックアップキャッシュは、非常に多数の頂点とエッジ、または RDF トリプルのプロパティを読み取りクエリが返す場合にのみ役立ちます。

クエリのパフォーマンスを最適化するために、Amazon Neptune は R5d インスタンスタイプを用いてそのようなプロパティ値またはリテラル用の大きなキャッシュを作成します。キャッシュからそれらを取得することは、クラスターストレージボリュームからそれらを取得するよりもはるかに高速です。

経験則として、次の 3 つの条件がすべて満たされた場合にのみ、ルックアップキャッシュを有効にすることをお勧めします。

- 読み取りクエリのレイテンシーの増加が観察されている。
- また、読み取りクエリを実行するときに `BufferCacheHitRatio`CloudWatch メトリクスの低下も観察されます (「」を参照Amazon を使用した Neptune のモニタリング CloudWatch)。
- 読み取りクエリは、結果をレンダリングする前に戻り値をマテリアライズするのに多くの時間を費やしている (クエリでマテリアライズされているプロパティ値の数を確認する方法については、以下の Gremlin-profile の例を参照してください)。

> **Note**
>
> この機能は上記の特定のシナリオでのみ役に立ちます。たとえば、ルックアップキャッシュは集計クエリにまったく役立ちません。ルックアップキャッシュの恩恵を受けるクエリを実行していない限り、同等で安価な R5 インスタンスタイプの代わりに R5d インスタンスタイプを使う理由はありません。

Gremlin を使用している場合は、クエリのマテリアライズコストを Gremlin `profile` API で査定できます。「インデックス操作」の下には、実行中にマテリアライズされた項の数が表示されます。

```
Index Operations
Query execution:
    # of statement index ops: 3
    # of unique statement index ops: 3
    Duplication ratio: 1.0
    # of terms materialized: 5273
Serialization:
    # of statement index ops: 200
    # of unique statement index ops: 140
    Duplication ratio: 1.43
    # of terms materialized: 32693
```

マテ#アライズされる非数値項の数は、Neptune が実行しなければならない項ルックアップの数に正比例します。

ルックアップキャッシュの使用

ルックアップキャッシュは、R5d インスタンスタイプで、デフォルトで自動的に有効になります。Neptune R5d インスタンスの仕様は、R5 インスタンスに加え、最大 1.8 TB のローカル NVME ベースの SSD ストレージです。ルックアップキャッシュはインスタンス固有であり、利益をもたらすワークロードは Neptune クラスター内の R5d インスタンスに限定され、他のワークロードは R5 または他のインスタンスタイプを使用できます。

Neptune インスタンスでルックアップキャッシュを使用するには、そのインスタンスを R5d インスタンスタイプにアップグレードするだけです。そうすると、Neptune は自動的に neptune_lookup_cache DB クラスターのパラメータを `'enabled'` に設定し、その特定のインスタンスにルックアップキャッシュを作成します。その後、インスタンスのステータス API を使ってキャッシュが有効になったことを確認できます。

同様に、特定のインスタンスのルックアップキャッシュを無効にするには、インスタンスを R5d インスタンスタイプから同等の R5 インスタンスタイプにダウングレードします。

R5d インスタンスが起動されると、ルックアップキャッシュが有効になり、コールドスタートモードになります。つまり、空です。Neptune は、クエリの処理中にプロパティ値または RDF リテラルを最初にルックアップキャッシュでチェックし、それらが存在しない場合は追加します。これにより、キャッシュが徐々にウォームアップされます。

プロパティ値または RDFリテラルルックアップを必要とする読み取りクエリを R5d リーダーインスタンスに送信する場合、キャッシュがウォームアップする間は、読み取りパフォーマンスがわずかに低下します。ただし、キャッシュがウォームアップされると、読み取りパフォーマンスが大幅に向上し、クラスターストレージではなくルックアップがキャッシュにヒットすることに関連する I/O コストも低下することがあります。メモリ使用率も向上します。

ライターインスタンスが R5d の場合、書き込み操作ごとにルックアップキャッシュを自動的にウォームアップします。この方法では、書き込みクエリのレイテンシーがわずかに増加しますが、ルックアップキャッシュはより効率的にウォームアップします。次に、プロパティ値または RDFリテラルルックアップを必要とする読み取りクエリをライターインスタンスに転送すると、値がすでにキャッシュされているため、すぐに読み取りパフォーマンスが向上します。

また、バルクローダを R5d で実行すると、ライターインスタンスでは、キャッシュのためにパフォーマンスがわずかに低下していることに気付くかもしれません。

ルックアップキャッシュは各ノードに固有であるため、ホスト置換によってキャッシュがコールドスタートにリセットされます。

DB クラスター内のすべてのインスタンスのルックアップキャッシュを一時的に無効にするには、neptune_lookup_cache DB クラスターのパラメータを `'disabled'` に設定します。ただし、一般的には、特定のインスタンスのキャッシュを R5d から R5 インスタンスタイプスケールダウンして無効にする方が理にかなっています。

Neptune でのトランザクションセマンティクス

Amazon Neptune は、データグラフに対する同時実行性の高いオンライントランザクション処理 (OLTP) ワークロードをサポートするように設計されています。RDF 仕様の W3C SPARQL クエリ言語と Apache TinkerPop Gremlin グラフトラバーサル言語ドキュメントでは、同時クエリ処理のトランザクションセマンティクスは定義されていません。ACID のサポートと明確に定義されたトランザクションの保証は非常に重要であるため、データの異常を回避するために厳格なセマンティクスを実施しています。

このセクションでは、これらのセマンティクスを定義し、Neptune のいくつかの一般的なユースケースにどのように適用されるかを説明します。

トピック

分離レベルの定義

ACID で「I」は分離を表します。トランザクションの分離度によって、他の同時トランザクションが処理するデータにどの程度影響するかが決まります。

SQL:1992 Standard では、分離レベルを記述するための語彙を作成しました。これは、3種類のインタラクションを定義します (現象と呼ぶものです)。これは、2 つの同時トランザクション Tx1 および Tx2 間で発生する可能性があります。

- Dirty read – これは、Tx1 が項目を変更し、Tx1 がその変更をコミットする前にその項目を Tx2 が読み取るときに発生します。その後、Tx1 が変更のコミットに成功しなかったり、ロールバックされたりすると、Tx2 はデータベースにそれを行わなかった値を読み取ります。
- Non-repeatable read – これは、Tx1 が項目を読み取り、Tx2 がその項目を変更または削除して変更をコミットし、Tx1 がその項目の再読み込みを試みるときに発生します。Tx1 は、以前とは異なる値を読み取るか、項目が存在しなくなったことを検出します。
- Phantom read – これは、Tx1 が検索条件を満たす一連の項目を読み取り、Tx2 が検索条件を満たす新しい項目を追加し、Tx1 が検索を繰り返したときに発生します。Tx1 は、以前とは異なる一連の項目を取得するようになりました。

これら 3 つのタイプの操作はそれぞれ、データベースの結果データに不整合が生じる可能性があります。

SQL:1992 標準では、3 つのタイプのインタラクションとそれらが生成できる不整合の点で異なる保証を持つ 4 つの分離レベルが定義されています。すべての 4 つのレベルで、トランザクションが完全に実行されるか、まったく実行されないかが保証されます。

- `READ UNCOMMITTED` – 3 種類のインタラクション (ダーティリード、非再現リード、ファントムリード) をすべて許可します。
- `READ COMMITTED` – ダーティリードは可能ではありませんが、非再現リードとファントムリードは可能です。
- `REPEATABLE READ` – ダーティリードも非再現リードも可能ではありませんが、ファントムリードは可能です。
- `SERIALIZABLE` – 3 つのタイプの相互作用現象は発生しません。

マルチバージョン同時実行制御 (MVCC) では、1 つの他の種類の分離、つまり SNAPSHOT 分離が可能です。これにより、トランザクションが開始されたときに存在するデータのスナップショットに対してトランザクションが実行され、他のトランザクションがそのスナップショットを変更できないことが保証されます。

Neptune でのトランザクション分離レベル

Amazon Neptune は、読み取り専用クエリとミューテーションクエリに異なるトランザクション分離レベルを実装します。SPARQL および Gremlin クエリは、以下の基準に基づいて読み取り専用またはミューテーションとして分類されます。

- SPARQL では、読み取りクエリ (`SELECT`、`ASK`、`CONSTRUCT`、および`DESCRIBE`SPARQL 1.1 クエリ言語仕様で定義される)、とミューテーションクエリ (`INSERT` および `DELETE`、SPARQL 1.1 アップデート仕様で定義される) の明確な区別があります。

  Neptune は、一緒に送信された複数のミューテーションクエリ (セミコロンで区切られた `POST` メッセージなど) を 1 つのトランザクションとして扱います。これらは、アトミックユニットとして成功するか失敗するかが保証されています。障害が発生した場合、部分的な変更がロールバックされます。
- ただし、Gremlin では、Neptune は、データを操作する `addE()`、`addV()`、`property()`、`drop()` などのクエリパスステップが含まれているかどうかに基づいて、クエリを読み取り専用クエリまたはミューテーションクエリとして分類します。クエ

リにそのようなパスステップが含まれている場合、クエリはミューテーションクエリとして分類され、実行されます。

Gremlin でスタンドセッションを使用することもできます。詳細については、「Gremlin スクリプトベースのセッション」を参照してください。これらのセッションでは、読み取り専用クエリを含め、すべてのクエリが、ライターエンドポイントに対するミューテーションクエリと同じ分離下で実行されます。

openCypher で Bolt 読み取り/書込みセッションを使用して、読み取り専用クエリを含め、すべてのクエリが、ライターエンドポイントに対するミューテーションクエリと同じ分離下で実行されます。

トピック

- Neptune での読み取り専用クエリの分離
- Neptune でのミューテーションクエリの分離
- ロック待機タイムアウトを使用した競合の解決
- 範囲ロックと誤った競合

Neptune での読み取り専用クエリの分離

Neptune は、スナップショット分離セマンティクスで読み取り専用クエリを評価します。つまり、読み取り専用クエリは、クエリ評価の開始時に作成されたデータベースの整合性のあるスナップショットで論理的に動作します。Neptune は、次の現象が発生しないことを保証します。

- `Dirty reads` – Neptune の読み取り専用クエリでは、同時トランザクションからのコミットされていないデータを参照しません。
- `Non-repeatable reads` – 同じデータを複数回読み取る読み取り専用トランザクションは、常に同じ値を返します。
- `Phantom reads` – 読み取り専用トランザクションは、トランザクションの開始後に追加されたデータを読み取ることはありません。

スナップショットの分離は複数バージョンの同時実行制御 (MVCC) を使用して実現されるため、読み取り専用クエリはデータをロックする必要がなく、ミューテーションクエリをブロックしません。

リードレプリカは読み取り専用クエリのみを受け付けるため、リードレプリカに対するすべてのクエリは `SNAPSHOT` 分離セマンティクスで実行されます。

リードレプリカをクエリするときの唯一の追加の考慮事項は、ライターとリードレプリカの間にわずかなレプリケーション遅延が生じる可能性があることです。つまり、ライターで行われた更新が、読み取り元のリードレプリカに反映されるまで少し時間がかかることがあります。実際のレプリケーション時間は、プライマリインスタンスに対する書き込み負荷によって異なります。Neptune アーキテクチャは低レイテンシーのレプリケーションをサポートし、レプリケーションラグは Amazon CloudWatch メトリクスに計測されます。

それでも、SNAPSHOT 分離レベルのため、読み込みクエリは、たとえ最新でない場合でも、常に一貫性のあるデータベースの状態を参照します。

クエリが以前の更新の結果を確認することを強く保証する必要がある場合は、リードレプリカではなくライターエンドポイント自体にクエリを送信します。

Neptune でのミューテーションクエリの分離

ミューテーションクエリの一部として行われた読み込みは、READ COMMITTED トランザクション分離下で実行され、ダーティな読み取りの可能性は除外されます。Neptune は、READ COMMITTED トランザクション分離で提供される通常の保証を超えて、NON-REPEATABLE も PHANTOM 読み取りも発生しないという強力な保証を提供します。

これらの強力な保証は、データの読み取り時にレコードやレコードの範囲をロックすることによって実現されます。これにより、読み取り後に同時トランザクションがインデックス範囲で挿入または削除を行わないため、同じ読み込みの繰り返しが保証されます。

> **Note**
>
> ただし、同時ミューテーショントランザクション Tx2 は、ミューテーショントランザクション Tx1 の開始後に開始でき、Tx1 がロックデータを読み取る前に変更をコミットできます。この場合、Tx1 は Tx1 の開始前に Tx2 が完了したかのように Tx2 の変更を参照します。これはコミットされた変更にのみ適用されるため、dirty read が発生することはありません。

Neptune がミューテーションクエリに使用するロックメカニズムを理解するには、まずNeptune の詳細を理解することが役立ちます。グラフデータモデルそしてインデックス作成戦略。Neptune は SPOG、POGS、および GPSO の 3 つのインデックスを使用してデータを管理します。

READ COMMITTED トランザクションレベルで繰り返し可能な読み込みを実現するために、Neptune は使用されているインデックスの範囲ロックを取得します。たとえば、ミューテーションクエリが person1 という名前の頂点のすべてのプロパティと出力エッジを読み取る場合、ノードはデータを

読み取る前に SPOG インデックスのプレフィックス S=person1 で定義された範囲全体をロックします。

他のインデックスを使用する場合も同じメカニズムが適用されます。たとえば、ミューテーショントランザクションが、POGS インデックスを使用して特定のエッジラベルのソースとターゲットの頂点ペアをすべて検索すると、P 位置のエッジラベルの範囲がロックされます。同時トランザクションは、読み取り専用クエリかミューテーションクエリかにかかわらず、ロックされた範囲内で読み取りを実行できます。ただし、ロックされたプレフィックス範囲内の新しいレコードの挿入または削除を含むミューテーションは、排他的なロックを必要とし、禁止されます。

つまり、インデックスの範囲がミューテーショントランザクションによって読み取られた場合、この範囲は読み取りトランザクションが終了するまで同時トランザクションによって変更されないという強力な保証があります。これにより、non-repeatable reads が発生しないことが保証されます。

ロック待機タイムアウトを使用した競合の解決

2 番目のトランザクションが、最初のトランザクションがロックした範囲のレコードを変更しようとすると、Neptune は競合を即座に検出し、2 番目のトランザクションをブロックします。

依存関係のデッドロックが検出されない場合、Neptune は自動的にロック待機タイムアウトメカニズムを適用します。このメカニズムでは、ブロックされたトランザクションは、ロックを保持しているトランザクションが終了してロックを解放するまで最大 60 秒待機します。

- ロックが解放される前にロック待機タイムアウトが期限切れになると、ブロックされたトランザクションはロールバックされます。
- ロックがロック待機タイムアウト内に解放された場合、2 番目のトランザクションはブロック解除され、再試行することなく正常に終了できます。

ただし、Neptune が 2 つのトランザクション間の依存関係デッドロックを検出した場合、競合の自動調整はできません。この場合、Neptune はロック待機タイムアウトを開始することなく、2 つのトランザクションの 1 つを直ちにキャンセルしてロールバックします。Neptune は、挿入または削除されたレコードが最も少ないトランザクションをロールバックするよう最善を尽くします。

範囲ロックと誤った競合

Neptune はギャップロックを使用して範囲ロックを行います。ギャップロックは、インデックスレコード間のギャップに対するロック、または最初のインデックスレコードの前または最後のインデックスレコードの後のギャップに対するロックです。

Neptune は、いわゆるディクショナリテーブルを使用して、数値 ID 値を特定の文字列リテラルに関連付けます。このような Neptune ディクショナリテーブルのサンプルステートは次のとおりです。

文字列	ID
type	1
default_graph	2
person_3	3
person_1	5
knows	6
person_2	7
年齢	8
edge_1	9
lives_in	10
New York	11
個人	12
Place	13
edge_2	14

上記の文字列はプロパティグラフモデルに属しますが、概念はすべての RDF グラフモデルにも等しく当てはまります。

対応する SPOG (Subject-Predicate-Object_Graph) インデックスの状態は、以下の左側に示されています。右側には、インデックスデータの意味を理解しやすいように、対応する文字列が示されています。

S (ID)	P (ID)	O (ID)	G (ID)		S (文字列)	P (文字列)	O (文字列)	G (文字列)
3	1	12	2		person_3	type	個人	default_graph
5	1	12	2		person_1	type	個人	default_graph
5	6	3	9		person_1	knows	person_3	edge_1
5	8	40	2		person_1	年齢	40	default_graph
5	10	11	14		person_1	lives_in	New York	edge_2
7	1	12	2		person_2	type	個人	default_graph
11	1	13	2		New York	type	Place	default_graph

今度は、ミューテーションクエリが `person_1` という名前の頂点のすべてのプロパティと出力エッジを読み取る場合、ノードはデータを読み取る前に SPOG インデックスのプレフィックス `S=person_1` で定義された範囲全体をロックします。範囲ロックは、一致するすべてのレコードと、一致しない最初のレコードにギャップロックをかけます。一致するレコードはロックされ、一致しないレコードはロックされません。Neptune は、次のようにギャップロックを掛けます。

- `5 1 12 2` (ギャップ 1)
- `5 6 3 9` (ギャップ 2)
- `5 8 40 2` (ギャップ 3)
- `5 10 11 14` (ギャップ 4)
- `7 1 12 2` (ギャップ 5)

これにより、以下のレコードがロックされます。

- 5 1 12 2
- 5 6 3 9
- 5 8 40 2
- 5 10 11 14

この状態では、以下の操作は合法的にブロックされます。

- S=person_1 について新しいプロパティまたはエッジの挿入。type と異なる新しいプロパティや新しいエッジは、ギャップ 2、ギャップ 3、ギャップ 4、ギャップ 5 のいずれかに配置する必要があり、これらはすべてロックされます。
- 既存のレコードのいずれかの削除。

同時に、いくつかの同時操作が誤ってブロックされます (誤った競合が発生します)。

- S=person_3 のプロパティやエッジの挿入は、ギャップ 1 に入らなければならないためブロックされます。
- 3 から 5 の間の ID を割り当てられた新しい頂点の挿入は、ギャップ 1 に入らなければならないためブロックされます。
- 5 から 7 の間の ID を割り当てられた新しい頂点の挿入は、ギャップ 5 に入らなければならないためブロックされます。

ギャップロックは、特定の 1 つの述部のギャップをロックするほど正確ではありません (例えば、述語 S=5 の場合、gap5 をロックするなど)。

範囲ロックは読み取りが行われるインデックスにのみ適用されます。上記の場合、レコードは SPOG インデックスでのみロックされ、POGS や GPSO ではロックされません。アクセスパターンによっては、クエリの読み取りがすべてのインデックスに対して行われる場合があります。アクセスパターンは explain API (Sparql および Gremlin の場合) を使用して一覧表示できます。

> **Note**
>
> また、基礎となるインデックスを安全に同時更新するためにギャップロックを適用することもできますが、これによって誤った競合が発生する可能性もあります。これらのギャップロックは、トランザクションによって実行される分離レベルや読み取り操作とは無関係に設定されます。

誤った競合は、ギャップロックが原因で同時実行中のトランザクションが衝突したときだけでなく、何らかの障害が発生した後にトランザクションが再試行された場合も発生する可能性があります。障害によってトリガーされたロールバックがまだ進行中で、そのトランザクションで以前に掛けられたロックがまだ完全に解除されていない場合、再試行は誤った競合が発生して失敗します。

負荷が高いと、通常、書き込みクエリの 3 ~ 4% が誤った競合のために失敗することがあります。外部クライアントの場合、このような誤った競合は予測が難しく、再試行を使用して処理する必要があります。

Neptune トランザクションセマンティクスの例

次の例は、Amazon Neptune でのトランザクションセマンティクスのさまざまなユースケースを示しています。

トピック

- 例 1 – 存在しない場合にのみプロパティを挿入する
- 例 2 – プロパティ値がグローバルに一意であるというアサート
- 例 3 – 別のプロパティに指定した値がある場合のプロパティの変更
- 例 4 – 既存のプロパティの置き換え
- 例 5 – ダングリングプロパティまたはエッジを回避する

例 1 – 存在しない場合にのみプロパティを挿入する

プロパティが 1 回のみ設定されることを保証したいとします。たとえば、複数のクエリが 1 人の人に同時にクレジットスコアを割り当てようとしているとします。プロパティのインスタンスを 1 つだけ挿入し、他のクエリはプロパティが既に設定されているため失敗するようにしたいとします。

```
# GREMLIN:
g.V('person1').hasLabel('Person').coalesce(has('creditScore'), property('creditScore',
 'AAA+'))

# SPARQL:
INSERT { :person1 :creditScore "AAA+" .}
WHERE  { :person1 rdf:type :Person .
         FILTER NOT EXISTS { :person1 :creditScore ?o .} }
```

Gremlin property() ステップは、指定されたキーと値を持つプロパティを挿入します。coalesce() ステップは最初のステップで最初の引数を実行し、失敗した場合、2 番目のステップを実行します。

特定の person1 頂点の creditScore プロパティの値を挿入する前に、トランザクションは person1 の存在しない可能性のある creditScore 値を読み取ろうとします。この試行された読み取りは、creditScore 値が存在するか書き込まれる SPOG インデックスの S=person1 および P=creditScore の SP 範囲がロックされます。

この範囲ロックを使用すると、同時トランザクションが同時に creditScore 値を挿入できなくなります。複数の並列トランザクションがある場合、それらのうちの 1 つが一度に値を更新できます。これにより、複数の creditScore プロパティが作成される異常が除外されます。

例 2 – プロパティ値がグローバルに一意であるというアサート

社会保障番号をプライマリキーとして、人を挿入するとします。ミューテーションクエリでは、グローバルレベルで、データベース内の他の誰も同じ社会保障番号を持っていないことを保証する必要があります。

```
# GREMLIN:
g.V().has('ssn', 123456789).fold()
  .coalesce(__.unfold(),
            __.addV('Person').property('name', 'John Doe').property('ssn', 123456789'))

# SPARQL:
INSERT { :person1 rdf:type :Person .
         :person1 :name "John Doe" .
         :person1 :ssn 123456789 .}
WHERE  { FILTER NOT EXISTS { ?person :ssn 123456789 } }
```

この例は、前の例に似ています。主な違いは、範囲ロックが SPOG インデックスではなく POGS インデックスに取られることです。

クエリを実行するトランザクションは、P および O の位置がバインドされているパターン ? person :ssn 123456789 を読み取る必要があります。範囲ロックは、POGS インデックスの P=ssn と O=123456789 で取得されます。

- パターンが存在する場合、アクションは実行されません。
- 存在しない場合、ロックは同時トランザクションがその社会保障番号を挿入するのを防ぎます。

例 3 – 別のプロパティに指定した値がある場合のプロパティの変更

ゲームのさまざまなイベントが、レベル 1 からレベル 2 に人を移動し、それらにゼロに設定された新しい level2Score プロパティを割り当てるとします。このようなトランザクションの複数の同時インスタンスが、レベル 2 スコアプロパティの複数のインスタンスを作成できないことを確認する必要があります。Gremlin と SPARQL のクエリは次のようになります。

```
# GREMLIN:
g.V('person1').hasLabel('Person').has('level', 1)
 .property('level2Score', 0)
 .property(Cardinality.single, 'level', 2)

# SPARQL:
DELETE { :person1 :level 1 .}
INSERT { :person1 :level2Score 0 .
         :person1 :level 2 .}
WHERE  { :person1 rdf:type :Person .
         :person1 :level 1 .}
```

Gremlin で、Cardinality.single が指定されている場合、property( ) ステップは新しいプロパティを追加するか、既存のプロパティの値を指定された新しい値に置き換えます。

level を 1 から 2 に増やすなど、プロパティ値に対する更新は、現在のレコードの削除と新しいプロパティ値を持つ新しいレコードの挿入として実装されます。この場合、レベル番号 1 のレコードは削除され、レベル番号 2 のレコードが再挿入されます。

トランザクションが level2Score を追加し、level を 1 から 2 に更新できるようにするには、まず level 値が現在 1 に等しいことを検証する必要があります。その際、SPOG インデックスの S=person1、P=level、および O=1 の SPO プレフィックスで範囲ロックを取得します。このロックにより、同時トランザクションがバージョン 1 のトリプルを削除できなくなり、その結果、競合する同時更新は発生しません。

例 4 – 既存のプロパティの置き換え

特定のイベントでは、個人のクレジットスコアが新しい値 (ここでは BBB) に更新される場合があります。ただし、そのタイプの同時イベントが 1 人の人に対して複数のクレジットスコアプロパティを作成できないようにする必要があります。

```
# GREMLIN:
g.V('person1').hasLabel('Person')
```

```
  .sideEffect(properties('creditScore').drop())
  .property('creditScore', 'BBB')

# SPARQL:
DELETE { :person1 :creditScore ?o .}
INSERT { :person1 :creditScore "BBB" .}
WHERE  { :person1 rdf:type :Person .
         :person1 :creditScore ?o .}
```

このケースは例 3 に似ていますが、SPO プレフィックスをロックするのではなく、Neptune は S=person1 と P=creditScore のみで SP プレフィックスをロックします。これにより、同時トランザクションが、person1 サブジェクトの creditScore プロパティを持つトリプルを挿入または削除することを防ぎます。

例 5 – ダングリングプロパティまたはエッジを回避する

エンティティの更新は、ダングリング要素、つまり、型付けされていないエンティティに関連付けられたプロパティまたはエッジを残すことはできません。Gremlin には、ダングリング要素を防ぐための制約が組み込まれているため、これは SPARQL でのみ問題になります。

```
# SPARQL:
tx1: INSERT { :person1 :age 23 } WHERE { :person1 rdf:type :Person }
tx2: DELETE { :person1 ?p ?o }
```

INSERT クエリでは、SPOG インデックスの S=person1 および P=rdf:type の SPO プレフィックス、および O=Person を読み取り、ロックする必要があります。ロックにより、 DELETE クエリが並列で成功するのを防ぎます。

:person1 rdf:type :Person レコードを削除しようとする DELETE クエリと、レコードを読み取り SPOG インデックスでその SPO で範囲ロックを作成する INSERT クエリとの競合では、次のような結果が生じる可能性があります。

- DELETE クエリが :person1 のすべてのレコードを読み取り、削除する前に INSERT クエリがコミットされた場合、:person1 は、新しく挿入されたレコードを含め、データベースから完全に削除されます。
- INSERT クエリが :person1 rdf:type :Person レコードを読み取る前に DELETE クエリがコミットされた場合、読み取りはコミットされた変更を確認します。つまり、:person1 rdf:type :Person レコードが見つからないため、no-op になります。

- DELETE クエリが実行される前に INSERT クエリが読み取ると、最初の例のように、`:person1 rdf:type :Person` トリプルがロックされ、INSERT クエリがコミットされるまで DELETE クエリがブロックされます。
- DELETE がクエリ INSERT の前に読み取り、INSERT クエリがレコードの SPO プレフィックスを読み取ってロックしようとすると、競合が検出されます。これは、トリプルが削除対象としてマークされ、INSERT が失敗するためです。

これらのさまざまなイベントシーケンスでは、ダングリングエッジは作成されません。

例外処理と再試行

解決できない競合またはロック待機タイムアウトのためにトランザクションがキャンセルされると、Amazon Neptune は ConcurrentModificationException で応答します。詳細については、「エンジンエラーコード」を参照してください。ベストプラクティスとして、クライアントは常にこれらの例外をキャッチして処理する必要があります。

多くの場合、ConcurrentModificationException インスタンスの数が少ない場合、エクスポネンシャルバックオフベースの再試行メカニズムが例外を処理する方法として機能します。このような再試行アプローチでは、最大の再試行回数と待機時間は、通常、トランザクションの最大サイズと期間によって異なります。

ただし、アプリケーションに同時実行数の多い更新ワークロードがあり、多数の ConcurrentModificationException イベントが発生した場合は、アプリケーションを変更して、競合する同時変更の回数を減らすことができます。

たとえば、頂点のセットを頻繁に更新し、これらの更新に複数の同時スレッドを使用して書き込みスループットを最適化するアプリケーションがあったとします。各スレッドが 1 つ以上のノードプロパティを更新するクエリを継続的に実行する場合、同じノードの同時更新は ConcurrentModificationException を生成できます。これにより、書き込みパフォーマンスが低下する可能性があります。

互いに競合する可能性が高い更新をシリアル化できると、このような競合の可能性を大幅に削減できます。たとえば、特定のノードに対する更新クエリがすべて同じスレッドで作成されるようにできる場合 (ハッシュベースの割り当てを使用する場合など)、それらのクエリが同時にではなく 1 つずつ実行されるようにできます。隣接するノードで範囲ロックによって ConcurrentModificationException が発生する可能性はありますが、同じノードに対する同時更新は排除されます。

Amazon Neptune DB クラスターとインスタンス

Amazon Neptune DB クラスタークエリを通じてデータへのアクセスを管理します。クラスターは、次のとおりです。

- 1 つのプライマリ DB インスタンス。。
- 最大 15 のリードレプリカ DB インスタンス。。

クラスターのすべてのインスタンスは同じ基盤となる管理ストレージボリューム共有します。これは、信頼性と高可用性を重視して設計されています。

DB クラスター内の DB インスタンスに接続するには、Neptune エンドポイントを使います。

Neptune DB クラスター内のプライマリ DB インスタンス

プライマリ DB インスタンスは、DB クラスターの基盤となるストレージボリュームへのすべての書き込み操作を調整します。読み取りオペレーションもサポートします。

Neptune DB クラスターには 1 つのプライマリ DB インスタンスしか存在できません。プライマリインスタンスが使用できなくなると、Neptune は、指定可能な優先度を使用して、リードレプリカインスタンスの 1 つに自動的にフェールオーバーします。

Neptune DB クラスター内のリードレプリカ DB インスタンス

DB クラスターのプライマリインスタンスを作成したら、DB クラスターに最大 15 の リードレプリカリードレプリカ レプリカを作成して、読み取り専用クエリをサポートできます。

Neptune リードレプリカ DB インスタンスは、クラスターボリュームでの読み取りオペレーションに特化しているため、読み取りのスケーリングに最適です。すべての書き込みオペレーションはプライマリインスタンスによって管理されます。各リードレプリカ DB インスタンスには、独自のエンドポイントがあります。

クラスターストレージボリュームはクラスター内のすべてのインスタンス間で共有されるため、すべてのリードレプリカインスタンスは、レプリケーションの遅れをほとんど伴わずクエリ結果に対して同じデータを返します。このラグは、通常はプライマリインスタンスが更新を書き込んだ後、100 ミリ秒未満です。ただし、書き込みオペレーションの数が非常に多い場合、多少長くなります。

異なるアベイラビリティーゾーンで 1 つ以上のリードレプリカインスタンスを利用可能にすると、リードレプリカがプライマリインスタンスのフェイルオーバーターゲットとして機能するため、可用性が向上します。つまり、プライマリインスタンスが失敗した場合、Neptune はリードレプリカを

プライマリインスタンスに昇格します。そうすると、プライマリインスタンスに対して行われた読み取り要求と書き込み要求が失敗して例外が発生すると、短時間の中断が発生し、昇格したインスタンスは再起動されます。

対照的に、DB クラスターにリードレプリカインスタンスが含まれていない場合、プライマリインスタンスが再作成されるまで障害が発生しても、DB クラスターは使用できないままになります。プライマリインスタンスの再作成は、リードレプリカの昇格よりもかなり時間がかかります。

高可用性を確保するために、プライマリインスタンスと同じ DB インスタンスクラスを持ち、プライマリインスタンスとは異なるアベイラビリティーゾーンに配置する 1 つ以上のリードレプリカインスタンスを作成することをお勧めします。Neptune DB クラスターの耐障害性 を参照してください。

コンソールを使用すると、DB クラスターを作成する際にマルチ AZ を指定するだけでマルチ AZ 配置を作成できます。DB クラスターが 1 つのアベイラビリティーゾーンにある場合は、別のアベイラビリティーゾーンに Neptune レプリカを追加して、マルチ AZ の DB クラスターにすることができます。

> **Note**
>
> 暗号化されていない Neptune DB クラスターには暗号化されたリードレプリカインスタンスや、暗号化された Neptune DB クラスターの暗号化されていないリードレプリカインスタンスを作成することはできません。

Neptune リードレプリカ DB インスタンスの作成方法の詳細については、コンソールを使用して Neptune リーダーインスタンスを作成する を参照してください。

Neptune DB クラスターでの DB インスタンスのサイジング

CPU とメモリの要件に基づいて、Neptune DB クラスター内のインスタンスのサイズを設定します。インスタンス上の vCPUs の数によって、着信クエリを処理するクエリスレッドの数が決まります。インスタンスのメモリ容量によって、基礎となるストレージボリュームからフェッチされたデータページのコピーを格納するために使用されるバッファキャッシュのサイズが決まります。

各 Neptune DB インスタンスには、そのインスタンスの vCPUs の 2 x 数に等しい数のクエリスレッドがあります。例えば、16 の vCPU がある `r5.4xlarge` では、32 のクエリスレッドがあるため、32 のクエリを同時に処理できます。

すべてのクエリスレッドが占有されている間に到着する追加のクエリは、サーバーサイドキューに入れられ、クエリスレッドが使用可能になると FIFO 方式で処理されます。このサー

バーサイドキューは、約 8000 の保留中の要求を保持できます。一杯になったら、Neptune は追加のリクエストに ThrottlingException で応答します。保留中のリクエストの数は、MainRequestQueuePendingRequests CloudWatch メトリクスでモニタリングするか、includeWaitingパラメータで Gremlin クエリステータスエンドポイントを使用してモニタリングできます。

クライアントの観点から見たクエリの実行時間には、実際にクエリを実行するのに要した時間に加えて、キューに費やされた時間も含まれます。

プライマリ DB インスタンスのすべてのクエリスレッドを使用する持続的な同時書き込みロードは、理想的には 90% 以上の CPU 使用率を示します。これは、サーバー上のすべてのクエリスレッドが有効な作業に積極的に従事していることを示します。ただし、同時書き込み負荷が持続していても、実際の CPU 使用率はやや低くなります。これは通常、クエリスレッドが基になるストレージボリュームへの I/O オペレーションが完了するのを待機中であるためです。Neptune は、3 つのアベイラビリティーゾーンにまたがるデータのコピーを 6 つ作成するクォーラム書き込みを使用します。また、これらの 6 つのストレージノードのうち 4 つの耐久性を考慮するために、書き込みを認識する必要があります。クエリスレッドがストレージボリュームからこのクォーラムを待っている間、スレッドは停止し、CPU 使用率が低下します。

次々に書き込みを実行し、次の書き込みを開始する前に最初の書き込みが完了するのを待っているシリアル書き込みロードがある場合、CPU 使用率が低下すると予想されます。正確な量は、vCPUs とクエリスレッドの数の関数になります (クエリスレッドが多いほど、クエリあたりの全体的な CPU は少なくなります)。I/Oを待機することでいくらか減少します。

DB インスタンスのサイズを決める方法の詳細については、「適切な Neptune DB インスタンスタイプを選択する」を参照してください。これらのファミリーの各インスタンスタイプの料金については、Neptune の料金表ページをご覧ください。

Neptune での DB インスタンスのパフォーマンスのモニタリング

Neptune の CloudWatch メトリクスを使用して DB インスタンスのパフォーマンスをモニタリングし、クライアントが観察したクエリレイテンシーを追跡できます。CloudWatch を使用して Neptune で DB インスタンスのパフォーマンスをモニタリングする を参照してください。

Amazon Neptune ストレージ、信頼性、可用性

Amazon Neptune では、データベースストレージのニーズが増大すると自動的に拡張される分散型および共有ストレージアーキテクチャが使用されます。

Neptune のデータは、NVMe (NVMe) SSD ベースのドライブを使用する単一の仮想ボリュームであるクラスターボリュームに保存されます。クラスターボリュームは、論理ブロックのコレクションで構成されます。これらのセグメントにはそれぞれ 10 ギガバイト (GB) のストレージが割り当てられます。各セグメントのデータは 6 つのコピーにレプリケートされ、その後、DB クラスターが存在する AWS リージョンの 3 つのアベイラビリティーゾーン (AZ) に割り当てられます。

Neptune DB クラスターが作成されると、10 GB の単一のセグメントが割り当てられます。データ量が増加し、現在割り当てられているストレージを超えると、Neptune は新しいセグメントを追加してクラスターボリュームを自動的に拡張します。Neptune クラスターボリュームは、中国と を除くサポートされているすべてのリージョンで最大サイズ 128 テビバイト (TiB) まで拡張できます。ただし GovCloud、64 TiB に制限されています。ただし、リリース： 1.0.2.2 (2020-03-09) より前のエンジンリリースでは、クラスターボリュームのサイズはすべてのリージョンで 64 TiB に制限されます。

DB クラスターボリュームには、すべてのユーザーデータ、インデックス、およびディクショナリが含まれます (Neptune のグラフデータモデル。 セクション、および内部トランザクションログなどの内部メタデータ。インデックスや内部ログを含め、このグラフデータはすべて、クラスターボリュームの最大サイズを超えることはできません。

I/O 最適化ストレージオプション

Neptune は、ストレージに対して、次の 2 つの料金モデルを提供しています。

- 標準ストレージ — 標準ストレージは、I/O 使用率が低いか中程度のアプリケーション向けの費用対効果の高いデータベースストレージです。
- I/O 最適化ストレージ — I/O 最適化ストレージでは、使用したストレージの分のみ料金が発生します。標準ストレージよりもコストが高く、使用した I/O に対する料金は発生しません。

  I/O 最適化ストレージは、コストが予測可能で、I/O レイテンシーが低く、I/O スループットが一貫している、I/O 負荷の高いグラフワークロードのニーズを満たすように設計されています。

  詳細については、「I/O 最適化ストレージ」を参照してください。

Neptune ストレージ割り当て

Neptune クラスターボリュームは 128 TiB (または一部のリージョンでは 64 TiB) まで拡大できますが、実際に割り当てられたスペースに対してのみ料金が請求されます。割り当てられる合計容量は、ストレージハイウォーターマークによって決まります。これは、クラスタボリュームの存在中の任意の時点でクラスタボリュームに割り当てられる最大量です。

つまり、ユーザーデータがクラスターボリュームから削除された (`g.V().drop()` のようなドロップクエリなど) の場合でも、割り当てられた領域の合計は同じままです。Neptune は、未使用の割り当て領域を自動的に最適化して、将来再利用します。

ユーザーデータに加えて、2 種類のコンテンツでは、ディクショナリデータと内部トランザクションログなどの内部記憶領域が消費されます。ディクショナリデータはグラフデータとともに保存されますが、サポートするグラフデータが削除された場合でも無期限に保持されます。つまり、データが再導入された場合にエントリを再利用できます。内部ログデータは、独自のハイウォーターマークが付いた個別の内部ストレージスペースに格納されます。内部ログの有効期限が切れると、占有したストレージを他のログに再利用できますが、グラフデータには使用できません。ログに割り当てられた内部スペースの量は、`VolumeBytesUsed`CloudWatch メトリクス によってレポートされる合計スペースに含まれます。

割り当てられたストレージを最小限に保ち、スペースを再利用する方法を ストレージのベストプラクティス で確認してください。

Neptune ストレージ請求

ストレージコストは上述の通り、ストレージハイウォーターマークに基づいて請求されます。データは 6 つのコピーにレプリケートされますが、課金されるのはデータの 1 つのコピーのみです。

DB クラスターの現在のストレージハイウォーターマークは、 `VolumeBytesUsed` CloudWatch メトリクスをモニタリングすることで確認できます (「」を参照Amazon を使用した Neptune のモニタリング CloudWatch) 。

Neptune ストレージコストに影響するその他の要因には、データベースのスナップショットとバックアップが含まれます。バックアップストレージとして別途請求され、Neptune ストレージコストに基づきます (Neptune バックアップストレージの管理に役立つ CloudWatch メトリクス参照)。

ただし、クローンを作成した場合、データベースのうち、クローンは DB クラスター自体が使用しているのと同じクラスターボリュームを指しているため、元のデータに対する追加のストレージ料金は発生しません。クローンをそれ以降に変更すると、copy-on-write プロトコル が使用され、追加のストレージコストが発生します。

Neptune の料金の詳細については、Amazon Neptune Pricing を参照してください。

Neptune ストレージのベストプラクティス

特定の種類のデータは Neptune の永続ストレージを消費するため、ストレージの大きな急増を回避するには、次のベストプラクティスを使用します。

- グラフデータモデルを設計するときは、プロパティキーとユーザー向けの一時的な値の使用をできるだけ避けてください。
- データモデルに変更を加える予定の場合は、高速リセット API を使ってDBクラスターのデータをクリアするまでは、新しいモデルを使用して既存の DB クラスターにデータをロードしないでください。多くの場合、新しいモデルを使用するデータを新しい DB クラスターにロードすることが最善策です。
- 大量のデータで動作するトランザクションでは、それに応じて大きな内部ログが生成され、内部ログスペースのハイウォーターマークが永続的に増加する可能性があります。たとえば、DB クラスター内のすべてのデータを削除する 1 つのトランザクションでは、大量の内部ストレージを割り当てる必要がある巨大な内部ログを生成し、グラフデータに使用できるスペースを永続的に減らすことができます。

  これを回避するには、大きなトランザクションを小さなトランザクションに分割し、関連する内部ログが期限切れになり、後続のログで再利用できるように内部ストレージを解放します。
- Neptune クラスターボリュームの増加をモニタリングするために、 `VolumeBytesUsed` CloudWatch メトリクスに CloudWatch アラームを設定できます。これは、データがクラスターボリュームの最大サイズに達している場合に特に役立ちます。詳細については、「Amazon CloudWatch アラームの使用」を参照してください。

大量の未使用の割り当て済み領域がある場合、DB クラスターが使用するストレージスペースを縮小する唯一の方法は、グラフ内のすべてのデータを エクスポートし、それを新しい DB クラスターに再ロードすることです。DB クラスターからデータをエクスポートする簡単な方法については、Neptune のデータエクスポートサービスとユーティリティを参照してください。また、Neptune にデータをインポートする簡単な方法については、Neptune のバルクローダーをご覧ください。

> **Note**
>
> スナップショットの作成と復元はDB クラスターに割り当てられるストレージの量を減らすことはありません。スナップショットはクラスターの基盤となるストレージの元のイメージを保持するためです。使用されている割り当て済みストレージの量があまり多くない場合、

割り当てられたストレージ量を収縮する唯一の方法として、グラフデータをエクスポートし、それを新しい DB クラスターに再ロードできます。

Neptune のストレージ、信頼性、高可用性。

Amazon Neptune は、信頼性、耐久性、および耐障害性を持つように設計されています。

Neptune データの 6 つのコピーが 3 つのアベイラビリティーゾーン (AZ) にまたがって維持されるため、データのストレージの耐久性が高く、データ損失の可能性が非常に低くなります。データは、DB インスタンスが存在するかどうかにかかわらず、アベイラビリティーゾーン間で自動的にレプリケートされます。レプリケーションの数は、クラスター内の DB インスタンスの数とは関係ありません。

つまり、Neptune はグラフデータの新しいコピーを作成しないため、リードレプリカをすばやく追加できます。代わりに、リードレプリカから、すべてのデータがすでに含まれているクラスターボリュームに接続します。同様に、リードレプリカを削除しても、基になるデータは削除されません。

クラスターボリュームとそのデータは、すべての DB インスタンスを削除した後にのみ削除できます。

Neptune はまた、クラスターボリュームを構成するディスクボリュームの障害を自動的に検出します。セグメント内のデータのコピーが破損すると、Neptune はそのセグメントを直ちに修復し、同じセグメント内のデータの他のコピーを使用して、修復されたデータが最新であることを確認します。その結果、Neptune はデータ損失を回避し、ディスク障害から回復するために point-in-time 復元を実行する必要性を減らします。

Amazon Neptune エンドポイントに接続する

Amazon Neptune は、単一の DB インスタンスではなく、DB インスタンスのクラスターを使用します。各 Neptune 接続は特定の DB インスタンスで処理されます。Neptune クラスターに接続すると、指定したホスト名とポートがエンドポイントと呼ばれる中間ハンドラーをポイントします。エンドポイントは、ホストアドレスおよびポートを含む URL です。Neptune エンドポイントは、暗号化された Transport Layer Security/Secure Sockets Layer (TLS/SSL) 接続を使用します。

Neptune は、エンドポイントメカニズムを使用してこれらの接続を抽象化するため、一部の DB インスタンスが使用できない場合に、ホスト名をハードコードしたり、接続を再ルーティングするための独自のロジックを記述したりする必要がありません。

エンドポイントを使用すると、ユースケースによって各接続を対応するインスタンスまたはインスタンスグループにマッピングできます。カスタムエンドポイントを使用すると、DB インスタンスのサブセットに接続できます。次のエンドポイントは、Neptune DB クラスターから取得できます。

Neptune クラスターエンドポイント

クラスターエンドポイントとは、DB クラスターの現在のプライマリ DB インスタンスに接続する Neptune DB クラスターのエンドポイントです。Neptune DB クラスターごとに 1 つのクラスターエンドポイントと 1 つのプライマリ DB インスタンスがあります。

クラスターエンドポイントは、DB クラスターへの読み取り/書き込み接続のフェイルオーバーサポートを提供します。クラスターエンドポイントは、DB クラスターのすべての書き込みオペレーション (挿入、更新、削除、データ定義言語 (DDL) の変更など) で使用します。クラスターエンドポイントは、クエリなどの読み取りオペレーションでも使用できます。

DB クラスターの現在のプライマリ DB インスタンスが失敗した場合、Neptune は新しいプライマリ DB インスタンスに自動的にフェイルオーバーします。フェイルオーバー中、DB クラスターは、新しいプライマリ DB インスタンスからクラスターエンドポイントへの接続リクエストに継続して対応し、サービスの中断は最小限に抑えられます。

次の例では、Neptune DB クラスターのエンドポイントを示します。

```
mydbcluster.cluster-123456789012.us-east-1.neptune.amazonaws.com:8182
```

Neptune リーダーエンドポイント

読み取りエンドポイントとは、DB クラスターで利用できる Neptune レプリカの 1 つに接続する Neptune DB クラスターのエンドポイントです。各 Neptune DB クラスターに読み取りエンドポイン

トがあります。複数の Neptune レプリカがある場合、読み取りエンドポイントは各接続リクエストを Neptune レプリカのいずれかにルーティングします。

読み取りエンドポイントでは、DB クラスターへの読み取り専用接続にラウンドロビンルーティングを使用できます。リーダーエンドポイントは、 クエリなどの読み取りオペレーションで使用します。

単一インスタンスのクラスター (リードレプリカのないクラスター) がない限り、書き込みオペレーションにリーダーエンドポイントを使用することはできません。その場合のみ、読み取りオペレーションと書き込みオペレーションにリーダーを使用できます。

読み込みエンドポイントのラウンドロビンルーティングを実行するには、DNS エントリがポイントするホストを変更します。DNS を解決する度に、別の IP が取得され、それらの IP に対して接続が開かれます。接続が確立された後、その接続のリクエストはすべて、同じホストに送信されます。クライアントは新しい接続を作成し、DNS レコードを再度解決して、基本的に異なるリードレプリカへの接続を確立する必要があります。

> **Note**
>
> WebSockets 多くの場合、 接続は長期間存続します。別のリードレプリカを取得するには、以下の操作を実行します。
>
> - クライアントから接続される度に DNS エントリが解決されるようにする。
> - 接続を閉じて再接続する。

DNS を解決する方法は、クライアントのソフトウェアによってさまざまです。たとえば、クライアントで DNS を解決し、各接続にその IP を使用する場合、リクエストはすべて単一ホストにルーティングされます。

クライアントまたはプロキシの DNS キャッシュは、DNS 名をキャッシュから同じエンドポイントに解決します。これは、ラウンドロビンルーティングとフェイルオーバーシナリオ両方の問題です。

> **Note**
>
> 毎回 DNS 解決されるように、DNS キャッシュ設定は無効にします。

DB クラスターは使用可能な Neptune レプリカ間の読み取りエンドポイントに接続リクエストを分散します。DB クラスターにプライマリ DB インスタンスのみが含まれている場合、読み取りエンドポ

イントはプライマリ DB インスタンスからの接続リクエストに対応します。Neptune レプリカが DB クラスターに対して作成された場合、読み取りエンドポイントは、新しい Neptune レプリカから読み取りエンドポイントへの接続リクエストに引き続き対応し、サービスの中断は最小限に抑えられます。

Neptune DB クラスターのリーダーエンドポイントを次の例に示します。

```
mydbcluster.cluster-ro-123456789012.us-east-1.neptune.amazonaws.com:8182
```

Neptune インスタンスエンドポイント

インスタンスエンドポイントは、特定の DB インスタンスに接続する Neptune DB クラスターの DB インスタンス用のエンドポイントです。DB クラスターの各 DB インスタンスには、インスタンスタイプにかかわらず、独自のインスタンスエンドポイントがあります。したがって、DB クラスターの現在のプライマリ DB インスタンスに対して 1 つのインスタンスエンドポイントがあります。また、DB クラスターの Neptune レプリカごとに 1 つのインスタンスエンドポイントがあります。

インスタンスエンドポイントは、クラスターエンドポイントやリーダーエンドポイントの使用が適切でないシナリオ向けに、DB クラスターへの接続の直接制御を提供します。たとえば、ワークロードタイプに基づき、きめ細かいロードバランシングがアプリケーションに必要になる場合があります。この場合、DB クラスター内の別の Neptune レプリカに接続するように複数のクライアントを設定して、読み取りワークロードを分散させることができます。

次の例では、Neptune DB クラスターの DB インスタンスのインスタンスエンドポイントを示します。

```
mydbinstance.123456789012.us-east-1.neptune.amazonaws.com:8182
```

Neptune カスタムエンドポイント

Neptune クラスターのカスタムエンドポイントは、選択した DB インスタンスのセットを表します。カスタムエンドポイントに接続すると、Neptune はグループ内のいずれかのインスタンスを選択して接続を処理します。ユーザーは、このエンドポイントで参照するインスタンスを定義し、エンドポイントの用途を決めます。

Neptune DB クラスターは、カスタムエンドポイントを作成するまで存在せず、プロビジョニングされた Neptune クラスターごとに最大 5 つのカスタムエンドポイントを作成できます。

カスタムエンドポイントでは、DB インスタンスの読み取り専用機能や読み取り/書き込み機能とは異なる基準に従ってデータベース接続を負荷分散できます。エンドポイントが関連付けられているいずれの DB インスタンスも接続先となるため、グループ内のすべてのインスタンス間で同様の特性を共

有します。カスタムエンドポイントを使用する場合は、通常、そのクラスターのリーダーエンドポイントは使用しません。

この機能は、クラスター内のすべての Neptune レプリカを同一に保つことが現実的ではない特種なワークロードを扱う上級ユーザー向けです。カスタムエンドポイントを使用すると、各接続に使用される DB インスタンスの容量を調整できます。

たとえば、異なるインスタンスクラスのインスタンスグループに接続する複数のカスタムエンドポイントを定義すると、パフォーマンスニーズの異なるユーザーを、ユースケースに最適なエンドポイントに誘導できます。

次の例は、Neptune DB クラスター内の DB インスタンスのカスタムエンドポイントを示しています。

```
myendpoint.cluster-custom-123456789012.us-east-1.neptune.amazonaws.com:8182
```

詳細については、「カスタムエンドポイントの操作」を参照してください。

Neptune エンドポイントに関する考慮事項

Neptune エンドポイントを操作する場合に、次の考慮事項があります。

- インスタンスエンドポイントを使用して DB クラスターの特定の DB インスタンスに接続する前に、代わりに DB クラスターのクラスターエンドポイントまたは読み取りエンドポイントの使用を検討してください。

  クラスターエンドポイントと読み取りエンドポイントは、可用性の高いシナリオのサポートを提供します。DB クラスターのプライマリ DB インスタンスが失敗した場合、Neptune は新しいプライマリ DB インスタンスに自動的にフェイルオーバーします。そのために、既存の Neptune レプリカが新しいプライマリ DB インスタンスに昇格されるか、新しいプライマリ DB インスタンスが作成されます。フェイルオーバーが発生した場合、クラスターエンドポイントを使用して新しく昇格したプライマリインスタンスまたは作成されたプライマリ DB インスタンスに接続できます。または、読み取りエンドポイントを使用して DB クラスター内の使用可能なその他の Neptune レプリカに接続することもできます。

  この手法を使用しない場合でも、目的のオペレーションについて、DB クラスターの適切な DB インスタンスに接続することを確認できます。そのためには、DB クラスターで利用可能な DB インスタンスの結果セットをプログラムで検出し、フェイルオーバー後にインスタンスタイプを確認してから、特定の DB インスタンスのインスタンスエンドポイントを使用できます。

フェイルオーバーについての詳細は、「Neptune DB クラスターの耐障害性」を参照してください。

- 読み取りエンドポイントは Neptune DB クラスターにある利用可能な Neptune レプリカへの接続のみをルーティングします。特定のクエリはルーティングされません。

  > **Important**
  >
  > Neptune では負荷分散されません。

  クエリを負荷分散してクラスターの読み取りワークロードを配分する場合は、アプリケーションでそれを管理する必要があります。インスタンスエンドポイントを使用して Neptune レプリカに直接接続し、負荷を分散する必要があります。

- 読み込みエンドポイントのラウンドロビンルーティングを実行するには、DNS エントリがポイントするホストを変更します。クライアントは新しい接続を作成し、DNS レコードを再度解決して、基本的に新しいリードレプリカへの接続を確立する必要があります。

- フェイルオーバー中、Neptune レプリカが新しいプライマリ DB インスタンスに昇格した場合に、読み取りエンドポイントが DB クラスターの新しいプライマリ DB インスタンスに短時間直接接続する場合があります。

Neptune でのカスタムエンドポイントの操作

DB インスタンスをカスタムエンドポイントに追加したり、カスタムエンドポイントから削除したりすると、この DB インスタンスへの既存の接続はアクティブのまま残ります。

カスタムエンドポイントに含める DB インスタンスのリスト (静的リスト)、またはカスタムエンドポイントから除外するもの (除外リスト) を定義できます。包含/除外機構を使用して、DB インスタンスをグループに細分化したり、カスタムエンドポイントがクラスターのすべての DB インスタンスをカバーしていることを確認したりできます。各カスタムエンドポイントには、これらのリストタイプの 1 つのみを含めることができます。

では AWS Management Console、この選択は、このクラスターに追加された将来のインスタンスをアタッチするチェックボックスで表されます。このチェックボックスをオフのままにすると、カスタムエンドポイントはダイアログで指定された DB インスタンスのみを含む静的リストを使用します。このチェックボックスをオンにすると、カスタムエンドポイントは除外リストを使用します。この場合、カスタムエンドポイントは、ダイアログで未選択のものを除いて、クラスター内のすべての DB インスタンス (今後追加するものも含む) を表します。

フェイルオーバーや昇格に伴って DB インスタンスのロールがプライマリインスタンスと Neptune レプリカの間で変わった場合、Neptune は静的リストまたは除外リストに指定されている DB インスタンスを変更しません。

DB インスタンスは、複数のカスタムエンドポイントに関連付けることができます。たとえば、新しい DB インスタンスをクラスターに追加するとします。このような場合、DB インスタンスは該当するすべてのカスタムエンドポイントに追加されます。定義される静的リストまたは除外リストによって、追加できる DB インスタンスが決まります。

たとえば、エンドポイントに DB インスタンスの静的リストが含まれている場合、新しく追加された Neptune レプリカはここに追加されません。逆に、エンドポイントに除外リストが含まれている場合、新しく追加された Neptune レプリカは、この除外リストに名前が存在しない場合、このリストに追加されます。

Neptune レプリカは、使用不可になっても、カスタムエンドポイントに関連付けられたままになります。これは、異常であるか、停止しているか、再起動しているか、別の理由で利用できないかに関わらず当てはまります。ただし、使用できない限り、エンドポイントを介して接続することはできません。

新しく作成された Neptune クラスターにはカスタムエンドポイントがないため、これらのオブジェクトを自分で作成して管理する必要があります。カスタムエンドポイントはスナップショットに含まれないため、スナップショットから復元された Neptune クラスターにも当てはまります。カスタムエンドポイントは、復元後に再度作成します。復元したクラスターが元のクラスターと同じリージョン内にある場合は、新しいエンドポイント名を選択します。

カスタムエンドポイントの作成

Neptune コンソールを使用してカスタムエンドポイントを管理します。これを行うには、Neptune クラスターの詳細ページに移動し、のコントロールを使用します。カスタムエンドポイントセクションに追加します。

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/neptune/home で Amazon Neptune コンソールを開きます。

2. クラスターの詳細ページに移動します。
3. エンドポイントセクションで `Create custom endpoint` のアクションを選択します。
4. カスタムエンドポイントの名前を選択します。名前はユーザー ID およびリージョンで一意なものを使用します。名前の長さは 63 文字以下で、次の形式をとる必要があります。

   endpointName`.cluster-custom-`*customerDnsIdentifier*`.`*dnsSuffix*

   カスタムエンドポイント名にはクラスター名が含まれないため、クラスターの名前変更に伴って、これらの名前を変更する必要はありません。しかしながら、同じカスタムエンドポイント名を同じリージョンの複数のクラスターで再利用することはできません。カスタムエンドポイント名は、リージョン別にユーザー ID が所有するすべてのクラスター間で一意の名前にします。
5. クラスターが拡張しても変更されない DB インスタンスのリストを選択するには、[Attach future instances added to this cluster (今後追加されるインスタンスをこのクラスターにアタッチ)] をオフのままにします。このチェックボックスをオンにすると、カスタムエンドポイントは、クラスターに新しく追加されたインスタンスを動的に追加します。

カスタムエンドポイントの表示

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/neptune/home で Amazon Neptune コンソールを開きます。
2. DB クラスターのクラスターの詳細ページに移動します。
3. エンドポイントセクションには、カスタムエンドポイントに関する情報のみが含まれています (組み込みエンドポイントに関する詳細は、メインの詳細セクションに表示されます)。特定のカスタムエンドポイントに関する詳細を表示するには、その名前を選択してエンドポイントの詳細ページを表示します。

カスタムエンドポイントの編集

カスタムエンドポイントのプロパティを編集して、関連付けられている DB インスタンスを変更できます。静的リストと除外リストの間で切り替えることもできます。

編集アクションによる変更の進行中は、カスタムエンドポイントへの接続やカスタムエンドポイントの使用はできません。変更後、エンドポイントのステータスが 使用可能 に戻り、再度接続できるようになるまでに数分かかることがあります。

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/neptune/home で Amazon Neptune コンソールを開きます。

2. クラスターの詳細ページに移動します。
3. エンドポイントセクションで、編集するカスタムエンドポイントの名前を選択します。
4. そのエンドポイントの詳細ページで、編集アクションを選択します。

カスタムエンドポイントの削除

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/neptune/home で Amazon Neptune コンソールを開きます。
2. クラスターの詳細ページに移動します。
3. エンドポイント セクションで、削除するエンドポイントを選択します。
4. または、エンドポイントの詳細ページで、[Delete] (削除) アクションを選択します。

Neptune Gremlin または SPARQL クエリにカスタム ID を挿入する

デフォルトでは、Neptune はすべてのクエリに一意の queryId 値を割り当てます。この ID を使用して、実行中のクエリに関する情報を取得する (「Gremlin クエリステータス API」または「SPARQL クエリステータス API」を参照) か、キャンセル (「Gremlin クエリのキャンセル」または「SPARQL クエリのキャンセル」を参照) できます。

また、Neptune では、queryId クエリヒントを使用して、HTTP ヘッダーまたは SPARQL クエリのいずれかで、Gremlin または SPARQL クエリに独自 queryId の値を指定することもできます。独自の queryID を割り当てると、クエリを簡単に追跡してステータスの取得やキャンセルを行えます。

> **Note**
>
> この機能は、リリース 1.0.1.0.200463.0 (2019-10-15) で始めることで使用できます。

HTTP ヘッダーを使用してカスタム **queryId** 値を挿入する

Gremlin と SPARQL の両方で、HTTP ヘッダーを使用して独自の queryId 値をクエリに挿入できます。

Gremlin の例

```
curl -XPOST https://your-neptune-endpoint:port \
    -d "{\"gremlin\": \
        \"g.V().limit(1).count()\" , \
        \"queryId\":\"4d5c4fae-aa30-41cf-9e1f-91e6b7dd6f47\"  }"
```

SPARQL の例

```
curl https://your-neptune-endpoint:port/sparql \
    -d "query=SELECT * WHERE { ?s ?p ?o } " \
       --data-urlencode \
       "queryId=4d5c4fae-aa30-41cf-9e1f-91e6b7dd6f47"
```

SPARQL クエリヒントを使用してカスタム **queryId** 値を挿入する

SPARQL queryId クエリヒントを使用して SPARQL クエリにカスタム queryId 値を挿入する方法の例を次に示します。

```
curl https://your-neptune-endpoint:port/sparql \
    -d "PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#> \
       SELECT * WHERE { hint:Query hint:queryId \"4d5c4fae-
aa30-41cf-9e1f-91e6b7dd6f47\" \
       {?s ?p ?o}}"
```

queryId 値を使用してクエリステータスを確認する

Gremlin の例

```
curl https://your-neptune-endpoint:port/gremlin/status \
    -d "queryId=4d5c4fae-aa30-41cf-9e1f-91e6b7dd6f47"
```

SPARQL の例

```
curl https://your-neptune-endpoint:port/sparql/status \
    -d "queryId=4d5c4fae-aa30-41cf-9e1f-91e6b7dd6f47"
```

Neptune ラボモード

Amazon Neptune ラボモードを使用して、現在の Neptune エンジンリリースの新機能を有効にできますが、まだ本番環境で使用できず、デフォルトで有効にはできません。開発環境およびテスト環境でこれらの機能を試すことができます。

> **Note**
>
> この機能は、リリース 1.0.1.0.200463.0 (2019-10-15) で始めることで使用できます。

Neptune ラボモードの使用

neptune_lab_mode DB クラスターパラメータ 使用して、機能を有効または無効にします。これを行うには、DB クラスターパラメータグループの neptune_lab_mode パラメータの値に *(feature name)*=enabled または *(feature name)*=disabled を含めます。

たとえば、このエンジンリリースでは、neptune_lab_mode パラメータを Streams=disabled,ReadWriteConflictDetection=enabled に設定できます。

データベースの DB クラスターパラメータグループを編集する方法については、「パラメータグループを編集する」を参照してください。デフォルトの DB クラスターパラメータグループは編集できません。デフォルトのグループを使用している場合は、neptune_lab_mode パラメータを設定する前に、新しい DB クラスターパラメータグループを作成する必要があります。

> **Note**
>
> neptune_lab_mode などの静的 DB クラスターパラメータを変更した場合、変更を有効にするには、クラスターのプライマリ (ライター) インスタンスを再起動する必要があります。リリース: 1.2.0.0 (2022-07-21) より前は、DB クラスター内のすべてのリードレプリカインスタンスは、プライマリインスタンスが再起動すると自動的に再起動されていました。リリース: 1.2.0.0 (2022-07-21) 以降では、プライマリインスタンスを再起動しても、レプリカは再起動しません。つまり、DB クラスターパラメータの変更を反映するには、各インスタンスを個別に再起動する必要があります (「パラメータグループ」を参照)。

Important

現時点では、間違ったラボモードパラメータを指定したり、別の理由でリクエストが失敗した場合、その失敗が通知されないことがあります。以下に示すようにステータス APIを呼び出し、常にラボモードの変更リクエストが成功したことを確認するようにしてください。

```
curl -G https://your-neptune-endpoint:port/status
```

ステータス結果には、リクエストした変更が行われたかどうかを示すラボモード情報が含まれます。

```
{
  "status":"healthy",
  "startTime":"Wed Dec 29 02:29:24 UTC 2021",
  "dbEngineVersion":"development",
  "role":"writer",
  "dfeQueryEngine":"viaQueryHint",
  "gremlin":{"version":"tinkerpop-3.5.2"},
  "sparql":{"version":"sparql-1.1"},
  "opencypher":{"version":"Neptune-9.0.20190305-1.0"},
  "labMode":{
    "ObjectIndex":"disabled",
    "ReadWriteConflictDetection":"enabled"
  },
  "features":{
    "LookupCache":{"status":"Available"},
    "ResultCache":{"status":"disabled"},
    "IAMAuthentication":"disabled",
    "Streams":"disabled",
    "AuditLog":"disabled"
  },
  "settings":{"clusterQueryTimeoutInMs":"120000"}
}
```

現在、ラボモードで以下の機能にアクセスできます。

OSGP インデックス

Neptune は、4 番目のインデックスとして OSGP インデックスを保持できるようになりました。これは、多数の述語を持つデータセットに役立つインデックスです (OSGP インデックスの有効化 を参照してください)。

> **Note**
>
> この機能は、Neptune エンジンリリース 1.0.2.1 からアクセスできます。

新しい空の Neptune DB クラスターで OSGP インデックスを有効にするには、`neptune_lab_mode` DB クラスターのパラメータで `ObjectIndex=enabled` を設定します。OSGP インデックスは、新しい空の DB クラスターでのみ有効にできます。

デフォルトでは、OSGP インデックスは無効になっています。

> **Note**
>
> OSGP インデックスを有効にするように `neptune_lab_mode` DB クラスターパラメータを設定したら、変更を有効にするためにクラスターのライターインスタンスを再起動する必要があります。

> **Warning**
>
> `ObjectIndex=disabled` を設定することによって、有効な OSGP インデックスを無効にした後、データを追加してから再び有効にした場合、インデックスは正しく構築されません。インデックスのオンデマンド再構築はサポートされていないため、データベースが空の場合にのみ OSGP インデックスを有効にしてください。

正式なトランザクションセマンティクス

Neptune は、同時トランザクションの正式なセマンティクスを更新しました (Neptune でのトランザクションセマンティクス を参照)。

形式化されたトランザクションセマンティクスを有効または無効にする `neptune_lab_mode` パラメータの名前として `ReadWriteConflictDetection` を使用します。

デフォルトでは、形式化されたトランザクションセマンティクスはすでに有効になっています。以前の動作に戻す場合は、DB Cluster neptune_lab_mode パラメータの値に ReadWriteConflictDetection=disabled を含めます。

日時サポートの延長

Neptune は日時機能のサポートを拡張しました。拡張形式で日時を有効にするには、DB クラスターneptune_lab_modeパラメータに設定された値DatetimeMillisecond=enabledに を含めます。

Amazon Neptune 代替クエリエンジン (DFE)

Amazon Neptune には、元のエンジンよりも効率的に CPU コア、メモリ、I/O などの DB インスタンスリソースを使用する DFE と呼ばれる代替クエリエンジンがあります。

> **Note**
>
> データセットが大きいと、T3 インスタンスでは DFE エンジンが適切に動作しない場合があります。

DFE エンジンは SPARQL、Gremlin、および openCypher クエリを実行し、左ディープ、ブッシー、ハイブリッドなど、多種多様なプランタイプをサポートします。　プランオペレータは、予約済みのコンピューティングコアのセットで実行されるコンピューティング操作と、I/O スレッドプール内の独自のスレッドで実行される I/O 操作の両方を呼び出すことができます。

DFE は、Neptune グラフデータに関する事前生成された統計情報を使用して、クエリの構造化方法について情報に基づいた判断を下します。　これらの統計の生成方法については、「DFE 統計」を参照してください。

プランタイプと使用されるコンピューティングスレッドの数は、事前に生成された統計と Neptune ヘッドノードで使用可能なリソースに基づいて自動的に選択されます。結果の順序は、内部計算の並列性を持つプランでは事前に決められていません。

Neptune DFE エンジンの使用場所の制御

デフォルトでは、インスタンスの neptune_dfe_query_engine インスタンスパラメータは `viaQueryHint` に設定されます。これにより、DFE エンジンは openCypher クエリと、`useDFE` クエリヒントが `true` に明示的に設定された Gremlin および SPARQL クエリにのみ使用されます。

`neptune_dfe_query_engine` インスタンスパラメータを `enabled` に設定することにより、DFE エンジンを完全に有効にして、可能な限りどこでも使用できるようにすることができます。

特定の Gremlin クエリまたは SPARQL クエリの `useDFE` クエリヒントを含めることにより、DFE を無効にすることもできます。このクエリヒントを使用すると、DFE がその特定のクエリを実行しないようにできます。

インスタンスで DFE が有効になっているかどうかは、次のように インスタンスのステータス 呼び出しを使用して調べることができます。

```
curl -G https://your-neptune-endpoint:port/status
```

次に、ステータス応答は DFE が有効かどうかを指定します。

```
{
  "status":"healthy",
  "startTime":"Wed Dec 29 02:29:24 UTC 2021",
  "dbEngineVersion":"development",
  "role":"writer",
  "dfeQueryEngine":"viaQueryHint",
  "gremlin":{"version":"tinkerpop-3.5.2"},
  "sparql":{"version":"sparql-1.1"},
  "opencypher":{"version":"Neptune-9.0.20190305-1.0"},
  "labMode":{
    "ObjectIndex":"disabled",
    "ReadWriteConflictDetection":"enabled"
  },
  "features":{
    "ResultCache":{"status":"disabled"},
    "IAMAuthentication":"disabled",
    "Streams":"disabled",
    "AuditLog":"disabled"
  },
  "settings":{"clusterQueryTimeoutInMs":"120000"}
}
```

Gremlin `explain` および `profile` 結果は、クエリが DFE によって実行されているかどうかを示します。 `explain` については Gremlin explain レポートに含まれる情報 を、`profile` については DFE profile レポート を参照してください。

同様に SPARQL `explain` は、SPARQL クエリが DFE によって実行されているかどうかを示します。 詳細については、DFE が無効の場合の SPARQL explain 出力の例 および DFENodeoperator を参照してください。

Neptune DFE でサポートされているクエリコンストラクト

現在、Neptune DFE は SPARQL および Gremlin クエリコンストラクトのサブセットをサポートしています。

SPARQL の場合、これは結合サブセットの基本的なグラフパターンです。

Gremlin では、一般に、より複雑なステップを含まない一連のトラバーサルを含むクエリのサブセットです。

次のように、DFE によってクエリの 1 つが全体または一部で実行されているかどうかを確認できます。

- Gremlin では `explain` および `profile` 結果は、DFE によって実行されているクエリのどの部分 (存在する場合) かを示します。 `explain` については Gremlin explain レポートに含まれる情報 を、`profile` については DFE profile レポート を参照してください。 また、explainおよびprofileを使用した Gremlin クエリのチューニング も参照してください。

  個々の Gremlin ステップに対する Neptune エンジンのサポートの詳細については、Gremlin ステップサポート を参照してください。
- 同様に SPARQL `explain` は、SPARQL クエリが DFE によって実行されているかどうかを示します。 詳細については、DFE が無効の場合の SPARQL explain 出力の例 および DFENodeoperator を参照してください。

Neptune DFE が使用する統計情報の管理

> **Note**
>
> Neptune での openCypher のサポートは、DFE クエリエンジンによって異なります。
> DFE エンジンは、まず、Neptune エンジンリリース 1.0.3.0 でラボモードで使用可能になり、Neptune エンジンリリース 1.0.5.0 から、デフォルトで有効になりましたが、クエリヒントと openCypher サポートでのみ使用されます。
> Neptune エンジンリリース 1.1.1.0 以降、DFE エンジンはラボモードではなくなり、インスタンスの DB パラメータグループの neptune_dfe_query_engine インスタンスパラメータを使用して制御されるようになりました。

DFE エンジンは、Neptune グラフのデータに関する情報を使用して、クエリの実行を計画する際に効果的なトレードオフを行います。この情報は、クエリ計画を導くことができる、いわゆる特性セットと述語統計を含む統計の形式をとります。

エンジンリリース 1.2.1.0 以降、概要 API またはsummaryエンドポイントを使用して、これらの統計からグラフに関するGetGraph概要情報を取得できます。

これらの DFE 統計は、現在、グラフ内のデータの 10% 以上が変更された場合、または最新の統計が 10 日以上経過した場合に再生成されます。ただし、これらのトリガーは将来、変更される可能性があります。

> **Note**
>
> 統計の生成は、T3 および T4g インスタンスでは無効です。それらのインスタンスタイプのメモリ容量を超える可能性があるためです。

DFE 統計情報の生成は、次のエンドポイントのいずれかを使用して管理できます。

- `https://your-neptune-host:port/rdf/statistics` (SPARQL の場合)
- `https://your-neptune-host:port/propertygraph/statistics` (Gremlin と openCypher 用)、およびその代替バージョン:`https://your-neptune-host:port/pg/statistics`。

> **Note**
>
> エンジンリリース 1.1.1.0 の時点で、Gremlin 統計エンドポイント (`https://your-neptune-host:port/gremlin/statistics`) は廃止され、propertygraph または pg エンドポイントが優先されます。下位互換性のために引き続きサポートされていますが、将来のリリースで削除される可能性があります。
> エンジンリリース 1.2.1.0 の時点で、SPARQL 統計エンドポイント (`https://your-neptune-host:port/sparql/statistics`) は廃止され、rdf エンドポイントが優先されます。下位互換性のために引き続きサポートされていますが、将来のリリースで削除される可能性があります。

以下の例では、$STATISTICS_ENDPOINT は、これらのエンドポイント URL のいずれかを表します。

> **Note**
>
> DFE 統計エンドポイントがリーダーインスタンス上にある場合、処理できるリクエストはステータスリクエストです。その他のリクエストは、ReadOnlyViolationException で失敗します。

DFE 統計生成のサイズ制限

現在、DFE 統計情報の生成は、次のいずれかのサイズ制限に達すると停止します。

- 生成する特性セットの数は 50,000 を超えることはできません。
- 生成する述語統計の数は 100 万を超えることはできません。

これらの制限は変更される可能性があります。

DFE 統計情報の現在のステータス

以下の curl リクエストを使用して DFE 統計の現在のステータスを確認できます。

```
curl -G "$STATISTICS_ENDPOINT"
```

ステータスリクエストに対する応答には、以下のフィールドが含まれています。

- `status`— リクエストの HTTP リターンコード。リクエストが成功した場合、コードは `200` です。共通エラーリストについては、一般的なエラー を参照してください。
- `payload`:
  - `autoCompute`— (ブール値) 統計情報の自動生成が有効になっているかどうかを示します。
  - `active`— (ブール値) DFE 統計情報の生成が有効になっているかどうかを示します。
  - `statisticsId` — 現在の統計生成の実行の ID を報告します。 `-1` の値は統計が生成されていないことを示します。
  - `date` — DFE 統計が最近生成された UTC 時刻 (ISO 8601 形式)。

    > **Note**
    >
    > エンジンリリース 1.2.1.0 より前は、これは分単位の精度で表されていましたが、エンジンリリース 1.2.1.0 以降は、ミリ秒の精度で表されます (例: `2023-01-24T00:47:43.319Z`)。

  - `note`— 統計情報が無効な場合の問題に関するメモ。
  - `signatureInfo` — 統計で生成された特性セットに関する情報を含みます (エンジンリリース 1.2.1.0 より前は、このフィールドは `summary` という名前でした)。これらは通常、直接的に実行できるものではありません。
    - `signatureCount`— すべての特性セットにおけるシグニチャの総数。
    - `instanceCount`— 特性セットインスタンスの合計数。
    - `predicateCount`— 一意の述語の合計数。

統計が生成されていない場合のステータスリクエストに対する応答は、次のようになります。

```
{
  "status" : "200 OK",
  "payload" : {
    "autoCompute" : true,
    "active" : false,
    "statisticsId" : -1
   }
}
```

DFE 統計が使用可能な場合、応答は次のようになります。

```
{
  "status" : "200 OK",
  "payload" : {
    "autoCompute" : true,
    "active" : true,
    "statisticsId" : 1588893232718,
    "date" : "2020-05-07T23:13Z",
    "summary" : {
      "signatureCount" : 5,
      "instanceCount" : 1000,
      "predicateCount" : 20
    }
  }
}
```

DFE 統計情報の生成が失敗した場合、たとえば、統計のサイズの制限の場合、レスポンスは次のようになります。

```
{
  "status" : "200 OK",
  "payload" : {
    "autoCompute" : true,
    "active" : false,
    "statisticsId" : 1588713528304,
    "date" : "2020-05-05T21:18Z",
    "note" : "Limit reached: Statistics are not available"
  }
}
```

DFE 統計情報の自動生成の無効化

デフォルトでは、DFE を有効にすると、DFE 統計情報の自動生成が有効になります。

自動生成は、次のように無効にできます。

```
curl -X POST "$STATISTICS_ENDPOINT" -d '{ "mode" : "disableAutoCompute" }'
```

リクエストが成功すると、HTTP レスポンスコードは 200 となり、レスポンスは次のとおりです。

```
{
```

```
  "status" : "200 OK"
}
```

自動生成が無効になっていることを確認するには、ステータスリクエストを発行し、レスポンス内で autoCompute フィールドが false に設定されていることを確認します。

統計の自動生成を無効にしても、進行中の統計計算は終了しません。

DB クラスターのライターインスタンスではなくリーダーインスタンスに対して自動生成を無効にするリクエストを行うと、HTTP リターンコード 400 でリクエストが失敗し、次のような出力が行われます。

```
{
  "detailedMessage" : "Writes are not permitted on a read replica instance",
  "code" : "ReadOnlyViolationException",
  "requestId":"8eb8d3e5-0996-4a1b-616a-74e0ec32d5f7"
}
```

他の共通エラーリストについては、一般的なエラーを参照してください。

DFE 統計情報の自動生成の再有効化

デフォルトでは、DFE を有効にすると、DFE 統計情報の自動生成がすでに有効になっています。自動生成を無効にすると、後で次のようにして再度有効にすることができます。

```
curl -X POST "$STATISTICS_ENDPOINT" -d '{ "mode" : "enableAutoCompute" }'
```

リクエストが成功すると、HTTP レスポンスコードは 200 となり、レスポンスは次のとおりです。

```
{
  "status" : "200 OK"
}
```

自動生成が無効になっていることを確認するには、ステータスリクエストと確認してレスポンス内で autoCompute フィールドが true に設定されていることを確認します。

DFE 統計情報の生成を手動でトリガーする

DFE 統計情報の生成は、次のように手動で開始できます。

```
curl -X POST "$STATISTICS_ENDPOINT" -d '{ "mode" : "refresh" }'
```

リクエストが成功した場合、出力は次のようになります。HTTP リターンコードは 200 です。

```
{
  "status" : "200 OK",
  "payload" : {
    "statisticsId" : 1588893232718
  }
}
```

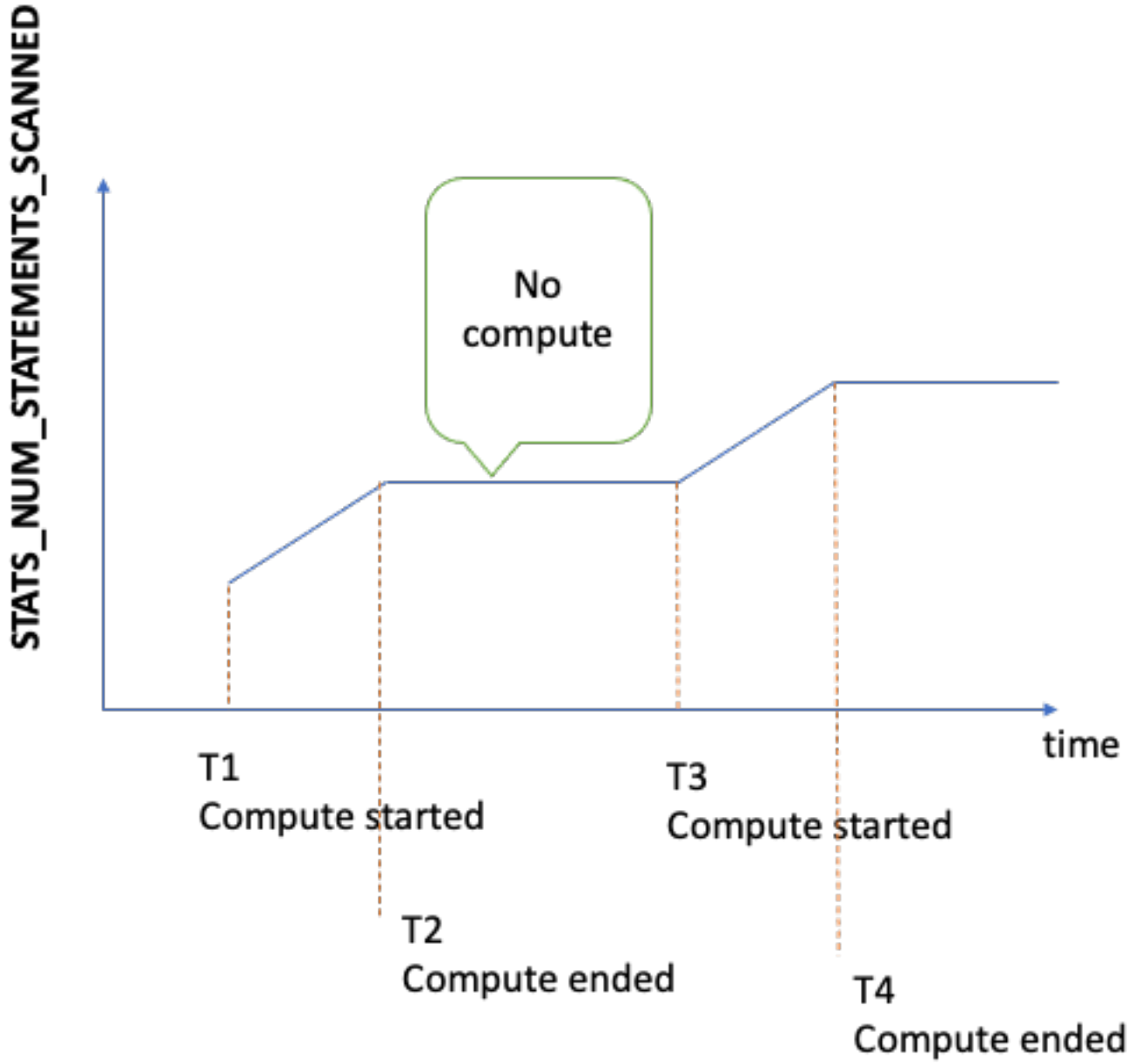
出力の statisticsId には、現在発生している統計生成の実行の ID が表示されます。リクエストの時点で実行がすでに処理中だった場合、リクエストは新しい実行を開始するのではなく、その実行の ID を返します。一度に実行できる統計生成は 1 つのみです。

DFE 統計の生成中にフェイルオーバーが発生した場合、新しいライターノードは最後に処理されたチェックポイントを取得し、そこから統計の実行を再開します。

StatsNumStatementsScanned CloudWatch メトリクスを使用した統計計算のモニタリング

StatsNumStatementsScanned CloudWatch メトリクスは、サーバーの開始以降に統計計算のためにスキャンされたステートメントの総数を返します。これは、各統計計算スライスで更新されます。

統計計算がトリガーされるたびに、この数は増加し、計算が行われていないときは一定のままです。時間の経過に伴いStatsNumStatementsScannedのプロットを見ると、統計計算がいつ行われたか、どのくらいの速さであったかがはっきりわかります。



計算が行われているとき、グラフの傾きはどのくらい速いかを示します (傾斜が急であるほど、より速い統計計算となります) 。

グラフが単に 0 のフラットラインである場合、統計機能は有効になっていますが、統計情報はまったく計算されていません。統計機能が無効になっている場合、または統計計算をサポートしていないエンジンバージョンを使用している場合は、`StatsNumStatementsScanned` は存在しません。

前述のように、統計情報 API を使用して統計情報の計算を無効にすることはできますが、オフにすると、統計が最新にならず、DFE エンジンのクエリプランの生成が不十分になる可能性があります。

の使用方法については、Amazon を使用した Neptune のモニタリング CloudWatch「」を参照してください CloudWatch。

DFE 統計エンドポイントでの AWS Identity and Access Management (IAM) 認証の使用

awscurl または、HTTP および IAM で動作するその他のツールを使用して、DFE 統計エンドポイントに IAM 認証で安全にアクセスできます。適切な認証情報を設定する方法については、「一時的な認証情報で awscurl を使用して、IAM 認証が有効になっている DB クラスターに安全に接続する」を参照してください。設定が完了したら、次のようなステータスリクエストを行うことができます。

```
awscurl "$STATISTICS_ENDPOINT" \
    --region (your region) \
    --service neptune-db
```

または、たとえば、以下を含む `request.json` という名前の JSON ファイルを作成できます。

```
{ "mode" : "refresh" }
```

その後、次のように統計情報の生成を手動で開始できます。

```
awscurl "$STATISTICS_ENDPOINT" \
    --region (your region) \
    --service neptune-db \
    -X POST -d @request.json
```

DFE 統計の削除

統計エンドポイントに対して HTTP DELETE 要求を実行することで、データベース内のすべての統計情報を削除できます。

```
curl -X "DELETE" "$STATISTICS_ENDPOINT"
```

有効な HTTP リターンコードは次のとおりです。

- `200`— 削除に成功しました。

  この場合、典型的な応答は次のようになります。

```
{
  "status" : "200 OK",
  "payload" : {
      "active" : false,
      "statisticsId" : -1
  }
}
```

- 204— 削除する統計情報はありませんでした。

  この場合、応答は空白 (応答なし) です。

リーダーノードの統計エンドポイントに削除リクエストを送信すると、ReadOnlyViolationException がスローされます。

DFE 統計リクエストの一般的なエラーコード

以下に、統計エンドポイントに対してリクエストを行うときに発生する可能性のある一般的なエラーのリストを示します。

- AccessDeniedException – リターンコード: 400。メッセージ: Missing Authentication Token。
- BadRequestException (Gremlin と openCypher の場合) — リターンコード: 400。メッセージ: Bad route: /pg/statistics。
- BadRequestException (RDF データの場合) — リターンコード: 400。メッセージ: Bad route: /rdf/statistics。
- InvalidParameterException – リターンコード: 400。メッセージ: Statistics command parameter 'mode' has unsupported value '*the invalid value*'。
- MissingParameterException – リターンコード: 400。メッセージ: Content-type header not specified.。
- ReadOnlyViolationException – リターンコード: 400。メッセージ: Writes are not permitted on a read replica instance。

たとえば、DFE および統計情報が有効でないときにリクエストを行った場合、次のようなレスポンスを受信します。

```
{
```

```
  "code" : "BadRequestException",
  "requestId" : "b2b8f8ee-18f1-e164-49ea-836381a3e174",
  "detailedMessage" : "Bad route: /sparql/statistics"
}
```

グラフに関する簡単なサマリーレポートを取得する

Neptune グラフサマリー API は、グラフに関する以下の情報を取得します。

- プロパティ (PG) グラフの場合、グラフサマリー API は、ノードおよびエッジラベルとプロパティキーの読み取り専用リストを、ノード、エッジ、プロパティの数とともに返します。
- リソース記述フレームワーク (RDF) グラフの場合、グラフサマリー API は、クラスと述語キーの読み取り専用リストを、クワッド、主語、述語の数とともに返します。

> **Note**
>
> グラフサマリー API は、Neptune エンジンリリース 1.2.1.0 で導入されました。

グラフサマリー API を使用すると、グラフのデータサイズと内容を大まかに把握できます。また、`%summary` Neptune Workbench マジックを使用して、Neptune ノートブック内でインタラクティブに API を使用することもできます。グラフアプリケーションでは、API を使用して、検出されたノードやエッジのラベルを検索の一部として指定することで、検索結果を改善できます。

グラフサマリーデータは、実行時に Neptune DFE エンジンによって計算された DFE 統計から抽出され、DFE 統計が利用可能な場合はいつでも利用できます。Neptune DB クラスターを新規作成すると、統計がデフォルトで有効になります。

> **Note**
>
> メモリを節約するため、`t3` および `t4` インスタンスタイプ (つまり、`db.t3.medium` および `db.t4g.medium` インスタンスタイプ) では、統計の生成が無効になっています。そのため、どちらのインスタンスタイプでもグラフサマリーデータは使用できません。

DFE 統計のステータスは、統計ステータス API を使用して確認できます。統計の自動生成が無効になっていない限り、統計は定期的に自動更新されます。

グラフサマリーをリクエストするときに統計をできるだけ最新の状態に保ちたい場合は、サマリーを取得する直前に、手動で統計の更新をトリガーできます。統計の計算中にグラフが変化する場合、必然的に少し遅れますが、それほど遅れることはありません。

グラフサマリー API を使用してグラフの概要情報を取得する

Gremlin または openCypher を使用してクエリを行うプロパティグラフの場合、プロパティグラフのサマリーエンドポイントからグラフの概要を取得できます。このエンドポイントには長い URI と短い URI の両方があります。

- `https://your-neptune-host:port/propertygraph/statistics/summary`
- `https://your-neptune-host:port/pg/statistics/summary`

SPARQL を使用してクエリを行う RDF グラフの場合、RDF サマリーエンドポイントからグラフの概要を取得できます。

- `https://your-neptune-host:port/rdf/statistics/summary`

これらのエンドポイントは読み取り専用であり、HTTP GET 操作のみをサポートします。$GRAPH_SUMMARY_ENDPOINT がクエリしたいエンドポイントのアドレスに設定されている場合、次のように curl と HTTP GET を使用してサマリーデータを取得できます。

```
curl -G "$GRAPH_SUMMARY_ENDPOINT"
```

グラフの概要を取得しようとしたときに、使用できる統計がない場合、応答は次のようになります。

```
{
  "detailedMessage": "Statistics are not available. Summary can only be generated after
 statistics are available.",
  "requestId": "48c1f788-f80b-b69c-d728-3f6df579a5f6",
  "code": "StatisticsNotAvailableException"
}
```

グラフサマリー API の **mode** URL クエリパラメータ

グラフサマリー API は、mode という名前の URL クエリパラメータを受け付けます。このパラメータには、basic (デフォルト) と detailed の 2 つの値のいずれかを指定できます。RDF グラフの場合、detailed モードグラフサマリーレスポンスには追加の subjectStructures フィールドが含まれます。プロパティグラフの場合、詳細グラフサマリーレスポンスには、nodeStructures と edgeStructures という 2 つの追加フィールドが含まれます。

detailed グラフサマリーレスポンスをリクエストするには、以下の mode パラメータを含めてください。

```
curl -G "$GRAPH_SUMMARY_ENDPOINT?mode=detailed"
```

mode パラメータが存在しない場合、デフォルトで basic モードが使用されるため、明示的に ?mode=basic を指定することも可能ですが、必須ではありません。

プロパティグラフ (PG) のグラフサマリーレスポンス

空のプロパティグラフの場合、詳細なグラフサマリーレスポンスは次のようになります。

```
{
  "status" : "200 OK",
  "payload" : {
    "version" : "v1",
    "lastStatisticsComputationTime" : "2023-01-10T07:58:47.972Z",
    "graphSummary" : {
      "numNodes" : 0,
      "numEdges" : 0,
      "numNodeLabels" : 0,
      "numEdgeLabels" : 0,
      "nodeLabels" : [ ],
      "edgeLabels" : [ ],
      "numNodeProperties" : 0,
      "numEdgeProperties" : 0,
      "nodeProperties" : [ ],
      "edgeProperties" : [ ],
      "totalNodePropertyValues" : 0,
      "totalEdgePropertyValues" : 0,
      "nodeStructures" : [ ],
      "edgeStructures" : [ ]
    }
  }
}
```

プロパティグラフ (PG) サマリーレスポンスには以下のフィールドがあります。

- **status** — リクエストの HTTP リターンコード。リクエストが成功した場合、コードは 200 です。

  共通エラーリストについては、一般的なグラフサマリーエラー を参照してください。

- **payload**
  - **version** - このグラフサマリーレスポンスのバージョン。
  - **lastStatisticsComputationTime** - Neptune が最後に統計を計算した時刻のタイムスタンプ (ISO 8601 形式)。
  - **graphSummary**
    - **numNodes** - グラフ内のノードの数。
    - **numEdges** - グラフ内のエッジの数。
    - **numNodeLabels** - グラフ内の異なるノードラベルの数。
    - **numEdgeLabels** - グラフ内の異なるエッジラベルの数。
    - **nodeLabels** - グラフ内の異なるノードラベルのリスト。
    - **edgeLabels** - グラフ内の異なるエッジラベルのリスト。
    - **numNodeProperties** - グラフ内の異なるノードプロパティの数。
    - **numEdgeProperties** - グラフ内の異なるエッジプロパティの数。
    - **nodeProperties** - グラフ内の異なるノードプロパティのリストと、各プロパティが使用されるノードの数。
    - **edgeProperties** - グラフ内の異なるエッジプロパティのリストと、各プロパティが使用される エッジの数。
    - **totalNodePropertyValues** - すべてのノードプロパティの使用回数の合計。
    - **totalEdgePropertyValues** - すべてのエッジプロパティの使用回数の合計。
    - **nodeStructures** — このフィールドは、リクエストで *mode=detailed* が指定されている場合にのみ表示されます。ノード構造のリストを含み、各ノード構造は以下のフィールドを含みます。
      - **count** - この特定の構造を持つノードの数。
      - **nodeProperties** - この特定の構造に存在するノードプロパティのリスト。
      - **distinctOutgoingEdgeLabels** - この特定の構造に存在する個別の出力エッジラベルのリスト。
    - **edgeStructures** — このフィールドは、リクエストで *mode=detailed* が指定されている場合にのみ表示されます。エッジ構造のリストを含み、各エッジ構造は以下のフィールドを含みます。
      - **count** - この特定の構造を持つエッジの数。
      - **edgeProperties** - この特定の構造に存在するエッジプロパティのリスト。

RDF グラフのグラフサマリーレスポンス

空の RDF グラフの場合、詳細なグラフサマリーレスポンスは次のようになります。

```
{
  "status" : "200 OK",
  "payload" : {
    "version" : "v1",
    "lastStatisticsComputationTime" : "2023-01-10T07:58:47.972Z",
    "graphSummary" : {
      "numDistinctSubjects" : 0,
      "numDistinctPredicates" : 0,
      "numQuads" : 0,
      "numClasses" : 0,
      "classes" : [ ],
      "predicates" : [ ],
      "subjectStructures" : [ ]
    }
  }
}
```

RDF グラフサマリーレスポンスには以下のフィールドがあります。

- **`status`** — リクエストの HTTP リターンコード。リクエストが成功した場合、コードは 200 です。

  共通エラーリストについては、一般的なグラフサマリーエラー を参照してください。
- **`payload`**
  - **`version`** - このグラフサマリーレスポンスのバージョン。
  - **`lastStatisticsComputationTime`** - Neptune が最後に統計を計算した時刻のタイムスタンプ (ISO 8601 形式)。
  - **`graphSummary`**
    - **`numDistinctSubjects`** — グラフ内の異なるサブジェクトの数。
    - **`numDistinctPredicates`** — グラフ内の異なる述語の数。
    - **`numQuads`** - グラフ内のクワッドの数。
    - **`numClasses`** - グラフ内のクラスの数。
    - **`classes`** — グラフ内のクラスのリスト。
    - **`predicates`** — グラフ内の述語のリストと述語の数。

- **subjectStructures** — このフィールドは、リクエストで *mode=detailed* が指定されている場合にのみ表示されます。サブジェクト構造のリストを含み、各構造は以下のフィールドを含みます。
  - **count** - この特定の構造の出現回数。
  - **predicates** - この特定の構造に存在する述語のリスト。

サンプルのプロパティグラフ (PG) サマリーレスポンス

以下は、サンプルプロパティグラフの航空路データセットを含むプロパティグラフの詳細なサマリーレスポンスです。


```
{
  "status" : "200 OK",
  "payload" : {
    "version" : "v1",
    "lastStatisticsComputationTime" : "2023-03-01T14:35:03.804Z",
    "graphSummary" : {
      "numNodes" : 3748,
      "numEdges" : 51300,
      "numNodeLabels" : 4,
      "numEdgeLabels" : 2,
      "nodeLabels" : [
        "continent",
        "country",
        "version",
        "airport"
      ],
      "edgeLabels" : [
        "contains",
        "route"
      ],
      "numNodeProperties" : 14,
      "numEdgeProperties" : 1,
      "nodeProperties" : [
        {
          "desc" : 3748
        },
        {
          "code" : 3748
        },
        {
```

```
          "type" : 3748
        },
        {
          "country" : 3503
        },
        {
          "longest" : 3503
        },
        {
          "city" : 3503
        },
        {
          "lon" : 3503
        },
        {
          "elev" : 3503
        },
        {
          "icao" : 3503
        },
        {
          "region" : 3503
        },
        {
          "runways" : 3503
        },
        {
          "lat" : 3503
        },
        {
          "date" : 1
        },
        {
          "author" : 1
        }
      ],
      "edgeProperties" : [
        {
          "dist" : 50532
        }
      ],
      "totalNodePropertyValues" : 42773,
      "totalEdgePropertyValues" : 50532,
      "nodeStructures" : [
```

```
        {
          "count" : 3471,
          "nodeProperties" : [
            "city",
            "code",
            "country",
            "desc",
            "elev",
            "icao",
            "lat",
            "lon",
            "longest",
            "region",
            "runways",
            "type"
          ],
          "distinctOutgoingEdgeLabels" : [
            "route"
          ]
        },
        {
          "count" : 161,
          "nodeProperties" : [
            "code",
            "desc",
            "type"
          ],
          "distinctOutgoingEdgeLabels" : [
            "contains"
          ]
        },
        {
          "count" : 83,
          "nodeProperties" : [
            "code",
            "desc",
            "type"
          ],
          "distinctOutgoingEdgeLabels" : [ ]
        },
        {
          "count" : 32,
          "nodeProperties" : [
            "city",
```

```
            "code",
            "country",
            "desc",
            "elev",
            "icao",
            "lat",
            "lon",
            "longest",
            "region",
            "runways",
            "type"
          ],
          "distinctOutgoingEdgeLabels" : [ ]
        },
        {
          "count" : 1,
          "nodeProperties" : [
            "author",
            "code",
            "date",
            "desc",
            "type"
          ],
          "distinctOutgoingEdgeLabels" : [ ]
        }
      ],
      "edgeStructures" : [
        {
          "count" : 50532,
          "edgeProperties" : [
            "dist"
          ]
        }
      ]
    }
  }
}
```

サンプルの RDF グラフサマリーレスポンス

以下は、サンプルの RDF 航空路データセットを含む RDF グラフの詳細なサマリーレスポンスです。

```
{
  "status" : "200 OK",
  "payload" : {
    "version" : "v1",
    "lastStatisticsComputationTime" : "2023-03-01T14:54:13.903Z",
    "graphSummary" : {
      "numDistinctSubjects" : 54403,
      "numDistinctPredicates" : 19,
      "numQuads" : 158571,
      "numClasses" : 4,
      "classes" : [
        "http://kelvinlawrence.net/air-routes/class/Version",
        "http://kelvinlawrence.net/air-routes/class/Airport",
        "http://kelvinlawrence.net/air-routes/class/Continent",
        "http://kelvinlawrence.net/air-routes/class/Country"
      ],
      "predicates" : [
        {
          "http://kelvinlawrence.net/air-routes/objectProperty/route" : 50656
        },
        {
          "http://kelvinlawrence.net/air-routes/datatypeProperty/dist" : 50656
        },
        {
          "http://kelvinlawrence.net/air-routes/objectProperty/contains" : 7004
        },
        {
          "http://kelvinlawrence.net/air-routes/datatypeProperty/code" : 3747
        },
        {
          "http://www.w3.org/2000/01/rdf-schema#label" : 3747
        },
        {
          "http://kelvinlawrence.net/air-routes/datatypeProperty/type" : 3747
        },
        {
          "http://kelvinlawrence.net/air-routes/datatypeProperty/desc" : 3747
        },
        {
          "http://www.w3.org/1999/02/22-rdf-syntax-ns#type" : 3747
        },
        {
          "http://kelvinlawrence.net/air-routes/datatypeProperty/icao" : 3502
```

```
        },
        {
          "http://kelvinlawrence.net/air-routes/datatypeProperty/lat" : 3502
        },
        {
          "http://kelvinlawrence.net/air-routes/datatypeProperty/region" : 3502
        },
        {
          "http://kelvinlawrence.net/air-routes/datatypeProperty/runways" : 3502
        },
        {
          "http://kelvinlawrence.net/air-routes/datatypeProperty/longest" : 3502
        },
        {
          "http://kelvinlawrence.net/air-routes/datatypeProperty/elev" : 3502
        },
        {
          "http://kelvinlawrence.net/air-routes/datatypeProperty/lon" : 3502
        },
        {
          "http://kelvinlawrence.net/air-routes/datatypeProperty/country" : 3502
        },
        {
          "http://kelvinlawrence.net/air-routes/datatypeProperty/city" : 3502
        },
        {
          "http://kelvinlawrence.net/air-routes/datatypeProperty/author" : 1
        },
        {
          "http://kelvinlawrence.net/air-routes/datatypeProperty/date" : 1
        }
      ],
      "subjectStructures" : [
        {
          "count" : 50656,
          "predicates" : [
            "http://kelvinlawrence.net/air-routes/datatypeProperty/dist"
          ]
        },
        {
          "count" : 3471,
          "predicates" : [
            "http://kelvinlawrence.net/air-routes/datatypeProperty/city",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/code",
```

```
            "http://kelvinlawrence.net/air-routes/datatypeProperty/country",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/desc",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/elev",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/icao",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/lat",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/lon",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/longest",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/region",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/runways",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/type",
            "http://kelvinlawrence.net/air-routes/objectProperty/route",
            "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
            "http://www.w3.org/2000/01/rdf-schema#label"
          ]
        },
        {
          "count" : 238,
          "predicates" : [
            "http://kelvinlawrence.net/air-routes/datatypeProperty/code",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/desc",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/type",
            "http://kelvinlawrence.net/air-routes/objectProperty/contains",
            "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
            "http://www.w3.org/2000/01/rdf-schema#label"
          ]
        },
        {
          "count" : 31,
          "predicates" : [
            "http://kelvinlawrence.net/air-routes/datatypeProperty/city",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/code",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/country",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/desc",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/elev",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/icao",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/lat",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/lon",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/longest",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/region",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/runways",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/type",
            "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
            "http://www.w3.org/2000/01/rdf-schema#label"
          ]
```

```
        },
        {
          "count" : 6,
          "predicates" : [
            "http://kelvinlawrence.net/air-routes/datatypeProperty/code",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/desc",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/type",
            "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
            "http://www.w3.org/2000/01/rdf-schema#label"
          ]
        },
        {
          "count" : 1,
          "predicates" : [
            "http://kelvinlawrence.net/air-routes/datatypeProperty/author",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/code",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/date",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/desc",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/type",
            "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
            "http://www.w3.org/2000/01/rdf-schema#label"
          ]
        }
      ]
    }
  }
}
```

グラフサマリーエンドポイントでの AWS Identity and Access Management (IAM) 認証の使用

グラフサマリーエンドポイントには、awscurl または、HTTP および IAM で動作するその他のツールを使用して IAM 認証で安全にアクセスできます。適切な認証情報を設定する方法については、「一時的な認証情報で awscurl を使用して、IAM 認証が有効になっている DB クラスターに安全に接続する」を参照してください。設定が完了したら、次のようなリクエストを行うことができます。

```
awscurl "$GRAPH_SUMMARY_ENDPOINT" \
    --region (your region) \
    --service neptune-db
```

> **Important**
>
> 一時的な認証情報を作成する IAM ID またはロールには、GetGraphサマリー IAM アクションを許可する IAM ポリシーがアタッチされている必要があります。

よく発生する IAM エラーのリストについては、「IAM 認証エラー」を参照してください。

グラフサマリーリクエストが返す可能性のある一般的なエラーコード

Neptune サービスのエラーコード	HTTステータス	メッセージ	エラーシナリオ	緩和
AccessDeniedException	403	認証トークンが見つかりません。	署名されていない、または正しく署名されていないリクエストが IAM が有効な Neptune データベースに送信されました。	送信する前に SigV4 でリクエストに署名してください (「IAM とグラフサマリー」を参照)。
	403	ユーザー: *### ## ARN)* には、neptune-db:GetGraphSummary on resource: *(resource ARN)* を実行する権限がありません。	IAM ポリシーでは、グラフサマリーリクエストが IAM GetGraphを有効にして Neptune データベースに送信されたときに、 アクション概要を許可しません。	リクエストを行うユーザーまたはロールにアタッチされている IAM ポリシーが `GetGraphSummary` アクションを許可していることを確認してください。
BadRequestException	400	統計は無効になっているため、グラフサ	統計が無効になっているバースト可能なインスタンスタイプ	統計の生成が有効になっているインスタンスタイプを使用し

Neptune サービスのエラーコード	HTTP ステータス	メッセージ	エラーシナリオ	緩和
		マリーも無効になっています。	(t3 または t4g) でサマリーを取得しようとしています。	てください (t3 と t4g を除くすべてのサポート対象インスタンス)。
	400	不正なルート: */rdf/statistics/summarypathapi*	無効なパスに送信されたリクエスト。	グラフサマリーエンドポイントには正しいルートを使用してください。
InvalidParameterException	400	リクエストに未知のパラメータが含まれています: '*(########)* '。	リクエストで無効なパラメータが指定されている場合。	リクエストには有効なパラメータ (mode など) のみを使用してください。
InvalidParameterException	400	URI クエリパラメータ 'mode' にサポートされていない値 '*(####)*' があります。	リクエストの URL パラメータ 'mode' の後に無効な値が続く場合。	URL パラメータ 'mode' を指定するときには、有効な値 (basic や detailed など) を使用してください。
MethodNotAllowedException	405	許されないメソッド。	GET 以外の HTTP メソッド (POST や DELETE など) を使用してサマリーエンドポイントを呼び出す。	サマリーエンドポイントを呼び出すときには、HTTP GET メソッドを使用してください。

Neptune サービスのエラーコード	HTTP ステータス	メッセージ	エラーシナリオ	緩和
StatisticsNotAvailableException	400	統計はまだ計算されていません。統計の計算が完了すると、グラフサマリーが使用可能になります。	リクエストがサマリーエンドポイントに送信されたとき、利用できる統計がありません。	統計の生成が完了するまで待ちます。統計生成のステータスは、統計ステータス API を使用して確認できます。
	400	統計の上限に達したため、グラフサマリーは使用できません。	統計サイズ制限に達したため、統計の生成が停止しました。	このグラフに関するグラフサマリーは使用可能ではありません。

例えば、IAM 認証が有効になっている Neptune データベースで、必要なアクセス許可がリクエスタの IAM ポリシーにないとき、グラフサマリーエンドポイントにリクエストを行うと、次のようなレスポンスを受信します。

```
{
  "detailedMessage": "User: arn:aws:iam::(account ID):(user or user name) is not
 authorized to perform: neptune-db:GetGraphSummary on resource: arn:aws:neptune-
db:(region):(account ID):(cluster resource ID)/*",
  "requestId": "7ac2b98e-b626-d239-1d05-74b4c88fce82",
  "code": "AccessDeniedException"
}
```

Amazon Neptune JDBC 接続

Amazon Neptune は、openCypher、Gremlin、SQL-gremlin、SPARQLクエリをサポートしているオープンソース JDBC ドライバーをリリースしました。JDBC 接続により、Tableau などのビジネスインテリジェンス (BI) ツールを使用して Neptune に簡単に接続できます。Neptune で JDBC ドライバーを使用することに追加費用はかかりません。消費された Neptune リソースに対してのみお支払いいただきます。

ドライバーは JDBC 4.2 と互換性があり、少なくとも Java 8 が必要です。JDBC ドライバーの使用方法については、JDBC API ドキュメントを参照してください。

問題を提出して機能リクエストを開くことができる GitHub プロジェクトには、ドライバーの詳細なドキュメントが含まれています。

Amazon Neptune 用 JDBC ドライバー

- JDBC ドライバーを介した SQL の使用
- JDBC ドライバーを介した Gremlin の使用
- JDBC ドライバーを介した openCypher の使用
- JDBC ドライバーを介した SPARQL の使用

Neptune JDBC トライバーの開始方法

Neptune JDBC ドライバーを使用して Neptune インスタンスに接続するには、JDBC ドライバーを Neptune DB クラスターと同じ VPC 内の Amazon EC2 インスタンスにデプロイするか、インスタンスが SSH トンネルまたはロードバランサーを介して利用可能である必要があります。SSH トンネルは、ドライバーの内部で設定することも、外部で設定することもできます。

ドライバーは、こちら からダウンロードできます。ドライバーは、次のような名前を持つ単一の JAR ファイルとしてパッケージ化されます。`neptune-jdbc-1.0.0-all.jar`。これを使用するには、JAR ファイルをアプリケーションの `classpath` に置きます。または、アプリケーションで Maven または Gradle を使用している場合は、適切な Maven または Gradle コマンドを使用して JAR からドライバーをインストールできます。

ドライバーは Neptune に接続するには、次のような形式で JDBC 接続 URL が必要です。

```
jdbc:neptune:(connection
 type)://(host);property=value;property=value;...;property=value
```

GitHub プロジェクト内の各クエリ言語のセクションでは、そのクエリ言語の JDBC 接続 URL で設定できるプロパティについて説明します。

JAR ファイルがアプリケーションの `classpath` 内にある場合、その他の設定は必要ありません。JDBC `DriverManager` インターフェイスと Neptune 接続文字列を使用してドライバーを接続できます。たとえば、ポート 8182 の `neptune-example.com` エンドポイントから Neptune DB クラスターにアクセスできる場合、openCypher で次のように接続できます。

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

void example() {
    String url = "jdbc:neptune:opencypher://bolt://neptune-example:8182";

    Connection connection = DriverManager.getConnection(url);
    Statement statement = connection.createStatement();

    connection.close();
}
```

各クエリ言語の GitHub プロジェクトのドキュメントセクションでは、そのクエリ言語を使用するときに接続文字列を構築する方法について説明します。

Neptune JDBC ドライバーで Tableau を使用する

Neptune JDBC ドライバーで Tableau を使用するには、まず Tableau デスクトップの最新バージョンをダウンロードしてインストールします。Neptune JDBC ドライバー用の JAR ファイルと Neptune Tableau コネクタファイル (`.taco`) をダウンロードします。

Mac で Tableau for Neptune に接続するには

1. Neptune JDBC ドライバー JAR ファイルを `/Users/`*`(your user name)`*`/Library/Tableau/Drivers` フォルダに置きます。
2. Neptune Tableau コネクタ `.taco` ファイルを `/Users/`*`(your user name)`*`/Documents/My Tableau Repository/Connectors` フォルダに置きます。
3. IAM 認証が有効になっている場合は、その環境を設定します。環境変数が`.zprofile/`、`.zshenv/`、`.bash_profile`などで設定されている機能しないことに注意してください。環境変数は、GUI アプリケーションで読み込めるように設定する必要があります。

認証情報を設定する 1 つの方法は、アクセスキーおよびシークレットキーを /Users/*(your user name)*/.aws/credentials ファイルに置くことです。

サービスリージョンを簡単に設定するには、ターミナルを開き、アプリケーションのリージョンを使用して次のコマンドを入力します (たとえば、us-east-1)。

```
launchctl setenv SERVICE_REGION region name
```

再起動後も持続する環境変数を設定する方法は他にもありますが、どの方法を使用する場合でも、GUI アプリケーションからアクセス可能な変数を設定する必要があります。

4. Mac の GUI に環境変数をロードするには、ターミナルで次のコマンドを入力します。

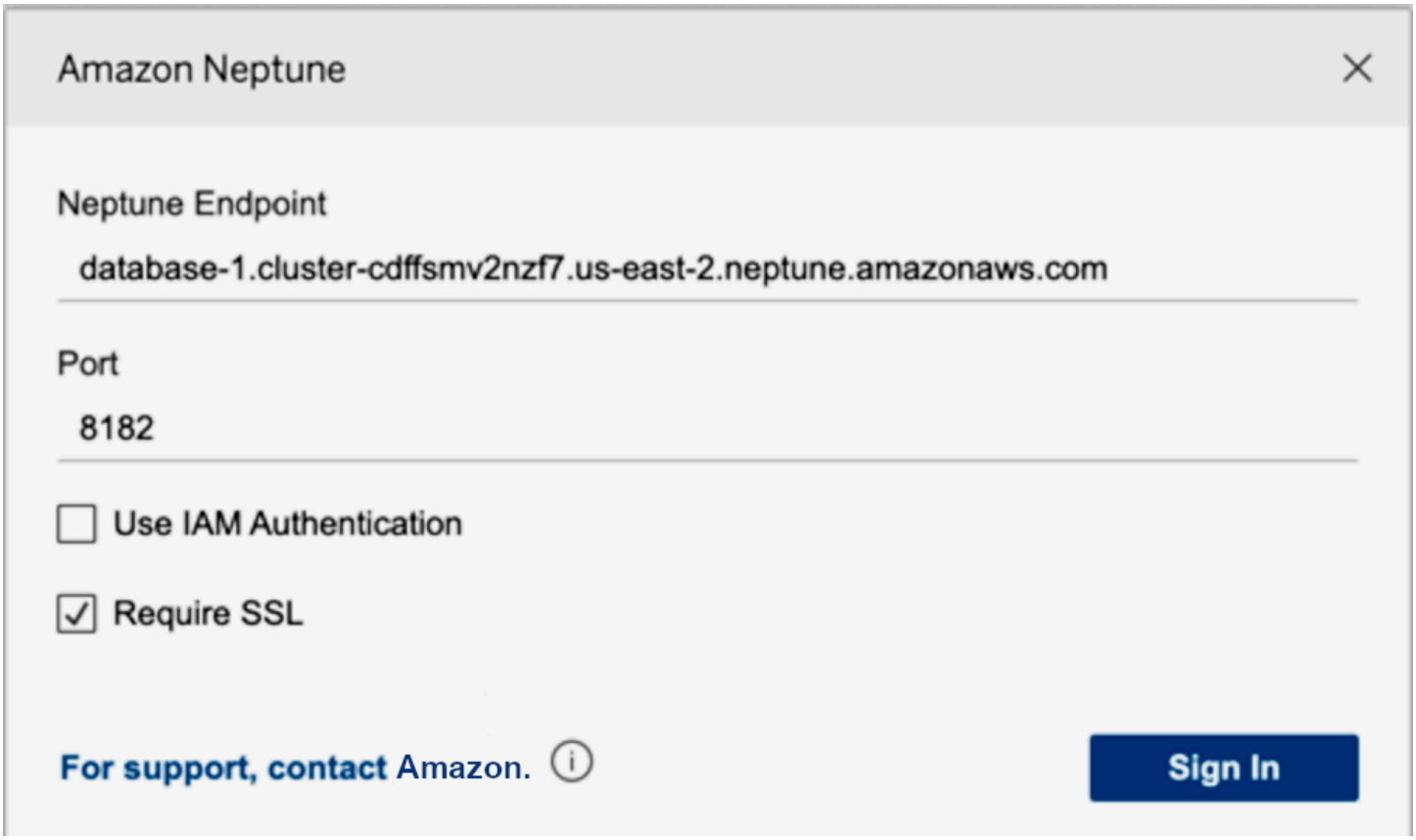
```
/Applications/Tableau/Desktop/2021.1.app/Contents/MacOS/Tableau
```

Windows マシンで Tableau for Neptune に接続するには

1. Neptune JDBC ドライバー JAR ファイルを C:\Program Files\Tableau\Drivers フォルダに置きます。
2. Neptune Tableau コネクタ .taco ファイルを C:\Users*(your user name)*\Documents\My Tableau Repository\Connectors フォルダに置きます。
3. IAM 認証が有効になっている場合は、その環境を設定します。

   これは、ユーザーのACCESS_KEY、SECRET_KEY、およびSERVICE_REGION環境変数設定と同じくらいシンプルです。

Tableau を開いた状態で、MOREウィンドウの左側にあります。Tableau コネクタファイルが適切に配置されている場合は、表示されたリストでAmazon Neptune を AWSでを選択します。



ポートを編集したり、接続オプションを追加したりする必要はありません。Neptune エンドポイントを入力し、IAM と SSL の設定を設定します (IAM を使用している場合は SSL を有効にする必要があります) 。

選択した場合サインインの場合、グラフが大きい場合、接続に 30 秒以上かかることがあります。Tableau では、頂点テーブルとエッジのテーブルを収集し、エッジ上の頂点を結合し、ビジュアライゼーションを作成しています。

JDBC ドライバー接続のトラブルシューティング

ドライバーがサーバーに接続できない場合は、JDBC `Connection` オブジェクトの `isValid` 関数を使用して、接続が有効かどうかをチェックします。関数が `false` を返した場合、接続が無効であることを示します。接続先のエンドポイントが正しいこと、および Neptune DB クラスターの VPC にいること、またはクラスターへの有効な SSH トンネルがあることを確認します。

`DriverManager.getConnection` を呼び出して `No suitable driver found for` *`(connection string)`* の応答があれば、接続文字列の先頭に問題がある可能性があります。接続文字列が次のように始まっていることを確認してください。

```
jdbc:neptune:opencypher://...
```

接続に関する詳細情報を収集するには、LogLevel 接続文字列を次のようにします。

```
jdbc:neptune:opencypher://(JDBC URL):(port);logLevel=trace
```

または、入力プロパティに properties.put("logLevel", "trace") を追加してトレース情報を記録します。

Amazon Neptune エンジンの更新

Amazon Neptune は定期的にエンジンの更新をリリースします。instance-status API を使用して、現在インストールされているエンジンリリースバージョンを確認できます。

エンジンのリリースは Amazon Neptune のエンジンリリース に一覧があります。パッチについては最新情報 に一覧があります。

データベースでの Neptune エンジンの更新のリリース方法とアップグレード方法の詳細については、「クラスターのメンテナンス」を参照してください。例えば、エンジンのバージョン番号については、「エンジンのバージョン番号」に説明があります。

Amazon Neptune のセキュリティ

のクラウドセキュリティが最優先事項 AWS です。お客様は AWS 、セキュリティを最も重視する組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャからメリットを得られます。

セキュリティは、 AWS とユーザーの間で共有される責任です。責任共有モデルでは、これをクラウドのセキュリティおよびクラウド内のセキュリティとして説明しています。

- クラウドのセキュリティ — AWS は、 AWS クラウドで AWS サービスを実行するインフラストラクチャを保護する責任を担います。 AWS また、 は、安全に使用できるサービスも提供します。AWS コンプライアンスプログラムの一環として、サードパーティーの監査が定期的にセキュリティの有効性をテストおよび検証しています。Amazon Neptune に適用されるコンプライアンスプログラムについては、コンプライアンスプログラムによるAWS 対象範囲内のサービスを参照してください。
- クラウドのセキュリティ — お客様の責任は、使用する AWS サービスによって決まります。また、お客様は、データの機密性、会社の要件、適用される法律や規制など、その他の要因についても責任を負います。

このドキュメントは、Neptune を使用する際に責任共有モデルを適用する方法を理解するのに役立ちます。以下のトピックでは、セキュリティおよびコンプライアンスの目的を達成するために Neptune を設定する方法を示します。また、Neptune リソースのモニタリングや保護に役立つ他の AWS のサービスの使用方法についても説明します。

トピック

- Amazon Neptune の耐障害性

Amazon Neptune におけるデータ保護

責任 AWS 共有モデル、Amazon Neptune でのデータ保護に適用されます。このモデルで説明されているように、 AWS はすべての を実行するグローバルインフラストラクチャを保護する責任があります AWS クラウド。お客様は、このインフラストラクチャでホストされているコンテンツに対する管理を維持する責任があります。また、使用する AWS のサービス のセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、「データプライバシーのよくある質問」を参照してください。欧州でのデータ保護の詳細については、AWS セキュリティブログに投稿された「AWS 責任共有モデルおよび GDPR」のブログ記事を参照してください。

データ保護の目的で、 認証情報を保護し AWS アカウント 、 AWS IAM Identity Center または AWS Identity and Access Management (IAM) を使用して個々のユーザーを設定することをお勧めします。この方法により、それぞれのジョブを遂行するために必要な権限のみが各ユーザーに付与されます。また、次の方法でデータを保護することもお勧めします:

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。TLS 1.2 は必須であり TLS 1.3 がお勧めです。
- を使用して API とユーザーアクティビティのログ記録を設定します AWS CloudTrail。
- AWS 暗号化ソリューションと、 内のすべてのデフォルトのセキュリティコントロールを使用します AWS のサービス。
- Amazon Macie などの高度なマネージドセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは API AWS を介して にアクセスするときに FIPS 140-2 検証済みの暗号化モジュールが必要な場合は、FIPS エンドポイントを使用します。利用可能な FIPS エンドポイントの詳細については、「連邦情報処理規格 (FIPS) 140-2」を参照してください。

お客様の E メールアドレスなどの極秘または機密情報は、タグ、または名前フィールドなどの自由形式のテキストフィールドに配置しないことを強くお勧めします。これは、コンソール、API、または SDK を使用して Neptune AWS CLIまたは他の AWS のサービス を使用する場合も同様です。AWS SDKs 名前に使用する自由記述のテキストフィールドやタグに入力したデータは、課金や診断ログに使用される場合があります。外部サーバーへの URL を提供する場合は、そのサーバーへのリクエストを検証するための認証情報を URL に含めないように強くお勧めします。

> **Important**
>
> TLS 1.3 は、Neptune エンジンバージョン 1.3.2.0 以降でのみサポートされています。

AWS が公開した API コールを使用して、ネットワーク経由で Neptune を管理します。クライアントは、転送時の暗号化 に記載のように、強力な暗号スイートを使用して Transport Layer Security (TLS) 1.2 以降をサポートする必要があります。これらのモードは、Java 7 以降など、ほとんどの最新システムでサポートされています。

以下のセクションでは、Neptune データが保護される仕組みについて説明します。

トピック

- すべての Amazon Neptune DB クラスターは Amazon VPC に存在します
- 転送中の暗号化: SSL/HTTPS を使用して Neptune に接続する
- 保管時の Neptune リソースの暗号化

すべての Amazon Neptune DB クラスターは Amazon VPC に存在します

Amazon Neptune DB クラスターは、Amazon Virtual Private Cloud (Amazon VPC) にのみ作成でき、そのエンドポイントはその VPC 内でのみアクセス可能であり、通常、その VPC で実行している Amazon Elastic Compute Cloud (Amazon EC2) インスタンスからです。

Amazon Neptune ングラフへの接続 で説明されているように、Neptune DB クラスターが配置されている VPC へのアクセスを制限することで、Neptune データを保護できます。

転送中の暗号化: SSL/HTTPS を使用して Neptune に接続する

エンジンバージョン 1.0.4.0 から、Amazon Neptune では、インスタンスまたはクラスターエンドポイントへの HTTPS 経由での Secure Sockets Layer (SSL) 接続のみが可能です。

Neptune には、次の強力な暗号スイートを使用する TLS バージョン 1.2 以上が必要です。

- `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`
- `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`
- `TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384`
- `TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256`

Neptune エンジンバージョン 1.3.2.0 以降、Neptune は次の暗号スイートを使用して TLS バージョン 1.3 をサポートしています。

- TLS_AES_128_GCM_SHA256
- TLS_AES_256_GCM_SHA384

以前のエンジンバージョンで HTTP 接続が許可されている場合でも、新しい DB クラスターパラメータグループを使用する DB クラスターは、デフォルトで SSL を使用する必要があります。データを保護するため、エンジンバージョン *1.0.4.0* 以上の Neptune エンドポイントは HTTPS リクエストのみをサポートします。詳細については、「HTTP REST エンドポイントを使用して Neptune DB インスタンスに接続する」を参照してください。

Neptune は Neptune DB インスタンスの SSL 証明書を自動的に提供します。証明書をリクエストする必要はありません。新しいインスタンスを作成するときに、証明書が提供されます。

Neptune は、各 AWS リージョンのアカウント内のインスタンスに 1 つのワイルドカード SSL 証明書を割り当てます。証明書では、クラスターエンドポイント、クラスターの読み取り専用エンドポイント、インスタンスエンドポイントのエントリが提供されます。

証明書の詳細

提供された証明書には以下のエントリが含まれます。

- クラスターエンドポイント — `*.cluster-`*`a1b2c3d4wxyz`*`.`*`region`*`.neptune.amazonaws.com`
- 読み取り専用エンドポイント — `*.cluster-ro-`*`a1b2c3d4wxyz`*`.`*`region`*`.neptune.amazonaws.com`
- インスタンスエンドポイント — `*.`*`a1b2c3d4wxyz`*`.`*`region`*`.neptune.amazonaws.com`

ここにリストされているエントリのみがサポートされています。

プロキシ接続

証明書では前のセクションでリストされたホスト名のみがサポートされています。

ロードバランサーまたはプロキシサーバー (HAProxy など) を使用している場合は、SSL ターミネーションを使用して独自の SSL 証明書をプロキシサーバーに保存する必要があります。

提供された SSL 証明書はプロキシサーバーのホスト名と一致しないため、SSL パススルーは機能しません。

ルート CA 証明書

Neptune インスタンスの証明書は通常、オペレーティングシステムまたは SDK (Java SDK など) のローカル信頼ストアを使用して検証されます。

ルート証明書を手動で入力する必要がある場合は、Amazon トラストサービスポリシーリポジトリから PEM 形式で Amazon Root CA 証明書をダウンロードしてください。

詳細情報

SSL を使用した Neptune エンドポイントへの接続の詳細については、the section called “Gremlin コンソールのインストール” および the section called “HTTP REST” を参照してください。

保管時の Neptune リソースの暗号化

Neptune の暗号化されたインスタンスは、基になるストレージへの不正アクセスからデータを保護することで、データ保護のレイヤーを追加します。Neptune の暗号化を使用すると、クラウドにデプロイされたアプリケーションのデータ保護を強化できます。また、暗号化のコンプライアンス要件を満たすためにも使用できます data-at-rest 。

Neptune リソースの暗号化と復号に使用されるキーを管理するには、 AWS Key Management Service （ AWS KMS ）. AWS KMS combines を使用して、安全で可用性の高いハードウェアとソフトウェアを使用し、クラウド向けにスケーリングされたキー管理システムを提供します。を使用すると AWS KMS、暗号化キーを作成し、これらのキーの使用方法を制御するポリシーを定義できます。は AWS KMS をサポートしているため AWS CloudTrail、キーの使用状況を監査して、キーが適切に使用されていることを確認できます。 AWS KMS キーは、Neptune および Amazon Simple Storage Service (Amazon S3)、Amazon Elastic Block Store (Amazon EBS)、Amazon Redshift などのサポートされている AWS サービスと組み合わせて使用できます。をサポートするサービスのリストについては AWS KMS、「 AWS Key Management Service デベロッパーガイド」の「 AWS のサービスがを使用する AWS KMS方法」を参照してください。

すべてのログ、バックアップ、スナップショットは、Neptune の暗号化されたインスタンス用に暗号化されます。

Neptune DB インスタンスの暗号化の有効化

新しい Neptune DB インスタンスの暗号化を有効にするには、 コンソールの [Enable encryption] (暗号化の有効化) で [Yes] (はい) を選択します。Neptune DB インスタンスの作成については、新しい Neptune DB クラスターの作成 を参照してください。

暗号化された Neptune DB インスタンスを作成するときに、暗号化 AWS KMS キーのキー識別子を指定することもできます。 AWS KMS キー識別子を指定しない場合、Neptune は新しい Neptune DB インスタンスにデフォルトの Amazon RDS 暗号化キー (aws/rds) を使用します。 AWS KMS は AWS 、アカウントの Neptune のデフォルトの暗号化キーを作成します。 AWS アカウントには、AWS リージョンごとに異なるデフォルトの暗号化キーがあります。

暗号化された Neptune DB インスタンスを作成したら、そのインスタンスの暗号化キーを変更することはできません。したがって、暗号化された Neptune DB インスタンスを作成する前に、暗号化キーの要件を確認してください。

別のアカウントのキーの Amazon リソースネーム (ARN) を使用して、Neptune DB インスタンスを暗号化できます。新しい Neptune DB インスタンスの AWS KMS 暗号化に使用される暗号化キーを所有するアカウントと同じ AWS アカウントで Neptune DB インスタンスを作成する場合、渡す AWS KMS キー ID は AWS KMS キーの ARN ではなくキーエイリアスになります。

> **Important**
>
> Neptune が Neptune DB インスタンスの暗号化キーにアクセスできなくなった場合 (Neptune のキーへのアクセス権が失効した場合など)、暗号化された DB インスタンスは終了状態になり、バックアップからのみ復元できます。データベース内の暗号化されたデータの消失を防ぐために、暗号化された Neptune DB インスタンスのバックアップは常に有効にしておくことを強くお勧めします。

暗号化を有効にするときに必要なキー許可

暗号化された Neptune DB インスタンスを作成する IAM ユーザーまたはロールには、少なくとも KMS キーに対する次の権限が必要です。

- `"kms:Encrypt"`
- `"kms:Decrypt"`
- `"kms:GenerateDataKey"`
- `"kms:ReEncryptTo"`
- `"kms:GenerateDataKeyWithoutPlaintext"`
- `"kms:CreateGrant"`
- `"kms:ReEncryptFrom"`
- `"kms:DescribeKey"`

次に、必要なアクセス権限を含むキーポリシーの例を示します。

```
{
  "Version": "2012-10-17",
  "Id": "key-consolepolicy-3",
  "Statement": [
    {
      "Sid": "Enable Permissions for root principal",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:root"
      },
      "Action": "kms:*",
      "Resource": "*"
    },
    {
      "Sid": "Allow use of the key for Neptune",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:role/NeptuneFullAccess"
      },
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:GenerateDataKey",
        "kms:ReEncryptTo",
        "kms:GenerateDataKeyWithoutPlaintext",
        "kms:CreateGrant",
        "kms:ReEncryptFrom",
        "kms:DescribeKey"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "kms:ViaService": "rds.us-east-1.amazonaws.com"
        }
      }
    },
    {
      "Sid": "Deny use of the key for non Neptune",
      "Effect": "Deny",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:role/NeptuneFullAccess"
      },
```

```
      "Action": [
        "kms:*"
      ],
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "kms:ViaService": "rds.us-east-1.amazonaws.com"
        }
      }
    }
  ]
}
```

- このポリシーの最初のステートメントはオプションです。これにより、ユーザーのルートプリンシパルにアクセスできます。
- 2 番目のステートメントは、RDS サービスプリンシパルにスコープダウンされた、このロールに必要なすべての AWS KMS APIs へのアクセスを提供します。
- 3 番目のステートメントでは、このキーを他の AWS サービスで使用できないように強制することで、セキュリティを強化します。

以下を追加して `createGrant` 許可をさらに範囲指定することもできます。

```
"Condition": {
  "Bool": {
    "kms:GrantIsForAWSResource": true
  }
}
```

Neptune 暗号化の制約事項

Neptune クラスターの暗号化には、以下の制約事項があります。

- 暗号化されていない DB クラスターを暗号化された DB クラスターに変換することはできません。

  ただし、暗号化されていない DB クラスタースナップショットを暗号化された DB クラスターに復元することができます。そのためには、暗号化されていない DB クラスタースナップショットから復元する場合は、KMS 暗号化キーを指定します。
- 暗号化されていない DB インスタンスを暗号化された DB クラスターに変換することはできません。DB インスタンスの暗号化は、DB インスタンスの作成時にのみ有効にすることができます。

- 暗号化された DB インスタンスを変更して暗号化を無効にすることはできません。
- 暗号化されていない DB インスタンスのリードレプリカを暗号化することや、暗号化された DB インスタンスのリードレプリカを暗号化しないことは設定できません。
- 暗号化されたリードレプリカは、ソース DB インスタンスと同じキーで暗号化する必要があります。

Amazon Neptune での AWS Identity and Access Management (IAM) の概要

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御 AWS のサービス するのに役立つ です。IAM 管理者は、誰を認証 (サインイン) し、誰に Neptune リソースの使用を許可する (アクセス許可を持たせる) かを制御します。IAM は、追加料金なしで AWS のサービス 使用できる です。

AWS Identity and Access Management (IAM) を使用して、Neptune DB インスタンスまたは DB クラスターを認証できます。IAM データベース認証が有効になっている場合、各リクエストは Signature Version AWS 4 を使用して署名する必要があります。

AWS 署名バージョン 4 は、認証情報を AWS リクエストに追加します。セキュリティ上の理由から、UAM 認証が有効である Neptune DB クラスターへのすべてのリクエストはアクセスキーを使用して署名する必要があります。このキーは、アクセスキー ID とシークレットアクセスキーで構成されます。この認証は、IAM ポリシーを使用して外部で管理されます。

Neptune は接続時に認証し、 WebSockets 接続に対して定期的にアクセス許可を検証して、ユーザーが引き続きアクセスできることを確認します。

> **Note**
>
> - IAM ユーザーに関連付けられている認証情報の取り消し、削除、更新によって確立済みのオープン接続は終了されないため、推奨されません。
> - データベースインスタンスあたりの同時 WebSocket 接続数と、接続を開いたままにできる時間には制限があります。詳細については、「WebSockets 制限」を参照してください。

IAM の使用はロールによって異なる

AWS Identity and Access Management (IAM) の使用方法は、Neptune で行う作業によって異なります。

サービスユーザー – ジョブを実行するために Neptune サービスを使用する場合、Neptune データプレーンの使用に必要な認証情報とアクセス許可が管理者から与えられます。作業を実行するために、より多くのアクセスが必要になるにつれて、アクセスの管理方法を理解しておくと、管理者に適切なアクセス許可をリクエストするうえで役立ちます。

サービス管理者 — 社内の Neptune リソースを担当している場合は、おそらく Neptune 管理 API に対応する Neptune 管理アクションにアクセスできます。また、サービスユーザーが仕事をするために必要な Neptune データアクセスアクションとリソースを決定するのもあなたの仕事かもしれません。その場合、IAM 管理者は IAM ポリシーを適用して、サービスユーザーのアクセス許可を変更できます。

IAM 管理者 — IAM 管理者は、IAM ポリシーを作成して、 Neptune の管理とデータアクセスの両方を管理する必要があります。使用できる Neptune アイデンティティベースのポリシーの例を表示するには、「Neptune へのアクセスを制御するためのさまざまな種類の IAM ポリシーの使用」 を参照してください。

アイデンティティを使用した認証

認証とは、ID 認証情報 AWS を使用して にサインインする方法です。として、IAM ユーザーとして AWS アカウントのルートユーザー、または IAM ロールを引き受けて認証 (にサインイン AWS) される必要があります。

ID ソースを介して提供された認証情報を使用して、フェデレーティッド ID AWS として にサインインできます。 AWS IAM Identity Center （ IAM Identity Center) ユーザー、会社のシングルサインオン認証、Google または Facebook の認証情報は、フェデレーティッド ID の例です。フェデレーティッド ID としてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーション AWS を使用して にアクセスすると、間接的にロールを引き受けることになります。

ユーザーのタイプに応じて、 AWS Management Console または AWS アクセスポータルにサインインできます。へのサインインの詳細については AWS、「 ユーザーガイド」の「 へのサインイン AWS アカウント方法AWS サインイン 」を参照してください。

AWS プログラムで にアクセスする場合、 は Software Development Kit (SDK) とコマンドラインインターフェイス (CLI) AWS を提供し、認証情報を使用してリクエストに暗号で署名します。 AWS

ツールを使用しない場合は、リクエストに自分で署名する必要があります。推奨される方法を使用してリクエストを自分で署名する方法の詳細については、IAM ユーザーガイドの API AWS リクエストの署名を参照してください。

使用する認証方法を問わず、追加セキュリティ情報の提供をリクエストされる場合もあります。例えば、 AWS では、多要素認証 (MFA) を使用してアカウントのセキュリティを向上させることをお勧めします。詳細については、「AWS IAM Identity Center ユーザーガイド」の「多要素認証」および「IAM ユーザーガイド」の「AWSでの多要素認証 (MFA) の使用」を参照してください。

AWS アカウント ルートユーザー

を作成するときは AWS アカウント、アカウント内のすべての AWS のサービス およびリソースへの完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。この ID は AWS アカウント ルートユーザーと呼ばれ、アカウントの作成に使用した E メールアドレスとパスワードでサインインすることでアクセスできます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザーの認証情報は保護し、ルートユーザーでしか実行できないタスクを実行するときに使用します。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、IAM ユーザーガイドの「ルートユーザー認証情報が必要なタスク」を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、単一のユーザーまたはアプリケーションに対して特定のアクセス許可 AWS アカウント を持つ 内のアイデンティティです。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、IAM ユーザーガイド の「長期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションする」を参照してください。

IAM グループは、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、IAMAdmins という名前のグループを設定して、そのグループに IAM リソースを管理する許可を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時的な認証情報が提供されます。詳

細については、「IAM ユーザーガイド」の「IAM ユーザー (ロールではなく) の作成が適している場合」を参照してください。

IAM ロール

IAM ロールは、特定のアクセス許可 AWS アカウント を持つ 内のアイデンティティです。これは IAM ユーザーに似ていますが、特定のユーザーには関連付けられていません。ロール を切り替える AWS Management Console ことで、 で IAM ロールを一時的に引き受けることができます。ロールを引き受けるには、 または AWS API AWS CLI オペレーションを呼び出すか、カスタム URL を使用します。ロールを使用する方法の詳細については、「IAM ユーザーガイド」の「IAM ロールの使用」を参照してください。

IAM ロールと一時的な認証情報は、次の状況で役立ちます:

- フェデレーションユーザーアクセス – フェデレーティッド ID に許可を割り当てるには、ロールを作成してそのロールの許可を定義します。フェデレーティッド ID が認証されると、その ID はロールに関連付けられ、ロールで定義されている許可が付与されます。フェデレーションの詳細については、「IAM ユーザーガイド」の「Creating a role for a third-party Identity Provider」(サードパーティーアイデンティティプロバイダー向けロールの作成) を参照してください。IAM Identity Center を使用する場合は、許可セットを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM Identity Center は、権限セットを IAM のロールに関連付けます。アクセス許可セットの詳細については、「AWS IAM Identity Center ユーザーガイド」の「アクセス許可セット」を参照してください。
- 一時的な IAM ユーザー権限 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる権限を一時的に IAM ロールで引き受けることができます。
- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。クロスアカウントアクセス権を付与する主な方法は、ロールを使用することです。ただし、一部の では AWS のサービス、 (ロールをプロキシとして使用する代わりに) ポリシーをリソースに直接アタッチできます。クロスアカウントアクセスのロールとリソースベースのポリシーの違いについては、「IAM ユーザーガイド」の「IAM でのクロスアカウントリソースアクセス」を参照してください。
- クロスサービスアクセス — 一部の は、他の の機能 AWS のサービス を使用します AWS のサービス。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスでは、呼び出し元プリンシパルの許可、サービスロール、またはサービスリンクロールを使用してこれを行う場合があります。

- 転送アクセスセッション (FAS) – IAM ユーザーまたはロールを使用して でアクションを実行する場合 AWS、ユーザーはプリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、 を呼び出すプリンシパルのアクセス許可を AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストリクエストリクエストと組み合わせて使用します。FAS リクエストは、サービスが他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。
- サービスロール - サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、「IAM ユーザーガイド」の「AWS のサービスにアクセス許可を委任するロールの作成」を参照してください。
- サービスにリンクされたロール – サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、サービスによって所有されます。IAM 管理者は、サービスリンクロールの許可を表示できますが、編集することはできません。

- Amazon EC2 で実行されているアプリケーション – IAM ロールを使用して、EC2 インスタンスで実行され、 AWS CLI または AWS API リクエストを行うアプリケーションの一時的な認証情報を管理できます。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。 AWS ロールを EC2 インスタンスに割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスにアタッチされたインスタンスプロファイルを作成します。インスタンスプロファイルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、「IAM ユーザーガイド」の「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用して許可を付与する」を参照してください。

IAM ロールと IAM ユーザーのどちらを使用するかについては、「IAM ユーザーガイド」の「(IAM ユーザーではなく) IAM ロールをいつ作成したら良いのか?」を参照してください。

Neptune で IAM データベース認証を有効にする

デフォルトでは、Amazon Neptune DB クラスターの作成時、IAM データベース認証は無効になっています。 AWS Management Consoleを使用して、IAM データベース認証を有効にする (またはもう一度無効にする) ことができます

コンソールを使用して IAM 認証で新しい Neptune DB クラスターを作成するには、AWS Management Console を使用して Neptune DB クラスターを起動する にある Neptune DB DB クラスターを作成するための手順に従う必要があります。

作成プロセスの 2 番目のページで、[Enable IAM DB Authentication] (IAM DB 認証を有効にする) に [Yes] (はい) を選択します。

既存の DB インスタンスまたはクラスターに対して IAM 認証を有効または無効にするには

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/neptune/home で Amazon Neptune コンソールを開きます。
2. ナビゲーションペインで [クラスター] を選択します。
3. 変更する Neptune DB クラスターを選択し、[Cluster actions] (クラスタークラスターアクション) を選択します。その後、[Modify Cluster] を選択します。
4. [Database options] セクションの [IAM DB Authentication] (IAM DB 認証) で、[Enable IAM DB authorization] (IAM DB 認証を有効にする) または [No] (いいえ) (無効にする場合) を選択します。[Continue] を選択します。
5. 変更をすぐに反映させるには、[Apply immediately] を選択します。
6. [クラスタークラスターの変更] を選択します。

AWS 署名バージョン 4 での接続と署名

IAM DB 認証が有効になっている Amazon Neptune リソースでは、署名バージョン 4 を使用してすべての HTTP AWS リクエストに署名する必要があります。Signature AWS Version 4 を使用したリクエストの署名に関する一般的な情報については、AWS 「 API リクエストの署名」を参照してください。

AWS 署名バージョン 4 は、認証情報を AWS リクエストに追加するプロセスです。セキュリティ上の理由から、 へのほとんどのリクエストは、アクセスキー ID とシークレットアクセスキーで構成されるアクセスキーで署名 AWS する必要があります。

> **Note**
>
> 一時的なセキュリティ認証情報を使用している場合は、セッショントークンを含めて、指定した期間が過ぎると失効します。

新しい認証情報をリクエストするときは、セッショントークンを更新する必要があります。詳細については、「一時的なセキュリティ認証情報を使用して AWS リソースへのアクセスをリクエストする」を参照してください。

Important

IAM ベースの認証を使用して Neptune にアクセスするには、HTTP リクエストを作成してリクエストに自分で署名することが必要です。

署名バージョン 4 の仕組み

1. 正規リクエストを作成します。
2. 正規リクエストとその他の情報を使用して を作成します string-to-sign。
3. AWS シークレットアクセスキーを使用して署名キーを取得し、その署名キーと string-to-sign を使用して署名を作成します。
4. 作成した署名をヘッダーの HTTP リクエストに追加するか、クエリ文字列パラメータとして追加します。

Neptune は、リクエストを受信すると、お客様が行ったのと同じステップで署名を計算します。次に、Neptune は、計算した署名とリクエストで送信された署名を比較します。署名が一致すると、リクエストが処理されます。署名が一致しない場合、リクエストは拒否されます。

AWS 署名バージョン 4 を使用したリクエストの署名に関する一般的な情報については、「」の「署名バージョン 4 の署名プロセス」を参照してくださいAWS 全般のリファレンス。

以下のセクションでは、 認証を有効にして Neptune DB インスタンスの Gremlin エンドポイントおよび SPARQL エンドポイントに署名付きリクエストを送信する方法について、例を挙げて説明します。

トピック

- Amazon Linux EC2 の前提条件
- コマンドラインツールを使用して Neptune DB クラスターにクエリを送信する
- 署名バージョン 4 で署名された Gremlin コンソールを使用して Neptune に接続する
- Java と Gremlin でバージョン 4 署名を使用して Neptune に接続する

- Java と SPARQL でバージョン 4 署名を使用して Neptune に接続する (RDF4J および Jena)
- SPARQL と Node.js でバージョン 4 署名を使用して Neptune に接続する
- 例: Python で署名バージョン 4 署名を使用して Neptune に接続

Amazon Linux EC2 の前提条件

以下に示しているのは、Apache Maven と Java 8 を Amazon EC2 インスタンスにインストールする手順です。これらは、Amazon Neptune 署名バージョン 4 認証サンプルで必要です。

Apache Maven と Java 8 を EC2 インスタンスにインストールするには

1. SSH クライアントを使用して Amazon EC2 インスタンスに接続します。
2. Apache Maven を EC2 インスタンスにインストールします。最初に、Maven パッケージを使用してリポジトリを追加するには、次のように入力します。

   ```
   sudo wget https://repos.fedorapeople.org/repos/dchen/apache-maven/epel-apache-maven.repo -O /etc/yum.repos.d/epel-apache-maven.repo
   ```

   パッケージのバージョン番号を設定するには、次のように入力します。

   ```
   sudo sed -i s/\$releasever/6/g /etc/yum.repos.d/epel-apache-maven.repo
   ```

   続いて、yum を使用して Maven をインストールできます。

   ```
   sudo yum install -y apache-maven
   ```
3. Gremlin ライブラリには Java 8 が必要です。EC2 インスタンスで Java 8 をインストールするには、次のように入力します。

   ```
   sudo yum install java-1.8.0-devel
   ```
4. EC2 インスタンスで Java 8 をデフォルトランタイムとして設定するには、次のように入力します。

   ```
   sudo /usr/sbin/alternatives --config java
   ```

   プロンプトが表示されたら、Java 8 の数を入力します。

5. EC2 インスタンスで Java 8 をデフォルトコンパイラとして設定するには、次のように入力します。

```
sudo /usr/sbin/alternatives --config javac
```

プロンプトが表示されたら、Java 8 の数を入力します。

コマンドラインツールを使用して Neptune DB クラスターにクエリを送信する

このドキュメントの多くの例に示されるように、Neptune DB クラスターにクエリを送信するためのコマンドラインツールがあると非常に便利です。curl ツールは、IAM 認証が有効になっていない場合に Neptune エンドポイントと通信するための優れたオプションです。

ただし、データを安全に保つには、IAM 認証を有効にするのが最善です。

IAM 認証が有効になっている場合、各リクエストは署名バージョン 4 (Sig4) を使用して署名する必要があります。サードパーティの awscurl コマンドラインツールは、curl と同じ構文を使用し、Sig4 署名を使用してクエリに署名できます。以下の awscurl を使用する セクションでは、一時的な認証情報で awscurl を安全に使用する方法について説明します。

HTTPS を使用するコマンドラインツールのセットアップ

Neptune では、すべての接続で HTTPS を使用する必要があります。HTTPS を使用するには、curl または awscurl などのコマンドラインツールが適切な証明書にアクセスする必要があります。curl または awscurl が適切な証明書を見つけられる限り、HTTP 接続と同じように HTTPS 接続を処理し、追加のパラメータを必要としません。このドキュメントの例はこのシナリオに基づいています。

そのような証明書を取得する方法と、curl が使える証明書を認証局 (CA) 証明書ストアに適切にフォーマットする方法については、curl ドキュメント内の「SSL 証明書の検証」を参照してください。

次に、CURL_CA_BUNDLE 環境変数を使用してこの CA 証明書ストアの場所を指定できます。Windows では、curl は自動的に curl-ca-bundle.crt という名前のファイルを検索します。まず curl.exe と同じディレクトリで curl.exe を検索し、次にこのパスの他の場所を検索します。詳細については、「SSL Certificate Verification」を参照してください。

一時的な認証情報で **awscurl** を使用して、IAM 認証が有効になっている DB クラスターに安全に接続する

awscurl ツールは、curl と同じ構文を使用しますが、追加情報も必要です。

- **--access_key** — 有効なアクセスキー。このパラメータを使用して指定しなかった場合は、AWS_ACCESS_KEY_ID 環境変数または設定ファイルで指定する必要があります。
- **--secret_key** - アクセスキーに対応する有効なシークレットキー。このパラメータを使用して指定しなかった場合は、AWS_SECRET_ACCESS_KEY 環境変数または設定ファイルで指定する必要があります。
- **--security_token** — 有効なセッショントークン。このパラメータを使用して指定しなかった場合は、AWS_SECURITY_TOKEN 環境変数または設定ファイルで指定する必要があります。

以前は、IAM ユーザー認証情報やルート認証情報などの永続認証情報を awscurl で使用するのが一般的でしたが、これは推奨されません。代わりに、AWS セキュリティトークンサービス (STS) API のいずれか、またはそのAWS CLI ラッパーのいずれかを使用して、一時的な認証情報を生成してください。

STS 呼び出しによって返される AccessKeyId、SecretAccessKey、および SessionToken 値は、設定ファイルではなく、シェルセッション内の適切な環境変数に置くのが最善です。その後、シェルを終了すると、認証情報は自動的に破棄されますが、設定ファイルの場合はそうではありません。同様に、一時的な認証情報について、必要と思われる期間よりも長い期間をリクエストしないでください。

次の例は、Linux シェルで sts assume-role を使用して 30 分有効な一時的な認証情報を取得し、awscurl で検索できる環境変数に格納する手順を示しています。

```
aws sts assume-role \
    --duration-seconds 1800 \
    --role-arn "arn:aws:iam::(account-id):role/(rolename)" \
    --role-session-name AWSCLI-Session > $output
AccessKeyId=$(cat $output | jq '.Credentials''.AccessKeyId')
SecretAccessKey=$(cat $output | jq '.Credentials''.SecretAccessKey')
SessionToken=$(cat $output | jq '.Credentials''.SessionToken')

export AWS_ACCESS_KEY_ID=$AccessKeyId
export AWS_SECRET_ACCESS_KEY=$SecretAccessKey
export AWS_SESSION_TOKEN=$SessionToken
```

その後、awscurl を使用して DB クラスターに次のような署名付きリクエストを行うことができます。

```
awscurl (your cluster endpoint):8182/status \
    --region us-east-1 \
    --service neptune-db
```

署名バージョン 4 で署名された Gremlin コンソールを使用して Neptune に接続する

Signature Version 4 認証で Gremlin コンソールを使用して Amazon Neptune に接続する方法は、TinkerPop バージョン 3.4.11以降を使用しているか、以前のバージョンを使用しているかによって異なります。いずれの場合も、以下の前提条件が必要です。

- リクエストに署名するために必要な IAM 認証情報を持っている必要があります。「 デベロッパーガイド」の「デフォルトの認証情報プロバイダーチェーンの使用」を参照してください。 AWS SDK for Java
- DB クラスターで使用されている Neptune エンジンのバージョンと互換性のある Gremlin コンソールバージョンをインストールしておく必要があります。

一時的な認証情報を使用している場合、セッショントークンと同様に、指定した間隔が過ぎると有効期限が切れるため、新しい認証情報をリクエストするときはセッショントークンを更新する必要があります。IAM ユーザーガイドの「一時的なセキュリティ認証情報を使用して AWS リソースへのアクセスをリクエストする」を参照してください。

SSL/TLS を使用して接続する方法については、「SSL/TLS 設定」を参照してください。

TinkerPop 3.4.11 以降を使用して Sig4 署名で Neptune に接続する

TinkerPop 3.4.11 以降ではhandshakeInterceptor()、 コマンドによって確立された接続に Sigv4 署名者を接続する方法を提供する を使用します:remote。Java で使用されている方法と同様に、Cluster オブジェクトを手動で設定して、それを :remote コマンドに渡す必要があります。

これは、:remote コマンドが設定ファイルを使用して接続を形成する一般的な状況とはかなり異なる点に注意してください。設定ファイルによるアプローチでは、handshakeInterceptor() をプログラムで設定する必要があり、ファイルから設定を読み込むことができないため、機能しません。

Sig4 署名を使用して Gremlin コンソール (TinkerPop 3.4.11 以降) を接続する Sig4

1. Gremlin コンソールを起動します。

```
$ bin/gremlin.sh
```

2. gremlin> プロンプトで、amazon-neptune-sigv4-signer ライブラリをインストールします (これはコンソールで 1 回だけ行う必要があります)。

```
:install com.amazonaws amazon-neptune-sigv4-signer 2.4.0
```

このステップで問題が発生した場合は、Grape 設定に関するTinkerPop ドキュメントを参照するのに役立ちます。

> **Note**
>
> HTTP プロキシを使用している場合、このステップで :install コマンドが完了しないというエラーが発生することがあります。この問題を解決するには、以下のコマンドを実行して、プロキシについてコンソールに通知します。
>
> ```
> System.setProperty("https.proxyHost", "(the proxy IP address)")
> System.setProperty("https.proxyPort", "(the proxy port)")
> ```

3. handshakeInterceptor() へのサインインの処理に必要なクラスをインポートします。

```
:import com.amazonaws.auth.DefaultAWSCredentialsProviderChain
:import com.amazonaws.neptune.auth.NeptuneNettyHttpSigV4Signer
```

4. 一時的な認証情報を使用する場合は、以下のようにセッショントークンも指定する必要があります。

```
System.setProperty("aws.sessionToken","(your session token)")
```

5. アカウント認証情報をまだ設定していない場合は、以下のように割り当てることができます。

```
System.setProperty("aws.accessKeyId","(your access key)")
System.setProperty("aws.secretKey","(your secret key)")
```

6. Neptune に接続するように、Cluster オブジェクトを手動で設定します。

```
cluster = Cluster.build("(host name)")  \
                .enableSsl(true) \
                .handshakeInterceptor { r ->  \
                  def sigV4Signer = new NeptuneNettyHttpSigV4Signer("(Amazon
 region)", \
                                        new DefaultAWSCredentialsProviderChain()); \
                  sigV4Signer.signRequest(r); \
                  return r; } \
                .create()
```

Neptune DB インスタンスのホスト名を見つける方法については、「Amazon Neptune エンドポイントに接続する」を参照してください。

7. 前のステップの `Cluster` オブジェクトの変数名を使用して `:remote` 接続を確立します。

```
:remote connect tinkerpop.server cluster
```

8. 次のコマンドを入力して、リモートモードに切り替えます。これにより、すべての Gremlin クエリがリモート接続に送信されます。

```
:remote console
```

3.4.11 より前のバージョンの TinkerPop を使用して Sig4 署名で Neptune に接続する

TinkerPop 3.4.10 以前では、以下で説明するように、Neptune が提供する`amazon-neptune-gremlin-java-sigv4`ライブラリを使用して、Sigv4 署名でコンソールを Neptune に接続します。

Sig4 署名を使用して Gremlin コンソール (3.4.11 より前のTinkerPop バージョン) を接続する

1. Gremlin コンソールを起動します。

```
$ bin/gremlin.sh
```

2. `gremlin>` プロンプトで、`amazon-neptune-sigv4-signer` ライブラリをインストールします (これはコンソールで 1 回だけ行う必要があります)。

```
:install com.amazonaws amazon-neptune-sigv4-signer 2.4.0
```

> **Note**
>
> HTTP プロキシを使用している場合、このステップで :install コマンドが完了しないというエラーが発生することがあります。この問題を解決するには、以下のコマンドを実行して、プロキシについてコンソールに通知します。
>
> ```
> System.setProperty("https.proxyHost", "(the proxy IP address)")
> System.setProperty("https.proxyPort", "(the proxy port)")
> ```
>
> また、Grape 設定に関するTinkerPop ドキュメントを参照するのにも役立ちます。

3. 抽出されたディレクトリの conf サブディレクトリに、neptune-remote.yaml という名前のファイルを作成します。

   AWS CloudFormation テンプレートを使用して Neptune DB クラスターを作成した場合、neptune-remote.yaml ファイルは既に存在します。その場合、しなければならないことは、既存のファイルを編集して、以下に示すチャネライザー設定を含めることだけです。

   それ以外の場合は、次のテキストをファイルにコピーし、*(host name)* を Neptune DB インスタンスのホスト名または IP アドレスに置き換えます。ホスト名を囲む角括弧 ([]) は必須であることに注意してください。

   ```
   hosts: [(host name)]
   port: 8182
   connectionPool: {
     channelizer: org.apache.tinkerpop.gremlin.driver.SigV4WebSocketChannelizer,
     enableSsl: true
   }
   serializer: { className:
    org.apache.tinkerpop.gremlin.driver.ser.GraphBinaryMessageSerializerV1,
                 config: { serializeResultToString: true }}
   ```

4. > **Important**
   >
   > リクエストに署名するために IAM の認証情報を指定する必要があります。認証情報を環境変数として設定するには、次のコマンドを入力して該当する項目を認証情報に置き換えます。
   >
   > ```
   > export AWS_ACCESS_KEY_ID=access_key_id
   > ```

```
export AWS_SECRET_ACCESS_KEY=secret_access_key
export SERVICE_REGION=us-east-1 or us-east-2 or us-west-1 or us-west-2 or
 ca-central-1 or
                      sa-east-1 or eu-north-1 or eu-west-1 or eu-west-2 or
 eu-west-3 or eu-central-1 or me-south-1 or
                      me-central-1 or il-central-1 or af-south-1 or
 ap-east-1 or ap-northeast-1 or ap-northeast-2 or ap-southeast-1 or ap-
southeast-2 or ap-south-1 or
                      cn-north-1 or cn-northwest-1 or
                      us-gov-east-1 or us-gov-west-1
```

Neptune バージョン 4 署名者はデフォルトの認証情報プロバイダチェーンを使用します。認証情報の提供の追加メソッドについては、AWS SDK for Java ディベロッパーガイドのデフォルトの認証情報プロバイダーチェーンの使用を参照してください。
認証情報ファイルを使用する場合でも、SERVICE_REGION 変数は必要です。

5. .yaml ファイルを使用して :remote 接続を確立します。

```
:remote connect tinkerpop.server conf/neptune-remote.yaml
```

6. 次のコマンドを入力してリモートモードに切り替え、すべての Gremlin クエリをリモート接続に送信します。

```
:remote console
```

Java と Gremlin でバージョン 4 署名を使用して Neptune に接続する

TinkerPop 3.4.11 以降を使用して Sig4 署名で Neptune に接続する

TinkerPop 3.4.11 以降を使用する場合に、Sig4 署名で Gremlin Java API を使用して Neptune に接続する方法の例を示します (Maven の使用に関する一般的な知識を前提としています)。まず、pom.xml ファイルの一部として依存関係を定義します。

```
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>amazon-neptune-sigv4-signer</artifactId>
  <version>2.4.0</version>
</dependency>
```

それから、次のようなコードを使用します。

```
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.neptune.auth.NeptuneNettyHttpSigV4Signer;
import com.amazonaws.neptune.auth.NeptuneSigV4SignerException;

 ...

System.setProperty("aws.accessKeyId","your-access-key");
System.setProperty("aws.secretKey","your-secret-key");

 ...

Cluster cluster = Cluster.build((your cluster))
                 .enableSsl(true)
                 .handshakeInterceptor( r ->
                  {
                    try {
                      NeptuneNettyHttpSigV4Signer sigV4Signer =
                        new NeptuneNettyHttpSigV4Signer("(your region)", new
 DefaultAWSCredentialsProviderChain());
                      sigV4Signer.signRequest(r);
                    } catch (NeptuneSigV4SignerException e) {
                      throw new RuntimeException("Exception occurred while signing the
 request", e);
                    }
                    return r;
                  }
                 ).create();
try {
  Client client = cluster.connect();
  client.submit("g.V().has('code','IAD')").all().get();
} catch (Exception e) {
  throw new RuntimeException("Exception occurred while connecting to cluster", e);
}
```

> **Note**
>
> 3.4.11 からアップグレードする場合は、amazon-neptune-gremlin-java-sigv4 ライブラリへの参照を削除してください。上の例に示されているように、handshakeInterceptor() を使用するときには不要になります。handshakeInterceptor() をちゃねライザー

(SigV4WebSocketChannelizer.class) と組み合わせて使用しようとしないでください。エラーが発生します。

3.4.11 より前のバージョンの TinkerPop を使用して Sig4 署名で Neptune に接続する

TinkerPop より前のバージョンでは、前のセクションで示したhandshakeInterceptor()設定がサポートされ3.4.11ていないため、 amazon-neptune-gremlin-java-sigv4パッケージに依存する必要があります。これは SigV4WebSocketChannelizer クラスを含む Neptune ライブラリで、標準の TinkerPop Channelizer を SigV4 署名を自動的に挿入できるものに置き換えます。amazon-neptune-gremlin-java-sigv4 ライブラリは非推奨であるため、可能な場合は TinkerPop 3.4.11 以上にアップグレードしてください。

以下は、3.4.11 より前の TinkerPop バージョンを使用する場合に、Sig4 署名付きの Gremlin Java API を使用して Neptune に接続する方法の例です (Maven の使用方法に関する一般的な知識を前提としています)。

まず、pom.xml ファイルの一部として依存関係を定義します。

```
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>amazon-neptune-gremlin-java-sigv4</artifactId>
  <version>2.4.0</version>
</dependency>
```

上記の依存関係には Gremlin ドライバーのバージョン 3.4.10 が含まれます。より新しい Gremlin ドライバーバージョン (3.4.13 まで) を使用することも可能ですが、3.4.10 以降にドライバーをアップグレードする場合は、上記の handshakeInterceptor() モデルを使用するように変更する必要があります。

その後、gremlin-driver Cluster オブジェクトを Java コードで次のように設定する必要があります。

```
import org.apache.tinkerpop.gremlin.driver.SigV4WebSocketChannelizer;

 ...

Cluster cluster = Cluster.build(your cluster)
                      .enableSsl(true)
                      .channelizer(SigV4WebSocketChannelizer.class)
```

```
                        .create();
Client client = cluster.connect();
client.submit("g.V().has('code','IAD')").all().get();
```

Java と SPARQL でバージョン 4 署名を使用して Neptune に接続する (RDF4J および Jena)

このセクションでは、RDF4J または Apache Jena で署名バージョン 4 認証を使用して Neptune に接続する方法を示します。

前提条件

- Java 8 以上
- Apache Maven 3.3 以上

  Amazon Linux を実行している EC2 インスタンスに前提条件となっているこれらのソフトウェアをインストールする方法については、「Amazon Linux EC2 の前提条件」を参照してください。
- リクエストに署名するための IAM 認証情報。詳細については、AWS SDK for Java デベロッパーガイドのデフォルトの認証情報プロバイダーチェーンの使用を参照してください。

  > **Note**
  >
  > 一時的なセキュリティ認証情報を使用している場合は、セッショントークンを含めて、指定した期間が過ぎると失効します。
  > 新しい認証情報をリクエストするときは、セッショントークンを更新する必要があります。詳細については、IAM ユーザーガイドの「一時的なセキュリティ認証情報を使用して AWS リソースへのアクセスをリクエストする」を参照してください。
- SERVICE_REGION 変数を次のいずれかに設定し、Neptune DB インスタンスのリージョンを指定します。
  - 米国東部 (バージニア北部): us-east-1
  - 米国東部 (オハイオ): us-east-2
  - 米国西部 (北カリフォルニア): us-west-1
  - 米国西部 (オレゴン): us-west-2
  - カナダ (中部): ca-central-1
  - 南米 (サンパウロ): sa-east-1
  - 欧州 (ストックホルム): eu-north-1

- 欧州 (アイルランド): eu-west-1
- 欧州 (ロンドン): eu-west-2
- 欧州 (パリ): eu-west-3
- 欧州 (フランクフルト): eu-central-1
- 中東 (バーレーン): me-south-1
- 中東 (アラブ首長国連邦): me-central-1
- イスラエル (テルアビブ): il-central-1
- アフリカ (ケープタウン): af-south-1
- アジアパシフィック (香港): ap-east-1
- アジアパシフィック (東京): ap-northeast-1
- アジアパシフィック (ソウル): ap-northeast-2
- アジアパシフィック (大阪): ap-northeast-3
- アジアパシフィック (シンガポール): ap-southeast-1
- アジアパシフィック (シドニー): ap-southeast-2
- アジアパシフィック (ムンバイ): ap-south-1
- 中国 (北京): cn-north-1
- 中国 (寧夏): cn-northwest-1
- AWS GovCloud （米国西部): us-gov-west-1
- AWS GovCloud （米国東部): us-gov-east-1

RDF4J または Apache Jena でバージョン 4 署名を使用して Neptune に接続するには

1. からサンプルリポジトリのクローンを作成します GitHub。

```
git clone https://github.com/aws/amazon-neptune-sparql-java-sigv4.git
```

2. 複製したディレクトリに変更します。

```
cd amazon-neptune-sparql-java-sigv4
```

3. 最新のタグブランチをチェックアウトして、プロジェクトの最新バージョンを取得します。

```
git checkout $(git describe --tags `git rev-list --tags --max-count=1`)
```

4. 以下のいずれかのコマンドを入力して、サンプルコードをコンパイルして実行します。

   *your-neptune-endpoint*を Neptune DB インスタンスのホスト名または IP アドレスで置き換えます。デフォルトのポート番号は 8182 です。

   > **Note**
   >
   > Neptune DB インスタンスのホスト名を見つける方法については、Amazon Neptune エンドポイントに接続する セクションを参照してください。

   Eclipse RDF4J

   以下のように入力して、RDF4J の例を実行します。

   ```
   mvn compile exec:java \
       -Dexec.mainClass="com.amazonaws.neptune.client.rdf4j.NeptuneRdf4JSigV4Example"
    \
       -Dexec.args="https://your-neptune-endpoint:port"
   ```

   Apache Jena

   Apache Jena 例を実行する次のコマンドを入力します。

   ```
   mvn compile exec:java \
       -Dexec.mainClass="com.amazonaws.neptune.client.jena.NeptuneJenaSigV4Example" \
       -Dexec.args="https://your-neptune-endpoint:port"
   ```

5. この例のソースコードを表示するには、`src/main/java/com/amazonaws/neptune/client/` ディレクトリにある例を参照してください。

独自の Java アプリケーションで SigV4 署名ドライバーを使用するには、`pom.xml` の `<dependencies>` セクションに `amazon-neptune-sigv4-signer` Maven パッケージを追加します。これらの例を出発点として使用することをお勧めします。

SPARQL と Node.js でバージョン 4 署名を使用して Neptune に接続する

Signature V4 署名と AWS SDK for Javascript V3 を使用したクエリ

署名バージョン 4 認証と AWS SDK for Javascript V3 を使用して Node.js を使用して Neptune SPARQL に接続する方法の例を次に示します。

```
const { HttpRequest }  = require('@smithy/protocol-http');
const { fromNodeProviderChain } = require('@aws-sdk/credential-providers');
const { SignatureV4 } = require('@smithy/signature-v4');
const { Sha256 } = require('@aws-crypto/sha256-universal');
const https = require('https');

var region = 'us-west-2'; // e.g. us-west-1
var neptune_endpoint = 'your-Neptune-cluster-endpoint';  // like: 'cluster-
id.region.neptune.amazonaws.com'
var query = `query=PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX class: <http://aws.amazon.com/neptune/csv2rdf/class/>
PREFIX resource: <http://aws.amazon.com/neptune/csv2rdf/resource/>
PREFIX prop: <http://aws.amazon.com/neptune/csv2rdf/datatypeProperty/>
PREFIX objprop: <http://aws.amazon.com/neptune/csv2rdf/objectProperty/>

SELECT ?movies ?title WHERE {
  ?jel prop:name "James Earl Jones" .
  ?movies ?p2 ?jel .
  ?movies prop:title ?title
} LIMIT 10`;

runQuery(query);

function runQuery(q) {
  var request = new HttpRequest({
    hostname: neptune_endpoint,
    port: 8182,
    path: 'sparql',
    body: encodeURI(query),
    headers: {
      'Content-Type': 'application/x-www-form-urlencoded',
      'host': neptune_endpoint + ':8182',
    },
    method: 'POST',
  });
```

```
  const credentialProvider = fromNodeProviderChain();
  let credentials = credentialProvider();
  credentials.then(
    (cred)=>{
      var signer = new SignatureV4({credentials: cred, region: region, sha256: Sha256,
 service: 'neptune-db'});
      signer.sign(request).then(
        (req)=>{
          var responseBody = '';
          var sendreq = https.request(
            {
              host: req.hostname,
              port: req.port,
              path: req.path,
              method: req.method,
              headers: req.headers,
            },
          (res) => {
            res.on('data', (chunk) => { responseBody += chunk; });
            res.on('end', () => {
                console.log(JSON.parse(responseBody));
            });
          });
          sendreq.write(req.body);
          sendreq.end();
        }
      );
    },
    (err)=>{
      console.error(err);
    }
  );
}
```

Signature V4 署名と AWS SDK for Javascript V2 を使用したクエリ

署名バージョン 4 認証と AWS SDK for Javascript V2 を使用して Node.js を使用して Neptune SPARQL に接続する方法の例を次に示します。

```
var AWS = require('aws-sdk');

var region = 'us-west-2'; // e.g. us-west-1
```

```
var neptune_endpoint = 'your-Neptune-cluster-endpoint';  // like: 'cluster-
id.region.neptune.amazonaws.com'
var query = `query=PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX class: <http://aws.amazon.com/neptune/csv2rdf/class/>
PREFIX resource: <http://aws.amazon.com/neptune/csv2rdf/resource/>
PREFIX prop: <http://aws.amazon.com/neptune/csv2rdf/datatypeProperty/>
PREFIX objprop: <http://aws.amazon.com/neptune/csv2rdf/objectProperty/>

SELECT ?movies ?title WHERE {
    ?jel prop:name "James Earl Jones" .
    ?movies ?p2 ?jel .
    ?movies prop:title ?title
} LIMIT 10`;

runQuery(query);

function runQuery(q) {

    var endpoint = new AWS.Endpoint(neptune_endpoint);
    endpoint.port = 8182;
    var request = new AWS.HttpRequest(endpoint, region);
    request.path += 'sparql';
    request.body = encodeURI(query);
    request.headers['Content-Type'] = 'application/x-www-form-urlencoded';
    request.headers['host'] = neptune_endpoint;
    request.method = 'POST';

    var credentials = new AWS.CredentialProviderChain();
    credentials.resolve((err, cred)=>{
        var signer = new AWS.Signers.V4(request, 'neptune-db');
        signer.addAuthorization(cred, new Date());
    });

    var client = new AWS.HttpClient();
    client.handleRequest(request, null, function(response) {
        console.log(response.statusCode + ' ' + response.statusMessage);
        var responseBody = '';
        response.on('data', function (chunk) {
            responseBody += chunk;
        });
        response.on('end', function (chunk) {
            console.log('Response body: ' + responseBody);
        });
    }, function(error) {
```

```
        console.log('Error: ' + error);
    });
}
```

例: Python で署名バージョン 4 署名を使用して Neptune に接続

このセクションでは、署名バージョン 4 を使用して Amazon Neptune に接続する方法について、Python で記述されたプログラム例を挙げて説明します。この例は、https://docs.aws.amazon.com/general/latest/gr/sigv4-signed-request-examples.html のセクション「Amazon Web Services 全般のリファレンス署名バージョン 4 の署名プロセス」内の例に基づいています。

このプログラム例を使用するには、以下が必要です。

- コンピュータにインストール済みの Python 3.x。Python のサイトから入手できます。これらのプログラムは、Python 3.6 でテスト済みです。
- Python のリクエストライブラリ。これは、ウェブリクエストを作成するスクリプト例で使用します。Python パッケージをインストールする便利な方法は、`pip` を使用することです。これは、Python パッケージインデックスサイトからパッケージを取得します。その後で、コマンドラインから `requests` を実行すると、`pip install requests` をインストールすることができます。
- `AWS_ACCESS_KEY_ID` および `AWS_SECRET_ACCESS_KEY` という名前の環境変数で指定したアクセスキー (アクセスキー ID およびシークレットアクセスキー) 。ベストプラクティスとして、認証情報をコードに埋め込まないことをお勧めします。詳細については、AWS Account Management リファレンスガイドの「AWS アカウントのベストプラクティス」を参照してください。

  `SERVICE_REGION` という環境変数の Neptune DB クラスターのリージョン。

  一時的なセキュリティ認証情報を使用している場合は、`AWS_ACCESS_KEY_ID`、`AWS_SECRET_ACCESS_KEY`、`SERVICE_REGION` に加えて `AWS_SESSION_TOKEN` を指定する必要があります。

> **Note**
>
> 一時的なセキュリティ認証情報を使用している場合は、セッショントークンを含めて、指定した期間が過ぎると失効します。

新しい認証情報をリクエストするときは、セッショントークンを更新する必要があります。詳細については、一時的なセキュリティ認証情報を使用して AWS リソースへのアクセスをリクエストするをご参照ください。

以下の例では、Python を使用して Neptune に署名付きリクエストを行う方法を示します。このリクエストは、GET リクエストまたは POST リクエストのいずれかを行います。認証情報は、`Authorization` リクエストヘッダーを使用して渡されます。

この例では、 AWS Lambda 関数としても機能します。詳細については、「the section called "Lambda のセットアップ"」を参照してください。

Gremlin および SPARQL Neptune エンドポイントへの署名付きリクエストを行うには

1. `neptunesigv4.py` という名前のファイルを作成し、テキストエディタで開きます。
2. 以下のテキストを `neptunesigv4.py` ファイルにコピーしてペーストしてください。

```
# Amazon Neptune version 4 signing example (version v3)

# The following script requires python 3.6+
#    (sudo yum install python36 python36-virtualenv python36-pip)
# => the reason is that we're using urllib.parse() to manually encode URL
#    parameters: the problem here is that SIGV4 encoding requires whitespaces
#    to be encoded as %20 rather than not or using '+', as done by previous/
#    default versions of the library.


# See: https://docs.aws.amazon.com/general/latest/gr/sigv4_signing.html
import sys, datetime, hashlib, hmac
import requests  # pip3 install requests
import urllib
import os
import json
from botocore.auth import SigV4Auth
from botocore.awsrequest import AWSRequest
from botocore.credentials import ReadOnlyCredentials
from types import SimpleNamespace
from argparse import RawTextHelpFormatter
from argparse import ArgumentParser

# Configuration. https is required.
```

```
protocol = 'https'

# The following lines enable debugging at httplib level (requests->urllib3-
>http.client)
# You will see the REQUEST, including HEADERS and DATA, and RESPONSE with HEADERS
 but without DATA.
#
# The only thing missing will be the response.body which is not logged.
#
# import logging
# from http.client import HTTPConnection
# HTTPConnection.debuglevel = 1
# logging.basicConfig()
# logging.getLogger().setLevel(logging.DEBUG)
# requests_log = logging.getLogger("requests.packages.urllib3")
# requests_log.setLevel(logging.DEBUG)
# requests_log.propagate = True


# Read AWS access key from env. variables. Best practice is NOT
# to embed credentials in code.
access_key = os.getenv('AWS_ACCESS_KEY_ID', '')
secret_key = os.getenv('AWS_SECRET_ACCESS_KEY', '')
region = os.getenv('SERVICE_REGION', '')

# AWS_SESSION_TOKEN is optional environment variable. Specify a session token only
 if you are using temporary
# security credentials.
session_token = os.getenv('AWS_SESSION_TOKEN', '')

### Note same script can be used for AWS Lambda (runtime = python3.6).
## Steps to use this python script for AWS Lambda
# 1. AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY and AWS_SESSION_TOKEN and AWS_REGION
 variables are already part of Lambda's Execution environment
#    No need to set them up explicitly.
# 3. Create Lambda deployment package https://docs.aws.amazon.com/lambda/latest/dg/
lambda-python-how-to-create-deployment-package.html
# 4. Create a Lambda function in the same VPC and assign an IAM role with neptune
 access

def lambda_handler(event, context):
    # sample_test_input = {
    #     "host": "END_POINT:8182",
    #     "method": "GET",
```

```
    #     "query_type": "gremlin",
    #     "query": "g.V().count()"
    # }

    # Lambda uses AWS_REGION instead of SERVICE_REGION
    global region
    region = os.getenv('AWS_REGION', '')

    host = event['host']
    method = event['method']
    query_type = event['query_type']
    query =  event['query']

    return make_signed_request(host, method, query_type, query)

def validate_input(method, query_type):
    # Supporting GET and POST for now:
    if (method != 'GET' and method != 'POST'):
        print('First parameter must be "GET" or "POST", but is "' + method + '".')
        sys.exit()

    # SPARQL UPDATE requires POST
    if (method == 'GET' and query_type == 'sparqlupdate'):
        print('SPARQL UPDATE is not supported in GET mode. Please choose POST.')
        sys.exit()

def get_canonical_uri_and_payload(query_type, query, method):
    # Set the stack and payload depending on query_type.
    if (query_type == 'sparql'):
        canonical_uri = '/sparql/'
        payload = {'query': query}

    elif (query_type == 'sparqlupdate'):
        canonical_uri = '/sparql/'
        payload = {'update': query}

    elif (query_type == 'gremlin'):
        canonical_uri = '/gremlin/'
        payload = {'gremlin': query}
        if (method == 'POST'):
            payload = json.dumps(payload)

    elif (query_type == 'openCypher'):
        canonical_uri = '/openCypher/'
```

```
        payload = {'query': query}

    elif (query_type == "loader"):
        canonical_uri = "/loader/"
        payload = query

    elif (query_type == "status"):
        canonical_uri = "/status/"
        payload = {}

    elif (query_type == "gremlin/status"):
        canonical_uri = "/gremlin/status/"
        payload = {}

    elif (query_type == "openCypher/status"):
        canonical_uri = "/openCypher/status/"
        payload = {}

    elif (query_type == "sparql/status"):
        canonical_uri = "/sparql/status/"
        payload = {}

    else:
        print(
            'Third parameter should be from ["gremlin", "sparql", "sparqlupdate",
 "loader", "status] but is "' + query_type + '".')
        sys.exit()
    ## return output as tuple
    return canonical_uri, payload

def make_signed_request(host, method, query_type, query):
    service = 'neptune-db'
    endpoint = protocol + '://' + host

    print()
    print('+++++ USER INPUT +++++')
    print('host = ' + host)
    print('method = ' + method)
    print('query_type = ' + query_type)
    print('query = ' + query)

    # validate input
    validate_input(method, query_type)
```

```
    # get canonical_uri and payload
    canonical_uri, payload = get_canonical_uri_and_payload(query_type, query,
method)

    # assign payload to data or params
    data = payload if method == 'POST' else None
    params = payload if method == 'GET' else None

    # create request URL
    request_url = endpoint + canonical_uri

    # create and sign request
    creds = SimpleNamespace(
        access_key=access_key, secret_key=secret_key, token=session_token,
region=region,
    )

    request = AWSRequest(method=method, url=request_url, data=data, params=params)
    SigV4Auth(creds, service, region).add_auth(request)

    r = None

    # ************* SEND THE REQUEST *************
    if (method == 'GET'):

        print('++++ BEGIN GET REQUEST +++++')
        print('Request URL = ' + request_url)
        r = requests.get(request_url, headers=request.headers, verify=False,
params=params)

    elif (method == 'POST'):

        print('\n+++++ BEGIN POST REQUEST +++++')
        print('Request URL = ' + request_url)
        if (query_type == "loader"):
            request.headers['Content-type'] = 'application/json'
        r = requests.post(request_url, headers=request.headers, verify=False,
data=data)

    else:
        print('Request method is neither "GET" nor "POST", something is wrong
here.')

    if r is not None:
```

```
        print()
        print('+++++ RESPONSE +++++')
        print('Response code: %d\n' % r.status_code)
        response = r.text
        r.close()
        print(response)

        return response

help_msg = '''
    export AWS_ACCESS_KEY_ID=[MY_ACCESS_KEY_ID]
    export AWS_SECRET_ACCESS_KEY=[MY_SECRET_ACCESS_KEY]
    export AWS_SESSION_TOKEN=[MY_AWS_SESSION_TOKEN]
    export SERVICE_REGION=[us-east-1|us-east-2|us-west-2|eu-west-1]

    python version >=3.6 is required.

    Examples: For help
    python3 program_name.py -h

    Examples: Queries
    python3 program_name.py -ho your-neptune-endpoint -p 8182 -a GET -q status
    python3 program_name.py -ho your-neptune-endpoint -p 8182 -a GET -q sparql/
status
    python3 program_name.py -ho your-neptune-endpoint -p 8182 -a GET -q sparql -d
 "SELECT ?s WHERE { ?s ?p ?o }"
    python3 program_name.py -ho your-neptune-endpoint -p 8182 -a POST -q sparql -d
 "SELECT ?s WHERE { ?s ?p ?o }"
    python3 program_name.py -ho your-neptune-endpoint -p 8182 -a POST -q
 sparqlupdate -d "INSERT DATA { <https://s> <https://p> <https://o> }"
    python3 program_name.py -ho your-neptune-endpoint -p 8182 -a GET -q gremlin/
status
    python3 program_name.py -ho your-neptune-endpoint -p 8182 -a GET -q gremlin -d
 "g.V().count()"
    python3 program_name.py -ho your-neptune-endpoint -p 8182 -a POST -q gremlin -d
 "g.V().count()"
    python3 program_name.py -ho your-neptune-endpoint -p 8182 -a GET -q openCypher/
status
    python3 program_name.py -ho your-neptune-endpoint -p 8182 -a GET -q openCypher
 -d "MATCH (n1) RETURN n1 LIMIT 1;"
    python3 program_name.py -ho your-neptune-endpoint -p 8182 -a POST -q openCypher
 -d "MATCH (n1) RETURN n1 LIMIT 1;"
    python3 program_name.py -ho your-neptune-endpoint -p 8182 -a GET -q loader -d
 '{"loadId": "68b28dcc-8e15-02b1-133d-9bd0557607e6"}'
```

```
    python3 program_name.py -ho your-neptune-endpoint -p 8182 -a GET -q loader -d
 '{}'
    python3 program_name.py -ho your-neptune-endpoint -p 8182 -a POST -q loader
 -d '{"source": "source", "format" : "csv", "failOnError": "fail_on_error",
 "iamRoleArn": "iam_role_arn", "region": "region"}'

    Environment variables must be defined as AWS_ACCESS_KEY_ID,
 AWS_SECRET_ACCESS_KEY and SERVICE_REGION.
    You should also set AWS_SESSION_TOKEN environment variable if you are using
 temporary credentials (ex. IAM Role or EC2 Instance profile).

    Current Limitations:
    - Query mode "sparqlupdate" requires POST (as per the SPARQL 1.1 protocol)
            '''

def exit_and_print_help():
    print(help_msg)
    exit()

def parse_input_and_query_neptune():


    parser = ArgumentParser(description=help_msg,
 formatter_class=RawTextHelpFormatter)
    group_host = parser.add_mutually_exclusive_group()
    group_host.add_argument("-ho", "--host", type=str)
    group_port = parser.add_mutually_exclusive_group()
    group_port.add_argument("-p", "--port", type=int, help="port ex. 8182,
 default=8182", default=8182)
    group_action = parser.add_mutually_exclusive_group()
    group_action.add_argument("-a", "--action", type=str, help="http action,
 default = GET", default="GET")
    group_endpoint = parser.add_mutually_exclusive_group()
    group_endpoint.add_argument("-q", "--query_type", type=str, help="query_type,
 default = status ", default="status")
    group_data = parser.add_mutually_exclusive_group()
    group_data.add_argument("-d", "--data", type=str, help="data required for the
 http action", default="")

    args = parser.parse_args()
    print(args)

    # Read command line parameters
    host = args.host
```

```
    port = args.port
    method = args.action
    query_type = args.query_type
    query = args.data

    if (access_key == ''):
        print('!!! ERROR: Your AWS_ACCESS_KEY_ID environment variable is
 undefined.')
        exit_and_print_help()

    if (secret_key == ''):
        print('!!! ERROR: Your AWS_SECRET_ACCESS_KEY environment variable is
 undefined.')
        exit_and_print_help()

    if (region == ''):
        print('!!! ERROR: Your SERVICE_REGION environment variable is undefined.')
        exit_and_print_help()

    if host is None:
        print('!!! ERROR: Neptune DNS is missing')
        exit_and_print_help()

    host = host + ":" + str(port)
    make_signed_request(host, method, query_type, query)


if __name__ == "__main__":
    parse_input_and_query_neptune()
```

3. ターミナルで、neptunesigv4.py ファイルの場所に移動します。
4. 以下のコマンドを入力して、アクセスキー、シークレットキー、リージョンを正しい値に置き換えます。

```
export AWS_ACCESS_KEY_ID=MY_ACCESS_KEY_ID
export AWS_SECRET_ACCESS_KEY=MY_SECRET_ACCESS_KEY
export SERVICE_REGION=us-east-1 or us-east-2 or us-west-1 or us-west-2 or ca-
central-1 or
                      sa-east-1 or eu-north-1 or eu-west-1 or eu-west-2 or eu-
west-3 or eu-central-1 or me-south-1 or
                      me-central-1 or il-central-1 or af-south-1 or ap-east-1 or
 ap-northeast-1 or ap-northeast-2 or ap-southeast-1 or ap-southeast-2 or ap-south-1
 or
```

```
                    cn-north-1 or cn-northwest-1 or
                    us-gov-east-1 or us-gov-west-1
```

一時的なセキュリティ認証情報を使用している場合は、AWS_ACCESS_KEY_ID、AWS_SECRET_ACCESS_KEY、SERVICE_REGION に加えて AWS_SESSION_TOKEN を指定する必要があります。

```
export AWS_SESSION_TOKEN=MY_AWS_SESSION_TOKEN
```

> **Note**
>
> 一時的なセキュリティ認証情報を使用している場合は、セッショントークンを含めて、指定した期間が過ぎると失効します。
> 新しい認証情報をリクエストするときは、セッショントークンを更新する必要があります。詳細については、一時的なセキュリティ認証情報を使用して AWS リソースへのアクセスをリクエストするをご参照ください。

5. 以下のコマンドのいずれかを入力して、Neptune DB インスタンスに署名付きリクエストを送信します。これらの例では Python バージョン 3.6 を使用します。

   エンドポイントのステータス

   ```
   python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a GET -q status
   ```

   Gremlin

   ```
   python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a GET -q gremlin -d
    "g.V().count()"

   python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a POST -q gremlin -d
    "g.V().count()"
   ```

   Gremlin ステータス

   ```
   python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a GET -q gremlin/
   status
   ```

   SPARQL

```
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a GET -q sparql -d
 "SELECT ?s WHERE { ?s ?p ?o }"
```

SPARQL UPDATE

```
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a POST -q sparqlupdate
 -d "INSERT DATA { <https://s> <https://p> <https://o> }"
```

SPARQL ステータス

```
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a GET -q sparql/status
```

openCypher

```
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a GET -q openCypher -d
 "MATCH (n1) RETURN n1 LIMIT 1;"

python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a POST -q openCypher -
d "MATCH (n1) RETURN n1 LIMIT 1;"
```

OpenCypherのステータス

```
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a GET -q openCypher/
status
```

[ローダー]

```
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a GET -q loader -d
 '{"loadId": "68b28dcc-8e15-02b1-133d-9bd0557607e6"}'

python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a GET -q loader -d
 '{}'

python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a POST -q loader
 -d '{"source": "source", "format" : "csv", "failOnError": "fail_on_error",
 "iamRoleArn": "iam_role_arn", "region": "region"}'
```

6. 以下に示しているのは、Python スクリプトを実行するための構文です。

```
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p port -a GET|POST -q gremlin|
sparql|sparqlupdate|loader|status -d "string0data"
```

SPARQL UPDATE では POST が必要です。

ポリシーを使用したアクセスの管理

IAM ポリシーは、アクションとリソースを使用するアクセス許可を定義する JSON オブジェクトです。

でアクセスを制御する AWS には、ポリシーを作成し、 AWS ID またはリソースにアタッチします。ポリシーは、アイデンティティまたはリソースに関連付けられているときにアクセス許可を定義する のオブジェクトです。 は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) AWS がリクエストを行うときに、これらのポリシー AWS を評価します。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。ほとんどのポリシーは JSON ドキュメント AWS として に保存されます。JSON ポリシードキュメントの構造と内容の詳細については、「IAM ユーザーガイド」の「JSON ポリシー概要」を参照してください。

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き継ぐことができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの許可を定義します。例えば、`iam:GetRole` アクションを許可するポリシーがあるとします。そのポリシーを持つユーザーは、 AWS Management Console、、 AWS CLIまたは AWS API からロール情報を取得できます。

アイデンティティベースのポリシー

アイデンティティベースポリシーは、IAM ユーザー、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 権限ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「IAM ポリシーの作成」を参照してください。

アイデンティティベースのポリシーは、さらにインラインポリシーまたはマネージドポリシーに分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれています。管理ポリシーは、 内の複数のユーザー、グループ、ロールにアタッチできるスタンドアロンポリシーです AWS アカウント。管理ポリシーには、 AWS 管理ポリシーとカスタマー管理ポリシーが含まれます。マネージドポリシーまたはインラインポリシーのいずれかを選択する方法については、「IAM ユーザーガイド」の「マネージドポリシーとインラインポリシーの比較」を参照してください。

AWS 組織でのサービスコントロールポリシー (SCP) の使用

サービスコントロールポリシー (SCPsは、 の組織または組織単位 (OU) に対する最大アクセス許可を指定する JSON ポリシーですAWS Organizations。 AWS Organizations は、ビジネスが所有する複数の AWS アカウントをグループ化して一元管理するサービスです。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウントに適用できます。SCP は、各アカウントのルートユーザーを含む、メンバー AWS アカウントのエンティティのアクセス許可を制限します。Organizations と SCPs」のSCPs」を参照してください。 AWS Organizations

Amazon Neptune を AWS 組織内の AWS アカウントにデプロイするお客様は、SCPs を活用して Neptune を使用できるアカウントを制御できます。メンバーアカウント内で Neptune に確実にアクセスできるようにするには、`neptune:*` と `neptune-db:*` をそれぞれ使用することによって、コントロールプレーンとデータプレーンの両方の IAM アクションへのアクセスを許可してください。

Amazon Neptune コンソールを使用するために必要なアクセス許可

Amazon Neptune コンソールを使用するユーザーには、最小限のアクセス権限のセットが必要です。これらのアクセス許可により、ユーザーは AWS アカウントの Neptune リソースを記述し、Amazon EC2 セキュリティやネットワーク情報など、その他の関連情報を提供できます。

これらの最小限必要なアクセス許可よりも制限された IAM ポリシーを作成している場合、その IAM ポリシーを使用するユーザーに対してコンソールは意図したとおりには機能しません。それらのユーザーが Neptune コンソールを引き続き使用するためには、`NeptuneReadOnlyAccess` で説明しているように、AWS Amazon Neptune の マネージド (事前定義) ポリシー 管理ポリシーをユーザーにもアタッチします。

AWS CLI または Amazon Neptune API のみを呼び出すユーザーには、最小限のコンソールアクセス許可を付与する必要はありません。

IAM ポリシーを IAM ユーザーにアタッチする

管理ポリシーまたはカスタムポリシーを適用するには、IAM ユーザーにアタッチします。このトピックに関するチュートリアルについては、IAM ユーザーガイドのはじめてのカスタマー管理ポリシーの作成とアタッチを参照してください。

チュートリアルを進める際に、このセクションに記載されているいずれかのポリシー例をスタート点として使用し、ニーズに合わせて調整することができます。チュートリアルを完了すると、`neptune-db:*` アクションの使用を許可するポリシーが IAM ユーザーにアタッチされます。

> **Important**
>
> - IAM ポリシーへの変更は、指定された Neptune リソースへの適用に最大で 10 分かかります。
> - Neptune DB クラスターに適用された IAM ポリシーは、そのクラスター内のすべてのインスタンスに適用されます。

Neptune へのアクセスを制御するためのさまざまな種類の IAM ポリシーの使用

Neptune の管理アクションまたは Neptune DB クラスター内のデータへのアクセスを提供するには、IAM ユーザーまたはロールにポリシーをアタッチします。IAM ポリシーをユーザーにアタッチする方法については、「IAM ポリシーを IAM ユーザーにアタッチする」を参照してください。ロールにポリシーをアタッチする方法については、IAM ポリシーの追加と削除のIAM ユーザーガイドを参照してください。

Neptune への一般的なアクセスには、Neptune の管理ポリシーのいずれかを使用できます。アクセスをさらに制限したい場合は、Neptune がサポートする管理アクションとリソースを使用して独自のカスタムポリシーを作成できます。

カスタム IAM ポリシーでは、Neptune DB クラスターへのさまざまなアクセスモードを制御する 2 種類のポリシーステートメントを使用できます。

- 管理ポリシーステートメント — 管理ポリシーステートメントは、DB クラスターとそのインスタンスの作成、設定、管理に使用する Neptune 管理 API へのアクセスを提供します。

Neptune は Amazon RDS と機能を共有しているため、Neptune ポリシーの管理アクション、リソース、および条件キーは設計により `rds:` プレフィックスを使用します。

- データアクセスポリシーステートメント — データアクセスポリシーステートメントは、データアクセスアクション、リソース、および条件キーを使用して、DB クラスターに含まれるデータへのアクセスを制御します。

  Neptune データアクセスアクション、リソース、および条件キーは、`neptune-db:` プレフィックスを使用します。

Amazon Neptune での IAM 条件コンテキストキーの使用

Neptune へのアクセスを制御する IAM ポリシーステートメントで条件を指定できます。ポリシーステートメントは、条件が true の場合にのみ有効です。

例えば、特定の日付の後にのみ適用されるポリシーステートメントや、リクエストに特定の値が存在する場合のみアクセスが許可されるポリシーステートメントが必要になる場合があります。

条件を表すには、「等しい」や「より小さい」などの IAM 条件ポリシー演算子 とともに、ポリシーステートメントの `Condition` 要素であらかじめ定義された条件キーを使用します。

1 つのステートメントに複数の `Condition` 要素を指定するか、1 つの `Condition` 要素に複数のキーを指定すると、 AWS は `AND` 論理演算子を使用してそれらを評価します。1 つの条件キーに複数の値を指定すると、 は論理`OR`オペレーションを使用して条件 AWS を評価します。ステートメントの権限が付与される前にすべての条件が満たされる必要があります。

条件を指定する際にプレースホルダー変数も使用できます。例えば IAM ユーザーに、IAM ユーザー名がタグ付けされている場合のみリソースにアクセスできる権限を付与することができます。詳細については、「IAM ユーザーガイド」の「IAM Policy Elements: Variables and Tags」(IAM ポリシーの要素: 変数およびタグ) を参照してください。

条件キーのデータ型によって、リクエスト内の値とポリシーステートメント内の値の比較に使用できる条件演算子が決まります。そのデータ型と互換性のない条件演算子を使用した場合、条件は一致しないため、ポリシーステートメントは適用されません。

Neptune は、管理ポリシーステートメントについて、データアクセスポリシーステートメントとは異なる条件キーのセットをサポートしています。

- 管理ポリシーステートメントの条件キー

- データアクセスポリシーステートメントの条件キー

Amazon Neptune での IAM ポリシーとアクセスコントロール機能のサポート

次の表は、Neptune が管理ポリシーステートメントとデータアクセスポリシーステートメントについてサポートする IAM 機能を示しています。

Neptune で使用できる IAM 機能

IAM 機能	管理	データアクセス
アイデンティティベースのポリシー	はい	はい
リソースベースのポリシー	いいえ	いいえ
ポリシーアクション	はい	はい
ポリシーリソース	はい	はい
グローバル条件キー	はい	(サブセット)
タグベースの条件キー	はい	いいえ
アクセスコントロールリスト (ACL)	いいえ	いいえ
サービスコントロールポリシー (SCP)	はい	はい
サービスリンクロール	はい	いいえ

IAM ポリシーの制限

IAM ポリシーへの変更は、指定された Neptune リソースへの適用に最大で 10 分かかります。

Neptune DB クラスターに適用された IAM ポリシーは、そのクラスター内のすべてのインスタンスに適用されます。

Neptune は現在、クロスアカウントアクセスコントロールをサポートしていません。

AWS Amazon Neptune の マネージド (事前定義) ポリシー

AWS は、 によって作成および管理されるスタンドアロン IAM ポリシーを提供することで、多くの一般的なユースケースに対処します AWS。マネージドポリシーは、一般的ユースケースに必要な許可を付与することで、どの許可が必要なのかをユーザーが調査する必要をなくすることができます。詳細については、「IAM ユーザーガイド」の「AWS マネージドポリシー」を参照してください。

アカウントのユーザーにアタッチできる以下の AWS マネージドポリシーは、Amazon Neptune 管理APIsを使用するためのものです。

- NeptuneReadOnlyAccess — ルート AWS アカウントの管理目的とデータアクセス目的の両方で、すべての Neptune リソースへの読み取り専用アクセスを許可します。
- NeptuneFullアクセス — ルート AWS アカウントの管理目的とデータアクセス目的の両方で、すべての Neptune リソースへのフルアクセスを許可します。これは、 AWS CLI または SDK からの完全な Neptune アクセスが必要な場合に推奨されますが、 AWS Management Console アクセスには推奨されません。
- NeptuneConsoleFullAccess — すべての Neptune 管理アクションとリソースへのフルアクセスをルート AWS アカウントで許可しますが、データアクセスアクションやリソースへのフルアクセスは許可しません。これには、IAM と Amazon EC2 (VPC) の制限されたアクセス許可を含め、コンソールからの Neptune アクセスを簡素化する追加のアクセス許可も含まれます。
- NeptuneGraphReadOnlyAccess — すべての Amazon Neptune Analytics リソースへの読み取り専用アクセスと、依存サービスの読み取り専用アクセス許可を提供します。
- AWSServiceRoleForNeptuneGraphPolicy — Neptune Analytics グラフが CloudWatch 運用および使用状況のメトリクスとログを公開できるようにします。

Neptune は特定の管理機能のために Amazon RDS とオペレーション技術を共有するため、Neptune IAM のロールおよびポリシーで Amazon RDS リソースへのアクセスを許可します。これには管理 API アクセス許可が含まれるため、Neptune 管理アクションには `rds:` プレフィックスが付いています。

Neptune AWS 管理ポリシーの更新

次の表は、Neptune がこれらの変更の追跡を開始した時点以降の Neptune 管理ポリシーの更新状況を示しています。

ポリシー	説明	日付
AWS Amazon Neptune の マネージドポリシー - 既存のポリシーの更新	`NeptuneReadOnlyAccess` および `NeptuneFullAccess` マネージドポリシーでは、ポリシーステートメントに識別子として `Sid` (ステートメント ID) が含まれるようになりました。	2024-01-22
NeptuneGraphReadOnlyアクセス (リリース)	Neptune Analytics のグラフとリソースへの読み取り専用アクセスを提供するためにリリースされました。	2023-11-29
AWSServiceRoleForNeptuneGraphPolicy (リリース)	Neptune Analytics グラフが CloudWatch にアクセスして運用および使用状況のメトリクスとログを公開できるようにリリースされました。「Neptune Analytics でのサービスリンクロール (SLR) の使用」を参照してください。	2023-11-29
NeptuneConsoleFullAccess (追加されたアクセス許可)	Neptune Analytics グラフを操作するために必要なすべてのアクセス権を付与するアクセス許可を追加しました。	2023 年 11 月 29 日
NeptuneFullアクセス (追加されたアクセス許可)	データアクセス用のアクセス許可と新しいグローバルデータベース API 用のアクセス許可を追加しました。	2022-07-28
NeptuneConsoleFullAccess (追加されたアクセス許可)	新しいグローバルデータベース API のアクセス許可を追加しました。	2022-07-21

ポリシー	説明	日付
Neptune は変更の追跡を開始しました	Neptune が AWS マネージドポリシーの変更の追跡を開始しました。	2022-07-21

NeptuneReadOnlyAccessAWS 管理ポリシー

以下の NeptuneReadOnlyAccessマネージドポリシーは、管理目的とデータアクセス目的の両方で、すべての Neptune アクションとリソースへの読み取り専用アクセスを許可します。

> **Note**
>
> このポリシーは 2022 年 7 月 21 日に更新され、読み取り専用のデータアクセスアクセス許可と読み取り専用の管理アクセス許可、およびグローバルデータベースアクションのアクセス許可が含まれるようになりました。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowReadOnlyPermissionsForRDS",
            "Effect": "Allow",
            "Action": [
                "rds:DescribeAccountAttributes",
                "rds:DescribeCertificates",
                "rds:DescribeDBClusterParameterGroups",
                "rds:DescribeDBClusterParameters",
                "rds:DescribeDBClusterSnapshotAttributes",
                "rds:DescribeDBClusterSnapshots",
                "rds:DescribeDBClusters",
                "rds:DescribeDBEngineVersions",
                "rds:DescribeDBInstances",
                "rds:DescribeDBLogFiles",
                "rds:DescribeDBParameterGroups",
                "rds:DescribeDBParameters",
                "rds:DescribeDBSubnetGroups",
                "rds:DescribeEventCategories",
                "rds:DescribeEventSubscriptions",
```

```
                "rds:DescribeEvents",
                "rds:DescribeGlobalClusters",
                "rds:DescribeOrderableDBInstanceOptions",
                "rds:DescribePendingMaintenanceActions",
                "rds:DownloadDBLogFilePortion",
                "rds:ListTagsForResource"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowReadOnlyPermissionsForCloudwatch",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:ListMetrics"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowReadOnlyPermissionsForEC2",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeAccountAttributes",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeInternetGateways",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcAttribute",
                "ec2:DescribeVpcs"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowReadOnlyPermissionsForKMS",
            "Effect": "Allow",
            "Action": [
                "kms:ListKeys",
                "kms:ListRetirableGrants",
                "kms:ListAliases",
                "kms:ListKeyPolicies"
            ],
            "Resource": "*"
        },
        {
```

```
            "Sid": "AllowReadOnlyPermissionsForLogs",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams",
                "logs:GetLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:*:*:log-group:/aws/rds/*:log-stream:*",
                "arn:aws:logs:*:*:log-group:/aws/neptune/*:log-stream:*"
            ]
        },
        {
            "Sid": "AllowReadOnlyPermissionsForNeptuneDB",
            "Effect": "Allow",
            "Action": [
                "neptune-db:Read*",
                "neptune-db:Get*",
                "neptune-db:List*"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

NeptuneFullAccessAWS 管理ポリシー

以下のNeptuneFullアクセス管理ポリシーは、管理目的とデータアクセス目的の両方で、すべての Neptune アクションとリソースへのフルアクセスを許可します。 AWS CLI または SDK からのフルアクセスが必要な場合は推奨されますが、 からのフルアクセスは推奨されません AWS Management Console。

> **Note**
>
> このポリシーは 2022 年 7 月 21 日に更新され、フルデータアクセスアクセス許可とフル管理アクセス許可、およびグローバルデータベースアクションのアクセス許可が含まれるようになりました。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowNeptuneCreate",
            "Effect": "Allow",
            "Action": [
                "rds:CreateDBCluster",
                "rds:CreateDBInstance"
            ],
            "Resource": [
                "arn:aws:rds:*:*:*"
            ],
            "Condition": {
                "StringEquals": {
                    "rds:DatabaseEngine": [
                        "graphdb",
                        "neptune"
                    ]
                }
            }
        },
        {
            "Sid": "AllowManagementPermissionsForRDS",
            "Effect": "Allow",
            "Action": [
                "rds:AddRoleToDBCluster",
                "rds:AddSourceIdentifierToSubscription",
                "rds:AddTagsToResource",
                "rds:ApplyPendingMaintenanceAction",
                "rds:CopyDBClusterParameterGroup",
                "rds:CopyDBClusterSnapshot",
                "rds:CopyDBParameterGroup",
                "rds:CreateDBClusterEndpoint",
                "rds:CreateDBClusterParameterGroup",
                "rds:CreateDBClusterSnapshot",
                "rds:CreateDBParameterGroup",
                "rds:CreateDBSubnetGroup",
                "rds:CreateEventSubscription",
                "rds:CreateGlobalCluster",
                "rds:DeleteDBCluster",
                "rds:DeleteDBClusterEndpoint",
                "rds:DeleteDBClusterParameterGroup",
                "rds:DeleteDBClusterSnapshot",
                "rds:DeleteDBInstance",
```

```
                "rds:DeleteDBParameterGroup",
                "rds:DeleteDBSubnetGroup",
                "rds:DeleteEventSubscription",
                "rds:DeleteGlobalCluster",
                "rds:DescribeDBClusterEndpoints",
                "rds:DescribeAccountAttributes",
                "rds:DescribeCertificates",
                "rds:DescribeDBClusterParameterGroups",
                "rds:DescribeDBClusterParameters",
                "rds:DescribeDBClusterSnapshotAttributes",
                "rds:DescribeDBClusterSnapshots",
                "rds:DescribeDBClusters",
                "rds:DescribeDBEngineVersions",
                "rds:DescribeDBInstances",
                "rds:DescribeDBLogFiles",
                "rds:DescribeDBParameterGroups",
                "rds:DescribeDBParameters",
                "rds:DescribeDBSecurityGroups",
                "rds:DescribeDBSubnetGroups",
                "rds:DescribeEngineDefaultClusterParameters",
                "rds:DescribeEngineDefaultParameters",
                "rds:DescribeEventCategories",
                "rds:DescribeEventSubscriptions",
                "rds:DescribeEvents",
                "rds:DescribeGlobalClusters",
                "rds:DescribeOptionGroups",
                "rds:DescribeOrderableDBInstanceOptions",
                "rds:DescribePendingMaintenanceActions",
                "rds:DescribeValidDBInstanceModifications",
                "rds:DownloadDBLogFilePortion",
                "rds:FailoverDBCluster",
                "rds:FailoverGlobalCluster",
                "rds:ListTagsForResource",
                "rds:ModifyDBCluster",
                "rds:ModifyDBClusterEndpoint",
                "rds:ModifyDBClusterParameterGroup",
                "rds:ModifyDBClusterSnapshotAttribute",
                "rds:ModifyDBInstance",
                "rds:ModifyDBParameterGroup",
                "rds:ModifyDBSubnetGroup",
                "rds:ModifyEventSubscription",
                "rds:ModifyGlobalCluster",
                "rds:PromoteReadReplicaDBCluster",
                "rds:RebootDBInstance",
```

```
                "rds:RemoveFromGlobalCluster",
                "rds:RemoveRoleFromDBCluster",
                "rds:RemoveSourceIdentifierFromSubscription",
                "rds:RemoveTagsFromResource",
                "rds:ResetDBClusterParameterGroup",
                "rds:ResetDBParameterGroup",
                "rds:RestoreDBClusterFromSnapshot",
                "rds:RestoreDBClusterToPointInTime",
                "rds:StartDBCluster",
                "rds:StopDBCluster"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "AllowOtherDepedentPermissions",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:ListMetrics",
                "ec2:DescribeAccountAttributes",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcAttribute",
                "ec2:DescribeVpcs",
                "kms:ListAliases",
                "kms:ListKeyPolicies",
                "kms:ListKeys",
                "kms:ListRetirableGrants",
                "logs:DescribeLogStreams",
                "logs:GetLogEvents",
                "sns:ListSubscriptions",
                "sns:ListTopics",
                "sns:Publish"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "AllowPassRoleForNeptune",
            "Effect": "Allow",
```

```
            "Action": "iam:PassRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:passedToService": "rds.amazonaws.com"
                }
            }
        },
        {
            "Sid": "AllowCreateSLRForNeptune",
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
AWSServiceRoleForRDS",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "rds.amazonaws.com"
                }
            }
        },
        {
            "Sid": "AllowDataAccessForNeptune",
            "Effect": "Allow",
            "Action": [
                "neptune-db:*"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

NeptuneConsoleFullAccessAWS 管理ポリシー

以下の NeptuneConsoleFullAccessマネージドポリシーは、管理目的ですべての Neptune アクションとリソースへのフルアクセスを許可しますが、データアクセス目的では許可しません。これには、IAM と Amazon EC2 (VPC) の制限されたアクセス許可を含め、コンソールからの Neptune アクセスを簡素化する追加のアクセス許可も含まれます。

> **Note**
>
> このポリシーは 2023 年 11 月 29 日に更新され、Neptune Analytics グラフを操作するために必要なアクセス許可が含まれるようになりました。
> このポリシーは 2022 年 7 月 21 日に更新され、グローバルデータベースアクションのアクセス許可が含まれるようになりました。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowNeptuneCreate",
      "Effect": "Allow",
      "Action": [
        "rds:CreateDBCluster",
        "rds:CreateDBInstance"
      ],
      "Resource": [
        "arn:aws:rds:*:*:*"
      ],
      "Condition": {
        "StringEquals": {
          "rds:DatabaseEngine": [
            "graphdb",
            "neptune"
          ]
        }
      }
    },
    {
      "Sid": "AllowManagementPermissionsForRDS",
      "Action": [
        "rds:AddRoleToDBCluster",
        "rds:AddSourceIdentifierToSubscription",
        "rds:AddTagsToResource",
        "rds:ApplyPendingMaintenanceAction",
        "rds:CopyDBClusterParameterGroup",
        "rds:CopyDBClusterSnapshot",
        "rds:CopyDBParameterGroup",
        "rds:CreateDBClusterParameterGroup",
        "rds:CreateDBClusterSnapshot",
```

```
        "rds:CreateDBParameterGroup",
        "rds:CreateDBSubnetGroup",
        "rds:CreateEventSubscription",
        "rds:DeleteDBCluster",
        "rds:DeleteDBClusterParameterGroup",
        "rds:DeleteDBClusterSnapshot",
        "rds:DeleteDBInstance",
        "rds:DeleteDBParameterGroup",
        "rds:DeleteDBSubnetGroup",
        "rds:DeleteEventSubscription",
        "rds:DescribeAccountAttributes",
        "rds:DescribeCertificates",
        "rds:DescribeDBClusterParameterGroups",
        "rds:DescribeDBClusterParameters",
        "rds:DescribeDBClusterSnapshotAttributes",
        "rds:DescribeDBClusterSnapshots",
        "rds:DescribeDBClusters",
        "rds:DescribeDBEngineVersions",
        "rds:DescribeDBInstances",
        "rds:DescribeDBLogFiles",
        "rds:DescribeDBParameterGroups",
        "rds:DescribeDBParameters",
        "rds:DescribeDBSecurityGroups",
        "rds:DescribeDBSubnetGroups",
        "rds:DescribeEngineDefaultClusterParameters",
        "rds:DescribeEngineDefaultParameters",
        "rds:DescribeEventCategories",
        "rds:DescribeEventSubscriptions",
        "rds:DescribeEvents",
        "rds:DescribeOptionGroups",
        "rds:DescribeOrderableDBInstanceOptions",
        "rds:DescribePendingMaintenanceActions",
        "rds:DescribeValidDBInstanceModifications",
        "rds:DownloadDBLogFilePortion",
        "rds:FailoverDBCluster",
        "rds:ListTagsForResource",
        "rds:ModifyDBCluster",
        "rds:ModifyDBClusterParameterGroup",
        "rds:ModifyDBClusterSnapshotAttribute",
        "rds:ModifyDBInstance",
        "rds:ModifyDBParameterGroup",
        "rds:ModifyDBSubnetGroup",
        "rds:ModifyEventSubscription",
        "rds:PromoteReadReplicaDBCluster",
```

```
        "rds:RebootDBInstance",
        "rds:RemoveRoleFromDBCluster",
        "rds:RemoveSourceIdentifierFromSubscription",
        "rds:RemoveTagsFromResource",
        "rds:ResetDBClusterParameterGroup",
        "rds:ResetDBParameterGroup",
        "rds:RestoreDBClusterFromSnapshot",
        "rds:RestoreDBClusterToPointInTime"
      ],
      "Effect": "Allow",
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "AllowOtherDepedentPermissions",
      "Action": [
        "cloudwatch:GetMetricStatistics",
        "cloudwatch:ListMetrics",
        "ec2:AllocateAddress",
        "ec2:AssignIpv6Addresses",
        "ec2:AssignPrivateIpAddresses",
        "ec2:AssociateAddress",
        "ec2:AssociateRouteTable",
        "ec2:AssociateSubnetCidrBlock",
        "ec2:AssociateVpcCidrBlock",
        "ec2:AttachInternetGateway",
        "ec2:AttachNetworkInterface",
        "ec2:CreateCustomerGateway",
        "ec2:CreateDefaultSubnet",
        "ec2:CreateDefaultVpc",
        "ec2:CreateInternetGateway",
        "ec2:CreateNatGateway",
        "ec2:CreateNetworkInterface",
        "ec2:CreateRoute",
        "ec2:CreateRouteTable",
        "ec2:CreateSecurityGroup",
        "ec2:CreateSubnet",
        "ec2:CreateVpc",
        "ec2:CreateVpcEndpoint",
        "ec2:CreateVpcEndpoint",
        "ec2:DescribeAccountAttributes",
        "ec2:DescribeAccountAttributes",
        "ec2:DescribeAddresses",
```

```
        "ec2:DescribeAvailabilityZones",
        "ec2:DescribeAvailabilityZones",
        "ec2:DescribeCustomerGateways",
        "ec2:DescribeInstances",
        "ec2:DescribeNatGateways",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribePrefixLists",
        "ec2:DescribeRouteTables",
        "ec2:DescribeSecurityGroupReferences",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcAttribute",
        "ec2:DescribeVpcAttribute",
        "ec2:DescribeVpcEndpoints",
        "ec2:DescribeVpcs",
        "ec2:DescribeVpcs",
        "ec2:ModifyNetworkInterfaceAttribute",
        "ec2:ModifySubnetAttribute",
        "ec2:ModifyVpcAttribute",
        "ec2:ModifyVpcEndpoint",
        "iam:ListRoles",
        "kms:ListAliases",
        "kms:ListKeyPolicies",
        "kms:ListKeys",
        "kms:ListRetirableGrants",
        "logs:DescribeLogStreams",
        "logs:GetLogEvents",
        "sns:ListSubscriptions",
        "sns:ListTopics",
        "sns:Publish"
      ],
      "Effect": "Allow",
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "AllowPassRoleForNeptune",
      "Action": "iam:PassRole",
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
```

```
        "StringEquals": {
          "iam:passedToService": "rds.amazonaws.com"
        }
      }
    },
    {
      "Sid": "AllowCreateSLRForNeptune",
      "Action": "iam:CreateServiceLinkedRole",
      "Effect": "Allow",
      "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
AWSServiceRoleForRDS",
      "Condition": {
        "StringLike": {
          "iam:AWSServiceName": "rds.amazonaws.com"
        }
      }
    },
    {
      "Sid": "AllowManagementPermissionsForNeptuneAnalytics",
      "Effect": "Allow",
      "Action": [
        "neptune-graph:CreateGraph",
        "neptune-graph:DeleteGraph",
        "neptune-graph:GetGraph",
        "neptune-graph:ListGraphs",
        "neptune-graph:UpdateGraph",
        "neptune-graph:ResetGraph",
        "neptune-graph:CreateGraphSnapshot",
        "neptune-graph:DeleteGraphSnapshot",
        "neptune-graph:GetGraphSnapshot",
        "neptune-graph:ListGraphSnapshots",
        "neptune-graph:RestoreGraphFromSnapshot",
        "neptune-graph:CreatePrivateGraphEndpoint",
        "neptune-graph:GetPrivateGraphEndpoint",
        "neptune-graph:ListPrivateGraphEndpoints",
        "neptune-graph:DeletePrivateGraphEndpoint",
        "neptune-graph:CreateGraphUsingImportTask",
        "neptune-graph:GetImportTask",
        "neptune-graph:ListImportTasks",
        "neptune-graph:CancelImportTask"
      ],
      "Resource": [
        "arn:aws:neptune-graph:*:*:*"
      ]
```

```
    },
    {
      "Sid": "AllowPassRoleForNeptuneAnalytics",
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "iam:passedToService": "neptune-graph.amazonaws.com"
        }
      }
    },
    {
      "Sid": "AllowCreateSLRForNeptuneAnalytics",
      "Effect": "Allow",
      "Action": "iam:CreateServiceLinkedRole",
      "Resource": "arn:aws:iam::*:role/aws-service-role/neptune-graph.amazonaws.com/
AWSServiceRoleForNeptuneGraph",
      "Condition": {
        "StringLike": {
          "iam:AWSServiceName": "neptune-graph.amazonaws.com"
        }
      }
    }
  ]
}
```

NeptuneGraphReadOnlyAccessAWS 管理ポリシー

以下のNeptuneGraphReadOnlyアクセス管理ポリシーは、すべての Amazon Neptune Analytics リソースへの読み取り専用アクセスと、依存サービスの読み取り専用アクセス許可を提供します。

このポリシーには以下を実行するための許可が含まれています。

- Amazon EC2 の場合 — VPC、サブネット、セキュリティグループ、アベイラビリティーゾーンに関する情報を取得します。
- の場合 AWS KMS — KMS キーとエイリアスに関する情報を取得します。
- の場合 CloudWatch — CloudWatch メトリクスに関する情報を取得します。
- CloudWatch ログの場合 — CloudWatch ログストリームとイベントに関する情報を取得します。

> **Note**
>
> このポリシーは 2023 年 11 月 29 日にリリースされました。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowReadOnlyPermissionsForNeptuneGraph",
      "Effect": "Allow",
      "Action": [
        "neptune-graph:Get*",
        "neptune-graph:List*",
        "neptune-graph:Read*"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AllowReadOnlyPermissionsForEC2",
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeVpcEndpoints",
        "ec2:DescribeVpcAttribute",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcs",
        "ec2:DescribeAvailabilityZones"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AllowReadOnlyPermissionsForKMS",
      "Effect": "Allow",
      "Action": [
        "kms:ListKeys",
        "kms:ListAliases"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AllowReadOnlyPermissionsForCloudwatch",
```

```
      "Effect": "Allow",
      "Action": [
        "cloudwatch:GetMetricData",
        "cloudwatch:ListMetrics",
        "cloudwatch:GetMetricStatistics"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AllowReadOnlyPermissionsForLogs",
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogStreams",
        "logs:GetLogEvents"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:/aws/neptune/*:log-stream:*"
      ]
    }
  ]
}
```

AWSServiceRoleForNeptuneGraphPolicyAWS 管理ポリシー

以下の AWSServiceRoleForNeptuneGraphPolicy管理ポリシーは、運用および使用状況のメトリクスとログを公開 CloudWatch するための へのアクセスをグラフに付与します。「nan-service-linked-roles」を参照してください。

> **Note**
>
> このポリシーは 2023 年 11 月 29 日にリリースされました。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "GraphMetrics",
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
```

```
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "cloudwatch:namespace": [
            "AWS/Neptune",
            "AWS/Usage"
          ]
        }
      }
    },
    {
      "Sid": "GraphLogGroup",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:/aws/neptune/*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "${aws:PrincipalAccount}"
        }
      }
    },
    {
      "Sid": "GraphLogEvents",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:/aws/neptune/*:log-stream:*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "${aws:PrincipalAccount}"
        }
      }
    }
  ]
```

```
}
```

Amazon Neptune によってサポートされる IAM 条件コンテキストキー

Neptune 管理アクションおよびリソースへのアクセスを制御する条件を IAM ポリシーで指定できます。ポリシーステートメントは、条件が true の場合にのみ有効です。

例えば、特定の日付の後にのみ適用されるポリシーステートメントや、API リクエストに特定の値が存在する場合のみアクセスが許可されるポリシーステートメントが必要になる場合があります。

条件を表すには、「等しい」や「より小さい」などの IAM 条件ポリシー演算子 とともに、ポリシーステートメントの `Condition` 要素であらかじめ定義された条件キーを使用します。

1 つのステートメントに複数の `Condition` 要素を指定するか、1 つの `Condition` 要素に複数のキーを指定すると、 AWS は AND 論理演算子を使用してそれらを評価します。1 つの条件キーに複数の値を指定すると、 は論理ORオペレーションを使用して条件 AWS を評価します。ステートメントの権限が付与される前にすべての条件が満たされる必要があります。

条件を指定する際にプレースホルダー変数も使用できます。例えば IAM ユーザーに、IAM ユーザー名がタグ付けされている場合のみリソースにアクセスできる権限を付与することができます。詳細については、「IAM ユーザーガイド」の「IAM Policy Elements: Variables and Tags」(IAM ポリシーの要素: 変数およびタグ) を参照してください。

条件キーのデータ型によって、リクエスト内の値とポリシーステートメント内の値の比較に使用できる条件演算子が決まります。そのデータ型と互換性のない条件演算子を使用した場合、条件は一致しないため、ポリシーステートメントは適用されません。

Neptune 管理ポリシーステートメントの IAM 条件キー

- グローバル条件キー – Neptune 管理ポリシーステートメントでは、ほとんどの AWS グローバル条件キーを使用できます。
- サービス固有の条件キー – これらは、特定の AWS サービス用に定義されたキーです。Neptune が管理ポリシーステートメントでサポートしているものは、Neptune IAM 管理ポリシーステートメントで使用できる条件キー に記載されています。

データアクセスポリシーステートメントの IAM 条件キー

- グローバル条件キー — Neptune がデータアクセスポリシーステートメントでサポートするこれらのキーのサブセットは、AWS データアクセスポリシーステートメントで Neptune がサポートするグローバル条件コンテキストキー に記載されています。
- Neptune がデータアクセスポリシーステートメント用に定義するサービス固有の条件キーは、条件キー に記載されています。

Amazon Neptune のカスタム IAM 管理ポリシーステートメント

管理ポリシーステートメントを使用すると、IAM ユーザーが Neptune データベースを管理するために実行できる操作を制御できます。

Neptune 管理ポリシーステートメントは、Neptune がサポートする 1 つ以上の管理アクションと管理リソースへのアクセスを付与します。条件キー を使用して、管理アクセス許可をより具体的にすることもできます。

> **Note**
>
> Neptune は Amazon RDS と機能を共有しているため、管理ポリシーステートメントの管理アクション、リソース、およびサービス固有の条件キーは、設計により `rds:` プレフィックスを使用します。

トピック

- Neptune IAM 管理ポリシーステートメントで利用可能なアクション
- IAM Neptune 管理ポリシーステートメントで使用できるリソースタイプ
- Neptune IAM 管理ポリシーステートメントで使用できる条件キー
- Neptune の IAM 管理ポリシーステートメントの例

Neptune IAM 管理ポリシーステートメントで利用可能なアクション

IAM ポリシーステートメントの `Action` 要素で以下に示す管理アクションを使用して、Neptune 管理 API へのアクセスを制御できます。ポリシーでアクションを使用する場合は、通常、同じ名前の API オペレーションまたは CLI コマンドへのアクセスを許可または拒否します。ただし、場合によっては、1 つのアクションによって複数のオペレーションへのアクセスが制御されます。あるいは、いくつかのオペレーションはいくつかの異なるアクションを必要とします。

以下のリストの Resource type フィールドは、各アクションがリソースレベルのアクセス許可をサポートしているかどうかを示します。このフィールドに値がない場合は、ポリシーステートメントの Resource 要素ですべてのリソース (「*」) を指定する必要があります。列にリソースタイプが含まれる場合、そのアクションを含むステートメントでそのタイプの ARN を指定できます。Neptune 管理リソースタイプは、このページに記載されています。

必須リソースは、下記のリストでアスタリスク (*) で示されています。このアクションを使用してステートメントでリソースレベルのアクセス許可 ARN を指定する場合、このタイプである必要があります。一部のアクションでは、複数のリソースタイプがサポートされています。リソースタイプがオプション (つまり、アスタリスクが付いていない) であれば、含める必要はありません。

ここにリストされているフィールドの詳細については、「IAM ユーザーガイド」の「アクションテーブル」を参照してください。

rds:AddRoleToDBCluster

AddRoleToDBCluster IAM ロールを Neptune DB クラスターに関連付けるには

アクセスレベル: Write。

依存アクション iam:PassRole。

リソースタイプ: クラスター (必須)。

rds:AddSourceIdentifierToサブスクリプション

AddSourceIdentifierToSubscription 既存の Neptune イベント通知サブスクリプションにソース識別子を追加します。

アクセスレベル: Write。

リソースタイプ: es (必須)。

rds:AddTagsToResource

AddTagsToResource IAM ロールを Neptune DB クラスターに関連付けるには

アクセスレベル: Write。

リソースタイプ:

- db

- es
- pg
- cluster-snapshot
- subgrp

条件キー:

- aws:RequestTag/*tag-key*
- aws:TagKeys

rds:ApplyPendingMaintenanceAction

ApplyPendingMaintenanceAction 保留中のメンテナンスアクションをリソースに適用します。

アクセスレベル: Write。

リソースタイプ: db (必須)。

rds:CopyDBClusterParameter グループ

CopyDBClusterParameterGroup 指定された DB クラスターパラメータグループをコピーします。

アクセスレベル: Write。

リソースタイプ: cluster-pg (必須)。

rds:CopyDBClusterSnapshot

CopyDBClusterSnapshot DB クラスターのスナップショットをコピーします。

アクセスレベル: Write。

リソースタイプ: クラススナップショット (必須)。

rds:CopyDBParameterGroup

CopyDBParameterGroup 指定された DB パラメータグループをコピーします。

アクセスレベル: Write。

リソースタイプ: pg (必須)。

rds:CreateDBCluster

CreateDBCluster 新しい Amazon Neptune DB クラスターを作成します。

アクセスレベル: Tagging。

依存アクション iam:PassRole。

リソースタイプ:

- クラスター (必須)。
- cluster-pg (必須)。
- subgrp (必須)。

条件キー:

- aws:RequestTag/*tag-key*
- aws:TagKeys
- neptune-rds_DatabaseEngine

rds:CreateDBClusterParameter グループ

CreateDBClusterParameterGroup新しい DB クラスターのパラメータグループを作成します。

アクセスレベル: Tagging。

リソースタイプ: cluster-pg (必須)。

条件キー:

- aws:RequestTag/*tag-key*
- aws:TagKeys

rds:CreateDBClusterSnapshot

CreateDBClusterSnapshotDB クラスターのスナップショットを作成します。

アクセスレベル: Tagging。

リソースタイプ:

- クラスター (必須)。
- クラスタースナップショット (必須)。

条件キー:

- aws:RequestTag/*tag-key*
- aws:TagKeys

rds:CreateDBInstance

CreateDBInstance新しい DB インスタンスを作成します。

アクセスレベル: Tagging。

依存アクション iam:PassRole。

リソースタイプ:

- db (必須)。
- pg (必須)。
- subgrp (必須)。

条件キー:

- aws:RequestTag/*tag-key*
- aws:TagKeys

rds:CreateDBParameterGroup

CreateDBParameterGroup は 新しい DB パラメータグループを作成します。

アクセスレベル: Tagging。

リソースタイプ: pg (必須)。

条件キー:

- aws:RequestTag/*tag-key*
- aws:TagKeys

rds:CreateDBSubnetGroup

CreateDBSubnetGroup新しい DB サブネットグループを作成します。

アクセスレベル: Tagging。

リソースタイプ: subgrp (必須)。

条件キー:

- aws:RequestTag/*tag-key*
- aws:TagKeys

rds:CreateEventサブスクリプション

CreateEventSubscription RDS イベント通知サブスクリプションを作成します。

アクセスレベル: Tagging。

リソースタイプ: es (必須)。

条件キー:

- aws:RequestTag/*tag-key*
- aws:TagKeys

rds:DeleteDBCluster

DeleteDBCluster 既存の Neptune DB クラスターを削除します。

アクセスレベル: Write。

リソースタイプ:

- クラスター (必須)。
- クラススナップショット (必須)。

rds:DeleteDBClusterParameter グループ

DeleteDBClusterParameterGroup 指定された DB クラスターパラメータグループを削除します。

アクセスレベル: Write。

リソースタイプ: cluster-pg (必須)。

rds:DeleteDBClusterSnapshot

DeleteDBClusterSnapshot DB クラススナップショットを削除します。

アクセスレベル: Write。

リソースタイプ: クラススナップショット (必須)。

rds:DeleteDBInstance

DeleteDBInstance 指定された DB インスタンスを削除します。

アクセスレベル: Write。

リソースタイプ: db (必須)。

rds:DeleteDBParameterGroup

DeleteDBParameterGroup は、指定された DB を削除しますParameterGroup。

アクセスレベル: Write。

リソースタイプ: pg (必須)。

rds:DeleteDBSubnetGroup

DeleteDBSubnetGroup DB サブネットグループを削除します。

アクセスレベル: Write。

リソースタイプ: subgrp (必須)。

rds:DeleteEventサブスクリプション

DeleteEventSubscription イベント通知サブスクリプションを削除します。

アクセスレベル: `Write`。

リソースタイプ: es (必須)。

rds:DescribeDBClusterParameter グループ

DescribeDBClusterParameterGroups は DB ClusterParameterGroup の説明のリストを返します。

アクセスレベル: `List`。

リソースタイプ: cluster-pg (必須)。

rds:DescribeDBClusterParameters

DescribeDBClusterParameters 特定の DB クラスターパラメータグループの詳細なパラメータリストを返します。

アクセスレベル: `List`。

リソースタイプ: cluster-pg (必須)。

rds:DescribeDB 属性ClusterSnapshot

DescribeDBClusterSnapshotAttributes 手動の DB クラスタースナップショットの DB クラスタースナップショットの属性名と値のリストを返します。

アクセスレベル: `List`。

リソースタイプ: クラススナップショット (必須)。

rds:DescribeDBClusterSnapshots

DescribeDBClusterSnapshots DB クラスタースナップショットに関する情報を返します。

アクセスレベル: `Read`。

rds:DescribeDBClusters

DescribeDBClusters プロビジョニングされた Neptune DB クラスターに関する情報を返します。

アクセスレベル: `List`。

リソースタイプ: クラスター (必須)。

rds:DescribeDBEngineVersions

DescribeDBEngineVersions 利用可能な DB エンジンのリストを返します。

アクセスレベル: List。

リソースタイプ: pg (必須)。

rds:DescribeDBInstances

DescribeDBInstances すべての DB インスタンスに関する情報を返します。

アクセスレベル: List。

リソースタイプ: es (必須)。

rds:DescribeDBParameterGroups

DescribeDBParameterGroups は DB ParameterGroup の説明のリストを返します。

アクセスレベル: List。

リソースタイプ: pg (必須)。

rds:DescribeDBParameters

DescribeDBParameters 特定の DB パラメータグループの詳細なパラメータリストを返します。

アクセスレベル: List。

リソースタイプ: pg (必須)。

rds:DescribeDBSubnetGroups

DescribeDBSubnetGroups は DB SubnetGroup の説明のリストを返します。

アクセスレベル: List。

リソースタイプ: subgrp (必須)。

rds: DescribeEventカテゴリ

DescribeEventCategories すべてのイベントソースタイプか、指定されている場合は、指定されたソースタイプのイベントカテゴリのリストを返します。

アクセスレベル: List。

rds:DescribeEventサブスクリプション

DescribeEventSubscriptions顧客アカウントのサブスクリプションの説明をすべて表示します。

アクセスレベル: List。

リソースタイプ: es (必須)。

rds:DescribeEvents

DescribeEvents DB インスタンス、DB セキュリティグループ、DB スナップショット、DB パラメータグループに関連する過去 14 日間のイベントを返します。

アクセスレベル: List。

リソースタイプ: es (必須)。

rds:DescribeOrderableDBInstanceOptions

DescribeOrderableDBInstanceOptions 指定されたエンジンの注文可能な DB インスタンスオプションのリストを返します。

アクセスレベル: List。

rds:DescribePendingMaintenanceActions

DescribePendingMaintenanceActions 少なくとも 1 つの保留中のメンテナンスアクションを含むリソース (例: DB インスタンス) のリストを返します。

アクセスレベル: List。

リソースタイプ: db (必須)。

rds:DescribeValidDBInstanceModifications

DescribeValidDBInstanceModifications DB インスタンスに加えることができる変更を一覧表示します。

アクセスレベル: List。

リソースタイプ: db (必須)。

rds:FailoverDBCluster

FailoverDBCluster DB クラスターのフェイルオーバーを強制実行します。

アクセスレベル: Write。

リソースタイプ: クラスター (必須)。

rds:ListTagsForResource

ListTagsForResource Amazon Neptune リソースのすべてのタグを一覧表示します。

アクセスレベル: Read。

リソースタイプ:

- cluster-snapshot
- db
- es
- pg
- subgrp

rds:ModifyDBCluster

ModifyDBCluster

Neptune DB クラスターの設定を変更します。

アクセスレベル: Write。

依存アクション iam:PassRole。

リソースタイプ:

- クラスター (必須)。
- cluster-pg (必須)。

rds:ModifyDBClusterParameter グループ

ModifyDBClusterParameterGroup DB クラスターパラメータグループのパラメータを変更します。

アクセスレベル: `Write`。

リソースタイプ: cluster-pg (必須)。

rds:ModifyDBClusterSnapshot 属性

`ModifyDBClusterSnapshotAttribute` 属性および値を、手動 DB クラスタースナップショットに追加するか、ここから属性および値を削除します。

アクセスレベル: `Write`。

リソースタイプ: クラスタースナップショット (必須)。

rds:ModifyDBInstance

`ModifyDBInstance` DB インスタンスの設定を変更します。

アクセスレベル: `Write`。

依存アクション `iam:PassRole`。

リソースタイプ:

- db (必須)。
- pg (必須)。

rds:ModifyDBParameterGroup

`ModifyDBParameterGroup` DB パラメータグループのパラメータを変更します。

アクセスレベル: `Write`。

リソースタイプ: pg (必須)。

rds:ModifyDBSubnetGroup

`ModifyDBSubnetGroup` 既存の DB サブネットグループを変更します。

アクセスレベル: `Write`。

リソースタイプ: subgrp (必須)。

rds:ModifyEventサブスクリプション

ModifyEventSubscription 既存の RDS イベント通知サブスクリプションを変更します。

アクセスレベル: Write。

リソースタイプ: es (必須)。

RDS:RebootDBInstance

RebootDBInstance インスタンスを再起動すると、データベースエンジンサービスが再起動されます。

アクセスレベル: Write。

リソースタイプ: db (必須)。

rds:RemoveRoleFromDBCluster

RemoveRoleFromDBCluster は、Amazon Neptune DB クラスターから AWS Identity and Access Management (IAM) ロールの関連付けを解除します。 Amazon Neptune

アクセスレベル: Write。

依存アクション iam:PassRole。

リソースタイプ: クラスター (必須)。

rds:RemoveSourceIdentifierFromサブスクリプション

RemoveSourceIdentifierFromSubscription 既存のイベント通知サブスクリプションからソース識別子を削除します。

アクセスレベル: Write。

リソースタイプ: es (必須)。

rds:RemoveTagsFromResource

RemoveTagsFromResource Neptune リソースからメタデータタグを削除します。

アクセスレベル: Tagging。

リソースタイプ:

- cluster-snapshot
- db
- es
- pg
- subgrp

条件キー:

- aws:RequestTag/*tag-key*
- aws:TagKeys

rds:ResetDBClusterParameter グループ

`ResetDBClusterParameterGroup` DB クラスターパラメータグループのパラメータをデフォルト値に変更します。

アクセスレベル: `Write`。

リソースタイプ: cluster-pg (必須)。

rds:ResetDBParameterGroup

`ResetDBParameterGroup` はDB パラメータグループのパラメータをエンジン/システムのデフォルト値に変更します。

アクセスレベル: `Write`。

リソースタイプ: pg (必須)。

rds:RestoreDBClusterFrom

`RestoreDBClusterFromSnapshot` DB クラススナップショットから新しい DB クラスターを作成します。

アクセスレベル: `Write`。

依存アクション `iam:PassRole`。

リソースタイプ:

- クラスター (必須)。
- クラスタースナップショット (必須)。

条件キー:

- aws:RequestTag/*tag-key*
- aws:TagKeys

rds:RestoreDBClusterToPointIn

RestoreDBClusterToPointInTime DB クラスターを任意の時点に復元します。

アクセスレベル: Write。

依存アクション iam:PassRole。

リソースタイプ:

- クラスター (必須)。
- subgrp (必須)。

条件キー:

- aws:RequestTag/*tag-key*
- aws:TagKeys

rds:StartDBCluster

StartDBCluster指定された DB クラスターを起動します。

アクセスレベル: Write。

リソースタイプ: クラスター (必須)。

rds:StopDBCluster

StopDBCluster指定された DB クラスターを停止します。

アクセスレベル: Write。

リソースタイプ: クラスター (必須)。

IAM Neptune 管理ポリシーステートメントで使用できるリソースタイプ

Neptune は、次の表のリソースタイプをサポートしており、IAM 管理ポリシーステートメントの Resource 要素で使うことができます。Resource 要素の詳細については、「IAM JSON ポリシーの要素: リソース」を参照してください。

Neptune 管理アクションのリストは、各アクションで指定できるリソースタイプを示しています。リソースタイプは、次の表最後の列で指定されているように、ポリシーに含めることができる条件キーを決定します。

下の表の ARN 列は、このタイプのリソースの参照に使用する必要がある Amazon リソースネーム (ARN) の形式を指定しています。 $ で始まる部分は、お客様の状況で実際の値に置き換える必要があります。例えば、ARN に $user-name と表示されている場合は、その文字列を実際の IAM ユーザー名か、IAM ユーザー名を含むポリシー変数に置き換える必要があります。ARN の詳細については、「IAM ARN」と「Amazon Neptune で ARN を使用する」を参照してください。

Condition Keys 列では、このリソースとサポートするアクションの両方がステートメントに含まれている場合にのみ IAM ポリシーステートメントに含むことができる条件コンテキストキーを指定します。

リソースタイプ	ARN	条件キー
cluster (DB クラスター)	arn:*partition* :rds:*region*:*account-id* :cluster: *instance-name*	aws:ResourceTag/*tag-key* rds:cluster-tag/*tag-key*
cluster-pg (DB クラスターのパラメータグループ)	arn:*partition* :rds:*region*:*account-id* :cluster-pg: *neptune-DBClusterParameterGroupName*	aws:ResourceTag/*tag-key*

リソースタイプ	ARN	条件キー
cluster-snapshot (DB クラスタースナップショット)	arn:*partition* :rds:*region*:*account-id* :cluster-snapshot: *neptune-DBClusterSnapshotName*	aws:ResourceTag/*tag-key* rds:cluster-snapshot-tag/*tag-key*
db (DB インスタンス)	arn:*partition* :rds:*region*:*account-id* :db:*neptune-DbInstanceName*	aws:ResourceTag/*tag-key* rds:DatabaseClass rds:DatabaseEngine rds: db-tag*tag-key*
es (イベントサブスクリプション)	arn:*partition* :rds:*region*:*account-id* :es:*neptune-CustSubscriptionId*	aws:ResourceTag/*tag-key* rds: es-tag*tag-key*
pg (DB パラメータグループを編集する)	arn:*partition* :rds:*region*:*account-id* :pg:*neptune-ParameterGroupName*	aws:ResourceTag/*tag-key* rds: pg-tag*tag-key*
subgrp (DB サブネットグループ)	arn:*partition* :rds:*region*:*account-id* :subgrp:*neptune-DBSubnetGroupName* }	aws:ResourceTag/*tag-key* rds: subgrp-tag/*tag-key*

Neptune IAM 管理ポリシーステートメントで使用できる条件キー

条件キーを使用すると、IAM ポリシーステートメントで条件を指定して、条件が満たされた場合にのみステートメントが有効になるようにすることができます。Neptune 管理ポリシーステートメントで使用できる条件キーは、次のカテゴリに分類されます。

- グローバル条件キー – これらは、 AWS のサービスで一般的に使用するために AWS で定義されます。ほとんどは Neptune 管理ポリシーステートメントで使用できます。
- 管理リソースプロパティ条件キー — 以下に示すこれらのキーは、管理リソースのプロパティに基づいています。
- タグベースのアクセス条件キー — 以下に示すこれらのキーは、管理リソースに付けられた AWS タグに基づいています。

Neptune 管理リソースプロパティの条件キー

条件キー	説明	[Type] (タイプ)
`rds:DatabaseClass`	DB インスタンスクラスのタイプによってアクセスをフィルタリングします。	文字列
`rds:DatabaseEngine`	データベースエンジンでアクセスをフィルタリングします。可能な値については、CreateDBInstance API のエンジンパラメータを参照してください。	文字列
`rds:DatabaseName`	DB インスタンス上のデータベースのユーザー定義名でアクセスをフィルタリングします。	文字列
`rds:EndpointType`	エンドポイントのタイプでアクセスをフィルタリングします。READER、WRITER、CUSTOM のいずれか。	文字列
`rds:Vpc`	DB インスタンスを Amazon Virtual Private Cloud (Amazon VPC) で実行するかどうかを指定する値でアクセスをフィルタリングします。DB インスタンスが Amazon VPC で実行されていることを示すには、`true` を指定します。	ブール値

管理タグベースの条件キー

Amazon Neptune では、カスタムタグを使用して IAM ポリシーで条件を指定することがサポートされており、管理 API リファレンス を介して Neptune へのアクセスを制御します。

たとえば、DB インスタンスに `environment` という名前のタグを追加するとします。`beta`、`staging`、および `production` のような値を使用します。そのタグの値に基づいてインスタンスへのアクセスを制限するポリシーを作成できます。

> **Important**
>
> タグを使用して Neptune リソースへのアクセスを管理する場合は、タグへのアクセスを保護してください。`AddTagsToResource` および `RemoveTagsFromResource` アクションのポリシーを作成することによって、タグへのアクセスを制限できます。
> 例えば、次のポリシーは、ユーザーがすべてのリソースのタグを追加または削除することを拒否します。次に、特定のユーザーがタグを追加または削除することを許可するポリシーを作成できます。
>
> ```
> { "Version": "2012-10-17",
>   "Statement":[
>     { "Sid": "DenyTagUpdates",
>       "Effect": "Deny",
>       "Action": [
>         "rds:AddTagsToResource",
>         "rds:RemoveTagsFromResource"
>       ],
>       "Resource":"*"
>     }
>   ]
> }
> ```

以下のタグベースの条件キーは、管理ポリシーステートメント内の管理リソースでのみ機能します。

タグベースの管理条件キー

条件キー	説明	[Type] (タイプ)
aws:RequestTag/${TagKey}	リクエスト内のタグキーと値のペアの有無に基づいてアクセスをフィルタリングします。	文字列
aws:ResourceTag/${TagKey}	リソースにアタッチされているタグキーと値のペアに基づいてアクセスをフィルタリングします。	文字列
aws:TagKeys	リクエスト内のタグキーの有無に基づいてアクセスをフィルタリングします	文字列
rds:cluster-pg-tag/${TagKey}	DB クラスターパラメータグループにアタッチされたタグによってアクセスをフィルタリングします。	文字列
rds:cluster-snapshot-tag/${TagKey}	DB クラススナップショットにアタッチされたタグによってアクセスをフィルタリングします。	文字列
rds:cluster-tag/${TagKey}	DB クラスターにアタッチされたタグによってアクセスをフィルタリングします。	文字列
rds:db-tag/${TagKey}	DB インスタンスにアタッチされたタグによってアクセスをフィルタリングします。	文字列
rds:es-tag/${TagKey}	イベントサブスクリプションにアタッチされたタグによってアクセスをフィルタリングします。	文字列
rds:pg-tag/${TagKey}	DB パラメータグループにアタッチされたタグによってアクセスをフィルタリングします。	文字列

条件キー	説明	[Type] (タイプ)
rds:req-tag/${TagKey}	リソースにタグを付けるために使用できるタグキーと値のセットによってアクセスをフィルタリングします。	文字列
rds:secgrp-tag/${TagKey}	DB セキュリティグループにアタッチされたタグによってアクセスをフィルタリングします。	文字列
rds:snapshot-tag/${TagKey}	DB スナップショットにアタッチされたタグによってアクセスをフィルタリングします	文字列
rds:subgrp-tag/${TagKey}	DB サブネットグループにアタッチされたタグでアクセスをフィルタリングします。	文字列

Neptune の IAM 管理ポリシーステートメントの例

一般的な管理ポリシーの例

以下の例は、DB クラスターに対してさまざまな管理アクションを実行するためのアクセス許可を付与する Neptune 管理ポリシーを作成する方法を示しています。

IAM ユーザーが指定した DB インスタンスを削除できないようにするポリシー

以下は、IAM ユーザーが指定した DB インスタンスを削除できないようにするポリシーの例です。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyDeleteOneInstance",
      "Effect": "Deny",
      "Action": "rds:DeleteDBInstance",
      "Resource": "arn:aws:rds:us-west-2:123456789012:db:my-instance-name"
    }
  ]
}
```

新しい DB インスタンスを作成するアクセス許可を付与するポリシー

以下は、IAM ユーザーが指定された Neptune DB クラスターに DB インスタンスを作成できるようにするポリシーの例です。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCreateInstance",
      "Effect": "Allow",
      "Action": "rds:CreateDBInstance",
      "Resource": "arn:aws:rds:us-west-2:123456789012:cluster:my-cluster"
    }
  ]
}
```

特定の DB パラメータグループを使用する新しい DB インスタンスを作成するアクセス許可を付与するポリシー

以下は、IAM ユーザーが、指定された DB パラメータグループのみを使用して、指定された Neptune DB クラスター内の指定された DB クラスター (ここでは us-west-2) に DB インスタンスを作成することを許可するポリシーの例です。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCreateInstanceWithPG",
      "Effect": "Allow",
      "Action": "rds:CreateDBInstance",
      "Resource": [
        "arn:aws:rds:us-west-2:123456789012:cluster:my-cluster",
        "arn:aws:rds:us-west-2:123456789012:pg:my-instance-pg"
      ]
    }
  ]
}
```

リソースを記述するアクセス許可を付与するポリシー

以下は、IAM ユーザーが任意の Neptune リソースを記述できるようにするポリシーの例です。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowDescribe",
      "Effect": "Allow",
      "Action": "rds:Describe*",
      "Resource": *
    }
  ]
}
```

タグベースの管理ポリシーの例

以下の例は、DB クラスターに対してさまざまな管理アクションを実行するためのアクセス許可をフィルタリングするタグを付ける Neptune 管理ポリシーを作成する方法を示しています。

例 1: 複数の値を取ることができるカスタムタグを使用して、リソースに対するアクションにアクセス許可を付与する

以下のポリシーでは、ModifyDBInstance、CreateDBInstanceまたはDeleteDBInstance API を dev または test どちらかに設定したenv タグを持つある DB インスタンスで使用できます。

```
{ "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowDevTestAccess",
      "Effect": "Allow",
      "Action": [
        "rds:ModifyDBInstance",
        "rds:CreateDBInstance",
        "rds:DeleteDBInstance"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "rds:db-tag/env": [
              "dev",
              "test"
          ],
          "rds:DatabaseEngine": "neptune"
        }
```

```
      }
    }
  ]
}
```

例 2: リソースにタグを付けるために使用できるタグキーと値のセットを制限する

このポリシーでは、Condition キーを使って、キー env を持つタグと、リソースに追加する test、qa、または dev の値の使用を許可します。

```
{ "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowTagAccessForDevResources",
      "Effect": "Allow",
      "Action": [
        "rds:AddTagsToResource",
        "rds:RemoveTagsFromResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "rds:req-tag/env": [
            "test",
            "qa",
            "dev"
          ],
          "rds:DatabaseEngine": "neptune"
        }
      }
    }
  ]
}
```

例 3: **aws:ResourceTag** に基づいて Neptune リソースへのフルアクセスを許可する

以下のポリシーは、上記の最初の例と似ていますが、代わりに aws:ResourceTag を使います。

```
{ "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowFullAccessToDev",
      "Effect": "Allow",
```

```
      "Action": [
        "rds:*"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/env": "dev",
          "rds:DatabaseEngine": "neptune"
        }
      }
    }
  ]
}
```

Amazon Neptune のカスタム IAM データアクセスポリシーステートメント

Neptune データアクセスポリシーステートメントは、データアクセスアクション、リソース、および条件キーを使用し、これらはすべて neptune-db: プレフィックスが付きます。

トピック

- Neptune データアクセスポリシーステートメントでのクエリアクションの使用
- Neptune IAM データアクセスポリシーステートメントで使用可能なアクション
- Neptune IAM データアクセスポリシーステートメントでのリソースの指定
- Neptune IAM データアクセスポリシーステートメントで使用可能な条件キー
- Neptune IAM データアクセスポリシーの例

Neptune データアクセスポリシーステートメントでのクエリアクションの使用

データアクセスポリシーステートメントで使用できる Neptune クエリアクションには、`ReadDataViaQuery`、`WriteDataViaQuery`、`DeleteDataViaQuery` の 3 つがあります。特定のクエリには、これらのアクションを複数実行するためのアクセス許可が必要な場合があり、クエリを実行するためにこれらのアクションのどの組み合わせを許可する必要があるかが常に明確であるとは限りません。

クエリを実行する前に、Neptune はクエリの各ステップを実行するのに必要なアクセス許可を決定し、それらを組み合わせてクエリに必要なアクセス許可のフルセットにします。このフルセットのアクセス許可には、クエリが実行する可能性のあるすべてのアクションが含まれますが、必ずしもクエリがデータに対して実際に実行するアクションのセットではないことに注意してください。

つまり、特定のクエリの実行を許可するには、そのクエリが実際に実行するかどうかにかかわらず、そのクエリが実行する可能性のあるすべてのアクションにアクセス許可を与える必要があります。

以下に、Gremlin クエリの例をいくつか示します。これについてさらに詳しく説明しています。

- ```
  g.V().count()
  ```

  g.V() および count() は読み取りアクセスだけを必要とするため、クエリ全体として必要なのは ReadDataViaQuery アクセスのみです。

- ```
  g.addV()
  ```

  addV() は、新しい ID の頂点を挿入する前に、特定の ID の頂点が存在するかどうかを確認する必要があります。つまり、ReadDataViaQuery と WriteDataViaQuery の両方のアクセスが必要です。

- ```
  g.V('1').as('a').out('created').addE('createdBy').to('a')
  ```

  g.V('1').as('a') と out('created') は読み取りアクセスのみを必要としますが、addE().from('a') は読み取りと書き込みの両方のアクセスを必要とします。addE() は from を読み取る必要があり、to は新しい ID のエッジを追加する前に、同じ ID のエッジが既に存在するかどうかを確認する必要があるためです。そのため、クエリ全体としては、ReadDataViaQuery と WriteDataViaQuery の両方のアクセスが必要です。

- ```
  g.V().drop()
  ```

  g.V() は読み取りアクセスのみを必要とします。drop() は頂点または エッジを削除する前にそれらを読み取る必要があるため、読み取りアクセスと削除アクセスの両方が必要です。そのため、クエリ全体としては ReadDataViaQuery と DeleteDataViaQuery の両方のアクセスが必要です。

- ```
  g.V('1').property(single, 'key1', 'value1')
  ```

  g.V('1') は読み取りアクセスのみを必要としますが、property(single, 'key1', 'value1') は読み取り、書き込み、および削除アクセスを必要とします。ここでは、property() ステップは、頂点にまだ存在しない場合はキーと値を挿入しますが、既に存在する場合は、既存のプロパティ値を削除し、代わりに新しい値を挿入します。したがって、クエ
  ```

リ全体としては、ReadDataViaQuery、WriteDataViaQuery、および DeleteDataViaQuery アクセスが必要です。

property() ステップを含むクエリには、ReadDataViaQuery、WriteDataViaQuery、および DeleteDataViaQuery アクセス許可が必要です。

openCypher の例をいくつか示します。

- ```
  MATCH (n)
  RETURN n
  ```

  このクエリはデータベース内のすべてのノードを読み込んで返します。必要なのは ReadDataViaQuery アクセスだけです。

- ```
  MATCH (n:Person)
  SET n.dept = 'AWS'
  ```

  このクエリには、ReadDataViaQuery、WriteDataViaQuery、および DeleteDataViaQuery アクセスが必要です。「Person」というラベルの付いたすべてのノードを読み取り、キー dept と値 AWS を持つ新しいプロパティを追加するか、dept プロパティが既に存在する場合は、古い値を削除して、代わりに AWS を挿入します。また、設定する値が null の場合、SET はプロパティをすべて削除します。

  SET 句は既存の値を削除しなければならない場合があるため、常に DeleteDataViaQuery アクセス許可だけでなく ReadDataViaQuery および WriteDataViaQuery アクセス許可も必要です。

- ```
  MATCH (n:Person)
  DETACH DELETE n
  ```

  このクエリには ReadDataViaQuery および DeleteDataViaQuery アクセス許可が必要です。ラベル Person の付いたノードをすべ r て検索し、それらのノードに接続されているエッジや関連するラベルやプロパティとともに削除します。

- ```
  MERGE (n:Person {name: 'John'})-[:knows]->(:Person {name: 'Peter'})
  RETURN n
  ```

このクエリには ReadDataViaQuery および WriteDataViaQuery アクセス許可が必要です。MERGE 句は、指定されたパターンと一致するか、それを作成します。パターンが一致しないと書き込みが発生する可能性があるため、読み取りアクセス許可だけでなく書き込みアクセス許可も必要です。

Neptune IAM データアクセスポリシーステートメントで使用可能なアクション

Neptune データアクセスアクションにはプレフィックス neptune-db: が付いていますが、Neptune の管理アクションにはプレフィックス rds: が付いていることに注意してください。

IAM でのデータの Amazon リソースネーム (ARN) は、作成時にクラスターに割り当てられた ARN とは異なります。「データリソースの指定」に示されているように ARN を構築する必要があります。このようなデータリソース ARN では、ワイルドカードを使用して複数のリソースを含めることができます。

データアクセスポリシーステートメントには、neptune-db:QueryLanguage 条件キーを含めて、クエリ言語によるアクセスを制限することもできます。

リリース: 1.2.0.0 (2022-07-21) 以降、Neptune は 1 つ以上の特定の Neptune アクションへのアクセス許可の制限をサポートしています。これにより、以前よりもきめ細かなアクセスコントロールが可能になります。

> **Important**
>
> - IAM ポリシーへの変更は、指定された Neptune リソースへの適用に最大で 10 分かかります。
> - Neptune DB クラスターに適用された IAM ポリシーは、そのクラスター内のすべてのインスタンスに適用されます。

クエリベースのデータアクセスアクション

> **Note**
>
> クエリは、処理するデータによっては複数のアクションを実行することがあるため、特定のクエリを実行するためにどのようなアクセス許可が必要かが常に明確であるとは限りません。詳細については、「クエリアクションの使用」を参照してください。

neptune-db:ReadDataViaQuery

ReadDataViaQuery を使用すると、ユーザーは、クエリを送信して Neptune データベースからデータを読み取ることができます。

アクショングループ: 読み取り専用、読み取り/書き込み。

アクションコンテキストキー: neptune-db:QueryLanguage。

必要なリソース: データベース。

neptune-db:WriteDataViaQuery

WriteDataViaQuery を使用すると、ユーザーは、クエリを送信して Neptune データベースにデータを書き込むことができます。

アクショングループ: 読み取り/書き込み。

アクションコンテキストキー: neptune-db:QueryLanguage。

必要なリソース: データベース。

neptune-db:DeleteDataViaQuery

DeleteDataViaQuery を使用すると、ユーザーは、クエリを送信して Neptune データベースからデータを削除できます。

アクショングループ: 読み取り/書き込み。

アクションコンテキストキー: neptune-db:QueryLanguage。

必要なリソース: データベース。

neptune-db:GetQueryStatus

GetQueryStatus を使用すると、ユーザーは、すべてのアクティブなクエリのステータスを確認できます。

アクショングループ: 読み取り専用、読み取り/書き込み。

アクションコンテキストキー: neptune-db:QueryLanguage。

必要なリソース: データベース。

neptune-db:GetStreamRecords

GetStreamRecords を使用すると、ユーザーは、Neptune からストリームレコードを取得できます。

アクショングループ: 読み取り/書き込み。

アクションコンテキストキー: neptune-db:QueryLanguage。

必要なリソース: データベース。

neptune-db:CancelQuery

CancelQuery を使用すると、ユーザーは、クエリをキャンセルできます。

アクショングループ: 読み取り/書き込み。

必要なリソース: データベース。

一般的なデータアクセスアクション

neptune-db:GetEngineStatus

GetEngineStatus を使用すると、ユーザーは、Neptune エンジンのステータスを確認できます。

アクショングループ: 読み取り専用、読み取り/書き込み。

必要なリソース: データベース。

neptune-db:GetStatisticsStatus

GetStatisticsStatus を使用すると、ユーザーは、データベースについて収集されている統計のステータスを確認できます。

アクショングループ: 読み取り専用、読み取り/書き込み。

必要なリソース: データベース。

neptune-db:GetGraphSummary

GetGraphSummary グラフサマリー API では、グラフの読み取り専用サマリーを取得できます。

アクショングループ: 読み取り専用、読み取り/書き込み。

必要なリソース: データベース。

neptune-db:ManageStatistics

ManageStatistics では、ユーザーは、データベースの統計収集を管理できます。

アクショングループ: 読み取り/書き込み。

必要なリソース: データベース。

neptune-db:DeleteStatistics

DeleteStatistics では、ユーザーは、データベースのすべての統計を削除できます。

アクショングループ: 読み取り/書き込み。

必要なリソース: データベース。

neptune-db:ResetDatabase

ResetDatabase では、ユーザーは、リセットに必要なトークンを取得し、Neptune データベースをリセットできます。

アクショングループ: 読み取り/書き込み。

必要なリソース: データベース。

バルクローダーのデータアクセスアクション

neptune-db:StartLoaderJob

StartLoaderJob では、ユーザーは、バルクローダージョブを開始できます。

アクショングループ: 読み取り/書き込み。

必要なリソース: データベース。

neptune-db:GetLoaderJobStatus

GetLoaderJobStatus では、ユーザーは、バルクローダージョブのステータスを確認できます。

アクショングループ: 読み取り専用、読み取り/書き込み。

必要なリソース: データベース。

neptune-db:ListLoaderJobs

ListLoaderJobs では、ユーザーは、すべてのバルクローダージョブを一覧表示できます。

アクショングループ: リスト専用、読み取り専用、読み取り/書き込み。

必要なリソース: データベース。

neptune-db:CancelLoaderJob

CancelLoaderJob では、ユーザーは、ローダージョブをキャンセルできます。

アクショングループ: 読み取り/書き込み。

必要なリソース: データベース。

機械学習データアクセスアクション

neptune-db:StartMLDataProcessingJob

StartMLDataProcessingJob では、ユーザーは、Neptune ML データ処理ジョブを開始できます。

アクショングループ: 読み取り/書き込み。

必要なリソース: データベース。

neptune-db:StartMLModelTrainingJob

StartMLModelTrainingJob では、ユーザーは、ML モデルトレーニングジョブを開始できます。

アクショングループ: 読み取り/書き込み。

必要なリソース: データベース。

neptune-db:StartMLModelTransformJob

StartMLModelTransformJob では、ユーザーは、ML モデル変換ジョブを開始できます。

アクショングループ: 読み取り/書き込み。

必要なリソース: データベース。

neptune-db:CreateMLEndpoint

CreateMLEndpoint では、ユーザーは、Neptune ML エンドポイントを作成できます。

アクショングループ: 読み取り/書き込み。

必要なリソース: データベース。

neptune-db:GetMLDataProcessingJobStatus

GetMLDataProcessingJobStatus では、ユーザーは、Neptune ML データ処理ジョブのステータスを確認できます。

アクショングループ: 読み取り専用、読み取り/書き込み。

必要なリソース: データベース。

neptune-db:GetMLModelTrainingJobStatus

GetMLModelTrainingJobStatus では、ユーザーは、Neptune ML モデルトレーニングジョブのステータスを確認できます。

アクショングループ: 読み取り専用、読み取り/書き込み。

必要なリソース: データベース。

neptune-db:GetMLModelTransformJobStatus

GetMLModelTransformJobStatus では、ユーザーは、Neptune ML モデル変換ジョブのステータスを確認できます。

アクショングループ: 読み取り専用、読み取り/書き込み。

必要なリソース: データベース。

neptune-db:GetMLEndpointStatus

GetMLEndpointStatus では、ユーザーは、Neptune ML エンドポイントのステータスを確認できます。

アクショングループ: 読み取り専用、読み取り/書き込み。

必要なリソース: データベース。

neptune-db:ListMLDataProcessingJobs

ListMLDataProcessingJobs では、ユーザーは、Neptune ML データ処理ジョブをすべて一覧表示できます。

アクショングループ: リスト専用、読み取り専用、読み取り/書き込み。

必要なリソース: データベース。

neptune-db:ListMLModelTrainingJobs

ListMLModelTrainingJobs では、ユーザーは、Neptune ML モデルトレーニングジョブをすべて一覧表示できます。

アクショングループ: リスト専用、読み取り専用、読み取り/書き込み。

必要なリソース: データベース。

neptune-db:ListMLModelTransformJobs

ListMLModelTransformJobs では、ユーザーは、すべての ML モデルの変換ジョブを一覧表示できます。

アクショングループ: リスト専用、読み取り専用、読み取り/書き込み。

必要なリソース: データベース。

neptune-db:ListMLEndpoints

ListMLEndpoints では、ユーザーは、すべての Neptune ML エンドポイントを一覧表示できます。

アクショングループ: リスト専用、読み取り専用、読み取り/書き込み。

必要なリソース: データベース。

neptune-db:CancelMLDataProcessingJob

CancelMLDataProcessingJob では、ユーザーは、Neptune ML データ処理ジョブをキャンセルできます。

アクショングループ: 読み取り/書き込み。

必要なリソース: データベース。

neptune-db:CancelMLModelTrainingJob

CancelMLModelTrainingJob では、ユーザーは、Neptune ML モデルトレーニングジョブをキャンセルできます。

アクショングループ: 読み取り/書き込み。

必要なリソース: データベース。

neptune-db:CancelMLModelTransformJob

CancelMLModelTransformJob では、ユーザーは、Neptune ML モデル変換ジョブをキャンセルできます。

アクショングループ: 読み取り/書き込み。

必要なリソース: データベース。

neptune-db:DeleteMLEndpoint

DeleteMLEndpoint では、ユーザーは、Neptune ML エンドポイントを削除できます。

アクショングループ: 読み取り/書き込み。

必要なリソース: データベース。

Neptune IAM データアクセスポリシーステートメントでのリソースの指定

データリソースには、データアクションと同様に、neptune-db: プレフィックスがあります。

Neptune データアクセスポリシーでは、アクセスを許可する DB クラスターを次の形式で ARN で指定します。

```
arn:aws:neptune-db:region:account-id:cluster-resource-id/*
```

このようなリソース ARN には次のパートが含まれます。

- *region* は Amazon Neptune DB クラスターの AWS リージョンです。
- *account-id* は DB クラスターの A AWS アカウント番号です。

- *cluster-resource-id* は DB クラスターのリソース ID です。

> **Important**
>
> cluster-resource-id はクラスター識別子とは異なります。Neptune でクラスターリソース ID を検索するには AWS Management Console、対象の DB クラスターの設定セクションを参照してください。

Neptune IAM データアクセスポリシーステートメントで使用可能な条件キー

条件キーを使用すると、IAM ポリシーステートメントで条件を指定して、条件が満たされた場合にのみステートメントが有効になるようにすることができます。

Neptune データアクセスポリシーステートメントで使用できる条件キーは、次のカテゴリに分類されます。

- グローバル条件キー — Neptune がデータアクセスポリシーステートメントでサポートする AWS グローバル条件キーのサブセットを以下に示します。
- サービス固有の条件キー — これらは Neptune がデータアクセスポリシーステートメントで使用するために特別に定義したキーです。現時点では、neptune-db:QueryLanguage は 1 つしかありません。これは、特定のクエリ言語が使用されている場合にのみアクセスを許可します。

AWS データアクセスポリシーステートメントで Neptune がサポートする グローバル条件コンテキストキー

次の表は、Amazon Neptune がデータアクセスポリシーステートメントでの使用をサポートしている AWS グローバル条件コンテキストキーのサブセットを示しています。

データアクセスポリシーステートメントで使用できるグローバル条件キー

条件キー	説明	[Type] (タイプ)
aws:CurrentTime	リクエストの現在日時によってアクセスをフィルタリングします。	String
aws:EpochTime	UNIX エポック値として表されるリクエストの日付と時刻によってアクセスをフィルタリングします。	Numeric

条件キー	説明	[Type] (タイプ)
aws:PrincipalAccount	リクエスト元のプリンシパルが属するアカウントによってアクセスをフィルタリングします。	String
aws:PrincipalArn	リクエストを実行したプリンシパルの ARN によってアクセスをフィルタリングします。	String
aws:PrincipalIsAWSService	呼び出しが AWS サービスプリンシパルによって直接行われている場合にのみアクセスを許可します。	Boolean
aws:PrincipalOrgID	リクエスト元のプリンシパルが属する AWS Organizations 内の組織の識別子でアクセスをフィルタリングします。	String
aws:PrincipalOrgPaths	リクエストを行うプリンシパルの AWS Organizations パスでアクセスをフィルタリングします。	String
aws:PrincipalTag	リクエストを行っているプリンシパルに付けられたタグによってアクセスをフィルタリングします。	String
aws:PrincipalType	リクエストを行っているプリンシパルのタイプによってアクセスをフィルタリングします。	String
aws:RequestedRegion	リクエストで呼び出された AWS リージョンでアクセスをフィルタリングします。	String
aws:SecureTransport	リクエストが SSL を使用して送信された場合にのみアクセスを許可します。	Boolean
aws:SourceIp	リクエスタの IP アドレスによってアクセスをフィルタリングします。	String
aws:TokenIssueTime	一時的セキュリティ認証情報が発行された日付と時刻によってアクセスをフィルタリングします。	String

条件キー	説明	[Type] (タイプ)
aws:UserAgent	リクエスタのクライアントアプリケーションによってアクセスをフィルタリングします。	String
aws:userid	リクエスタのプリンシパル識別子によってアクセスをフィルタリングします。	String
aws:ViaAWSService	AWS サービスがユーザーに代わってリクエストを行った場合にのみアクセスを許可します。	Boolean

Neptune サービス固有の条件キー

Neptune は IAM ポリシーについて、以下のサービス固有の条件キーをサポートします。

Neptune サービス固有の条件キー

条件キー	説明	[Type] (タイプ)
neptune-db:QueryLanguage	使用されているクエリ言語によってアクセスをフィルタリングします。 有効な値は、Gremlin、OpenCypher 、Sparql です。 サポートされるアクションは、ReadDataViaQuery 、WriteDataViaQuery 、DeleteDataViaQuery 、GetQueryStatus 、および CancelQuery です。	String

Neptune IAM データアクセスポリシーの例

以下の例は、Neptune エンジンリリースバージョン 1.2.0.0 で導入された、データプレーン API とアクションのきめ細かいアクセスコントロールを使用するカスタム IAM ポリシーを作成する方法を示しています。

Neptune DB クラスター内のデータへの無制限アクセスを許可するポリシーの例

次のサンプルポリシーは、IAM ユーザーが IAM データベース認証を使用して Neptune DB クラスターに接続し、「*」文字を使用して使用可能なすべてのアクションに一致するのを許可します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "neptune-db:*",
      "Resource": "arn:aws:neptune-db:us-east-1:123456789012:cluster-
ABCD1234EFGH5678IJKL90MNOP/*"
    }
  ]
}
```

上記のリソースセクションには、Neptune の IAM 認証に固有の形式であるリソース ARN が含まれます。ARN を作成するには、「データリソースの指定」を参照してください。IAM 承認 Resource に使用される ARN は、クラスターの作成時に割り当てられる ARN とは異なることに注意してください。

Neptune DB クラスターへの読み取り専用アクセスを許可するポリシーの例

以下のポリシーは、Neptune DB クラスター内のデータへの完全な読み取り専用アクセスのためのアクセス許可を付与します。

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect": "Allow",
      "Action": [
        "neptune-db:Read*",
        "neptune-db:Get*",
        "neptune-db:List*"
      ],
      "Resource": "arn:aws:neptune-db:us-east-1:123456789012:cluster-
ABCD1234EFGH5678IJKL90MNOP/*"
    }
  ]
}
```

Neptune DB クラスターへのすべてのアクセスを拒否するポリシーの例

デフォルトの IAM アクションでは、Allow Effect が付与されない限り、DB クラスターへのアクセスが拒否されます。ただし、次のポリシーでは、特定の AWS アカウントとリージョンの DB クラス

ターへのすべてのアクセスを拒否します。このアクセスは、すべてのAllow効果よりも優先されます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "neptune-db:*",
      "Resource": "arn:aws:neptune-db:us-east-1:123456789012:cluster-
ABCD1234EFGH5678IJKL90MNOP/*"
    }
  ]
}
```

クエリを通じて読み取りアクセスを許可するポリシーの例

次のポリシーは、クエリを使用して Neptune DB クラスターから読み取るアクセス許可のみを付与します。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "neptune-db:ReadDataViaQuery",
      "Resource": "arn:aws:neptune-db:us-east-1:123456789012:cluster-
ABCD1234EFGH5678IJKL90MNOP/*"
    }
  ]
}
```

Gremlin クエリのみを許可するポリシーの例

次のポリシーは、neptune-db:QueryLanguage 条件キーを使用して、Gremlin クエリ言語のみを使用して Neptune をクエリするアクセス許可を付与します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
      "Effect": "Allow",
      "Action": [
        "neptune-db:ReadDataViaQuery",
        "neptune-db:WriteDataViaQuery",
        "neptune-db:DeleteDataViaQuery"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "neptune-db:QueryLanguage": "Gremlin"
        }
      }
    }
  ]
}
```

Neptune ML モデル管理を除くすべてのアクセスを許可するポリシーの例

次のポリシーは、Neptune ML モデル管理機能を除く Neptune グラフ操作へのフルアクセスを許可します。

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect": "Allow",
      "Action": [
        "neptune-db:CancelLoaderJob",
        "neptune-db:CancelQuery",
        "neptune-db:DeleteDataViaQuery",
        "neptune-db:DeleteStatistics",
        "neptune-db:GetEngineStatus",
        "neptune-db:GetLoaderJobStatus",
        "neptune-db:GetQueryStatus",
        "neptune-db:GetStatisticsStatus",
        "neptune-db:GetStreamRecords",
        "neptune-db:ListLoaderJobs",
        "neptune-db:ManageStatistics",
        "neptune-db:ReadDataViaQuery",
        "neptune-db:ResetDatabase",
        "neptune-db:StartLoaderJob",
        "neptune-db:WriteDataViaQuery"
      ],
```

```
      "Resource": "arn:aws:neptune-db:us-east-1:123456789012:cluster-
ABCD1234EFGH5678IJKL90MNOP/*"
    }
  ]
}
```

Neptune ML モデル管理へのアクセスを許可するポリシーの例

このポリシーは、Neptune ML モデル管理機能へのアクセスを許可します。

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect": "Allow",
      "Action": [
        "neptune-db:CancelMLDataProcessingJob",
        "neptune-db:CancelMLModelTrainingJob",
        "neptune-db:CancelMLModelTransformJob",
        "neptune-db:CreateMLEndpoint",
        "neptune-db:DeleteMLEndpoint",
        "neptune-db:GetMLDataProcessingJobStatus",
        "neptune-db:GetMLEndpointStatus",
        "neptune-db:GetMLModelTrainingJobStatus",
        "neptune-db:GetMLModelTransformJobStatus",
        "neptune-db:ListMLDataProcessingJobs",
        "neptune-db:ListMLEndpoints",
        "neptune-db:ListMLModelTrainingJobs",
        "neptune-db:ListMLModelTransformJobs",
        "neptune-db:StartMLDataProcessingJob",
        "neptune-db:StartMLModelTrainingJob",
        "neptune-db:StartMLModelTransformJob"
      ],
      "Resource": "arn:aws:neptune-db:us-east-1:123456789012:cluster-
ABCD1234EFGH5678IJKL90MNOP/*"
    }
  ]
}
```

クエリのフルアクセスを許可するポリシーの例

次のポリシーは、Neptune グラフクエリ操作へのフルアクセスを許可しますが、高速リセット、ストリーム、バルクローダー、Neptune ML モデル管理などの機能へのアクセスは許可しません。

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect": "Allow",
      "Action": [
        "neptune-db:ReadDataViaQuery",
        "neptune-db:WriteDataViaQuery",
        "neptune-db:DeleteDataViaQuery",
        "neptune-db:GetEngineStatus",
        "neptune-db:GetQueryStatus",
        "neptune-db:CancelQuery"
      ],
      "Resource": "arn:aws:neptune-db:us-east-1:123456789012:cluster-
ABCD1234EFGH5678IJKL90MNOP/*"
    }
  ]
}
```

Gremlin クエリにのみフルアクセスを許可するポリシーの例

次のポリシーは、Gremlin クエリ言語を使用した Neptune グラフクエリ操作へのフルアクセスを許可しますが、他の言語でのアクセスや、高速リセット、ストリーム、バルクローダー、Neptune ML モデル管理などの機能へのアクセスは許可しません。

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect": "Allow",
      "Action": [
        "neptune-db:ReadDataViaQuery",
        "neptune-db:WriteDataViaQuery",
        "neptune-db:DeleteDataViaQuery",
        "neptune-db:GetEngineStatus",
        "neptune-db:GetQueryStatus",
        "neptune-db:CancelQuery"
      ],
      "Resource": [
        "arn:aws:neptune-db:us-east-1:123456789012:cluster-ABCD1234EFGH5678IJKL90MNOP/
*"
      ],
      "Condition": {
```

```
        "StringEquals": {
           "neptune-db:QueryLanguage":"Gremlin"
        }
      }
    }
  ]
}
```

高速リセット以外のフルアクセスを許可するポリシーの例

次のポリシーは、高速リセットの使用を除き、Neptune DB クラスターへのフルアクセスを付与します。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "neptune-db:*",
      "Resource": "arn:aws:neptune-db:us-east-1:123456789012:cluster-
ABCD1234EFGH5678IJKL90MNOP/*"
    },
    {
      "Effect": "Deny",
      "Action": "neptune-db:ResetDatabase",
      "Resource": "arn:aws:neptune-db:us-east-1:123456789012:cluster-
ABCD1234EFGH5678IJKL90MNOP/*"
    }
  ]
}
```

Neptune でのサービスにリンクされた IAM ロールの使用

Amazon Neptune は AWS Identity and Access Management (IAM) サービスにリンクされたロール を使用します。サービスにリンクされたロールは、Neptune に直接リンクされた一意のタイプの IAM ロールです サービスにリンクされたロールは Neptune によって事前定義されており、ユーザーに代わってサービスから他の AWS のサービスを呼び出すために必要なすべてのアクセス許可が含まれています。

> **Important**
>
> 特定の管理機能のために、Amazon Neptune は Amazon RDS と共有される運用テクノロジーを使用します。これには、サービスにリンクされたロールと管理 API のアクセス権限が含まれます。

サービスにリンクされたロールを使用することで、必要なアクセス権限を手動で追加する必要がなくなるため、Neptune の使用が簡単になります。Neptune は、サービスにリンクされたロールのアクセス許可を定義します。特に定義されている場合を除き、Neptune のみがそのロールを引き受けることができます。定義されたアクセス許可には、信頼ポリシーとアクセス許可ポリシーが含まれ、そのアクセス許可ポリシーを他の IAM エンティティにアタッチすることはできません。

ロールを削除するには、まず関連リソースを削除します。これにより、リソースへの意図しないアクセスによるアクセス許可の削除が防止され、Neptune リソースは保護されます。

サービスリンクロールをサポートする他のサービスについては、IAM と連携するAWS のサービスを参照して、サービスリンクロール列がはいになっているサービスを探してください。サービスにリンクされたロールに関するドキュメントをサービスで表示するには、[はい] リンクを選択します。

Neptune のサービスにリンクされたロールにおけるアクセス許可

Neptune は、`AWSServiceRoleForRDS`サービスにリンクされたロールを使用して、Neptune と Amazon RDS がデータベースインスタンスに代わって AWS サービスを呼び出すことを許可します。`AWSServiceRoleForRDS` サービスにリンクされたロールは、ロールを継承するために `rds.amazonaws.com` のサービスを信頼します。

ロールのアクセス許可ポリシーは、指定したリソースに対して以下のアクションを実行することを Neptune に許可します。

- `ec2` でのアクション:
  - `AssignPrivateIpAddresses`
  - `AuthorizeSecurityGroupIngress`
  - `CreateNetworkInterface`
  - `CreateSecurityGroup`
  - `DeleteNetworkInterface`
  - `DeleteSecurityGroup`

  - DescribeAvailabilityZones
  - DescribeInternetGateways
  - DescribeSecurityGroups
  - DescribeSubnets
  - DescribeVpcAttribute
  - DescribeVpcs
  - ModifyNetworkInterfaceAttribute
  - RevokeSecurityGroupIngress
  - UnassignPrivateIpAddresses
- sns でのアクション:
  - ListTopic
  - Publish
- cloudwatch でのアクション:
  - PutMetricData
  - GetMetricData
  - CreateLogStream
  - PullLogEvents
  - DescribeLogStreams
  - CreateLogGroup

> **Note**
>
> サービスリンクロールの作成、編集、削除を IAM エンティティ (ユーザー、グループ、ロールなど) に許可するには、アクセス許可を設定する必要があります。次のエラーメッセージが返される場合があります。
> リソースを作成できません。サービスにリンクされたロールを作成するために必要なアクセス許可があることを確認します。それ以外の場合は、時間をおいてからもう一度お試しください。
> このメッセージが表示された場合は、次のアクセス許可が有効であることを確認します。

```
{
    "Action": "iam:CreateServiceLinkedRole",
    "Effect": "Allow",
```

```
    "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
AWSServiceRoleForRDS",
    "Condition": {
        "StringLike": {
            "iam:AWSServiceName":"rds.amazonaws.com"
        }
    }
}
```

詳細については、「IAM ユーザーガイド」の「サービスリンクロール権限」を参照してください。

Neptune 用に作成されたサービスリンクロール

サービスリンクロールを手動で作成する必要はありません。インスタンスまたはクラスターを作成すると、サービスにリンクされたロールが再度 Neptune で自動的に作成されます。

> **Important**
>
> 詳細については、IAM ユーザーガイドのIAM アカウントに新しいロールが表示されるを参照してください。

サービスにリンクされたこのロールを削除したが、再作成する必要がある場合は、同じプロセスで、アカウントにロールを再作成することができます。インスタンスまたはクラスターを作成すると、サービスにリンクされたロールが再度 Neptune で自動的に作成されます。

Neptune のサービスにリンクされたロールの編集

Neptune では、AWSServiceRoleForRDS のサービスにリンクされたロールを編集することはできません。サービスリンクロールを作成した後は、多くのエンティティによってロールが参照される可能性があるため、ロール名を変更することはできません。ただし、IAM を使用したロールの説明の編集はできます。詳細については、「IAM ユーザーガイド」の「サービスリンクロールの編集」を参照してください。

Neptune でのサービスにリンクされたロールの削除

サービスリンクロールが必要な機能またはサービスが不要になった場合には、そのロールを削除することをお勧めします。そうすることで、モニタリングや保守が積極的に行われていない未使用のエン

ティティを排除できます。ただし、関連付けられているサービスにリンクされたロールを削除する前に、すべてのインスタンスとクラスターを削除する必要があります。

サービスにリンクされたロールを削除する前にクリーンアップする

IAM を使用してサービスにリンクされたロールを削除するには、まずそのロールにアクティブなセッションがないことを確認し、そのロールで使用されているリソースをすべて削除する必要があります。

サービスにリンクされたロールにアクティブなセッションがあるかどうかを、IAM コンソールで確認するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. IAM コンソールのナビゲーションペインで [ロール] を選択します。次に、`AWSServiceRoleForRDS` ロールの名前 (チェックボックスではありません) を選択します。
3. 選択したロールの [概要] ページで、[アクセスアドバイザー] タブを選択します。
4. アクセスアドバイザー タブで、サービスにリンクされたロールの最新のアクティビティを確認します。

> **Note**
>
> Neptune で `AWSServiceRoleForRDS` ロールが使用されているかどうかが不明な場合は、このロールの削除を試みることができます。サービスでロールが使用されている場合、削除は失敗し、ロールが使用されている リージョンが表示されます。ロールが使用されている場合は、ロールを削除する前にセッションが終了するのを待つ必要があります。サービスにリンクされたロールのセッションを取り消すことはできません。

`AWSServiceRoleForRDS` ロールを削除する場合、最初にすべてのインスタンスとクラスターを削除する必要があります。

すべてのインスタンスの削除

以下のいずれかの手順を使用して、インスタンスをそれぞれ削除します。

インスタンスを削除するには (コンソール)

1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[インスタンス] を選択します。
3. [Instances] リストで、削除するインスタンスを選択します。
4. [Instance actions] を選択し、[Delete] を選択します。
5. [最終スナップショットを作成しますか?] で、[はい] または [いいえ] を選択します。
6. 前のステップで [はい] を選択した場合は、[最終スナップショット名] に最終スナップショットの名前を入力します。
7. [削除] を選択します。

インスタンスを削除するには (AWS CLI)

AWS CLI コマンドリファレンス の「`delete-db-instance`」を参照してください。

インスタンスを削除するには (API)

`DeleteDBInstance` を参照してください。

すべての クラスターの削除

次のいずれかの手順を使用して単一のクラスターを削除してから、各クラスターに対してこの手順を繰り返します。

クラスターを削除するには (コンソール)

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/neptune/home で Amazon Neptune コンソールを開きます。
2. [Clusters] リストで、削除するクラスターを選択します。
3. [Cluster Actions] を選択してから、[Delete] を選択します。
4. [削除] を選択します。

クラスターを削除するには (CLI)

AWS CLI コマンドリファレンス の「`delete-db-cluster`」を参照してください。

クラスターを削除するには (API)

`DeleteDBCluster` を参照してください。

AWSServiceRoleForRDS サービスにリンクされたロールは、IAM コンソール、IAM CLI、または IAM API を使用して削除することができます。詳細については、「IAM ユーザーガイド」の「サービスリンクロールの削除」を参照してください。

一時的な認証情報を使用した IAM 認証

Amazon Neptune は一時的な認証情報を使用した IAM 認証をサポートしています。

前のセクションのポリシー例などの IAM 認証ポリシーを使用して認証するために、引き受けたロールを使用できます。

一時的なセキュリティ認証情報を使用している場合は、AWS_ACCESS_KEY_ID、AWS_SECRET_ACCESS_KEY、SERVICE_REGION に加えて AWS_SESSION_TOKEN を指定する必要があります。

> **Note**
>
> 一時的なセキュリティ認証情報は、セッショントークンを含めて、指定した期間が過ぎると失効します。
> 新しい認証情報をリクエストするときは、セッショントークンを更新する必要があります。
> 詳細については、「一時的なセキュリティ認証情報を使用して AWS リソースへのアクセスをリクエストする」を参照してください。

以下のセクションでは、一時的な認証情報のアクセスを許可する方法と取得方法について説明します。

一時的な認証情報を使用して認証するには

1. Neptune クラスターにアクセスする権限を持つ IAM ロールを作成します。このロールの作成の詳細については、「the section called “IAM ポリシーのタイプ”」を参照してください。
2. 認証情報へのアクセスを許可するロールに信頼関係を追加します。

   AWS_ACCESS_KEY_ID、AWS_SECRET_ACCESS_KEY、AWS_SESSION_TOKEN を含む一時的な認証情報を取得します。
3. Neptune クラスターに接続し、一時的な認証情報を使用してリクエストに署名します。リクエストへの接続と署名の詳細については、「the section called “接続と署名”」を参照してください。

一時的な認証情報を取得する方法は、環境に応じてさまざまです。

トピック

- AWS CLIを使用した一時的な認証情報の取得
- Neptune IAM 認証用の AWS Lambda のセットアップ
- Neptune IAM 認証用の Amazon EC2 のセットアップ

AWS CLIを使用した一時的な認証情報の取得

AWS Command Line Interface（ AWS CLI) を使用して認証情報を取得するには、まず、 AWS CLI コマンドを実行する AWS ユーザーにロールを引き受けるアクセス許可を付与する信頼関係を追加する必要があります。

Neptune IAM 認証ロールに次の信頼関係を追加します。Neptune IAM 認証ロールがない場合は、the section called “IAM ポリシーのタイプ” を参照してください。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:user/test"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

ロールに信頼関係を追加する方法の詳細については、AWS Directory Service 管理ガイドの既存のロールの信頼関係の編集を参照してください。

まだ Neptune ポリシーがロールにアタッチされていない場合は、新しいロールを作成します。Neptune IAM 認証ポリシーをアタッチし、その後信頼ポリシーを追加します。新しいロールの作成の詳細については、「新しいロールの作成」を参照してください。

> **Note**
>
> 以下のセクションでは、 AWS CLI がインストールされていることを前提としています。

AWS CLI を手動で実行するには

1. AWS CLIを使用して認証情報をリクエストするには、次のコマンドを入力します。ロール ARN、セッション名、プロフィールを独自の値に置き換えます。

```
aws sts assume-role  --role-arn  arn:aws:iam::123456789012:role/NeptuneIAMAuthRole
 --role-session-name test --profile testprofile
```

2. コマンドからの出力例を次に示します。Credentials セクションには、必要な値が含まれます。

> **Note**
>
> これ以降に新しい認証情報を取得する必要があるため、Expiration 値を記録します。

```
{
    "AssumedRoleUser": {
        "AssumedRoleId": "AROA3XFRBF535PLBIFPI4:s3-access-example",
        "Arn": "arn:aws:sts::123456789012:assumed-role/xaccounts3access/s3-access-
example"
    },
    "Credentials": {
        "SecretAccessKey": "9drTJvcXLB89EXAMPLELB8923FB892xMFI",
        "SessionToken": "AQoXdzELDDY//////////
wEaoAK1wvxJY12r2IrDFT2IvAzTCn3zHoZ7YNtpiQLF0MqZye/qwjzP2iEXAMPLEbw/
m3hsj8VBTkPORGvr9jM5sgP+w9IZWZnU+LWhmg
+a5fDi2oTGUYcdg9uexQ4mtCHIHfi4citgqZTgco40Yqr4lIlo4V2b2Dyauk0eYFNebHtYlFVgAUj
+7Indz3LU0aTWk1WKIjHmmMCIoTkyYp/k7kUG7moeEYKSitwQIi6Gjn+nyzM
+PtoA3685ixzv0R7i5rjQi0YE0lf1oeie3bDiNHncmzosRM6SFiPzSvp6h/32xQuZsjcypmwsPSDtTPYcs0+YN/8BRi
IcrxSpnWEXAMPLEXSDFTAQAM6Dl9zR0tXoybnlrZIwMLlMi1Kcgo5OytwU=",
        "Expiration": "2016-03-15T00:05:07Z",
        "AccessKeyId": "ASIAJEXAMPLEXEG2JICEA"
    }
}
```

3. 返された認証情報を使用して環境変数を設定します。

```
export AWS_ACCESS_KEY_ID=ASIAJEXAMPLEXEG2JICEA
export AWS_SECRET_ACCESS_KEY=9drTJvcXLB89EXAMPLELB8923FB892xMFI
```

```
export AWS_SESSION_TOKEN=AQoXdzELDDY//////////
wEaoAK1wvxJY12r2IrDFT2IvAzTCn3zHoZ7YNtpiQLF0MqZye/qwjzP2iEXAMPLEbw/
m3hsj8VBTkPORGvr9jM5sgP+w9IZWZnU+LWhmg
+a5fDi2oTGUYcdg9uexQ4mtCHIHfi4citgqZTgco40Yqr4lIlo4V2b2Dyauk0eYFNebHtYlFVgAUj
+7Indz3LU0aTWk1WKIjHmmMCIoTkyYp/k7kUG7moeEYKSitwQIi6Gjn+nyzM
+PtoA3685ixzv0R7i5rjQi0YE0lf1oeie3bDiNHncmzosRM6SFiPzSvp6h/32xQuZsjcypmwsPSDtTPYcs0+YN/8BRi
IcrxSpnWEXAMPLEXSDFTAQAM6Dl9zR0tXoybnlrZIwMLlMi1Kcgo5OytwU=

export SERVICE_REGION=us-east-1 or us-east-2 or us-west-1 or us-west-2 or ca-
central-1 or
                      sa-east-1 or eu-north-1 or eu-west-1 or eu-west-2 or eu-
west-3 or eu-central-1 or me-south-1 or
                      me-central-1 or il-central-1 or af-south-1 or ap-east-1 or
 ap-northeast-1 or ap-northeast-2 or ap-southeast-1 or ap-southeast-2 or ap-south-1
 or
                      cn-north-1 or cn-northwest-1 or
                      us-gov-east-1 or us-gov-west-1
```

4. 以下のいずれかの方法を使用して接続します。

   - the section called “Gremlin コンソール”
   - the section called “Gremlin Java”
   - the section called “SPARQL Java (RDF4J および Jena)”
   - the section called “Python の例”

スクリプトを使用して認証情報を取得する

1. 以下のコマンドを実行して、jq コマンドをインストールします。スクリプトは、このコマンドを使用して AWS CLI コマンドの出力を解析します。

   ```
   sudo yum -y install jq
   ```

2. テキストエディターで credentials.sh という名前のファイルを作成して、次のテキストを追加します。サービスリージョン、ロール ARN、セッション名、プロフィールを独自の値に置き換えます。

   ```
   #!/bin/bash

   creds_json=$(aws sts assume-role  --role-arn  arn:aws:iam::123456789012:role/
   NeptuneIAMAuthRole --role-session-name test --profile testprofile)
   ```

```
export AWS_ACCESS_KEY_ID=$(echo "$creds_json" | jq .Credentials.AccessKeyId |tr -d
 '"')
export AWS_SECRET_ACCESS_KEY=$(echo "$creds_json" |
 jq .Credentials.SecretAccessKey| tr -d '"')
export AWS_SESSION_TOKEN=$(echo "$creds_json" | jq .Credentials.SessionToken|tr -d
 '"')

export SERVICE_REGION=us-east-1 or us-east-2 or us-west-1 or us-west-2 or ca-
central-1 or
                      sa-east-1 or eu-north-1 or eu-west-1 or eu-west-2 or eu-
west-3 or eu-central-1 or me-south-1 or
                      me-central-1 or il-central-1 or af-south-1 or ap-east-1 or
 ap-northeast-1 or ap-northeast-2 or ap-southeast-1 or ap-southeast-2 or ap-south-1
 or
                      cn-north-1 or cn-northwest-1 or
                      us-gov-east-1 or us-gov-west-1
```

3. 以下のいずれかの方法を使用して接続します。

- the section called “Gremlin コンソール”
- the section called “Gremlin Java”
- the section called “SPARQL Java (RDF4J および Jena)”
- the section called “Python の例”

Neptune IAM 認証用の AWS Lambda のセットアップ

AWS Lambda には、Lambda 関数が実行されるたびに自動的に認証情報が含まれます。

最初に Lambda サービスにロールを引き受けるアクセス権限を与える信頼関係を追加します。

Neptune IAM 認証ロールに次の信頼関係を追加します。Neptune IAM 認証ロールがない場合は、the section called “IAM ポリシーのタイプ” を参照してください。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
```

```
      "Action": "sts:AssumeRole"
    }
  ]
}
```

ロールに信頼関係を追加する方法の詳細については、AWS Directory Service管理ガイドの既存のロールの信頼関係の編集を参照してください。

まだ Neptune ポリシーがロールにアタッチされていない場合は、新しいロールを作成します。Neptune IAM 認証ポリシーをアタッチし、その後信頼ポリシーを追加します。新しいロールの作成の詳細については、AWS Directory Service 管理ガイドの新しいロールの作成を参照してください。

Lambda から Neptune にアクセスするには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/lambda/ で AWS Lambda コンソールを開きます。
2. Python バージョン 3.6 用に新しい Lambda 関数を作成します。
3. Lambda 関数に `AWSLambdaVPCAccessExecutionRole` ロールを割り当てます。これは、VPC のみの Neptune リソースにアクセスするために必要です。
4. Lambda 関数に Neptune 認証 IAM ロールを割り当てます。

   詳細については、AWS Lambda デベロッパーガイドのAWS Lambda 関数のエラーを参照してください。
5. IAM 認証の Python サンプルを Lambda 関数コードにコピーします。

   サンプルおよびサンプルコードの詳細については、「the section called “Python の例”」を参照してください。

Neptune IAM 認証用の Amazon EC2 のセットアップ

Amazon EC2 ではインスタンスプロファイルを使用して、認証情報を自動的に提供することができます。詳細については、IAM ユーザーガイドのインスタンスプロファイルの使用を参照してください。

最初に Amazon EC2 サービスにロールを引き受けるアクセス権限を与える信頼関係を追加します。

Neptune IAM 認証ロールに次の信頼関係を追加します。Neptune IAM 認証ロールがない場合は、the section called “IAM ポリシーのタイプ” を参照してください。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

ロールに信頼関係を追加する方法の詳細については、AWS Directory Service 管理ガイドの既存のロールの信頼関係の編集を参照してください。

まだ Neptune ポリシーがロールにアタッチされていない場合は、新しいロールを作成します。Neptune IAM 認証ポリシーをアタッチし、その後信頼ポリシーを追加します。新しいロールの作成の詳細については、AWS Directory Service 管理ガイドの新しいロールの作成を参照してください。

スクリプトを使用して認証情報を取得する

1. 以下のコマンドを実行して、jq コマンドをインストールします。スクリプトはこのコマンドを使用して、curl コマンドの出力を解析します。

   ```
   sudo yum -y install jq
   ```

2. テキストエディターで `credentials.sh` という名前のファイルを作成して、次のテキストを追加します。サービスリージョンを独自の値に置き換えます。

   ```
   role_name=$( curl -s http://169.254.169.254/latest/meta-data/iam/security-
   credentials/ )
   creds_json=$(curl -s http://169.254.169.254/latest/meta-data/iam/security-
   credentials/${role_name})

   export AWS_ACCESS_KEY_ID=$(echo "$creds_json" | jq .AccessKeyId |tr -d '"')
   export AWS_SECRET_ACCESS_KEY=$(echo "$creds_json" | jq .SecretAccessKey| tr -d '"')
   export AWS_SESSION_TOKEN=$(echo "$creds_json" | jq .Token|tr -d '"')
   ```

```
export SERVICE_REGION=us-east-1 or us-east-2 or us-west-1 or us-west-2 or ca-
central-1 or
                      sa-east-1 or eu-north-1 or eu-west-1 or eu-west-2 or eu-
west-3 or eu-central-1 or me-south-1 or
                      me-central-1 or il-central-1 or af-south-1 or ap-east-1 or
 ap-northeast-1 or ap-northeast-2 or ap-southeast-1 or ap-southeast-2 or ap-south-1
 or
                      cn-north-1 or cn-northwest-1 or
                      us-gov-east-1 or us-gov-west-1
```

3. source コマンドを使用して、bash シェルでスクリプトを実行します。

```
source credentials.sh
```

さらに良い方法は、このスクリプトのコマンドを EC2 インスタンス上の .bashrc ファイルに追加して、ログイン時に自動的に呼び出されるようにします。これにより、一時的な認証情報が Gremlin コンソールで使用できるようになります。

4. 以下のいずれかの方法を使用して接続します。

- the section called “Gremlin コンソール”
- the section called “Gremlin Java”
- the section called “SPARQL Java (RDF4J および Jena)”
- the section called “Python の例”

Amazon Neptune リソースのロギングとモニタリング

Amazon Neptune では、パフォーマンスと使用状況をモニタリングするためのさまざまな方法をサポートしています。

- クラスターのステータス – クラスターのグラフデータベースエンジンの正常性をチェックします。詳細については、「the section called “インスタンスのステータス”」を参照してください。
- Amazon CloudWatch – Neptune は にメトリクスを自動的に送信 CloudWatch し、 CloudWatch アラームもサポートします。詳細については、「the section called “の使用 CloudWatch”」を参照してください。
- 監査ログファイル – Neptune コンソールを使用して、データベースログファイルを表示、ダウンロード、調査します。詳細については、「the section called “Neptune による監査ログ ”」を参照してください。

- Amazon CloudWatch Logs へのログの発行 – 監査ログデータを Amazon Logs のロググループに発行するように Neptune DB CloudWatch クラスターを設定できます。 CloudWatch Logs を使用すると、ログデータのリアルタイム分析を実行し、 CloudWatch を使用してアラームを作成し、メトリクスを表示し、 CloudWatch ログを使用してログレコードを耐久性の高いストレージに保存できます。詳細については、「Neptune CloudWatch ログ」を参照してください。
- AWS CloudTrail — Neptune は、 を使用した API ログ記録をサポートしています CloudTrail。詳細については、「the section called “を使用した Neptune API コールのログ記録 AWS CloudTrail”」を参照してください。
- タグ付け – タグを使用してメタデータを Neptune リソースに追加し、タグに基づいて使用量を追跡します。詳細については、「the section called “Neptune リソースにタグを付ける”」を参照してください。

Amazon Neptune のコンプライアンス検証

AWS のサービス が特定のコンプライアンスプログラムの範囲内にあるかどうかを確認するには、コンプライアンスプログラムAWS のサービス による対象範囲内のコンプライアンスプログラムを参照し、関心のあるコンプライアンスプログラムを選択します。一般的な情報については、AWS 「コンプライアンスプログラム」を参照してください。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「 でのレポートのダウンロード AWS Artifact」の」を参照してください。

を使用する際のお客様のコンプライアンス責任 AWS のサービス は、お客様のデータの機密性、貴社のコンプライアンス目的、適用される法律および規制によって決まります。 は、コンプライアンスに役立つ以下のリソース AWS を提供しています。

- セキュリティとコンプライアンスのクイックスタートガイド – これらのデプロイガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスに重点を置いたベースライン環境 AWS を にデプロイする手順について説明します。
- アマゾン ウェブ サービスにおける HIPAA セキュリティとコンプライアンスのアーキテクチャ — このホワイトペーパーでは、企業が AWS を使用して HIPAA 対象アプリケーションを作成する方法について説明します。

> **Note**
>
> すべて AWS のサービス HIPAA の対象となるわけではありません。詳細については、「HIPAA 対応サービスのリファレンス」を参照してください。

- AWS コンプライアンスリソース – このワークブックとガイドのコレクションは、お客様の業界や地域に適用される場合があります。
- AWS カスタマーコンプライアンスガイド – コンプライアンスの観点から責任共有モデルを理解します。このガイドには、ガイダンスを保護し AWS のサービス 、複数のフレームワーク (米国国立標準技術研究所 (NIST)、Payment Card Industry Security Standards Council (PCI)、国際標準化機構 (ISO) を含む) のセキュリティコントロールにマッピングするためのベストプラクティスがまとめられています。
- 「 デベロッパーガイド」の「 ルールによるリソースの評価」 – この AWS Config サービスは、リソース設定が社内プラクティス、業界ガイドライン、および規制にどの程度準拠しているかを評価します。 AWS Config
- AWS Security Hub – これにより AWS のサービス 、 内のセキュリティ状態を包括的に確認できます AWS。Security Hub では、セキュリティコントロールを使用して AWS リソースを評価し、セキュリティ業界標準とベストプラクティスに対するコンプライアンスをチェックします。サポートされているサービスとコントロールのリストについては、「Security Hub のコントロールリファレンス」を参照してください。
- Amazon GuardDuty — これにより AWS アカウント、疑わしいアクティビティや悪意のあるアクティビティがないか環境を監視することで、、、ワークロード、コンテナ、データに対する潜在的な脅威 AWS のサービス を検出します。 GuardDuty は、特定のコンプライアンスフレームワークで義務付けられている侵入検知要件を満たすことで、PCI DSS などのさまざまなコンプライアンス要件への対応に役立ちます。
- AWS Audit Manager – これにより AWS のサービス 、 AWS 使用状況を継続的に監査し、リスクの管理方法と規制や業界標準への準拠を簡素化できます。

Amazon Neptune の耐障害性

AWS グローバルインフラストラクチャは、 AWS リージョンとアベイラビリティーゾーンを中心に構築されています。 AWS リージョンは、低レイテンシー、高スループット、および高度に冗長なネットワークで接続された、物理的に分離された複数のアベイラビリティーゾーンを提供します。アベイラビリティーゾーンでは、アベイラビリティーゾーン間で中断せずに、自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリ

ティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性、耐障害性、およびスケーラビリティが優れています。

Amazon Neptune DB クラスターは、最低 2 つのアベイラビリティーゾーンに最低 2 つのサブネットがある Amazon VPC 内にのみ作成できます。少なくとも 2 つのアベイラビリティーゾーンにまたがってクラスターインスタンスを配布することで、万一アベイラビリティーゾーンに障害が発生した場合でも、Neptune では、DB クラスター内でインスタンスを使用できます。Neptune DB クラスターのクラスターボリュームは、データ損失の可能性が少ない耐久性のあるストレージを提供するために、常に 3 つのアベイラビリティーゾーンにまたがっています。

AWS リージョンとアベイラビリティーゾーンの詳細については、AWS 「グローバルインフラストラクチャ」を参照してください。

既存のグラフを Amazon Neptune に移行する

既存のグラフデータを別のデータストアから Amazon Neptune に移行するのに役立つツールとテクニックがいくつかあります。

単純な移行ワークフローには、以下のステップが含まれます。

- 既存のストアから Amazon Simple Storage Service (Amazon S3) にデータをエクスポートします。
- インポート用にクリーンアップしてフォーマットします。
- Neptune 一括ローダー を使用して Neptune DB クラスターにロードします。
- Gremlin または SPARQL アプリケーションが、Neptune が提供する対応するエンドポイントを使用するように設定します。

> **Note**
>
> Neptune クラスターは、アプリケーションがアクセスできる VPC で実行されている必要があります。

データの保存場所に応じて、これらの手順のいくつかを簡素化および自動化する方法があります。

トピック

- Neo4j から Amazon Neptune への移行
- 既存のグラフを Apache TinkerPop Gremlin サーバーから Amazon Neptune に移行する
- 既存のグラフを RDF トリプルストアから Amazon Neptune に移行する
- AWS Database Migration Service (AWS DMS) を使用してリレーショナルデータベースまたは NoSQL データベースから Amazon Neptune に移行する
- Blazegraph から Amazon Neptune への移行

Neo4j から Amazon Neptune への移行

Neo4j と Amazon Neptune はどちらもグラフデータベースであり、ラベル付きプロパティグラフデータモデルをサポートするオンラインのトランザクショングラフワークロード向けに設計されています。これらの類似点により、Neptune は現在の Neo4j アプリケーションの移行を検討 しているお客様にとって一般的な選択肢となっています。ただし、この 2 つのデータベースには、言語、機能サポート、運用特性、サーバーアーキテクチャ、およびストレージ機能に違いがあるため、移行は単純にリフトアンドシフトではありません。

このページでは、移行プロセスを構成し、Neo4j グラフアプリケーションを Neptune に移行する前に考慮すべき点について説明します。これらの考慮事項は、Community、Enterprise、または Aura データベースのいずれを利用しているかにかかわらず、あらゆる Neo4j グラフアプリケーションに一般的に当てはまります。ソリューションはそれぞれ異なり、追加の手順が必要になる場合がありますが、移行はすべて同じ一般的なパターンに従います。

以下のセクションで説明する各手順には、移行プロセスを簡略化するための考慮事項と推奨事項が含まれています。さらに、オープンソースのツールやプロセスを説明するブログ記事、推奨アーキテクチャオプションが記載された機能の互換性に関するセクションもあります。

トピック

- Neo4j から Neptune への移行に関する全般的情報
- Neo4j から Neptune への移行の準備
- Neo4j から Neptune へ移行するときのインフラストラクチャのプロビジョニング
- Neo4j から Neptune へのデータ移行
- Neo4j から Neptune へのアプリケーションの移行
- Neptune の Neo4j との互換性
- Neptune 上の openCypherで実行するように Cypher クエリを書き直す
- Neo4j から Neptune へ移行するためのリソース

Neo4j から Neptune への移行に関する全般的情報

Neptune は openCypher クエリ言語をサポートしているため、Bolt プロトコルまたは HTTPS を使用するほとんどの Neo4j ワークロードを Neptune に移行できます。ただし、openCypher はオープンソースの仕様であり、Neo4j など, 他のデータベースでサポートされている機能のほとんどが含まれていますが、すべてではありません。

多くの点で互換性があるにもかかわらず、Neptune は Neo4j の完全互換品ではありません。Neptune は、Neo4j とはアーキテクチャ的に異なる高可用性や高耐久性などのエンタープライズ機能を備えたフルマネージドグラフデータベースサービスです。Neptune はインスタンスベースであり、1 つのプライマリライターインスタンスと最大 15 個のリードレプリカインスタンスを備えているため、読み取り容量を水平方向に拡張できます。Neptune サーバーレスを使用すると、クエリ量に応じてコンピューティング容量を自動的にスケールアップまたはスケールダウンできます。これは、データを追加すると自動的にスケーリングされる Neptune ストレージとは無関係です。

Neptune は、オープンソースの openCypher 標準仕様バージョン 9 をサポートしています。AWS では、オープンソースは誰にとっても良いことだと考えており、オープンソースの価値をお客様にもたらし、AWS のオペレーショナルエクセレンスをオープンソースコミュニティにもたらすことに尽力しています。

ただし、Neo4j 上で動作する多くのアプリケーションは、オープンソースではなく、Neptune がサポートしていない独自の機能も使用しています。例えば、Neptune は APOC プロシージャ、一部の Cypher 固有の句と関数、および `Char`、`Date`、または `Duration` データ型をサポートしていません。Neptune は、欠落しているデータ型をサポートされているデータ型に自動キャストします。

openCypher に加えて、Neptune はプロパティグラフ用の Apache TinkerPop Gremlin クエリ言語 (および RDF データ用の SPARQL) もサポートしています。Gremlin は同じプロパティグラフで openCypher と相互運用が可能で、多くの場合、openCypher が提供していない機能を Gremlin を使用して提供することができます。以下は、この 2 つの言語を簡単に比較したものです。

	openCypher	Gremlin
[Style] (スタイル)	宣言型	命令型

構文	パターンマッチング
	`Match p=(a)-[:route]->(d)`
	`WHERE a.code='ANC'`
	`RETURN p`
		トラバーサルベース
		`g.V().has('code', 'ANC').`
		`out('route').path().`
		`by(elementMap())`

使いやすさ:	SQL に触発され、プログラマーでなくても読みやすい	Java などのプログラミング言語と同様、習得までの時間が短い
柔軟性	低	高

	openCypher	Gremlin
クエリサポート	文字列ベースのクエリ	文字列ベースのクエリ、またはクライアントライブラリによってサポートされるインラインコード
クライアント	HTTPS および Bolt	HTTPS とウェブソケット

一般に、Neo4j と Neptune はどちらもラベル付きプロパティグラフ (LPG) データをサポートしているため、Neo4j から Neptune に移行するためにデータモデルを変更する必要はありません。ただし、Neptune には、パフォーマンスを最適化するために活用できるアーキテクチャとデータモデルの違いがいくつかあります。例:

- Neptune ID は一級市民として扱われます。
- Neptune は AWS Identity and Access Management (IAM) ポリシーを使用して、柔軟かつ詳細な方法でグラフデータへのアクセスを保護します。
- Neptune には、Jupyter ノートブックを使用してクエリを実行し、結果を視覚化する方法がいくつか用意されています。Neptune はサードパーティ製視覚化ツールとも連携します。
- >Neptune には Neo4j グラフデータサイエンス (GDS) ライブラリの完全互換品はありませんが、Neptune は現在、さまざまなソリューションを通じてグラフ分析をサポートしています。例えば、いくつかのサンプルノートブックは、Python 環境で Neptune と AWS Pandas SDK との統合を活用してグラフデータの分析を実行する方法を示しています。

さらに質問がある場合は、AWS サポートに連絡するか、AWS アカウントチームに連絡してください。お客様からのフィードバックをもとに、ニーズを満たす新機能の優先順位を決定します。

Neo4j から Neptune への移行の準備

移行へのアプローチ

Neo4j アプリケーションを Neptune に移行する場合、リプラットフォームまたはリファクタリング/リアーキテクチャの2つの戦略のうちの 1 つをお勧めします。移行戦略の詳細については、Stephen Orban によるブログ記事「アプリケーションをクラウドに移行するための 6 つの戦略」を参照してください。

リプラットフォームアプローチは、lift-tinker-and-shift と呼ばれることもあり、次のステップが含まれます。

- アプリケーションによって満たされる想定のユースケースを特定します。
- Neptune の機能を使用して、既存のグラフデータモデルとアプリケーションアーキテクチャを変更し、これらのワークロードのニーズに最も対応できるようにします。
- ソースアプリケーションのデータ、クエリ、その他の部分をターゲットモデルとアーキテクチャに移行する方法を決定します。

この逆算アプローチにより、まったく新しいプロジェクトの場合に設計するような種類の Neptune ソリューションにアプリケーションを移行できます。

これとは対照的に、リファクタリングアプローチには以下が含まれます。

- インフラストラクチャ、データ、クエリ、アプリケーション機能など、既存の実装の構成要素を特定します。
- 同等の実装を構築するために使用できるものを Neptune で見つけます。

この逆算アプローチは、ある実装を別の実装に置き換えることを目的としています。

実際には、これら 2 つのアプローチを組み合わせて採用することになるでしょう。ユースケースから始めて、ターゲットの Neptune アーキテクチャを設計しますが、次に、既存の Neo4j 実装に目を向けて、維持しなければならない制約や不変条件を特定することもできます。例えば、他の外部システムとの統合を継続したり、グラフアプリケーションのコンシューマーに特定の API を提供し続ける必要があるかもしれません。この情報を使用して、ターゲットモデルに移行するデータがすでに存在し、どのデータを他の場所にソースする必要があるかを判断できます。

また、アプリケーションが意図している仕事に関する最良の情報源として、Neo4j 実装の特定の部分を分析することから始めることもできます。既存のアプリケーションにおけるこのような考古学は、Neptuneの機能を使用するように設計できるユースケースを定義するのに役立ちます。

Neptune を使用して新しいアプリケーションを構築する場合でも、Neo4j から既存のアプリケーションを移行する場合でも、ユースケースから逆算して、ビジネスニーズに対応するデータモデル、クエリのセット、およびアプリケーションアーキテクチャを設計することをお勧めします。

Neptune と Neo4j のアーキテクチャの違い

お客様が最初に Neo4j から NNeptune へのアプリケーションの移行を検討するとき、インスタンスサイズに基づいて類似品を比較したくなることがよくあります。ただし、Neo4j と NNeptune のアーキテクチャには根本的な違いがあります。Neo4j は、データの読み込み、データ ETL、アプリケーションクエリ、データストレージ、管理操作のすべてが EC2 インスタンスなどの同じコンピューティングリソース内で行われるオールインワンアプローチに基づいています。

対照的に、Neptune は OLTP に焦点を当てたグラフデータベースであり、アーキテクチャによって責任が分離され、リソースが分離されているため、動的かつ独立してスケーリングできます。

Neo4j から Neptune に移行するときは、アプリケーションのデータ耐久性、可用性、スケーラビリティの要件を判断してください。Neptune のクラスターアーキテクチャは、高い耐久性、可用性、スケーラビリティを必要とするアプリケーションの設計を簡素化します。Neptune のクラスターアーキテクチャを理解すれば、これらの要件を満たす Neptune クラスタートポロジを設計できます。

Neo4j のクラスターアーキテクチャ

多くの実稼働アプリケーションでは、Neo4j の因果クラスタリングを使用して, データの耐久性、高可用性、スケーラビリティを実現しています。Neo4j のクラスタリングアーキテクチャは、コアサーバーインスタンスとリードレプリカインスタンスを使用します。

- コアサーバーは Raft プロトコルを使用してデータを複製することにより、データの耐久性と耐障害性を実現します。
- リードレプリカは、トランザクションログ配布を使用して、読み取りスループットの高いワークロードのデータを非同期でレプリケートします。

コアサーバーであれ、リードレプリカであれ、クラスター内のすべてのインスタンスにはグラフデータの完全なコピーが含まれます。

Neptune のクラスターアーキテクチャ

Neptune クラスターは、1 つのプライマリライターインスタンスと最大 15 個のリードレプリカインスタンスで構成されます。クラスター内のすべてのインスタンスは、インスタンスとは別の同じ基盤となる分散ストレージサービスを共有します。

- プライマリライターインスタンスは、データベースへのすべての書き込み操作を調整し、垂直方向に拡張できるため、さまざまな書き込みワークロードを柔軟にサポートできます。読み取りオペレーションもサポートします。
- リードレプリカインスタンスは、基盤となるストレージボリュームからの読み取り操作をサポートし、高い読み取りワークロードに対応するように水平方向に拡張できます。また、プライマリインスタンスのフェイルオーバーターゲットとして機能して、可用性を高めます。

> **Note**
>
> 書き込みワークロードが多い場合は、読み取り側がデータの変更に対して一貫性を保てるように、リードレプリカインスタンスをライターインスタンスと同じサイズにスケーリングするのが最善です。

- 基盤となるストレージボリュームは、データベース内のデータの増加に応じて、ストレージ容量を最大 128 テビバイト (TiB) まで自動的に拡張します。

インスタンスサイズは動的で独立しています。各インスタンスはクラスターの実行中にサイズを変更でき、リードレプリカはクラスターの実行中に追加または削除できます。

Neptune サーバーレス機能では、需要の増減に応じてコンピューティング容量を自動的にスケールアップまたはスケールダウンできます。これにより、管理オーバーヘッドが軽減されるだけでなく、パフォーマンスを低下させたり、過剰なプロビジョニングを行ったりすることなく、需要の大幅な急増に対応するようにデータベースを構成できます。

Neptune クラスターは最大 7 日間停止できます。

Neptune は自動スケーリングもサポートしており、ワークロードに基づいてリーダーインスタンスのサイズを自動的に調整します。

Neptune のグローバルデータベース機能を使用すると、最大 5 つの他のリージョンにクラスターをミラーリングできます。

また、Neptune は、耐障害性を持つように設計されています。

- クラスター内のすべてのインスタンスにデータストレージを提供するクラスターボリュームは、1 つの AWS リージョン 内の複数のアベイラビリティーゾーン (AZ) にまたがります。各 AZ には、クラスターデータの完全なコピーが含まれます。
- プライマリインスタンスが使用できなくなると、Neptune は一般に 30 秒未満で、データ損失なしで既存のリードレプリカに自動的にフェイルオーバーします。クラスターに既存のリードレプリカがない場合、Neptune は自動的に新しいプライマリインスタンスをプロビジョニングします。これもデータ損失ゼロです。

つまり、Neo4j の因果クラスターからNNeptune に移行すると、高いデータ耐久性と高可用性を実現するためにクラスタートポロジーを明示的に設計する必要がありません。これにより、予想される読み取りおよび書き込みワークロードや、可用性要件の増加に合わせて、いくつかの方法でクラスターのサイズを決定できます。

- 読み取り操作をスケーリングするには、リードレプリカインスタンスを追加するか、Neptune サーバーレス機能を有効にします。
- 可用性を高めるには、クラスターのプライマリインスタンスとリードレプリカを複数のアベイラビリティーゾーン (AZ) に分散します。
- フェイルオーバー時間を短縮するには、プライマリのフェイルオーバーターゲットとして機能するリードレプリカインスタンスを少なくとも 1 つプロビジョニングしてください。各レプリカに優先度を割り当てることで、障害後にリードレプリカインスタンスがプライマリに昇格される順序を決めることができます。フェイルオーバーターゲットには、プライマリに昇格した場合にアプリケーションの書き込みワークロードを処理できるインスタンスクラスがあることを確認するのがベストプラクティスです。

Neptune と Neo4j のデータストレージの違い

Neptune は、ネイティブのクアッドモデルに基づくグラフデータモデルを使用します。データを Neptune に移行する場合、データモデルとストレージレイヤーのアーキテクチャにはいくつかの違いがあり、Neptune が提供する分散型でスケーラブルな共有ストレージを最大限に活用するために注意する必要があります。

- Neptune は明示的に定義されたスキーマや制約を使用しません。事前にスキーマを定義しなくても、ノード、エッジ、プロパティを動的に追加できます。Neptune は、「Neptune の制限」に記載されている場合を除き、保存されるデータの値とタイプを制限しません。Neptune のストレージアーキテクチャの一部として、データは最も一般的なアクセスパターンの多くを処理する方法で自動的にインデックスが付けられます。このストレージアーキテクチャにより、データベース

スキーマの作成と管理、およびインデックスの最適化に伴う運用上のオーバーヘッドがなくなります。

- Neptune は、データベースのストレージニーズが最大 128 テビバイト (TiB) まで増加するにつれて、10 GB のチャンクで自動的にスケーリングする独自の分散型共有ストレージアーキテクチャを提供します。このストレージレイヤーは信頼性、耐久性、耐障害性に優れ、3 つのアベイラビリティーゾーンのそれぞれに 2 回ずつ、合計 6 回データがコピーされます。デフォルトでは、すべての Neptune クラスターに可用性が高く耐障害性のあるデータストレージレイヤーを提供します。Neptune のストレージアーキテクチャは、コストを削減し、将来のデータ増加に対応するためにストレージをプロビジョニングしたり、過剰にプロビジョニングしたりする必要をなくします。

データを Neptune に移行する前に、Neptune のプロパティグラフデータモデルとトランザクションセマンティクスについてよく理解しておくとよいでしょう。

Neptune と Neo4j の運用上の違い

Neptune は、フルマネージドサービスであり、Neo4j Enterprise や Community Edition など、オンプレミスまたは自己管理型のデータベースを使用する場合に実行しなければならない通常の運用タスクの多くを自動化します。

- 自動バックアップ — Neptune はクラスターボリュームを自動的にバックアップし、指定した保持期間 (1 ~ 35 日) だけバックアップを保持します。これらのバックアップは連続的かつ増分的であるため、保持期間内の任意の時点にすばやく復元できます。バックアップデータが書き込まれるときに、データベースサービスのパフォーマンスに影響が出たり、中断が発生したりすることはありません。
- 手動スナップショット — Neptune では DB クラスターのストレージボリュームのスナップショットを作成し、DB クラスター全体をバックアップできます。この種類のスナップショットは、データベースの復元、コピーの作成、アカウント間の共有に使用できます。
- クローニング — Neptune は、コスト効率の高いデータベースのクローンをすばやく作成できるクローニング機能をサポートしています。クローンはコピーオンライトプロトコルを使用するため、作成後の追加スペースは最小限で済みます。データベースのクローニングは、元のクラスターを中断することなく Neptune の新機能やアップグレードを試すのに効果的な方法です。
- 監視 — Neptune には、クラスターのパフォーマンスと使用状況を監視するためのさまざまな方法が用意されています。
  - インスタンスの状態
  - Amazon CloudWatch および AWS CloudTrail との統合

  - 監査ログ機能
  - イベント通知
  - タグ付け
- セキュリティ — Neptune はデフォルトで安全な環境を提供します。クラスターは、他のリソースからネットワークを分離するプライベート VPC 内にあります。すべてのトラフィックは SSL を使用して暗号化され、保管時のデータはすべて AWS KMS を使用して暗号化されます。

  さらに、Neptune は AWS Identity and Access Management (IAM) と統合して認証機能も提供します。IAM 条件キーを指定することで、IAM ポリシーを使用してデータアクションに対するきめ細かなアクセス制御ができます。

Neptune と Neo4j のツーリングと統合の違い

Neptune の統合とツールのアーキテクチャは Neo4j とは異なるため、アプリケーションのアーキテクチャに影響を与える可能性があります。Neptune はクラスターのコンピューティングリソースを使用してクエリを処理しますが、フルテキスト検索 (OpenSearch を使用) や ETL (Glue を使用) などの機能には、クラス最高の AWS サービスを活用します。これらの統合の完全なリストについては、「Neptune 統合」を参照してください。

Neo4j から Neptune へ移行するときのインフラストラクチャのプロビジョニング

Amazon Neptune クラスターは、ストレージ、書き込み容量、読み取り容量の 3 つの次元でスケーリングできるように構築されています。以下のセクションでは、移行時に考慮すべき具体的なオプションについて説明します。

ストレージのプロビジョニング

Neptune クラスターのストレージは、自動的にプロビジョニングされるため、ユーザー側の管理オーバーヘッドはありません。クラスターのストレージニーズが増えるにつれて、10 GB チャンクで動的にサイズ変更されます。そのため、将来のデータ増加に対応するために、ストレージを見積もってプロビジョニングしたり、過剰にプロビジョニングしたりする必要がありません。

書き込み容量のプロビジョニング

Neptune は 1 つのライターインスタンスを提供しており、Neptune の料金ページに記載されている任意のインスタンスサイズに合わせて垂直方向にスケーリングできます。ライターインスタンスにデータを読み書きする場合、すべてのトランザクションは ACID に準拠し、Neptune でのトランザクション分離レベル で定義されているデータ分離が行われます。

ライターインスタンスに最適なサイズを選択するには、負荷テストを実行して、ワークロードに最適なインスタンスサイズを決定する必要があります。Neptune 内のインスタンスは、DB インスタンスクラスを変更することでいつでもサイズを変更できます。開始時のインスタンスサイズは、クラスターをプロビジョニングするときに最適なインスタンスサイズを見積もる で説明するように、同時実行性と平均クエリレイテンシーに基づいて見積もることができます。

書き込み容量のプロビジョニング

Neptune は、リードレプリカインスタンスをクラスター内で最大 15 個 (または Neptune グローバルデータベースではそれ以上) 追加することで水平方向にスケーリングできるように構築されています。また、Neptune の料金表ページで確認できる任意のインスタンスサイズに合わせて垂直方向にもスケーリングできるように構築されています。すべての Neptune リードレプリカインスタンスは同じ基盤となるストレージボリュームを使用するため、遅延を最小限に抑えてデータを透過的に複製できます。

Neptune クラスター内の読み取りリクエストの水平スケーリングを有効にするだけでなく、リードレプリカはライターインスタンスのフェイルオーバーターゲットとしても機能し、高可用性を実現

します。クラスター内のリードレプリカの適切な数と配置を決定する方法については、「Amazon Neptune 基本操作ガイドライン」を参照してください。

接続やワークロードが予測できないアプリケーション向けに、Neptune は、指定した条件に基づいて Neptune レプリカの数を自動的に調整できる自動スケーリング機能もサポートしています。

リードレプリカインスタンスの最適なサイズと数を決定するには、負荷テストを実行して、サポートする必要のある読み取りワークロードの特性を判断する必要があります。Neptune 内のインスタンスは、DB インスタンスクラスを変更することでいつでもサイズを変更できます。開始時のインスタンスサイズは、次のセクションで説明するように、同時実行性と平均クエリレイテンシーに基づいて見積もることができます。

Neptune サーバーレスを使用して、必要に応じてリーダーインスタンスとライターインスタンスを自動的にスケーリングする

予想されるワークロードに必要なコンピューティング容量を見積もることができると便利な場合が多いですが、読み取りと書き込みの容量を自動的にスケールアップまたはスケールダウンするように Neptune サーバーレス機能を設定できます。これにより、ピーク時の要件を満たすと同時に、需要が減少すると自動的にスケールバックできます。

クラスターをプロビジョニングするときに最適なインスタンスサイズを見積もる

最適なインスタンスサイズを見積もるには、ワークロードの実行中の Neptune での平均クエリレイテンシー、および処理中の同時クエリ数を知る必要があります。インスタンスサイズの概算見積もりは、平均クエリレイテンシーに同時実行クエリ数を掛けたものとして計算できます。これにより、ワークロードを処理するのに必要な同時スレッドの平均数がわかります。

Neptune インスタンスの各 vCPU は 2 つの同時クエリスレッドをサポートできるため、スレッドを 2 で割ると、必要な vCPU の数が算出され、Neptune の料金ページの適切なインスタンスサイズと関連付けることができます。例:

```
Average Query Latency:          30ms (0.03s)
Number of concurrent queries:   1000/second

Number of threads needed:       0.03 x 1000 = 30 threads
Number of vCPUs needed:         30 / 2 = 15 vCPUs
```

これをインスタンス内のvCPU の数と相関させると、`r5.4xlarge` がこのワークロードで試すべき推奨インスタンスであるという概算見積もりが得られます。この見積もりは概算であり、インスタン

スサイズの選択に関する最初のガイダンスを提供することのみを目的としています。どのようなアプリケーションでも、適切なサイジングを行って、ワークロードに適した適切なインスタンス数とタイプを決定する必要があります。

処理要件だけでなく、メモリ要件も考慮する必要があります。Neptune は、クエリによってアクセスされるデータがメインメモリのバッファプールキャッシュにある場合に最もパフォーマンスが向上します。十分なメモリをプロビジョニングすれば、I/O コストも大幅に削減できます。

Neptune クラスター内のインスタンスのサイズ設定に関するその他の詳細とガイダンスは、「Neptune DB クラスターでの DB インスタンスのサイジング」ページにあります。

Neo4j から Neptune へのデータ移行

Neo4j から Amazon Neptune への移行を実行する場合、データの移行はプロセスの主要なステップです。データを移行するには複数の方法があります。適切な方法は、アプリケーションのニーズ、データサイズ、必要な移行の種類によって決まります。ただし、これらの移行の多くでは、すべて同じ考慮事項を評価する必要があります。そのいくつかを以下に述べます。

> **Note**
>
> オフラインデータ移行の実行例の順を追った詳しい説明については、AWS データベースブログの「完全自動化ユーティリティによる Neo4j グラフデータベースから Amazon Neptune への移行」を参照してください。

Neo4j から Neptune へのデータ移行の評価

データ移行を評価する際の最初のステップは、データの移行方法を決定することです。オプションは、移行するアプリケーションのアーキテクチャ、データサイズ、移行中の可用性ニーズによって異なります。一般に、移行はオンラインとオフラインの 2 つのカテゴリのいずれかに分類されます。

オフライン移行は、移行中にアプリケーションが読み取りまたは書き込みトラフィックを受け付けないため、簡単に実行できる傾向があります。アプリケーションがトラフィックを受け付けなくなったら、データのエクスポート、最適化、インポートを行うことができ、アプリケーションを再度有効にする前にアプリケーションをテストできます。

オンライン移行では、データの移行中もアプリケーションが読み取り/書き込みトラフィックを受け入れる必要があるため、より複雑です。各オンライン移行の正確なニーズはそれぞれ異なる場合がありますが、一般的なアーキテクチャは以下のようになります。

- Neo4j ストリームを Kafka クラスターのソースとして設定することにより、データベースニーズに対する継続的な変更のフィードを Neo4j で有効にする必要があります。
- これが完了すると、Neptune への移行時に Neo4j からデータをエクスポートする に記載されている指示に従って、実行中のシステムをエクスポートできます。この時間は後で Kafka のトピックと関連付けられます。
- エクスポートされたデータは、Neptune への移行時に Neo4j からデータをインポートする の説明に従って Neptune にインポートされます。
- Kafka ストリームから変更されたデータは、「Amazon Kinesis Data Streams から Amazon Neptune への書き込み」で説明されているものと同様のアーキテクチャを使用して、Neptune ク

ラスターにコピーできます。変更の複製を並行して実行し、新しいアプリケーションアーキテクチャとパフォーマンスを検証できることに注意してください。

- データ移行が検証されたら、アプリケーショントラフィックを Neptune クラスターにリダイレクトし、Neo4j インスタンスを廃止できます。

Neo4j から Neptune への移行のためのデータモデルの最適化

Neptune と Neo4j はどちらもラベル付きプロパティグラフ (LPG) をサポートしています。ただし、Neptune には、パフォーマンスを最適化するために活用できるアーキテクチャとデータモデルの違いがいくつかあります。

ノード ID とエッジ ID の最適化

Neo4j は数字の長い ID を自動的に生成します。Cypher を使用して ID でノードを参照できますが、インデックス付きのプロパティでノードを検索するほうが好ましいので、一般的にはお勧めできません。

Neptune では、頂点とエッジに独自の文字列ベースの ID を指定できます。独自の ID を指定しなかった場合、Neptune は新しいエッジと頂点の UUID の文字列表現を自動的に生成します。

Neo4j からエクスポートしてから Neptune に一括インポートすることによって Neo4j から Neptune にデータを移行すると、Neo4j の ID を保持できます。Neo4j によって生成された数値は、Neptune にインポートする際にユーザー指定の ID として機能し、数値ではなく文字列として表されます。

ただし、頂点プロパティを頂点 ID に昇格させたい場合もあります。インデックス付きプロパティを使用してノードを検索するのが Neo4j でノードを見つける最も速い方法であるように、ID で頂点を検索するのが Neptune で頂点を見つける最も速い方法です。したがって、一意の値を含む適切な頂点プロパティを特定できる場合は、一括読み込み CSV ファイルの vertex~id を指定されたプロパティ値に置き換えることを検討してください。その場合は、CSV ファイル内の対応する ~from および ~to エッジ値もすべて書き換える必要があります。

Neo4j から Neptune にデータを移行する際のスキーマの制約

Neptune 内で利用できる唯一のスキーマ制約は、ノードまたはエッジの ID の一意性です。一意性制約を利用する必要があるアプリケーションでは、ノードまたはエッジ ID を指定して一意性制約を実現するこのアプローチを検討することをお勧めします。アプリケーションが一意性制約として複数の列を使用している場合は、ID をこれらの値の組み合わせに設定できます。例えば、`id=123, code='SEA'` を `ID='123_SEA'` として表して、複雑な一意性制約を実現できます。

Neo4j から Neptune へのデータ移行時のエッジ方向の最適化

ノード、エッジ、またはプロパティが Neptune に追加されると、3 つの異なる方法で自動的にインデックスが付けられ、オプションで 4 番目のインデックスが作成されます。Neptune がインデックスを構築して使用する方法により、出力エッジに従うクエリの方が、入力エッジを使用するクエリよりも効率的です。Neptune のグラフデータストレージモデルでは、これらは SPOG インデックスを使用するサブジェクトベースの検索です。

データモデルとクエリを Neptune に移行する際、最も重要なクエリが、ファンアウトの度合いが高い入力エッジのトラバースに依存していることがわかった場合は、特にトラバースするエッジラベルを指定できないときには、これらのトラバーサルが代わりに出力エッジに従うようにモデルを変更することを検討してください。そのためには、関連するエッジの方向を逆にして、この方向変更のセマンティクスを反映するようにエッジラベルを更新します。例えば、次のように変更します。

```
person_A – parent_of – person_B
   to:
person_B – child_of – person_A
```

この変更を一括読み込みエッジ CSV ファイルで行うには、~from と ~to の列見出しを入れ替えて、~label 列の値を更新するだけです。

エッジ方向を逆にする代わりに、4 番目の Neptune インデックスである OSGP インデックスを有効にできます。これにより、入力エッジのトラバースやオブジェクトベースの検索がより効率的になります。ただし、この 4 番目のインデックスを有効にすると、挿入速度が低下し、より多くのストレージが必要になります。

Neo4j から Neptune へのデータ移行時のフィルタリングの最適化

Neptune は、プロパティが利用可能な最も選択的なプロパティにフィルタリングされたときに最もよく機能するように最適化されています。複数のフィルターを使用すると、それぞれに一致する項目のセットが見つかり、一致するすべてのセットのオーバーラップが計算されます。可能であれば、複数のプロパティを 1 つのプロパティにまとめることで、インデックス検索の回数を最小限に抑え、クエリのレイテンシーを短縮できます。

例えば、次のクエリでは 2 つのインデックス検索と 1 つの結合を使用します。

```
MATCH (n) WHERE n.first_name='John' AND n.last_name='Doe' RETURN n
```

このクエリは 1 回のインデックス検索で同じ情報を取得します。

```
MATCH (n) WHERE n.name='John Doe' RETURN n
```

Neptune は Neo4j とは異なるデータ型をサポートしています。

NNeptune がサポートするデータ型への Neo4j データ型マッピング

- 論理: `Boolean`

  これを Neptune で `Bool` または `Boolean` にマッピングします。
- 数値: `Number`

  これを Neptune で次の Neptune openCypher 型のうち、問題の数値プロパティのすべての値をサポートできる最も狭いものにマッピングします。

  ```
  Byte
  Short
  Integer
  Long
  Float
  Double
  ```
- テキスト: `String`

  これを Neptune で `String` にマッピングします。
- ポイントインタイム:

  ```
  Date
  Time
  LocalTime
  DateTime
  LocalDateTime
  ```

  Neptune がサポートする次のいずれかの ISO-8601 形式を使用して、これらを Neptune で UTC として `Date` にマッピングします。

  ```
  yyyy-MM-dd
  yyyy-MM-ddTHH:mm
  yyyy-MM-ddTHH:mm:ss
  yyyy-MM-ddTHH:mm:ssZ
  ```

- 期間: `Duration`

  必要に応じて、これを Neptune で数値にマッピングして日付計算を行います。

- 空間: `Point`

  これを Neptune でコンポーネントの数値にマッピングします。各数値は個別のプロパティになるか、クライアントアプリケーションによって解釈される文字列値として表現されます。OpenSearch を使用した Neptune のフルテキスト検索統合により、位置情報プロパティをインデックス化できることに注意してください。

Neo4j から Neptune への多値プロパティの移行

Neo4j では、単純型の同種リストをノードとエッジの両方のプロパティとして保存できます。これらのリストには重複する値が含まれる場合があります。

ただし、Neptune では、頂点プロパティとしては設定された、または単一のカーディナリティのみが許され、プロパティグラフデータのエッジプロパティとしては単一カーディナリティのみが許されます。その結果、重複する値を含む Neo4j ノードリストプロパティをNeptune 頂点プロパティに直接移行したり、Neo4j リレーションシップリストプロパティを Neptune エッジプロパティに直接移行したりすることはできません。

重複する値を持つ Neo4j 多値ノードプロパティを Neptune に移行するためのいくつかの可能な戦略は次のとおりです。

- 重複する値を破棄し、多値の Neo4j ノードプロパティを設定されたカーディナリティの Neptune 頂点プロパティに変換します。その場合、Neptune セットは元の Neo4j 多値プロパティの項目の順序を反映しない場合があることに注意してください。
- 多値 Neo4j ノードプロパティを、Neptune 頂点文字列プロパティ内の JSON 形式リストの文字列表現に変換します。
- 多値プロパティ値のそれぞれを値プロパティを持つ個別の頂点に抽出し、プロパティ名のラベルが付いたエッジを使用して、それらの頂点を親頂点に接続します。

同様に、Neo4j 多値リレーションシッププロパティを Neptune に移行するためのいくつかの可能な戦略は次のとおりです。

- 多値 Neo4j リレーションシッププロパティを、JSON 形式リストの文字列表現に変換して、Neptune エッジ文字列プロパティとして保存します。

- Neo4j のリレーションシップを、中間頂点に接続された Neptune 入力エッジと出力エッジにリファクタリングします。多値リレーションシッププロパティ値のそれぞれを値プロパティを持つ個別の頂点に抽出し、プロパティ名のラベルが付いたエッジを使用して、それらの頂点をこの中間頂点に接続します。

openCypher には文字列値の中を簡単に検索できる `CONTAINS` 述語が含まれていますが、JSON 形式のリストの文字列表現は openCypher クエリ言語では不透明であることに注意してください。

Neptune への移行時に Neo4j からデータをエクスポートする

Neo4j からデータをエクスポートするときには、APOC プロシージャを使用して CSV または GraphML にエクスポートしてください。他の形式にエクスポートすることも可能ですが、Neo4j からエクスポートされた CSV データを Neptune のバルクロード形式に変換するためのオープンソースツールや、Neo4j からエクスポートされた GraphML データを Neptune バルクロード形式に変換するためのオープンソースツールがあります。

さまざまな APOC プロシージャを使用して、データを Amazon S3 に直接エクスポートすることもできます。Amazon S3 バケットへのエクスポートはデフォルトでは無効になっていますが、Neo4j APOC ドキュメントの「Amazon S3 へのエクスポート」で説明されている手順を使用して有効にできます。

Neptune への移行時に Neo4j からデータをインポートする

Neptune にデータをインポートするには、Neptune バルクローダーを使用するか、openCypher など, サポートされているクエリ言語のアプリケーションロジックを使用します。

Neptune バルクローダーは、大量のデータをインポートする場合に推奨される方法であり、ベストプラクティスに従えば、インポートパフォーマンスが最適化されます。バルクローダーは 2 種類の CSV 形式をサポートしており、Neo4j からエクスポートされたデータは、データのエクスポート セクションで説明したオープンソースユーティリティを使用して変換できます。

openCypher を使用して、解析、変換、インポート用のカスタムロジックでデータをインポートすることもできます。openCypher クエリは HTTPS エンドポイント (推奨) またはボルトドライバーを使用して送信できます。

Neo4j から Neptune へのアプリケーションの移行

Neo4j から Neptune にデータを移行したら、次のステップはアプリケーション自体を移行することです。データと同様に、アプリケーションを移行する方法は、使用するツール、要件、アーキテクチャの違いなどに基づいて複数あります。このプロセスで通常考慮する必要がある事項を以下に概説します。

Neo4j から Neptune に移行する際の接続の移行

現在 Bolt ドライバーを使用していない場合、または代替ドライバーを使用したい場合は、返されたデータへのフルアクセスを提供する HTTPS エンドポイントに接続できます。

Bolt プロトコルを使用するアプリケーションがある場合は、これらの接続を Neptune に移行し、Neo4j で行ったのと同じドライバーを使用してアプリケーションを接続することができます。Neptune に接続するには、アプリケーションに次の 1 つ以上の変更を加える必要がある場合があります。

- URL とポートは、クラスターエンドポイントとクラスターポート (デフォルトは 8182) を使用するように更新する必要があります。
- Neptune では、すべての接続で SSL を使用する必要があるため、接続ごとに暗号化を指定する必要があります。
- Neptune は IAM ポリシーとロールの割り当てを通じて認証を管理します。IAM ポリシーとロールによって、アプリケーション内のユーザー管理を非常に柔軟に行うことができるため、クラスターを設定する前に「IAM の概要」の情報を読んで理解することが重要です。
- Neptune での Bolt 接続動作 で説明されているように、Neptune と Neo4j では Bolt 接続の動作がいくつかの点で異なります。
- 詳細な情報は、「OpenCypher と Bolt を使用した Neptune のベストプラクティス」にあります。

Java、Python、.NET、NodeJS などの一般的に使用される言語のコードサンプルと、IAM 認証の使用などの接続シナリオのコードサンプルについては、「Bolt プロトコルを使用して openCypher クエリを Neptune に作成する」を参照してください。

Neo4j から Neptune に移行する際のクラスターインスタンスへのクエリのルーティング

Neo4j クライアントアプリケーションはルーティングドライバーを使用し、アクセスモードを指定して読み取り/書き込みリクエストを因果クラスター内の適切なサーバーにルーティングします。

クライアントアプリケーションを Neptune に移行するときには、Neptune エンドポイントを使用して、クエリをクラスター内の適切なインスタンスに効率的にルーティングします。

- Neptune へのすべての接続は、URL で `bolt+routing://` または `neo4j://` ではなく `bolt://` を使用する必要があります。
- クラスターエンドポイントはクラスターの現在のプライマリインスタンスに接続します。クラスターエンドポイントを使用して、書き込みリクエストをプライマリにルーティングします。
- リーダーエンドポイントは、クラスター内のリードレプリカインスタンス間で接続を分散します。リードレプリカインスタンスのない単一インスタンスのクラスターを使用している場合、リーダーエンドポイントは書き込み操作をサポートするプライマリインスタンスに接続します。クラスターに 1 つ以上のリードレプリカインスタンスが含まれている場合、リーダーエンドポイントに書き込みリクエストを送信すると、例外が生成されます。
- クラスター内の各インスタンスは、独自のインスタンスエンドポイントを持つこともできます。クライアントアプリケーションがクラスター内の特定のインスタンスにリクエストを送信する必要がある場合は、インスタンスエンドポイントを使用してください。

詳細については、「Neptune エンドポイントに関する考慮事項」を参照してください。

Neptune でのデータ整合性

Neo4j 因果クラスターを使用する場合、リードレプリカは最終的にコアサーバーと一致しますが、クライアントアプリケーションは因果連鎖を使用することで因果の一貫性を確保できます。因果連鎖では、トランザクション間でブックマークを渡す必要があるため、クライアントアプリケーションはコアサーバーに書き込みを行い、リードレプリカから独自の書き込みを読み取ることができます。

Neptune では、リードレプリカインスタンスは最終的にライターと一致し、レプリカの遅延は通常 100 ミリ秒未満です。ただし、変更が複製されるまでは、既存のエッジと頂点の更新、および新しいエッジと頂点の追加は、レプリカインスタンスでは確認できません。そのため、アプリケーションが Neptune 上で各書き込みを読み取ることで即時整合性を保つ必要がある場合は、書き込み後の読み取り操作にクラスターエンドポイントを使用してください。クラスターエンドポイントを読み取り操作で使用するのは、このときだけです。それ以外の場合は、読み取りにはリーダーエンドポイントを使用してください。

Neo4j から Neptune へのクエリの移行

Neptune が openCypher をサポートしているため、Neo4j からクエリを移行するのに必要な作業量は大幅に減りますが、移行の際にはまだ評価すべき相違点がいくつかあります。

- データモデルの最適化 で説明したように、Neptune 用に最適化されたグラフデータモデルを作成するために、データモデルに変更を加える必要がある場合があります。そのためには、クエリとテストを変更する必要があります。
- Neo4j は、Neptune によって実装された openCypher 仕様には含まれていない, さまざまな Cypher 固有の言語拡張を提供しています。ユースケースと使用する機能によっては、openCypher 言語内、Gremlin 言語の使用、または Neptune 上の openCypherで実行するように Cypher クエリを書き直す で説明されているその他のメカニズムによる回避策がある場合があります。
- 多くの場合、アプリケーションは Bolt ドライバーそのものではなく、他のミドルウェアコンポーネントを使用してデータベースとやり取りします。Neptune の Neo4j との互換性 を参照して、使用しているツールやミドルウェアがサポートされているかどうかを確認してください。
- フェイルオーバーが発生した場合、接続に提供されたクラスターエンドポイントが IP アドレスに解決されたために、Bolt ドライバーは以前のライターまたはリーダーインスタンスに接続し続ける可能性があります。フェイルオーバー後の新しい接続の作成 で説明されているように、アプリケーションでの適切なエラー処理によって、これに対処する必要があります。
- 解決できない競合またはロック待機タイムアウトのためにトランザクションがキャンセルされると、Neptune は `ConcurrentModificationException` で応答します。詳細については、「エンジンエラーコード」を参照してください。ベストプラクティスとして、クライアントは常にこれらの例外をキャッチして処理する必要があります。

  `ConcurrentModificationException` は、複数のスレッドや複数のアプリケーションが同時にシステムに書き込みを行っているときに発生することがあります。トランザクションの分離レベルにより、このような競合が避けられない場合があります。
- Neptune は、同じデータに対する Gremlin クエリと openCypher クエリの両方の実行をサポートしています。つまり、シナリオによっては、より強力なクエリ機能を備えた Gremlin を使用してクエリの機能の一部を実行することを検討する必要があるかもしれません。

インフラストラクチャーのプロビジョニングをします。 で説明したように、インスタンス数、インスタンスサイズ、クラスタートポロジがすべてアプリケーションの特定のワークロードに合わせて最適化されるように、アプリケーションごとに適切なサイジングを行う必要があります。

ここで説明したアプリケーションの移行に関する考慮事項は最も一般的なものですが、すべてを網羅しているわけではありません。アプリケーションはそれぞれ一意であるためです。さらに質問がある場合は、AWS サポートに連絡するか、アカウントチームに連絡してください。

Neo4j に固有の機能やツールの移行

Neo4j には、アプリケーションが依存する可能性のある機能を備えたさまざまなカスタム機能とアドオンがあります。この機能を移行する必要性を評価する場合、同じ目標を達成するためのより優れたアプローチが AWS 内にあるかどうかを調べると役立つことがよくあります。Neo4j と Neptune のアーキテクチャの違いを考慮すると、他の AWS サービスや統合を活用する効果的な代替案が見つかることがよくあります。

Neo4j 固有の機能と推奨回避策のリストについては、「Neptune の Neo4j との互換性」を参照してください。

Neptune の Neo4j との互換性

Neo4j は、データの読み込み、データ ETL、アプリケーションクエリ、データストレージ、管理操作のすべてが EC2 インスタンスなど、同じコンピューティングリソース内で行われるオールインワンアーキテクチャアプローチに基づいています。Amazon Neptune は、OLTP に焦点を当てたオープン仕様のグラフデータベースであり、アーキテクチャによって操作が分離され、リソースが分離されるため、動的にスケーリングできます。

Neo4j には、openCypher 仕様に含まれていない、openCypher と互換性のない、または Neptune の openCypher の実装と互換性のないサードパーティのツールなど、さまざまな機能やツールがあります。次のリストには、これらの中で最も一般的なもののいくつかが含まれています。

Neptune にはない Neo4J 固有の機能

- **`LOAD CSV`** — Neptune は、Neo4j とは異なるアーキテクチャアプローチでデータをロードします。スケーリングとコストの最適化を図るため、Neptune ではリソースに関する懸念を分離し、必要な ETL プロセスを実行して Neptune バルクローダーがサポートする形式でデータを準備するなど、AWS Glue などの AWS サービス統合の 1 つを使用することを推奨しています。

  もう 1 つのオプションは、Amazon EC2 インスタンス、Lambda 関数、Amazon Elastic Container Service、AWS Batch ジョブなどの AWS コンピューティングリソース上で実行されるアプリケーションコードを使用して同じことを行うことです。コードでは、Neptune の HTTPS エンドポイントか Bolt エンドポイントのどちらかを使用できます。
- きめ細かなアクセスコントロール — Neptune は IAM 条件キーを使用して、データアクセスアクションに対するきめ細かなアクセスコントロールをサポートします。アプリケーション層では、さらにきめ細かいアクセスコントロールを実装できます。
- Neo4j Fabric — Neptune は、SPARQL `SERVICE` キーワードを使用した RDF ワークロードのデータベース間のクエリフェデレーションをサポートしています。現在、プロパティグラフワークロードのクエリフェデレーションに関するオープンスタンダードや仕様はないため、この機能はアプリケーション層で実装する必要があります。
- ロールベースのアクセスコントロール (RBAC) - Neptune は IAM ポリシーとロールの割り当てを通じて認証を管理します。IAM ポリシーとロールによって、アプリケーション内のユーザー管理を非常に柔軟に行うことができるため、クラスターを設定する前に「IAM の概要」の情報を読んで理解することが重要です。

- ブックマーク — Neptune クラスターは 1 つのライターインスタンスと最大 15 のリードレプリカインスタンスで構成されます。ライターインスタンスに書き込まれるデータは ACID に準拠してお

り、それ以降の読み取りでは強い一貫性が保証されます。リードレプリカはライターインスタンスと同じストレージボリュームを使用し、通常はデータが書き込まれてから 100 ミリ秒未満で最終的に一貫性が保たれます。新しい書き込みの読み取りの一貫性をすぐに保証する必要があるユースケースでは、これらの読み取りをリーダーエンドポイントではなくクラスターエンドポイントに送る必要があります。

- APOC プロシージャ — APOC プロシージャは openCypher 仕様に含まれていないため、Neptune は外部プロシージャを直接サポートしていません。代わりに、Neptune は他の AWS サービスとの統合を利用して、同様のエンドユーザー機能をスケーラブルで安全、かつ堅牢な方法で実現しています。APOC プロシージャを openCypher や Gremlin で書き直すことができる場合もありますが、Neptune アプリケーションには関係のないものもあります。

  一般に、APOC プロシージャは以下のカテゴリに分類されます。

  - インポート — Neptune は、クエリ言語、Neptune バルクローダー、または AWS Database Migration Service のターゲットとして、さまざまな形式でのデータのインポートをサポートしています。データに対する ETL 操作は、AWS Glue および `neptune-python-utils` オープンソースパッケージを使用して実行できます。
  - エクスポート — Neptune は、さまざまな一般的なエクスポート形式と方法をサポートする `neptune-export` ユーティリティを使用してデータのエクスポートをサポートしています。
  - データベース統合 — Neptune は、AWS Glue などの ETL ツールや AWS Database Migration Service などの移行ツールを使用して、他のデータベースとの統合をサポートしています。
  - グラフの更新 — Neptune は openCypher と Gremlin の両方のクエリ言語のサポートを通じて、プロパティグラフデータを更新するための豊富な機能をサポートしています。よく使われるプロシージャの書き換えの例については、「Cypher の書き換え」を参照してください。
  - データ構造 — Neptune は openCypher と Gremlin の両方のクエリ言語のサポートを通じて、プロパティグラフデータを更新するための豊富な機能をサポートしています。よく使われるプロシージャの書き換えの例については、「Cypher の書き換え」を参照してください。
  - 時間的 (日付時刻) — Neptune は openCypher と Gremlin の両方のクエリ言語のサポートを通じて、プロパティグラフデータを更新するための豊富な機能をサポートしています。よく使われるプロシージャの書き換えの例については、「Cypher の書き換え」を参照してください。
  - データ構造 — Neptune は openCypher と Gremlin の両方のクエリ言語のサポートを通じて、プロパティグラフデータを更新するための豊富な機能をサポートしています。よく使われるプロシージャの書き換えの例については、「Cypher の書き換え」を参照してください。
  - 高度なグラフクエリ — Neptune は openCypher と Gremlin の両方のクエリ言語のサポートを通じて、プロパティグラフデータを更新するための豊富な機能をサポートしています。よく使われるプロシージャの書き換えの例については、「Cypher の書き換え」を参照してください。

  - グラフの比較 — Neptune は openCypher と Gremlin の両方のクエリ言語のサポートを通じて、プロパティグラフデータを更新するための豊富な機能をサポートしています。よく使われるプロシージャの書き換えの例については、「Cypher の書き換え」を参照してください。
  - Cypher の実行 — Neptune は openCypher と Gremlin の両方のクエリ言語のサポートを通じて、プロパティグラフデータを更新するための豊富な機能をサポートしています。よく使われるプロシージャの書き換えの例については、「Cypher の書き換え」を参照してください。
- カスタムプロシージャ — Neptune はユーザーによって作成されたカスタムプロシージャをサポートしていません。この機能はアプリケーション層で実装する必要があります。
- 地理空間 — Neptune は地理空間機能のネイティブサポートを提供していませんが、他の AWS サービスと統合することで同様の機能を実現できます。Ross Gabay と Abhilash Vinod によるブログ記事「Amazon Neptune と Amazon OpenSearch Service を組み合わせて、地理空間クエリを行う」(2022 年 2 月 1 日) を参照してください。
- グラフデータサイエンス — Neptune は現在、グラフ分析アルゴリズムのライブラリをサポートするメモリ最適化エンジンである Neptune Analytics を通じて、グラフ分析をサポートしています。

  Neptune は、AWS Pandas SDK との統合と、この統合を Python 環境内で活用してグラフデータに対して分析を実行する方法を示すサンプルノートブックをいくつか提供しています。

- スキーマ制約 - Neptune 内で利用できる唯一のスキーマ制約は、ノードまたはエッジの ID の一意性です。グラフ内の要素に追加のスキーマ制約や、追加の一意性や値の制約を指定する機能はありません。Neptune の ID 値は文字列であり、次のように Gremlin を使用して設定できます。

  ```
  g.addV('person').property(id, '1') )
  ```

  一意性制約として ID を利用する必要があるアプリケーションでは、一意性制約を達成するために、このアプローチを試すことをお勧めします。アプリケーションが一意性制約として複数の列を使用している場合は、ID をこれらの値の組み合わせに設定できます。例えば、`id=123, code='SEA'` を `ID='123_SEA'` として表して、複雑な一意性制約を実現できます。
- マルチテナンシー — Neptune はクラスターごとに 1 つのグラフのみをサポートします。Neptune を使用してマルチテナントシステムを構築するには、複数のクラスターを使用するか、単一のグラフ内でテナントを論理的に分割し、アプリケーション側のロジックを使用して分離を強制します。例えば、次のように、プロパティ `tenantId` を追加して、各クエリに含めます。

  ```
  MATCH p=(n {tenantId:1})-[]->({tenantId:1}) RETURN p LIMIT 5)
  ```

Neptune サーバーレスでは、複数の DB クラスターを使用してマルチテナンシーを比較的簡単に実装できます。各クラスターは、必要に応じて独立して自動的にスケーリングされます。

Neo4j ツールの Neptune サポート

Neptune は、Neo4j ツールに代わる以下の代替手段を提供しています。

- Neo4j ブラウザ — Neptune は、クエリを実行して結果を視覚化するための開発者向け IDE を提供するオープンソースのグラフノートブックを提供しています。
- Neo4j Bloom — Neptune は、グラフエクスプローラー、Tom Sawyer、Cambridge Intelligence、Graphistry、メタファクト、G.V() などのサードパーティ製視覚化ソリューションを使用した豊富なグラフ視覚化をサポートしています。
- GraphQL — Neptune は、現在、カスタム AWS AppSync 統合を通じて GraphQL をサポートしています。「Amazon Neptune と AWS Amplify によるグラフアプリケーションの構築」ブログ記事、およびサンプルプロジェクト「AWS AppSync と Amazon Neptune によるサーバーレスカロリートラッカーアプリケーションの構築」を参照してください。
- NeoSemantics — Neptune は RDF データモデルをネイティブにサポートしているため、RDF ワークロードを実行したいお客様は、Neptune の RDF モデルサポートを使用することをお勧めします。
- Arrows.app — エクスポートコマンドを使用してモデルをエクスポートするときに作成される Cypher は、Neptune と互換性があります。
- Linkurious Ogma — Linkurious Ogma とのサンプル統合はこちらから入手できます。
- Spring Data Neo4j — これは現在、Neptune と互換性がありません。
- Neo4j スパークコネクタ — Neo4j スパークコネクタをスパークジョブ内で使用して、openCypher を使用して Neptune に接続できます。以下はサンプルコードとアプリケーション設定です。

  サンプルコード:

```
SparkSession spark = SparkSession
            .builder()
            .config("encryption.enabled", "true")
            .appName("Simple Application").config("spark.master",
 "local").getOrCreate();

Dataset<Row> df = spark.read().format("org.neo4j.spark.DataSource")
            .option("url", "bolt://(your cluster endpoint):8182")
```

```
            .option("encryption.enabled", "true")
            .option("query", "MATCH (n:airport) RETURN n")
            .load();

System.out.println("TOTAL RECORD COUNT: " + df.count());
spark.stop();
```

アプリケーションの設定:

```
<dependency>
    <groupId>org.neo4j</groupId>
    <artifactId>neo4j-connector-apache-spark_2.12-4.1.0</artifactId>
    <version>4.0.1_for_spark_3</version>
</dependency>
```

ここに記載されていない Neo4j の機能とツール

ここに記載されていないツールや機能を使用している場合、Neptune や AWS 内のその他のサービスとの互換性は不明です。さらに質問がある場合は、AWS サポートに連絡するか、アカウントチームに連絡してください。

Neptune 上の openCypherで実行するように Cypher クエリを書き直す

openCypher 言語は、プロパティグラフ用の宣言型クエリ言語であり、当初は Neo4j によって開発され、その後、2015 年にオープンソース化され、Apache 2 オープンソースライセンスの下で openCypher プロジェクトに貢献しました。AWS では、オープンソースは誰にとっても良いことだと考えており、オープンソースの価値をお客様にもたらし、AWS のオペレーショナルエクセレンスをオープンソースコミュニティにもたらすことに尽力しています。

openCypher 構文は、「Cypher クエリ言語リファレンス、バージョン 9」で説明されています。

openCypher には Cypher クエリ言語の構文と機能の一部が含まれているため、移行シナリオによっては、openCypher 準拠の形式でクエリを書き直したり、必要な機能を実現するための代替方法を検討したりする必要があります。

このセクションには、一般的な相違点を処理するための推奨事項が含まれていますが、すべてを網羅しているわけではありません。これらの書き換えを使用するアプリケーションを徹底的にテストして、期待どおりの結果になることを確認する必要があります。

`None`、`All`、および `Any` 述語関数の書き換え

これらの関数は openCypher 仕様には含まれていません。openCypher でもリスト内包表記を使用して同等の結果を得ることができます。

例えば、ノード `Start` からノード `End` へのすべてのパスを検索しても、クラスプロパティが `D` のノードを通過することはできません。

```
# Neo4J Cypher code
match p=(a:Start)-[:HOP*1..]->(z:End)
where none(node IN nodes(p) where node.class ='D')
return p

# Neptune openCypher code
match p=(a:Start)-[:HOP*1..]->(z:End)
where size([node IN nodes(p) where node.class = 'D']) = 0
return p
```

リスト内包表記を使用すると、次のような結果が得られます。

```
all  => size(list_comprehension(list)) = size(list)
any  => size(list_comprehension(list)) >= 1
```

```
none => size(list_comprehension(list)) = 0
```

openCypher で Cypher **reduce()** 関数を書き換える

reduce() 関数は openCypher 仕様には含まれていません。リスト内の要素からデータの集約を作成するためによく使用されます。多くの場合、リスト内包表記と UNWIND 句を組み合わせて、同様の結果が得られます。

例えば、次の Cypher クエリは、アンカレッジ (ANC) とオースティン (AUS) の間の、1 ~ 3 か所の経由地があるパス上のすべての空港を検索し、各パスの合計距離を返します。

```
MATCH p=(a:airport {code: 'ANC'})-[r:route*1..3]->(z:airport {code: 'AUS'})
RETURN p, reduce(totalDist=0, r in relationships(p) | totalDist + r.dist) AS totalDist
ORDER BY totalDist LIMIT 5
```

openCypher for Neptune でも同じクエリを次のように記述できます。

```
MATCH p=(a:airport {code: 'ANC'})-[r:route*1..3]->(z:airport {code: 'AUS'})
UNWIND [i in relationships(p) | i.dist] AS di
RETURN p, sum(di) AS totalDist
ORDER BY totalDist
LIMIT 5
```

openCypher での「Cypher FOREACH」句の書き換え

FOREACH 句は openCypher 仕様には含まれていません。クエリの途中でデータを更新する場合によく使用され、多くの場合、集計やパス内の要素からのデータを更新します。

パスの例として、アンカレッジ (ANC) とオースティン (AUS) の間の経由地が 2 つ以下のパス上にあるすべての空港を検索し、それぞれに訪問済みのプロパティを設定します。

```
# Neo4J Example
MATCH p=(:airport {code: 'ANC'})-[*1..2]->({code: 'AUS'})
FOREACH (n IN nodes(p) | SET n.visited = true)

# Neptune openCypher
MATCH p=(:airport {code: 'ANC'})-[*1..2]->({code: 'AUS'})
WITH nodes(p) as airports
UNWIND airports as a
SET a.visited=true
```

別の例は、次のとおりです。

```
# Neo4J Example
MATCH p=(start)-[*]->(finish)
WHERE start.name = 'A' AND finish.name = 'D'
FOREACH (n IN nodes(p) | SET n.marked = true)

# Neptune openCypher
MATCH p=(start)-[*]->(finish)
WHERE start.name = 'A' AND finish.name = 'D'
UNWIND nodes(p) AS n
SET n.marked = true
```

Neptune で Neo4j APOC プロシージャを書き換える

以下の例では、最も一般的に使用されている APOC プロシージャの一部を openCypher を使用して置き換えています。これらの例は参照用であり、一般的なシナリオの処理方法に関するいくつかの提案を提供することを目的としています。実際には、アプリケーションはそれぞれ異なるため、必要な機能をすべて提供するための独自の戦略を考え出す必要があります。

`apoc.export` プロシージャの書き換え

Neptune には、neptune-export ユーティリティを使用して、CSV や JSON など, さまざまな出力形式で完全なグラフのエクスポートとクエリベースのエクスポートの両方を行うためのオプションが多数用意されています (「Neptune DB クラスターからデータをエクスポートする」を参照)。

`apoc.schema` プロシージャの書き換え

Neptune には、明示的に定義されたスキーマ、インデックス、または制約はないため、多くの `apoc.schema` プロシージャは不要になりました。例:

- `apoc.schema.assert`
- `apoc.schema.node.constraintExists`
- `apoc.schema.node.indexExists`,
- `apoc.schema.relationship.constraintExists`
- `apoc.schema.relationship.indexExists`
- `apoc.schema.nodes`
- `apoc.schema.relationships`

Neptune openCypher は、以下に示すように、プロシージャと同様の値の取得をサポートしていますが、大きなグラフでは、グラフの大部分をスキャンして回答を返す必要があるため、パフォーマンスの問題が発生する可能性があります。

```
# openCypher replacement for apoc.schema.properties.distinct
MATCH (n:airport)
RETURN DISTINCT n.runways
```

```
# openCypher replacement for apoc.schema.properties.distinctCount
MATCH (n:airport)
RETURN DISTINCT n.runways, count(n.runways)
```

apoc.do プロシージャの代替方法

これらのプロシージャは、他の openCypher 句を使用して簡単に実装できる条件付きクエリを実行するために使用されます。Neptune では、同様の動作を実現する方法が少なくとも 2 つあります。

- 1 つの方法は、openCypher のリスト内包表記機能を UNWIND 句と組み合わせることです。
- もう 1 つの方法は、Gremlin の choose() ステップと coalesce() ステップを使用することです。

これらのアプローチの例を次に示します。

apoc.do.when の代替手段

```
# Neo4J Example
MATCH (n:airport {region: 'US-AK'})
CALL apoc.do.when(
 n.runways>=3,
 'SET n.is_large_airport=true RETURN n',
 'SET n.is_large_airport=false RETURN n',
 {n:n}
) YIELD value
WITH collect(value.n) as airports
RETURN size([a in airports where a.is_large_airport]) as large_airport_count,
size([a in airports where NOT a.is_large_airport]) as small_airport_count


# Neptune openCypher
MATCH (n:airport {region: 'US-AK'})
WITH n.region as region, collect(n) as airports
```

```
WITH [a IN airports where a.runways >= 3] as large_airports,
[a IN airports where a.runways < 3] as small_airports, airports
UNWIND large_airports as la
SET la.is_large_airport=true
WITH DISTINCT small_airports, airports
UNWIND small_airports as la
    SET la.small_airports=true
WITH DISTINCT airports
RETURN size([a in airports where a.is_large_airport]) as large_airport_count,
size([a in airports where NOT a.is_large_airport]) as small_airport_count

#Neptune Gremlin using choose()
g.V().
  has('airport', 'region', 'US-AK').
  choose(
    values('runways').is(lt(3)),
    property(single, 'is_large_airport', false),
    property(single, 'is_large_airport', true)).
  fold().
  project('large_airport_count', 'small_airport_count').
    by(unfold().has('is_large_airport', true).count()).
    by(unfold().has('is_large_airport', false).count())

 #Neptune Gremlin using coalesce()
g.V().
  has('airport', 'region', 'US-AK').
  coalesce(
    where(values('runways').is(lt(3))).
    property(single, 'is_large_airport', false),
    property(single, 'is_large_airport', true)).
  fold().
  project('large_airport_count', 'small_airport_count').
    by(unfold().has('is_large_airport', true).count()).
    by(unfold().has('is_large_airport', false).count())
```

apoc.do.case の代替手段

```
# Neo4J Example
MATCH (n:airport {region: 'US-AK'})
CALL apoc.case([
 n.runways=1, 'RETURN "Has one runway" as b',
 n.runways=2, 'RETURN "Has two runways" as b'
 ],
```

```
 'RETURN "Has more than 2 runways" as b'
) YIELD value
RETURN {type: value.b,airport: n}

# Neptune openCypher
MATCH (n:airport {region: 'US-AK'})
WITH n.region as region, collect(n) as airports
WITH [a IN airports where a.runways =1] as single_runway,
[a IN airports where a.runways =2] as double_runway,
[a IN airports where a.runways >2] as many_runway
UNWIND single_runway as sr
    WITH {type: "Has one runway",airport: sr} as res, double_runway, many_runway
WITH DISTINCT double_runway as double_runway, collect(res) as res, many_runway
UNWIND double_runway as dr
    WITH {type: "Has two runways",airport: dr} as two_runways, res, many_runway
WITH collect(two_runways)+res as res, many_runway
UNWIND many_runway as mr
    WITH {type: "Has more than 2 runways",airport: mr} as res2, res, many_runway
WITH collect(res2)+res as res
UNWIND res as r
RETURN r

#Neptune Gremlin using choose()
g.V().
  has('airport', 'region', 'US-AK').
  project('type', 'airport').
    by(
      choose(values('runways')).
        option(1, constant("Has one runway")).
        option(2, constant("Has two runways")).
        option(none, constant("Has more than 2 runways"))).
    by(elementMap())

 #Neptune Gremlin using coalesce()
 g.V().
  has('airport', 'region', 'US-AK').
  project('type', 'airport').
    by(
      coalesce(
        has('runways', 1).constant("Has one runway"),
        has('runways', 2).constant("Has two runways"),
        constant("Has more than 2 runways"))).
    by(elementMap())
```

リストベースのプロパティの代替

Neptune は現在、リストベースのプロパティの保存をサポートしていません。ただし、リストの値をカンマ区切り文字列として保存し、join() および split() 関数を使用してリストプロパティを構築および分解することにより、同様の結果が得られます。

例えば、タグのリストをプロパティとして保存したい場合、rewrite という例を使用できます。これは、カンマ区切りのプロパティを取得し、split() および join() 関数をリスト内包表記とともに使用して同等の結果を得る方法を示します。

```
# Neo4j Example (In this example, tags is a durable list of string.
MATCH (person:person {name: "TeeMan"})
WITH person, [tag in person.tags WHERE NOT (tag IN ['test1', 'test2', 'test3'])] AS
 newTags
SET person.tags = newTags
RETURN person

# Neptune openCypher
MATCH (person:person {name: "TeeMan"})
WITH person, [tag in split(person.tags, ',') WHERE NOT (tag IN ['test1', 'test2',
 'test3'])] AS newTags
SET person.tags = join(newTags,',')
RETURN person
```

Neo4j から Neptune へ移行するためのリソース

Neptune には、移行プロセスを支援するツールやリソースがいくつか用意されています。

Neo4j から Neptune への移行を容易にするツール

- openCypher CheatSheet。
- neo4j-to-neptune — Neo4j から Neptune にデータを移行するためのコマンドラインユーティリティ。
- fully-automated-neo4j-to-neptune — AWS CDK アプリケーション。シンプルな Neo4j データベースを Amazon Neptune に移行する方法を示します。
- csv-to-neptune-bulk-format — このツールは、設定ベースのアプローチを使用して、1 つ以上の CSV ファイルを、サポートされている Neptune 一括ロード形式に再フォーマットします。

ブログ記事

- Sanjeet Sahay による「Amazon Managed Streaming for Apache Kafka を使用してデータキャプチャを Neo4j から Amazon Neptune に変更する」を参照してください。
- Sanjeet Sahay による「完全自動ユーティリティによって Neo4j グラフデータベースを Amazon Neptune に移行する」を参照してください。

既存のグラフを Apache TinkerPop Gremlin サーバーから Amazon Neptune に移行する

Apache TinkerPop Gremlin サーバーに Amazon Neptune に移行するグラフデータがある場合は、次の手順を実行します。

1. Gremlin サーバーから Amazon Simple Storage Service (Amazon S3) にデータをエクスポートします。
2. エクスポートしたデータを Neptune バルクローダがインポートできる CSV 形式へ変換します。
3. Neptune 一括ローダー を使用して、準備した Neptune DB クラスターにデータをインポートします。
4. 既存のアプリケーションを変更して Neptune の Gremlin エンドポイントに接続し、Neptune Gremlin の実装差異に準拠するために必要な変更を加えます。

既存のグラフを RDF トリプルストアから Amazon Neptune に移行する

Amazon Neptune に移行するグラフデータが RDF/SPARQL にある場合は、次の手順を実行します。

1. RDF トリプルストアからデータをエクスポートします。
2. エクスポートしたデータを Neptune バルクローダがインポートできる形式に変換します。
3. インポートするデータを Amazon Simple Storage Service (Amazon S3) に保存します。
4. Neptune 一括ローダー を使用して、Amazon S3 から準備した Neptune DB クラスターにデータをインポートします。
5. 既存のアプリケーションを変更して Neptune の SPARQL エンドポイントに接続します。

プロパティグラフの CSV データを RDF に移行してみたい場合は、Amazon Neptune CSV - RDFコンバータもご使用いただけます。

AWS Database Migration Service (AWS DMS) を使用してリレーショナルデータベースまたは NoSQL データベースから Amazon Neptune に移行する

AWS Database Migration Service (AWS DMS) は、リレーショナルデータベース、データウェアハウス、NoSQL データベース、他の種類のデータストアを移行しやすくするクラウドサービスです。リレーショナルまたは NoSQL データベース (AWS DMS がサポートする) のいずれかにグラフデータが格納されている場合、AWS DMS は、現在のデータベースからダウンタイムを必要とせずに、迅速かつ安全に Neptune に移行するのに役立ちます。詳細については、「AWS Database Migration Service を使用して別のデータストアから Amazon Neptune にデータをロードする」を参照してください。

AWS DMS を使用する移行データフローは次のとおりです。

- AWS DMS table-mapping オブジェクトの作成。この JSON オブジェクトは、ソースデータベースから読み取る必要があるテーブル、その順序、および列の命名方法を指定します。また、コピーされる行をフィルタリングし、小文字への変換や四捨五入などの単純な値の変換を提供することもできます。
- ソースデータベースから抽出されたデータを Neptune にロードする方法を指定するには、Neptune `GraphMappingConfig` を作成する必要があります。
  - RDF データ (SPARQL を使用してクエリされる) の場合、`GraphMappingConfig` は W3 の標準 R2RML マッピング言語で記述されます。
  - プロパティグラフデータ (Gremlin を使用してクエリされる) の場合、`GraphMappingConfig` は JSON オブジェクトです (GraphMappingConfig Property-Graph/Gremlin データのレイアウトを参照)。
- Neptune DB クラスターと同じ VPC に AWS DMS レプリケーションインスタンスを作成し、移行を実行します。
- 移行するデータをステージングするための中間ストレージとして使用する Amazon S3 バケットを作成します。
- AWS DMS 移行タスクを実行します。

詳細については、AWS Database Migration Service を使用して別のデータストアから Amazon Neptune にデータをロードする を参照してください。また、Chris Smith の 4 ピースブログ記事「AWS Database Migration Service (DMS) を使用して Amazon Neptune のグラフをリレーショナルデータベースから取り込む」もご覧ください。

- パート 1: ステージの設定
- パート 2: プロパティグラフモデルの設計
- パート 3: RDF モデルの設計
- パート 4: すべてまとめる

Blazegraph から Amazon Neptune への移行

オープンソースBlazegraph RDF トリプルストアにグラフがあるなら、次の手順を使用して Amazon Neptune にグラフデータに移行できます。

- AWS インフラストラクチャーのプロビジョニングをします。まず、AWS CloudFormation テンプレートを使用して、必要な Neptune インフラストラクチャをプロビジョニングを開始します (DB クラスターを作成する を参照)。
- Blazegraph からデータをエクスポートします。Blazegraph からデータをエクスポートするには、主に SPARQL CONSTRUCT クエリを使用するか、Blazegraph エクスポートユーティリティを使用する方法があります。
- Neptune にデータをインポートします。その後、Neptune Workbench および Neptune 一括ローダー を使用して、エクスポートされたデータファイルを Neptune に読み込むことができます。

このアプローチは、一般に、他の RDF トリプルストアデータベースからの移行にも適用できます。

Blazegraph と Neptune の互換性

グラフデータを Neptune に移行する前に、Blazegraph と Neptune の間に重要な違いがいくつかあることにご注意ください。これらの違いにより、クエリ、アプリケーションアーキテクチャ、またはその両方の変更が必要になる場合や、移行が実用的でない場合があります。

- **`Full-text search`** — Blazegraph では、Apache Solr との統合により、内部フルテキスト検索または外部のフルテキスト検索機能を使用できます。これらの機能のいずれかを使用する場合は、Neptune がサポートするフルテキスト検索機能の最新の更新情報をご確認ください。「Neptune フルテキスト検索」を参照してください。
- **`Query hints`** — Blazegraph と Neptune は両方とも、クエリヒントの概念を使用して SPARQL を拡張します。移行中は、使用するクエリヒントを移行する必要があります。Neptune がサポートする最新のクエリヒントについては、SPARQL クエリヒント を参照してください。
- 推論 — Blazegraph は、トリプルモードでは設定可能なオプションとして推論をサポートしますが、クワッドモードではサポートしません。Neptune は推論をまだサポートしていません。
- 地理空間検索 — Blazegraph は、地理空間サポートを可能にする名前空間の構成をサポートします。Neptune ではこの機能はまだ使用できません。
- マルチテナンシー — Blazegraph は、単一のデータベース内でマルチテナントをサポートします。Neptune では、マルチテナントは、名前付きグラフにデータを格納し、SPARQL クエリに

USING NAMED 句を使用するか、テナントごとに個別のデータベースクラスターを作成することによってサポートされます。

- フェデレーション — Neptune は現在、プライベート VPC 内、VPC 間、または外部のインターネットエンドポイントなど、Neptune インスタンスからアクセス可能な場所への SPARQL 1.1 フェデレーションをサポートしています。特定のセットアップと必要なフェデレーションエンドポイントによっては、追加のネットワーク構成が必要になる場合があります。
- Blazegraph の標準拡張 — Blazegraph には SPARQL と REST API 標準の両方に対する複数の拡張機能が含まれていますが、Neptune は標準仕様そのものとのみ互換性があります。これにより、アプリケーションの変更が必要になる場合や、移行が困難になる場合があります。

Neptune の AWS インフラストラクチャのプロビジョニング

必要な AWS インフラストラクチャを AWS Management Console または AWS CLI から手動で構築することはできますが、以下で説明するように、代わりに CloudFormation テンプレートを使用する方が便利です。

CloudFormation テンプレートを使用した Neptune のプロビジョニング。

1. AWS CloudFormation スタックを使用して Neptune DB クラスターを作成する に移動します。
2. 任意のリージョンで [Launch Stack] (スタックを起動する) を選択します。
3. 必要なパラメータ (スタック名と`EC2SSHKeyPairName`) を設定します。また、移行プロセスを容易にするために、次のオプションパラメータを設定します。

   - `AttachBulkloadIAMRoleToNeptuneCluster` を true に設定します。このパラメータを使用すると、データを一括ロードできるように、適切な IAM ロールを作成してクラスターにアタッチできます。
   - `NotebookInstanceType` を任意のインスタンスタイプに設定します。このパラメーターは、Neptune への一括ロードを実行し、移行を検証するために使用する Neptune ワークブックを作成します。
4. [Next] (次へ) をクリックします。
5. その他のスタックオプションを設定します。
6. [Next] (次へ) をクリックします。
7. オプションを確認し、両方のチェックボックスをオンにして、AWSCloudFormation には追加の機能が必要になる場合があることを承認します。
8. [スタックの作成] を選択します。

プロセスには数分かかることがあります。

Blazegraph からのデータのエクスポート

次のステップでは、Blazegraph からデータを Neptune バルクローダと互換性がある形式でエクスポートします。

Blazegraph でのデータの保存方法 (トリプルまたはクワッド) および使用されている名前付きグラフの数に応じて、Blazegraph ではエクスポート処理を複数回実行し、複数のデータファイルを生成する必要があります。

- データがトリプルとして格納されている場合は、名前付きグラフごとに 1 つのエクスポートを実行する必要があります。
- データがクアッドとして格納されている場合は、N-Quads 形式でデータをエクスポートするか、名前付きグラフをトリプル形式でエクスポートするかを選択できます。

以下では、単一のネームスペースを N-Quads としてエクスポートすることを前提としていますが、追加のネームスペースまたは目的のエクスポート形式に対してこのプロセスを繰り返すことができます。

Blazegraph をオンラインにして、移行中に使用できるようにする必要がある場合は、SPARQL CONSTRUCT クエリを使用します。これには、アクセス可能な SPARQL エンドポイントを使用して Blazegraph インスタンスをインストール、設定、および実行する必要があります。

Blazegraph をオンラインにする必要がない場合は、BlazeGraph 書き出しユーティリティを使用します。これを行うには Blazegraph をダウンロードする必要があり、データファイルと構成ファイルはアクセス可能である必要がありますが、サーバーが実行されている必要はありません。

SPARQL コンストラクトを使用した Blazegraph からのデータのエクスポート

SPARQL コンストラクトは、指定されたクエリテンプレートに一致する RDF グラフを返す SPARQL の機能です。このユースケースでは、次のようなクエリを使用して、一度に 1 つの名前空間データをエクスポートします。

```
CONSTRUCT WHERE { hint:Query hint:analytic "true" . hint:Query
 hint:constructDistinctSPO "false" . ?s ?p ?o }
```

このデータをエクスポートする他の RDF ツールが存在しますが、このクエリを実行する最も簡単な方法は、Blazegraph で提供される REST API エンドポイントを使用することです。次のスクリプト

は、Python (3.6+) スクリプトを使用してデータを N-Quads としてエクスポートする方法を示しています。

```
import requests

# Configure the URL here: e.g. http://localhost:9999/sparql
url = "http://localhost:9999/sparql"
payload = {'query': 'CONSTRUCT WHERE { hint:Query hint:analytic "true" . hint:Query
 hint:constructDistinctSPO "false" . ?s ?p ?o }'}
# Set the export format to be n-quads
headers = {
'Accept': 'text/x-nquads'
}
# Run the http request
response = requests.request("POST", url, headers=headers, data = payload, files = [])
#open the file in write mode, write the results, and close the file handler
f = open("export.nq", "w")
f.write(response.text)
f.close()
```

データがトリプルとして格納されている場合は、`Accept` ヘッダーパラメーターを変更し、Blazegraph GitHub レポで指定した値を使用して、適切な形式 (N-トリプル、RDF/XML、または Turtle) でデータをエクスポートします。。

Blazegraph エクスポートユーティリティを使用してデータをエクスポートする

Blazegraph には、データをエクスポートするためのユーティリティメソッド、すなわち `ExportKB` クラスが含まれています。`ExportKB` によって Blazegraph からのデータのエクスポートが容易になりますが、以前の方法とは異なり、エクスポートの実行中はサーバーをオフラインにする必要があります。移行中にBlazegraph をオフラインにしたり、データのバックアップから移行を実行したりする場合に使用するのが理想的な方法です。

Blazegraph がインストールされているが実行されていないマシンで、Java コマンドラインからユーティリティを実行します。このコマンドを実行する最も簡単な方法は、GitHub にある最新の blazegraph.jar リリースをダウンロードすることです。このコマンドを実行するには、いくつかのパラメータが必要です。

- **`log4j.primary.configuration`** — log4j プロパティファイルの場所。
- **`log4j.configuration`** — log4j プロパティファイルの場所。

- **`output`** — エクスポートされたデータの出力ディレクトリ。ファイルは `tar.gz` として、ナレッジベースに記載されているとおりの名前のサブディレクトリにあります。
- **`format`** — 目的の出力形式には `RWStore.properties` ファイルの場所が続きます。トリプルズで作業している場合は、`-format` パラメータを `N-Triples`、`Turtle` または `RDF/XML` に変更する必要があります。

たとえば、Blazegraph ジャーナルファイルとプロパティファイルがある場合、次のコードを使用してデータを N-Quads としてエクスポートします。

```
java -cp blazegraph.jar \
    com.bigdata.rdf.sail.ExportKB \
    -outdir ~/temp/ \
    -format N-Quads \
    ./RWStore.properties
```

エクスポートが成功した場合は、次のような出力が表示されます。

```
Exporting kb as N-Quads on /home/ec2-user/temp/kb
Effective output directory: /home/ec2-user/temp/kb
Writing /home/ec2-user/temp/kb/kb.properties
Writing /home/ec2-user/temp/kb/data.nq.gz
Done
```

Amazon Simple Storage Service (Amazon S3) バケットを作成し、エクスポートしたデータをコピーします。

Blazegraph からデータをエクスポートしたら、ターゲットの Neptune DB クラスターと同じリージョンに Amazon Simple Storage Service (Amazon S3) バケットを作成し、Neptune バルクローダーがデータをインポートするために使用します。

Amazon S3 のバケットの作成方法については、Amazon Simple Storage Service ユーザーガイドにあるS3 バケットを作成する方法および Amazon Simple Storage Service ユーザーガイドにあるバケットを作成する例を参照してください。

新しい Amazon S3 バケットにエクスポートしたデータファイルのコピー方法の手順については、Amazon Simple Storage Service ユーザーガイドにあるバケットにオブジェクトをアップロードする、またはAWS CLI と高レベル (s3) コマンドを使用するを参照してください。次のような Python コードを使用して、ファイルを 1 つずつコピーすることもできます。

```
import boto3

region = 'region name'
bucket_name = 'bucket name'
s3 = boto3.resource('s3')
s3.meta.client.upload_file('export.nq', bucket_name, 'export.nq')
```

Neptune バルクローダを使用して Neptune にデータをインポートする

Blazegraph からデータをエクスポートして Amazon S3 バケットにコピーしたら、データを Neptune にインポートする準備が整いました。Neptune には、SPARQL を使用してロード操作を実行するよりも高速で少ないオーバーヘッドでデータをロードできる一括ローダがあります。一括ローダープロセスは、特定された S3 バケットに格納されているデータを Neptune にロードするためのローダーエンドポイント API の呼び出しによって開始されます。

ローダー REST エンドポイントへの直接呼び出しでこれを行うことができますが、ターゲット Neptune インスタンスが実行されるプライベート VPC にアクセスできる必要があります。踏み台ホストをセットアップし、そのマシンに SSH をセットアップし、cURL コマンドを実行できますが、Neptune Workbenchを使う方が簡単です。

Neptune Workbench は、Amazon SageMaker ノートブックとして実行される、あらかじめ構成された Jupyter ノートブックであり、いくつかの Neptune 固有のノートブック magic がインストールされています。これらのマジックは、クラスターの状態のチェック、SPARQL および Gremlin トラバーサルの実行、一括ロード操作の実行など、一般的な Neptune オペレーションを簡素化します。

一括ロードプロセスを開始するには、`%load` magic を使いますが、これは Neptune ローダーコマンド を実行するためのインターフェースの役割をします。

1. AWS マネジメントコンソールにサインインして Amazon Neptune コンソール (https://console.aws.amazon.com/neptune/home) を開きます。
2. aws-neptune-blazegraph to Neptune を選択します。
3. [Open notebook] (ノートブックを開く) を選択します。
4. Jupyter の実行中のインスタンスで、既存のノートブックを選択するか、Python 3 カーネルを使用して新しいノートブックを作成します。
5. ノートブックでセルを開き、`%load` を入力し、セルを実行します。
6. 一括ローダーのパラメータを設定します。

a. 送信元には、インポートするソースファイルの場所を次のように入力します。`s3://{bucket_name}/{file_name}` 。
b. 形式には、適切な形式を選択します。この例では `nquads` です。
c. ARN のロードには、`IAMBulkLoad` ロールの ARN を入力します (この情報はロールの IAM コンソールにあります)。

7. [Submit] (送信) を選択します。

結果には、リクエストのステータスが含まれます。一括ロードは多くの場合、長時間実行されるプロセスであるため、応答はロードが完了したことを意味するものではなく、開始されたという意味しかありません。このステータス情報は、ジョブが完了したことを報告するまで定期的に更新されます。

> **Note**
>
> この情報は、ブログ記事、クラウドへの移行:Amazon Neptune への Blazegraph の移行でも入手できます。

Amazon Neptune にデータをロードする

Amazon Neptune にグラフデータをロードするには、いくつかの方法があります。

- 比較的少量のデータのみロードする必要がある場合、SPARQL `INSERT` ステートメントや Gremlin `addV` などのクエリと、`addE` ステップを使用できます。
- Neptune 一括ローダー を使用すると、外部ファイルに存在する大量のデータを取り込むことができます。バルクローダーコマンドは、クエリ言語コマンドよりも高速で、オーバーヘッドが少なくなります。これは、大規模なデータセットに最適化されており、RDF (リソース記述フレームワーク) データおよび Gremlin データの両方をサポートしています。
- AWS Database Migration Service （ AWS DMS) を使用して、他のデータストアからデータをインポートできます (AWS Database Migration Service を使用して別のデータストアから Amazon Neptune にデータをロードする「」、AWS Database Migration Service 「 ユーザーガイド」を参照) 。
- 最後に、Gremlin の `g.io(URL).read()` ステップを使用して、GraphML (XML 形式)、GraphSON (JSON 形式)、およびその他の形式のデータファイルを読み込むことができます。詳細については、「 TinkerPopドキュメント」を参照してください。

トピック

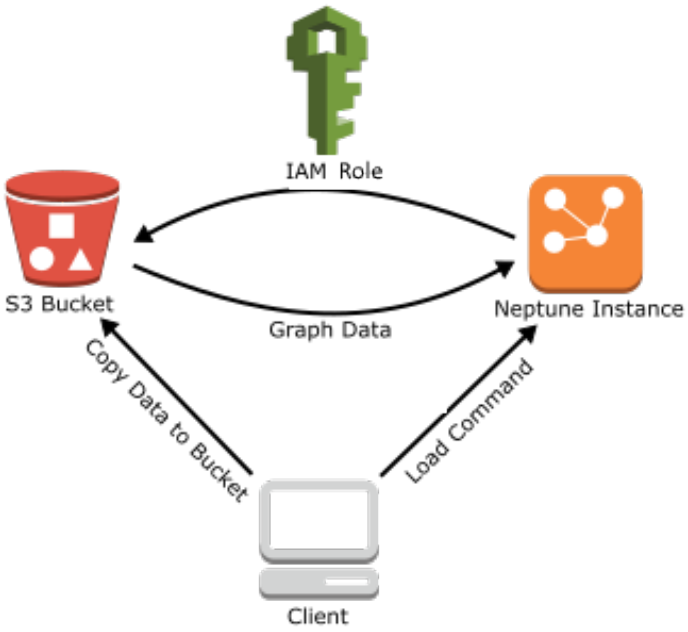
- Amazon Neptune 一括ローダーを使用したデータの取り込み
- AWS Database Migration Service を使用して別のデータストアから Amazon Neptune にデータをロードする

Amazon Neptune 一括ローダーを使用したデータの取り込み

Amazon Neptune は、外部ファイルから Neptune DB クラスターに直接データをロードする `Loader` コマンドを提供します。多数の `INSERT` ステートメント、`addV` および `addE` ステップ、その他の API 呼び出しを実行する代わりに、このコマンドを使用できます。

この Neptune Loader コマンドは高速で、オーバーヘッドが少なく、大規模なデータセットに最適化されており、Gremlin データと RDF (リソース記述フレームワーク) データおよび SPARQL が使用するデータの両方をサポートしています。

次の図に示しているのは、このロードプロセスの概要です。



以下に示しているのは、ロードプロセスの手順です。

1. Amazon Simple Storage Service (Amazon S3) バケットに、 ファイルをコピーします。
2. バケットへの読み取りアクセスとリストアクセスのある IAM ロールを作成します。
3. Amazon S3 VPC エンドポイントを作成します。
4. HTTP 経由で Neptune DB インスタンスにリクエストを送信して、Neptune ローダーを起動します。
5. Neptune DB インスタンスでは、バケットからデータをロードする IAM ロールを前提としています。

> **Note**
>
> Amazon S3 SSE-S3 または SSE-KMS モードのいずれかを使用して暗号化されている場合は、暗号化されたデータを Amazon S3 からロードできます。ただし、一括ロードに使用するロールが Amazon S3 オブジェクトにアクセスでき、SSE-KMS の場合は `kms:decrypt` にもアクセスできることが条件です。その場合、Neptune はユーザーの認証情報を偽装し、ユーザーに代わって `s3:getObject` 呼び出しを発行することができます。
> ただし、Neptune は現在 SSE-C モードを使用して暗号化されたデータの読み込みをサポートしていません。

次のセクションでは、Neptune へのデータの準備とロードの手順を説明します。

トピック

- 前提条件: IAM ロールと Amazon S3 アクセス
- ロードデータ形式
- 例: Neptune DB インスタンスにデータをロードする
- Amazon Neptune 一括ロードの最適化
- Neptune ローダーリファレンス

前提条件: IAM ロールと Amazon S3 アクセス

Amazon Simple Storage Service (Amazon S3) バケットからデータをロードするには、バケットにアクセスできる AWS Identity and Access Management (IAM) ロールが必要です。Amazon Neptune はデータをロードするためにこのロールを引き受けます。

> **Note**
>
> Amazon S3 SSE-S3 モードを使用して暗号化されている場合は、Amazon S3 から暗号化されたデータを読み込むことができます。その場合、Neptune はユーザーの認証情報を偽装し、ユーザーに代わって `s3:getObject` 呼び出しを発行することができます。
> また、IAM ロールに AWS KMSにアクセスするために必要なアクセス権限が含まれている限り、SSE-KMS モードを使用して暗号化された Amazon S3 から、暗号化されたデータをロードすることもできます。適切な AWS KMS アクセス許可がない場合、一括ロードオペレーションは失敗し、`LOAD_FAILED`レスポンスを返します。
> Neptune は現在 SSE-C モードを使用して暗号化された Amazon S3 データのロードをサポートしていません。

以下のセクションでは、マネージド IAM ポリシーを使用して Amazon S3 リソースにアクセスするための IAM ロールを作成し、Neptune クラスターにロールをアタッチする方法を示します。

トピック

- Amazon Neptune から Amazon S3 リソースにアクセスするための IAM ロールの作成
- Amazon Neptune クラスターに　IAM ロールを追加する
- Amazon S3 VPC エンドポイントの作成

- Amazon Neptune で IAM ロールを連鎖する

> **Note**
>
> これらの手順では、IAM コンソールにアクセスでき、IAM ロールとポリシーを管理するアクセス許可が必要です。詳細については、「IAM ユーザーガイド」の「 AWS マネジメントコンソールで作業するためのアクセス許可」を参照してください。
> Amazon Neptune コンソールでは、Neptune クラスターにロールをアタッチする以下の IAM アクセス権限が必要です。
>
> ```
> iam:GetAccountSummary on resource: *
> iam:ListAccountAliases on resource: *
> iam:PassRole on resource: * with iam:PassedToService restricted to
>  rds.amazonaws.com
> ```

Amazon Neptune から Amazon S3 リソースにアクセスするための IAM ロールの作成

`AmazonS3ReadOnlyAccess` マネージド IAM ポリシーを使用して、Amazon Neptune が Amazon S3 リソースにアクセスできるようにする新しい IAM ロールを作成します。

Amazon Neptune が Amazon S3 サービスにアクセスすることを許可する新しい IAM ロールを作成するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで Roles (ロール) を選択します。
3. [ロールの作成] を選択します。
4. [AWS のサービス] で [S3] を選択します。
5. [Next: Permissions] (次のステップ: 許可) を選択します。
6. フィルターボックスを使用して S3 という用語でフィルタリングし、AmazonS3ReadOnlyAccess の横にあるチェックボックスをオンにします。

> **Note**
>
> このポリシーによって、すべてのバケットに対する `s3:Get*` および `s3:List*` アクセス許可が付与されます。後の手順では、信頼ポリシーを使用してこのロールへのアクセスを制限します。

ローダーに必要なのは、ロード元のバケットに対する `s3:Get*` および `s3:List*` アクセス権限のみです。したがって、これらのアクセス権限を Amazon S3 リソース別に制限することもできます。
S3 バケットが暗号化されている場合には、`kms:Decrypt` 許可を追加する必要があります。

7. [次へ: レビュー] を選択します。
8. [Role Name] (ロール名) を IAM ロールの名前 (例: `NeptuneLoadFromS3`) に設定します。
オプションの [Role Description] (ロールの説明) 値 (「Allows Neptune to access Amazon S3 resources on your behalf.」など) を追加することもできます。
9. [ロールの作成] を選択します。
10. ナビゲーションペインで、[ロール] を選択します。
11. [検索] フィールドで、作成したロールの名前を入力し、リストに表示されたらそのロールを選択します。
12. [Trust Relationships] (信頼関係) タブで、[Edit trust relationship] (信頼関係の編集) を選択します。
13. テキストフィールドに次の信頼ポリシーをコピーして貼り付けます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "rds.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

14. [信頼ポリシーの更新] を選択します。
15. 「Amazon Neptune クラスターに　IAM ロールを追加する」のステップを完了します。

Amazon Neptune クラスターに　IAM ロールを追加する

コンソールを使用して IAM ロールを Amazon Neptune クラスターに追加します。これにより、クラスター内の任意の Neptune DB インスタンスがロールを引き受け、Amazon S3 からロードできるようになります。

> **Note**
>
> Amazon Neptune コンソールでは、Neptune クラスターにロールをアタッチする以下の IAM アクセス権限が必要です。
>
> ```
> iam:GetAccountSummary on resource: *
> iam:ListAccountAliases on resource: *
> iam:PassRole on resource: * with iam:PassedToService restricted to
>  rds.amazonaws.com
> ```

Amazon Neptune クラスターに　IAM ロールを追加するには

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/neptune/home で Amazon Neptune コンソールを開きます。
2. ナビゲーションペインで [データベース] を選択します。
3. 変更するクラスターのクラスター識別子を選択します。
4. Connectivity & Security タブを選択します。
5. IAM ロール セクションで、前のセクションで作成したロールを選択します。
6. [Add role] を選択します。
7. IAM ロールがクラスターにアクセスできるようになるまで待ってから、使用します。

Amazon S3 VPC エンドポイントの作成

Neptune ローダーには Amazon S3 のためにゲートウェイタイプの VPC エンドポイントが必要です。

Amazon S3 のアクセスをセットアップするには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/vpc/ で Amazon VPC コンソールを開きます。

2. ナビゲーションペインで、[エンドポイント] を選択します。
3. [エンドポイントの作成] を選択します。
4. ゲートウェイタイプのエンドポイントの サービス名 `com.amazonaws.region.s3` を選択します。

> **Note**
>
> このリージョンが正しくない場合は、コンソールのリージョンが正しいことを確認してください。

5. Neptune DB インスタンスを含む VPC を選択します (Neptune コンソールに DB インスタンスとして一覧表示されます)。
6. クラスターに関連するサブネットに関連付けられているルートテーブルの横にあるチェックボックスをオンにします。ルートテーブルが 1 つだけの場合は、そのボックスを選択する必要があります。
7. [エンドポイントの作成] を選択します。

エンドポイント作成の詳細については、Amazon VPC ユーザーガイドの VPC エンドポイントを参照してください。VPC エンドポイントの制限については、Amazon S3 の VPC エンドポイントを参照してください。

次のステップ

これで、Amazon S3 バケットへのアクセスが許可され、データをロードする準備ができました。サポートされる形式については、ロードデータ形式 を参照してください。

Amazon Neptune で IAM ロールを連鎖する

> **Important**
>
> エンジンリリース 1.2.1.0.R3 で導入された IAM ロールの連鎖を利用する新しい一括読み込みクロスアカウント機能では、一括読み込みのパフォーマンスが低下する場合があります。そのため、この機能をサポートするエンジンリリースへのアップグレードは、問題が解決されるまで一時的に中断されました。

クラスターにロールをアタッチすると、クラスターはそのロールを引き受けて、Amazon S3 に保存されているデータにアクセスできるようになります。エンジンリリース 1.2.1.0.R3 以降では、その

ロールが必要なすべてのリソースにアクセスできない場合、クラスターが他のリソースにアクセスできるように引き受けることができる 1 つ以上の追加のロールを連鎖できます。チェーン内の各ロールは、クラスターがチェーンの末尾のロールを引き受けるまで、チェーン内の次のロールを引き受けます。

ロールを連鎖するには、ロール間で信頼関係を確立します。例えば、RoleB を RoleA に連鎖するには、RoleA は RoleB の引き受けを許可するアクセス許可ポリシーを持っている必要があり、RoleB はそのアクセス許可を RoleA に戻すことを許可する信頼ポリシーが必要です。詳細については、「IAM ロールの使用」を参照してください。

チェーン内の最初のロールは、データをロードするクラスターにアタッチされる必要があります。

最初のロールと、チェーン内の次のロールを引き受ける後続の各ロールには、以下のものが必要です。

- `sts:AssumeRole` アクションに対する `Allow` 効果を持つ特定のステートメントを含むポリシー。
- `Resource` 要素内の次のロールの Amazon リソースネーム (ARN)。

> **Note**
>
> ターゲットの Amazon S3 バケットは、クラスターと同じ AWS リージョンに存在する必要があります。

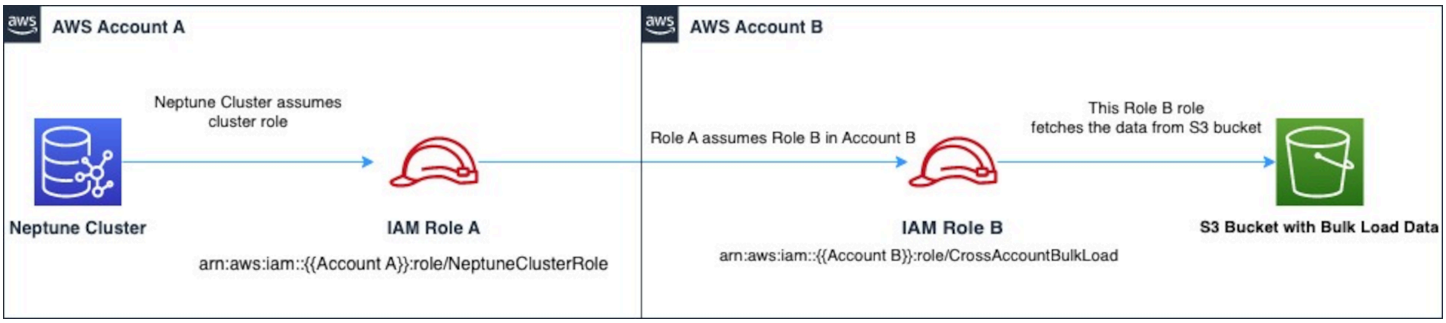
連鎖したロールを使用したクロスアカウントアクセス

別のアカウントに属する 1 つまたは複数のロールを連鎖することによって、クロスアカウントアクセスを付与できます。クラスターが別のアカウントに属するロールを一時的に引き受けると、そのアカウントのリソースにアクセスできるようになります。

例えば、アカウント A がアカウント B に属する Amazon S3 バケットのデータにアクセスしたいとします。

- アカウント A は、 という名前の Neptune AWS のサービスロールRoleAを作成し、クラスターにアタッチします。
- アカウント B は、アカウント B バケット内のデータにアクセスすることが承認されている RoleB という名前のロールを作成します。

- アカウント A は、RoleB を引き受けることを許可するアクセス許可ポリシーを RoleA にアタッチします。
- アカウント B は、RoleBアクセス許可を渡すことができる信頼ポリシーをアタッチします。RoleA
- アカウント B バケットのデータにアクセスするには、アカウント A は RoleA と RoleB を連鎖する iamRoleArn パラメータを使用してローダーコマンドを実行します。ローダー操作の継続期間中、RoleA は一時的に RoleB を引き受けて、アカウント B で Amazon S3 バケットにアクセスします。



例えば、RoleA には、Neptune との信頼関係を確立する信頼ポリシーがあるとします。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
          "Service": "rds.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

RoleA には、アカウント B が所有する RoleB を引き受けることを許可するアクセス許可ポリシーもあります。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
      "Sid": "Stmt1487639602000",
      "Effect": "Allow",
      "Action": [
        "sts:AssumeRole"
      ],
      "Resource": "arn:aws:iam::(Account B ID):role/RoleB"
    }
  ]
}
```

逆に、RoleB には、RoleA との信頼関係を確立する次のような信頼ポリシーがあるとします。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Principal": {
        "AWS": "arn:aws:iam::(Account A ID):role/RoleA"
      }
    }
  ]
}
```

RoleB には、アカウント B にある Amazon S3 バケット内のデータにアクセスするアクセス許可も必要です。

AWS Security Token Service (STS) VPC エンドポイントの作成

Neptune ローダーには、プライベート IP アドレスを介して AWS STS APIs にプライベートにアクセスするために IAM ロールを連鎖させる AWS STS 場合の VPC エンドポイントが必要です。Amazon VPC から VPC エンドポイント AWS STS を介して、安全でスケーラブルな方法で に直接接続できます。インターフェイス VPC エンドポイントを使用すると、アウトバウンドトラフィックのファイアウォールを開く必要がないため、セキュリティ体制が強化されます。また、Amazon VPC エンドポイントを使用する利点は他にもあります。

VPC エンドポイントを使用する場合、 へのトラフィック AWS STS はインターネット経由で送信されず、Amazon ネットワークを離れることもありません。VPC は、ネットワークトラフィックの可用性リスクや帯域幅の制約 AWS STS なしに、 に安全に接続されます。詳細については、「AWS STS インターフェイス VPC エンドポイントの使用」を参照してください。

AWS Security Token Service (STS) のアクセスを設定するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/vpc/ で Amazon VPC コンソールを開きます。
2. ナビゲーションペインで、[エンドポイント] を選択します。
3. [エンドポイントの作成] を選択します。
4. インターフェイスタイプのエンドポイントの サービス名 `com.amazonaws.region.sts` を選択します。
5. Neptune DB インスタンスと EC2 インスタンスが含まれている VPC を選択します。
6. EC2 インスタンスが存在するサブネットの横にあるチェックボックスを選択します。同じアベイラビリティーゾーンから複数のサブネットを選択することはできません。
7. [IP address type] (IP アドレスのタイプ) で、次のオプションから選択します。

   - [IPv4] — IPv4 アドレスをエンドポイントのネットワークインターフェイスに割り当てます。このオプションは、選択したすべてのサブネットに IPv4 アドレス範囲がある場合にのみサポートされます。
   - [IPv6] — IPv6 アドレスをエンドポイントのネットワークインターフェイスに割り当てます。このオプションは、選択されたすべてのサブネットが IPv6 のみのサブネットである場合にのみサポートされます。
   - [Dualstack] — IPv4 と IPv6 の両方のアドレスをエンドポイントのネットワークインターフェイスに割り当てます。このオプションは、選択したすべてのサブネットに IPv4 と IPv6 の両方のアドレス範囲がある場合にのみサポートされます。
8. [セキュリティグループ] で、VPC エンドポイントのエンドポイントネットワークインターフェイスに関連付けるセキュリティグループを選択します。Neptune DB インスタンスと EC2 インスタンスにアタッチされているすべてのセキュリティグループを選択する必要があります。
9. [Policy] (ポリシー) で [Full access] (フルアクセス) を選択して、すべてのリソースに対するすべてのプリンシパルによる VPC エンドポイント経由のすべてのオペレーションを許可します。それ以外の場合は、[Custom] (カスタム) を選択して、VPC エンドポイント経由でリソースに対してアクションを実行するためにプリンシパルが持つ許可を制御する VPC エンドポイントポリシーをアタッチします。このオプションは、サービスが VPC エンドポイントポリシーをサポートしている場合にのみ使用できます。詳細については、「エンドポイントポリシー」を参照してください。
10. (オプション) タグを追加するには、[新しいタグを追加] を選択し、そのタグのキーと値を入力します。
11. [エンドポイントの作成] を選択します。

エンドポイント作成の詳細については、「Amazon VPC ユーザーガイド」の「VPC エンドポイント」を参照してください。Amazon STS VPC エンドポイントは IAM ロール連鎖に必要な前提条件であることに注意してください。

AWS STS エンドポイントへのアクセス権を付与したので、データをロードする準備をすることができます。サポートされる形式の詳細については、「データ読み込み形式」を参照してください。

ローダーコマンド内でのロールの連鎖

`iamRoleArn` パラメータにロールの ARN のカンマ区切りリストを含めることによって、ローダーコマンドの実行時にロールの連鎖を指定できます。

ほとんどの場合、チェーンに含める必要があるのは 2 つのロールだけですが、3 つ以上のロールを連鎖することも可能です。例えば、このローダーコマンドは次の 3 つのロールを連鎖します。

```
curl -X POST https://localhost:8182/loader \
  -H 'Content-Type: application/json' \
  -d '{
        "source" : "s3://(the target bucket name)/(the target date file name)",
        "iamRoleArn" : "arn:aws:iam::(Account A ID):role/(RoleA),arn:aws:iam::(Account
 B ID):role/(RoleB),arn:aws:iam::(Account C ID):role/(RoleC)",
        "format" : "csv",
        "region" : "us-east-1"
      }'
```

ロードデータ形式

Amazon Neptune Load API は、さまざまな形式のデータのロードをサポートしています。

プロパティグラフのロード形式

以下のプロパティグラフ形式のいずれかでロードされたデータは、Gremlin と openCypher の両方を使用してクエリできます。

- Gremlin ロードデータ形式 (`csv`): カンマ区切り値 (CSV) 形式。
- openCypher データロード形式 (`opencypher`): カンマ区切り値 (CSV) 形式。

RDF ロード形式

SPARQL を使用してクエリするリソース記述フレームワーク (RDF) データをロードするには、World Wide Web Consortium (W3C) によって指定されている次の標準形式のいずれかを使用できます。

- N -Triples (`ntriples`) https://www.w3.org/TR/n-triples/ の仕様より。
- N-Quads (`nquads`) https://www.w3.org/TR/n-quads/ の仕様より。
- RDF/XML (`rdfxml`) https://www.w3.org/TR/rdf-syntax-grammar/ の仕様より。
- Turtle (`turtle`) https://www.w3.org/TR/turtle/ の仕様より。

ロードデータは UTF-8 エンコードを使用する必要がある

> **Important**
>
> すべてのロードデータファイルは、UTF-8 形式でエンコードされている必要があります。ファイルが UTF-8 でエンコードされていない場合でも、Neptune は UTF-8 形式のデータとしてロードしようとします。

Unicode 文字を含む N-Quads および N-Triples データの場合、\u*xxxxx* エスケープシーケンスがサポートされています。ただし、Neptune では正規化はサポートされていません。正規化が必要な値が存在する場合、クエリ byte-to-byte 中に一致しません。正規化の詳細については、Unicode.org の正規化ページを参照してください。

データがサポートされている形式でない場合は、ロードする前に変換する必要があります。

GraphML を Neptune CSV 形式に変換するためのツールは、 の GraphML2CSV プロジェクトで使用できますGitHub。

ロードデータファイルの圧縮サポート

Neptune は、`gzip` または `bzip2` 形式の個別ファイルの圧縮をサポートしています。

圧縮されたファイルには `.gz` または `.bz2` 拡張子を付ける必要があり、UTF-8 形式でエンコードされた単一のテキストファイルである必要があります。複数のファイルをロードできますが、それぞれ個別の `.gz`、`.bz2`、または圧縮されていないテキストファイルである必要があります。`.tar`、`.tar.gz`、`.tgz` などの拡張子の付いたアーカイブファイルはサポートされていません。

以下のセクションで、これらの形式についてさらに詳しく説明します。

トピック

- Gremlin ロードデータ形式
- openCypher データのロード形式
- RDF ロードデータ形式

Gremlin ロードデータ形式

CSV 形式を使用して Apache TinkerPop Gremlin データをロードするには、頂点とエッジを別々のファイルで指定する必要があります。

ローダーは、単一のロードジョブで複数の頂点ファイルおよび複数のエッジファイルからロードできます。

各ロードコマンドは、ロードされる一連のファイルと同じ Amazon S3 バケットのフォルダにある必要があり、`source` パラメータにフォルダ名を指定します。このファイル名とファイル名拡張子は重要ではありません。

Amazon Neptune CSV 形式は RFC 4180 の CSV の仕様に従います。詳細については、Internet Engineering Task Force (IETF) ウェブサイトの、CSV ファイルの一般形式と MIME タイプを参照してください。

> **Note**
>
> すべてのファイルは、UTF-8 形式でエンコードする必要があります。

各ファイルには、カンマ区切りのヘッダー行があります。ヘッダー行は、システム列ヘッダーとプロパティ列ヘッダーの両方で構成されます。

システム列ヘッダー

頂点ファイルとエッジファイルでは、必須および許可されたシステム列ヘッダーが異なります。

各システム列は、ヘッダーに 1 回のみ表示できます。

すべてのラベルで大文字と小文字が区別されます。

頂点ヘッダー

- `~id` - 必須

  頂点の ID。

- `~label`

  頂点のラベル。複数のラベル値を使用できます。セミコロン (`;`) で区切ります。

  `~label` が存在しない場合、すべての頂点に少なくとも 1 つのラベルが必要であるため`vertex`、は値 を持つラベル TinkerPop を提供します。

エッジヘッダー

- `~id` - 必須

  エッジの ID。

- `~from` - 必須

  from 頂点の頂点 ID。

- `~to` - 必須

  to 頂点の頂点 ID。

- `~label`

  エッジのラベル。エッジには 1 つのラベルのみを含めることができます。

  `~label` が存在しない場合、すべてのエッジにラベルが必要であるため`edge`、 は値 を持つラベル TinkerPop を提供します。

プロパティ列ヘッダー

次の構文を使用して、プロパティ列 (`:`) を指定できます。タイプ名では大文字と小文字が区別されません。ただし、プロパティ名内にコロンがある場合は、次のように、その前にバックスラッシュを付けてエスケープする必要があります。`\:`。

```
propertyname:type
```

> **Note**
>
> 列ヘッダーでは、スペース、カンマ、キャリッジリターン、改行文字は使用できません。そのため、プロパティ名にこれらの文字を含めることはできません。

タイプに [] を追加することで、配列型の列を指定できます。

```
propertyname:type[]
```

> **Note**
>
> エッジのプロパティに指定できるのは 1 つの値のみです。配列型が指定された場合や 2 つ目の値が指定された場合はエラーが発生します。

次の例は、Int 型の age という名前のプロパティの列ヘッダーを示しています。

```
age:Int
```

ファイルのすべての行は、その位置に整数があるか、空のままにする必要があります。

文字列の配列は許可されますが、バックスラッシュを使用してエスケープされないり限配列内の文字列にはセミコロン (;) 文字を含めることはできません (次のように:\;)。

列の濃度を指定する

リリース 1.0.1.0.200366.0 (2019-07-26) では、列ヘッダーを使用して、列で識別されたプロパティの濃度を指定できます。これにより、バルクローダーで Gremlin クエリと同様に濃度を重視できます。

列の濃度は、次のように指定します。

```
propertyname:type(cardinality)
```

##値は single または set となります。デフォルトは set であると想定されます。これは、列が複数の値を受け入れられることを意味します。エッジファイルの場合、濃度は常に単一であり、他の濃度を指定すると、ローダーは例外をスローします。

濃度が single のとき、値がロードされたときに前の値がすでに存在する場合、または複数の値がロードされた場合、ローダーによりエラーがスローされます。updateSingleCardinalityProperties フラグを使用して新しい値がロードされたとき、既存の値が置き換えられるように、 この動作をオーバーライドできます。ローダーコマンド を参照してください。

通常、その必要はありませんが、配列型で濃度設定を使用できます。可能な組み合わせは次のとおりです。

- name:type - 濃度は set で、コンテンツは単一の値です。
- name:type[] - 濃度は set で、コンテンツは複数の値です。
- name:type(single) - 濃度は single で、コンテンツは単一の値です。
- name:type(set) - 濃度は、デフォルトと同じ set で、コンテンツは単一の値です。
- name:type(set)[] - 濃度は set で、コンテンツは複数の値です。
- name:type(single)[] - これは矛盾しており、エラーがスローされます。

次のセクションでは、使用可能なすべての Gremlin データ型を示します。

Gremlin データ型

これは許可されたプロパティタイプの一覧で、各タイプの説明を含みます。

Bool (または Boolean)

ブールフィールドであることを示しています。許可された値: false、true

> **Note**
>
> true 以外の値は false として扱われます。

整数型

定義された範囲外の値の場合、エラーが発生します。

タイプ	[Range] (範囲)
バイト	-128 ~ 127

ショート	-32768 ~ 32767
Int	-2^31 ~ 2^31-1
Long	-2^63 ~ 2^63-1

10 進数型

指数表記または 10 進表記の両方をサポートしています。また、(+/-) INFINITY や NaN などの記号も使用できます。INF はサポートされていません。

タイプ	[Range] (範囲)
浮動小数点数	32 ビット IEEE 754 浮動小数点
ダブル	64 ビット IEEE 754 浮動小数点

長すぎる浮動小数点数や倍精度浮動小数点数の値は、24 ビット (浮動小数点数) および 53 ビット (倍精度浮動小数点数) の精度で最も近い値にロードされ、丸められます。中間の値は、ビットレベルの最後の残りの桁で 0 に丸められます。

文字列

引用符はオプションです。カンマ、改行、およびキャリッジリターン文字は、二重引用符 (") で囲まれた文字列に含まれると自動的にエスケープされます。例: `"Hello, World"`

引用符で囲まれた文字列に引用符を含めるには、行内で 2 つ使用して引用符をエスケープします。例 : `"Hello ""World"""`

文字列の配列は許可されますが、バックスラッシュを使用してエスケープされないり限配列内の文字列にはセミコロン (;) 文字を含めることはできません (次のように:\;)。

配列内の文字列を引用符で囲む場合は、配列全体を 1 組の引用符で囲む必要があります。例: `"String one; String 2; String 3"`

日付

ISO-8601 形式の Java の日付。以下の形式がサポートされています。yyyy-MM-dd、yyyy-MM-ddTHH:mm、yyyy-MM-ddTHH:mm:ss、yyyy-MM-ddTHH:mm:ssZ

Gremlin 行形式

区切り記号

行内のフィールドはカンマで区切られます。レコードは、改行またはキャリッジリターンとそれに続く改行で区切られます。

空のフィールド

空のフィールドは、必須ではない列 (ユーザー定義のプロパティなど) に許可されています。空のフィールドにもカンマ区切り記号が必要です。必須列の空白のフィールドは解析エラーになります。空の文字列値は、空のフィールドではなく、 フィールドの空の文字列値として解釈されます。次のセクションの例では、各頂点の例に空白のフィールドがあります。

頂点 ID

~id 値はすべての頂点ファイル内のすべての頂点に対して一意である必要があります。~id 値が同一の複数の頂点行はグラフの単一の頂点に適用されます。空の文字列 ("") は有効な ID であり、頂点は空の文字列を ID として作成します。

エッジ ID

また、~id 値はすべてのエッジファイル内のすべてのエッジに対して一意である必要があります。~id 値が同一の複数のエッジ行はグラフの単一のエッジに適用されます。空の文字列 ("") は有効な ID であり、エッジは空の文字列を ID として作成します。

ラベル

ラベルでは大文字と小文字が区別され、空にすることはできません。の値を指定すると、エラー""が発生します。

文字列値

引用符はオプションです。カンマ、改行、およびキャリッジリターン文字は、二重引用符 (") で囲まれた文字列に含まれると自動的にエスケープされます。空の文字列値は、空のフィールドではなく、フィールドの空の文字列値として解釈("")されます。

CSV 形式の仕様

Neptune CSV 形式は RFC 4180 の CSV の仕様に従い、次の要件を含みます。

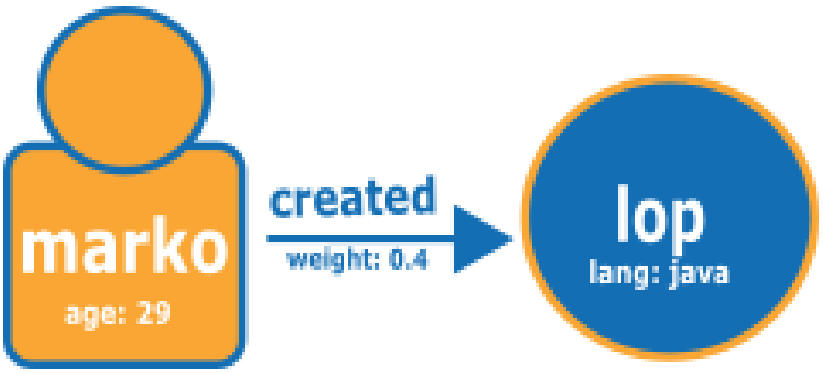
- Unix と Windows の両方のスタイルの行末処理がサポートされています (\n または \r\n)。

- フィールドはすべて引用符で囲むことができます (二重引用符を使用)。
- 改行、二重引用符、またはカンマを含むフィールドは、引用符で囲む必要があります。(そうでない場合、ロードは即座に中止されます)。
- フィールド内の二重引用符文字 (") は、2 つの二重引用符文字で示す必要があります。たとえば、データ内で、文字列 `Hello "World"` は、`"Hello ""World"""` であることが必要です。
- 区切り記号間のスペースは無視されます。行が `value1, value2` として存在する場合は、`"value1"` および `"value2"` として保存されます。
- その他のエスケープ文字はすべてそのまま保存されます。たとえば、`"data1\tdata2"` であれば `"data1\tdata2"` として保存されます。これらの文字が引用符で囲まれている場合、これ以上エスケープする必要はありません。
- 空のフィールドは許可されます。空白のフィールドは空の値と見なされます。
- フィールドの複数の値は、値と値の間にセミコロン (;) を使用して指定します。

詳細については、Internet Engineering Task Force (IETF) ウェブサイトの、CSV ファイルの一般形式と MIME タイプを参照してください。

Gremlin の例

次の図は、2 つの頂点と TinkerPop 、モダングラフから取得したエッジの例を示しています。



以下に示しているのは、Neptune CSV ロード形式のグラフです。

頂点ファイル:

```
~id,name:String,age:Int,lang:String,interests:String[],~label
v1,"marko",29,,"sailing;graphs",person
v2,"lop",,"java",,software
```

頂点ファイルの表形式のビュー:

~id	名前: 文字列	年齢: 整数	言語: 文字列	関心：文字列[]	~label
v1	"marko"	29		["sailing", "graphs"]	person
v2	"lop"		"java"		ソフトウェア

エッジファイル:

```
~id,~from,~to,~label,weight:Double
e1,v1,v2,created,0.4
```

エッジファイルの表形式のビュー:

~id	~from	~to	~label	weight:Double
e1	v1	v2	作成済み	0.4

次のステップ

次に、ロード形式の詳細については、例: Neptune DB インスタンスにデータをロードする を参照してください。

openCypher データのロード形式

openCypher CSV 形式を使用して openCypher データをロードするには、ノードとリレーションシップを別々のファイルで指定する必要があります。ローダーは、単一のロードジョブで、これらのノードファイルとリレーションシップファイルの複数のからロードできます。

各ロードコマンドは、ロードされる一連のファイルと同じAmazon Simple Storage Service バケットのパスプレフィックスを持つ必要があります。そのプレフィックスは始点パラメータで指定します。実際のファイル名と拡張子は重要ではありません。

Amazon Neptune では openCypher CSV 形式は RFC 4180 の CSV の仕様に従います。詳細については、Internet Engineering Task Force (IETF) ウェブサイトの、CSV ファイルの一般形式と MIME タイプ (https://tools.ietf.org/html/rfc4180) を参照してください。

> **Note**
>
> これらのファイルは、UTF-8 形式でエンコードする必要があります。

各ファイルには、システム列ヘッダーとプロパティ列ヘッダーの両方を含むカンマ区切りのヘッダー行があります。

openCypher データロードファイルのシステム列ヘッダー

特定のシステム列は、各ファイルに 1 回のみ表示できます。すべてのシステム列ヘッダーラベルで、大文字と小文字が区別されます。

openCypher ノードのロードファイルおよびリレーションシップロードファイルでは、必須および許可されるシステム列ヘッダーが異なります。

ノードファイル内のシステム列ヘッダー

- **`:ID`** — (必須) ノードの ID。

  オプションの ID スペースを `:ID(`*`ID Space`*`)` のように、ノード `:ID` 列ヘッダーに追加できます。例は `:ID(movies)` です。

  このファイル内のノードを接続するリレーションシップをロードするときは、リレーションシップファイルの `:START_ID` および/または `:END_ID` 列で同じ ID スペースを使用します。

  ノード `:ID` はオプションで、フォーム、*`property name`*`:ID` にプロパティとして格納できます。例は `name:ID` です。

  ノード ID は、現在および以前のロードのすべてのノードファイルで一意である必要があります。ID スペースを使用する場合、ノード ID は、現在のロードと以前のロードで同じ ID スペースを使用するすべてのノードファイルで一意である必要があります。
- **`:LABEL`** — ノードのラベル。

  複数のラベル値を使用できます。セミコロン (`;`) で区切ります。

リレーションシップファイル内のシステム列ヘッダー

- **`:ID`** — リレーションシップの ID。これは、`userProvidedEdgeIds` が true (デフォルト) である場合に必要ですが、`userProvidedEdgeIds` が `false` の場合は無効となります。

リレーションシップ ID は、現在および以前のロードのすべてのリレーションシップファイルで一意である必要があります。

- **:START_ID** — (必須) このリレーションシップが始まるノードのノード ID。

  必要に応じて、ID スペースをフォーム :START_ID(*ID Space*) の開始 ID 列に関連付けることができます。開始ノード ID に割り当てられた ID スペースは、ノードファイル内のノードに割り当てられている ID スペースと一致する必要があります。

- **:END_ID** — (必須) このリレーションシップが終了するノードのノード ID。

  必要に応じて、ID スペースをフォーム :END_ID(*ID Space*) の終了 ID 列に関連付けることができます。終了ノード ID に割り当てられた ID スペースは、ノードファイル内のノードに割り当てられた ID スペースと一致する必要があります。

- **:TYPE** — リレーションシップのタイプ。リレーションシップには、単一タイプのみを含めることができます。

> **Note**
>
> 一括ロードプロセスで重複するノードまたはリレーションシップ ID がどのように処理されるかについては、openCypher データをロードする を参照してください。

openCypher データロードファイルのプロパティ列ヘッダー

次の形式のプロパティ列ヘッダーを使用して、列が特定のプロパティの値を保持するように指定できます。

```
propertyname:type
```

列ヘッダーでは、スペース、カンマ、キャリッジリターン、改行文字は使用できません。そのため、プロパティ名にこれらの文字を含めることはできません。以下に示しているのは、タイプ Int の age という名前のプロパティの列ヘッダーの例です。

```
age:Int
```

列ヘッダーとして age:Int がある列は、すべての行に整数または空の値を含める必要があります。

Neptune openCypher データロードファイルのデータ型

- **Bool** または **Boolean** — ブールフィールド。指定できる値は `true` と `false` です。

  `true` 以外の値は `false` として扱われます。
- **Byte** — -128 から 127 範囲内の整数。
- **Short** — -32,768 から 32,767 範囲内の整数。
- **Int** — -2^31 から 2^31 - 1 範囲内の整数。
- **Long** — -2^63 から 2^63 - 1 範囲内の整数。
- **Float** — 32 ビット IEEE 754 浮動小数点数。十進表記と科学記法の両方がサポートされています。Infinity、-Infinity および NaN はすべて認識されますが、INF はされません。

  桁数が多すぎて収まらない値は、最も近い値に丸められます (中間の値は、ビットレベルの最後の残りの桁で 0 に丸められます)。
- **Double** — 64 ビット IEEE 754 浮動小数点数。十進表記と科学記法の両方がサポートされています。Infinity、-Infinity および NaN はすべて認識されますが、INF はされません。

  桁数が多すぎて収まらない値は、最も近い値に丸められます (中間の値は、ビットレベルの最後の残りの桁で 0 に丸められます)。
- **String** - 引用符はオプションです。カンマ、改行、およびキャリッジリターン文字は、"Hello, World" のような二重引用符 (") で囲まれた文字列に含まれると自動的に工スケープされます。

  引用符で囲まれた文字列に引用符を含めるには、"Hello ""World""" のように、行内で 2 つ使用してください。
- **DateTime** — 次のいずれかの ISO-8601 形式の Java 日付。
  - yyyy-MM-dd
  - yyyy-MM-ddTHH:mm
  - yyyy-MM-ddTHH:mm:ss
  - yyyy-MM-ddTHH:mm:ssZ

Neptune openCypher データロードファイルのオートキャストデータ型

オートキャストデータ型は、現在 Neptune がネイティブにサポートしていないデータ型をロードするために提供されます。このような列のデータは文字列として保存され、意図した形式に対する検証は行われません。次のオートキャストデータ型を使用できます。

- **Char** — Char フィールド。文字列として格納されます。
- **Date**、**LocalDate** および **LocalDateTime**、 — date、localdate および localdatetime タイプの説明については、Neo4j 一時的インスタントを参照してください。値は、検証なしで文字列として逐語的にロードされます。
- **Duration** — Neo4j デュレーション形式を参照してください。値は、検証なしで文字列として逐語的にロードされます。
- ポイント — 空間データを格納するためのポイントフィールド。「空間インスタント」を参照してください。値は、検証なしで文字列として逐語的にロードされます。

openCypher ロードフォーマットの例

Modern Graph から取得した次の図は、2 TinkerPop つのノードとリレーションシップの例を示しています。

以下に示しているのは、通常の Neptune openCypher ロード形式のグラフです。

ノードファイル:

```
:ID,name:String,age:Int,lang:String,:LABEL
v1,"marko",29,,person
v2,"lop",,"java",software
```

リレーションシップファイル:

```

:ID,:START_ID,:END_ID,:TYPE,weight:Double
e1,v1,v2,created,0.4
```

または、次のようにプロパティとして ID スペースと ID を使用することもできます。

最初のノードファイル:

```
name:ID(person),age:Int,lang:String,:LABEL
"marko",29,,person
```

2 番目のノードファイル:

```
name:ID(software),age:Int,lang:String,:LABEL
"lop",,"java",software
```

リレーションシップファイル:

```
:ID,:START_ID,:END_ID,:TYPE,weight:Double
e1,"marko","lop",created,0.4
```

RDF ロードデータ形式

Resource Description Framework (RDF) データをロードするには、World Wide Web Consortium (W3C) によって指定されている次の標準形式のいずれかを使用できます。

- N -Triples (`ntriples`) https://www.w3.org/TR/n-triples/ の仕様より。
- N-Quads (`nquads`) https://www.w3.org/TR/n-quads/ の仕様より
- RDF/XML (`rdfxml`) https://www.w3.org/TR/rdf-syntax-grammar/ の仕様より
- Turtle (`turtle`) https://www.w3.org/TR/turtle/ の仕様より

> **Important**
>
> すべてのファイルは、UTF-8 形式でエンコードする必要があります。
> Unicode 文字を含む N-Quads および N-Triples データの場合、\u*xxxxx* エスケープシーケンスがサポートされています。ただし、Neptune では正規化はサポートされていません。正規化が必要な値が存在する場合、クエリ byte-to-byte 中に一致しません。正規化の詳細については、Unicode.org の正規化ページを参照してください。

次のステップ

次に、ロード形式の詳細については、例: Neptune DB インスタンスにデータをロードする を参照してください。

例: Neptune DB インスタンスにデータをロードする

この例では、Amazon Neptune にデータをロードする方法を示しています。別途指定されていない限り、Neptune DB インスタンスと同じ Amazon Virtual Private Cloud (VPC) の Amazon Elastic Compute Cloud (Amazon EC2) インスタンスからの手順に従います。

データをロードする例の前提条件

始めるには以下のものが必要です。

- Neptune DB インスタンス。

  Neptune DB インスタンスの起動方法についての詳細は、新しい Neptune DB クラスターの作成 を参照してください。
- データファイルを入れる Amazon Simple Storage Service (Amazon S3) バケット。

  既存のバケットを使用することもできます。S3 バケットがない場合は、Amazon S3 入門ガイドのバケットの作成を参照してください。
- ロードするグラフデータ (Neptune ローダーでサポートされている形式のいずれか):

  グラフのクエリに Gremlin を使用している場合、Neptune は comma-separated-values 「」で説明されているように (CSV) 形式でデータをロードできますGremlin ロードデータ形式。

  openCypher を使用してグラフのクエリを実行している場合、Neptune では、openCypher データのロード形式 の説明に従って、openCypher 固有の CSV 形式でデータをロードすることもできます。

  SPARQL を使用している場合、Neptune は多くの RDF 形式でデータをロードできます (RDF ロードデータ形式 を参照)。
- S3 バケット内のデータファイルへのアクセスを許可する IAM ポリシーが設定されている Neptune DB インスタンスの IAM ロール。このポリシーが読み取りとリストアクセス許可を付与する必要があります。

  Amazon S3 にアクセスできるロールを作成し、Neptune クラスターに関連付ける方法の詳細については、前提条件: IAM ロールと Amazon S3 アクセス を参照してください。

> **Note**
>
> Neptune Load API はデータファイルへの読み取りアクセスが必要です。IAM ポリシーに書き込みアクセスまたはバケット全体へのアクセスを許可する必要はありません。

- Amazon S3 VPC エンドポイント 詳細については、「Amazon S3 VPC エンドポイントの作成」セクションを参照してください。

Amazon S3 VPC エンドポイントの作成

Neptune ローダーには Amazon S3 の VPC エンドポイントが必要です。

Amazon S3 のアクセスをセットアップするには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/vpc/ で Amazon VPC コンソールを開きます。
2. 左のナビゲーションペインで [エンドポイント] を選択します。
3. [エンドポイントの作成] を選択します。
4. [Service Name] (サービス名) `com.amazonaws.region.s3` を選択します。

   > **Note**
   >
   > このリージョンが正しくない場合は、コンソールのリージョンが正しいことを確認してください。

5. [VPC] で、Neptune DB インスタンスが含まれている VPC を選択します。
6. クラスターに関連するサブネットに関連付けられているルートテーブルの横にあるチェックボックスをオンにします。ルートテーブルが 1 つだけの場合は、そのボックスを選択する必要があります。
7. [エンドポイントの作成] を選択します。

エンドポイント作成の詳細については、Amazon VPC ユーザーガイドの VPC エンドポイントを参照してください。VPC エンドポイントの制限については、Amazon S3 の VPC エンドポイントを参照してください。

Neptune DB インスタンスにデータをロードするには

1. データファイルを Amazon S3 バケットにコピーします。S3 バケットは、データをロードするクラスターと同じ AWS リージョンに存在する必要があります。

   次の AWS CLI コマンドを使用して、ファイルをバケットにコピーできます。

   > **Note**
   >
   > このコマンドは、Amazon EC2 インスタンスから実行する必要はありません。

   ```
   aws s3 cp data-file-name s3://bucket-name/object-key-name
   ```

   > **Note**
   >
   > Amazon S3 で、オブジェクトキー名は、ファイル名を含むファイルの完全なパスです。例: コマンド `aws s3 cp datafile.txt s3://examplebucket/mydirectory/datafile.txt` では、オブジェクトキー名は **`mydirectory/datafile.txt`** です。

   または、 を使用して AWS Management Console S3 バケットにファイルをアップロードすることもできます。Amazon S3 コンソール (https://console.aws.amazon.com/s3/) を開きバケットを選択ます。左上隅の [Upload] (アップロード) を選択して、ファイルをアップロードします。

2. コマンドラインウィンドウから次のように入力し、エンドポイント、Amazon S3 パス、フォーマット、IAM ロール ARN の正しい値を使用して Neptune ローダーを実行します。

   `format` パラメータには、Gremlin の場合は `csv`、openCypher の場合は `opencypher`、または RDF の場合は `ntriples`、`nquads`、`turtle`、および `rdfxml` のいずれかの値を指定できます。他のパラメータについての詳細は、「Neptune ローダーコマンド」を参照してください。

   Neptune DB インスタンスのホスト名を見つける方法については、Amazon Neptune エンドポイントに接続する セクションを参照してください。

   region パラメータの値はクラスターおよび S3 バケットのリージョンと一致する必要があります。

   Amazon Neptune は、次の AWS リージョンで利用できます。

- 米国東部 (バージニア北部): us-east-1
- 米国東部 (オハイオ): us-east-2
- 米国西部 (北カリフォルニア): us-west-1
- 米国西部 (オレゴン): us-west-2
- カナダ (中部): ca-central-1
- 南米 (サンパウロ): sa-east-1
- 欧州 (ストックホルム): eu-north-1
- 欧州 (アイルランド): eu-west-1
- 欧州 (ロンドン): eu-west-2
- 欧州 (パリ): eu-west-3
- 欧州 (フランクフルト): eu-central-1
- 中東 (バーレーン): me-south-1
- 中東 (アラブ首長国連邦): me-central-1
- イスラエル (テルアビブ): il-central-1
- アフリカ (ケープタウン): af-south-1
- アジアパシフィック (香港): ap-east-1
- アジアパシフィック (東京): ap-northeast-1
- アジアパシフィック (ソウル): ap-northeast-2
- アジアパシフィック (大阪): ap-northeast-3
- アジアパシフィック (シンガポール): ap-southeast-1
- アジアパシフィック (シドニー): ap-southeast-2
- アジアパシフィック (ムンバイ): ap-south-1
- 中国 (北京): cn-north-1
- 中国 (寧夏): cn-northwest-1
- AWS GovCloud （米国西部): us-gov-west-1
- AWS GovCloud （米国東部): us-gov-east-1

```
curl -X POST \
    -H 'Content-Type: application/json' \
    https://your-neptune-endpoint:port/loader -d '
```

```
    {
      "source" : "s3://bucket-name/object-key-name",
      "format" : "format",
      "iamRoleArn" : "arn:aws:iam::account-id:role/role-name",
      "region" : "region",
      "failOnError" : "FALSE",
      "parallelism" : "MEDIUM",
      "updateSingleCardinalityProperties" : "FALSE",
      "queueRequest" : "TRUE",
      "dependencies" : ["load_A_id", "load_B_id"]
    }'
```

Neptune クラスターへの IAM ロールの作成と関連付けの詳細については、前提条件: IAM ロールと Amazon S3 アクセス を参照してください。

> **Note**
>
> ロードリクエストパラメータの詳細については、Neptune ローダーのリクエストパラメータ を参照してください。概要:
> `source` パラメータは、単一のファイルまたはフォルダを指す Amazon S3 URI を受け取ります。フォルダを指定すると、Neptune はフォルダ内のすべてのデータファイルをロードします。
> フォルダには複数の頂点ファイルおよび複数のエッジファイルが含まれている場合があります。
> URI は、以下の形式のいずれかになります。
>
> - `s3://bucket_name/object-key-name`
> - `https://s3.amazonaws.com/bucket_name/object-key-name`
> - `https://s3-us-east-1.amazonaws.com/bucket_name/object-key-name`
>
> この `format` パラメータは、次のいずれかになります。
>
> - Gremlin プロパティグラフの Gremlin CSV 形式 (`csv`)
> - openCypher プロパティグラフの openCypher CSV 形式 (`opencypher`)
> - RDF の N -Triples (`ntriples`) 形式 / SPARQL
> - RDF の N-Quads (`nquads`) 形式 / SPARQL
> - RDF の RDF/XML (`rdfxml`) 形式 / SPARQL

- RDF の Turtle (`turtle`) 形式 / SPARQL

オプションの `parallelism` パラメータを使用すると、バルクロードプロセスで使用されるスレッドの数を制限できます。`LOW`、`MEDIUM`、`HIGH`、または `OVERSUBSCRIBE` に設定できます。
`updateSingleCardinalityProperties` を `"FALSE"` に設定すると、エッジまたは単一カーディナリティ頂点プロパティにロードされているソースファイルに複数の値が指定されている場合、ローダーはエラーを返します。
`queueRequest` を `"TRUE"` に設定すると、ロードジョブがすでに実行されている場合、キューにロードリクエストが配置されます。
`dependencies` パラメータは、すでにキューに配置されている 1 つ以上のロードジョブが正常に完了した場合に、ロードリクエストを実行します。

3. Neptune ローダーは、ステータスを確認したり、ロードプロセスをキャンセルしたりできるジョブ `id` を返します。

```
{
    "status" : "200 OK",
    "payload" : {
        "loadId" : "ef478d76-d9da-4d94-8ff1-08d9d4863aa5"
    }
}
```

4. ステップ 3 の `loadId` でロードのステータスを取得するには、次のように入力します。

```
curl -G 'https://your-neptune-endpoint:port/loader/ef478d76-
d9da-4d94-8ff1-08d9d4863aa5'
```

ロードのステータスにエラーが表示されている場合は、より詳細なステータスとエラーのリストをリクエストできます。詳細な説明と例については、「Neptune Loader Get-Status API」を参照してください。

5. (オプション) Load ジョブをキャンセルします。

ステップ 3 のジョブからの `id` とともに、次の `Delete` をローダージョブに入力します。

```
curl -X DELETE 'https://your-neptune-endpoint:port/loader/ef478d76-
d9da-4d94-8ff1-08d9d4863aa5'
```

この DELETE コマンドを使用すると、キャンセルが成功したときに HTTP コード 200 OK が返されます。

ロードが完了したロードジョブのファイルからのデータはロールバックされません。データは Neptune DB インスタンスに残ります。

Amazon Neptune 一括ロードの最適化

Neptune 一括ロードのロード時間を最小限に抑えるには、次の方法を使用します。

- データを消去します。
  - データを必ずロードする前にサポートされているデータ形式へ変換してください。
  - 重複または既知のエラーをすべて削除します。
  - 一意の述語 (エッジや頂点のプロパティなど) の数をできるだけ減らします。
- ファイルを最適化します:
  - Amazon S3 バケットから CSV ファイルなどの大きなファイルをロードする場合、ローダーはそれらを並行してロードできるチャンクに解析することによって同時性を管理します。非常に多くの小さなファイルを使用すると、このプロセスが遅くなる可能性があります。
  - Amazon S3 フォルダから複数のファイルをロードする場合、ローダーは最初に頂点ファイルをロードし、次にエッジファイルをロードします。
  - ファイルを圧縮すると、転送時間が短縮されます。ローダーは gzip ソースファイルの圧縮をサポートしています。
- ローダーの設定を確認します。
  - ロード中に他の操作を行う必要がない場合は、OVERSUBSCRIBEparallelism パラメータを使用します。このパラメータ設定では、一括ローダーは、実行時に使用可能なすべての CPU リソースを使用するようになります。一般に、I/O 制約が許す限り速くオペレーションを実行し続けるには、CPU 容量の 60% ~ 70% が必要です。

> **Note**
>
> parallelism が OVERSUBSCRIBE または HIGH (デフォルト設定) に設定されていると、openCypher データをロードするときに、スレッドが競合状態となりデッドロックが発生する可能性があり、LOAD_DATA_DEADLOCK というエラーが表示されます。この場合は、parallelism を低い設定にして、ロードを再試行します。

- ロードジョブに複数のロードリクエストが含まれる場合は、queueRequest パラメータを使用します。queueRequest を TRUE に設定すると、Neptune はリクエストをキューに入れるので、別のリクエストを発行する前にリクエストが完了するのを待つ必要がありません。
- ロードリクエストがキューに入れられている場合は、dependencies パラメータを指定すると、1 つのジョブが失敗すると、依存ジョブが失敗します。これにより、ロードされたデータの不整合を防ぐことができます。
- ロードジョブで以前にロードされた値の更新が含まれる場合は、必ず updateSingleCardinalityProperties パラメータを TRUE に設定してください。そうでない場合、ローダーは既存の単一カーディナリティ値の更新試行をエラーとして扱います。Gremlin データの場合、カーディナリティはプロパティの列ヘッダーにも指定されます (プロパティ列ヘッダー)。

  > **Note**
  >
  > updateSingleCardinalityProperties パラメータは Resource Description Framework (RDF) データには使用できません。

- failOnError パラメータを使用して、エラーが発生したときにバルクロード操作が失敗するか、続行するかを決定します。また、mode パラメータを使用して、ロードジョブがすでにロードされたデータをリロードするのではなく、前のジョブが失敗した時点からロードを再開するようにします。

- スケールアップ — 一括ロードする前に、DB クラスターのライターインスタンスを最大サイズに設定します。この場合、DB クラスター内のリードレプリカインスタンスもスケールアップするか、データのロードが完了するまで削除する必要があります。

  一括ロードが完了したら、ライターインスタンスを再度スケールダウンしてください。

> **Important**
>
> 一括ロード中のレプリケーション遅延が原因で、リードレプリカが繰り返し再起動されるサイクルが発生した場合、レプリカは DB クラスター内のライターに追いつけない可能性があります。リーダーをライターよりも大きくするか、一括ロード中にリーダーを一時的に削除し、完了後に再作成してください。

ローダーリクエストパラメータ設定の詳細については、リクエストパラメータ を参照してください。

Neptune ローダーリファレンス

このセクションでは、Neptune DB インスタンスの HTTP エンドポイントから利用可能な Amazon Neptune の `Loader` API について説明します。

> **Note**
>
> エラー発生時にローダーから返されるエラーおよびフィードメッセージのリストについては、Neptune ローダーのエラーおよびフィードメッセージ を参照してください。

目次

    - 詳細なステータスのリクエストの例
  - Neptune Loader Get-Status errorLogs の例
    - エラー発生時の詳細なステータス応答の例
    - Data prefetch task interrupted エラーの例
- Neptune Loader Cancel Job
  - ジョブキャンセルリクエストの構文
  - ジョブキャンセルリクエストのパラメータ
  - ジョブキャンセルレスポンスの構文
  - ジョブキャンセルエラー
  - ジョブキャンセルエラーメッセージ
  - ジョブキャンセルの例

Neptune ローダーコマンド

Amazon S3 バケットから Neptune DB インスタンスにデータをロードします。

データをロードするには、POST エンドポイントに、HTTP `https://`*`your-neptune-endpoint`*`:`*`port`*`/loader` リクエストを送信する必要があります。`loader` リクエストのパラメータは、POST 本文、または URL エンコードされたパラメータとして送信できます。

> **Important**
>
> MIME タイプは、`application/json` である必要があります。

S3 バケットは、クラスターと同じ AWS リージョンに存在する必要があります。

> **Note**
>
> Amazon S3 SSE-S3 モードを使用して暗号化されている場合は、Amazon S3 から暗号化されたデータを読み込むことができます。その場合、Neptune はユーザーの認証情報を偽装し、ユーザーに代わって `s3:getObject` 呼び出しを発行することができます。
> また、IAM ロールに AWS KMSにアクセスするために必要なアクセス権限が含まれている限り、SSE-KMS モードを使用して暗号化された Amazon S3 から、暗号化されたデータをロードすることもできます。適切な AWS KMS アクセス許可がない場合、一括ロードオペレーションは失敗し、`LOAD_FAILED`レスポンスを返します。

Neptune は現在 SSE-C モードを使用して暗号化された Amazon S3 データのロードをサポートしていません。

別のジョブを開始する前に、1 つのロードジョブが完了するのを待つ必要はありません。Neptune は、queueRequest パラメータがすべて "TRUE" に設定されている場合、一度に最大 64 個のロードリクエストをキューに入れることができます。ジョブのキュー順序は first-in-first-out (FIFO) になります。一方、ロードジョブをキューに入れたくない場合は、queueRequest パラメータを "FALSE" (デフォルト) に設定して、別のジョブがすでに進行中の場合にロードジョブが失敗するようにできます。

dependencies パラメータを使用して、キュー内で指定した以前のジョブが正常に完了した場合にのみ実行するジョブをキューに入れることができます。その場合、指定したジョブのいずれかが失敗すると、ジョブは実行されず、ステータスは LOAD_FAILED_BECAUSE_DEPENDENCY_NOT_SATISFIED に設定されます。

Neptune ローダーリクエストの構文

```
{
  "source" : "string",
  "format" : "string",
  "iamRoleArn" : "string",
  "mode": "NEW|RESUME|AUTO",
  "region" : "us-east-1",
  "failOnError" : "string",
  "parallelism" : "string",
  "parserConfiguration" : {
    "baseUri" : "http://base-uri-string",
    "namedGraphUri" : "http://named-graph-string"
  },
  "updateSingleCardinalityProperties" : "string",
  "queueRequest" : "TRUE",
  "dependencies" : ["load_A_id", "load_B_id"]
}
```

Neptune ローダーのリクエストパラメータ

- **`source`** — Amazon S3 URI。

SOURCE パラメータは、単一のファイル、複数のファイル、1 つのフォルダ、または複数のフォルダを識別する Amazon S3 URI を受け入れます。Neptune は、指定されたフォルダ内のすべてのデータファイルをロードします。

URI は、以下の形式のいずれかになります。

- `s3://bucket_name/object-key-name`
- `https://s3.amazonaws.com/bucket_name/object-key-name`
- `https://s3.us-east-1.amazonaws.com/bucket_name/object-key-name`

URI の `object-key-name`要素は、Amazon S3 API コールのプレフィックスパラメータと同等です。 ListObjectsこれは、名前がそのプレフィックスで始まる指定された Amazon S3 バケット内のすべてのオブジェクトを識別します。これは、単一のファイルまたはフォルダ、または複数のファイルやフォルダにすることができます。

特定のフォルダには複数の頂点ファイルおよび複数のエッジファイルが含まれている場合があります。

例えば、 という名前の Amazon S3 バケットに次のフォルダ構造とファイルがある場合ですbucket-name。

```
s3://bucket-name/a/bc
s3://bucket-name/ab/c
s3://bucket-name/ade
s3://bucket-name/bcd
```

ソースパラメータが として指定されている場合s3://bucket-name/a、最初の 3 つのファイルがロードされます。

```
s3://bucket-name/a/bc
s3://bucket-name/ab/c
s3://bucket-name/ade
```

- **`format`** - データの形式。Neptune Loader コマンドのデータ形式の詳細については、Amazon Neptune 一括ローダーを使用したデータの取り込み を参照してください。

  許可される値

  - **`csv`** Gremlin CSV データ形式用。
  - **`opencypher`** openCypher CSV データ形式用。

- **ntriples** N-Triples RDF データ形式用。
- **nquads** N-Quads RDF データ形式用。
- **rdfxml** RDF\XML RDF データ形式用。
- **turtle** Turtle RDF データ形式用。

- **iamRoleArn** - S3 バケットにアクセスするために、Neptune DB インスタンスによって想定される IAM ロールの Amazon リソースネーム (ARN)。Amazon S3 にアクセスできるロールを作成し、Neptune クラスターに関連付ける方法の詳細については、前提条件: IAM ロールと Amazon S3 アクセス を参照してください。

  エンジンリリース 1.2.1.0.R3 以降、Neptune DB インスタンスと Amazon S3 バケットが異なる アカウントにある場合、複数の IAM ロールを連鎖させることもできます。 AWS この場合、Amazon Neptune で IAM ロールを連鎖する で説明されているように、`iamRoleArn` にはロール ARN のカンマ区切りのリストが含まれます。例:

```
curl -X POST https://localhost:8182/loader \
  -H 'Content-Type: application/json' \
  -d '{
        "source" : "s3://(the target bucket name)/(the target date file name)",
        "iamRoleArn" : "arn:aws:iam::(Account A
 ID):role/(RoleA),arn:aws:iam::(Account B ID):role/(RoleB),arn:aws:iam::(Account C
 ID):role/(RoleC)",
        "format" : "csv",
        "region" : "us-east-1"
      }'
```

- **region** – `region`パラメータは、クラスターの AWS リージョンと S3 バケットと一致する必要があります。

  Amazon Neptune は、次のリージョンで利用できます。

  - 米国東部 (バージニア北部): `us-east-1`
  - 米国東部 (オハイオ): `us-east-2`
  - 米国西部 (北カリフォルニア): `us-west-1`
  - 米国西部 (オレゴン): `us-west-2`
  - カナダ (中部): `ca-central-1`
  - 南米 (サンパウロ): `sa-east-1`
  - 欧州 (ストックホルム): `eu-north-1`

- 欧州 (アイルランド): eu-west-1
- 欧州 (ロンドン): eu-west-2
- 欧州 (パリ): eu-west-3
- 欧州 (フランクフルト): eu-central-1
- 中東 (バーレーン): me-south-1
- 中東 (アラブ首長国連邦): me-central-1
- イスラエル (テルアビブ): il-central-1
- アフリカ (ケープタウン): af-south-1
- アジアパシフィック (香港): ap-east-1
- アジアパシフィック (東京): ap-northeast-1
- アジアパシフィック (ソウル): ap-northeast-2
- アジアパシフィック (大阪): ap-northeast-3
- アジアパシフィック (シンガポール): ap-southeast-1
- アジアパシフィック (シドニー): ap-southeast-2
- アジアパシフィック (ムンバイ): ap-south-1
- 中国 (北京): cn-north-1
- 中国 (寧夏): cn-northwest-1
- AWS GovCloud （米国西部): us-gov-west-1
- AWS GovCloud （米国東部): us-gov-east-1

- **mode** - ロードジョブモード。

  使用できる値: RESUME、NEW、AUTO。

  デフォルト値: AUTO

  - RESUME - RESUME モードでは、ローダーはこのソースからの以前のロードを検索し、見つかった場合は、そのロードジョブを再開します。以前のロードジョブが見つからない場合、ローダーは停止します。

    ローダーは、以前のジョブで正常にロードされたファイルの再ロードを回避します。失敗したファイルの処理のみを試行します。以前にロードしたデータを Neptune クラスターから削除している場合、そのデータはこのモードでは再ロードされません。以前のロードジョブが同じソー

スからすべてのファイルを正常にロードした場合、何も再ロードされず、ローダーは成功を返します。

- NEW - NEW モードでは、以前のロードに関係なく、新しいロードリクエストが作成されます。このモードを使用して、以前にロードされたデータを Neptune クラスターから削除した後にソースからすべてのデータを再ロードしたり、同じソースから利用可能な新しいデータをロードしたりするために使用できます。
- AUTO - AUTO モードでは、ローダーは同じソースから以前のロードジョブを検索し、見つかった場合は、RESUME モードと同様にそのジョブを再開します。

  ローダーが同じソースから以前のロードジョブを見つけられない場合、NEW モードの場合と同様に、ソースからすべてのデータがロードされます。

- **failOnError** - エラー時に完全な停止を切り替えるフラグ。

  使用できる値: "TRUE"、"FALSE"。

  デフォルト値: "TRUE"。

  このパラメータを "FALSE" に設定すると、ローダーは指定された場所のすべてのデータをロードし、エラーのあるエントリはスキップします。

  このパラメータを "TRUE" に設定すると、ローダーはエラー発生時にすぐに停止します。その時点までロードされたデータは保持されます。

- **parallelism** - これは、一括ロードプロセスで使用されるスレッド数を減らすように設定することができるオプションのパラメータです。

  許可される値:

  - LOW - 使用されるスレッドの数は、コア数を 8 で割った値です。
  - MEDIUM - 使用されるスレッドの数は、コア数を 2 で割った値です。
  - HIGH - 使用されるスレッドの数は、コア数と同じです。
  - OVERSUBSCRIBE - 使用されるスレッドの数は、コア数に 2 をかけた値です。この値を使用する場合、バルクローダーは利用可能なすべてのリソースを消費します。

    ただし、これは、OVERSUBSCRIBE 設定すると、CPU 使用率が 100% になります。ロードオペレーションは I/O バウンドであるため、予想される CPU 最高使用率は 60% ~ 70% の範囲にあります。

  デフォルト値: HIGH

parallelism openCypher データをロードするときに、設定によってスレッド間でデッドロックが発生することがあります。こうなると、Neptune は LOAD_DATA_DEADLOCK というエラーを返します。通常、この問題は次のようにして修正できます。parallelism より低い設定にし、ロードコマンドを再試行します。

- **parserConfiguration** - 追加のパーサー設定値のあるオプションのオブジェクト。それぞれの子パラメータもオプションです。

名前	値の例	説明
namedGraphUri	*http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph*	グラフが指定されていない場合の、すべての RDF 形式のデフォルトのグラフ (四角形でない形式、およびグラフのない NQUAD エントリ)。デフォルトは です。http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph
baseUri	*http://aws.amazon.com/neptune/default*	RDF/XML および Turtle 形式の基本 URI。デフォルトは http://aws.amazon.com/neptune/default です。
allowEmptyStrings	*true*	Gremlin ユーザーは、CSV データをロードするときに、ノードおよびエッジプロパティとして空の文字列値 ("") を渡す必要があります。allowEmptyStrings が false (デフォルト) に設定されている場合、そのような空の文字列は NULL として扱われ、ロードされません。

	allowEmptyStrings が true に設定されている場合、ローダーは空の文字列を有効なプロパティ値として扱い、それに応じてロードします。

詳細については、「SPARQL デフォルトグラフと名前が付いたグラフ」を参照してください。

- **updateSingleCardinalityProperties** - これは、バルクローダーが単一濃度の頂点またはエッジプロパティの新しい値を処理する方法を制御するオプションのパラメータです。これは openCypher データのロードではサポートされていません (openCypher データをロードする を参照)。

  使用できる値: "TRUE"、"FALSE"。

  デフォルト値: "FALSE"。

  デフォルトでは、または updateSingleCardinalityProperties が明示的に "FALSE" に設定されている場合、ローダーは単一の濃度に違反するため、新しい値をエラーとして扱います。

  一方、updateSingleCardinalityProperties が "TRUE" に設定されている場合、バルクローダーは既存の値を新しい値に置き換えます。ロードされるソースファイルで複数のエッジまたは単一濃度の頂点プロパティ値が指定されている場合、バルクロードの終了時の最終値は、これらの新しい値のいずれかになります。ローダーは、既存の値が新しい値の 1 つに置き換えられたことを保証します。

- **queueRequest** - これは、ロードリクエストをキューに入れることができるかどうかを示すオプションのフラグパラメータです。

  queueRequest パラメータをすべて "TRUE"に設定している場合、Neptune は一度に最大 64 個のジョブをキューに入れることができるので、次のジョブを発行する前に 1 つのロードジョブが完了するのを待つ必要はありません。ジョブのキュー順序は first-in-first-out (FIFO) になります。

  queueRequest パラメータを省略するるか、"FALSE" に設定した場合、別のロードジョブがすでに実行されていると、ロードリクエストは失敗します。

  使用できる値: "TRUE"、"FALSE"。

デフォルト値: "FALSE"。

- **dependencies** - これは、キュー内の 1 つ以上の前のジョブが正常に完了することを条件に、キューに入れられたロードリクエストを作成することができるオプションのパラメータです。

  Neptune は、queueRequest パラメータが "TRUE" に設定されている場合、一度に最大 64 個のロードリクエストをキューに入れることができます。dependencies パラメータを使用すると、キュー内で指定された 1 つ以上前のリクエストが正常に完了したかどうかに応じて、キューに入れられたそのようなリクエストを実行できます。

  たとえば、ロード Job-A と Job-B が互いに独立しているものの、ロード Job-C を開始する前に Job-A および Job-B を終了する必要がある場合は、次の手順を実行します。

  1. 任意の順序で load-job-A と load-job-B を 1 つずつ送信し、load-id を保存します。
  2. dependencies フィールドで 2 つのジョブの load-id を付けて load-job-C を送信します。

  ```
  "dependencies" : ["job_A_load_id", "job_B_load_id"]
  ```

  dependencies パラメータのため、Job-A と Job-B が正常に完了するまで、バルクローダーは Job-C を起動しません。いずれかが失敗すると、Job-C は実行されず、そのステータスは LOAD_FAILED_BECAUSE_DEPENDENCY_NOT_SATISFIED に設定されます。

  この方法で複数のレベルの依存関係を設定できます。これにより、1 つのジョブが失敗すると、直接的または間接的に依存するすべてのリクエストがキャンセルされます。

- **userProvidedEdgeIds** — このパラメータは、リレーションシップ ID を含む openCypher データをロードする場合にのみ必要です。openCypher 関係 ID がロードデータに明示的に指定されている場合 (推奨)、必ず含められ True に設定されています。

  userProvidedEdgeIds がないか、True に設定されている場合、:ID 列は、ロード内のすべてのリレーションシップファイルにある必要があります。

  userProvidedEdgeIds があり、False に設定されている場合、ロード内のリレーションシップファイルに :ID 列があってはなりません。代わりに、Neptune ローダーは各リレーションシップの ID を自動的に生成します。

  CSV データのエラーが修正された後にローダーがロードを再開できるように、リレーションシップ ID を明示的に指定すると便利です。すでにロードされているリレーションシップをリロードする必要はありません。リレーションシップ ID が明示的に割り当てられていない場合、リレーショ

ンシップファイルを修正する必要があれば、ローダーは失敗したロードを再開できず、代わりにすべてのリレーションシップを再ロードする必要があります。

- `accessKey` – [非推奨] S3 バケットおよびデータファイルにアクセスするための IAM ロールのアクセスキー ID。

  代わりに、`iamRoleArn` パラメータを使用することをお勧めします。Amazon S3 にアクセスできるロールを作成し、Neptune クラスターに関連付ける方法の詳細については、前提条件: IAM ロールと Amazon S3 アクセス を参照してください。

  詳細については、「アクセスキー (アクセスキー ID とシークレットアクセスキー)」を参照してください。

- `secretKey` – [非推奨] 代わりに `iamRoleArn` パラメータを使用することをお勧めします。Amazon S3 にアクセスできるロールを作成し、Neptune クラスターに関連付ける方法の詳細については、前提条件: IAM ロールと Amazon S3 アクセス を参照してください。

  詳細については、「アクセスキー (アクセスキー ID とシークレットアクセスキー)」を参照してください。

openCypher データのロードに関する特別な考慮事項

- openCypher データを CSV 形式でロードする場合、形式パラメータを `opencypher` に設定する必要があります。
- すべての openCypher プロパティが単一のカーディナリティを持つため、`updateSingleCardinalityProperties` パラメータはopenCypher のロードではサポートされていません。openCypher ロードフォーマットは配列をサポートしておらず、ID 値が複数回表示される場合は、重複エラーまたは挿入エラーとして扱われます (下記参照)。
- Neptune ローダーは openCypher データで検出された重複を次のように処理します。
  - ローダーが同じノードIDを持つ複数の行を検出すると、次のルールを使用してマージされます。
    - 行のすべてのラベルがノードに追加されます。
    - 各プロパティには、プロパティ値が 1 つだけ読み込まれます。ロードする値の選択は決定的ではありません。
  - ローダーが同じリレーションシップ ID を持つ複数の行を検出すると、そのうちの 1 つだけがロードされます。ロードする行の選択は決定的ではありません。
  - ローダーは、既存のノードまたはリレーションシップの ID を持つロード· データに遭遇した場合、データベース内の既存のノードまたはリレーションシップのプロパティ値を更新することは

ありません。ただし、既存のノードやリレーションシップには存在しないノードラベルやプロパティがロードされます。

- リレーションシップに ID を割り当てる必要はありませんが、通常はお勧めします (上記の `userProvidedEdgeIds` パラメータを参照)。明示的なリレーションシップ ID がない場合、ローダーは、リレーションシップファイルにエラーが発生した場合にロードが失敗した場所からロードを再開するのではなく、すべてのリレーションシップをリロードする必要があります。

  また、ロードデータに明示的なリレーションシップ ID が含まれていない場合、ローダーは重複リレーションシップを検出することはありません。

以下に示しているのは openCypher ロードコマンドの例です。

```
curl -X POST https://your-neptune-endpoint:port/loader \
     -H 'Content-Type: application/json' \
     -d '
     {
       "source" : "s3://bucket-name/object-key-name",
       "format" : "opencypher",
       "userProvidedEdgeIds": "TRUE",
       "iamRoleArn" : "arn:aws:iam::account-id:role/role-name",
       "region" : "region",
       "failOnError" : "FALSE",
       "parallelism" : "MEDIUM",
     }'
```

ローダーの応答は通常と同じです。例:

```
{
  "status" : "200 OK",
  "payload" : {
    "loadId" : "guid_as_string"
  }
}
```

Neptune ローダーレスポンスの構文

```
{
    "status" : "200 OK",
    "payload" : {
```

```
        "loadId" : "guid_as_string"
    }
}
```

200 OK

ロードジョブが正常に開始されると 200 コードが返されます。

Neptune ローダーエラー

エラーが発生すると、JSON オブジェクトがレスポンスの BODY に返されます。message オブジェクトには、エラーの説明が含まれています。

エラーカテゴリ

- `Error 400` - 構文エラーは HTTP 400 無効なリクエストエラーを返します。エラーを説明するメッセージ。
- `Error 500` - 処理できない有効なリクエストは、HTTP 500 内部サーバーエラーを返します。エラーを説明するメッセージ。

以下は、ローダーからエラーの説明とともに返される可能性があるエラーメッセージです。

ローダーエラーメッセージ

- `Couldn't find the AWS credential for iam_role_arn` (HTTP 400)

  認証情報が見つかりませんでした。提供された認証情報を IAM コンソールまたは AWS CLI 出力と照合します。iamRoleArn で指定した IAM ロールをクラスターに追加したことを確認します。
- `S3 bucket not found for source` (HTTP 400)

  その S3 バケットは存在しません。バケットの名前を確認します。
- `The source source-uri does not exist/not reachable` (HTTP 400)

  S3 バケットに一致するファイルがありません。
- `Unable to connect to S3 endpoint. Provided source = source-uri and region = aws-region` (HTTP 500)

  Amazon S3 に接続できません。このリージョンはクラスターのリージョンと一致する必要があります。VPC エンドポイントがあることを確認します。VPC エンドポイント作成の詳細については、「Amazon S3 VPC エンドポイントの作成」を参照してください。

- `Bucket is not in provided Region (aws-region)` (HTTP 400)

  バケットは Neptune DB インスタンスと同じ AWS リージョンに存在する必要があります。

- `Unable to perform S3 list operation` (HTTP 400)

  提供された IAM ユーザーまたはロールに、バケットまたはフォルダに対する `List` アクセス許可がありません。バケットのポリシーまたはアクセスコントロールリスト (ACL) を確認します。

- `Start new load operation not permitted on a read replica instance` (HTTP 405)

  ロードは書き込み操作です。読み取り/書き込みクラスターエンドポイントの再試行。

- `Failed to start load because of unknown error from S3` (HTTP 500)

  Amazon S3 が不明なエラーを返しました。AWS Support に連絡する。

- `Invalid S3 access key` (HTTP 400)

  アクセスキーが無効です。提供された認証情報を確認します。

- `Invalid S3 secret key` (HTTP 400)

  シークレットキーが無効です。提供された認証情報を確認します。

- `Max concurrent load limit breached` (HTTP 400)

  ロードリクエストが `"queueRequest" : "TRUE"` なしで送信され、ロードジョブが現在実行中の場合、リクエストはこのエラーで失敗します。

- `Failed to start new load for the source "source name". Max load task queue size limit breached. Limit is 64` (HTTP 400)

  Neptune は、一度に 64 個のローダージョブのキューイングをサポートしています。すでにキューに 64 個のジョブが含まれている場合、追加のロードリクエストが送信されると、リクエストはこのメッセージで失敗します。

Neptune ローダーの例

Example リクエスト

以下は、`curl` コマンドを使用して HTTP POST 経由で送信されるリクエストです。Neptune CSV 形式のファイルをロードします。詳細については、「Gremlin ロードデータ形式」を参照してください。

```
curl -X POST \
    -H 'Content-Type: application/json' \
    https://your-neptune-endpoint:port/loader -d '
    {
      "source" : "s3://bucket-name/object-key-name",
      "format" : "csv",
      "iamRoleArn" : "ARN for the IAM role you are using",
      "region" : "region",
      "failOnError" : "FALSE",
      "parallelism" : "MEDIUM",
      "updateSingleCardinalityProperties" : "FALSE",
      "queueRequest" : "FALSE"
    }'
```

Example レスポンス

```
{
    "status" : "200 OK",
    "payload" : {
        "loadId" : "ef478d76-d9da-4d94-8ff1-08d9d4863aa5"
    }
}
```

Neptune Loader Get-Status API

`loader` ジョブのステータスを取得します。

ロードステータスを取得するには、`https://your-neptune-endpoint:port/loader` エンドポイントに、HTTP GET リクエストを送信する必要があります。特定のロードリクエストのステータスを取得するには、URL パラメータに `loadId` を含めるか、`loadId` を URL パスに追加する必要があります。

Neptune は、直近 1,024 個のバルクロードジョブのみを追跡し、ジョブごとに直近 10,000 個のエラーの詳細のみを保存します。

エラー発生時にローダーから返されるエラーおよびフィードメッセージのリストについては、Neptune ローダーのエラーおよびフィードメッセージ を参照してください。

目次

Neptune Loader Get-Status リクエスト

Loader Get-Status リクエストの構文

```
GET https://your-neptune-endpoint:port/loader?loadId=loadId
```

```
GET https://your-neptune-endpoint:port/loader/loadId
```

```
GET https://your-neptune-endpoint:port/loader
```

Neptune Loader Get-Status リクエストパラメータ

- **`loadId`** - ロードジョブの ID。`loadId` を指定しないと、ロード ID のリストが返されます。
- **`details`** - 全体的なステータスを超えた詳細を含めます。

  使用できる値: `TRUE`、`FALSE`。

  デフォルト値: `FALSE`。
- **`errors`** - エラーのリストを含めます。

使用できる値: TRUE、FALSE。

デフォルト値: FALSE。

エラーのリストはページ分割されます。page および errorsPerPage パラメータで、すべてのエラーをページ分割できます。

- **page** - エラーページ番号。errors パラメータが TRUE に設定されている場合にのみ有効です。

  使用できる値: 正の整数。

  デフォルト値: 1。

- **errorsPerPage** - 各ページあたりのエラーの数。errors パラメータが TRUE に設定されている場合にのみ有効です。

  使用できる値: 正の整数。

  デフォルト値: 10。

- **limit** - 一覧表示されるロード ID の数。loadId が指定されていない GET リクエストを送信して、ロード ID のリストをリクエストするときにのみ有効です。

  使用できる値: 1 ~ 100 の正の整数。

  デフォルト値: 100。

- **includeQueuedLoads** - ロード ID のリストがリクエストされたときに、キューに入れられたロードリクエストのロード ID を除外するために使用できるオプションのパラメータ。

  > **Note**
  >
  > このパラメータは Neptune エンジンリリース 1.0.3.0 からご利用いただけます。

  デフォルトでは、ステータスが LOAD_IN_QUEUE のすべてのロードジョブのロード ID がこのようなリストに含まれます。これらは、他のジョブのロード ID の前に表示され、キューに追加された時間順に最新のものから古いものにソートされます。

  使用できる値: TRUE、FALSE。

  デフォルト値: TRUE。

Neptune Loader Get-Status レスポンス

Neptune Loader Get-Status レスポンスのレイアウト

ローダーステータスレスポンスの一般的なレイアウトは次のとおりです。

```
{
    "status" : "200 OK",
    "payload" : {
        "feedCount" : [
            {
                "LOAD_FAILED" : number
            }
        ],
        "overallStatus" : {
            "fullUri" : "s3://bucket/key",
            "runNumber" : number,
            "retryNumber" : number,
            "status" : "string",
            "totalTimeSpent" : number,
            "startTime" : number,
            "totalRecords" : number,
            "totalDuplicates" : number,
            "parsingErrors" : number,
            "datatypeMismatchErrors" : number,
            "insertErrors" : number,
        },
        "failedFeeds" : [
            {
                "fullUri" : "s3://bucket/key",
                "runNumber" : number,
                "retryNumber" : number,
                "status" : "string",
                "totalTimeSpent" : number,
                "startTime" : number,
                "totalRecords" : number,
                "totalDuplicates" : number,
                "parsingErrors" : number,
                "datatypeMismatchErrors" : number,
                "insertErrors" : number,
            }
        ],
        "errors" : {
            "startIndex" : number,
```

```
            "endIndex" : number,
            "loadId" : "string,
            "errorLogs" : [ ]
        }
    }
}
```

Neptune Loader Get-Status **overallStatus** および **failedFeeds** レスポンスオブジェクト

エラーの説明とともに、ローダーからの失敗したフィードごとに返される可能性のある応答は、Get-Status レスポンスの overallStatus オブジェクトの場合と同じです。

これらのフィールドは、すべてのロードの overallStatus オブジェクト、および失敗した各フィードの failedFeeds オブジェクトに表示されます。

- **fullUri** - ロードされる 1 つ以上のファイルの URI。

  タイプ: 文字列

  形式: s3://*bucket*/*key*。
- **runNumber** - このロードまたはフィードの実行数。これは、ロードが再開されると増加します。

  タイプ: 符号なし long。
- **retryNumber** - このロードまたはフィードの再試行回数。これは、ローダーがフィードまたはロードを自動的に再試行するときに増分されます。

  タイプ: 符号なし long。
- **status** - ロードあるいはフィードの返されたステータス。LOAD_COMPLETED はロードが問題なく成功したことを示します。その他のロードステータスメッセージのリストについては、Neptune ローダーのエラーおよびフィードメッセージ を参照してください。

  タイプ: 文字列。
- **totalTimeSpent** - ロードまたはフィードのデータの解析や挿入に費やした時間 (秒単位)。これには、ソースファイルのリストを取得するのに費やされた時間は含まれません。

  タイプ: 符号なし long。
- **totalRecords** - ロードされた、またはロードしようとした全レコード。

  タイプ: 符号なし long。

CSV ファイルからロードする場合、レコード数はロードされた行数ではなく、その行に含まれる個々のレコードの数を指すことに注意してください。例えば、次のような小さな CSV ファイルを考えてみましょう。

```
~id,~label,name,team
'P-1','Player','Stokes','England'
```

Neptune は、このファイルには次の 3 つのレコードが含まれていると見なします。

```
P-1  label Player
P-1  name  Stokes
P-1  team  England
```

- **`totalDuplicates`** - 発生した重複レコードの数。

  タイプ: 符号なし long。

  `totalRecords` カウントの場合と同様に、この値には重複行の数ではなく、CSV ファイル内の個々の重複レコードの数が含まれます。例えば、次のような小さな CSV ファイルを考えてみましょう。

```
~id,~label,name,team
P-2,Player,Kohli,India
P-2,Player,Kohli,India
```

  ロード後に返されるステータスは以下のようになり、合計で 6 件のレコードが報告され、そのうちの 3 件は重複しています。

```
{
  "status": "200 OK",
  "payload": {
    "feedCount": [
      {
        "LOAD_COMPLETED": 1
      }
    ],
    "overallStatus": {
      "fullUri": "(the URI of the CSV file)",
      "runNumber": 1,
      "retryNumber": 0,
```

```
      "status": "LOAD_COMPLETED",
      "totalTimeSpent": 3,
      "startTime": 1662131463,
      "totalRecords": 6,
      "totalDuplicates": 3,
      "parsingErrors": 0,
      "datatypeMismatchErrors": 0,
      "insertErrors": 0
    }
  }
}
```

openCypher ロードの場合、次の場合に重複がカウントされます。

- ローダーは、ノードファイル内の行が、別の行または既存のノードに属する ID スペースのない別の ID 値と同じ ID スペースを持たないIDを持っていることを検出します。
- ローダーは、ノードファイル内の行が、別の行にあるか、既存のノードに属する ID スペースを持つ別の ID 値と同じ ID スペースを持つ ID を持っていることを検出します。

openCypher データのロードに関する特別な考慮事項 を参照してください。

- **`parsingErrors`** - 発生した解析エラーの数。

  タイプ: 符号なし long。

- **`datatypeMismatchErrors`** - 指定されたデータとデータ型が一致しないレコードの数。

  タイプ: 符号なし long。

- **`insertErrors`** - エラーにより挿入できなかったレコード数。

  タイプ: 符号なし long。

Neptune Loader Get-Status **`errors`** レスポンスオブジェクト

エラーは以下のカテゴリに分類されます。

- **`Error 400`** - `loadId` が無効の場合、HTTP `400` 無効な要求エラーが返されます。エラーを説明するメッセージ。
- **`Error 500`** - 処理できない有効なリクエストは、HTTP `500` 内部サーバーエラーを返します。エラーを説明するメッセージ。

エラー発生時にローダーから返されるエラーおよびフィードメッセージのリストについては、Neptune ローダーのエラーおよびフィードメッセージ を参照してください。

エラーが発生すると、JSON `errors` オブジェクトがレスポンスの BODY に次のフィールドと共に返されます。

- **`startIndex`** - 最初に含まれたエラーのインデックス。

  タイプ: 符号なし long。
- **`endIndex`** - 最後に含まれたエラーのインデックス。

  タイプ: 符号なし long。
- **`loadId`** - ロードの ID。この ID を使用すると、`errors` パラメータを TRUE に設定してロードのエラーを出力できます。

  タイプ: 文字列。
- **`errorLogs`** — エラーのリスト。

  タイプ: リスト。

Neptune Loader Get-Status **`errorLogs`** レスポンスオブジェクト

ローダー Get-Status レスポンスの `errors` にある `errorLogs` オブジェクトには、次のフィールドを使用して各エラーを説明するオブジェクトが含まれています。

- **`errorCode`** — エラーの性質を識別します。

  これには、次のいずれかの値を指定できます。
  - `PARSING_ERROR`
  - `S3_ACCESS_DENIED_ERROR`
  - `FROM_OR_TO_VERTEX_ARE_MISSING`
  - `ID_ASSIGNED_TO_MULTIPLE_EDGES`
  - `SINGLE_CARDINALITY_VIOLATION`
  - `FILE_MODIFICATION_OR_DELETION_ERROR`
  - `OUT_OF_MEMORY_ERROR`
  - `INTERNAL_ERROR` (一括ローダーがエラーのタイプを特定できない場合に返されます)。
- **`errorMessage`** - エラーを説明するメッセージ。

これは、エラーコードに関連付けられた一般的なメッセージ、または詳細を含む特定のメッセージ (たとえば、From/To 頂点の欠落、解析エラーなど) です。

- **`fileName`** — フィードの名前。
- **`recordNum`** — 解析エラーの場合、これは解析できなかったレコードのファイル内のレコード番号です。レコード番号がエラーに適用されない場合、または特定できなかった場合は 0 に設定されます。

たとえば、RDF nquads ファイルで次のような障害のある行に遭遇した場合、バルクローダは解析エラーを生成します。

```
<http://base#subject> |http://base#predicate> <http://base#true> .
```

ご覧のとおり、上の 2 番目の http 行の前には | ではなく < とがあります。ステータスレスポンスの errorLogs における結果のエラーオブジェクトは次のようになります。

```
{
    "errorCode" : "PARSING_ERROR",
    "errorMessage" : "Expected '<', found: |",
    "fileName" : "s3://bucket/key",
    "recordNum" : 12345
},
```

Neptune Loader Get-Status の例

ロードステータスのリクエスト例

以下は、curl コマンドを使用して HTTP GET 経由で送信されるリクエストです。

```
curl -X GET 'https://your-neptune-endpoint:port/loader/loadId (a UUID)'
```

Example レスポンス

```
{
    "status" : "200 OK",
    "payload" : {
        "feedCount" : [
            {
                "LOAD_FAILED" : 1
```

```
            }
        ],
        "overallStatus" : {
            "datatypeMismatchErrors" : 0,
            "fullUri" : "s3://bucket/key",
            "insertErrors" : 0,
            "parsingErrors" : 5,
            "retryNumber" : 0,
            "runNumber" : 1,
            "status" : "LOAD_FAILED",
            "totalDuplicates" : 0,
            "totalRecords" : 5,
            "totalTimeSpent" : 3.0
        }
    }
}
```

loadId のリクエストの例

以下は、curl コマンドを使用して HTTP GET 経由で送信されるリクエストです。

```
curl -X GET 'https://your-neptune-endpoint:port/loader?limit=3'
```

Example レスポンス

```
{
    "status" : "200 OK",
    "payload" : {
         "loadIds" : [
            "a2c0ce44-a44b-4517-8cd4-1dc144a8e5b5",
            "09683a01-6f37-4774-bb1b-5620d87f1931",
            "58085eb8-ceb4-4029-a3dc-3840969826b9"
        ]
    }
}
```

詳細なステータスのリクエストの例

以下は、curl コマンドを使用して HTTP GET 経由で送信されるリクエストです。

```
curl -X GET 'https://your-neptune-endpoint:port/loader/loadId (a UUID)?details=true'
```

Example レスポンス


```
{
    "status" : "200 OK",
    "payload" : {
        "failedFeeds" : [
            {
                "datatypeMismatchErrors" : 0,
                "fullUri" : "s3://bucket/key",
                "insertErrors" : 0,
                "parsingErrors" : 5,
                "retryNumber" : 0,
                "runNumber" : 1,
                "status" : "LOAD_FAILED",
                "totalDuplicates" : 0,
                "totalRecords" : 5,
                "totalTimeSpent" : 3.0
            }
        ],
        "feedCount" : [
            {
                "LOAD_FAILED" : 1
            }
        ],
        "overallStatus" : {
            "datatypeMismatchErrors" : 0,
            "fullUri" : "s3://bucket/key",
            "insertErrors" : 0,
            "parsingErrors" : 5,
            "retryNumber" : 0,
            "runNumber" : 1,
            "status" : "LOAD_FAILED",
            "totalDuplicates" : 0,
            "totalRecords" : 5,
            "totalTimeSpent" : 3.0
        }
    }
}
```

Neptune Loader Get-Status **errorLogs** の例

エラー発生時の詳細なステータス応答の例

これは curl を使用し HTTP GET 経由で送信されるリクエストです。

```
curl -X GET 'https://your-neptune-endpoint:port/loader/0a237328-afd5-4574-a0bc-c29ce5f54802?details=true&errors=true&page=1&errorsPerPage=3'
```

Example エラー発生時の詳細なレスポンス

これは、発生したロードエラーを一覧表示するオブジェクト errorLogs とともに、上記のクエリから得られる可能性のあるレスポンスの例です。

```
{
    "status" : "200 OK",
    "payload" : {
        "failedFeeds" : [
            {
                "datatypeMismatchErrors" : 0,
                "fullUri" : "s3://bucket/key",
                "insertErrors" : 0,
                "parsingErrors" : 5,
                "retryNumber" : 0,
                "runNumber" : 1,
                "status" : "LOAD_FAILED",
                "totalDuplicates" : 0,
                "totalRecords" : 5,
                "totalTimeSpent" : 3.0
            }
        ],
        "feedCount" : [
            {
                "LOAD_FAILED" : 1
            }
        ],
        "overallStatus" : {
            "datatypeMismatchErrors" : 0,
            "fullUri" : "s3://bucket/key",
            "insertErrors" : 0,
            "parsingErrors" : 5,
            "retryNumber" : 0,
            "runNumber" : 1,
```

```
            "status" : "LOAD_FAILED",
            "totalDuplicates" : 0,
            "totalRecords" : 5,
            "totalTimeSpent" : 3.0
        },
        "errors" : {
            "endIndex" : 3,
            "errorLogs" : [
                {
                    "errorCode" : "PARSING_ERROR",
                    "errorMessage" : "Expected '<', found: |",
                    "fileName" : "s3://bucket/key",
                    "recordNum" : 1
                },
                {
                    "errorCode" : "PARSING_ERROR",
                    "errorMessage" : "Expected '<', found: |",
                    "fileName" : "s3://bucket/key",
                    "recordNum" : 2
                },
                {
                    "errorCode" : "PARSING_ERROR",
                    "errorMessage" : "Expected '<', found: |",
                    "fileName" : "s3://bucket/key",
                    "recordNum" : 3
                }
            ],
            "loadId" : "0a237328-afd5-4574-a0bc-c29ce5f54802",
            "startIndex" : 1
        }
    }
}
```

Data prefetch task interrupted エラーの例

LOAD_FAILED ステータスが発生し、その後より詳細な情報をリクエストした場合、次のように PARSING_ERROR エラーが Data prefetch task interrupted メッセージとともに返されることがあります。

```
"errorLogs" : [
    {
        "errorCode" : "PARSING_ERROR",
```

```
        "errorMessage" : "Data prefetch task interrupted: Data prefetch task for 11467
 failed",
        "fileName" : "s3://some-source-bucket/some-source-file",
        "recordNum" : 0
    }
]
```

このエラーが発生するのは、一般的にリクエストやデータが原因で発生しない、データロードプロセスの一時的な中断があった場合です。これは通常、単にバルクアップロードリクエストを再実行することで解決できます。デフォルト設定 ("mode":"AUTO" および "failOnError":"TRUE") を使用している場合、ローダーはすでに正常にロードされたファイルをスキップし、中断が発生したときにまだロードされていなかったファイルのロードを再開します。

Neptune Loader Cancel Job

ロードジョブをキャンセルします。

ジョブをキャンセルするには、DELETE エンドポイントに、HTTP https://*your-neptune-endpoint*:*port*/loader リクエストを送信する必要があります。loadId は /loader URL パスに追加することも、URL に変数として含めることもできます。

ジョブキャンセルリクエストの構文

```
DELETE https://your-neptune-endpoint:port/loader?loadId=loadId
```

```
DELETE https://your-neptune-endpoint:port/loader/loadId
```

ジョブキャンセルリクエストのパラメータ

loadId

ロードジョブの ID。

ジョブキャンセルレスポンスの構文

```
no response body
```

200 OK

ロードジョブが正常に削除されると 200 コードが返されます。

ジョブキャンセルエラー

エラーが発生すると、JSON オブジェクトがレスポンスの BODY に返されます。message オブジェクトには、エラーの説明が含まれています。

エラーカテゴリ

- **`Error 400`** - loadId が無効の場合、HTTP 400 無効な要求エラーが返されます。エラーを説明するメッセージ。
- **`Error 500`** - 処理できない有効なリクエストは、HTTP 500 内部サーバーエラーを返します。エラーを説明するメッセージ。

ジョブキャンセルエラーメッセージ

以下は、キャンセル API からエラーの説明とともに返される可能性があるエラーメッセージです。

- `The load with id = load_id does not exist or not active` (HTTP 404) - ロードが見つかりませんでした。id パラメータの値を確認します。
- `Load cancellation is not permitted on a read replica instance.` (HTTP 405) - ロードは書き込み操作です。読み取り/書き込みクラスターエンドポイントの再試行。

ジョブキャンセルの例

Example リクエスト

以下は、curl コマンドを使用して HTTP DELETE 経由で送信されるリクエストです。

```
curl -X DELETE 'https://your-neptune-endpoint:port/loader/0a237328-afd5-4574-a0bc-c29ce5f54802'
```

AWS Database Migration Service を使用して別のデータストアから Amazon Neptune にデータをロードする

AWS Database Migration Service（ AWS DMS) は、サポートされているソースデータベースから Neptune にデータを迅速かつ安全にロードできます。移行中でもソースデータベースは完全に利用可能な状態に保たれ、それを利用するアプリケーションのダウンタイムを最小限に抑えられます。

の詳細については、「AWS Database Migration Service ユーザーガイド」および AWS DMS AWS Database Migration Service 「API リファレンス」を参照してください。特に、Neptune クラスターを移行のターゲットとして設定する方法については、AWS Database Migration Serviceのターゲットとしての Amazon Neptune の使用」を参照してください。

AWS DMSを使用して Neptune にデータをインポートするための前提条件は次のとおりです。

- ソースデータベースからデータを抽出する方法を定義するには、 AWS DMS テーブルマッピングオブジェクトを作成する必要があります (詳細については、 AWS DMS ユーザーガイドの「JSON を使用したテーブルマッピングによるテーブル選択と変換の指定」を参照してください)。このテーブルマッピング設定オブジェクトは、読み取る必要があるテーブル、その順序、および列の命名方法を指定します。また、コピーされる行をフィルタリングし、小文字への変換や四捨五入などの単純な値の変換を提供することもできます。
- ソースデータベースから抽出されたデータを Neptune にロードする方法を指定するには、Neptune `GraphMappingConfig` を作成する必要があります。RDF データ (SPARQL を使用してクエリされる) の場合、`GraphMappingConfig` は W3 の標準 R2RML マッピング言語で記述されます。プロパティグラフデータ (Gremlin を使用してクエリされる) の場合、`GraphMappingConfig` は JSON オブジェクトです (GraphMappingConfig Property-Graph/Gremlin データのレイアウト を参照)。
- を使用して AWS DMS 、Neptune DB クラスターと同じ VPC にレプリケーションインスタンスを作成し、データ転送を仲介する必要があります。
- また、移行データをステージングするための中間ストレージとして使用する Amazon S3 バケットも必要です。

Neptune の作成 GraphMappingConfig

作成した `GraphMappingConfig` は、ソースデータストアから抽出されたデータを Neptune DB クラスターにロードする方法を指定します。形式は、目的が RDF データのロードか、プロパティグラフデータのロードかによって異なります。

RDF データの場合、W3 R2RML 言語を使用してリレーショナルデータを RDF にマッピングできます。

Gremlin を使用してクエリするプロパティグラフデータをロードする場合、`GraphMappingConfig` の JSON オブジェクトを作成します。

GraphMappingConfig RDF/SPARQL データのレイアウト

SPARQL を使用して照会する RDF データをロードする場合は、R2RML に GraphMappingConfig を書き込みます。R2RML はリレーショナルデータを RDF にマッピングするための標準 W3 言語です。1 つの例を次に示します。

```
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix ex: <http://example.com/ns#> .

<#TriplesMap1>
    rr:logicalTable [ rr:tableName "nodes" ];
    rr:subjectMap [
        rr:template "http://data.example.com/employee/{id}";
        rr:class ex:Employee;
    ];
    rr:predicateObjectMap [
        rr:predicate ex:name;
        rr:objectMap [ rr:column "label" ];
    ] .
```

次に、別の例を示します。

```
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix ex: <http://example.com/#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<#TriplesMap2>
    rr:logicalTable [ rr:tableName "Student" ];
    rr:subjectMap   [ rr:template "http://example.com/{ID}{Name}";
                      rr:class foaf:Person ];
    rr:predicateObjectMap [
        rr:predicate ex:id ;
        rr:objectMap  [ rr:column "ID";
                        rr:datatype xsd:integer ]
    ];
    rr:predicateObjectMap [
        rr:predicate foaf:name ;
        rr:objectMap  [ rr:column "Name" ]
    ] .
```

「R2RML: RDB to RDF Mapping Language」にある「W3 Recommendation」には、言語の詳細が記載されています。

GraphMappingConfig Property-Graph/Gremlin データのレイアウト

プロパティグラフデータの同等の GraphMappingConfig は、ソースデータから生成される各グラフエンティティのマッピングルールを提供する JSON オブジェクトです。次のテンプレートは、このオブジェクトの各ルールがどのようになるかを示しています。

```
{
  "rules": [
    {
      "rule_id": "(an identifier for this rule)",
      "rule_name": "(a name for this rule)",
      "table_name": "(the name of the table or view being loaded)",
      "vertex_definitions": [
        {
          "vertex_id_template": "{col1}",
          "vertex_label": "(the vertex to create)",
          "vertex_definition_id": "(an identifier for this vertex)",
          "vertex_properties": [
            {
              "property_name": "(name of the property)",
              "property_value_template": "{col2} or text",
              "property_value_type": "(data type of the property)"
            }
          ]
        }
      ]
    },
    {
      "rule_id": "(an identifier for this rule)",
      "rule_name": "(a name for this rule)",
      "table_name": "(the name of the table or view being loaded)",
      "edge_definitions": [
        {
          "from_vertex": {
            "vertex_id_template": "{col1}",
            "vertex_definition_id": "(an identifier for the vertex referenced above)"
          },
          "to_vertex": {
            "vertex_id_template": "{col3}",
            "vertex_definition_id": "(an identifier for the vertex referenced above)"
```

```
          },
          "edge_id_template": {
            "label": "(the edge label to add)",
            "template": "{col1}_{col3}"
          },
          "edge_properties":[
            {
              "property_name": "(the property to add)",
              "property_value_template": "{col4} or text",
              "property_value_type": "(data type like String, int, double)"
            }
          ]
        }
      ]
    }
  ]
}
```

頂点ラベルが存在する場合、頂点がここで作成されることを意味しますが、ない場合、頂点が別のソースによって作成されることを意味します。この定義は、頂点プロパティの追加のみです。

従業員レコードのサンプルルールを次に示します。

```
{
  "rules": [
    {
      "rule_id": "1",
      "rule_name": "vertex_mapping_rule_from_nodes",
      "table_name": "nodes",
      "vertex_definitions": [
        {
          "vertex_id_template": "{emp_id}",
          "vertex_label": "employee",
          "vertex_definition_id": "1",
          "vertex_properties": [
            {
              "property_name": "name",
              "property_value_template": "{emp_name}",
              "property_value_type": "String"
            }
          ]
        }
      ]
```

```
    },
    {
      "rule_id": "2",
      "rule_name": "edge_mapping_rule_from_emp",
      "table_name": "nodes",
      "edge_definitions": [
        {
          "from_vertex": {
            "vertex_id_template": "{emp_id}",
            "vertex_definition_id": "1"
          },
          "to_vertex": {
            "vertex_id_template": "{mgr_id}",
            "vertex_definition_id": "1"
          },
          "edge_id_template": {
            "label": "reportsTo",
            "template": "{emp_id}_{mgr_id}"
          },
          "edge_properties":[
            {
              "property_name": "team",
              "property_value_template": "{team}",
              "property_value_type": "String"
            }
          ]
        }
      ]
    }
  ]
}
```

Neptune をターゲットとする AWS DMS レプリケーションタスクの作成

テーブルマッピングおよびグラフマッピング設定を作成したら、次のプロセスを使用してソースストアから Neptune にデータをロードします。問題の APIs の詳細については、 AWS DMS ドキュメントを参照してください。

ステップ 1: AWS DMS レプリケーションインスタンスを作成する

Neptune DB クラスターが実行されている VPC にレ AWS DMS プリケーションインスタンスを作成します (「 AWS DMS ユーザーガイド」の AWS 「DMS レプリケーションインスタン

スとCreateReplicationインスタンスの使用」を参照してください)。そのためには、次のような AWS CLI コマンドを使用できます。

```
aws dms create-replication-instance \
    --replication-instance-identifier (the replication instance identifier) \
    --replication-instance-class (the size and capacity of the instance, like
 'dms.t2.medium') \
    --allocated-storage (the number of gigabytes to allocate for the instance
 initially) \
    --engine-version (the DMS engine version that the instance should use) \
    --vpc-security-group-ids (the security group to be used with the instance)
```

Step 2. ソースデータベースの AWS DMS エンドポイントを作成する

次のステップでは、ソースデータストアの AWS DMS エンドポイントを作成します。API は AWS DMS CreateEndpoint、 AWS CLI 次のような で使用できます。

```
aws dms create-endpoint \
    --endpoint-identifier (source endpoint identifier) \
    --endpoint-type source \
    --engine-name (name of source database engine) \
    --username (user name for database login) \
    --password (password for login) \
    --server-name (name of the server) \
    --port (port number) \
    --database-name (database name)
```

ステップ 3。Neptune がステージングデータに使用する Amazon S3 バケットをセットアップする

ステージングデータに使用できる Amazon S3 バケットがない場合は、Amazon S3 入門ガイドのバケットの作成、またはコンソールユーザーガイドのS3 バケットを作成する方法の説明に従ってバケットを作成します。

バケットに GetObject、PutObject、DeleteObject、および、ListObject アクセス許可を付与する IAM ポリシーを作成する必要があります (まだない場合)。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
      "Sid": "ListObjectsInBucket",
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::(bucket-name)"
      ]
    },
    {
      "Sid": "AllObjectActions",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject",
        "s3:ListObject"
       ],
      "Resource": [
        "arn:aws:s3:::(bucket-name)/*"
      ]
    }
  ]
}
```

Neptune DB クラスターで IAM 認証が有効になっている場合、次のポリシーも含める必要があります。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": "neptune-db:*",
      "Resource": "(the ARN of your Neptune DB cluster resource)"
    }
  ]
}
```

ポリシーをアタッチする信頼ドキュメントとして IAM ロールを作成します。

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "dms.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    },
    {
      "Sid": "neptune",
      "Effect": "Allow",
      "Principal": {
        "Service": "rds.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

ポリシーをロールにアタッチした後、ロールを Neptune DB クラスターにアタッチします。これにより、 はロードされるデータをステージングするために バケット AWS DMS を使用できます。

ステップ 4。Neptune VPC で Amazon S3 エンドポイントを作成する

ここでは、Neptune クラスターが配置されている VPC に、中間 Amazon S3 バケットの VPC ゲートウェイエンドポイントを作成します。 AWS Management Console または を使用して、 AWS CLI 「ゲートウェイエンドポイントの作成」で説明されているように、これを行うことができます。

Step 5. Neptune の AWS DMS ターゲットエンドポイントを作成する

ターゲット Neptune DB クラスターの AWS DMS エンドポイントを作成します。 AWS DMS CreateEndpoint API は、次のような `NeptuneSettings`パラメータで使用できます。

```
aws dms create-endpoint \
    --endpoint-identifier (target endpoint identifier) \
    --endpoint-type target \
    --engine-name neptune \
    --server-name (name of the server) \
    --port (port number) \
    --neptune-settings '{ \
      "ServiceAccessRoleArn": "(ARN of the service access role)", \
```

```
      "S3BucketName": "(name of S3 bucket to use for staging files when migrating)", \
      "S3BucketFolder": "(name of the folder to use in that S3 bucket)", \
      "ErrorRetryDuration": (number of milliseconds to wait between bulk-load retries),
\
      "MaxRetryCount": (the maximum number of times to retry a failing bulk-load job),
\
      "MaxFileSize": (maximum file size, in bytes, of the staging files written to S3),
\
      "isIamAuthEnabled": (set to true if IAM authentication is enabled on the Neptune
cluster) }'
```

NeptuneSettings パラメータで AWS DMS CreateEndpoint API に渡される JSON オブジェクトには、次のフィールドがあります。

- **ServiceAccessRoleArn** – (必須) Neptune へのデータの移行をステージングするために使用される S3 バケットへのきめ細かなアクセスを許可する IIAM ロールの ARN。IAM 認証が有効になっている場合、このロールには Neptune DB クラスターにアクセスするアクセス許可も必要です。
- **S3BucketName** – (必須)フルロード移行の場合、レプリケーションインスタンスはすべての RDS データを CSV、クワッドファイルに変換し、S3 のこのステージングバケットにアップロードし、それらを Neptune に一括ロードします。
- **S3BucketFolder** – (必須) S3 ステージングバケットで使用するフォルダー。
- **ErrorRetryDuration** – (オプション) Neptune リクエストが失敗してから再試行リクエストを行うまでに待機するミリ秒数。デフォルトは 250 です。
- **MaxRetryCount** – (オプション) 再試行可能な失敗後に AWS DMS 行う再試行リクエストの最大数。デフォルトは 5 です。
- **MaxFileSize** – (オプション) 移行中に S3 に保存された各ステージングファイルの最大サイズ (バイト単位)。デフォルトは 1,048,576 KB (1 GB) です。
- **IsIAMAuthEnabled** – (オプション) IAM 認証が Neptune DB クラスターで有効になっている場合 `true` に、そうでない場合 `false` に設定します。デフォルトは `false` です。

ステップ 6。新しいエンドポイントへの接続をテストする

次のように API を使用して、これらの新しい各エンドポイントへの接続を AWS DMS TestConnectionテストできます。

```
aws dms test-connection \
```

```
    --replication-instance-arn (the ARN of the replication instance) \
    --endpoint-arn (the ARN of the endpoint you are testing)
```

ステップ 7。 AWS DMS レプリケーションタスクを作成する

前のステップを正常に完了したら、次のように Task API を使用して、ソースデータストアから Neptune にデータを移行するための AWS DMS レプリケーションCreateReplicationタスクを作成します。

```
aws dms create-replication-task \
    --replication-task-identifier (name for the replication task) \
    --source-endpoint-arn (ARN of the source endpoint) \
    --target-endpoint-arn (ARN of the target endpoint) \
    --replication-instance-arn (ARN of the replication instance) \
    --migration-type full-load \
    --table-mappings (table-mapping JSON object or URI like 'file:///tmp/table-
mappings,json') \
    --task-data (a GraphMappingConfig object or URI like 'file:///tmp/graph-mapping-
config.json')
```

TaskData パラメータは、コピーされたデータを Neptune に保存する方法を指定するGraphMapping設定 を提供します。

ステップ 8: AWS DMS レプリケーションタスクを開始する

ここでは、レプリケーションタスクを開始できます。

```
aws dms start-replication-task
    --replication-task-arn (ARN of the replication task started in the previous step)
    --start-replication-task-type start-replication
```

Neptune Graph のクエリ

Neptune では、グラフにアクセスするための次のグラフクエリ言語がサポートされています。

- プロパティグラフの作成とクエリのために Apache TinkerPopによって定義される Gremlin 。

  Gremlin のクエリは個別のステップで構成されたトラバーサルで、各ステップはエッジからノードに従います。

  Neptune での Gremlin の使用については Gremlin を使用した Neptune グラフへのアクセス を参照し、Amazon Neptune の Gremlin 標準への準拠 Gremlin の Neptune 実装に関する具体的な詳細をご覧ください。
- openCypher は、プロパティグラフの宣言型クエリ言語です。当初は Neo4j が開発し、その後 2015 年にオープンソース化され、Apache 2 オープンソースライセンスの下で opencyPher プロジェクトで活用されました。その構文は openCypher 仕様書に記載されています。
- SPARQL は、RDF データクエリ用のグラフパターンマッチングに基づく宣言型言語です。これは、ワールド・ワイド・ウェブ・コンソーシアムが対応しています。

  Neptune で SPARQL を使用する方法について、SPARQL を使用した Neptune グラフへのアクセス を参照してください。また、SPARQL の Neptune 実装に関する具体的な詳細については Amazon Neptune の SPARQL 標準準拠 ご覧ください。

> **Note**
>
> Gremlin と openCypher はどちらも、ロード方法に関係なく、Neptune に保存されているプロパティグラフデータのクエリに使用できます。

トピック

- Amazon Neptune でのクエリキューイング
- Gremlin を使用した Neptune グラフへのアクセス
- openCypher でNeptune グラフにアクセスする
- SPARQL を使用した Neptune グラフへのアクセス

Amazon Neptune でのクエリキューイング

グラフアプリケーションを開発およびチューニングするときは、データベースによってクエリがキューに入れられる方法の意味を知っておくと便利です。Amazon Neptune では、クエリキューイングは次のように実行されます。

- インスタンスのサイズに関係なく、インスタンスごとにキューに入れることができるクエリの最大数は 8,192 です。その数を超えるクエリは拒否され、`ThrottlingException` で失敗します。
- 一度に実行できるクエリの最大数は、割り当てられたワーカースレッドの数によって決まります。通常、使用可能な仮想 CPU コア (vCPU) の数の 2 倍に設定されます。
- クエリのレイテンシーには、クエリがキューに費やす時間、ネットワークのラウンドトリップ、および実際に実行に要する時間が含まれます。

特定の瞬間にキューにあるクエリの数を調べる

`MainRequestQueuePendingRequests` CloudWatch メトリクスは、入力キューで待機しているリクエストの数を 5 分単位で記録します (「」を参照Neptune CloudWatch メトリクス)。

Gremlin の場合、`acceptedQueryCount` により返された Gremlin クエリステータス API 値を使用して、キュー内の現在のクエリ数を取得できます。ただし、`acceptedQueryCount` により返される SPARQL クエリステータス API 値には、完了したクエリを含め、サーバーの起動後に受け入れられたすべてのクエリが含まれます。

クエリキューイングがタイムアウトに与える影響

前述のように、クエリのレイテンシーには、クエリがキューで費やす時間と、実行に要する時間が含まれます。

通常、クエリのタイムアウト期間はキューに入った時点から測定されるため、キューの動きが遅い場合、キューから出されると多くのクエリがすぐにタイムアウトになる可能性があります。これは明らかに望ましくないため、迅速に実行できる場合を除き、多数のクエリをキューに入れることは避けてください。

Gremlin を使用した Neptune グラフへのアクセス

Amazon Neptune は Apache TinkerPop3 および Gremlin と互換性があります。つまり、Neptune DB インスタンスに接続し、Gremlin トラバーサル言語を使用してグラフをクエリできます (Apache

TinkerPop3 ドキュメントの「グラフ」を参照) 。Gremlin の Neptune 実装の相違点については、Gremlin の標準コンプライアンスを参照してください。

Neptune エンジンのバージョンによって、サポートされる Gremlin のバージョンは異なります。実行している Neptune バージョンのエンジンのリリースページをチェックし、サポートしている Gremlin リリースを特定します。

Gremlin のトラバーサルは、一連の連鎖ステップです。頂点 (またはエッジ) で始まります。各頂点から出ていくエッジに沿って、さらに、これらの頂点から出ていくエッジをたどってグラフを描きます。各ステップはトラバーサルの操作です。詳細については、 TinkerPop3 ドキュメントの「トラバーサル」を参照してください。

さまざまなプログラミング言語による Gremlin 言語バリアントおよび Gremlin アクセスのサポートがあります。詳細については、 TinkerPop3 ドキュメントの「On Gremlin Language Variants」を参照してください。

このドキュメントでは、以下のバリアントやプログラミング言語を使用して Neptune にアクセスする方法について説明します。

転送中の暗号化: SSL/HTTPS を使用して Neptune に接続する で説明されているように、すべての AWS リージョンで Neptune に接続するときには、Transport Layer Security/Secure Sockets Layer (TLS/SSL) を使用する必要があります。

Gremlin-Groovy

このセクションの Gremlin コンソールと HTTP REST サンプルは、Gremlin-Groovy バリアントを使用します。Gremlin コンソールと Amazon Neptune の詳細については、クイックスタートのthe section called “Gremlin を使用する”セクションを参照してください。

Gremlin-Java

Java サンプルは、公式の TinkerPop3 Java 実装で記述され、Gremlin-Java バリアントを使用します。

Gremlin-Python

Python サンプルは、公式の TinkerPop3 Python 実装で記述され、Gremlin-Python バリアントを使用します。

以下のセクションでは、Gremlin コンソール、RESTHTTPS を介した およびさまざまなプログラミング言語を使用して Neptune DB インスタンスに接続する方法について説明します。

始めるには以下のものが必要です。

- Neptune DB インスタンス。Neptune DB インスタンスの作成については、新しい Neptune DB クラスターの作成 を参照してください。
- Neptune DB インスタンスと同じ Virtual Private Cloud (VPC) にある Amazon EC2; インスタンス。

前提条件、ロード形式、およびロードパラメータを含む Neptune へのデータのロードの詳細については、Amazon Neptune にデータをロードするを参照してください。

トピック

- Gremlin コンソールをセットアップして Neptune DB インスタンスに接続する
- HTTPS REST エンドポイントを使用して Neptune DB インスタンスに接続する
- Amazon Neptune で使用する Java ベースの Gremlin クライアント
- Python を使用して Neptune DB インスタンスに接続する
- .NET を使用して Neptune DB インスタンスに接続する
- Node.js を使用して Neptune DB インスタンスに接続する
- Go を使用して Neptune DB インスタンスに接続する
- Gremlin クエリヒント
- Gremlin クエリステータス API
- Gremlin クエリのキャンセル
- Gremlin スクリプトベースのセッションのサポート
- Neptune での Gremlin トランザクション
- Amazon Neptune で Gremlin API を使用する
- Amazon Neptune Gremlin でクエリ結果をキャッシュする
- Gremlin mergeV() および mergeE() ステップによる効率的なアップサートの実行
- fold()/coalesce()/unfold() による効率的な Gremlin アップサートの実行
- Gremlin を使用して Neptune クエリ実行を分析するexplain
- Gremlin と Neptune DFE クエリエンジンを使用する

Gremlin コンソールをセットアップして Neptune DB インスタンスに接続する

Gremlin コンソールでは、REPL (read-eval-print ループ) 環境で TinkerPop グラフとクエリを試すことができます。

Gremlin コンソールをインストールし、通常の方法で接続する

Gremlin Console を使用して、リモートグラフデータベースに接続できます。次のセクションでは、Neptune DB インスタンスにリモートで接続するための Gremlin Console のインストールと設定について説明します。Neptune DB インスタンスと同じ仮想プライベートクラウド (VPC) の Amazon EC2 インスタンスからこれらの手順を実行してください。

SSL/TLS (必須) で Neptune に接続する方法については、「SSL/TLS 設定」を参照してください。

> **Note**
>
> IAM 認証の有効化を Neptune DB クラスターで行った場合、ここでの手順ではなく 署名バージョン 4 で署名された Gremlin コンソールを使用して Neptune に接続する の手順に従い、Gremlin Console をインストールします。

Gremlin Console をインストールして Neptune に接続するには

1. Gremlin Console バイナリには Java 8 または Java 11 が必要です。これらの手順は Java 11 の使用を前提としています。次のように EC2 インスタンスに Java 11 をインストールできます。

   - Amazon Linux 2 (AL2) を使用している場合:

     ```
     sudo amazon-linux-extras install java-openjdk11
     ```

   - Amazon Linux 2023 (AL2023) を使用している場合:

     ```
     sudo yum install java-11-amazon-corretto-devel
     ```

   - 他のディストリビューションでは、以下のうち適切なものを使用してください。

     ```
     sudo yum install java-11-openjdk-devel
     ```

     または

```
sudo apt-get install openjdk-11-jdk
```

2. 次のように入力して、EC2 インスタンスで Java 11 をデフォルトのランタイムとして設定します。

```
sudo /usr/sbin/alternatives --config java
```

プロンプトが表示されたら、Java 11 の数を入力します。

3. Apache ウェブサイトから該当するバージョンの Gremlin コンソールをダウンロードします。実行している Neptune バージョンのエンジンのリリースページをチェックし、サポートしている Gremlin バージョンを特定できます。例えば、バージョン 3.6.5 の場合、次のように、Apache TinkerPop3 Web サイトから Gremlin コンソールを EC2 インスタンスにダウンロードできます。

```
wget https://archive.apache.org/dist/tinkerpop/3.6.5/apache-tinkerpop-gremlin-
console-3.6.5-bin.zip
```

4. Gremlin Console zip ファイルを解凍します。

```
unzip apache-tinkerpop-gremlin-console-3.6.5-bin.zip
```

5. ディレクトリを解凍ディレクトリに変更します。

```
cd apache-tinkerpop-gremlin-console-3.6.5
```

6. 抽出されたディレクトリにある conf サブディレクトリで、以下のテキストを含む neptune-remote.yaml という名前のファイルを作成します。*your-neptune-endpoint* を Neptune DB インスタンスのホスト名または IP アドレスで置き換えます。角括弧 ([ ]) が必要です。

> **Note**
>
> Neptune DB インスタンスのホスト名を見つける方法については、Amazon Neptune エンドポイントに接続する セクションを参照してください。

```
hosts: [your-neptune-endpoint]
port: 8182
connectionPool: { enableSsl: true }
```

```
serializer: { className:
 org.apache.tinkerpop.gremlin.util.ser.GraphBinaryMessageSerializerV1,
              config: { serializeResultToString: true }}
```

> **Note**
>
> バージョン 3.7.0 では、シリアライザーがgremlin-driverモジュールから新しいgremlin-utilモジュールに移動されました。パッケージが org.apache.tinkerpop.gremlin.driver.ser から org.apache.tinkerpop.gremlin.util.ser に変更されました。

7. ターミナルで Gremlin コンソールディレクトリ (apache-tinkerpop-gremlin-console-3.6.5) に移動し、次のコマンドを入力して Gremlin コンソールを実行します。

```
bin/gremlin.sh
```

以下の出力が表示されます。

```
         \,,,/
         (o o)
-----oOOo-(3)-oOOo-----
plugin activated: tinkerpop.server
plugin activated: tinkerpop.utilities
plugin activated: tinkerpop.tinkergraph
gremlin>
```

gremlin> プロンプトが表示されます。このプロンプトで残りのステップを入力します。

8. gremlin> プロンプトで、次のように入力して Neptune DB インスタンスに接続します。

```
:remote connect tinkerpop.server conf/neptune-remote.yaml
```

9. gremlin> プロンプトで、次のように入力してリモートモードに切り替えます。これにより、すべての Gremlin クエリがリモート接続に送信されます。

```
:remote console
```

10. Gremlin グラフにクエリを送信するには、次のように入力します。

```
g.V().limit(1)
```

11. 完了したら、次のように入力して Gremlin コンソールを終了します。

```
:exit
```

> **Note**
>
> 各ステートメントを区切るには、セミコロン (;) または改行文字 (\n) を使用します。
> 最終的なトラバーサルに先行する各トラバーサルは、`next()` を実行して終わる必要があります。最終的なトラバーサルからのデータのみが返されます。

Gremlin の Neptune 実装の詳細については、the section called “Gremlin の標準コンプライアンス”を参照してください。

Gremlin コンソールに接続する別の方法

通常の接続方法の欠点

Gremlin コンソールに接続する最も一般的な方法は上で説明したものであり、`gremlin>` プロンプトで次のようなコマンドを使用します。

```
gremlin> :remote connect tinkerpop.server conf/(file name).yaml
gremlin> :remote console
```

これはうまく機能し、Neptune にクエリを送信できます。ただし、Groovy スクリプトエンジンがループから外れるため、Neptune はすべてのクエリを純粋な Gremlin として扱います。つまり、以下のクエリフォームは失敗します。

```
gremlin> 1 + 1
gremlin> x = g.V().count()
```

このように接続したときに変数を使うのに一番近いのは、コンソールに保持されている `result` 変数を使用し、次のようにして `:>` を使用してクエリを送信することです。

```
gremlin> :remote console
```

```
==>All scripts will now be evaluated locally - type ':remote console' to return
 to remote mode for Gremlin Server - [krl-1-cluster.cluster-ro-cm9t6tfwbtsr.us-
east-1.neptune.amazonaws.com/172.31.19.217:8182]
gremlin> :> g.V().count()
==>4249

gremlin> println(result)
[result{object=4249 class=java.lang.Long}]

gremlin> println(result['object'])
[4249]
```

別の接続方法

次のように、別の方法で Gremlin コンソールに接続することもできます。

```
gremlin> g = traversal().withRemote('conf/neptune.properties')
```

ここでは、neptune.properties は次の形式です。

```
gremlin.remote.remoteConnectionClass=org.apache.tinkerpop.gremlin.driver.remote.DriverRemoteCon
gremlin.remote.driver.clusterFile=conf/my-cluster.yaml
gremlin.remote.driver.sourceName=g
```

my-cluster.yaml ファイル名は次のようになります。

```
hosts: [my-cluster-abcdefghijk.us-east-1.neptune.amazonaws.com]
port: 8182
serializer: { className:
 org.apache.tinkerpop.gremlin.util.ser.GraphBinaryMessageSerializerV1,
              config: { serializeResultToString: false } }
connectionPool: { enableSsl: true }
```

> **Note**
>
> バージョン 3.7.0 では、シリアライザーがgremlin-driverモジュールから新しいgremlin-utilモジュールに移動されました。パッケージが org.apache.tinkerpop.gremlin.driver.ser から org.apache.tinkerpop.gremlin.util.ser に変更されました。

Gremlin コンソール接続をこのように設定すると、以下の種類のクエリを正常に実行できます。

```
gremlin> 1+1
==>2

gremlin> x=g.V().count().next()
==>4249

gremlin> println("The answer was ${x}")
The answer was 4249
```

次のように、結果を表示せずに済みます。

```
gremlin> x=g.V().count().next();[]
gremlin> println(x)
4249
```

通常のクエリ方法 (ターミナルステップなし) はすべて引き続き機能します。例:

```
gremlin> g.V().count()
==>4249
```

`g.io().read()` ステップを使用して、このような接続でファイルを読み込むこともできます。

HTTPS REST エンドポイントを使用して Neptune DB インスタンスに接続する

Amazon Neptune では、Gremlin クエリ用の HTTPS エンドポイントが用意されています。REST インターフェイスは、DB クラスターが使用している Gremlin バージョンすべてと互換性があります (サポートしている Gremlin リリースを特定するには、実行中の Neptune エンジンバージョンのエンジンリリースページを参照してください)。

> **Note**
>
> 転送中の暗号化: SSL/HTTPS を使用して Neptune に接続する で説明したように、現在 Neptune では、HTTP ではなく HTTPS を使用して接続する必要があります。

次の手順は、curl コマンドおよび HTTPS を使用して Gremlin エンドポイントに接続する方法について説明します。Neptune DB インスタンスと同じ仮想プライベートクラウド (VPC) の Amazon EC2 インスタンスからこれらの手順を実行してください。

Neptune DB インスタンスへの Gremlin クエリ用の HTTPS エンドポイントは `https://your-neptune-endpoint:port/gremlin` です。

> **Note**
>
> Neptune DB インスタンスのホスト名を見つける方法については、Amazon Neptune エンドポイントに接続する を参照してください。

HTTP REST エンドポイントを使用して Neptune に接続するには

次の例では、curl を使用して、HTTP POST を通じて Gremlin クエリを送信します。クエリは、投稿の本文にある JSON 形式で gremlin プロパティとして送信されます。

```
curl -X POST -d '{"gremlin":"g.V().limit(1)"}' https://your-neptune-endpoint:port/
gremlin
```

この例では、g.V().limit(1) トラバーサルを使用してグラフの最初の頂点を返します。その他の対象にクエリを実行するには、別の Gremlin トラバーサルで置き換えます。

> **Important**
>
> デフォルトでは、REST エンドポイントは、単一の JSON 結果セットですべての結果を返します。この結果セットが大きすぎる場合、Neptune DB インスタンスでOutOfMemoryError 例外が発生する可能性があります。
> これを回避するには、チャンク化応答 (結果は一連の個別の応答で返される) を有効にします。オプションの HTTP 末尾ヘッダーを使用して、複数パートの Gremlin 応答を有効にするを参照してください。

Gremlin クエリの送信には HTTP POST リクエストが推奨されますが、HTTP GETリクエストを使用することもできます。

```
curl -G "https://your-neptune-endpoint:port?gremlin=g.V().count()"
```

> **Note**
>
> Neptune は、`bindings` プロパティをサポートしていません。

オプションの HTTP 末尾ヘッダーを使用して、複数パートの Gremlin 応答を有効にする

デフォルトでは、Gremlin クエリに対する HTTP 応答は、単一の JSON 結果セットで返されます。結果セットが非常に大きい場合、これにより DB インスタンスの `OutOfMemoryError` 例外が生じます。

ただし、チャンク化応答 (複数の別々のパートで返される応答) を有効にすることができます。これを行うには、転送エンコーディング (TE) トレーラーのヘッダー (`te: trailers`) をリクエストします。TE ヘッダーの詳細については、TE リクエストヘッダーに関する MDN ページ」を参照してください。

応答が複数のパートで返された場合、最初のパートが受信された後に発生する問題を診断するのは難しい場合があります。これは、最初のパートが `200` (OK)の HTTP ステータスコードで到着するためです。その後は、通常、メッセージ本文に破損した応答が含まれるというエラー状態となり、その最後に Neptune はエラーメッセージを追加します。

この種の障害の検出と診断を容易にするために、Neptune では各応答チャンクの末尾ヘッダー内に 2 つの新しいヘッダーフィールドも含まれます。

- `X-Neptune-Status`— 応答コードの後ろに短い名前が続きます。たとえば、成功した場合、末尾ヘッダーは次のようになります。`X-Neptune-Status: 200 OK`。失敗の場合、応答コードは、`X-Neptune-Status: 500 TimeLimitExceededException` といった Neptune エンジンのエラーコードとなる可能性があります。
- `X-Neptune-Detail`— 成功したリクエストでは空です。エラーの場合は、JSON エラーメッセージが含まれます。HTTP ヘッダー値には ASCII 文字しか使用できないため、JSON 文字列は URL 符号化されます。

> **Note**
>
> Neptune はチャンク化応答の gzip 圧縮は現在サポートしていません。クライアントがチャンクエンコーディングと圧縮の両方を同時に要求すると、Neptune は圧縮をスキップします。

Amazon Neptune で使用する Java ベースの Gremlin クライアント

Amazon Neptune では、Apache Java Gremlin クライアント または Amazon Neptune の Gremlin クライアント の 2 つのオープンソース TinkerPop Java ベースの Gremlin クライアントのいずれかを使用できます。 Amazon Neptune

Apache TinkerPop Java Gremlin クライアント

可能な場合は、エンジンバージョンがサポートする最新バージョンの Apache TinkerPop Java Gremlin クライアントを常に使用してください。新しいバージョンには、クライアントの安定性、パフォーマンス、使いやすさを向上させる多数のバグ修正が含まれています。

次の表に、さまざまな Neptune エンジンバージョンでサポートされている TinkerPop クライアントの最も古いバージョンと最新バージョンを示します。

Neptune エンジンバージョン	最小 TinkerPop バージョン	最大 TinkerPop バージョン
1.3.2.0	3.6.2	3.7.1
1.3.1.0	3.6.2	3.6.5
1.3.0.0	3.6.2	3.6.4
1.2.1.1	3.6.2	3.6.2
1.2.1.0	3.6.2	3.6.2
1.2.0.2	3.5.2	3.5.6
1.2.0.1	3.5.2	3.5.6
1.2.0.0	3.5.2	3.5.6

Neptune エンジンバージョン	最小 TinkerPop バージョン	最大 TinkerPop バージョン
1.1.1.0	3.5.2	3.5.6
1.1.0.0	3.4.0	3.4.13
1.0.5.1 以前	(廃止)	(廃止)

TinkerPop クライアントは通常、シリーズ内で下位互換性があります (3.3.x例えば、 または 3.4.x)。下位互換性を破る必要がある例外的なケースがあるため、新しいクライアントバージョンにアップグレードする前にTinkerPopアップグレードの推奨事項を確認することをお勧めします。

クライアントは、サーバがサポートするバージョンよりも後のバージョンで導入された新手順や新機能を使用できない可能性がありますが、アップグレードの推奨事項が重要な変更を呼び出さない限り、基本的には既存のクエリや特徴が作動します。

> **Note**
>
> Neptune エンジンリリース 1.1.1.0 以降では、 より前の TinkerPop バージョンを使用しません3.5.2。
> Python ユーザーは、直接設定が必要なデフォルトのタイムアウト設定3.4.9のため、TinkerPop バージョンの使用を避ける必要があります (TINKERPOP-2505」を参照)。

Amazon Neptune 用 Gremlin Java クライアント

Amazon Neptune の Gremlin クライアントは、標準の Java クライアントのドロップイン代替として機能するオープンソースの Java ベースの Gremlin クライアントです。 TinkerPop

Neptune Gremlin クライアントは Neptune クラスター用に最適化されています。これにより、クラスター内の複数のインスタンス間のトラフィックディストリビューションを管理し、レプリカを追加または削除するときに、クラスタートポロジの変更に適応できます。ロール、インスタンスタイプ、アベイラビリティーゾーン (AZ)、またはインスタンスに関連付けられたタグに基づいて、クラスター内のインスタンスのサブセットにリクエストを分散するようにクライアントを構成することもできます。

Neptune Gremlin Java クライアントの最新バージョンは Maven Central で利用できます。

Neptune Gremlin Java クライアントの詳細については、このブログ投稿を参照してください。。
コードサンプルとデモについては、クライアントの GitHub プロジェクト を確認してください。

Java クライアントを使用して Neptune DB インスタンスに接続する

次のセクションでは、Neptune DB インスタンスに接続し、Apache TinkerPop Gremlin クライアントを使用して Gremlin トラバーサルを実行する完全な Java サンプルの実行について説明します。

Neptune DB インスタンスと同じ仮想プライベートクラウド (VPC) の Amazon EC2 インスタンスからこれらの手順を実行してください。

Java を使用して Neptune に接続するには

1. Apache Maven を EC2 インスタンスにインストールします。最初に、Maven パッケージを使用してリポジトリを追加するには、次のように入力します。

   ```
   sudo wget https://repos.fedorapeople.org/repos/dchen/apache-maven/epel-apache-maven.repo -O /etc/yum.repos.d/epel-apache-maven.repo
   ```

   パッケージのバージョン番号を設定するには、次のように入力します。

   ```
   sudo sed -i s/\$releasever/6/g /etc/yum.repos.d/epel-apache-maven.repo
   ```

   続いて、yum を使用して Maven をインストールします。

   ```
   sudo yum install -y apache-maven
   ```

2. Java をインストールします。Gremlin ライブラリには Java 8 または 11 が必要です。Java 11 は以下のようにインストールできます。

   - Amazon Linux 2 (AL2) を使用している場合:

     ```
     sudo amazon-linux-extras install java-openjdk11
     ```

   - Amazon Linux 2023 (AL2023) を使用している場合:

     ```
     sudo yum install java-11-amazon-corretto-devel
     ```

   - 他のディストリビューションでは、以下のうち適切なものを使用してください。

```
sudo yum install java-11-openjdk-devel
```

または

```
sudo apt-get install openjdk-11-jdk
```

3. Java 11 を EC2 インスタンスのデフォルトランタイムとして設定: 以下を入力して、Java 8 を EC2 インスタンスのデフォルトランタイムとして設定します。

```
sudo /usr/sbin/alternatives --config java
```

プロンプトが表示されたら、Java 11 の数を入力します。

4. **gremlinjava** という名前の新しいディレクトリを作成します。

```
mkdir gremlinjava
cd gremlinjava
```

5. gremlinjava ディレクトリで pom.xml ファイルを作成してから、テキストエディタで開きます。

```
nano pom.xml
```

6. 以下の内容を pom.xml ファイルにコピーして保存します。

```
<project xmlns="https://maven.apache.org/POM/4.0.0"
         xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="https://maven.apache.org/POM/4.0.0 https://
maven.apache.org/maven-v4_0_0.xsd">
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.amazonaws</groupId>
  <artifactId>GremlinExample</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>GremlinExample</name>
  <url>https://maven.apache.org</url>
  <dependencies>
    <dependency>
```

```
      <groupId>org.apache.tinkerpop</groupId>
      <artifactId>gremlin-driver</artifactId>
      <version>3.6.5</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.apache.tinkerpop/gremlin-groovy
      (Not needed for TinkerPop version 3.5.2 and up)
    <dependency>
      <groupId>org.apache.tinkerpop</groupId>
      <artifactId>gremlin-groovy</artifactId>
      <version>3.6.5</version>
    </dependency> -->
    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-jdk14</artifactId>
      <version>1.7.25</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>2.5.1</version>
        <configuration>
          <source>11</source>
          <target>11</target>
        </configuration>
      </plugin>
        <plugin>
          <groupId>org.codehaus.mojo</groupId>
          <artifactId>exec-maven-plugin</artifactId>
          <version>1.3</version>
          <configuration>
            <executable>java</executable>
            <arguments>
              <argument>-classpath</argument>
              <classpath/>
              <argument>com.amazonaws.App</argument>
            </arguments>
            <mainClass>com.amazonaws.App</mainClass>
            <complianceLevel>1.11</complianceLevel>
            <killAfter>-1</killAfter>
          </configuration>
        </plugin>
```

```
    </plugins>
  </build>
</project>
```

> **Note**
>
> 既存の Maven プロジェクトを変更する場合、必要な依存関係が前述のコードにおいて強調表示されます。

7. コマンドラインで次のように入力して、ソースコード例 (src/main/java/com/amazonaws/) のサブディレクトリを作成します。

```
mkdir -p src/main/java/com/amazonaws/
```

8. src/main/java/com/amazonaws/ ディレクトリで App.java という名前のファイルを作成してから、テキストエディタで開きます。

```
nano src/main/java/com/amazonaws/App.java
```

9. App.java ファイルに次の内容をコピーします。*your-neptune-endpoint* を Neptune DB インスタンスのアドレスで置き換えます。addContactPoint メソッドに https:// プレフィックスを含めることはできません。

> **Note**
>
> Neptune DB インスタンスのホスト名を見つける方法については、Amazon Neptune エンドポイントに接続する を参照してください。

```
package com.amazonaws;
import org.apache.tinkerpop.gremlin.driver.Cluster;
import org.apache.tinkerpop.gremlin.driver.Client;
import
 org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.GraphTraversalSource;
import org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.GraphTraversal;
import static
 org.apache.tinkerpop.gremlin.process.traversal.AnonymousTraversalSource.traversal;
import org.apache.tinkerpop.gremlin.driver.remote.DriverRemoteConnection;
import org.apache.tinkerpop.gremlin.structure.T;
```

```
public class App
{
  public static void main( String[] args )
  {
    Cluster.Builder builder = Cluster.build();
    builder.addContactPoint("your-neptune-endpoint");
    builder.port(8182);
    builder.enableSsl(true);

    Cluster cluster = builder.create();

    GraphTraversalSource g =
 traversal().withRemote(DriverRemoteConnection.using(cluster));

    // Add a vertex.
    // Note that a Gremlin terminal step, e.g. iterate(), is required to make a
 request to the remote server.
    // The full list of Gremlin terminal steps is at https://tinkerpop.apache.org/
docs/current/reference/#terminal-steps
    g.addV("Person").property("Name", "Justin").iterate();

    // Add a vertex with a user-supplied ID.
    g.addV("Custom Label").property(T.id, "CustomId1").property("name", "Custom id
 vertex 1").iterate();
    g.addV("Custom Label").property(T.id, "CustomId2").property("name", "Custom id
 vertex 2").iterate();

    g.addE("Edge Label").from(__.V("CustomId1")).to(__.V("CustomId2")).iterate();

    // This gets the vertices, only.
    GraphTraversal t = g.V().limit(3).elementMap();

    t.forEachRemaining(
      e ->  System.out.println(t.toList())
    );

    cluster.close();
  }
}
```

SSL/TLS (必須) で Neptune に接続する方法については、「SSL/TLS 設定」を参照してください。

10. 次の Maven コマンドを使用してサンプルをコンパイルおよび実行します。

```
mvn compile exec:exec
```

前述の例では、g.V().limit(3).elementMap() トラバーサルを使用して最初の 2 つの頂点の各プロパティのキーと値のマップを返します。その他の対象にクエリを実行するには、いずれかの適切な終了メソッドを持つ Gremlin トラバーサルで置き換えます。

> **Note**
>
> Gremlin クエリの最後の部分、.toList() では、評価のためにトラバーサルをサーバーに送信する必要があります。そのメソッドまたは別の同等のメソッドを含めない場合、クエリは Neptune DB インスタンスに送信されません。
> addV( ) ステップの使用時などの頂点またはエッジを追加するときに、適切な終点を追加する必要もあります。

以下のメソッドは Neptune DB インスタンスにクエリを送信します。

- toList()
- toSet()
- next()
- nextTraverser()
- iterate()

Gremlin Java クライアントの SSL/TLS 設定

Neptune では、SSL/TLS をデフォルトで有効にする必要があります。通常、Java ドライバーが enableSsl(true) で設定されている場合、証明書のローカルコピーで trustStore() または keyStore() を設定しなくても、Neptune に接続できます。以前のバージョンの では、 を使用してローカルkeyCertChainFile()に保存された.pemファイルを設定する TinkerPop ことをお勧めしますが、3.5.x 以降は非推奨となり、使用できなくなります。このセットアップをパブリック証明書で使用していた場合は、SFSRootCAG2.pem を使用して、ローカルコピーを削除できるようになりました。

ただし、接続先のインスタンスがパブリック証明書を検証するためのインターネット接続を持っていない場合や、使用している証明書がパブリックでない場合は、次の手順を実行してローカル証明書のコピーを設定できます。

SSL/TLS を有効にするためのローカル証明書コピーの設定

1. Oracle から keytool をダウンロードしてインストールします。これにより、ローカルキーストアのセットアップが大幅に簡単になります。
2. SFSRootCAG2.pem CA 証明書をダウンロードします (Gremlin Java SDK には、リモート証明書を検証する証明書が必要です)。

```
wget https://www.amazontrust.com/repository/SFSRootCAG2.pem
```

3. JKS 形式または PKCS12 形式でキーストアを作成します。この例では JKS を使用します。プロンプトが表示されたら、次の質問に答えます。ここで作成したパスワードは後で必要になります。

```
keytool -genkey -alias (host name) -keyalg RSA -keystore server.jks
```

4. ダウンロードした SFSRootCAG2.pem ファイルを、新しく作成したキーストアにインポートします。

```
keytool -import -keystore server.jks -file .pem
```

5. Cluster オブジェクトをプログラムで設定します。

```
Cluster cluster = Cluster.build("(your neptune endpoint)")
                         .port(8182)
                         .enableSSL(true)
                         .keyStore('server.jks')
                         .keyStorePassword("(the password from step 2)")
                         .create();
```

Gremlin コンソールの場合と同じように、必要に応じて設定ファイルでも同じことを行うことができます。

```
hosts: [(your neptune endpoint)]
port: 8182
```

```
connectionPool: { enableSsl: true, keyStore: server.jks, keyStorePassword: (the
 password from step 2) }
serializer: { className:
 org.apache.tinkerpop.gremlin.driver.ser.GraphBinaryMessageSerializerV1, config:
 { serializeResultToString: true }}
```

再接続ロジックを使用して Neptune DB インスタンスに接続する Java の例

次の Java の例は、再接続ロジックを使用して Gremlin クライアントに接続して、予期しない切断から回復する方法を示しています。

これには、以下の依存関係があります。

```
<dependency>
    <groupId>org.apache.tinkerpop</groupId>
    <artifactId>gremlin-driver</artifactId>
    <version>${gremlin.version}</version>
</dependency>

<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>amazon-neptune-sigv4-signer</artifactId>
    <version>${sig4.signer.version}</version>
</dependency>

<dependency>
    <groupId>com.evanlennick</groupId>
    <artifactId>retry4j</artifactId>
    <version>0.15.0</version>
</dependency>
```

サンプルコードは次のとおりです。

```
public static void main(String args[]) {
  boolean useIam = true;

  // Create Gremlin cluster and traversal source
  Cluster.Builder builder = Cluster.build()
         .addContactPoint(System.getenv("neptuneEndpoint"))
         .port(Integer.parseInt(System.getenv("neptunePort")))
         .enableSsl(true)
         .minConnectionPoolSize(1)
```

```
        .maxConnectionPoolSize(1)
        .serializer(Serializers.GRAPHBINARY_V1D0)
        .reconnectInterval(2000);

 if (useIam) {
     builder.handshakeInterceptor( r -> {
        try {
           NeptuneNettyHttpSigV4Signer sigV4Signer =
                       new NeptuneNettyHttpSigV4Signer("(your region)", new
DefaultAWSCredentialsProviderChain());
           sigV4Signer.signRequest(r);
        } catch (NeptuneSigV4SignerException e) {
           throw new RuntimeException("Exception occurred while signing the request",
e);
        }
        return r;
     });
  }

 Cluster cluster = builder.create();

 GraphTraversalSource g = AnonymousTraversalSource
     .traversal()
     .withRemote(DriverRemoteConnection.using(cluster));

 // Configure retries
 RetryConfig retryConfig = new RetryConfigBuilder()
     .retryOnCustomExceptionLogic(getRetryLogic())
     .withDelayBetweenTries(1000, ChronoUnit.MILLIS)
     .withMaxNumberOfTries(5)
     .withFixedBackoff()
     .build();

 @SuppressWarnings("unchecked")
 CallExecutor<Object> retryExecutor = new CallExecutorBuilder<Object>()
     .config(retryConfig)
     .build();

 // Do lots of queries
 for (int i = 0; i < 100; i++){
   String id = String.valueOf(i);

   @SuppressWarnings("unchecked")
   Callable<Object> query = () -> g.V(id)
```

```
        .fold()
        .coalesce(
            unfold(),
            addV("Person").property(T.id, id))
        .id().next();

    // Retry query
    // If there are connection failures, the Java Gremlin client will automatically
    // attempt to reconnect in the background, so all we have to do is wait and retry.
    Status<Object> status = retryExecutor.execute(query);

    System.out.println(status.getResult().toString());
  }

  cluster.close();
}

private static Function<Exception, Boolean> getRetryLogic() {

  return e -> {

    Class<? extends Exception> exceptionClass = e.getClass();

    StringWriter stringWriter = new StringWriter();
    String message = stringWriter.toString();


    if (RemoteConnectionException.class.isAssignableFrom(exceptionClass)){
      System.out.println("Retrying because RemoteConnectionException");
      return true;
    }

    // Check for connection issues
    if (message.contains("Timed out while waiting for an available host") ||
        message.contains("Timed-out") && message.contains("waiting for connection on
 Host") ||
        message.contains("Connection to server is no longer active") ||
        message.contains("Connection reset by peer") ||
        message.contains("SSLEngine closed already") ||
        message.contains("Pool is shutdown") ||
        message.contains("ExtendedClosedChannelException") ||
        message.contains("Broken pipe") ||
        message.contains(System.getenv("neptuneEndpoint")))
    {
```

```
      System.out.println("Retrying because connection issue");
      return true;
    };

    // Concurrent writes can sometimes trigger a ConcurrentModificationException.
    // In these circumstances you may want to backoff and retry.
    if (message.contains("ConcurrentModificationException")) {
      System.out.println("Retrying because ConcurrentModificationException");
      return true;
    }

    // If the primary fails over to a new instance, existing connections to the old
 primary will
    // throw a ReadOnlyViolationException. You may want to back and retry.
    if (message.contains("ReadOnlyViolationException")) {
      System.out.println("Retrying because ReadOnlyViolationException");
      return true;
    }

    System.out.println("Not a retriable error");
    return false;
  };
}
```

Python を使用して Neptune DB インスタンスに接続する

可能な場合は、エンジンバージョンがサポートする最新バージョンの Apache TinkerPop Python Gremlin クライアント gremlinpython を常に使用してください。新しいバージョンには、クライアントの安定性、パフォーマンス、使いやすさを向上させる多数のバグ修正が含まれています。使用するgremlinpythonバージョンは通常、Java Gremlin クライアント の表で説明されているバージョンと一致し TinkerPopます。 ???

> **Note**
>
> gremlinpython 3.5.x バージョンは、記述する Gremlin クエリで 3.4.x 機能のみを使用している限り、3.4.x バージョンと TinkerPop互換性があります。

次のセクションでは、Amazon Neptune DB インスタンスに接続し、Gremlin トラバーサルを実施する Python サンプル実行方法について説明します。

Neptune DB インスタンスと同じ仮想プライベートクラウド (VPC) の Amazon EC2 インスタンスからこれらの手順を実行してください。

開始する前に、以下を実行します。

- Python.org ウェブサイトから Python 3.6 以降をダウンロードしてインストールします。
- pip がインストールされていることを確認します。pip がインストールされていないか、または不明な場合は、pip ドキュメンテーション内のpip をインストールする必要がありますか?を参照してください。
- Python のインストールにない場合は、次に示すように `futures` をダウンロードします。`pip install futures`

Python を使用して Neptune に接続するには

1. `gremlinpython` パッケージをインストールするには、次のように入力します。

```
pip install --user gremlinpython
```

2. `gremlinexample.py` という名前のファイルを作成して、テキストエディタで開きます。
3. `gremlinexample.py` ファイルに次の内容をコピーします。*`your-neptune-endpoint`* を Neptune DB インスタンスのアドレスで置き換えます。

   Neptune DB インスタンスのアドレスを見つける方法については、Amazon Neptune エンドポイントに接続する セクションを参照してください。

```
from __future__  import print_function  # Python 2/3 compatibility

from gremlin_python import statics
from gremlin_python.structure.graph import Graph
from gremlin_python.process.graph_traversal import __
from gremlin_python.process.strategies import *
from gremlin_python.driver.driver_remote_connection import DriverRemoteConnection

graph = Graph()

remoteConn = DriverRemoteConnection('wss://your-neptune-endpoint:8182/gremlin','g')
g = graph.traversal().withRemote(remoteConn)

print(g.V().limit(2).toList())
```

```
remoteConn.close()
```

4. サンプルを実行するには、次のコマンドを入力します。

```
python gremlinexample.py
```

この例の最後にある Gremlin クエリは、リストの頂点 (`g.V().limit(2)`) を返します。次に、このリストは標準の Python `print` 関数で表示されます。

> **Note**
>
> Gremlin クエリの最後の部分、`toList()` では、評価のためにトラバーサルをサーバーに送信する必要があります。そのメソッドまたは別の同等のメソッドを含めない場合、クエリは Neptune DB インスタンスに送信されません。

以下のメソッドは Neptune DB インスタンスにクエリを送信します。

- `toList()`
- `toSet()`
- `next()`
- `nextTraverser()`
- `iterate()`

前述の例では、`g.V().limit(2).toList()` トラバーサルを使用してグラフの最初の 2 つの頂点を返します。その他の対象にクエリを実行するには、いずれかの適切な終了メソッドを持つ Gremlin トラバーサルで置き換えます。

.NET を使用して Neptune DB インスタンスに接続する

可能な場合は、エンジンバージョンがサポートする最新バージョンの Apache TinkerPop .NET Gremlin クライアント Gremlin.Net を常に使用してください。新しいバージョンには、クライアントの安定性、パフォーマンス、使いやすさを向上させる多数のバグ修正が含まれています。使用する`Gremlin.Net`バージョンは通常、Java Gremlin クライアント の表で説明されているバージョンと一致し TinkerPopます。 ???

次のセクションには、Neptune DB インスタンスに接続して Gremlin トラバーサルを実行する、C# で記述されたサンプルコードが含まれています。

Amazon Neptune への接続は、Neptune DB インスタンスと同じ仮想プライベートクラウド (VPC) の Amazon EC2 インスタンスから行ってください。このサンプルコードは Ubuntu を実行する Amazon EC2 インスタンスでテスト済みです。

開始する前に、以下を実行します。

- Amazon EC2 インスタンスに .NET をインストールします。Windows、Linux、および macOS を含む複数のオペレーティングシステムで .NET をインストールする手順については、「.NET の開始方法」を参照してください。
- `dotnet add package gremlin.net` を実行して、パッケージに Gremlin.NET をインストールします。詳細については、 TinkerPop ドキュメントのGremlin.NET。

Gremlin.NET を使用して Neptune に接続するには

1. 新しい .NET プロジェクトを作成します。

   ```
   dotnet new console -o gremlinExample
   ```

2. ディレクトリを、新しいプロジェクトディレクトリに変更します。

   ```
   cd gremlinExample
   ```

3. `Program.cs` ファイルに次の内容をコピーします。*`your-neptune-endpoint`* を Neptune DB インスタンスのアドレスで置き換えます。

   Neptune DB インスタンスのアドレスを見つける方法については、Amazon Neptune エンドポイントに接続する セクションを参照してください。

   ```
   using System;
   using System.Threading.Tasks;
   using System.Collections.Generic;
   using Gremlin.Net;
   using Gremlin.Net.Driver;
   using Gremlin.Net.Driver.Remote;
   using Gremlin.Net.Structure;
   using static Gremlin.Net.Process.Traversal.AnonymousTraversalSource;
   namespace gremlinExample
   ```

```
{
  class Program
  {
    static void Main(string[] args)
    {
      try
      {
        var endpoint = "your-neptune-endpoint";
        // This uses the default Neptune and Gremlin port, 8182
        var gremlinServer = new GremlinServer(endpoint, 8182, enableSsl: true );
        var gremlinClient = new GremlinClient(gremlinServer);
        var remoteConnection = new DriverRemoteConnection(gremlinClient, "g");
        var g = Traversal().WithRemote(remoteConnection);
        g.AddV("Person").Property("Name", "Justin").Iterate();
        g.AddV("Custom Label").Property("name", "Custom id vertex 1").Iterate();
        g.AddV("Custom Label").Property("name", "Custom id vertex 2").Iterate();
        var output = g.V().Limit<Vertex>(3).ToList();
        foreach(var item in output) {
            Console.WriteLine(item);
        }
      }
      catch (Exception e)
      {
          Console.WriteLine("{0}", e);
      }
    }
  }
}
```

4. サンプルを実行するには、次のコマンドを入力します。

```
dotnet run
```

この例の最後にある Gremlin クエリは、テスト目的で単一の頂点の数を返します。その後、コンソールに表示されます。

> **Note**
>
> Gremlin クエリの最後の部分、`Next()` では、評価のためにトラバーサルをサーバーに送信する必要があります。そのメソッドまたは別の同等のメソッドを含めない場合、クエリは Neptune DB インスタンスに送信されません。

以下のメソッドは Neptune DB インスタンスにクエリを送信します。

- `ToList()`
- `ToSet()`
- `Next()`
- `NextTraverser()`
- `Iterate()`

クエリ結果をシリアル化して返す必要がある場合、`Next()` を、そうでない場合は `Iterate()` を使用します。

前述の例では、`g.V().Limit(3).ToList()` トラバーサルを使用してリストを返します。その他の対象にクエリを実行するには、いずれかの適切な終了メソッドを持つ Gremlin トラバーサルで置き換えます。

Node.js を使用して Neptune DB インスタンスに接続する

可能な場合は、エンジンバージョンがサポートする最新バージョンの Apache TinkerPop JavaScript Gremlin クライアント gremlin を常に使用してください。新しいバージョンには、クライアントの安定性、パフォーマンス、使いやすさを向上させる多数のバグ修正が含まれています。`gremlin` 使用する のバージョンは、通常、Java Gremlin クライアント の表で説明されている TinkerPop バージョンと一致します。 ???

次のセクションでは、Amazon Neptune DB インスタンスに接続し、Gremlin トラバーサルを実施する Node.js サンプル実行方法について説明します。

Neptune DB インスタンスと同じ仮想プライベートクラウド (VPC) の Amazon EC2 インスタンスからこれらの手順を実行してください。

開始する前に、以下を実行します。

- Node.js バージョン 8.11 以降がインストールされていることを確認します。そうでない場合、Nodejs.org ウェブサイトから Node.js をダウンロードしてインストールします。

Node.js を使用して Neptune に接続するには

1. `gremlin-javascript` パッケージをインストールするには、次のように入力します。

```
npm install gremlin
```

2. `gremlinexample.js` という名前のファイルを作成して、テキストエディタで開きます。
3. `gremlinexample.js` ファイルに次の内容をコピーします。*`your-neptune-endpoint`* を Neptune DB インスタンスのアドレスで置き換えます。

   Neptune DB インスタンスのアドレスを見つける方法については、Amazon Neptune エンドポイントに接続する セクションを参照してください。

```
const gremlin = require('gremlin');
const DriverRemoteConnection = gremlin.driver.DriverRemoteConnection;
const Graph = gremlin.structure.Graph;

dc = new DriverRemoteConnection('wss://your-neptune-endpoint:8182/gremlin',{});

const graph = new Graph();
const g = graph.traversal().withRemote(dc);

g.V().limit(1).count().next().
    then(data => {
        console.log(data);
        dc.close();
    }).catch(error => {
        console.log('ERROR', error);
        dc.close();
    });
```

4. サンプルを実行するには、次のコマンドを入力します。

```
node gremlinexample.js
```

前述の例では、`g.V().limit(1).count().next()` トラバーサルを使用してグラフの単一の頂点の数を返します。その他の対象にクエリを実行するには、いずれかの適切な終了メソッドを持つ Gremlin トラバーサルで置き換えます。

> **Note**
>
> Gremlin クエリの最後の部分、`next()` では、評価のためにトラバーサルをサーバーに送信する必要があります。そのメソッドまたは別の同等のメソッドを含めない場合、クエリは Neptune DB インスタンスに送信されません。

以下のメソッドは Neptune DB インスタンスにクエリを送信します。

- `toList()`
- `toSet()`
- `next()`
- `nextTraverser()`
- `iterate()`

クエリ結果をシリアル化して返す必要がある場合、`next()` を、そうでない場合は `iterate()` を使用します。

> **Important**
>
> これは、スタンドアロンの Node.js 例です。 AWS Lambda 関数でこのようなコードを実行する予定がある場合は、Neptune Lambda 関数で を JavaScript 効率的に使用する方法の詳細については、Lambda 関数の例「」を参照してください。

Go を使用して Neptune DB インスタンスに接続する

可能な場合は、エンジンバージョンがサポートする最新バージョンの Apache TinkerPop Go Gremlin クライアント gremlingo を常に使用してください。新しいバージョンには、クライアントの安定性、パフォーマンス、使いやすさを向上させる多数のバグ修正が含まれています。

使用する`gremlingo`バージョンは通常、Java Gremlin クライアント の表で説明されているバージョンと一致し TinkerPopます。 ???

> **Note**
>
> Gremlingo 3.5.x バージョンは、記述する Gremlin クエリで TinkerPop 3.4.x 機能のみを使用している限り、3.4.x バージョンと下位互換性があります。

次のセクションでは、Amazon Neptune DB インスタンスに接続して、Gremlin トラバーサルを実行する Go サンプルの実行方法について説明します。

Neptune DB インスタンスと同じ仮想プライベートクラウド (VPC) の Amazon EC2 インスタンスからこれらの手順を実行してください。

開始する前に、以下を実行します。

- go.dev ウェブサイトから Go 1.17 以降をダウンロードしてインストールします。

Go を使用して Neptune に接続するには

1. 空のディレクトリから始めて、新しい Go モジュールを初期化します。

```
go mod init example.com/gremlinExample
```

2. gremlin-go を新しいモジュールの依存関係として追加します。

```
go get github.com/apache/tinkerpop/gremlin-go/v3/driver
```

3. 「gremlinExample.go」という名前のファイルを作成し、テキストエディタで開きます。
4. 以下を gremlinExample.go ファイルにコピーし、*(your neptune endpoint)* を Neptune DB インスタンスのアドレスに置き換えます。

```
package main

import (
  "fmt"
  gremlingo "github.com/apache/tinkerpop/gremlin-go/v3/driver"
)

func main() {
  // Creating the connection to the server.
```

```
  driverRemoteConnection, err := gremlingo.NewDriverRemoteConnection("wss://(your
 neptune endpoint):8182/gremlin",
    func(settings *gremlingo.DriverRemoteConnectionSettings) {
      settings.TraversalSource = "g"
    })
  if err != nil {
    fmt.Println(err)
    return
  }
  // Cleanup
  defer driverRemoteConnection.Close()

  // Creating graph traversal
  g := gremlingo.Traversal_().WithRemote(driverRemoteConnection)

  // Perform traversal
  results, err := g.V().Limit(2).ToList()
  if err != nil {
    fmt.Println(err)
    return
  }
  // Print results
  for _, r := range results {
    fmt.Println(r.GetString())
  }
}
```

> **Note**
>
> Neptune TLS 証明書形式は、現在、macOS 搭載の Go 1.18+ ではサポートされていないため、接続を開始しようとすると 509 エラーが発生する可能性があります。ローカルテストでは、インポートに「crypto/tls」を追加し、DriverRemoteConnection 設定を次のように変更することで、これをスキップできます。
>
> ```
> // Creating the connection to the server.
> driverRemoteConnection, err := gremlingo.NewDriverRemoteConnection("wss://
> your-neptune-endpoint:8182/gremlin",
>   func(settings *gremlingo.DriverRemoteConnectionSettings) {
>       settings.TraversalSource = "g"
>       settings.TlsConfig = &tls.Config{InsecureSkipVerify: true}
>   })
> ```

5. サンプルを実行するには、次のコマンドを入力します。

```
go run gremlinExample.go
```

この例の最後にある Gremlin クエリは、頂点 (`g.V().Limit(2)`) をスライスで返します。このスライスは、標準の `fmt.Println` 関数で繰り返され、出力されます。

> **Note**
>
> Gremlin クエリの最後の部分、`ToList()` では、評価のためにトラバーサルをサーバーに送信する必要があります。そのメソッドまたは別の同等のメソッドを含めない場合、クエリは Neptune DB インスタンスに送信されません。

以下のメソッドは Neptune DB インスタンスにクエリを送信します。

- `ToList()`
- `ToSet()`
- `Next()`
- `GetResultSet()`
- `Iterate()`

前述の例では、`g.V().Limit(2).ToList()` トラバーサルを使用してグラフの最初の 2 つの頂点を返します。その他の対象にクエリを実行するには、いずれかの適切な終了メソッドを持つ Gremlin トラバーサルで置き換えます。

Gremlin クエリヒント

クエリヒントを使用して Amazon Neptune の特定の Gremlin クエリの最適化と評価戦略を指定することができます。

クエリヒントは、次の構文を使用して `withSideEffect` ステップをクエリに追加することによって指定されます。

```
g.withSideEffect(hint, value)
```

- hint – 適用するヒントのタイプを特定します。

- value – 検討中のシステムアスペクトの動作を決めます。

たとえば、以下は Gremlin トラバーサルに `repeatMode` ヒントを含める方法を示します。

> **Note**
>
> すべての Gremlin クエリヒントの副作用にはプレフィックス `Neptune#` が付けられます。

```
g.withSideEffect('Neptune#repeatMode',
 'DFS').V("3").repeat(out()).times(10).limit(1).path()
```

前述のクエリは、Neptune エンジンにデフォルトの Neptune 幅優先(BFS)ではなく、深さ優先(DFS)グラフをトラバースするように指示します。

以下のセクションでは、使用可能なクエリヒントとその使用方法に関する詳しい情報が記載されています。

トピック

- Gremlin repeatMode クエリヒント
- Gremlin noReordering クエリヒント
- Gremlin typePromotion クエリヒント
- Gremlin useDFE クエリヒント
- 結果キャッシュを使用する Gremlin クエリヒント

Gremlin repeatMode クエリヒント

Neptune`repeatMode` クエリヒントは、Neptune エンジンが Gremlin トラバーサル (幅優先、深さ優先、チャンク深さ優先) の`repeat()` ステップを評価する方法を指定します。

`repeat()` ステップの評価モードは、パスを探したりパスに従ったりするために使用される場合、単に制限回数までステップを繰り返すことよりも重要です。

構文

`repeatMode` クエリヒントは、クエリに `withSideEffect` ステップを追加して指定します。

```
g.withSideEffect('Neptune#repeatMode', 'mode').gremlin-traversal
```

> **Note**
>
> すべての Gremlin クエリヒントの副作用にはプレフィックス Neptune# が付けられます。

利用可能なモード

- BFS

  幅優先検索

  repeat() ステップのデフォルトの実行モードです。これにより、パスに沿って深くなる前にすべての兄弟ノードを取得します。

  このバージョンはメモリを大量に消費し、領域が非常に大きくなる可能性があります。クエリがメモリ不足になって、Neptune エンジンによりキャンセルされるリスクが高くなります。これは他の Gremlin 実装に最も適合します。
- DFS

  深さ優先検索

  次のソリューションに移る前に最大深度への各パスに従います。

  メモリ使用量は少なくなります。複数のホップ外の開始点から単一のパスを検索するような状況では、パフォーマンスが向上する可能性があります。
- CHUNKED_DFS

  チャンク深さ優先検索

  1 個のノード (DFS) またはすべてのノード (BFS) ではなく、1,000 個のノードのチャンクでグラフ深さ優先を調べるハイブリッドアプローチです。

  Neptune エンジンは、より深くパスに従う前に各レベルで最大 1,000 個のノードを取得します。

  これは、速度とメモリ使用量のバランスが取れたアプローチです。

  また、BFS を使用する場合に便利ですが、このクエリはメモリを過度に使用しています。

例

次のセクションでは、Gremlin トラバーサルに対する繰り返しモードの効果について説明します。

Neptune では、`repeat()` ステップのデフォルトモードは、すべてのトラバーサルに対して幅優先 (BFS) 実行戦略を行うことになっています。

ほとんどの場合、 TinkerGraph 実装は同じ実行戦略を使用しますが、トラバーサルの実行を変更する場合もあります。

例えば、 TinkerGraph 実装は次のクエリを変更します。

```
g.V("3").repeat(out()).times(10).limit(1).path()
```

このトラバーサルの `repeat()` ステップは次のトラバーサルに「アンロール」され、深さ優先 (DFS) 戦略になります。

```
g.V(<id>).out().out().out().out().out().out().out().out().out().out().limit(1).path()
```

> **Important**
>
> Neptune クエリエンジンでは、これは自動的に実行されません。

ブロードファースト (BFS) はデフォルトの実行戦略であり、ほとんどの場合 TinkerGraph に似ています。ただし、深さ優先 (DFS) 戦略が望ましい場合があります。

BFS (デフォルト)

幅優先 (BFS) は、`repeat()` 演算子のデフォルトの実行戦略です。

```
g.V("3").repeat(out()).times(10).limit(1).path()
```

Neptune エンジンは、10 のホップまでソリューションを見つける前に、最初の 9 つのホップ領域を完全に探します。これは、最短パスのクエリなど多くの場合で効果的です。

ただし、前述の例では、`repeat()` 演算子の深さ優先 (DFS) モードを使用するとトラバーサルははるかに高速になります。

DFS

次のクエリでは、repeat() 演算子の深さ優先 (DFS) モードを使用します。

```
g.withSideEffect("Neptune#repeatMode", "DFS").V("3").repeat(out()).times(10).limit(1)
```

ここでは、次のソリューションを探す前に、最大深度まで個別のソリューションに従います。

Gremlin noReordering クエリヒント

Gremlin トラバーサルを送信するとき、Neptune クエリエンジンは、評価に必要な作業の量とクエリ応答時間を最小限にしようとするために、トラバーサルの構造を調査し、クエリの各パートの順序を変更します。たとえば、複数の has() ステップなど、複数の制約を持つトラバーサルは通常、指定された順序では評価されません。代わりに、クエリが静的分析でチェックされた後に順序が変更されます。

Neptune クエリエンジンは、より細かく選択される制約を特定しようと試み、その 1 つを最初に実行します。多くの場合、これによりパフォーマンスが向上しますが、Neptune で選択されるクエリの評価の順序は常に最適であるとは限りません。

データの正確な特性がわかっていて、クエリ実行の順序を手動で指定する場合は、Neptune noReordering クエリヒントを使用して特定の順序でトラバーサルが評価されるように指定できます。

構文

noReordering クエリヒントは、クエリに withSideEffect ステップを追加して指定します。

```
g.withSideEffect('Neptune#noReordering', true or false).gremlin-traversal
```

> **Note**
>
> すべての Gremlin クエリヒントの副作用にはプレフィックス Neptune# が付けられます。

指定できる値

- true
- false

Gremlin typePromotion クエリヒント

数値または範囲でフィルタリングする Gremlin トラバーサルを送信する場合、Neptune クエリエンジンは通常、クエリの実行時に型の上位変換を使用する必要があります。つまり、フィルタリングする値を保持できるすべてのタイプの値を調べる必要があります。

たとえば、55 に等しい値をフィルタリングする場合、エンジンは 55 に等しい整数、55L に等しい長整数、55.0 に等しい浮動小数点数などを探す必要があります。各型の上位変換では、ストレージに対する追加のルックアップが必要なため、単純なクエリの完了に予期せず長い時間がかかることがあります。

たとえば、顧客年齢プロパティが 5 より大きいすべての頂点を検索するとします。

```
g.V().has('customerAge', gt(5))
```

この探索を徹底的に実行するには、Neptune はクエリを展開して、クエリ対象の値を上位変換できるすべての数値型を調べる必要があります。この場合は、gtフィルターは、5 を超える任意の整数、5 L を超える任意の長整数、5.0 を超える任意の浮動小数点数、5.0 を超える任意の倍数に適用する必要があります。これらの型の上位変換はそれぞれ、ストレージに関する追加のルックアップを必要とするため、数値フィルターごとに複数のフィルターが表示されます。Gremlin `profile` APIこのクエリでは、完了までに予想よりもかなり長い時間がかかります。

多くの場合、特定のタイプの値を見つけるだけで済むことが事前にわかっているため、型の上位変換は不要です。この場合、クエリを劇的に高速化するには、型の上位変換をオフにする `typePromotion` クエリヒントを使います。

構文

`typePromotion` クエリヒントは、クエリに `withSideEffect` ステップを追加して指定します。

```
g.withSideEffect('Neptune#typePromotion', true or false).gremlin-traversal
```

> **Note**
>
> すべての Gremlin クエリヒントの副作用にはプレフィックス `Neptune#` が付けられます。

指定できる値

- `true`

- `false`

上記のクエリの型の上位変換をオフにするには、以下を使用します。

```
g.withSideEffect('Neptune#typePromotion', false).V().has('customerAge', gt(5))
```

Gremlin useDFE クエリヒント

このクエリヒントを使用して、クエリを実行するための DFE の使用を有効にします。neptune_dfe_query_engine インスタンスパラメータはデフォルトでは `viaQueryHint` に設定されるため、デフォルトでは、このクエリヒントが `true` に設定されていないと、Neptune は DFE を使用しません。このインスタンスパラメータを `enabled` に設定した場合、`useDFE` クエリヒントが `false` に設定されているクエリを除き、すべてのクエリに DFE エンジンが使用されます。

クエリの DFE を有効にする例:

```
g.withSideEffect('Neptune#useDFE', true).V().out()
```

結果キャッシュを使用する Gremlin クエリヒント

次のクエリヒントは、クエリ結果のキャッシュが有効である場合に使用できます。

Gremlin **`enableResultCache`**クエリヒント

`enableResultCache` の値を持つ `true` クエリヒントは、クエリ結果が既にキャッシュされている場合、キャッシュから返されます。そうでない場合は、新しい結果を返し、キャッシュからクリアされるまでキャッシュします。例:

```
g.with('Neptune#enableResultCache', true)
 .V().has('genre','drama').in('likes')
```

後で、まったく同じクエリを再度発行することで、キャッシュされた結果にアクセスできます。

このクエリヒントの値が `false` または、存在しない場合、クエリ結果はキャッシュされません。ただし、それを`false` に設定すると、既存のキャッシュされた結果をクリアしません。キャッシュされた結果をクリアするには、`invalidateResultCache`または`invalidateResultCachekey`ヒントを使用します。

Gremlin **enableResultCacheWithTTL**クエリヒント

enableResultCacheWithTTLクエリヒントは、キャッシュされた結果がある場合、キャッシュ内にすでにある結果の TTL に影響を与えずに、キャッシュされた結果を返します。キャッシュされた結果が現在存在しない場合、クエリはenableResultCacheWithTTLクエリヒントにより指定される有効期限(TTL) に対して新しい結果を返し、それをキャッシュします。その有効期限は秒単位で指定します。たとえば、次のクエリでは 60 秒の有効期限を指定します。

```
g.with('Neptune#enableResultCacheWithTTL', 60)
 .V().has('genre','drama').in('likes')
```

60 秒が経過する前に、 enableResultCacheまたはクエリヒントのいずれかで同じenableResultCacheWithTTLクエリ (ここで time-to-live は g.V().has('genre','drama').in('likes')) を使用して、キャッシュされた結果にアクセスできます。

> **Note**
>
> 指定される有効期限enableResultCacheWithTTLは、すでにキャッシュされている結果には影響しません。
>
> - 結果が enableResultCache を使って以前にキャッシュされていた場合、enableResultCacheWithTTL 新しい結果を生成し、指定した TTL 用にキャッシュする前に、まずキャッシュを明示的にクリアする必要があります。
> - 結果が enableResultCachewithTTL を使って以前にキャッシュされていた場合、enableResultCacheWithTTL が新しい結果を生成し、指定した TTL 用にキャッシュする前に、まず前の TTL を期限切れとする必要があります。

有効期限が過ぎると、クエリのキャッシュされた結果がクリアされ、同じクエリの後続のインスタンスが新しい結果を返します。後続のクエリに enableResultCacheWithTTL がアタッチされている場合、新しい結果は指定された TTL でキャッシュされます。

Gremlin **invalidateResultCacheKey**クエリヒント

invalidateResultCacheKey クエリヒントは、true または false 値を取ることができます。あるtrue 値を指定すると、invalidateResultCacheKey がアタッチされてクリアされるクエリに対してキャッシュされた結果が発生します。たとえば、次の例では、クエリキー g.V().has('genre','drama').in('likes') のキャッシュされた結果がクリアされます。

```
g.with('Neptune#invalidateResultCacheKey', true)
 .V().has('genre','drama').in('likes')
```

上記のクエリ例では、新しい結果がキャッシュされることはありません。既存のキャッシュされた結果をクリアした後に新しい結果をキャッシュする場合は、同じクエリで、enableResultCache (またはenableResultCacheWithTTL) を含めることができます。

```
g.with('Neptune#enableResultCache', true)
 .with('Neptune#invalidateResultCacheKey', true)
 .V().has('genre','drama').in('likes')
```

Gremlin **invalidateResultCache**クエリヒント

invalidateResultCache クエリヒントは、true または false 値を取ることができます。ある true 値を指定すると、結果キャッシュ内のすべての結果がクリアされます。例:

```
g.with('Neptune#invalidateResultCache', true)
 .V().has('genre','drama').in('likes')
```

上記のクエリ例では、結果がキャッシュされることはありません。既存のキャッシュを完全にクリアした後に新しい結果をキャッシュする場合は、同じクエリで、enableResultCache (またはenableResultCacheWithTTL) を含めることができます。

```
g.with('Neptune#enableResultCache', true)
 .with('Neptune#invalidateResultCache', true)
 .V().has('genre','drama').in('likes')
```

Gremlin **numResultsCached**クエリヒント

numResultsCached クエリヒントは、iterate() を含むクエリでのみ使用でき、アタッチ先のクエリに対してキャッシュする結果の最大数を指定します。numResultsCached が存在する場合にキャッシュされる結果は返されず、キャッシュされるだけであることに注意してください。

たとえば、次のクエリでは、結果のうち最大 100 個をキャッシュするよう指定していますが、キャッシュされた結果は返されません。

```
g.with('Neptune#enableResultCache', true)
 .with('Neptune#numResultsCached', 100)
```

```
  .V().has('genre','drama').in('likes').iterate()
```

次のようなクエリを使用して、キャッシュされた結果の範囲 (ここでは最初の 10 件) を取得できます。

```
g.with('Neptune#enableResultCache', true)
 .with('Neptune#numResultsCached', 100)
 .V().has('genre','drama').in('likes').range(0, 10)
```

Gremlin **noCacheExceptions**クエリヒント

noCacheExceptions クエリヒントは、true または false 値を取ることができます。ある true 値を指定すると、結果キャッシュに関連するすべての例外が抑制されます。例:

```
g.with('Neptune#enableResultCache', true)
 .with('Neptune#noCacheExceptions', true)
 .V().has('genre','drama').in('likes')
```

特に、これは QueryLimitExceededException を抑制し、これはクエリの結果が大きすぎて結果キャッシュに収まらない場合に発生します。

Gremlin クエリステータス API

Gremlin クエリのステータスを取得するには、HTTP GET または POST を使用して、https://*your-neptune-endpoint*:*port*/gremlin/status エンドポイントにリクエストを送信します。

Gremlin クエリステータスのリクエストパラメータ

- queryId (オプショナル) — 実行中の Gremlin クエリの ID。指定したクエリのステータスのみを表示します。
- includeWaiting (オプショナル) — 待機中のすべてのクエリのステータスを返します。

  通常、実行中のクエリだけが応答に含まれますが、includeWaiting パラメータを指定すると、待機中のすべてのクエリのステータスも返されます。

Gremlin クエリステータスのレスポンスの構文

```
{
```

```
  "acceptedQueryCount": integer,
  "runningQueryCount": integer,
  "queries": [
    {
      "queryId":"guid",
      "queryEvalStats":
        {
          "waited": integer,
          "elapsed": integer,
          "cancelled": boolean
        },
      "queryString": "string"
    }
  ]
}
```

Gremlin クエリステータスのレスポンス値

- acceptedQueryCount – キュー内のクエリを含め、受け入れられたが、まだ完了していないクエリの数。
- runningQueryCount – 現在実行中の Gremlin クエリの数。
- queries – 現在の Gremlin クエリのリスト。
- queryId – クエリの GUID ID。Neptune は、この ID 値を各クエリに自動的に割り当てます。または、独自の ID を割り当てることもできます (Neptune Gremlin または SPARQL クエリにカスタム ID を挿入するを参照)。
- queryEvalStats – このクエリの統計。
- subqueries — このクエリのサブクエリの数。
- elapsed – これまでクエリが実行されていた時間 (マイクロ秒)。
- cancelled – True はクエリがキャンセルされたことを示します。
- queryString – 送信されたクエリ。それより長い場合、これは 1024 文字に切り捨てられます。
- waited - クエリが待機していた時間をミリ秒単位で示します。

Gremlin クエリステータスの例

以下は、`curl` と HTTP `GET` を使用したステータスコマンドの例です。

```
curl https://your-neptune-endpoint:port/gremlin/status
```

この出力には、実行中の単一のクエリが表示されます。

```
{
  "acceptedQueryCount":9,
  "runningQueryCount":1,
  "queries": [
    {
      "queryId":"fb34cd3e-f37c-4d12-9cf2-03bb741bf54f",
      "queryEvalStats":
        {
          "waited": 0,
          "elapsed": 23,
          "cancelled": false
        },
      "queryString": "g.V().out().count()"
    }
  ]
}
```

Gremlin クエリのキャンセル

Gremlin クエリのステータスを取得するには、HTTP GET または POST を使用して、https://*your-neptune-endpoint*:*port*/gremlin/status エンドポイントにリクエストを送信します。

Gremlin クエリのキャンセルリクエストパラメータ

- cancelQuery – キャンセルで必須です。このパラメータには対応する値がありません。
- queryId – キャンセルする実行中の Gremlin クエリの ID。

Gremlin クエリのキャンセルの例

以下は、クエリのキャンセルする curl コマンドの例です。

```
curl https://your-neptune-endpoint:port/gremlin/status \
  --data-urlencode "cancelQuery" \
  --data-urlencode "queryId=fb34cd3e-f37c-4d12-9cf2-03bb741bf54f"
```

正常にキャンセルされると、HTTP 200 OK が返されます。

Gremlin スクリプトベースのセッションのサポート

Amazon Neptune で暗示的なトランザクションを伴う Gremlin セッションを使用できます。Gremlin セッションの詳細については、Apache TinkerPop ドキュメントの「セッションの検討」を参照してください。以下のセクションでは、Java で Gremlin セッションを使用する方法について説明します。

> **Note**
>
> この機能は、Neptune エンジンリリース 1.0.1.0.200463.0 からアクセスできます。Neptune エンジンリリース 1.1.1.0 および TinkerPop バージョン 3.5.2 以降では、 を使用することもできますGremlin トランザクション。

> **Important**
>
> 現在、Neptune がスクリプトベースのセッションを開いたままにできる最長時間は 10 分です。それ以前にセッションを閉じないと、セッションはタイムアウトし、その中のすべてがロールバックされます。

トピック

- Gremlin コンソール上の Gremlin セッション
- Gremlin 言語バリアントの Gremlin セッション

Gremlin コンソール上の Gremlin セッション

`session` パラメータを指定せずに Gremlin コンソールでリモート接続を作成すると、リモート接続はセッションレスモードで作成されます。このモードでは、サーバーに送信される各リクエストはそれ自体で完全なトランザクションとして扱われ、リクエスト間の状態は保存されません。リクエストが失敗すると、そのリクエストのみがロールバックされます。

`session` パラメータを使用するリモート接続を作成した場合、リモート接続を閉じるまで続くスクリプトベースのセッションを作成します。すべてのセッションは、コンソールが生成してユーザーに返す固有の UUID によって識別されます。

次に示すのは、セッションを作成する 1 つのコンソール呼び出しの例です。クエリが送信されると、別の呼び出しによってセッションが閉じられ、クエリがコミットされます。

> **Note**
>
> サーバー側のリソースを解放するには、Gremlin クライアントを常に閉じておく必要があります。

```
gremlin> :remote connect tinkerpop.server conf/neptune-remote.yaml session
  . . .
  . . .
gremlin> :remote close
```

詳細と例については、 TinkerPop ドキュメントの「セッション」を参照してください。

セッション中に実行するすべてのクエリは、単一のトランザクションを形成します。そのトランザクションは、クエリがすべて成功してリモート接続を閉じるまでコミットされません。クエリが失敗した場合、または Neptune でサポートされるセッションの最大有効期間内に接続を閉じなかった場合、セッショントランザクションはコミットされず、その中のすべてのクエリがロールバックされます。

Gremlin 言語バリアントの Gremlin セッション

次の例のように、Gremlin 言語バリアント (GLV) では、単一のトランザクションで複数のクエリを発行するための `SessionedClient` オブジェクトを作成する必要があります。

```
try {                                // line 1
  Cluster cluster = Cluster.open();                    // line 2
  Client client = cluster.connect("sessionName");      // line 3
   ...
   ...
} finally {
  // Always close. If there are no errors, the transaction is committed; otherwise,
 it's rolled back.
  client.close();
}
```

前の例の行 3 は、問題のクラスターについて設定されている設定オプションに従って `SessionedClient` オブジェクトを作成します。接続メソッドに渡す *`sessionName`* 文字列は、セッションの一意の名前になります。競合を避けるために、名前に UUID を使用してください。

クライアントは初期化されると、セッショントランザクションを開始します。セッションの形成中に実行したすべてのクエリは、`client.close( )` を呼び出すときにのみコミットされます。繰り返しますが、単一のクエリが失敗した場合、または Neptune でサポートされるセッションの最大有効期間内に接続を閉じなかった場合、セッショントランザクションは失敗し、その中のすべてのクエリがロールバックされます。

> **Note**
>
> サーバー側のリソースを解放するには、Gremlin クライアントを常に閉じておく必要があります。

```
GraphTraversalSource g = traversal().withRemote(conn);

Transaction tx = g.tx();

// Spawn a GraphTraversalSource from the Transaction.
// Traversals spawned from gtx are executed within a single transaction.
GraphTraversalSource gtx = tx.begin();
try {
  gtx.addV('person').iterate();
  gtx.addV('software').iterate();

  tx.commit();
} finally {
    if (tx.isOpen()) {
        tx.rollback();
    }
}
```

Neptune での Gremlin トランザクション

Gremlin トランザクションが実行されるコンテキストはいくつかあります。Gremlin を使用する際には、作業対象となるコンテキストとその意味を理解することが重要です。

- **`Script-based`** — リクエストは、次のようなテキストベースの Gremlin 文字列を使用して行われます。
  - Java ドライバーと `Client.submit(`*`string`*`)` の使用。
  - Gremlin コンソールと `:remote connect` の使用。

  - バルク API の使用。
- **`Bytecode-based`** — リクエストは、Gremlin 言語バリアント (GLV) のシリアル化された Gremlin バイトコードを使用して行われます。

  例えば、Java ドライバー `g = traversal().withRemote(...)` を使用する場合などです。

上記のコンテキストのどちらでも、リクエストがセッションレスまたはセッションにバインドされた状態で送信されるという追加のコンテキストがあります。

> **Note**
>
> Gremlin トランザクションは、サーバー側のリソースを解放できるように、常にコミットまたはロールバックする必要があります。

セッションレスリクエスト

セッションレスの場合、リクエストは 1 回のトランザクションに相当します。

スクリプトの場合、1 回のリクエストで送信された 1 つ以上の Gremlin ステートメントが 1 つのトランザクションとしてコミットまたはロールバックされることを意味します。例:

```
Cluster cluster = Cluster.open();
Client client = cluster.connect(); // sessionless
// 3 vertex additions in one request/transaction:
client.submit("g.addV();g.addV();g.addV()").all().get();
```

バイトコードの場合、g からトラバーサルが生成され実行されるたびに、セッションレスリクエストが行われます。

```
GraphTraversalSource g = traversal().withRemote(...);

// 3 vertex additions in three individual requests/transactions:
g.addV().iterate();
g.addV().iterate();
g.addV().iterate();

// 3 vertex additions in one single request/transaction:
g.addV().addV().addV().iterate();
```

セッションにバインドされたリクエスト

セッションにバインドすると、1 つのトランザクションのコンテキスト内で複数のリクエストを適用できます。

スクリプトの場合、すべてのグラフ操作を連結して 1 つの埋め込み文字列値にする必要がないということです。

```
Cluster cluster = Cluster.open();
Client client = cluster.connect(sessionName); // session
try {
    // 3 vertex additions in one request/transaction:
    client.submit("g.addV();g.addV();g.addV()").all().get();
} finally {
    client.close();
}

try {
    // 3 vertex additions in three requests, but one transaction:
    client.submit("g.addV()").all().get(); // starts a new transaction with the same
 sessionName
    client.submit("g.addV()").all().get();
    client.submit("g.addV()").all().get();
} finally {
    client.close();
}
```

バイトコードの場合、 の後 TinkerPop `3.5.x`、トランザクションを明示的に制御し、セッションを透過的に管理できます。Gremlin 言語バリアント (GLV) は、以下のように Gremlin の `tx()` 構文をサポートして、トランザクションを `commit()` または `rollback()` します。

```
GraphTraversalSource g = traversal().withRemote(conn);

Transaction tx = g.tx();

// Spawn a GraphTraversalSource from the Transaction.
// Traversals spawned from gtx are executed within a single transaction.
GraphTraversalSource gtx = tx.begin();
try {
    gtx.addV('person').iterate();
    gtx.addV('software').iterate();
```

```
    tx.commit();
} finally {
    if (tx.isOpen()) {
        tx.rollback();
    }
}
```

上記の例は Java で記述されていますが、この `tx()` 構文は、Python、Javascript、.NET でも使用できます。

> **Warning**
>
> セッションレス読み取り専用クエリは SNAPSHOT 分離下で実行されますが、明示的なトランザクション内で実行される読み取り専用クエリは SERIALIZABLE 分離下で実行されます。SERIALIZABLE 分離下で実行される読み取り専用クエリは、SNAPSHOT 分離下で実行されるクエリとは異なり、オーバーヘッドが大きくなり、同時書き込みによってブロックしたり、ブロックされたりする可能性があります。

Amazon Neptune で Gremlin API を使用する

> **Note**
>
> Amazon Neptune は、`bindings` プロパティをサポートしていません。

Gremlin HTTPS では、すべて単一のエンドポイント `https://`*`your-neptune-endpoint`*`:`*`port`*`/gremlin` を使用する必要があります。すべての Neptune 接続は HTTPS を使用する必要があります。

を介して Gremlin コンソールを Neptune グラフに直接接続できます WebSockets。

Gremlin エンドポイントへの接続の詳細については、「Gremlin を使用した Neptune グラフへのアクセス」を参照してください。

Gremlin の Amazon Neptune 実装には、考慮する必要がある特定の詳細と相違点があります。詳細については、「Amazon Neptune の Gremlin 標準への準拠」を参照してください。

Gremlin 言語とトラバーサルの詳細については、Apache ドキュメントの「トラバーサル」を参照してください。 TinkerPop

Amazon Neptune Gremlin でクエリ結果をキャッシュする

エンジンリリース 1.0.5.1 から、Amazon Neptune は Gremlin クエリの結果キャッシュをサポートしています。

クエリ結果キャッシュを有効にし、クエリヒントを使用して Gremlin 読み取り専用クエリの結果をキャッシュできます。

クエリを再実行すると、キャッシュ内に残っている限り、低レイテンシーで I/O コストなしでキャッシュされた結果を取得します。これは、HTTP エンドポイントと WebSocket を使用して、バイトコードまたは文字列形式で送信されたクエリに対して機能します。

> **Note**
>
> プロファイルエンドポイントに送信されたクエリは、クエリキャッシュが有効になっていてもキャッシュされません。

Neptune クエリ結果キャッシュの動作は、いくつかの方法で制御できます。例:

- キャッシュされた結果をブロック単位でページ分割できます。
- 指定されたクエリの time-to-live (TTL) を指定できます。
- 指定したクエリのキャッシュをクリアすることができます。
- キャッシュ全体をクリアできます。
- 結果がキャッシュサイズを超えた場合に通知されるように設定できます。

キャッシュは least-recently-used (LRU) ポリシーを使用して維持されます。つまり、キャッシュに割り当てられた領域がいっぱいになると、新しい least-recently-used 結果がキャッシュされるときに結果が削除され、スペースが確保できるようになります。

> **Important**
>
> クエリ結果キャッシュは、`t3.medium` または `t4.medium` インスタンスタイプでは使用できません。

Neptune でクエリ結果キャッシュを有効にする

Neptune でクエリ結果キャッシュを有効にするには、コンソールを使用して `neptune_result_cache` DB インスタンスのパラメータを 1 (有効に) 設定します。

結果キャッシュを有効にすると、Neptune は現在のメモリの一部をクエリ結果をキャッシュするために確保します。使用しているインスタンスタイプが大きくなり、使用可能なメモリが多いほど、Neptune がキャッシュに割り当てるメモリが多くなります。

結果のキャッシュメモリがいっぱいになると、Neptune は least-recently-used (LRU) キャッシュされた結果を自動的に削除して、新しい結果に取り掛かります。

結果キャッシュの現在のステータスを確認するには、インスタンスのステータス コマンドを使います。

ヒントを使用してクエリ結果をキャッシュする

クエリ結果キャッシュを有効にすると、クエリヒントを使用してクエリキャッシュを制御します。以下の例はすべて、同じクエリトラバーサルに適用されます。

```
g.V().has('genre','drama').in('likes')
```

`enableResultCache` を使用する

クエリ結果キャッシュを有効にした状態で、Gremlin クエリの結果をキャッシュするには、次のように `enableResultCache` クエリヒントを使います。

```
g.with('Neptune#enableResultCache', true)
 .V().has('genre','drama').in('likes')
```

Neptune はクエリ結果を返し、キャッシュします。後で、まったく同じクエリを再度発行することで、キャッシュされた結果にアクセスできます。

```
g.with('Neptune#enableResultCache', true)
 .V().has('genre','drama').in('likes')
```

キャッシュされた結果を識別するキャッシュキーは、クエリ文字列そのものです。

```
g.V().has('genre','drama').in('likes')
```

enableResultCacheWithTTL を使用する

クエリ結果をキャッシュする期間を指定するには、enableResultCacheWithTTL クエリヒントを使います。たとえば、次のクエリでは、クエリ結果が 120 秒後に期限切れになるよう指定しています。

```
g.with('Neptune#enableResultCacheWithTTL', 120)
 .V().has('genre','drama').in('likes')
```

ここでもキャッシュされた結果を識別するキャッシュキーは、ベースクエリ文字列そのものです。

```
g.V().has('genre','drama').in('likes')
```

また、enableResultCache クエリヒントでそのクエリ文字列を使用して、キャッシュされた結果にアクセスできます。

```
g.with('Neptune#enableResultCache', true)
 .V().has('genre','drama').in('likes')
```

結果がキャッシュされてから 120 秒以上経過した場合、そのクエリは新しい結果を返し、 なしでキャッシュします time-to-live。

キャッシュされた結果にアクセスするには、enableResultCacheWithTTL クエリヒントで同じクエリを発行します。例:

```
g.with('Neptune#enableResultCacheWithTTL', 140)
 .V().has('genre','drama').in('likes')
```

120 秒が経過するまで (つまり、現在有効になっている TTL)、enableResultCacheWithTTL クエリヒントを使うこの新しいクエリは、キャッシュされた結果を返します。120 秒後、新しい結果が返され、140 秒 time-to-live の でキャッシュされます。

> **Note**
>
> クエリキーの結果がすでにキャッシュされている場合、 と同じクエリキーenableResultCacheWithTTLは新しい結果を生成せず、現在キャッシュされている結果 time-to-live の には影響しません。

- 結果が enableResultCache を使って以前にキャッシュされていた場合、enableResultCacheWithTTL が新しい結果を生成し、指定した TTL 用にキャッシュする前に、まずキャッシュをクリアする必要があります。
- 結果が enableResultCachewithTTL を使って以前にキャッシュされていた場合、enableResultCacheWithTTL が新しい結果を生成し、指定した TTL 用にキャッシュする前に、まず前の TTL を期限切れとする必要があります。

invalidateResultCacheKey を使用する

invalidateResultCacheKey クエリヒントを使って特定のクエリのキャッシュされた結果をクリアできます。例:

```
g.with('Neptune#invalidateResultCacheKey', true)
 .V().has('genre','drama').in('likes')
```

このクエリは、クエリキー、g.V().has('genre','drama').in('likes')、のキャッシュをクリアし、そのクエリの新しい結果を返します。

また、invalidateResultCacheKey と enableResultCache または enableResultCacheWithTTL を組み合わせることもできます。たとえば、次のクエリは、現在のキャッシュされた結果をクリアし、新しい結果をキャッシュして返します。

```
g.with('Neptune#enableResultCache', true)
 .with('Neptune#invalidateResultCacheKey', true)
 .V().has('genre','drama').in('likes')
```

invalidateResultCache を使用する

invalidateResultCache クエリヒントを使ってクエリ結果キャッシュのすべてのキャッシュされた結果をクリアできます。例:

```
g.with('Neptune#invalidateResultCache', true)
 .V().has('genre','drama').in('likes')
```

このクエリは、結果キャッシュ全体をクリアし、そのクエリの新しい結果を返します。

また、invalidateResultCache と enableResultCache または enableResultCacheWithTTL を組み合わせることもできます。たとえば、次のクエリは、結果キャッシュ全体をクリアし、このクエリの新しい結果をキャッシュして返します。

```
g.with('Neptune#enableResultCache', true)
 .with('Neptune#invalidateResultCache', true)
 .V().has('genre','drama').in('likes')
```

キャッシュされたクエリ結果のページ割り

次のような多数の結果をすでにキャッシュしているとします。

```
g.with('Neptune#enableResultCache', true)
 .V().has('genre','drama').in('likes')
```

それから、次の範囲クエリを発行するとします。

```
g.with('Neptune#enableResultCache', true)
 .V().has('genre','drama').in('likes').range(0,10)
```

Neptune は最初に完全なキャッシュキーを探します。つまり g.V().has('genre','drama').in('likes').range(0,10) です。そのキーが存在しない場合、Neptune は次に範囲のないクエリ文字列のキーがあるかどうかを調べます (すなわちg.V().has('genre','drama').in('likes'))。そのキーが見つかると、Neptune は範囲が指定しているように、キャッシュから最初の 10 個の結果を取得します。

> **Note**
>
> 末尾に範囲があるクエリの invalidateResultCacheKey クエリヒントを使うと、Neptune は、範囲とクエリと完全に一致するものが見つからない場合、範囲のないクエリのキャッシュをクリアします。

numResultsCachedと使用する.iterate()

numResultsCached クエリヒントを使用すると、キャッシュされているすべての結果を返さずに結果キャッシュに入力できます。これは、多数の結果をページ分割したい場合に便利です。

numResultsCached クエリヒントは、iterate() で終わるクエリでのみ機能します。

たとえば、サンプルクエリの最初の 50 の結果をキャッシュする場合は、次のようにします。

```
g.with("Neptune#enableResultCache", true)
 .with("Neptune#numResultsCached", 50)
 .V().has('genre','drama').in('likes').iterate()
```

この場合、キャッシュ内のクエリキーは次のようになります。`g.with("Neptune#numResultsCached", 50).V().has('genre','drama').in('likes')`。このクエリを使用して、キャッシュされた結果の最初の 10 個を取得できます。

```
g.with("Neptune#enableResultCache", true)
 .with("Neptune#numResultsCached", 50)
 .V().has('genre','drama').in('likes').range(0, 10)
```

また、次のように、クエリから次の 10 個の結果を取得できます。

```
g.with("Neptune#enableResultCache", true)
 .with("Neptune#numResultsCached", 50)
 .V().has('genre','drama').in('likes').range(10, 20)
```

忘れずに numResultsCached ヒントを含めます！これはクエリキーの不可欠な部分であるため、キャッシュされた結果にアクセスするには、これが存在していなければなりません。

numResultsCached 使用時には、心に留めておくべきことがいくつかあります。

- **numResultsCached** で入力する番号がクエリの最後に適用されます。 これは、たとえば、次のクエリが実際に結果を (1000, 1500) 範囲内にキャッシュしていることを意味します。

  ```
  g.with("Neptune#enableResultCache", true)
   .with("Neptune#numResultsCached", 500)
   .V().range(1000, 2000).iterate()
  ```

- **numResultsCached** で入力する番号はキャッシュする結果の最大数を指定します。 これは、たとえば、次のクエリが実際に結果を (1000, 2000) 範囲内にキャッシュしていることを意味します。

  ```
  g.with("Neptune#enableResultCache", true)
   .with("Neptune#numResultsCached", 100000)
  ```

```
 .V().range(1000, 2000).iterate()
```

- **.range().iterate()** で終わるクエリによってキャッシュされる結果には独自の範囲があります。 たとえば、次のようなクエリを使用して結果をキャッシュするとします。

```
g.with("Neptune#enableResultCache", true)
 .with("Neptune#numResultsCached", 500)
 .V().range(1000, 2000).iterate()
```

キャッシュから最初の 100 個の結果を取得するには、次のようなクエリを記述します。

```
g.with("Neptune#enableResultCache", true)
 .with("Neptune#numResultsCached", 500)
 .V().range(1000, 2000).range(0, 100)
```

この100の結果は、範囲 (1000, 1100) 内の基本クエリの結果と同等になります。

キャッシュされた結果を検索するために使用されるクエリキャッシュキー

クエリの結果がキャッシュされた後、後続のクエリは新しい結果を生成するのではなく、キャッシュからのクエリキャッシュキー検索結果と同じものになります。クエリのクエリキャッシュキーは、次のように評価されます。

1. numResultsCached 以外のキャッシュ関連のクエリヒントはすべて無視されます。
2. 最後の iterate() ステップは無視されます。
3. 残りのクエリは、バイトコード表現に従って順序付けられます。

結果の文字列は、すでにキャッシュ内にあるクエリ結果のインデックスと照合され、クエリにキャッシュヒットがあるかどうかを判断します。

例えば、次のようなクエリがあるとします。

```
g.withSideEffect('Neptune#typePromotion', false).with("Neptune#enableResultCache",
 true)
 .with("Neptune#numResultsCached", 50)
 .V().has('genre','drama').in('likes').iterate()
```

このクエリは、次のバイトコードバージョンとして保存されます。

```
g.withSideEffect('Neptune#typePromotion', false)
 .with("Neptune#numResultsCached", 50)
 .V().has('genre','drama').in('likes')
```

結果キャッシュに関連する例外

キャッシュしようとしているクエリの結果が大きすぎてキャッシュメモリに収まらない場合は、以前にキャッシュされたものをすべて削除した後でも、Neptune は QueryLimitExceededException 障害を見つけます。結果は返されず、例外によって次のエラーメッセージが生成されます。

```
The result size is larger than the allocated cache,
      please refer to results cache best practices for options to rerun the query.
```

noCacheExceptions クエリヒントを使って、次のようにこのメッセージを抑制できます。

```
g.with('Neptune#enableResultCache', true)
 .with('Neptune#noCacheExceptions', true)
 .V().has('genre','drama').in('likes')
```

Gremlin **mergeV()** および **mergeE()** ステップによる効率的なアップサートの実行

アップサート (または条件付き挿入) は、頂点やエッジが既に存在する場合は再利用し、存在しない場合は作成します。効率的なアップサートは、Gremlin クエリのパフォーマンスに大きな違いをもたらすことができます。

アップサートを使用すると、冪等挿入操作、つまり、そのような操作を何回実行しても、全体的な結果は同じ操作を記述できます。これは、グラフの同じ部分に同時に変更を加えると、1 つ以上のトランザクションが ConcurrentModificationException で強制的にロールバックされ、再試行が必要になるような、同時書き込みの多いシナリオで役立ちます。

例えば、次のクエリは、指定された Map を使用して、T.id が "v-1" の頂点を最初に探すことによって、頂点をアップサートします。その頂点が見つかると、その頂点が返されます。見つからなかった場合は、その id とプロパティを持つ頂点が onCreate 句によって作成されます。

```
g.mergeV([(id):'v-1']).
  option(onCreate, [(label): 'PERSON', 'email': 'person-1@example.org'])
```

アップサートをバッチ処理してスループットを向上させる

スループットの高い書き込みシナリオでは、`mergeV()` ステップと `mergeE()` ステップを連鎖して、頂点とエッジをバッチでアップサートできます。バッチ処理を行うと、多数の頂点やエッジをアップサートすることによるトランザクションのオーバーヘッドが軽減されます。その後、複数のクライアントを使用してバッチリクエストを並行して更新することで、スループットをさらに高めることができます。

経験則として、バッチリクエストごとに約 200 件のレコードをアップサートすることをお勧めします。レコードは 1 つの頂点、またはエッジラベル、またはプロパティです。例えば、1 つのラベルと 4 つのプロパティを持つ頂点では、5 つのレコードが作成されます。1 つのラベルと 1 つのプロパティを持つエッジでは、2 つのレコードが作成されます。それぞれ 1 つのラベルと 4 つのプロパティを持つ頂点のバッチをアップサートしたい場合、`200 / (1 + 4) = 40` のため、バッチサイズは 40 から始める必要があります。

バッチサイズを試してみることもできます。バッチあたり 200 レコードから始めるのが適切ですが、理想的なバッチサイズは作業負荷に応じて大きくなったり小さくなったりします。ただし、Neptune ではリクエストごとの Gremlin ステップの総数を制限する場合があることに注意してください。この制限は文書化されていませんが、念のため、リクエストに含まれるグレムリンステップ数は 1,500 を超えないようにしてください。Neptune は、1,500 ステップを超える大規模なバッチリクエストを拒否する場合があります。

スループットを高めるには、複数のクライアントを使用して、バッチを並行してアップサートできます (「効率的なマルチスレッドの Gremlin 書き込みの作成」を参照)。クライアントの数は、Neptune ライターインスタンスのワーカースレッドの数と同じである必要があり、通常、これはサーバー上の vCPU の数の 2 倍です。例えば、`r5.8xlarge` インスタンスには 32 個の vCPU と 64 個のワーカースレッドがあります。`r5.8xlarge` を使用する高スループットの書き込みシナリオでは、64 台のクライアントが Neptune にバッチアップサートを並行して書き込みます。

各クライアントはバッチリクエストを送信し、リクエストが完了するのを待ってから、別のリクエストを送信する必要があります。複数のクライアントは並行して実行されますが、個々のクライアントはそれぞれ順番にリクエストを送信します。これにより、サーバー側のリクエストキューがフラッディングすることなく、すべてのワーカースレッドを占有するリクエストのストリームがサーバーに安定して供給されます (「Neptune DB クラスターでの DB インスタンスのサイジング」を参照)。

複数のトラバーサーを生成するステップは避ける

Gremlin ステップを実行すると、入力トラバーサーが 1 つ取得され、1 つ以上の出力トラバーサーが出力されます。1 つのステップで発生するトラバーサーの数によって、次のステップが実行される回数が決まります。

通常、バッチ操作を実行するときは、頂点 A のアップサートなどの各操作を 1 回実行する必要があります。これにより、一連の操作は、頂点 A をアップサート、頂点 B をアップサート、頂点 C をアップサートするというようになります。ステップが 1 つの要素のみを作成または変更する限り、そのステップはトラバーサーを 1 つだけ出力し、次の操作を表すステップは 1 回だけ実行されます。一方、1 つの操作で複数の要素を作成または変更すると、複数のトラバーサーが出力され、それ以降のステップは、発行されたトラバーサーごとに 1 回ずつ、というように複数回実行されます。その結果、データベースが不必要な追加作業を行うことになり、場合によっては不要な頂点、エッジ、またはプロパティ値が作成されることもあります。

問題が発生する例としては、`g.V().addV()` のようなクエリがあります。この単純なクエリは、グラフ内の頂点ごとに頂点を追加します。これは、`V()` がグラフ内の頂点ごとにトラバーサーを発行し、それらのトラバーサーのそれぞれが `addV()` への呼び出しをトリガーするためです。

複数のトラバーサーを発行する操作を処理する方法については、「アップサートと挿入の混合」を参照してください。

頂点のアップサート

`mergeV()` ステップは、頂点をアップサートするために特別に設計されています。グラフ内の既存の頂点と一致する要素を表す `Map` を引数として取り、要素が見つからない場合は、その `Map` を使用して新しい頂点を作成します。このステップでは、作成時やマッチ時の動作を変更することも可能であり、`option()` モジュレータに `Merge.onCreate` および `Merge.onMatch` トークンを適用して、それぞれの動作を制御できます。このステップの使用方法の詳細については、 TinkerPop リファレンスドキュメントを参照してください。

頂点 ID を使用して、特定の頂点が存在するかどうかを判断できます。Neptune では ID に関する同時実行の多いユースケースに合わせてアップサートを最適化するため、この方法が推奨されます。例として、次のクエリは、特定の頂点 ID を持つ頂点がまだ存在しない場合はその頂点を作成し、存在する場合はそれを再利用します。

```
g.mergeV([(T.id): 'v-1']).
    option(onCreate, [(T.label): 'PERSON', email: 'person-1@example.org', age: 21]).
    option(onMatch, [age: 22]).
```

```
  id()
```

このクエリは id() ステップで終わることに注意してください。頂点をアップサートする目的では必ずしも必要ではありませんが、アップサートクエリの最後に id() ステップを追加することで、サーバーはすべての頂点プロパティをクライアントにシリアル化し直さずに済むため、クエリのロックコストを削減できます。

あるいは、頂点プロパティを使用して頂点を識別することもできます。

```
g.mergeV([email: 'person-1@example.org']).
    option(onCreate, [(T.label): 'PERSON', age: 21]).
    option(onMatch, [age: 22]).
  id()
```

可能であれば、ユーザー指定の独自の ID を使用して頂点を作成し、これらの ID を使用してアップサート操作中に頂点が存在するかどうかを判断します。これにより、Neptune はアップサートを最適化できます。ID ベースのアップサートは、同時変更が頻繁に行われる場合、プロパティベースのアップサートよりもはるかに効率的です。

頂点アップサートの連鎖

頂点アップサートを連鎖して、まとめて挿入することができます。

```
g.V('v-1')
 .fold()
 .coalesce(unfold(),
           addV('Person').property(id, 'v-1')
                         .property('email', 'person-1@example.org'))
 .V('v-2')
 .fold()
 .coalesce(unfold(),
           addV('Person').property(id, 'v-2')
                         .property('email', 'person-2@example.org'))
 .V('v-3')
 .fold()
 .coalesce(unfold(),
           addV('Person').property(id, 'v-3')
                         .property('email', 'person-3@example.org'))
 .id()
```

または、次の mergeV() 構文を使用することもできます。

```
g.mergeV([(T.id): 'v-1', (T.label): 'PERSON', email: 'person-1@example.org']).
  mergeV([(T.id): 'v-2', (T.label): 'PERSON', email: 'person-2@example.org']).
  mergeV([(T.id): 'v-3', (T.label): 'PERSON', email: 'person-3@example.org'])
```

ただし、この形式のクエリには、id による基本的な検索方法では不要な要素が検索条件に含まれているため、前のクエリほど効率的ではありません。

エッジのアップサート

mergeE() ステップは、エッジをアップサートするために特別に設計されています。グラフ内の既存のエッジと一致する要素を表す Map を引数として取り、要素が見つからない場合は、その Map を使用して新しいエッジを作成します。このステップでは、作成時やマッチ時の動作を変更することも可能であり、option() モジュレータに Merge.onCreate および Merge.onMatch トークンを適用して、それぞれの動作を制御できます。このステップの使用方法の詳細については、 TinkerPop リファレンスドキュメントを参照してください。

カスタム頂点 ID を使用して頂点をアップサートするのと同じ方法で、エッジ ID を使用してエッジをアップサートできます。繰り返しますが、Neptune がクエリを最適化できるようになるため、この方法が推奨されます。例えば、次のクエリは、エッジ ID がまだ存在しない場合はそのエッジ ID に基づいてエッジを作成し、存在する場合は再利用します。このクエリでは、新しいエッジを作成する必要がある場合には Direction.from および Direction.to 頂点の ID も使用します。

```
g.mergeE([(T.id): 'e-1']).
    option(onCreate, [(from): 'v-1', (to): 'v-2', weight: 1.0]).
    option(onMatch, [weight: 0.5]).
  id()
```

このクエリは id() ステップで終わることに注意してください。エッジをアップサートする目的では必ずしも必要ではありませんが、アップサートクエリの最後に id() ステップを追加することで、サーバーはすべてのエッジプロパティをクライアントにシリアル化し直さずに済むため、クエリのロックコストを削減できます。

多くのアプリケーションはカスタム頂点 ID を使用しますが、エッジ ID の生成は Neptune に任せています。エッジの ID はわからないが、from および to 頂点 ID はわかっている場合は、次のようなクエリを使用して、エッジをアップサートできます。

```
g.mergeE([(from): 'v-1', (to): 'v-2', (T.label): 'KNOWS']).
  id()
```

ステップで エッジを作成するには、mergeE() によって参照されるすべての頂点が存在している必要があります。

エッジのアップサートの連鎖

頂点アップサートの場合と同様、mergeE() ステップを連鎖して、バッチリクエストにするのは簡単です。

```
g.mergeE([(from): 'v-1', (to): 'v-2', (T.label): 'KNOWS']).
  mergeE([(from): 'v-2', (to): 'v-3', (T.label): 'KNOWS']).
  mergeE([(from): 'v-3', (to): 'v-4', (T.label): 'KNOWS']).
  id()
```

頂点とエッジのアップサートの組み合わせ

頂点とそれらを接続するエッジの両方をアップサートしたい場合があります。ここで紹介したバッチサンプルをミックスしてもかまいません。次の例では、3 つの頂点と 2 つのエッジをアップサートしています。

```
g.mergeV([(id):'v-1']).
    option(onCreate, [(label): 'PERSON', 'email': 'person-1@example.org']).
  mergeV([(id):'v-2']).
    option(onCreate, [(label): 'PERSON', 'email': 'person-2@example.org']).
  mergeV([(id):'v-3']).
    option(onCreate, [(label): 'PERSON', 'email': 'person-3@example.org']).
  mergeE([(from): 'v-1', (to): 'v-2', (T.label): 'KNOWS']).
  mergeE([(from): 'v-2', (to): 'v-3', (T.label): 'KNOWS']).
 id()
```

アップサートと挿入の混合

頂点とそれらを接続するエッジの両方をアップサートしたい場合があります。ここで紹介したバッチサンプルをミックスしてもかまいません。次の例では、3 つの頂点と 2 つのエッジをアップサートしています。

アップサートは通常、一度に 1 つの要素を処理します。ここで説明したアップサートパターンに従うと、アップサート操作ごとにトラバーサーが 1 回発生し、それ以降の操作は 1 回だけ実行されます。

ただし、アップサートと挿入を混在させたい場合もあります。例えば、エッジを使用してアクションやイベントのインスタンスを表す場合などが該当します。リクエストでは、必要な頂点がすべて存在

することを確認するためにアップサートを使用し、エッジを追加するためにインサートを使用する場合があります。この種のリクエストでは、各操作で発生する可能性のあるトラバーサーの数に注意してください。

アップサートとインサートを組み合わせて、イベントを表すエッジをグラフに追加する次の例を考えてみましょう。

```
// Fully optimized, but inserts too many edges
g.mergeV([(id):'v-1']).
    option(onCreate, [(label): 'PERSON', 'email': 'person-1@example.org']).
  mergeV([(id):'v-2']).
    option(onCreate, [(label): 'PERSON', 'email': 'person-2@example.org']).
  mergeV([(id):'v-3']).
    option(onCreate, [(label): 'PERSON', 'email': 'person-3@example.org']).
  mergeV([(T.id): 'c-1', (T.label): 'CITY', name: 'city-1']).
  V('p-1', 'p-2').
  addE('FOLLOWED').to(V('p-1')).
  V('p-1', 'p-2', 'p-3').
  addE('VISITED').to(V('c-1')).
  id()
```

クエリでは 5 つのエッジを挿入する必要があります。2 つは FOLLOWED エッジであり、3 つは VISITED エッジです。ただし、記述されているクエリでは、8 つのエッジが挿入されます (2 つは FOLLOWED、6 つは VISITED)。これは、2 つの FOLLOWED エッジを挿入する操作で 2 つのトラバーサーが発生し、3 つのエッジを挿入する後続の挿入操作が 2 回実行されるためです。

この問題を解決するには、複数のトラバーサーが発生する可能性のある各操作の後に `fold()` ステップを追加します。

```
g.mergeV([(T.id): 'v-1', (T.label): 'PERSON', email: 'person-1@example.org']).
  mergeV([(T.id): 'v-2', (T.label): 'PERSON', email: 'person-2@example.org']).
  mergeV([(T.id): 'v-3', (T.label): 'PERSON', email: 'person-3@example.org']).
  mergeV([(T.id): 'c-1', (T.label): 'CITY', name: 'city-1']).
  V('p-1', 'p-2').
  addE('FOLLOWED').
    to(V('p-1')).
  fold().
  V('p-1', 'p-2', 'p-3').
  addE('VISITED').
    to(V('c-1')).
  id()
```

ここでは、FOLLOWED エッジを挿入する操作の後に、fold() ステップを挿入しました。これにより、1 つのトラバーサーが発生し、それ以降の操作は 1 回だけ実行されます。

この方法の欠点は、fold() が最適化されていないため、クエリが完全には最適化されないことです。fold() の後に続く挿入操作も最適化されなくなってしまいます。

fold() を使用して、後続のステップの代わりにトラバーサーの数を減らす必要がある場合は、最もコストの低いものがクエリの最適化されていない部分を占めるように操作を順序付けてください。

カーディナリティの設定

Neptune の頂点プロパティのデフォルトのカーディナリティが設定されています。つまり、 mergeV() を使用する場合、マップで指定された値にはすべてそのカーディナリティが与えられます。単一基数を使用するには、その使用法が明示的である必要があります。 TinkerPop 3.7.0 以降、次の例に示すように、カーディナリティをマップの一部として指定できる新しい構文があります。

```
g.mergeV([(T.id): 1234]).
  option(onMatch, ['age': single(20), 'name': single('alice'), 'city': set('miami')])
```

または、option次のようにカーディナリティをデフォルトとして設定することもできます。

```
// age and name are set to single cardinality by default
g.mergeV([(T.id): 1234]).
  option(onMatch, ['age': 22, 'name': 'alice', 'city': set('boston')], single)
```

バージョン 3.7.0 mergeV()より前の でカーディナリティを設定するオプションが少なくなります。一般的なアプローチは、次のようにproperty()ステップにフォールバックすることです。

```
g.mergeV([(T.id): '1234']).
  option(onMatch, sideEffect(property(single,'age', 20).
  property(set,'city','miami')).constant([:]))
```

> **Note**
>
> このアプローチは、開始ステップで使用されるmergeV()場合にのみ で機能します。したがって、この構文を使用する開始ステップのmergeV()後に最初に 1 つのトラバーサルmergeV()内で連鎖することはできません。受信トラバーサーがグラフ要素の場合、エラーが発生します。この場合、mergeV()呼び出しを複数のリクエストに分割し、それぞれを開始ステップにすることができます。

fold()/coalesce()/unfold() による効率的な Gremlin アップサートの実行

アップサート (または条件付き挿入) は、頂点やエッジが既に存在する場合は再利用し、存在しない場合は作成します。効率的なアップサートは、Gremlin クエリのパフォーマンスに大きな違いをもたらすことができます。

このページでは、fold()/coalesce()/unfold() Gremlin パターンを使用して効率的なアップサートを行う方法を説明します。ただし、エンジン TinkerPop バージョン 1.2.1.0 で Neptune で導入されたバージョン 3.6.x のリリースでは、ほとんどの場合、新しい mergeV()および mergeE() ステップが推奨されます。ここで説明する fold()/coalesce()/unfold() パターンは、一部の複雑な状況ではまだ役に立つかもしれませんが、一般的な用途では、Gremlin mergeV() および mergeE() ステップによる効率的なアップサートの実行 で説明されているように、できれば mergeV() および mergeE() を使用してください。

アップサートを使用すると、冪等挿入操作、つまり、そのような操作を何回実行しても、全体的な結果は同じ操作を記述できます。これは、グラフの同じ部分に同時に変更を加えると、1 つ以上のトランザクションが強制的に ConcurrentModificationException でロールバックされ、再試行が必要になるような、同時書き込みの多いシナリオに役立ちます。

例えば、次のクエリは、最初にデータセット内の指定された頂点を探し、その結果をリストにまとめることで頂点を更新します。coalesce() ステップで最初に指定されたトラバーサルで、クエリはこのリストを展開します。展開されたリストが空でない場合、結果は coalesce() から出力されます。ただし、頂点が現在存在しないために unfold() が空のコレクションを返した場合、coalesce() は、その頂点が指定された 2 番目のトラバーサルの評価に移り、この 2 番目のトラバーサルで、クエリは欠落している頂点を作成します。

```
g.V('v-1').fold()
          .coalesce(
             unfold(),
             addV('Person').property(id, 'v-1')
                           .property('email', 'person-1@example.org')
           )
```

アップサートには最適化された coalesce() の形式を使用する

Neptune は fold().coalesce(unfold(), ...) イディオムを最適化して高スループットの更新を行うことができますが、この最適化は、coalesce() の両方の部分が頂点またはエッジのいずれかを返し、それ以外は何も返さない場合にのみ機能します。coalesce() のいずれかの部分からプ

ロパティなど異なるものを返そうとした場合、Neptune の最適化は行われません。クエリは成功するかもしれませんが、特に大きなデータセットに対しては、最適化されたバージョンほどには機能しません。

最適化されていないアップサートクエリは実行時間を増加させ、スループットを低下させるため、Gremlin `explain` エンドポイントを使用して Upsert クエリが完全に最適化されているかどうかを判断する価値があります。`explain` プランを見直すときは、`+ not converted into Neptune steps` と `WARNING: >>` で始まる行を探してください。例:

```
+ not converted into Neptune steps: [FoldStep, CoalesceStep([[UnfoldStep],
 [AddEdgeSte...
WARNING: >> FoldStep << is not supported natively yet
```

これらの警告は、クエリが完全に最適化されない原因となっている部分を特定するのに役立ちます。

クエリを完全に最適化できない場合もあります。このような場合は、最適化できないステップをクエリの最後に配置して、エンジンができるだけ多くのステップを最適化できるようにする必要があります。この手法はバッチアップサートの例の一部で使用されています。バッチアップサートの例では、ある頂点またはエッジのセットに対して最適化されたアップサートをすべて実行してから、最適化されていない可能性のある追加の修正を同じ頂点またはエッジに適用します。

アップサートをバッチ処理してスループットを向上させる

高スループットの書き込みシナリオでは、アップサートのステップを連鎖させて頂点とエッジをバッチでアップサートできます。バッチ処理を行うと、多数の頂点やエッジをアップサートすることによるトランザクションのオーバーヘッドが軽減されます。その後、複数のクライアントを使用してバッチリクエストを並行して更新することで、スループットをさらに高めることができます。

経験則として、バッチリクエストごとに約 200 件のレコードをアップサートすることをお勧めします。レコードは 1 つの頂点、またはエッジラベル、またはプロパティです。例えば、1 つのラベルと 4 つのプロパティを持つ頂点では、5 つのレコードが作成されます。1 つのラベルと 1 つのプロパティを持つエッジでは、2 つのレコードが作成されます。それぞれ 1 つのラベルと 4 つのプロパティを持つ頂点のバッチをアップサートしたい場合、`200 / (1 + 4) = 40` のため、バッチサイズは 40 から始める必要があります。

バッチサイズを試してみることもできます。バッチあたり 200 レコードから始めるのが適切ですが、理想的なバッチサイズは作業負荷に応じて大きくなったり小さくなったりします。ただし、Neptune ではリクエストごとの Gremlin ステップの総数を制限する場合があることに注意してください。この制限は文書化されていませんが、念のため、リクエストに含まれるグレムリンステッ

プ数は 1500 を超えないようにしてください。Neptune は、1,500 ステップを超える大規模なバッチリクエストを拒否する場合があります。

スループットを高めるには、複数のクライアントを使用して、バッチを並行してアップサートできます (「効率的なマルチスレッドの Gremlin 書き込みの作成」を参照)。クライアントの数は、Neptune ライターインスタンスのワーカースレッドの数と同じである必要があり、通常、これはサーバー上の vCPU の数の 2 倍です。例えば、`r5.8xlarge` インスタンスには 32 個の vCPU と 64 個のワーカースレッドがあります。`r5.8xlarge` を使用する高スループットの書き込みシナリオでは、64 台のクライアントが Neptune にバッチアップサートを並行して書き込みます。

各クライアントはバッチリクエストを送信し、リクエストが完了するのを待ってから、別のリクエストを送信する必要があります。複数のクライアントは並行して実行されますが、個々のクライアントはそれぞれ順番にリクエストを送信します。これにより、サーバー側のリクエストキューがフラッディングすることなく、すべてのワーカースレッドを占有するリクエストのストリームがサーバーに安定して供給されます (「Neptune DB クラスターでの DB インスタンスのサイジング」を参照)。

複数のトラバーサーを生成するステップは避ける

Gremlin ステップを実行すると、入力トラバーサーが 1 つ取得され、1 つ以上の出力トラバーサーが出力されます。1 つのステップで発生するトラバーサーの数によって、次のステップが実行される回数が決まります。

通常、バッチ操作を実行するときは、頂点 A のアップサートなどの各操作を 1 回実行する必要があります。これにより、一連の操作は、頂点 A をアップサート、頂点 B をアップサート、頂点 C をアップサートするというようになります。ステップが 1 つの要素のみを作成または変更する限り、そのステップはトラバーサーを 1 つだけ出力し、次の操作を表すステップは 1 回だけ実行されます。一方、1 つの操作で複数の要素を作成または変更すると、複数のトラバーサーが出力され、それ以降のステップは、発行されたトラバーサーごとに 1 回ずつ、というように複数回実行されます。その結果、データベースが不必要な追加作業を行うことになり、場合によっては不要な頂点、エッジ、またはプロパティ値が作成されることもあります。

問題が発生する例としては、`g.V().addV()` のようなクエリがあります。この単純なクエリは、グラフ内の頂点ごとに頂点を追加します。これは、`V()` がグラフ内の頂点ごとにトラバーサーを発行し、それらのトラバーサーのそれぞれが `addV()` への呼び出しをトリガーするためです。

複数のトラバーサーを発行する操作を処理する方法については、「アップサートと挿入の混合」を参照してください。

頂点のアップサート

頂点 ID を使用して、対応する頂点が存在するかどうかを判断できます。Neptune では ID に関する同時実行の多いユースケースに合わせてアップサートを最適化するため、この方法が推奨されます。例として、次のクエリは、特定の頂点 ID を持つ頂点がまだ存在しない場合はその頂点を作成し、存在する場合はそれを再利用します。

```
g.V('v-1')
 .fold()
  .coalesce(unfold(),
            addV('Person').property(id, 'v-1')
                          .property('email', 'person-1@example.org'))
  .id()
```

このクエリは id() ステップで終わることに注意してください。頂点をアップサートする目的では必ずしも必要ではありませんが、アップサートクエリの最後に id() ステップを追加すると、サーバーはすべての頂点プロパティをクライアントにシリアル化し直さなくなるため、クエリのロックコストを削減できます。

あるいは、頂点プロパティを使用して頂点が存在するかどうかを調べることもできます。

```
g.V()
 .hasLabel('Person')
 .has('email', 'person-1@example.org')
 .fold()
 .coalesce(unfold(),
           addV('Person').property('email', 'person-1@example.org'))
 .id()
```

可能であれば、ユーザー指定の独自の ID を使用して頂点を作成し、これらの ID を使用してアップサート操作中に頂点が存在するかどうかを判断します。これにより、Neptune は ID 周辺のアップサートを最適化できます。ID ベースのアップサートは、同時変更が頻繁に行われる場合、プロパティベースのアップサートよりもはるかに効率的です。

頂点アップサートの連鎖

頂点アップサートを連鎖して、まとめて挿入することができます。

```
g.V('v-1')
 .fold()
 .coalesce(unfold(),
```

```
          addV('Person').property(id, 'v-1')
                        .property('email', 'person-1@example.org'))
 .V('v-2')
 .fold()
 .coalesce(unfold(),
          addV('Person').property(id, 'v-2')
                        .property('email', 'person-2@example.org'))
 .V('v-3')
 .fold()
 .coalesce(unfold(),
          addV('Person').property(id, 'v-3')
                        .property('email', 'person-3@example.org'))
 .id()
```

エッジのアップサート

カスタム頂点 ID を使用して頂点をアップサートするのと同じ方法で、エッジ ID を使用してエッジをアップサートできます。繰り返しますが、Neptune がクエリを最適化できるようになるため、この方法が推奨されます。例えば、次のクエリは、エッジ ID がまだ存在しない場合はその エッジ ID に基づいてエッジを作成し、存在する場合は再利用します。新しいエッジを作成する必要がある場合、クエリは `from` および `to` 頂点の ID も使用します。

```
g.E('e-1')
 .fold()
 .coalesce(unfold(),
          addE('KNOWS').from(V('v-1'))
                       .to(V('v-2'))
                       .property(id, 'e-1'))
 .id()
```

多くのアプリケーションはカスタム頂点 ID を使用しますが、エッジ ID の生成は Neptune に任せています。エッジの ID はわからないが、`from` および `to` 頂点 ID はわかっている場合は、次のような式を使用して、エッジをアップサートできます。

```
g.V('v-1')
 .outE('KNOWS')
 .where(inV().hasId('v-2'))
 .fold()
 .coalesce(unfold(),
          addE('KNOWS').from(V('v-1'))
                       .to(V('v-2')))
```

```
  .id()
```

where() 句の頂点ステップは otherV() ではなく inV() (または、inE() を使用してエッジを見つけた場合は outV()) でなければならないことに注意してください。ここでは otherV() を使用しないでください。使用すると、クエリが最適化されず、パフォーマンスが低下します。例えば、Neptune は次のクエリを最適化しません。

```
// Unoptimized upsert, because of otherV()
g.V('v-1')
 .outE('KNOWS')
 .where(otherV().hasId('v-2'))
 .fold()
 .coalesce(unfold(),
           addE('KNOWS').from(V('v-1'))
                        .to(V('v-2')))
 .id()
```

エッジまたは頂点 ID が事前にわからない場合は、頂点プロパティを使用してアップサートできます。

```
g.V()
 .hasLabel('Person')
 .has('name', 'person-1')
 .outE('LIVES_IN')
 .where(inV().hasLabel('City').has('name', 'city-1'))
 .fold()
 .coalesce(unfold(),
           addE('LIVES_IN').from(V().hasLabel('Person')
                                    .has('name', 'person-1'))
                           .to(V().hasLabel('City')
                                  .has('name', 'city-1')))
 .id()
```

頂点アップサートと同様に、Neptune がアップサートを完全に最適化できるように、プロパティベースのアップサートよりも、エッジ ID または from および to 頂点 ID を使用する ID ベースのエッジアップサートを使用する方が好ましいです。

from および to 頂点の有無の確認

新しいエッジを作成するステップの構造 addE().from().to() に注意してください。この構造により、クエリは from と to の両方の頂点の存在を確実にチェックします。これらのいずれかが存在しない場合、クエリは次のようなエラーを返します。

```
{
  "detailedMessage": "Encountered a traverser that does not map to a value for child...
  "code": "IllegalArgumentException",
  "requestId": "..."
}
```

from または to のいずれかの頂点が存在しない可能性がある場合は、頂点間のエッジをアップセットする前に、頂点をアップサートしてみてください。頂点とエッジのアップサートの組み合わせ を参照してください。

エッジを作成するには、使用すべきではない別の方法 V().addE().to() があります。from 頂点が存在する場合にのみ、エッジを追加します。to 頂点が存在しない場合、前述のようにクエリはエラーを生成しますが、from 頂点が存在しない場合、エラーを発生せずに、エッジの挿入に失敗します。例えば、次のアップサートは、from 頂点が存在しない場合、エッジをアップサートせずに完了します。

```
// Will not insert edge if from vertex does not exist
g.V('v-1')
 .outE('KNOWS')
 .where(inV().hasId('v-2'))
 .fold()
 .coalesce(unfold(),
           V('v-1').addE('KNOWS')
                   .to(V('v-2')))
 .id()
```

エッジのアップサートの連鎖

エッジアップサートを連結してバッチリクエストを作成する場合は、エッジ ID がわかっていても、各アップサートを頂点ルックアップから始める必要があります。

アップサートするエッジの ID と、from および to 頂点の ID がわかっている場合は、次の計算式を使用できます。

```
g.V('v-1')
```

```
  .outE('KNOWS')
  .hasId('e-1')
  .fold()
  .coalesce(unfold(),
            V('v-1').addE('KNOWS')
                    .to(V('v-2'))
                    .property(id, 'e-1'))
  .V('v-3')
  .outE('KNOWS')
  .hasId('e-2').fold()
  .coalesce(unfold(),
            V('v-3').addE('KNOWS')
                    .to(V('v-4'))
                    .property(id, 'e-2'))
  .V('v-5')
  .outE('KNOWS')
  .hasId('e-3')
  .fold()
  .coalesce(unfold(),
            V('v-5').addE('KNOWS')
                    .to(V('v-6'))
                    .property(id, 'e-3'))
  .id()
```

おそらく最も一般的なバッチエッジアップサートのシナリオは、from および to 頂点 ID はわかっているが、アップサートするエッジの ID はわからないというものです。その場合は、次の式を使用してください。

```
g.V('v-1')
  .outE('KNOWS')
  .where(inV().hasId('v-2'))
  .fold()
  .coalesce(unfold(),
            V('v-1').addE('KNOWS')
                    .to(V('v-2')))

  .V('v-3')
  .outE('KNOWS')
  .where(inV().hasId('v-4'))
  .fold()
  .coalesce(unfold(),
            V('v-3').addE('KNOWS')
                    .to(V('v-4')))
```

```
  .V('v-5')
  .outE('KNOWS')
  .where(inV().hasId('v-6'))
  .fold()
  .coalesce(unfold(),
            V('v-5').addE('KNOWS').to(V('v-6')))
  .id()
```

アップサートするエッジの ID はわかっているが、from および to 頂点の ID はわからない場合は、次の計算式を使用できます。

```
g.V()
  .hasLabel('Person')
  .has('email', 'person-1@example.org')
  .outE('KNOWS')
  .hasId('e-1')
  .fold()
  .coalesce(unfold(),
            V().hasLabel('Person')
               .has('email', 'person-1@example.org')
               .addE('KNOWS')
               .to(V().hasLabel('Person')
                      .has('email', 'person-2@example.org'))
                      .property(id, 'e-1'))
  .V()
  .hasLabel('Person')
  .has('email', 'person-3@example.org')
  .outE('KNOWS')
  .hasId('e-2')
  .fold()
  .coalesce(unfold(),
            V().hasLabel('Person')
               .has('email', 'person-3@example.org')
               .addE('KNOWS')
               .to(V().hasLabel('Person')
                      .has('email', 'person-4@example.org'))
               .property(id, 'e-2'))
  .V()
  .hasLabel('Person')
  .has('email', 'person-5@example.org')
  .outE('KNOWS')
  .hasId('e-1')
  .fold()
```

```
 .coalesce(unfold(),
           V().hasLabel('Person')
              .has('email', 'person-5@example.org')
              .addE('KNOWS')
              .to(V().hasLabel('Person')
                     .has('email', 'person-6@example.org'))
                     .property(id, 'e-3'))
 .id()
```

頂点と エッジのアップサートの組み合わせ

頂点とそれらを接続する エッジの両方をアップサートしたい場合があります。ここで紹介したバッチサンプルをミックスしてもかまいません。次の例では、3 つの頂点と 2 つの エッジをアップサートしています。

```
g.V('p-1')
 .fold()
 .coalesce(unfold(),
           addV('Person').property(id, 'p-1')
                         .property('email', 'person-1@example.org'))
 .V('p-2')
 .fold()
 .coalesce(unfold(),
           addV('Person').property(id, 'p-2')
                         .property('name', 'person-2@example.org'))
 .V('c-1')
 .fold()
 .coalesce(unfold(),
           addV('City').property(id, 'c-1')
                       .property('name', 'city-1'))
 .V('p-1')
 .outE('LIVES_IN')
 .where(inV().hasId('c-1'))
 .fold()
 .coalesce(unfold(),
           V('p-1').addE('LIVES_IN')
                   .to(V('c-1')))
 .V('p-2')
 .outE('LIVES_IN')
 .where(inV().hasId('c-1'))
 .fold()
 .coalesce(unfold(),
           V('p-2').addE('LIVES_IN')
```

```
                    .to(V('c-1')))
  .id()
```

アップサートと挿入の混合

頂点とそれらを接続するエッジの両方をアップサートしたい場合があります。ここで紹介したバッチサンプルをミックスしてもかまいません。次の例では、3 つの頂点と 2 つのエッジをアップサートしています。

アップサートは通常、一度に 1 つの要素を処理します。ここで説明したアップサートパターンに従うと、アップサート操作ごとにトラバーサーが 1 回発生し、それ以降の操作は 1 回だけ実行されます。

ただし、アップサートと挿入を混在させたい場合もあります。例えば、エッジを使用してアクションやイベントのインスタンスを表す場合などが該当します。リクエストでは、必要な頂点がすべて存在することを確認するためにアップサートを使用し、エッジを追加するためにインサートを使用する場合があります。この種のリクエストでは、各操作で発生する可能性のあるトラバーサーの数に注意してください。

アップサートとインサートを組み合わせて、イベントを表すエッジをグラフに追加する次の例を考えてみましょう。

```
// Fully optimized, but inserts too many edges
g.V('p-1')
  .fold()
  .coalesce(unfold(),
            addV('Person').property(id, 'p-1')
                          .property('email', 'person-1@example.org'))
  .V('p-2')
  .fold()
  .coalesce(unfold(),
            addV('Person').property(id, 'p-2')
                          .property('name', 'person-2@example.org'))
  .V('p-3')
  .fold()
  .coalesce(unfold(),
            addV('Person').property(id, 'p-3')
                          .property('name', 'person-3@example.org'))
  .V('c-1')
  .fold()
```

```
  .coalesce(unfold(),
            addV('City').property(id, 'c-1')
                        .property('name', 'city-1'))
  .V('p-1', 'p-2')
  .addE('FOLLOWED')
  .to(V('p-1'))
  .V('p-1', 'p-2', 'p-3')
  .addE('VISITED')
  .to(V('c-1'))
  .id()
```

クエリでは 5 つのエッジを挿入する必要があります。2 つは FOLLOWED エッジであり、3 つは VISITED エッジです。ただし、記述されているクエリでは、8 つのエッジが挿入されます (2 つは FOLLOWED、6 つは VISITED)。これは、2 つの FOLLOWED エッジを挿入する操作で 2 つのトラバーサーが発生し、3 つのエッジを挿入する後続の挿入操作が 2 回実行されるためです。

この問題を解決するには、複数のトラバーサーが発生する可能性のある各操作の後に `fold()` ステップを追加します。

```
g.V('p-1')
  .fold()
  .coalesce(unfold(),
            addV('Person').property(id, 'p-1')
                          .property('email', 'person-1@example.org'))
  .V('p-2')
  .fold()
  .coalesce(unfold(),
            addV('Person').property(id, 'p-2').
                          .property('name', 'person-2@example.org'))
  .V('p-3')
  .fold()
  .coalesce(unfold(),
            addV('Person').property(id, 'p-3').
                          .property('name', 'person-3@example.org'))
  .V('c-1')
  .fold().
  .coalesce(unfold(),
             addV('City').property(id, 'c-1').
                         .property('name', 'city-1'))
  .V('p-1', 'p-2')
  .addE('FOLLOWED')
  .to(V('p-1'))
  .fold()
```

```
  .V('p-1', 'p-2', 'p-3')
  .addE('VISITED')
  .to(V('c-1')).
  .id()
```

ここでは、FOLLOWED エッジを挿入する操作の後に、fold() ステップを挿入しました。これにより、1 つのトラバーサーが発生し、それ以降の操作は 1 回だけ実行されます。

この方法の欠点は、fold() が最適化されていないため、クエリが完全には最適化されないことです。fold() の後に続く挿入操作は最適化されなくなります。

fold() を使用して、後続のステップの代わりにトラバーサーの数を減らす必要がある場合は、最もコストの低いものがクエリの最適化されていない部分を占めるように操作を順序付けてください。

既存の頂点とエッジを変更するアップサート

頂点やエッジが存在しない場合は作成し、その頂点やエッジが新規か既存かに関わらず、プロパティを追加または更新したい場合があります。

プロパティを追加または変更するには、property() ステップを使用します。このステップは coalesce() ステップの外部で使用してください。coalesce() ステップ内で既存の頂点またはエッジのプロパティを変更しようとすると、クエリはクエリエンジンによって最適化されない場合があります。

次のクエリは、アップサートされた各頂点のカウンタープロパティを追加または更新します。各 property() ステップには単一のカーディナリティがあり、新しい値が既存の値のセットに追加されるのではなく、既存の値と置き換えられるようにします。

```
g.V('v-1')
  .fold()
  .coalesce(unfold(),
            addV('Person').property(id, 'v-1')
                          .property('email', 'person-1@example.org'))
  .property(single, 'counter', 1)
  .V('v-2')
  .fold()
  .coalesce(unfold(),
            addV('Person').property(id, 'v-2')
                          .property('email', 'person-2@example.org'))
  .property(single, 'counter', 2)
  .V('v-3')
```

```
  .fold()
  .coalesce(unfold(),
            addV('Person').property(id, 'v-3')
                          .property('email', 'person-3@example.org'))
  .property(single, 'counter', 3)
  .id()
```

lastUpdated タイムスタンプ値など、アップサートされたすべての要素に適用されるプロパティ値がある場合は、クエリの最後にそのプロパティ値を追加または更新できます。

```
g.V('v-1')
  .fold()
  .coalesce(unfold(),
            addV('Person').property(id, 'v-1')
                          .property('email', 'person-1@example.org'))
  .V('v-2').
  .fold().
  .coalesce(unfold(),
            addV('Person').property(id, 'v-2')
                          .property('email', 'person-2@example.org'))
  .V('v-3')
  .fold()
  .coalesce(unfold(),
            addV('Person').property(id, 'v-3')
                          .property('email', 'person-3@example.org'))
  .V('v-1', 'v-2', 'v-3')
  .property(single, 'lastUpdated', datetime('2020-02-08'))
  .id()
```

頂点やエッジをさらに変更するかどうかを決める条件が他にもある場合は、has() ステップを使用して、変更を適用する要素をフィルタリングできます。次の例では、has() ステップを使用して、アップサートされた頂点を version プロパティの値に基づいてフィルタリングしています。次に、クエリは version が 3 未満のすべての頂点の version を 3 倍に更新します。

```
g.V('v-1')
  .fold()
  .coalesce(unfold(),
            addV('Person').property(id, 'v-1')
                          .property('email', 'person-1@example.org')
                          .property('version', 3))
```

```
.V('v-2')
.fold()
.coalesce(unfold(),
          addV('Person').property(id, 'v-2')
                        .property('email', 'person-2@example.org')
                        .property('version', 3))
.V('v-3')
.fold()
.coalesce(unfold(),
          addV('Person').property(id, 'v-3')
                        .property('email', 'person-3@example.org')
                        .property('version', 3))
.V('v-1', 'v-2', 'v-3')
.has('version', lt(3))
.property(single, 'version', 3)
.id()
```

Gremlin を使用して Neptune クエリ実行を分析する`explain`

Amazon Neptune に explain という名前の Gremlin 機能が追加されました。この機能は、Neptune エンジンが使用する実行アプローチを理解するためのセルフサービスツールです。この機能を呼び出すには、Gremlin クエリを送信する HTTP コールに `explain` パラメータを追加します。

`explain` 機能は、クエリ実行プランの論理構造に関する情報を提供します。この情報を使用して潜在的な評価と実行障害を明らかにし、Gremlin クエリのチューニング で説明されているようにクエリを調整します。また、クエリに関するヒントを使用して、クエリ実行プランを改善できます。

> **Note**
>
> この機能は、リリース 1.0.1.0.200463.0 (2019-10-15) で始めることで使用できます。

トピック

- Neptune での Gremlin クエリの動作を理解する
- Neptune での Gremlin explain APIの使用
- Neptune の Gremlin profile API
- explainおよびprofileを使用した Gremlin クエリのチューニング
- Amazon Neptune でのネイティブ Gremlin ステップサポート

Neptune での Gremlin クエリの動作を理解する

Amazon Neptune で Gremlin `explain` と `profile` レポートを最大限に活用するには、Gremlin クエリに関する背景情報を理解しておくと役立ちます。

トピック

- Neptune の Gremlin ステートメント
- Neptune がステートメントインデックスを使用して Gremlin クエリを処理する方法
- Neptune での Gremlin クエリの処理方法

Neptune の Gremlin ステートメント

Amazon Neptune のプロパティグラフデータは、4 つの位置 (四角形) のステートメントで構成されます。これらの各ステートメントは、プロパティグラフデータの個々のアトミック単位を表します。詳細については、「Neptune のグラフデータモデル。」を参照してください。リソース記述フレームワーク (RDF) データモデルと同様に、これらの 4 つの位置は次のとおりです。

- `subject (S)`
- `predicate (P)`
- `object (O)`
- `graph (G)`

各ステートメントは、1 つ以上のリソースに関するアサーションです。たとえば、ステートメントでは、2 つのリソース間の関係の有無に対してアサーションを行ったり、いくつかのリソースにプロパティ (キーと値のペア) を付加したりすることができます。

述語は、関係の型またはプロパティの型を記述するステートメントの動詞と考えることができます。オブジェクトは、関係のターゲット、またはプロパティの値を表します。グラフの位置はオプションであり、さまざまな方法で使用できます。Neptune プロパティグラフ (PG) データの場合、未使用 (null グラフ) であるか、エッジ識別子を表すために使用されます。共有リソース識別子を持つ一連のステートメントはグラフを作成します。

Neptune プロパティグラフデータモデルには、次の 3 つのクラスのステートメントがあります。

トピック

- Gremlin 頂点ラベルステートメント

- Gremlin エッジステートメント
- Gremlin プロパティステートメント

Gremlin 頂点ラベルステートメント

Neptune の頂点ラベルステートメントは、次の 2 つの目的を果たします。

- 頂点のラベルを追跡します。
- これらのステートメントが少なくとも 1 つ存在することは、グラフに特定の頂点が存在することを意味します。

これらのステートメントのサブジェクトは頂点識別子、オブジェクトはラベルで、どちらもユーザーが指定します。これらのステートメントには、<~label> として表示される特別な固定述語と、<~> として表示されるデフォルトのグラフ識別子 (null グラフ) を使用します。

たとえば、次の addV トラバーサルを考えてみます。

```
g.addV("Person").property(id, "v1")
```

このトラバーサルにより、次のステートメントがグラフに追加されます。

```
StatementEvent[Added(<v1> <~label> <Person> <~>) .]
```

Gremlin エッジステートメント

Gremlin エッジステートメントは、Neptune のグラフの 2 つの頂点間にエッジが存在することを意味します。エッジステートメントのサブジェクト (S) はソースの from 頂点です。述語 (P) はユーザー指定のエッジラベルです。オブジェクト (O) はターゲットの to 頂点です。グラフ (G) は、ユーザーが指定したエッジ識別子です。

たとえば、次の addE トラバーサルを考えてみます。

```
g.addE("knows").from(V("v1")).to(V("v2")).property(id, "e1")
```

トラバーサルにより、次のステートメントがグラフに追加されます。

```
StatementEvent[Added(<v1> <knows> <v2> <e1>) .]
```

Gremlin プロパティステートメント

Neptune の Gremlin プロパティステートメントは、頂点またはエッジの個々のプロパティ値をアサートします。サブジェクトは、ユーザーが指定した頂点またはエッジの識別子です。述語はプロパティ名 (キー) で、オブジェクトは個々のプロパティ値です。グラフ (G) はここでもデフォルトのグラフ識別子 (null グラフ) で、<~> と表示されます。

次の例を考えます。

```
g.V("v1").property("name", "John")
```

このステートメントの結果は次のようになります。

```
StatementEvent[Added(<v1> <name> "John" <~>) .]
```

オブジェクトがプリミティブ値 (`string`、`date`、`byte`、`short`、`int`、`long`、`float`、または `double`) のため、プロパティステートメントはお互いに異なります。そのオブジェクトは、別のアサーションのサブジェクトとして使用できるリソース識別子ではありません。

複数プロパティの場合、セット内の各プロパティ値は独自のステートメントを受け取ります。

```
g.V("v1").property(set, "phone", "956-424-2563").property(set, "phone", "956-354-3692
 (tel:9563543692)")
```

この結果は以下のようになります。

```
StatementEvent[Added(<v1> <phone> "956-424-2563" <~>) .]
StatementEvent[Added(<v1> <phone> "956-354-3692" <~>) .]
```

Neptune がステートメントインデックスを使用して Gremlin クエリを処理する方法

Amazon Neptune では、3 つのステートメントインデックスを使用してステートメントにアクセスします。詳細については、Neptune でステートメントのインデックスを作成する方法 に記載しています。Neptune は一部の位置が既知の Gremlin クエリからステートメントパターンを抽出し、残りはインデックス検索によって検出するように残されます。

Neptune は、プロパティグラフスキーマサイズが大きくないことを前提としています。つまり、個別のエッジラベルとプロパティ名の数がかなり少なく、個別の述語の総数が少ないことを意味します。Neptune は、個別のインデックスで個別の述語を追跡します。OSGP インデックスを使用する

のではなく、この述語のキャッシュを使用して { all P x POGS } のユニオンスキャンを実行します。リバーストラバーサル OSGP インデックスの必要性を回避することで、ストレージ領域とロードスループットの両方を節約できます。

Neptune Gremlin Explain/Profile API を使用すると、グラフの述語数を取得できます。その後、プロパティグラフスキーマが小さいという Neptune の前提をアプリケーションで無効にするかどうかを判断できます。

次の例は、Neptune がインデックスを使用して Gremlin クエリを処理する方法を示しています。

質問: 頂点 **v1** のラベルは何か?

```
Gremlin code:      g.V('v1').label()
Pattern:           (<v1>, <~label>, ?, ?)
Known positions:   SP
Lookup positions:  OG
Index:             SPOG
Key range:         <v1>:<~label>:*
```

質問: 頂点 **v1** の「知っている」アウトエッジは何か?

```
Gremlin code:      g.V('v1').out('knows')
Pattern:           (<v1>, <knows>, ?, ?)
Known positions:   SP
Lookup positions:  OG
Index:             SPOG
Key range:         <v1>:<knows>:*
```

質問: どの頂点に **Person** 頂点ラベルがあるか?

```
Gremlin code:      g.V().hasLabel('Person')
Pattern:           (?, <~label>, <Person>, <~>)
Known positions:   POG
Lookup positions:  S
Index:             POGS
Key range:         <~label>:<Person>:<~>:*
```

質問 : 指定された エッジ **e1** の始点と終点は何か?

```
Gremlin code:      g.E('e1').bothV()
```

```
Pattern:           (?, ?, ?, <e1>)
Known positions:   G
Lookup positions:  SPO
Index:             GPSO
Key range:         <e1>:*
```

Neptune が持たないステートメントインデックスの 1 つは、リバーストラバーサル OSGP インデックスです。このインデックスは、次の例のように、すべてのエッジラベルにまたがるすべての受信エッジを収集するために使用できます。

質問:入ってくる隣接頂点は何か**v1**?

```
Gremlin code:      g.V('v1').in()
Pattern:           (?, ?, <v1>, ?)
Known positions:   O
Lookup positions:  SPG
Index:             OSGP  // <-- Index does not exist
```

Neptune での Gremlin クエリの処理方法

Amazon Neptune では、より複雑なトラバーサルは、結合を作成するためにパターン間で共有できる名前付き変数の定義に基づいて関係を作成する一連のパターンで表すことができます。以下の例ではこれを示しています。

質問: 頂点 **v1** の 2 ホップ近傍とは何か?

```
Gremlin code:      g.V('v1').out('knows').out('knows').path()
Pattern:           (?1=<v1>, <knows>, ?2, ?) X Pattern(?2, <knows>, ?3, ?)

The pattern produces a three-column relation (?1, ?2, ?3) like this:
                   ?1     ?2     ?3
                   ================
                   v1     v2     v3
                   v1     v2     v4
                   v1     v5     v6
```

2 つのパターン (最初のパターンの O 位置と 2 番目のパターンの S 位置) 間で ?2 変数を共有することで、最初のホップ近傍から 2 番目のホップ近傍への結合を作成します。各 Neptune ソリューションには、トラTinkerPopバーサーを再作成するために使用できる 3 つの名前付き変数 (パス情報を含む) のバインディングがあります。

Gremlin クエリ処理の最初のステップは、一連の TinkerPop ステップ で構成される TinkerPop トラバーサルオブジェクトにクエリを解析することです。オープンソースの Apache TinkerPop プロジェクト の一部であるこれらのステップは、参照実装で Gremlin トラバーサルを構成する論理演算子と物理演算子の両方です。どちらも、クエリのモデルを表すために使用されます。これらは、それらが表す演算子のセマンティクスに従ってソリューションを生成できる実行可能な演算子です。例えば、`.V()`は によって表され、実行されます TinkerPop GraphStep。

これらの off-the-shelf TinkerPop ステップは実行可能であるため、このような TinkerPop トラバーサルは任意の Gremlin クエリを実行して正しい回答を生成できます。ただし、大きなグラフに対して実行すると、 TinkerPop ステップが非常に非効率で遅くなることがあります。Neptune はそれらを使用する代わりに、前に説明したように、トラバーサルをパターンのグループで構成される宣言形式に変換しようとします。

Neptune は現在、ネイティブクエリエンジンですべての Gremlin 演算子 (ステップ) をサポートしていません。そのため、可能な限り多くのステップを、変換されたすべてのステップの宣言型論理クエリプランを含む単一の `NeptuneGraphQueryStep` にまとめようとします。理想的には、すべてのステップが変換されます。ただし、変換できないステップが発生すると、Neptune はネイティブ実行から抜け出し、その時点からのすべてのクエリ実行を TinkerPop ステップに延期します。ネイティブ実行の入出力は行われません。

ステップが論理クエリプランに変換されると、Neptune は一連のクエリオプティマイザを実行し、静的分析と推定基数に基づいてクエリプランを書き換えます。これらのオプティマイザは、範囲カウントに基づく演算子の順序変更、不要または冗長な演算子の除外、フィルタの再配置、異なるグループへの演算子のプッシュなどを行います。

最適化されたクエリプランが作成されると、Neptune はクエリの実行作業を行う物理演算子のパイプラインを作成します。これには、ステートメントインデックスからのデータの読み取り、さまざまなタイプの結合の実行、フィルタリング、順序付けなどが含まれます。パイプラインはソリューションストリームを生成し、トラバーサーオブジェクトの TinkerPopストリームに変換されます。

クエリ結果のシリアル化

Amazon Neptune は現在、 TinkerPop レスポンスメッセージシリアライザーを使用して、クエリ結果 (TinkerPop トラバーサー) をシリアル化されたデータに変換し、ワイヤ経由でクライアントに送り返します。これらのシリアル化形式は、かなり冗長になる傾向があります。

たとえば、`g.V().limit(1)` などの頂点クエリの結果をシリアル化するには、 Neptune クエリエンジンが 1 つの検索を実行してクエリ結果を生成する必要があります。ただし、`GraphSON` シリア

ライザーは、頂点をシリアル化形式にパッケージ化するために、多数の追加の検索を実行します。ラベルを取得するには 1 つの検索、プロパティキーを取得するには 1 つの検索、各キーのすべての値を取得するには、頂点のプロパティキーごとに 1 つの検索を実行する必要があります。

一部のシリアル化形式はより効率的ですが、すべて追加検索が必要です。さらに、 TinkerPop シリアライザーは重複した検索を回避しようとせず、多くの場合、多くの検索が不必要に繰り返されます。

このため、必要な情報のみを具体的に尋ねるように、クエリを記述することが非常に重要になります。たとえば、`g.V().limit(1).id()` は頂点 ID のみを返し、追加のシリアライザー検索をすべて排除します。Neptune の Gremlin `profile` API で、クエリ実行中およびシリアル化中に行われた検索呼び出しの数を確認できます。

Neptune での Gremlin **`explain`** APIの使用

Amazon Neptune Gremlin `explain` API は、指定されたクエリが実行された場合に実行されるクエリプランを返します。API はクエリを実際に実行しないため、プランはほぼ瞬時に返されます。

Neptune エンジンに固有の情報をレポートできるように、 TinkerPop .explain() ステップとは異なります。

Gremlin **`explain`** レポートに含まれる情報

`explain` レポートには、以下の情報が含まれています。

- リクエストされたクエリ文字列。
- 元のトラバーサル。これは、クエリ文字列を TinkerPop ステップに解析することによって生成される TinkerPop トラバーサルオブジェクトです。これは、 に対して TinkerPopクエリ`.explain()`で を実行することで生成される元のクエリと同等です TinkerGraph。
- 変換されたトラバーサル。これは、トラバーサルを Neptune 論理クエリプラン表現に変換することによって生成される Neptune TinkerPop トラバーサルです。多くの場合、 TinkerPop トラバーサル全体が 2 つの Neptune ステップに変換されます。1 つはクエリ全体を実行するステップ (`NeptuneGraphQueryStep`)、もう 1 つは Neptune クエリエンジン出力を TinkerPop トラバーサーに変換するステップ () ですNeptuneTraverserConverterStep。
- 最適化されたトラバーサル。これは、静的分析と推定基数に基づいてクエリを書き換える一連の静的作業削減オプティマイザを実行した後の Neptune クエリプランの最適化バージョンです。これらのオプティマイザは、範囲カウントに基づく演算子の順序変更、不要または冗長な演算子の除外、フィルタの再配置、異なるグループへの演算子のプッシュなどを行います。

- 述語のカウント。前述の Neptune インデックス作成戦略のため、多数の異なる述語があると、パフォーマンスの問題が発生する可能性があります。これは、エッジラベル (.in または .both) のないリバーストラバーサル演算子を使用するクエリに特に当てはまります。このような演算子が使用され、述語数が十分に高い場合、explain レポートには警告メッセージが表示されます。
- DFE 情報 DFE 代替エンジンが有効な場合、最適化トラバーサルに次のトラバーサルコンポーネントが表示されることがあります。
  - **DFEStep** – 子 DFENode を含むトラバーサルの Neptune 最適化 DFE ステップ。DFEStep は、DFE エンジンで実行されるクエリプランの一部を表します。
  - **DFENode** – 中間表現を 1 つ以上の子 DFEJoinGroupNodes として含みます。
  - **DFEJoinGroupNode** – 1 つ以上の DFENode または DFEJoinGroupNode 要素の結合を表します。
  - **NeptuneInterleavingStep** – 子 DFEStep を含むトラバーサルの Neptune 最適化 DFE ステップ。

    また、フロンティア要素、使用されるパス要素など、トラバーサルに関する情報を含む stepInfo 要素を含みます。この情報は、子 DFEStep の処理に使用されます。

  クエリが DFE によって評価されているかどうかを簡単に調べるには、explain 出力に DFEStep が含まれるかを確認します。の一部ではないトラバーサルの部分はDFEStep、DFE によって実行されず、 TinkerPop エンジンによって実行されます。

  サンプルレポートに関しては DFE が有効な場合の例 を参照してください。

Gremlin **explain** 構文

explain API の構文は、次の例のように、/gremlin/explain を /gremlin の代わりにエンドポイントとして使用すること以外はクエリの HTTP API の構文と同じです。

```
curl -X POST https://your-neptune-endpoint:port/gremlin/explain -d
 '{"gremlin":"g.V().limit(1)"}'
```

前述のクエリでは、次の出力が生成されます。

```
=======================================================
                Neptune Gremlin Explain
=======================================================

Query String
```

```
============
g.V().limit(1)

Original Traversal
==================
[GraphStep(vertex,[]), RangeGlobalStep(0,1)]

Converted Traversal
===================
Neptune steps:
[
    NeptuneGraphQueryStep(Vertex) {
        JoinGroupNode {
            PatternNode[(?1, <~label>, ?2, <~>) . project distinct ?1 .]
        }, finishers=[limit(1)], annotations={path=[Vertex(?1):GraphStep], maxVarId=3}
    },
    NeptuneTraverserConverterStep
]

Optimized Traversal
===================
Neptune steps:
[
    NeptuneGraphQueryStep(Vertex) {
        JoinGroupNode {
            PatternNode[(?1, <~label>, ?2, <~>) . project distinct ?1 .],
 {estimatedCardinality=INFINITY}
        }, finishers=[limit(1)], annotations={path=[Vertex(?1):GraphStep], maxVarId=3}
    },
    NeptuneTraverserConverterStep
]

Predicates
==========
# of predicates: 18
```

未変換の TinkerPop ステップ

トラバーサルのすべての TinkerPop ステップにネイティブ Neptune オペレータカバレッジがあるのが理想的です。そうでない場合、Neptune は TinkerPop ステップ実行にフォールバックして、オペレータカバレッジのギャップを確認します。Neptune がネイティブカバレッジを持っていないステップをトラバーサルで使用している場合、`explain` レポートにはギャップが発生した場所を示す警告が表示されます。

対応するネイティブ Neptune 演算子を持たないステップが検出されると、それ以降の TinkerPop ステップにネイティブ Neptune 演算子がある場合でも、その時点からのトラバーサル全体がステップを使用して実行されます。

ただし、Neptune フルテキスト検索が呼び出される場合は例外です。は、全文検索ステップとしてネイティブの同等のものを持たないステップ NeptuneSearchStep を実装します。

クエリのすべてのステップに同等のネイティブがある **explain** 出力例

以下に、すべてのステップに同等のネイティブがあるクエリの explain レポートの例を示します。

```
*******************************************************
                Neptune Gremlin Explain
*******************************************************

Query String
============
g.V().out()

Original Traversal
==================
[GraphStep(vertex,[]), VertexStep(OUT,vertex)]

Converted Traversal
===================
Neptune steps:
[
    NeptuneGraphQueryStep(Vertex) {
        JoinGroupNode {
            PatternNode[(?1, <~label>, ?2, <~>) . project distinct ?1 .]
            PatternNode[(?1, ?5, ?3, ?6) . project ?1,?3 . IsEdgeIdFilter(?6) .]
            PatternNode[(?3, <~label>, ?4, <~>) . project ask .]
        }, annotations={path=[Vertex(?1):GraphStep, Vertex(?3):VertexStep], maxVarId=7}
    },
    NeptuneTraverserConverterStep
]

Optimized Traversal
===================
Neptune steps:
[
    NeptuneGraphQueryStep(Vertex) {
        JoinGroupNode {
```

```
            PatternNode[(?1, ?5, ?3, ?6) . project ?1,?3 . IsEdgeIdFilter(?6) .],
 {estimatedCardinality=INFINITY}
        }, annotations={path=[Vertex(?1):GraphStep, Vertex(?3):VertexStep], maxVarId=7}
    },
    NeptuneTraverserConverterStep
]

Predicates
==========
# of predicates: 18
```

クエリ内の一部のステップに同等のネイティブがない例

Neptune は GraphStep と VertexStep の両方をネイティブに処理しますが、FoldStep と UnfoldStep を導入すると、結果 explain 出力は異なります。

```
*******************************************************
                Neptune Gremlin Explain
*******************************************************

Query String
============
g.V().fold().unfold().out()

Original Traversal
==================
[GraphStep(vertex,[]), FoldStep, UnfoldStep, VertexStep(OUT,vertex)]

Converted Traversal
===================
Neptune steps:
[
    NeptuneGraphQueryStep(Vertex) {
        JoinGroupNode {
            PatternNode[(?1, <~label>, ?2, <~>) . project distinct ?1 .]
        }, annotations={path=[Vertex(?1):GraphStep], maxVarId=3}
    },
    NeptuneTraverserConverterStep
]
+ not converted into Neptune steps: [FoldStep, UnfoldStep, VertexStep(OUT,vertex)]

Optimized Traversal
===================
```

```
Neptune steps:
[
    NeptuneGraphQueryStep(Vertex) {
        JoinGroupNode {
            PatternNode[(?1, <~label>, ?2, <~>) . project distinct ?1 .],
 {estimatedCardinality=INFINITY}
        }, annotations={path=[Vertex(?1):GraphStep], maxVarId=3}
    },
    NeptuneTraverserConverterStep,
    NeptuneMemoryTrackerStep
]
+ not converted into Neptune steps: [FoldStep, UnfoldStep, VertexStep(OUT,vertex)]

WARNING: >> FoldStep << is not supported natively yet
```

この場合、FoldStep はネイティブ実行を中断します。ただし、Fold/Unfold ステップの下流に表示されるため、後続の VertexStep もネイティブに処理されなくなります。

パフォーマンスとコスト削減のためには、 TinkerPop トラバーサルを定式化して、ステップ実装ではなく Neptune クエリエンジン内で可能な限り多くの作業をネイティブに実行することが重要です。

Neptune を使用するクエリの例 full-text-search

次のクエリでは、Neptune フルテキスト検索を使用します。

```
g.withSideEffect("Neptune#fts.endpoint", "some_endpoint")
  .V()
  .tail(100)
  .has("Neptune#fts mark*")
  -------
  .has("name", "Neptune#fts mark*")
  .has("Person", "name", "Neptune#fts mark*")
```

.has("name", "Neptune#fts mark*") パートは、検索の対象を name を持つ頂点に制限します。.has("Person", "name", "Neptune#fts mark*") は、検索の対象を name およびラベル Person を持つ頂点に制限します。これにより、explain レポートのトラバーサルは次のようになります。

```
Final Traversal
[NeptuneGraphQueryStep(Vertex) {
    JoinGroupNode {
```

```
        PatternNode[(?1, termid(1,URI), ?2, termid(0,URI)) . project distinct ?1 .],
 {estimatedCardinality=INFINITY}
    }, annotations={path=[Vertex(?1):GraphStep], maxVarId=4}
}, NeptuneTraverserConverterStep, NeptuneTailGlobalStep(10),
 NeptuneTinkerpopTraverserConverterStep, NeptuneSearchStep {
    JoinGroupNode {
        SearchNode[(idVar=?3, query=mark*, field=name) . project ask .],
 {endpoint=some_endpoint}
    }
    JoinGroupNode {
        SearchNode[(idVar=?3, query=mark*, field=name) . project ask .],
 {endpoint=some_endpoint}
    }
}]
```

DFE が有効な場合の **explain** 使用例

DFE 代替クエリエンジンが有効な場合の explain レポート例を次に示します。

```
********************************************************
                Neptune Gremlin Explain
********************************************************

Query String
============

g.V().as("a").out().has("name", "josh").out().in().where(eq("a"))


Original Traversal
==================
[GraphStep(vertex,[])@[a], VertexStep(OUT,vertex), HasStep([name.eq(josh)]),
 VertexStep(OUT,vertex), VertexStep(IN,vertex), WherePredicateStep(eq(a))]

Converted Traversal
===================
Neptune steps:
[
    DFEStep(Vertex) {
      DFENode {
        DFEJoinGroupNode[ children={
          DFEPatternNode[(?1, <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>, ?2,
 <http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph>) . project DISTINCT[?1]
 {rangeCountEstimate=unknown}],
```

```
          DFEPatternNode[(?1, ?3, ?4, ?5) . project ALL[?1, ?4] graphFilters=(!
= <http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph> . ),
 {rangeCountEstimate=unknown}]
        }, {rangeCountEstimate=unknown}
        ]
      } [Vertex(?1):GraphStep@[a], Vertex(?4):VertexStep]
    } ,
    NeptuneTraverserConverterDFEStep
]
+ not converted into Neptune steps: HasStep([name.eq(josh)]),
Neptune steps:
[
    NeptuneInterleavingStep {
      StepInfo[joinVars=[?7, ?1], frontierElement=Vertex(?7):HasStep,
 pathElements={a=(last,Vertex(?1):GraphStep@[a])}, listPathElement={}, indexTime=0ms],
      DFEStep(Vertex) {
        DFENode {
          DFEJoinGroupNode[ children={
            DFEPatternNode[(?7, ?8, ?9, ?10) . project ALL[?7, ?9]
 graphFilters=(!= <http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph> . ),
 {rangeCountEstimate=unknown}],
            DFEPatternNode[(?12, ?11, ?9, ?13) . project ALL[?9, ?12]
 graphFilters=(!= <http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph> . ),
 {rangeCountEstimate=unknown}]
          }, {rangeCountEstimate=unknown}
          ]
        } [Vertex(?9):VertexStep, Vertex(?12):VertexStep]
      }
    }
]
+ not converted into Neptune steps: WherePredicateStep(eq(a)),
Neptune steps:
[
    DFECleanupStep
]


Optimized Traversal
===================
Neptune steps:
[
    DFEStep(Vertex) {
      DFENode {
        DFEJoinGroupNode[ children={
```

```
          DFEPatternNode[(?1, ?3, ?4, ?5) . project ALL[?1, ?4] graphFilters=(!=
 defaultGraph[526] . ), {rangeCountEstimate=9223372036854775807}]
        }, {rangeCountEstimate=unknown}
        ]
      } [Vertex(?1):GraphStep@[a], Vertex(?4):VertexStep]
    } ,
    NeptuneTraverserConverterDFEStep
]
+ not converted into Neptune steps: NeptuneHasStep([name.eq(josh)]),
Neptune steps:
[
    NeptuneMemoryTrackerStep,
    NeptuneInterleavingStep {
      StepInfo[joinVars=[?7, ?1], frontierElement=Vertex(?7):HasStep,
 pathElements={a=(last,Vertex(?1):GraphStep@[a])}, listPathElement={}, indexTime=0ms],
      DFEStep(Vertex) {
        DFENode {
          DFEJoinGroupNode[ children={
            DFEPatternNode[(?7, ?8, ?9, ?10) . project ALL[?7, ?9] graphFilters=(!=
 defaultGraph[526] . ), {rangeCountEstimate=9223372036854775807}],
            DFEPatternNode[(?12, ?11, ?9, ?13) . project ALL[?9, ?12] graphFilters=(!=
 defaultGraph[526] . ), {rangeCountEstimate=9223372036854775807}]
          }, {rangeCountEstimate=unknown}
          ]
        } [Vertex(?9):VertexStep, Vertex(?12):VertexStep]
      }
    }
]
+ not converted into Neptune steps: WherePredicateStep(eq(a)),
Neptune steps:
[
    DFECleanupStep
]


WARNING: >> [NeptuneHasStep([name.eq(josh)]), WherePredicateStep(eq(a))] << (or one of
 the children for each step) is not supported natively yet

Predicates
==========
# of predicates: 8
```

レポート内の DFE 固有のセクションの詳細については、explain 内の情報 を参照してください。

Neptune の Gremlin **`profile`** API

Neptune Gremlin `profile` API は、指定された Gremlin トラバーサルを実行し、実行に関するさまざまなメトリクスを収集して、出力として Profile レポートを生成します。

> **Note**
>
> この機能は、リリース 1.0.1.0.200463.0 (2019-10-15) で始めることで使用できます。

Neptune エンジンに固有の情報をレポートできるように、 TinkerPop .profile() ステップとは異なります。

Profile レポートには、クエリプランに関する次の情報が含まれます。

- 物理演算子のパイプライン
- クエリの実行とシリアル化のインデックスオペレーション
- 結果のサイズ

`profile` API は、エンドポイントとして `/gremlin` の代わりに `/gremlin/profile` を使用して、クエリに HTTP API 構文の拡張バージョンを使用します。

Neptune Gremlin に固有のパラメータ**`profile`**

- profile.results–`boolean`、使用できる値:`TRUE` および `FALSE`、デフォルト値:`TRUE`。

  true の場合、クエリ結果が収集され、`profile` レポートの一部として表示されます。false の場合、結果カウントのみが表示されます。
- profile.chop–`int`、デフォルト値:250。

  ゼロ以外の場合、結果の文字列はその文字数で切り捨てられます。これにより、すべての結果がキャプチャされなくなるわけではありません。Profile レポートの文字列のサイズを制限するだけです。ゼロに設定すると、文字列にはすべての結果が含まれます。
- profile.serializer – `string`、デフォルト値: `<null>`。

  null 以外の場合、収集された結果は、このパラメータで指定された形式でシリアル化されたレスポンスメッセージで返されます。そのレスポンスメッセージを生成するために必要なインデックスオペレーションの数は、クライアントに送信されるバイト単位のサイズとともにレポートされます。

使用できる値は、 <null> または有効な MIME タイプまたは TinkerPop ドライバーの「シリアライザー」列挙値のいずれかです。

```
"application/json" or "GRAPHSON"
"application/vnd.gremlin-v1.0+json" or "GRAPHSON_V1"
"application/vnd.gremlin-v1.0+json;types=false" or "GRAPHSON_V1_UNTYPED"
"application/vnd.gremlin-v2.0+json" or "GRAPHSON_V2"
"application/vnd.gremlin-v2.0+json;types=false" or "GRAPHSON_V2_UNTYPED"
"application/vnd.gremlin-v3.0+json" or "GRAPHSON_V3"
"application/vnd.gremlin-v3.0+json;types=false" or "GRAPHSON_V3_UNTYPED"
"application/vnd.graphbinary-v1.0" or "GRAPHBINARY_V1"
```

- profile.indexOps–boolean、使用できる値:TRUE および FALSE、デフォルト値:FALSE。

  true の場合、クエリの実行およびシリアル化中に行われたすべてのインデックスオペレーションの詳細レポートが表示されます。警告: このレポートは冗長な場合があります。

Neptune Gremlin のサンプル出力**profile**

サンプル profile クエリを以下に示します。

```
curl -X POST https://your-neptune-endpoint:port/gremlin/profile \
     -d '{"gremlin":"g.V().hasLabel(\"airport\")
                          .has(\"code\", \"AUS\")
                          .emit()
                          .repeat(in().simplePath())
                          .times(2)
                          .limit(100)",
          "profile.serializer":"application/vnd.gremlin-v3.0+gryo"}'
```

このクエリは、ブログ記事 Let Me Graph That For You – Part 1 – Air Routes からエアルートサンプルグラフで実行された場合、次の profile レポートを生成します。

```
*******************************************************
                Neptune Gremlin Profile
*******************************************************

Query String
==================
```

```
g.V().hasLabel("airport").has("code",
 "AUS").emit().repeat(in().simplePath()).times(2).limit(100)

Original Traversal
==================
[GraphStep(vertex,[]), HasStep([~label.eq(airport), code.eq(AUS)]),
 RepeatStep(emit(true),[VertexStep(IN,vertex), PathFilterStep(simple),
 RepeatEndStep],until(loops(2))), RangeGlobalStep(0,100)]

Optimized Traversal
===================
Neptune steps:
[
    NeptuneGraphQueryStep(Vertex) {
        JoinGroupNode {
            PatternNode[(?1, <code>, "AUS", ?) . project ?1 .],
 {estimatedCardinality=1, indexTime=84, hashJoin=true, joinTime=3, actualTotalOutput=1}
            PatternNode[(?1, <~label>, ?2=<airport>, <~>) . project ask .],
 {estimatedCardinality=3374, indexTime=29, hashJoin=true, joinTime=0,
 actualTotalOutput=61}
            RepeatNode {
                Repeat {
                    PatternNode[(?3, ?5, ?1, ?6) . project ?1,?3 . IsEdgeIdFilter(?6) .
 SimplePathFilter(?1, ?3)) .], {hashJoin=true, estimatedCardinality=50148, indexTime=0,
 joinTime=3}
                }
                Emit {
                    Filter(true)
                }
                LoopsCondition {
                    LoopsFilter([?1, ?3],eq(2))
                }
            }, annotations={repeatMode=BFS, emitFirst=true, untilFirst=false, leftVar=?
1, rightVar=?3}
        }, finishers=[limit(100)], annotations={path=[Vertex(?1):GraphStep,
 Repeat[Vertex(?3):VertexStep]], joinStats=true, optimizationTime=495, maxVarId=7,
 executionTime=323}
    },
    NeptuneTraverserConverterStep
]

Physical Pipeline
=================
NeptuneGraphQueryStep
```

```
    |-- StartOp
    |-- JoinGroupOp
        |-- SpoolerOp(100)
        |-- DynamicJoinOp(PatternNode[(?1, <code>, "AUS", ?) . project ?1 .],
 {estimatedCardinality=1, indexTime=84, hashJoin=true})
        |-- SpoolerOp(100)
        |-- DynamicJoinOp(PatternNode[(?1, <~label>, ?2=<airport>, <~>) . project
 ask .], {estimatedCardinality=3374, indexTime=29, hashJoin=true})
        |-- RepeatOp
            |-- <upstream input> (Iteration 0) [visited=1, output=1 (until=0, emit=1),
 next=1]
            |-- BindingSetQueue (Iteration 1) [visited=61, output=61 (until=0,
 emit=61), next=61]
                |-- SpoolerOp(100)
                |-- DynamicJoinOp(PatternNode[(?3, ?5, ?1, ?6) . project ?
1,?3 . IsEdgeIdFilter(?6) . SimplePathFilter(?1, ?3)) .], {hashJoin=true,
 estimatedCardinality=50148, indexTime=0})
            |-- BindingSetQueue (Iteration 2) [visited=38, output=38 (until=38,
 emit=0), next=0]
                |-- SpoolerOp(100)
                |-- DynamicJoinOp(PatternNode[(?3, ?5, ?1, ?6) . project ?
1,?3 . IsEdgeIdFilter(?6) . SimplePathFilter(?1, ?3)) .], {hashJoin=true,
 estimatedCardinality=50148, indexTime=0})
        |-- LimitOp(100)

Runtime (ms)
============
Query Execution:  392.686
Serialization:   2636.380

Traversal Metrics
=================
Step                                                               Count  Traversers
    Time (ms)    % Dur
-------------------------------------------------------------------------------------------------
NeptuneGraphQueryStep(Vertex)                                        100         100
      314.162    82.78
NeptuneTraverserConverterStep                                        100         100
       65.333    17.22
                                            >TOTAL                     -           -
      379.495        -

Repeat Metrics
==============
```

```
Iteration  Visited   Output    Until     Emit     Next
-----------------------------------------------------
        0        1        1        0        1        1
        1       61       61        0       61       61
        2       38       38       38        0        0
-----------------------------------------------------
               100      100       38       62       62

Predicates
==========
# of predicates: 16

WARNING: reverse traversal with no edge label(s) - .in() / .both() may impact query
 performance

Results
=======
Count: 100
Output: [v[3], v[3600], v[3614], v[4], v[5], v[6], v[7], v[8], v[9], v[10], v[11],
 v[12], v[47], v[49], v[136], v[13], v[15], v[16], v[17], v[18], v[389], v[20], v[21],
 v[22], v[23], v[24], v[25], v[26], v[27], v[28], v[416], v[29], v[30], v[430], v[31],
 v[9...
Response serializer: GRYO_V3D0
Response size (bytes): 23566

Index Operations
================
Query execution:
    # of statement index ops: 3
    # of unique statement index ops: 3
    Duplication ratio: 1.0
    # of terms materialized: 0
Serialization:
    # of statement index ops: 200
    # of unique statement index ops: 140
    Duplication ratio: 1.43
    # of terms materialized: 393
```

Neptune explain の呼び出しによって返されるクエリプランに加えて、profile の結果にはクエリ実行に関するランタイム統計が含まれます。各 Join オペレーションには、その結合の実行にかかった時間と、それを通過した実際のソリューションの数でタグが付けられます。

profile 出力には、コアクエリ実行フェーズでの所要時間と、profile.serializer オプションが指定されている場合にはシリアル化フェーズも含まれます。

各フェーズで実行されたインデックスオペレーションの内訳は、profile 出力の下部にも含まれます。

同じクエリを連続して実行すると、キャッシュのため、実行時とインデックスオペレーションに関して異なる結果が表示される場合があります。

repeat() ステップを使用したクエリの場合、repeat() ステップが NeptuneGraphQueryStep の一部としてプッシュダウンされた場合、各反復処理のフロンティアの内訳が利用可能です。

DFE が有効な時の **profile** レポートの違い

Neptune DFE 代替クエリエンジンが有効になっている場合、profile出力は多少異なります。

最適トラバーサル: このセクションは、explain 出力のセクションに似ていますが、追加情報が含まれています。これには、計画で考慮された DFE 演算子のタイプと、関連する最悪および最善のコスト見積もりが含まれます。

物理パイプライン: このセクションでは、クエリの実行に使用される演算子について説明します。DFESubQuery 要素は、DFE が担当するプランの一部を実行するために使用する物理プランを抽象化します。DFESubQuery 要素は次のセクションで展開され、DFE 統計情報が表示されます。

DFE QueryEngine 統計： このセクションは、クエリの少なくとも一部が DFE によって実行された場合にのみ表示されます。DFE に固有の各種ランタイム統計を概説し、DFESubQuery による、クエリ実行のさまざまな部分に費やされた時間の詳細な内訳を以下に示します。

別の DFESubQuery 要素にネストされたサブクエリは、このセクションでフラット化され、一意の識別子に subQuery= で始まるヘッダーが付けられます。

トラバーサルメトリクス: このセクションには、ステップレベルのトラバーサルメトリックが表示され、DFE エンジンがクエリの全部または一部を実行すると、DFEStepおよび/またはNeptuneInterleavingStepのメトリクスが表示されます。explainおよびprofileを使用した Gremlin クエリのチューニング を参照してください。

> **Note**
>
> DFE はラボモードでリリースされる実験的な機能なので、profile 出力はまだ変更される可能性があります。

Neptune データフローエンジン (DFE) が有効な場合のサンプル**profile** 出力

DFE エンジンが Gremlin クエリの実行に使用されている場合、Gremlin `profile` API の出力は、次の例に示すようにフォーマットされます。

クエリ:

```
curl https://localhost:8182/gremlin/profile \
  -d "{\"gremlin\": \"g.withSideEffect('Neptune#useDFE', true).V().has('code',
 'ATL').out()\"}"
```

```
********************************************************
                    Neptune Gremlin Profile
    ********************************************************

    Query String
    ==================
    g.withSideEffect('Neptune#useDFE', true).V().has('code', 'ATL').out()

    Original Traversal
    ==================
    [GraphStep(vertex,[]), HasStep([code.eq(ATL)]), VertexStep(OUT,vertex)]

    Optimized Traversal
    ===================
    Neptune steps:
    [
        DFEStep(Vertex) {
          DFENode {
            DFEJoinGroupNode[null](
              children=[
                DFEPatternNode((?1, vp://code[419430926], ?4, defaultGraph[526]) .
 project DISTINCT[?1] objectFilters=(in(ATL[452987149]) . ), {rangeCountEstimate=1},
                    opInfo=(type=PipelineJoin,
 cost=(exp=(in=1.00,out=1.00,io=0.00,comp=0.00,mem=0.00),wc=(in=1.00,out=1.00,io=0.00,comp=0.00
                    disc=(type=PipelineScan,
 cost=(exp=(in=1.00,out=1.00,io=0.00,comp=0.00,mem=34.00),wc=(in=1.00,out=1.00,io=0.00,comp=0.0
                DFEPatternNode((?1, ?5, ?6, ?7) . project ALL[?1, ?6] graphFilters=(!=
 defaultGraph[526] . ), {rangeCountEstimate=9223372036854775807})],
              opInfo=[
                OperatorInfoWithAlternative[
                  rec=(type=PipelineJoin,
 cost=(exp=(in=1.00,out=27.76,io=0.00,comp=0.00,mem=0.00),wc=(in=1.00,out=27.76,io=0.00,comp=0.
```

```
                    disc=(type=PipelineScan,
 cost=(exp=(in=1.00,out=27.76,io=Infinity,comp=0.00,mem=295147905179352830000.00),wc=(in=1.00,o
                  alt=(type=PipelineScan,
 cost=(exp=(in=1.00,out=27.76,io=Infinity,comp=0.00,mem=295147905179352830000.00),wc=(in=1.00,o
          } [Vertex(?1):GraphStep, Vertex(?6):VertexStep]
        } ,
        NeptuneTraverserConverterDFEStep,
        DFECleanupStep
    ]


    Physical Pipeline
    =================
    DFEStep
        |-- DFESubQuery1

    DFEQueryEngine Statistics
    =================
    DFESubQuery1

 ########################################################################################################
    # ID # Out #1 # Out #2 # Name                   # Arguments
                                                                                    # Mode #
 Units In # Units Out # Ratio  # Time (ms) #

 ########################################################################################################
    # 0  # 1      # -      # DFESolutionInjection  # solutions=[]
                                                                                  # -    # 0
      # 1         # 0.00   # 0.01      #
    #    #        #        #                       # outSchema=[]
                                                                                  #      #
      #           #        #           #

 ########################################################################################################
    # 1  # 2      # -      # DFEChunkLocalSubQuery # subQuery=http://aws.amazon.com/
neptune/vocab/v01/dfe/past/graph#089f43e3-4d71-4259-8d19-254ff63cee04/graph_1 # -    #
 1        # 1         # 1.00   # 0.02      #

 ########################################################################################################
    # 2  # 3      # -      # DFEChunkLocalSubQuery # subQuery=http://aws.amazon.com/
neptune/vocab/v01/dfe/past/graph#089f43e3-4d71-4259-8d19-254ff63cee04/graph_2 # -    #
 1        # 242       # 242.00 # 0.02      #

 ########################################################################################################
```

```
   # 3  # 4      # -      # DFEMergeChunks        # -
                                                                  # -    # 242
     # 242       # 1.00   # 0.01      #

 ########################################################################################
   # 4  # -      # -      # DFEDrain               # -
                                                                  # -    # 242
     # 0         # 0.00   # 0.01      #

 ########################################################################################


   subQuery=http://aws.amazon.com/neptune/vocab/v01/dfe/past/
graph#089f43e3-4d71-4259-8d19-254ff63cee04/graph_1

 ########################################################################################
   # ID # Out #1 # Out #2 # Name                 # Arguments
                        # Mode # Units In # Units Out # Ratio # Time (ms) #

 ########################################################################################
   # 0  # 1      # -      # DFEPipelineScan      # pattern=Node(?1) with property
 'code' as ?4 and label 'ALL' # -    # 0        # 1         # 0.00  # 0.22      #
   #    #        #        #                     # inlineFilters=[(?4 IN ["ATL"])]
                        #      #          #           #       #           #
   #    #        #        #                     # patternEstimate=1
                        #      #          #           #       #           #

 ########################################################################################
   # 1  # 2      # -      # DFEMergeChunks       # -
                        # -    # 1        # 1         # 1.00  # 0.02      #

 ########################################################################################
   # 2  # 4      # -      # DFERelationalJoin    # joinVars=[]
                        # -    # 2        # 1         # 0.50  # 0.09      #

 ########################################################################################
   # 3  # 2      # -      # DFESolutionInjection # solutions=[]
                        # -    # 0        # 1         # 0.00  # 0.01      #
   #    #        #        #                     # outSchema=[]
                        #      #          #           #       #           #

 ########################################################################################
   # 4  # -      # -      # DFEDrain             # -
                        # -    # 1        # 0         # 0.00  # 0.01      #
```

```
 ########################################################################################################


    subQuery=http://aws.amazon.com/neptune/vocab/v01/dfe/past/
graph#089f43e3-4d71-4259-8d19-254ff63cee04/graph_2

 ########################################################################################################
    # ID # Out #1 # Out #2 # Name                 # Arguments
 # Mode # Units In # Units Out # Ratio  # Time (ms) #

 ########################################################################################################
    # 0  # 1      # -      # DFESolutionInjection # solutions=[]
 # -    # 0        # 1         # 0.00   # 0.01      #
    #    #        #         #                     # outSchema=[?1]
 #      #          #           #        #           #

 ########################################################################################################
    # 1  # 2      # 3      # DFETee               # -
 # -    # 1        # 2         # 2.00   # 0.01      #

 ########################################################################################################
    # 2  # 4      # -      # DFEDistinctColumn    # column=?1
 # -    # 1        # 1         # 1.00   # 0.21      #
    #    #        #         #                     # ordered=false
 #      #          #           #        #           #

 ########################################################################################################
    # 3  # 5      # -      # DFEHashIndexBuild    # vars=[?1]
 # -    # 1        # 1         # 1.00   # 0.03      #

 ########################################################################################################
    # 4  # 5      # -      # DFEPipelineJoin      # pattern=Edge((?1)-[?7:?5]->(?6))
 # -    # 1        # 242       # 242.00 # 0.51      #
    #    #        #         #                     # constraints=[]
 #      #          #           #        #           #
    #    #        #         #                     # patternEstimate=9223372036854775807
 #      #          #           #        #           #

 ########################################################################################################
    # 5  # 6      # 7      # DFESync              # -
 # -    # 243      # 243       # 1.00   # 0.02      #

 ########################################################################################################
```

```
   # 6  # 8      # -      # DFEForwardValue      # -
# -    # 1        # 1         # 1.00   # 0.01      #

################################################################################################
   # 7  # 8      # -      # DFEForwardValue      # -
# -    # 242      # 242       # 1.00   # 0.02      #

################################################################################################
   # 8  # 9      # -      # DFEHashIndexJoin     # -
# -    # 243      # 242       # 1.00   # 0.31      #

################################################################################################
   # 9  # -      # -      # DFEDrain             # -
# -    # 242      # 0         # 0.00   # 0.01      #

################################################################################################


   Runtime (ms)
   ============
   Query Execution: 11.744

   Traversal Metrics
   =================
   Step                                                               Count
Traversers       Time (ms)    % Dur

-----------------------------------------------------------------------------------------------
   DFEStep(Vertex)                                                      242
242          10.849    95.48
   NeptuneTraverserConverterDFEStep                                     242
242           0.514     4.52
                                                >TOTAL                    -
-          11.363        -

   Predicates
   ==========
   # of predicates: 18

   Results
   =======
   Count: 242
```

```
Index Operations
================
Query execution:
    # of statement index ops: 0
    # of terms materialized: 0
```

> **Note**
>
> DFE エンジンはラボモードでリリースされる実験的な機能なので、`profile` 出力は変更される可能性があります。

`explain`および`profile`を使用した Gremlin クエリのチューニング

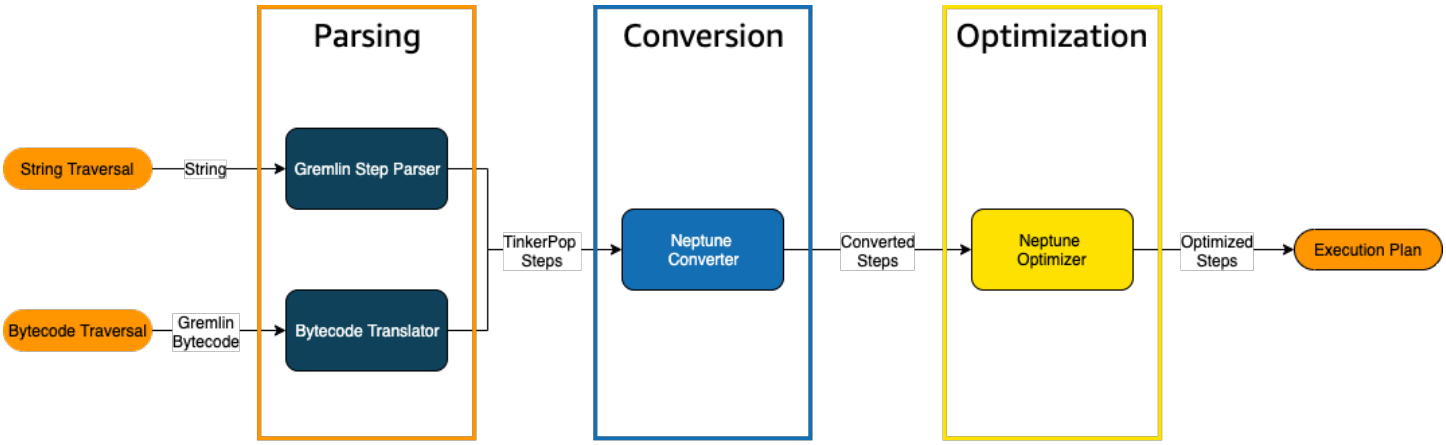
Amazon Neptune で Gremlin クエリを調整して、Neptune explain および profile API から取得したレポートで入手できる情報を使用して、パフォーマンスを向上させることができます。そうすることで、Neptune が Gremlin トラバーサルをどのように処理するのかを理解するのに役立ちます。

> **Important**
>
> TinkerPop バージョン 3.4.11 で変更が行われ、クエリの処理方法の正確性が向上しましたが、現時点ではクエリのパフォーマンスに重大な影響を与える可能性があります。
> たとえば、この種類のクエリの実行速度が大幅に遅くなる可能性があります。
>
> ```
> g.V().hasLabel('airport').
>   order().
>     by(out().count(),desc).
>   limit(10).
>   out()
> ```
>
> 制限ステップ後の頂点は、 TinkerPop 3.4.11 変更の最適ではない方法で取得されるようになりました。これを回避するには、barrier() ステップを `order().by()` の次の任意のポイントに追加して、クエリを変更できます。例:
>
> ```
> g.V().hasLabel('airport').
>   order().
>     by(out().count(),desc).
>   limit(10).
>   barrier().
>   out()
> ```

Neptune エンジンバージョン 1.0.5.0 で TinkerPop 3.4.11 が有効になっていました。

Neptune の Gremlin トラバーサル処理について理解する

Gremlin トラバーサルが Neptune に送信されると、トラバーサルをエンジンが実行するための基礎となる実行計画に変換する 主要なプロセスが 3 つあります。パーシング、変換、最適化です。



トラバーサル解析プロセス

トラバーサルを処理する最初のステップは、それを共通言語に解析することです。Neptune では、共通言語は TinkerPop API の一部である一連の TinkerPop ステップです。これらの各ステップは、トラバーサル内の計算単位を表します。

Gremlin トラバーサルを文字列またはバイトコードとしてNeptune に送信できます。REST エンドポイントと Java クライアントドライバ `submit()` メソッドは次の例のように、トラバーサルを文字列として送信します。

```
client.submit("g.V()")
```

Gremlin 言語変種 (GLV)を使用するアプリケーションおよび言語ドライバはバイトコードでトラバーサルを送信します。

トラバーサル変換プロセス

トラバーサルを処理する 2 番目のステップは、その TinkerPop ステップを変換された Neptune ステップと変換されていない Neptune ステップのセットに変換することです。Apache TinkerPop Gremlin クエリ言語のほとんどのステップは、基盤となる Neptune エンジンで実行するように最

適化された Neptune 固有のステップに変換されます。トラバーサルで Neptune と同等のもののない TinkerPop ステップが発生すると、そのステップとトラバーサルの後続のすべてのステップが TinkerPop クエリエンジンによって処理されます。

どのような状況でどのようなステップが変換できるかの詳細については、Gremlin ステップサポートを参照してください。

トラバーサル最適化プロセス

トラバーサル処理の最後のステップは、オプティマイザを介して変換されたステップと未変換の一連のステップを実行し、最適な実行プランを決定することです。この最適化の出力は、Neptune エンジンが処理する実行計画です。

Neptune Gremlin **explain** API を使用してクエリを調整する

Neptune explain APIは、Gremlin`explain()` ステップとは同じではありません。これは、クエリの実行時に Neptune エンジンが処理する最終実行プランを返します。処理を実行しないため、使用されるパラメータに関係なく同じプランが返され、実際の実行に関する統計が出力に含まれません。

アンカレッジのすべての空港の頂点を見つける次の単純なトラバーサルを考えます。

```
g.V().has('code','ANC')
```

このトラバーサルを Neptune `explain` APIで実行するには、2 つの方法があります。最初の方法は、EXPLAIN エンドポイントに対して、次のように REST 呼び出しを行うことです。

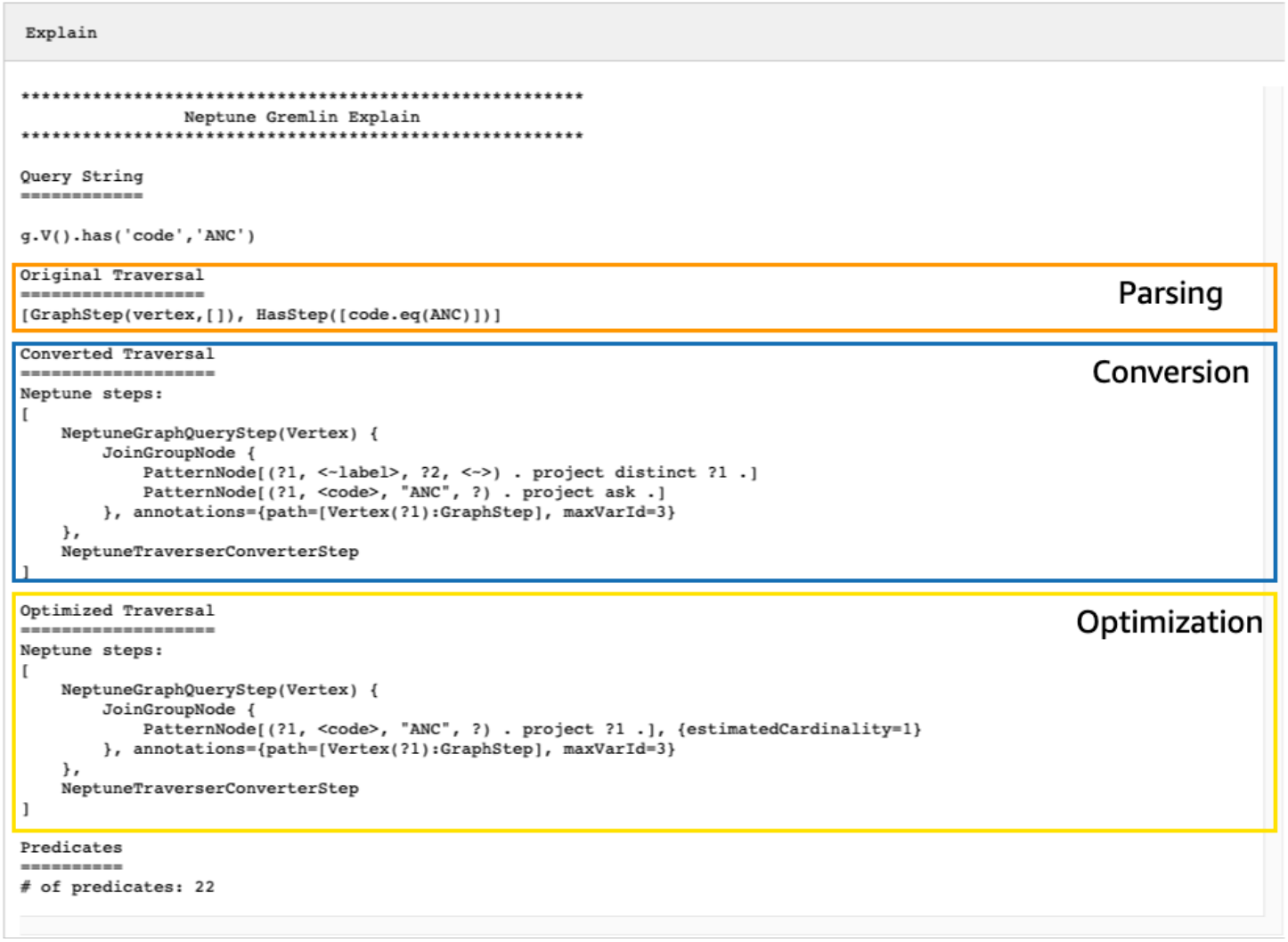
```
curl -X POST https://your-neptune-endpoint:port/gremlin/explain -d
 '{"gremlin":"g.V().has('code','ANC')"}'
```

2 番目は、`explain` パラメータで Neptune ワークベンチの %%gremlin セルマジックを使う方法です。これにより、セル本文に含まれるトラバーサルが Neptune `explain`API に渡され、セルを実行すると、結果の出力が表示されます。

```
%%gremlin explain

g.V().has('code','ANC')
```

結果としての `explain` API 出力は、Neptune のトラバーサルの実行プランを記述します。次の図に示すように、計画には、処理パイプラインに次の 3 つの各ステップが含まれています。



変換されないステップを見てトラバーサルを調整する。

Neptune `explain` API 出力で最初に探すものの 1 つは、Neptune ネイティブステップに変換されない Gremlin ステップ用です。クエリプランで、Neptune ネイティブステップに変換できないステップが検出されると、そのステップと計画内の後続のすべてのステップが Gremlin サーバーによって処理されます。

上記の例では、トラバーサル内のすべてのステップが変換されました。このトラバーサルの `explain` API 出力を調べてみましょう。

```
g.V().has('code','ANC').out().choose(hasLabel('airport'), values('code'), constant('Not
 an airport'))
```

下の画像でわかるように、Neptune は `choose()` ステップを変換できませんでした。

```
Explain
```

```
*******************************************************
                Neptune Gremlin Explain
*******************************************************

Query String
============

g.V().has('code','ANC').out().choose(hasLabel('airport'), values('code'), constant('Not an airport'))

Original Traversal
==================
[GraphStep(vertex,[]), HasStep([code.eq(ANC)]), VertexStep(OUT,vertex), ChooseStep([HasStep([~label.eq(airport)]), HasNex

Converted Traversal
===================
Neptune steps:
[
    NeptuneGraphQueryStep(Vertex) {
        JoinGroupNode {
            PatternNode[(?1, <~label>, ?2, <~>) . project distinct ?1 .]
            PatternNode[(?1, <code>, "ANC", ?) . project ask .]
            PatternNode[(?1, ?5, ?3, ?6) . project ?1,?3 . IsEdgeIdFilter(?6) .]
            PatternNode[(?3, <~label>, ?4, <~>) . project ask .]
        }, annotations={path=[Vertex(?1):GraphStep, Vertex(?3):VertexStep], maxVarId=7}
    },
    NeptuneTraverserConverterStep
]
+ not converted into Neptune steps: [ChooseStep([HasStep([~label.eq(airport)]), HasNextStep],[[(eq(true)), [PropertiesSte

Optimized Traversal
===================
Neptune steps:
[
    NeptuneGraphQueryStep(Vertex) {
        JoinGroupNode {
            PatternNode[(?1, <code>, "ANC", ?) . project ?1 .], {estimatedCardinality=1}
            PatternNode[(?1, ?5, ?3, ?6) . project ?1,?3 . IsEdgeIdFilter(?6) .], {estimatedCardinality=INFINITY}
        }, annotations={path=[Vertex(?1):GraphStep, Vertex(?3):VertexStep], maxVarId=7}
    },
    NeptuneTraverserConverterStep
]
+ not converted into Neptune steps: [ChooseStep([NeptuneHasStep([~label.eq(airport)]), HasNextStep],[[(eq(true)), [Proper

WARNING: >> ChooseStep([NeptuneHasStep([~label.eq(airport)]), HasNextStep],[[(eq(true)), [PropertiesStep([code],value), E

Predicates
==========
# of predicates: 26
```

トラバーサルのパフォーマンスを調整するには、いくつかの方法があります。最初の方法では、変換できなかったステップを排除するような方法で書き直します。もう 1 つの方法では、ステップをトラバーサルの最後に移動して、他のすべてのステップをネイティブステップに変換できるようにします。

変換されないステップを含むクエリプランは、常に調整する必要はありません。変換できないステップがトラバーサルの終わりにあり、グラフのトラバース方法ではなく出力のフォーマットに関連している場合、パフォーマンスにはほとんど影響しません。

Neptune `explain` APIからの出力を調べる他の方法で使うのは、インデックスを使用しないステップです。次のトラバーサルは、アンカレッジに着陸するフライトがあるすべての空港を検索します。

```
g.V().has('code','ANC').in().values('code')
```

このトラバーサルの explain API からの出力は次のとおりです。

```
*******************************************************
                Neptune Gremlin Explain
*******************************************************

Query String
============

g.V().has('code','ANC').in().values('code')

Original Traversal
==================
[GraphStep(vertex,[]), HasStep([code.eq(ANC)]), VertexStep(IN,vertex),
 PropertiesStep([code],value)]

Converted Traversal
===================
Neptune steps:
[
    NeptuneGraphQueryStep(PropertyValue) {
        JoinGroupNode {
            PatternNode[(?1, <~label>, ?2, <~>) . project distinct ?1 .]
            PatternNode[(?1, <code>, "ANC", ?) . project ask .]
            PatternNode[(?3, ?5, ?1, ?6) . project ?1,?3 . IsEdgeIdFilter(?6) .]
            PatternNode[(?3, <~label>, ?4, <~>) . project ask .]
            PatternNode[(?3, ?7, ?8, <~>) . project ?3,?8 . ContainsFilter(?7 in
 (<code>)) .]
        }, annotations={path=[Vertex(?1):GraphStep, Vertex(?3):VertexStep,
 PropertyValue(?8):PropertiesStep], maxVarId=9}
    },
    NeptuneTraverserConverterStep
]

Optimized Traversal
===================
Neptune steps:
[
    NeptuneGraphQueryStep(PropertyValue) {
        JoinGroupNode {
```

```
            PatternNode[(?1, <code>, "ANC", ?) . project ?1 .],
 {estimatedCardinality=1}
            PatternNode[(?3, ?5, ?1, ?6) . project ?1,?3 . IsEdgeIdFilter(?6) .],
 {estimatedCardinality=INFINITY}
            PatternNode[(?3, ?7=<code>, ?8, <~>) . project ?3,?8 .],
 {estimatedCardinality=7564}
       }, annotations={path=[Vertex(?1):GraphStep, Vertex(?3):VertexStep,
 PropertyValue(?8):PropertiesStep], maxVarId=9}
    },
    NeptuneTraverserConverterStep
]

Predicates
==========
# of predicates: 26

WARNING: reverse traversal with no edge label(s) - .in() / .both() may impact query
 performance
```

出力の下部にある WARNING メッセージが発生するのは、トラバーサルの in() ステップは、Neptune が維持する 3 つのインデックスのいずれかを使用して処理することができないからです (Neptune でステートメントのインデックスを作成する方法 および Neptune の Gremlin ステートメント を参照)。in() ステップにはエッジフィルターが含まれていないため、SPOG、POGSまたはGPSO インデックスを使って解決できないのです。代わりに、Neptune は要求された頂点を見つけるためにユニオンスキャンを実行しなければならず、これは非常に非効率です。

この状況において、トラバーサルを調整するには、2 つの方法があります。1 つ目は、インデックス付きルックアップを使用してクエリを解決できるように、1 つ以上のフィルタ条件を in() ステップに追加します。上記の例で、次のようになります。

```
g.V().has('code','ANC').in('route').values('code')
```

修正済みトラバーサルに対する Neptune explain API からの出力には、もう WARNING メッセージは含まれていません。

```
*******************************************************
                Neptune Gremlin Explain
*******************************************************

Query String
============
```

```
g.V().has('code','ANC').in('route').values('code')

Original Traversal
==================
[GraphStep(vertex,[]), HasStep([code.eq(ANC)]), VertexStep(IN,[route],vertex),
 PropertiesStep([code],value)]

Converted Traversal
===================
Neptune steps:
[
    NeptuneGraphQueryStep(PropertyValue) {
        JoinGroupNode {
            PatternNode[(?1, <~label>, ?2, <~>) . project distinct ?1 .]
            PatternNode[(?1, <code>, "ANC", ?) . project ask .]
            PatternNode[(?3, ?5, ?1, ?6) . project ?1,?3 . IsEdgeIdFilter(?6) .
 ContainsFilter(?5 in (<route>)) .]
            PatternNode[(?3, <~label>, ?4, <~>) . project ask .]
            PatternNode[(?3, ?7, ?8, <~>) . project ?3,?8 . ContainsFilter(?7 in
 (<code>)) .]
       }, annotations={path=[Vertex(?1):GraphStep, Vertex(?3):VertexStep,
 PropertyValue(?8):PropertiesStep], maxVarId=9}
    },
    NeptuneTraverserConverterStep
]

Optimized Traversal
===================
Neptune steps:
[
    NeptuneGraphQueryStep(PropertyValue) {
        JoinGroupNode {
            PatternNode[(?1, <code>, "ANC", ?) . project ?1 .],
 {estimatedCardinality=1}
            PatternNode[(?3, ?5=<route>, ?1, ?6) . project ?1,?3 . IsEdgeIdFilter(?
6) .], {estimatedCardinality=32042}
            PatternNode[(?3, ?7=<code>, ?8, <~>) . project ?3,?8 .],
 {estimatedCardinality=7564}
       }, annotations={path=[Vertex(?1):GraphStep, Vertex(?3):VertexStep,
 PropertyValue(?8):PropertiesStep], maxVarId=9}
    },
    NeptuneTraverserConverterStep
]
```

```
Predicates
==========
# of predicates: 26
```

この種のトラバーサルを多数実行している場合に取る方法は、有効なオプショナル OSGP インデックスを持つ Neptune DB クラスターでそれらを実行することです (OSGP インデックスの有効化 を参照)。OSGP インデックスの有効化には欠点があります。

- データをロードする前に、DB クラスターで有効にする必要があります。
- 頂点とエッジの挿入速度が最大 23% 遅くなる場合があります。
- ストレージ使用量は約 20% 増加します。
- すべてのインデックスにリクエストを分散する読み取りクエリでは、レイテンシーが増加する可能性があります。

OSGP インデックスがあることは、制限された一連のクエリパターンに対して非常に意味がありますが、それらを頻繁に実行しない限り、3 つのプライマリインデックスを使用して記述するトラバーサルを確実に解決できるようにすることをお勧めします。

多数の述語を使用する

Neptune は、グラフ内の各エッジラベルと各個別の頂点またはエッジプロパティ名を述語として扱い、デフォルトで異なる述語の数が比較的少なくなるように設計されています。グラフデータに数千を超える述語が含まれていると、パフォーマンスが低下する可能性があります。

次のような場合は、Neptune `explain` 出力によって警告が表示されます。

```
Predicates
==========
# of predicates: 9549
WARNING: high predicate count (# of distinct property names and edge labels)
```

ラベルとプロパティの数、これにしたがった述語数を減らすためにデータモデルを再加工するのが容易でない場合、トラバーサルを調整する最良の方法は、上で説明したように、有効化した OSGP インデックスを持つ DB クラスタ内でこれらを実行することです。

Neptune Gremlin **`profile`** API を使用してトラバーサルを調整する

Neptune `profile` API は Gremlin `profile()` ステップとはかなり異なります。`explain` API のように、その出力には、Neptune エンジンがトラバーサルの実行時に使用するクエリプランが含まれ

ます。また、profile 出力には、パラメーターの設定方法に基づいて、トラバーサルの実際の実行統計が含まれます。

アンカレッジのすべての空港の頂点を見つける次の単純なトラバーサルを考えます。

```
g.V().has('code','ANC')
```

explain API の場合は、profile REST 呼び出しを使用する API を呼び出せます。

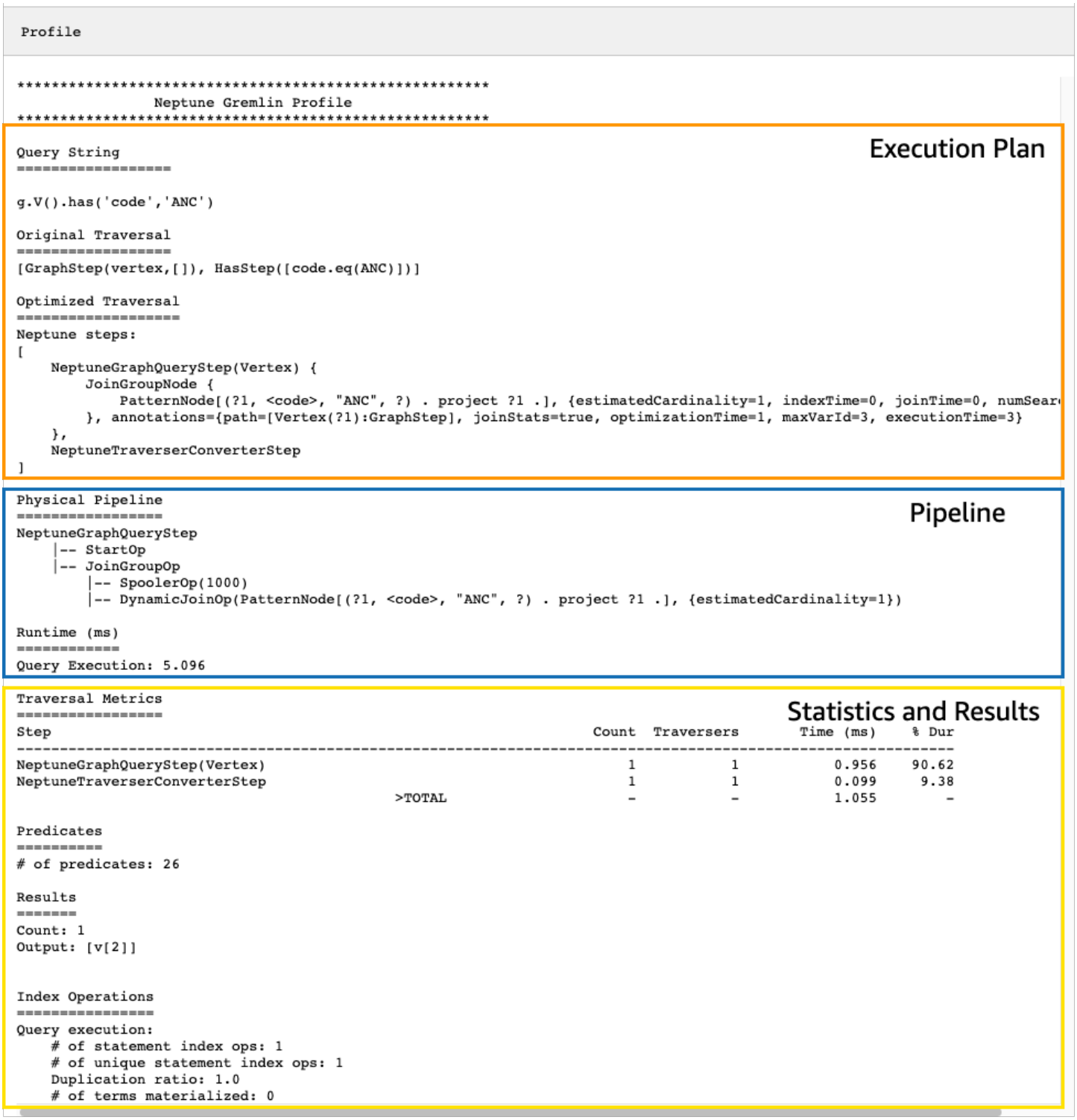
```
curl -X POST https://your-neptune-endpoint:port/gremlin/profile -d
 '{"gremlin":"g.V().has('code','ANC')"}'
```

2番目は、profile パラメータで Neptune ワークベンチの %%gremlin セルマジックを使う方法です。これにより、セル本文に含まれるトラバーサルが Neptune profileAPI に渡され、セルを実行すると、結果の出力が表示されます。

```
%%gremlin profile

g.V().has('code','ANC')
```

結果としての profile API 出力には、次の図に示すように、Neptune のトラバーサル実行プランとプランの実行に関する統計の両方が含まれます。



`profile` 出力の場合、実行計画セクションにはトラバーサルの最終実行計画のみが含まれ、中間ステップは含まれません。パイプラインセクションには、実行された物理パイプライン操作と、トラバーサル実行にかかった実際の時間 (ミリ秒単位) が含まれます。ランタイムメトリクスは、2 つの異なるバージョンのトラバーサルの最適化にかかる時間を比較する際に非常に役立ちます。

> **Note**
>
> トラバーサルの初期ランタイムは、通常、後続のランタイムよりも長くなります。これは、最初のランタイムによって関連するデータがキャッシュされるためです。

profile 出力の第 3 セクションには、実行統計とトラバーサルの結果が含まれます。この情報がトラバーサルのチューニングにどのように役立つかを確認するには、次のトラバーサルを検討してください。このトラバーサルでは、名前が「Anchora」で始まるすべての空港と、それらの空港から 2 回のホップで到達可能なすべての空港、戻り空港コード、飛行ルート、および距離が検索されます。

```
%%gremlin profile

g.withSideEffect("Neptune#fts.endpoint", "{your-OpenSearch-endpoint-URL").
    V().has("city", "Neptune#fts Anchora~").
    repeat(outE('route').inV().simplePath()).times(2).
    project('Destination', 'Route').
        by('code').
        by(path().by('code').by('dist'))
```

Neptune **profile** API 出力のトラバーサルメトリクス

すべての profile 出力で使用できる最初のメトリクスセットは、トラバーサルメトリクスです。これらは Gremlin profile() ステップメトリクスに似ていますが、いくつかの違いがあります。

```
Traversal Metrics
=================
Step                                                               Count  Traversers
    Time (ms)    % Dur
-------------------------------------------------------------------------------------------------------
NeptuneGraphQueryStep(Vertex)                                       3856        3856
       91.701     9.09
NeptuneTraverserConverterStep                                       3856        3856
       38.787     3.84
ProjectStep([Destination, Route],[value(code), ...                  3856        3856
      878.786    87.07
  PathStep([value(code), value(dist)])                              3856        3856
      601.359
                                            >TOTAL                   -           -
     1009.274        -
```

トラバーサルメトリクステーブルの最初の列には、トラバーサルによって実行されるステップがリストされます。最初の 2 つのステップは、一般に Neptune 固有のステップ NeptuneGraphQueryStep および NeptuneTraverserConverterStep です。

NeptuneGraphQueryStep は、Neptune エンジンによってネイティブに変換および実行できるトラバーサルの全体の実行時間を表します。

NeptuneTraverserConverterStep は、変換されたステップの出力を TinkerPop トラバーサーに変換するプロセスを表します。トラバーサーを使用すると、変換できなかったステップを処理したり、結果を TinkerPop互換形式で返したりすることができます。

上記の例では、変換されていないステップがいくつかあるため、これらの各 TinkerPop ステップ (ProjectStep、PathStep) がテーブルに行として表示されます。

プロファイルステップのTinkerPopドキュメント で説明されているようにCount、テーブルの 2 番目の列はステップを通過したトラバーサーの数を報告し、3 番目の列 はステップを通過したトラバーサーの数Traversersを報告します。

この例では、3,856 個の頂点と 3,856 個のトラバーサーが NeptuneGraphQueryStep で返され、これらの数字は残りの処理を通して同じままです。これは ProjectStep および PathStep 結果をフィルタリングするのではなく、フォーマットしているからです。

> **Note**
>
> とは異なり TinkerPop、Neptune エンジンは NeptuneGraphQueryStepおよび NeptuneTraverserConverterStep ステップを一括処理してパフォーマンスを最適化しません。一括処理は、 TinkerPopトラバーサーを同じ頂点に組み合わせて運用オーバーヘッドを削減するオペレーションであり、 Countと のTraversers数値が異なる原因となります。一括処理は、Neptune が に委任するステップでのみ発生するため TinkerPop、Neptune がネイティブに処理するステップでは発生しないため、 列Countと Traverser列はほとんど異なります。

時間]列には、ステップが要したミリ秒数が報告され、% Dur 列には、ステップが要した合計処理時間の割合が報告されます。これらは、最も時間がかかったステップを示すことで、チューニング作業を集中させる場所を示すメトリクスです。

Neptune **profile** API 出力でのインデックスオペレーションメトリクス

Neptune プロファイル API の出力にあるもう 1 つのメトリクスセットは、インデックスオペレーションです。

```
Index Operations
================
Query execution:
    # of statement index ops: 23191
    # of unique statement index ops: 5960
    Duplication ratio: 3.89
    # of terms materialized: 0
```

次のレポート:

- インデックスルックアップの合計数。
- 実行された一意のインデックスルックアップの数。
- 一意のインデックスルックアップに対する総インデックスルックアップの比率。比が低いほど、冗長性が低くなります。
- 用語辞書からマテリアライズされた用語の数。

Neptune **profile** API 出力のリピートメトリクス

トラバーサルが上記の例のように `repeat()` ステップを実行すると、リピートメトリックを含むセクションが`profile`出力に表示されます。

```
Repeat Metrics
==============
Iteration  Visited   Output    Until     Emit      Next
-------------------------------------------------------
        0        2        0        0        0        2
        1       53        0        0        0       53
        2     3856     3856     3856        0        0
-------------------------------------------------------
             3911     3856     3856        0       55
```

次のレポート:

- 行のループカウント (`Iteration`列)。
- ループが訪問した要素の数 (`Visited`列)。

- ループが出力した要素の数 (Output列)。
- ループが出力した最後の要素 (Until列)。
- ループが発した要素の数 (Emit列)。
- ループから後続のループに渡される要素の数 (Next列)。

これらのリピートメトリクスは、トラバーサルの分岐係数を理解し、データベースによって処理されている作業量を把握するのに非常に役立ちます。これらの数値を使用して、パフォーマンスの問題を診断できます。特に、同じトラバーサルが異なるパラメータで劇的に異なる場合です。

Neptune **`profile`** API 出力のフルテキスト検索メトリクス

トラバーサルがフルテキスト検索ルックアップを実行すると、上記の例のように、フルテキスト検索 (FTS) メトリクスを含むセクションが `profile` 出力に表示されます。

```
FTS Metrics
==============
SearchNode[(idVar=?1, query=Anchora~, field=city) . project ?1 .],
    {endpoint=your-OpenSearch-endpoint-URL, incomingSolutionsThreshold=1000,
 estimatedCardinality=INFINITY,
    remoteCallTimeSummary=[total=65, avg=32.500000, max=37, min=28],
    remoteCallTime=65, remoteCalls=2, joinTime=0, indexTime=0, remoteResults=2}

    2 result(s) produced from SearchNode above
```

これは、 ElasticSearch (ES) クラスターに送信されたクエリを示し、フルテキスト検索に関連するパフォーマンスの問題を特定する ElasticSearch のに役立つ とのやり取りに関するいくつかのメトリクスをレポートします。

- ElasticSearch インデックスへの呼び出しに関する概要情報 :
  - すべての remoteCalls がクエリを満たすのに必要な合計ミリ秒数 (`total`)。
  - remoteCall で費やされた平均ミリ秒数 (`avg`)。
  - remoteCall で費やされた最低ミリ秒数 (`min`)。
  - remoteCall で費やされた最大ミリ秒数 (`max`)。
- remoteCalls が ElasticSearch () に消費した合計時間`remoteCallTime`。
- ElasticSearch () に対して行われた remoteCalls の数`remoteCalls`。
- ElasticSearch 結果の結合に費やされたミリ秒数 (`joinTime`) 。
- インデックスルックアップに費やされたミリ秒数 (`indexTime`)。

- ElasticSearch () によって返された結果の合計数`remoteResults`。

Amazon Neptune でのネイティブ Gremlin ステップサポート

Amazon Neptune エンジンでは、Gremlin クエリのチューニング で説明されているように、すべての Gremlin ステップに対する完全なネイティブサポートはありません。。現在のサポートは 4 つのカテゴリに分類されます。

- 常にネイティブ Neptune エンジンオペレーションに変換できる Gremlin ステップ
- 場合によってはネイティブ Neptune エンジンオペレーションに変換できる Gremlin ステップ
- ネイティブ Neptune エンジンオペレーションに変換されない Gremlin ステップ
- Neptune ではまったくサポートされていない Gremlin ステップ

常にネイティブ Neptune エンジンオペレーションに変換できる Gremlin ステップ

多くの Gremlin ステップは、次の条件を満たす限り、ネイティブ Neptune エンジンオペレーションに変換できます。

- クエリでは、変換できないステップの前には表示されません。
- 親ステップがあれば、その親ステップは変換できます。
- もしあれば、すべての子トラバーサルは変換できます。

以下の Gremlin ステップは、次の条件を満たす限り、ネイティブ Neptune エンジンオペレーションに変換できます。

- and( )
- as( )
- count( )
- E( )
- emit( )
- explain( )
- group( )
- groupCount( )
- has( )

- identity( )
- is( )
- key( )
- label( )
- limit( )
- local( )
- loops( )
- not( )
- or( )
- profile( )
- properties( )
- subgraph( )
- until( )
- V( )
- value( )
- valueMap( )
- values( )

場合によってはネイティブ Neptune エンジンオペレーションに変換できる Gremlin ステップ

一部の Gremlin ステップは、状況によってはネイティブ Neptune エンジンオペレーションに変換できますが、他の状況では変換できません。

- addE( ) – トラバーサルをキーとして含む `property()` ステップが直後に続く場合を除き、`addE()` ステップは、通常ネイティブ Neptune エンジンオペレーションに変換できます。
- addV( ) – トラバーサルをキーとして含む `property()` ステップが直後に続く場合を除き、または複数のラベルが割り当てられる場合を除き、`addV()` ステップは、通常ネイティブ Neptune エンジンオペレーションに変換できます。
- aggregate( ) – 子トラバーサルまたはサブトラバーサルで使用されている場合を除き、または格納される値が頂点、エッジ、ID、ラベル、またはプロパティ値以外のものでない限り、通常 `aggregate()` ステップは、ネイティブ Neptune エンジンオペレーションに変換できます。

  次の例では、子トラバーサルで使用されているため、`aggregate()` は変換されません。

```
g.V().has('code','ANC').as('a')
     .project('flights').by(select('a')
     .outE().aggregate('x'))
```

この例では、格納される値が min() であるため、aggregate () は変換されません。

```
g.V().has('code','ANC').outE().aggregate('x').by(values('dist').min())
```

- barrier( ) – barrier()ステップは、その後のステップが変換されない限り、通常ネイティブ Neptune エンジンオペレーションに変換できます。
- cap( ) – cap() ステップが変換される唯一のケースは、unfold() ステップと組み合わせて頂点、エッジ、ID、またはプロパティ値の集約の展開バージョンを返す場合です。この例では、次に .unfold() が後続するので、cap() は変換されます。

```
g.V().has('airport','country','IE').aggregate('airport').limit(2)
     .cap('airport').unfold()
```

ただし、.unfold() を削除すると、cap() は変換されません。

```
g.V().has('airport','country','IE').aggregate('airport').limit(2)
     .cap('airport')
```

- coalesce( ) – coalesce() ステップが変換される唯一のケースは、TinkerPop レシピページ で推奨されるアップサートパターンに従う場合です。その他の coalesce() パターンは使用できません。変換は、すべての子トラバーサルが変換できる場合に限られ、それらはすべて出力と同じタイプ (頂点、エッジ、ID、値、キー、またはラベル) を生成し、新しい要素にトラバースし、repeat() ステップは含みません。
- constant( ) – constant () ステップは現在、次のように、定数値を割り当てるためのトラバーサルの sack().by() 一部内で使われる場合にのみ変換されます。

```
g.V().has('code','ANC').sack(assign).by(constant(10)).out().limit(2)
```

- cyclicPath( ) – cyclicPath() ステップは、それが by()、from()、to() モジュレータのいずれかと使われている場合を除き、通常ネイティブ Neptune エンジンオペレーションに変換できます。次のクエリでは、たとえば、cyclicPath() は変換されません。

```
g.V().has('code','ANC').as('a').out().out().cyclicPath().by('code')
g.V().has('code','ANC').as('a').out().out().cyclicPath().from('a')
```

```
g.V().has('code','ANC').as('a').out().out().cyclicPath().to('a')
```

- drop( ) – drop() ステップは、それが sideEffect(または optional() ステップのいずれかで使われている場合を除き、通常、ネイティブ Neptune エンジンオペレーションに変換できます。
- fold( ) – fold() ステップを変換できる状況は 2 つだけです。つまり、TinkerPop レシピページ で推奨されるアップサートパターンで使用される場合と、次のようなgroup().by()コンテキストで使用される場合です。

```
g.V().has('code','ANC').out().group().by().by(values('code', 'city').fold())
```

- id( ) – id() ステップは、プロパティで使用されない限り、次のように変換されます。

```
g.V().has('code','ANC').properties('code').id()
```

- order( ) – order() ステップは、その後のステップが変換されない限り、通常 ネイティブ Neptune エンジンオペレーションに変換できます。
  - order() ステップは、次のようにネストされた子トラバーサル内にあります。

```
g.V().has('code','ANC').where(V().out().order().by(id))
```

  - たとえば、order(local) で、ローカル順序付けが使用されています。
  - カスタムコンパレータは、by() モジュレーションで順序付けするために使われます。一例として、この使用法sack()があります。

```
g.withSack(0).
  V().has('code','ANC').
      repeat(outE().sack(sum).by('dist').inV()).times(2).limit(10).
      order().by(sack())
```

  - 同じ要素に複数の順序があります。
- project( ) – project()に従うby()ステートメントの数が、次のように、指定されたラベルの数と一致しないのでなければ、project()ステップは、通常ネイティブ Neptune エンジンオペレーションに変換できます。

```
g.V().has('code','ANC').project('x', 'y').by(id)
```

- range( ) – range()ステップは、対象範囲の下端がゼロの場合にのみ変換されます (たとえば、range(0,3))。

- repeat( ) – 次のように別のrepeat( )ステップノード内にネストされている場合を除き、repeat( )ステップは、通常ネイティブ Neptune エンジンオペレーションに変換できます。

```
g.V().has('code','ANC').repeat(out().repeat(out()).times(2)).times(2)
```

- sack( ) – sack( )ステップは、次の場合を除き、通常ネイティブ Neptune エンジンオペレーションに変換できます。
  - 数値以外のサック演算子が使用されている場合。
  - +、-、mult、div、min および max 以外の数値サック演算子が使用されている場合。
  - 次のように、サック値に基づいてフィルタリングするために where( ) ステップの中で sack( ) が使われる場合。

```
g.V().has('code','ANC').sack(assign).by(values('code')).where(sack().is('ANC'))
```

- sum( ) – sum( ) ステップは、通常、ネイティブ Neptune エンジンオペレーションに変換できますが、次のようにグローバル総和の計算に使用される場合は変換できません。

```
g.V().has('code','ANC').outE('routes').values('dist').sum()
```

- union() — union( ) ステップは、ターミナルステップ以外のクエリの最後のステップである限り、ネイティブ Neptune エンジン操作に変換できます。
- unfold( ) – unfold( ) ステップは、TinkerPopレシピページ で推奨されているアップサートパターンで使用されている場合、およびcap( )次のように一緒に使用されている場合にのみ、ネイティブ Neptune エンジンオペレーションに変換できます。

```
g.V().has('airport','country','IE').aggregate('airport').limit(2)
     .cap('airport').unfold()
```

- where( ) – where( ) ステップは、次の場合を除き、通常 ネイティブ Neptune エンジンオペレーションに変換できます。
  - 次のように、by () モジュレーションを使用する場合。

```
g.V().hasLabel('airport').as('a')
     .where(gt('a')).by('runways')
```

  - eq、neq、within、および without 以外の比較演算子が用いられる場合。
  - ユーザー指定の集計が使用される場合。

ネイティブ Neptune エンジンオペレーションに変換されない Gremlin ステップ

次の Gremlin ステップは Neptune でサポートされていますが、ネイティブ Neptune エンジンオペレーションに変換されることはありません。代わりに、Gremlin サーバーによって実行されます。

- choose( )
- coin( )
- inject( )
- match( )
- math( )
- max( )
- mean( )
- min( )
- option( )
- optional( )
- path( )
- propertyMap( )
- sample( )
- skip( )
- tail( )
- timeLimit( )
- tree( )

Neptune ではまったくサポートされていない Gremlin ステップ

次の Gremlin ステップは、Neptune ではまったくサポートされていません。ほとんどの場合、これは `GraphComputer` が必要なためで、これは Neptune が現在サポートしていません。

- connectedComponent( )
- io( )
- shortestPath( )
- withComputer( )

- pageRank( )
- peerPressure( )
- program( )

`io( )` ステップは実際には部分的にサポートされていて、URL からの `read( )` には使用できますが、`write( )` には使用できません。

Gremlin と Neptune DFE クエリエンジンを使用する

ラボモードで Neptune 代替クエリエンジン、いわゆる DFE を完全に有効にしたら (`neptune_lab_mode`DB クラスターのパラメータから`DFEQueryEngine=enabled`に設定する)、Neptune は読み取り専用の Gremlin クエリ/トラバーサルを中間論理表現に変換し、可能な限り DFE エンジン上で実行します。

ただし、DFE はまだ Gremlin のすべてのステップをサポートしていません。ステップを DFE でネイティブに実行できない場合、Neptune はステップを実行する TinkerPop ためにフォールバックします。これが発生すると、`explain` および `profile` レポートに警告が含まれます。

> **Note**
>
> エンジンリリース 1.0.5.0 から、ネイティブサポートなしで Gremlin ステップを処理するためのデフォルトの DFE 動作が変更されました。以前は DFE エンジンが Neptune Gremlin エンジンにフォールバックしていましたが、現在は vanilla TinkerPop エンジンにフォールバックしています。

DFE エンジンでネイティブにサポートされている Gremlin ステップ

- **`GraphStep`**
- **`VertexStep`**
- **`EdgeVertexStep`**
- **`IdStep`**
- **`TraversalFilterStep`**
- **`PropertiesStep`**
- **`HasStep`** プロパティと ID とラベルの頂点とエッジのフィルタリングサポート (テキストと `Without` 述語を除く)。

- **WherePredicateStep** と Path-スコープ付きフィルタ。ただし、ByModulation、SideEffect または Map ルックアップサポートなし
- **DedupGlobalStep**、ただし ByModulation、SideEffect および Map ルックアップサポートは除く。

クエリ計画インターリーブ

変換プロセスで、対応するネイティブ DFE 演算子を持たない Gremlin ステップが検出されると、Tinkerpop の使用にフォールバックする前に、DFE エンジンでネイティブに実行できる他の中間クエリ部分を見つけようとします。これは、最上位レベルのトラバーサルにインターリーブロジックを適用することによって行われます。その結果、サポートされているステップは可能な限り使用されます。

このような中間的な非プレフィックスクエリ変換は、NeptuneInterleavingStep およびexplain 出力にある profile を使って表されます。

パフォーマンスの比較のために、DFE エンジンを使用してプレフィックス部分を実行しながら、クエリでのインターリーブをオフにしたい場合があります。または、プレフィックス以外のクエリ実行には TinkerPop エンジンのみを使用することをお勧めします。これを行うには、disableInterleaving クエリヒントを使用します。

useDFE 値の false クエリヒントにより DFE でクエリがまったく実行されなくなるように、disableInterleaving 値の true クエリヒントはクエリの変換の DFE インターリーブをオフにします。例:

```
g.with('Neptune#disableInterleaving', true)
 .V().has('genre','drama').in('likes')
```

Gremlin **explain** および **profile** 出力の更新

Gremlin explain は、Neptune がクエリの実行に使用する最適化されたトラバーサルの詳細を示します。DFE エンジンが無効化されているときの explain 出力はどのようなものかの例について、サンプル DFE explain 出力をご覧ください。

Gremlin profile API は指定された Gremlin トラバーサルを実行し、実行に関するさまざまなメトリクスを収集して、最適化されたクエリプランの詳細と、さまざまな演算子の実行時統計情報を含むプロファイルレポートを生成します。profile DFE エンジンが無効化されているときの出力はどのようなものかの例について、サンプル DFE profile 出力をご覧ください。

> **Note**
>
> DFE はラボモードでリリースされる実験的な機能なので、`explain` および `profile` 出力の正確な形式は変更される可能性があります。

openCypher でNeptune グラフにアクセスする

Neptune は、openCypher を使用したグラフアプリケーションの構築をサポートしています。これは、現在、グラフデータベースを操作する開発者にとって最も人気のあるクエリ言語の 1 つです。開発者、ビジネスアナリスト、データサイエンティストは、openCypher の SQL の影響が大きい構文を好みます。これは、グラフアプリケーションのクエリを作成するためによく知られた構造を提供するからです。

openCypher は、プロパティグラフの宣言型クエリ言語です。当初は Neo4j が 開発し、その後 2015 年にオープンソース化され、Apache 2 オープンソースライセンスの下で openCypher プロジェクトで活用されました。構文は Cypher クエリ言語リファレンス、バージョン 9 で説明されています。

openCypher 仕様の Neptune サポートの制限と相違点については、「Amazon Neptune での openCypher 仕様コンプライアンス Amazon Neptune」を参照してください。

> **Note**
>
> Cypher クエリ言語の現在の Neo4j 実装は、openCypher 仕様とはいくつかの点で異なっています。現在の Neo4j Cypher コードを Neptune に移行する場合、詳細については,「Neptune の Neo4j との互換性」と「Neptune 上の openCypherで実行するように Cypher クエリを書き直す」を参照してください。

エンジンリリース 1.1.1.0 以降、openCypher は Neptune での本番使用が可能になりました。

Gremlin 対 openCypher：類似点と相違点

Gremlin と openCypher はどちらもプロパティグラフクエリ言語であり、多くの点で補完的です。

Gremlin は、プログラマーに魅力的で、コードにシームレスに収まるように設計されました。その結果、Gremlin は設計上命令的ですが、OpenenCypher の宣言構文は SQL や SPARQL の経験を持つ人々はより身近に感じるかもしれません。Gremlin は Jupyter ノートブックで Python を使用する

データサイエンティストにとってより自然に見えるかもしれませんが、openCypher は SQL の背景を持つビジネスユーザーにはより直感的だと思えるかもしれません。

うれしいことに Neptune でGremlin と openCypher いずれかを選ぶ必要はありません。どちらの言語でも、データの入力に使用された 2 つの言語に関係なく、同じグラフでクエリを実行できます。何をするかによって、いくつかのことには Gremlinを、他のことには openCypher を、と使い分ける方が便利になるかもしれません。

Gremlin は、一連のステップでグラフ内を移動する方法を制御できる命令構文を使用します。各ステップは、データのストリームを取り込み、そのデータに対して何らかのアクション (フィルター、マップなどを使用して) を実行し、結果を次のステップに出力します。Gremlin クエリは、通常、`g.V()` 形式をとります。そして追加のステップが続きます。

openCypher では、SQL の影響が大きい宣言構文を使用します。この構文は、`(()-[]->()` のような) モチーフ構文を使用してグラフ内で検索するノードと関係のパターンを指定します。openCypher クエリは、多くの場合、`MATCH` 句、それに `WHERE`、`WITH`、および `RETURN` のような他の句が続きます。

openCypher の使用を開始する方法

Neptune のプロパティグラフデータは、ロードされた方法に関係なく openCypher を使用してクエリできますが、RDF としてロードされたデータのクエリに openCypher を使用することはできません。

Neptune バルクローダーは、プロパティグラフデータをGremlin の CSV 形式と、openCypher の CSV 形式で受領します。また、もちろん、Gremlin や openCypher クエリを使用して、プロパティデータをグラフに追加することもできます。

Cypher クエリ言語を学習するためのオンラインチュートリアルが数多く用意されています。ここでは、openCypher クエリの簡単な例をいくつか挙げていますので、言語を理解するのに役立つかもしれませんが、openCypher を使用して Neptune グラフをクエリする最も簡単で簡単な方法は、openCypher ノートブックをNeptune ワークベンチで使うことです。ワークベンチはオープンソースで、https://github.com/aws-samples/amazon-neptune-samples GitHub でホストされています。

openCypher ノートブックは、 GitHub Neptune グラフノートブックリポジトリ にあります。特に、openCypher のエアルートの可視化、およびイングリッシュ・プレミア・チームのノートブックをご確認ください。。

openCypher によって処理されるデータは、順序付けられていない一連のキー/値マップの形式をとります。これらのマップを調整、操作、および拡張する主な方法は、キーと値のペアに対してパターンマッチング、挿入、更新、削除などのタスクを実行する句を使用することです。

openCypher には、グラフ内のデータパターンを見つけるためのいくつかの句があり、そのうちの MATCH が最も一般的です。MATCH グラフ内で検索するノード、リレーションシップ、およびフィルタのパターンを指定できます。例:

- すべてのノードを取得する

```
MATCH (n) RETURN n
```

- 接続されたノードを検索する

```
MATCH (n)-[r]->(d) RETURN n, r, d
```

- パスを見つける

```
MATCH p=(n)-[r]->(d) RETURN p
```

- ラベル付きのすべてのノードを取得する

```
MATCH (n:airport) RETURN n
```

上記の最初のクエリはグラフ内のすべてのノードを返し、次の 2 つはリレーションシップを持つすべてのノードを返すことに注意してください。これは一般的にはお勧めしません！ほとんどの場合、返されるデータを絞り込む必要があります。これは、4 番目の例のように、ノードまたはリレーションシップのラベルとプロパティを指定することで実行できます。

openCypher 構文の便利なチートシートは Neptune github サンプルリポジトリにあります。

Neptune openCypher ステータスサーブレットとステータスエンドポイント

openCypher ステータスエンドポイントは、サーバー上で現在実行されている、または実行を待っているクエリに関する情報へのアクセスを提供します。また、これらのクエリをキャンセルすることもできます。エンドポイントは、次の通りです。

```
https://(the server):(the port number)/openCypher/status
```

サーバーから現在のステータスを取得する、またはクエリをキャンセルするHTTP GET および POST メソッドを使用できます。また、実行中のクエリまたは待機中のクエリをキャンセルする DELETE メソッドを使用することもできます。

ステータスリクエストのパラメータ

ステータスクエリパラメータ

- **includeWaiting** (true または false) — true に設定し、かつその他のパラメータが存在しない場合は、待機中のクエリと実行中のクエリのステータス情報が返されます。
- **cancelQuery** — GET および POST メソッドを使用してのみ使用し、これがキャンセル要求であることを示します。DELETE メソッドはこのパラメータを必要としません。

  cancelQuery パラメータの値は使用されませんが、cancelQuery が存在する場合、キャンセルするクエリを識別するには、queryId パラメータが必要です。
- **queryId** — 特定のクエリの ID が含まれます。

  GET または POST メソッドと併用およびし、なおかつ cancelQuery パラメータが存在しない場合 queryId は、識別する特定のクエリのステータス情報を返します。cancelQuery パラメータが存在する場合、queryId が特定するクエリはキャンセルされます。

  DELETE メソッドと併用する場合、queryId は常に特定のクエリがキャンセルされることを示します。
- **silent** — クエリをキャンセルするときにのみ使用されます。true に設定されている場合、キャンセルは予告なしに行われます。

ステータスリクエストのレスポンスフィールド

特定のクエリの ID が指定されていない場合、ステータスレスポンスフィールドは表示されない

- acceptedQueryCount – キュー内のクエリを含む、受け入れられたが、まだ完了していないクエリの数。
- runningQueryCount – 現在実行中の openCypher クエリの数。
- queries – 現在の openCypher クエリのリスト。

特定のクエリのステータスレスポンスフィールド

- queryId – クエリの GUID ID。Neptune は、この ID 値を各クエリに自動的に割り当てます。または、独自の ID を割り当てることもできます (Neptune Gremlin または SPARQL クエリにカスタム ID を挿入するを参照)。
- queryString – 送信されたクエリ。それより長い場合、これは 1024 文字に切り捨てられます。
- queryEvalStats – このクエリの統計 :
  - waited - クエリが待機していた時間をミリ秒単位で示します。
  - elapsed – これまでクエリが実行されていた時間 (マイクロ秒)。
  - cancelled — `True` はクエリがキャンセルされたことを示し、`False` はキャンセルされていないことを示します。

ステータスリクエストとレスポンスの例

- 待機中のクエリを含む、すべてのクエリのステータスを要求します。

```
curl https://server:port/openCypher/status \
  --data-urlencode "includeWaiting=true"
```

レスポンス:

```
{
  "acceptedQueryCount" : 0,
  "runningQueryCount" : 0,
  "queries" : [ ]
}
```

- 待機中のクエリを含まない、すべてのクエリのステータスを要求します。

```
curl https://server:port/openCypher/status
```

レスポンス:

```
{
  "acceptedQueryCount" : 0,
  "runningQueryCount" : 0,
  "queries" : [ ]
```

```
}
```

- 1 つのクエリのステータスをリクエストします。

```
curl https://server:port/openCypher/status \
 --data-urlencode "queryId=eadc6eea-698b-4a2f-8554-5270ab17ebee"
```

レスポンス:

```
{
  "queryId" : "eadc6eea-698b-4a2f-8554-5270ab17ebee",
  "queryString" : "MATCH (n1)-[:knows]->(n2), (n2)-[:knows]->(n3), (n3)-[:knows]-
>(n4), (n4)-[:knows]->(n5), (n5)-[:knows]->(n6), (n6)-[:knows]->(n7), (n7)-[:knows]-
>(n8), (n8)-[:knows]->(n9), (n9)-[:knows]->(n10) RETURN COUNT(n1);",
  "queryEvalStats" : {
    "waited" : 0,
    "elapsed" : 23463,
    "cancelled" : false
  }
}
```

- クエリのキャンセルをリクエストします

1. POST を使用する:

```
curl -X POST https://server:port/openCypher/status \
  --data-urlencode "cancelQuery" \
  --data-urlencode "queryId=f43ce17b-db01-4d37-a074-c76d1c26d7a9"
```

レスポンス:

```
{
  "status" : "200 OK",
  "payload" : true
}
```

2. GET を使用する:

```
curl -X GET https://server:port/openCypher/status \
  --data-urlencode "cancelQuery" \
  --data-urlencode "queryId=588af350-cfde-4222-bee6-b9cedc87180d"
```

レスポンス:

```
{
  "status" : "200 OK",
  "payload" : true
}
```

3. DELETE を使用する:

```
curl -X DELETE \
  -s "https://server:port/openCypher/status?queryId=b9a516d1-d25c-4301-
bb80-10b2743ecf0e"
```

レスポンス:

```
{
  "status" : "200 OK",
  "payload" : true
}
```

Amazon Neptune openCypher HTTPS エンドポイント

トピック

- openCypherが HTTPS エンドポイントでクエリを読み書きする
- デフォルトの openCypher JSON 結果フォーマット

openCypherが HTTPS エンドポイントでクエリを読み書きする

openCypher HTTPS エンドポイントは、GET と POST メソッドの両方を使ってクエリを読み更新します。DELETE と PUT メソッドはサポートされていません。

次の手順は、curl コマンドおよび HTTPS を使用して openCypher エンドポイントに接続する方法について説明します。Neptune DB インスタンスと同じ仮想プライベートクラウド (VPC) の Amazon EC2 インスタンスからこれらの手順を実行してください。

構文は次のとおりです。

```
HTTPS://(the server):(the port number)/openCypher
```

以下に、読み取りクエリーのサンプルを示します。片方は POST を、もう一方は GET を使います。

1. POST を使用する:

```
curl HTTPS://server:port/openCypher \
  -d "query=MATCH (n1) RETURN n1;"
```

2. GET を使用する (クエリ文字列は URL に符号化されています)。

```
curl -X GET \
  "HTTPS://server:port/openCypher?query=MATCH%20(n1)%20RETURN%20n1"
```

以下に、書き込み/更新クエリのサンプルを示します。片方はPOST を、もう一方は GET を使います。

1. POST を使用する:

```
curl HTTPS://server:port/openCypher \
  -d "query=CREATE (n:Person { age: 25 })"
```

2. GET を使用する (クエリ文字列は URL に符号化されています)。

```
curl -X GET \
  "HTTPS://server:port/openCypher?query=CREATE%20(n%3APerson%20%7B%20age%3A
%2025%20%7D)"
```

デフォルトの openCypher JSON 結果フォーマット

次の JSON 形式がデフォルトで返されます。または、リクエストヘッダーを明示的に `Accept: application/json` に設定することで返されます。このフォーマットは、ほとんどのライブラリのネイティブ言語機能を使用してオブジェクトに簡単に解析できるように設計されています。

返される JSON ドキュメントには、1 つのフィールドが含まれています。`results` には、クエリの戻り値が含まれています。以下の例は、一般的な値の JSON フォーマットを示しています。

値のレスポンスの例:

```
{
```

```
  "results": [
    {
      "count(a)": 121
    }
  ]
}
```

ノードレスポンスの例:

```
{
  "results": [
    {
      "a": {
        "~id": "22",
        "~entityType": "node",
        "~labels": [
          "airport"
        ],
        "~properties": {
          "desc": "Seattle-Tacoma",
          "lon": -122.30899810791,
          "runways": 3,
          "type": "airport",
          "country": "US",
          "region": "US-WA",
          "lat": 47.4490013122559,
          "elev": 432,
          "city": "Seattle",
          "icao": "KSEA",
          "code": "SEA",
          "longest": 11901
        }
      }
    }
  ]
}
```

リレーションシップレスポンスの例:

```
{
  "results": [
    {
      "r": {
```

```
        "~id": "7389",
        "~entityType": "relationship",
        "~start": "22",
        "~end": "151",
        "~type": "route",
        "~properties": {
          "dist": 956
        }
      }
    }
  ]
}
```

パスレスポンスの例:

```
{
  "results": [
    {
      "p": [
        {
          "~id": "22",
          "~entityType": "node",
          "~labels": [
            "airport"
          ],
          "~properties": {
            "desc": "Seattle-Tacoma",
            "lon": -122.30899810791,
            "runways": 3,
            "type": "airport",
            "country": "US",
            "region": "US-WA",
            "lat": 47.4490013122559,
            "elev": 432,
            "city": "Seattle",
            "icao": "KSEA",
            "code": "SEA",
            "longest": 11901
          }
        },
        {
          "~id": "7389",
          "~entityType": "relationship",
```

```
          "~start": "22",
          "~end": "151",
          "~type": "route",
          "~properties": {
            "dist": 956
          }
        },
        {
          "~id": "151",
          "~entityType": "node",
          "~labels": [
            "airport"
          ],
          "~properties": {
            "desc": "Ontario International Airport",
            "lon": -117.600997924805,
            "runways": 2,
            "type": "airport",
            "country": "US",
            "region": "US-CA",
            "lat": 34.0559997558594,
            "elev": 944,
            "city": "Ontario",
            "icao": "KONT",
            "code": "ONT",
            "longest": 12198
          }
        }
      ]
    }
  ]
}
```

Bolt プロトコルを使用して openCypher クエリを Neptune に作成する

Bolt は、当初 Neo4j によって開発され、クリエイティブコモンズ 3.0 アトリビューションライセンスShareAlikeに基づいてライセンスされたステートメント指向のクライアント/サーバープロトコルです。これは、クライアント駆動型です。つまり、クライアントは常にメッセージ交換を開始します。

Neo4j の Bolt ドライバを使用して Neptune に接続するには、URL とポート番号を `bolt` URI スキームを使ってクラスターエンドポイントに置き換えるだけです。1 つの Neptune インスタンスが実行

されている場合は、read_write エンドポイントを使用します。複数のインスタンスが実行されている場合は、ライター用とすべてのリードレプリカ用に 2 つのドライバを推奨します。デフォルトの 2 つのエンドポイントしかない場合は、read_write と read_only ドライバで十分ですが、カスタムエンドポイントがある場合は、それぞれにドライバインスタンスを作成することを検討してください。

> **Note**
>
> Bolt 仕様では、Bolt は TCP または のいずれかを使用して接続できると記載されていますが WebSockets、Neptune は Bolt の TCP 接続のみをサポートしています。

Neptune では、最大 1000 の Bolt の同時接続が可能です。

Bolt ドライバを使用するさまざまな言語の openCypher クエリの例については、Neo4j ドライバ & 言語ガイドドキュメントを参照してください。

> **Important**
>
> Python、.NET、および Golang 用の Neo4j Bolt ドライバーは JavaScript、当初 Signature v4 AWS 認証トークンの自動更新をサポートしていませんでした。つまり、署名の有効期限が切れると (多くの場合 5 分後)、ドライバーは認証に失敗し、それ以降のリクエストも失敗しました。以下の Python、.NET JavaScript、Go の例はすべて、この問題の影響を受けています。
> 詳細については、Neo4j Python ドライバーの問題 #834」、Neo4j .NET の問題 #664」、Neo4j JavaScript ドライバーの問題 #993」、およびNeo4j goLang ドライバーの問題 #429」を参照してください。
> ドライバーバージョン 5.8.0 以降、Go ドライバー用の新しいプレビュー再認証 API がリリースされました (「v5.8.0 - 再認証に関するフィードバック募集」を参照)。

Bolt と Java を使用して Neptune に接続するには

Maven から使用する任意のバージョンのドライバをダウンロードできます。MVN リポジトリまたは、この依存関係をプロジェクトに追加できます。

```
<dependency>
  <groupId>org.neo4j.driver</groupId>
  <artifactId>neo4j-java-driver</artifactId>
  <version>4.3.3</version>
```

```
</dependency>
```

次に、これらの Bolt ドライバの 1 つを使用して Java で Neptune に接続するには、次のようなコードを使用して、クラスター内のプライマリ/ライターインスタンスのドライバインスタンスを作成します。

```
import org.neo4j.driver.Driver;
import org.neo4j.driver.GraphDatabase;

final Driver driver =
  GraphDatabase.driver("bolt://(your cluster endpoint URL):(your cluster port)",
    AuthTokens.none(),
    Config.builder().withEncryption()
                    .withTrustStrategy(TrustStrategy.trustSystemCertificates())
                    .build());
```

1 つ以上のリーダーレプリカがある場合は、同様に、次のようなコードを使用して、それらのドライバインスタンスを作成できます。

```
final Driver read_only_driver =              // (without connection timeout)
  GraphDatabase.driver("bolt://(your cluster endpoint URL):(your cluster port)",
      Config.builder().withEncryption()
                      .withTrustStrategy(TrustStrategy.trustSystemCertificates())
                      .build());
```

または、タイムアウトを設定して:

```
final Driver read_only_timeout_driver =      // (with connection timeout)
  GraphDatabase.driver("bolt://(your cluster endpoint URL):(your cluster port)",
    Config.builder().withConnectionTimeout(30, TimeUnit.SECONDS)
                    .withEncryption()
                    .withTrustStrategy(TrustStrategy.trustSystemCertificates())
                    .build());
```

カスタムエンドポイントがある場合は、各エンドポイントにドライバインスタンスを作成することをお勧めします。

Bolt を使用した Python openCypher クエリの例

以下は、Bolt を使用して Python で Python openCypher クエリを作成する方法です。

```
python -m pip install neo4j
```

```
from neo4j import GraphDatabase
uri = "bolt://(your cluster endpoint URL):(your cluster port)"
driver = GraphDatabase.driver(uri, auth=("username", "password"), encrypted=True)
```

auth パラメータは無視される点に注意してください。

Bolt を使用した .NET openCypher クエリの例

Bolt を使用して .NET で openCypher クエリを行う最初のステップは、 を使用して Neo4j ドライバーをインストールすることです NuHet。同期呼び出しを行うには、次のように .Simple バージョンを使用します。

```
Install-Package Neo4j.Driver.Simple-4.3.0
```

```
using Neo4j.Driver;

namespace hello
{
  // This example creates a node and reads a node in a Neptune
  // Cluster where IAM Authentication is not enabled.
  public class HelloWorldExample : IDisposable
  {
    private bool _disposed = false;
    private readonly IDriver _driver;
    private static string url = "bolt://(your cluster endpoint URL):(your cluster
 port)";
    private static string createNodeQuery = "CREATE (a:Greeting) SET a.message =
 'HelloWorldExample'";
    private static string readNodeQuery = "MATCH(n:Greeting) RETURN n.message";

    ~HelloWorldExample() => Dispose(false);

    public HelloWorldExample(string uri)
    {
      _driver = GraphDatabase.Driver(uri, AuthTokens.None, o =>
 o.WithEncryptionLevel(EncryptionLevel.Encrypted));
    }

    public void createNode()
```

```
    {
      // Open a session
      using (var session = _driver.Session())
      {
         // Run the query in a write transaction
        var greeting = session.WriteTransaction(tx =>
        {
          var result = tx.Run(createNodeQuery);
          // Consume the result
          return result.Consume();
        });

        // The output will look like this:
        //   ResultSummary{Query=`CREATE (a:Greeting) SET a.message =
'HelloWorldExample".....
        Console.WriteLine(greeting);
      }
    }

    public void retrieveNode()
    {
      // Open a session
      using (var session = _driver.Session())
      {
        // Run the query in a read transaction
        var greeting = session.ReadTransaction(tx =>
        {
          var result = tx.Run(readNodeQuery);
          // Consume the result. Read the single node
          // created in a previous step.
          return result.Single()[0].As<string>();
        });
        // The output will look like this:
        //   HelloWorldExample
        Console.WriteLine(greeting);
      }
    }

    public void Dispose()
    {
      Dispose(true);
      GC.SuppressFinalize(this);
    }
```

```
    protected virtual void Dispose(bool disposing)
    {
      if (_disposed)
        return;
      if (disposing)
      {
        _driver?.Dispose();
      }
      _disposed = true;
    }

    public static void Main()
    {
      using (var apiCaller = new HelloWorldExample(url))
      {
        apiCaller.createNode();
        apiCaller.retrieveNode();
      }
    }
  }
}
```

Bolt と IAM 認証を使用した Java openCypher クエリの例

以下の Java コードは、IAM 認証で Bolt を使用して Java で openCypher クエリを作成する方法を示しています。 JavaDoc コメントにはその使用法が説明されています。ドライバーインスタンスが使用可能になると、そのインスタンスを使用して複数の認証リクエストを行うことができます。

```
package software.amazon.neptune.bolt;

import com.amazonaws.DefaultRequest;
import com.amazonaws.Request;
import com.amazonaws.auth.AWS4Signer;
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.http.HttpMethodName;
import com.google.gson.Gson;
import lombok.Builder;
import lombok.Getter;
import lombok.NonNull;
import org.neo4j.driver.Value;
import org.neo4j.driver.Values;
import org.neo4j.driver.internal.security.InternalAuthToken;
import org.neo4j.driver.internal.value.StringValue;
```

```
import java.net.URI;
import java.util.Collections;
import java.util.HashMap;
import java.util.Map;

import static com.amazonaws.auth.internal.SignerConstants.AUTHORIZATION;
import static com.amazonaws.auth.internal.SignerConstants.HOST;
import static com.amazonaws.auth.internal.SignerConstants.X_AMZ_DATE;
import static com.amazonaws.auth.internal.SignerConstants.X_AMZ_SECURITY_TOKEN;

/**
 * Use this class instead of `AuthTokens.basic` when working with an IAM
 * auth-enabled server. It works the same as `AuthTokens.basic` when using
 * static credentials, and avoids making requests with an expired signature
 * when using temporary credentials. Internally, it generates a new signature
 * on every invocation (this may change in a future implementation).
 *
 * Note that authentication happens only the first time for a pooled connection.
 *
 * Typical usage:
 *
 * NeptuneAuthToken authToken = NeptuneAuthToken.builder()
 *     .credentialsProvider(credentialsProvider)
 *     .region("aws region")
 *     .url("cluster endpoint url")
 *     .build();
 *
 * Driver driver = GraphDatabase.driver(
 *     authToken.getUrl(),
 *     authToken,
 *     config
 * );
 */

public class NeptuneAuthToken extends InternalAuthToken {
  private static final String SCHEME = "basic";
  private static final String REALM = "realm";
  private static final String SERVICE_NAME = "neptune-db";
  private static final String HTTP_METHOD_HDR = "HttpMethod";
  private static final String DUMMY_USERNAME = "username";
  @NonNull
  private final String region;
  @NonNull
```

```
  @Getter
  private final String url;
  @NonNull
  private final AWSCredentialsProvider credentialsProvider;
  private final Gson gson = new Gson();

  @Builder
  private NeptuneAuthToken(
      @NonNull final String region,
      @NonNull final String url,
      @NonNull final AWSCredentialsProvider credentialsProvider
  ) {
      // The superclass caches the result of toMap(), which we don't want
      super(Collections.emptyMap());
      this.region = region;
      this.url = url;
      this.credentialsProvider = credentialsProvider;
  }

  @Override
  public Map<String, Value> toMap() {
    final Map<String, Value> map = new HashMap<>();
    map.put(SCHEME_KEY, Values.value(SCHEME));
    map.put(PRINCIPAL_KEY, Values.value(DUMMY_USERNAME));
    map.put(CREDENTIALS_KEY, new StringValue(getSignedHeader()));
    map.put(REALM_KEY, Values.value(REALM));

    return map;
  }

  private String getSignedHeader() {
    final Request<Void> request = new DefaultRequest<>(SERVICE_NAME);
    request.setHttpMethod(HttpMethodName.GET);
    request.setEndpoint(URI.create(url));
    // Comment out the following line if you're using an engine version older than
1.2.0.0
    request.setResourcePath("/opencypher");

    final AWS4Signer signer = new AWS4Signer();
    signer.setRegionName(region);
    signer.setServiceName(request.getServiceName());
    signer.sign(request, credentialsProvider.getCredentials());

    return getAuthInfoJson(request);
```

```
  }

  private String getAuthInfoJson(final Request<Void> request) {
    final Map<String, Object> obj = new HashMap<>();
    obj.put(AUTHORIZATION, request.getHeaders().get(AUTHORIZATION));
    obj.put(HTTP_METHOD_HDR, request.getHttpMethod());
    obj.put(X_AMZ_DATE, request.getHeaders().get(X_AMZ_DATE));
    obj.put(HOST, request.getHeaders().get(HOST));
    obj.put(X_AMZ_SECURITY_TOKEN, request.getHeaders().get(X_AMZ_SECURITY_TOKEN));

    return gson.toJson(obj);
  }
}
```

Bolt と IAM 認証を使用した Python openCypher クエリの例

以下の Python クラスを使用すると、IAM 認証付きの Bolt を使用して Python で openCypher クエリを作成できます。

```
import json

from neo4j import Auth
from botocore.awsrequest import AWSRequest
from botocore.credentials import Credentials
from botocore.auth import (
  SigV4Auth,
  _host_from_url,
)

SCHEME = "basic"
REALM = "realm"
SERVICE_NAME = "neptune-db"
DUMMY_USERNAME = "username"
HTTP_METHOD_HDR = "HttpMethod"
HTTP_METHOD = "GET"
AUTHORIZATION = "Authorization"
X_AMZ_DATE = "X-Amz-Date"
X_AMZ_SECURITY_TOKEN = "X-Amz-Security-Token"
HOST = "Host"


class NeptuneAuthToken(Auth):
  def __init__(
```

```
    self,
    credentials: Credentials,
    region: str,
    url: str,
    **parameters
  ):
    # Do NOT add "/opencypher" in the line below if you're using an engine version
 older than 1.2.0.0
    request = AWSRequest(method=HTTP_METHOD, url=url + "/opencypher")
    request.headers.add_header("Host", _host_from_url(request.url))
    sigv4 = SigV4Auth(credentials, SERVICE_NAME, region)
    sigv4.add_auth(request)

    auth_obj = {
      hdr: request.headers[hdr]
      for hdr in [AUTHORIZATION, X_AMZ_DATE, X_AMZ_SECURITY_TOKEN, HOST]
    }
    auth_obj[HTTP_METHOD_HDR] = request.method
    creds: str = json.dumps(auth_obj)
    super().__init__(SCHEME, DUMMY_USERNAME, creds, REALM, **parameters)
```

このクラスを使用してドライバーを次のように作成します。

```
  authToken = NeptuneAuthToken(creds, REGION, URL)
  driver = GraphDatabase.driver(URL, auth=authToken, encrypted=True)
```

IAM 認証と Bolt を使用した Node.js の例

以下の Node.js コードは、 JavaScript バージョン 3 および ES6 の AWS SDK 構文を使用して、リクエストを認証するドライバーを作成します。

```
import neo4j from "neo4j-driver";
import { HttpRequest }  from "@aws-sdk/protocol-http";
import { defaultProvider } from "@aws-sdk/credential-provider-node";
import { SignatureV4 } from "@aws-sdk/signature-v4";
import crypto from "@aws-crypto/sha256-js";
const { Sha256 } = crypto;
import assert from "node:assert";

const region = "us-west-2";
const serviceName = "neptune-db";
const host = "(your cluster endpoint URL)";
const port = 8182;
```

```
const protocol = "bolt";
const hostPort = host + ":" + port;
const url = protocol + "://" + hostPort;
const createQuery = "CREATE (n:Greeting {message: 'Hello'}) RETURN ID(n)";
const readQuery = "MATCH(n:Greeting) WHERE ID(n) = $id RETURN n.message";

async function signedHeader() {
  const req = new HttpRequest({
    method: "GET",
    protocol: protocol,
    hostname: host,
    port: port,
    // Comment out the following line if you're using an engine version older than
 1.2.0.0
    path: "/opencypher",
    headers: {
      host: hostPort
    }
  });

  const signer = new SignatureV4({
    credentials: defaultProvider(),
    region: region,
    service: serviceName,
    sha256: Sha256
  });

  return signer.sign(req, { unsignableHeaders: new Set(["x-amz-content-sha256"]) })
    .then((signedRequest) => {
      const authInfo = {
        "Authorization": signedRequest.headers["authorization"],
        "HttpMethod": signedRequest.method,
        "X-Amz-Date": signedRequest.headers["x-amz-date"],
        "Host": signedRequest.headers["host"],
        "X-Amz-Security-Token": signedRequest.headers["x-amz-security-token"]
      };
      return JSON.stringify(authInfo);
    });
}

async function createDriver() {
  let authToken = { scheme: "basic", realm: "realm", principal: "username",
 credentials: await signedHeader() };
```

```
  return neo4j.driver(url, authToken, {
      encrypted: "ENCRYPTION_ON",
      trust: "TRUST_SYSTEM_CA_SIGNED_CERTIFICATES",
      maxConnectionPoolSize: 1,
      // logging: neo4j.logging.console("debug")
    }
  );
}

function unmanagedTxn(driver) {
  const session = driver.session();
  const tx = session.beginTransaction();
  tx.run(createQuery)
  .then((res) => {
    const id = res.records[0].get(0);
    return tx.run(readQuery, { id: id });
  })
  .then((res) => {
    // All good, the transaction will be committed
    const msg = res.records[0].get("n.message");
    assert.equal(msg, "Hello");
  })
  .catch(err => {
    // The transaction will be rolled back, now handle the error.
    console.log(err);
  })
  .then(() => session.close());
}

createDriver()
.then((driver) => {
  unmanagedTxn(driver);
  driver.close();
})
.catch((err) => {
  console.log(err);
});
```

Bolt と IAM 認証を使用した .NET openCypher クエリの例

.NET で IAM 認証を有効にするには、接続を確立するときにリクエストに署名する必要があります。以下の例は、認証トークンを生成する NeptuneAuthToken ヘルパーを作成する方法を示しています。

```
using Amazon.Runtime;
using Amazon.Util;
using Neo4j.Driver;
using System.Security.Cryptography;
using System.Text;
using System.Text.Json;
using System.Web;

namespace Hello
{
  /*
   * Use this class instead of `AuthTokens.None` when working with an IAM-auth-enabled
 server.
   *
   * Note that authentication happens only the first time for a pooled connection.
   *
   * Typical usage:
   *
   * var authToken = new NeptuneAuthToken(AccessKey, SecretKey,
 Region).GetAuthToken(Host);
   * _driver = GraphDatabase.Driver(Url, authToken, o =>
 o.WithEncryptionLevel(EncryptionLevel.Encrypted));
   */

  public class NeptuneAuthToken
  {
    private const string ServiceName = "neptune-db";
    private const string Scheme = "basic";
    private const string Realm = "realm";
    private const string DummyUserName = "username";
    private const string Algorithm = "AWS4-HMAC-SHA256";
    private const string AWSRequest = "aws4_request";

    private readonly string _accessKey;
    private readonly string _secretKey;
    private readonly string _region;

    private readonly string _emptyPayloadHash;

    private readonly SHA256 _sha256;
```

```
    public NeptuneAuthToken(string awsKey = null, string secretKey = null, string
region = null)
    {
      var awsCredentials = awsKey == null || secretKey == null
        ? FallbackCredentialsFactory.GetCredentials().GetCredentials()
        : null;

      _accessKey = awsKey ?? awsCredentials.AccessKey;
      _secretKey = secretKey ?? awsCredentials.SecretKey;
      _region = region ?? FallbackRegionFactory.GetRegionEndpoint().SystemName; //ex:
us-east-1

      _sha256 = SHA256.Create();
      _emptyPayloadHash = Hash(Array.Empty<byte>());
    }

    public IAuthToken GetAuthToken(string url)
    {
      return AuthTokens.Custom(DummyUserName, GetCredentials(url), Realm, Scheme);
    }

    /******************* AWS SIGNING FUNCTIONS ********************/
    private string Hash(byte[] bytesToHash)
    {
      return ToHexString(_sha256.ComputeHash(bytesToHash));
    }

    private static byte[] HmacSHA256(byte[] key, string data)
    {
      return new HMACSHA256(key).ComputeHash(Encoding.UTF8.GetBytes(data));
    }

    private byte[] GetSignatureKey(string dateStamp)
    {
      var kSecret = Encoding.UTF8.GetBytes($"AWS4{_secretKey}");
      var kDate = HmacSHA256(kSecret, dateStamp);
      var kRegion = HmacSHA256(kDate, _region);
      var kService = HmacSHA256(kRegion, ServiceName);
      return HmacSHA256(kService, AWSRequest);
    }

    private static string ToHexString(byte[] array)
    {
      return Convert.ToHexString(array).ToLowerInvariant();
```

```
    }

    private string GetCredentials(string url)
    {
      var request = new HttpRequestMessage
      {
        Method = HttpMethod.Get,
        RequestUri = new Uri($"https://{url}/opencypher")
      };

      var signedrequest = Sign(request);

      var headers = new Dictionary<string, object>
      {
        [HeaderKeys.AuthorizationHeader] =
signedrequest.Headers.GetValues(HeaderKeys.AuthorizationHeader).FirstOrDefault(),
        ["HttpMethod"] = HttpMethod.Get.ToString(),
        [HeaderKeys.XAmzDateHeader] =
signedrequest.Headers.GetValues(HeaderKeys.XAmzDateHeader).FirstOrDefault(),
        // Host should be capitalized, not like in Amazon.Util.HeaderKeys.HostHeader
        ["Host"] =
signedrequest.Headers.GetValues(HeaderKeys.HostHeader).FirstOrDefault(),
      };

      return JsonSerializer.Serialize(headers);
    }

    private HttpRequestMessage Sign(HttpRequestMessage request)
    {
      var now = DateTimeOffset.UtcNow;
      var amzdate = now.ToString("yyyyMMddTHHmmssZ");
      var datestamp = now.ToString("yyyyMMdd");

      if (request.Headers.Host == null)
      {
        request.Headers.Host = $"{request.RequestUri.Host}:{request.RequestUri.Port}";
      }

      request.Headers.Add(HeaderKeys.XAmzDateHeader, amzdate);

      var canonicalQueryParams = GetCanonicalQueryParams(request);

      var canonicalRequest = new StringBuilder();
      canonicalRequest.Append(request.Method + "\n");
```

```
      canonicalRequest.Append(request.RequestUri.AbsolutePath + "\n");
      canonicalRequest.Append(canonicalQueryParams + "\n");

      var signedHeadersList = new List<string>();
      foreach (var header in request.Headers.OrderBy(a => a.Key.ToLowerInvariant()))
      {
        canonicalRequest.Append(header.Key.ToLowerInvariant());
        canonicalRequest.Append(':');
        canonicalRequest.Append(string.Join(",", header.Value.Select(s => s.Trim())));
        canonicalRequest.Append('\n');
        signedHeadersList.Add(header.Key.ToLowerInvariant());
      }
      canonicalRequest.Append('\n');

      var signedHeaders = string.Join(";", signedHeadersList);
      canonicalRequest.Append(signedHeaders + "\n");
      canonicalRequest.Append(_emptyPayloadHash);

      var credentialScope = $"{datestamp}/{_region}/{ServiceName}/{AWSRequest}";
      var stringToSign = $"{Algorithm}\n{amzdate}\n{credentialScope}\n"
        + Hash(Encoding.UTF8.GetBytes(canonicalRequest.ToString()));

      var signing_key = GetSignatureKey(datestamp);
      var signature = ToHexString(HmacSHA256(signing_key, stringToSign));

      request.Headers.TryAddWithoutValidation(HeaderKeys.AuthorizationHeader,
        $"{Algorithm} Credential={_accessKey}/{credentialScope},
 SignedHeaders={signedHeaders}, Signature={signature}");

      return request;
    }

    private static string GetCanonicalQueryParams(HttpRequestMessage request)
    {
      var querystring = HttpUtility.ParseQueryString(request.RequestUri.Query);

      // Query params must be escaped in upper case (i.e. "%2C", not "%2c").
      var queryParams = querystring.AllKeys.OrderBy(a => a)
        .Select(key => $"{key}={Uri.EscapeDataString(querystring[key])}");
      return string.Join("&", queryParams);
    }
  }
}
```

以下は、.NET で Bolt と IAM 認証を使用して openCypher クエリを作成する方法です。次の例では、NeptuneAuthToken ヘルパーを使用しています。

```
using Neo4j.Driver;

namespace Hello
{
  public class HelloWorldExample
  {
    private const string Host = "(your hostname):8182";
    private const string Url = $"bolt://{Host}";
    private const string CreateNodeQuery = "CREATE (a:Greeting) SET a.message =
 'HelloWorldExample'";
    private const string ReadNodeQuery = "MATCH(n:Greeting) RETURN n.message";

    private const string AccessKey = "(your access key)";
    private const string SecretKey = "(your secret key)";
    private const string Region = "(your AWS region)"; // e.g. "us-west-2"

    private readonly IDriver _driver;

    public HelloWorldExample()
    {
      var authToken = new NeptuneAuthToken(AccessKey, SecretKey,
 Region).GetAuthToken(Host);

      // Note that when the connection is reinitialized after max connection lifetime
      // has been reached, the signature token could have already been expired (usually
 5 min)
      // You can face exceptions like:
      //   `Unexpected server exception 'Signature expired: XXXX is now earlier than
 YYYY (ZZZZ - 5 min.)`
      _driver = GraphDatabase.Driver(Url, authToken, o =>

 o.WithMaxConnectionLifetime(TimeSpan.FromMinutes(60)).WithEncryptionLevel(EncryptionLevel.Encr
    }

    public async Task CreateNode()
    {
      // Open a session
      using (var session = _driver.AsyncSession())
      {
        // Run the query in a write transaction
```

```
        var greeting = await session.WriteTransactionAsync(async tx =>
        {
          var result = await tx.RunAsync(CreateNodeQuery);
          // Consume the result
          return await result.ConsumeAsync();
        });

        // The output will look like this:
        //   ResultSummary{Query=`CREATE (a:Greeting) SET a.message =
 'HelloWorldExample".....
        Console.WriteLine(greeting.Query);
      }
    }

    public async Task RetrieveNode()
    {
      // Open a session
      using (var session = _driver.AsyncSession())
      {
        // Run the query in a read transaction
        var greeting = await session.ReadTransactionAsync(async tx =>
        {
          var result = await tx.RunAsync(ReadNodeQuery);
          var records = await result.ToListAsync();

          // Consume the result. Read the single node
          // created in a previous step.
          return records[0].Values.First().Value;
        });
        // The output will look like this:
        //   HelloWorldExample
        Console.WriteLine(greeting);
      }
    }
  }
}
```

この例は、以下のコードを以下のパッケージで .NET 6 または .NET 7 に対して実行することで起動できます。

- **Neo4j**.Driver=4.3.0
- **AWSSDK**.Core=3.7.102.1

```
namespace Hello
{
  class Program
  {
    static async Task Main()
    {
      var apiCaller = new HelloWorldExample();

      await apiCaller.CreateNode();
      await apiCaller.RetrieveNode();
    }
  }
}
```

Bolt と IAM 認証を使用した Golang openCypher クエリの例

以下の Golang パッケージは、Bolt と IAM 認証を使用して Go 言語で openCypher クエリを作成する方法を示しています。

```
package main

import (
  "context"
  "encoding/json"
  "fmt"
  "github.com/aws/aws-sdk-go/aws/credentials"
  "github.com/aws/aws-sdk-go/aws/signer/v4"
  "github.com/neo4j/neo4j-go-driver/v5/neo4j"
  "log"
  "net/http"
  "os"
  "time"
)

const (
  ServiceName   = "neptune-db"
  DummyUsername = "username"
)

// Find node by id using Go driver
func findNode(ctx context.Context, region string, hostAndPort string, nodeId string)
 (string, error) {
```

```
 req, err := http.NewRequest(http.MethodGet, "https://"+hostAndPort+"/opencypher",
nil)

 if err != nil {
   return "", fmt.Errorf("error creating request, %v", err)
 }

 // credentials must have been exported as environment variables
 signer := v4.NewSigner(credentials.NewEnvCredentials())
 _, err = signer.Sign(req, nil, ServiceName, region, time.Now())

 if err != nil {
   return "", fmt.Errorf("error signing request: %v", err)
 }

 hdrs := []string{"Authorization", "X-Amz-Date", "X-Amz-Security-Token"}
 hdrMap := make(map[string]string)
 for _, h := range hdrs {
   hdrMap[h] = req.Header.Get(h)
 }

 hdrMap["Host"] = req.Host
 hdrMap["HttpMethod"] = req.Method

 password, err := json.Marshal(hdrMap)
 if err != nil {
   return "", fmt.Errorf("error creating JSON, %v", err)
 }
 authToken := neo4j.BasicAuth(DummyUsername, string(password), "")
 // +s enables encryption with a full certificate check
 // Use +ssc to disable client side TLS verification
 driver, err := neo4j.NewDriverWithContext("bolt+s://"+hostAndPort+"/opencypher",
authToken)
 if err != nil {
   return "", fmt.Errorf("error creating driver, %v", err)
 }

 defer driver.Close(ctx)

 if err := driver.VerifyConnectivity(ctx); err != nil {
   log.Fatalf("failed to verify connection, %v", err)
 }

 config := neo4j.SessionConfig{}
```

```
  session := driver.NewSession(ctx, config)
  defer session.Close(ctx)

  result, err := session.Run(
    ctx,
    fmt.Sprintf("MATCH (n) WHERE ID(n) = '%s' RETURN n", nodeId),
    map[string]any{},
  )
  if err != nil {
    return "", fmt.Errorf("error running query, %v", err)
  }

  if !result.Next(ctx) {
    return "", fmt.Errorf("node not found")
  }

  n, found := result.Record().Get("n")
  if !found {
    return "", fmt.Errorf("node not found")
  }

  return fmt.Sprintf("+%v\n", n), nil
}

func main() {
  if len(os.Args) < 3 {
    log.Fatal("Usage: go main.go (region) (host and port)")
  }
  region := os.Args[1]
  hostAndPort := os.Args[2]
  ctx := context.Background()

  res, err := findNode(ctx, region, hostAndPort,
 "72c2e8c1-7d5f-5f30-10ca-9d2bb8c4afbc")
  if err != nil {
    log.Fatal(err)
  }
  fmt.Println(res)
}
```

Neptune での Bolt 接続動作

Neptune Bolt 接続に関する注意事項をいくつか次に示します。

- Bolt 接続は TCP レイヤーで作成されるため、HTTP エンドポイントでできるように、それらの前ではApplication Load Balancerは使えません。
- Neptune が Bolt 接続に使用するポートは、DB クラスターのポートです。
- 渡された Bolt プリアンブルに基づいて、Neptune サーバーは最適な Bolt バージョン (1、2、3、または 4.0) を選択します。
- クライアントが任意の時点でオープンできる Neptune サーバーへの最大接続数は 1,000 です。
- クエリの後にクライアントが接続を閉じない場合、その結合を使用して次のクエリを実行できます。
- ただし、接続が 20 分間アイドル状態になると、サーバーは自動的に閉じます。
- IAM 認証が有効になっていない場合は、ダミーのユーザー名とパスワードを入力する代わりに `AuthTokens.none()` を使用できます。例えば、Java では次のようになります。

```
GraphDatabase.driver("bolt://(your cluster endpoint URL):(your cluster port)",
 AuthTokens.none(),

 Config.builder().withEncryption().withTrustStrategy(TrustStrategy.trustSystemCertificates())
```

- IAM 認証が有効になっている場合、Bolt 接続は確立されてから 10 日経過すると、他の理由で Bolt 接続がまだ閉じられていない場合、常に数分で切断されます。
- クライアントが前のクエリの結果を消費せずに、接続を介して実行用のクエリを送信すると、新しいクエリは破棄されます。代わりに以前の結果を破棄するには、クライアントは接続を介してリセットメッセージを送信する必要があります。
- 特定の接続では、一度に作成できるトランザクションは 1 つだけです。
- トランザクション中に例外が発生すると、Neptune サーバーはトランザクションをロールバックし、接続を閉じます。この場合、ドライバーは次のクエリ用に新しい接続を作成します。
- セッションはスレッドセーフではないことに注意してください。複数の並列操作では、複数の別々のセッションを使用する必要があります。

openCypher のパラメーター化されたクエリの例

Neptune はパラメーター化された openCypher クエリをサポートしています。これにより、同じクエリ構造を異なる引数で複数回使用できます。クエリ構造は変更されないため、Neptune は抽象構文ツリー (AST) を何度も解析しなくてもキャッシュできます。

HTTPS エンドポイントを使用する openCypher パラメーター化されたクエリの例

以下は、Neptune openCypher HTTPS エンドポイントでパラメーター化されたクエリを使用する例です。クエリ は、次のとおりです。

```
MATCH (n {name: $name, age: $age})
RETURN n
```

パラメータの定義は次のとおりです。

```
parameters={"name": "john", "age": 20}
```

GET を使用すると、次のようなパラメータ化されたクエリを送信できます。

```
curl -k \
  "https://localhost:8182/openCypher?query=MATCH%20%28n%20%7Bname:\$name,age:\$age%7D
%29%20RETURN%20n&parameters=%7B%22name%22:%22john%22,%22age%22:20%7D"
```

または、POST を使用することもできます。

```
curl -k \
  https://localhost:8182/openCypher \
  -d "query=MATCH (n {name: \$name, age: \$age}) RETURN n" \
  -d "parameters={\"name\": \"john\", \"age\": 20}"
```

または、DIRECT POST を使用します。

```
curl -k \
   -H "Content-Type: application/opencypher" \
  "https://localhost:8182/openCypher?parameters=%7B%22name%22:%22john%22,%22age
%22:20%7D" \
  -d "MATCH (n {name: \$name, age: \$age}) RETURN n"
```

Bolt を使った openCypher のパラメーター化されたクエリの例

以下は、Bolt プロトコルを使用した openCypher パラメーター化されたクエリの Python の例です。

```
from neo4j import GraphDatabase
uri = "bolt://[neptune-endpoint-url]:8182"
driver = GraphDatabase.driver(uri, auth=("", ""))

def match_name_and_age(tx, name, age):
  # Parameterized Query
  tx.run("MATCH (n {name: $name, age: $age}) RETURN n", name=name, age=age)

with driver.session() as session:
  # Parameters
  session.read_transaction(match_name_and_age, "john", 20)

driver.close()
```

以下は、Bolt プロトコルを使用する openCypher パラメーター化されたクエリの Java の例です。

```
Driver driver = GraphDatabase.driver("bolt+s://(your cluster endpoint URL):8182");
HashMap<String, Object> parameters = new HashMap<>();
parameters.put("name", "john");
parameters.put("age", 20);
String queryString = "MATCH (n {name: $name, age: $age}) RETURN n";
Result result = driver.session().run(queryString, parameters);
```

openCypher データモデル

Neptune openCypher エンジンは Gremlin と同じプロパティグラフモデルに基づいて構築されます。特に、次のことに注意してください。

- すべてのノードには 1 つ以上のラベルがあります。ラベルなしでノードを挿入すると、`vertex` という名前のデフォルトラベルが付きます。ノードのラベルをすべて削除しようとすると、エラーがスローされます。
- リレーションシップとは、1 つのリレーションシップタイプを持ち、2 つのノード間に単方向接続を形成するエンティティです (つまり、ノードの 1 つからもう 1 つに)。
- ノードとリレーションシップは両方ともプロパティを持つことができますが、そうする必要はありません。Neptune は、ゼロのプロパティを持つノードとリレーションシップをサポートしています。

- Neptune はメタプロパティをサポートしていません。メタプロパティも openCypher 仕様に含まれていません。
- Grumlin を使用して作成された場合は、グラフ内のプロパティを複数値にすることができます。つまり、ノードまたはリレーションシッププロパティには、1 つの値だけでなく、さまざまな値のセットを設定できます。Neptune は openCypher セマンティクスを拡張して、複数値のプロパティを正常に処理します。

サポートされているデータ型については、openCypher データ形式 を参照してください。ただし、現在、`Array` プロパティ値を openCypher グラフに挿入することはお勧めしていません。同様に、バルクローダーを使用して配列プロパティ値を挿入できます。現在の Neptune openCypher リリースでは、単一のリスト値ではなく、複数の値を持つプロパティのセットとして扱われます。

このリリースでサポートされるデータ型のリストを次に示します。

- `Bool`
- `Byte`
- `Short`
- `Int`
- `Long`
- `Float` (プラスマイナスインフィニティと NaN を含みますが、INF は含まれません)
- `Double` (プラスマイナスインフィニティと NaN を含みますが、INF は含まれません)
- `DateTime`
- `String`

openCypher **`explain`** 機能

openCypher の `explain` 機能は、Neptune エンジンが取る実行アプローチの理解に役立つ Amazon Neptune のセルフサービスツールです。explain を呼び出すには、openCypher HTTPS リクエストに `explain=`*`mode`* でパラメータを渡します。*`mode`* の値は次のいずれかになります。

- **`static`** - `static` モードでは、`explain` はクエリプランの静的構造のみを表示します。実際にはクエリを実行しません。

- **dynamic** — dynamic モードでは、explain はクエリも実行し、クエリプランの動的な要素も含まれます。これらには、演算子を介して通過する中間バインドの数、送信バインドに対する着信バインドの割合、各演算子の所要合計時間が含まれる場合があります。
- **details** - details モードでは、explain は動的モードで表示される情報に加えて、実際の openCypher クエリ文字列や、結合演算子が基づくパターンの推定範囲数などの詳細を出力します。

例えば、POST を使用します。

```
curl HTTPS://server:port/openCypher \
  -d "query=MATCH (n) RETURN n LIMIT 1;" \
  -d "explain=dynamic"
```

または、GET を使用します。

```
curl -X GET \
  "HTTPS://server:port/openCypher?query=MATCH%20(n)%20RETURN%20n%20LIMIT
%201&explain=dynamic"
```

Neptune における openCypher **explain** の制限事項

openCypher explain の現在のリリースには、次のような制約事項があります。

- explain プランは、現在、読み取り専用操作を実行するクエリでのみ使用できます。CREATE、DELETE、MERGE、SET など、何らかのミューテーションを行うクエリはサポートされていません。
- 特定のプランの演算子と出力は、将来のリリースで変更される可能性があります。

openCypher **explain** 出力に含まれる DFE 演算子

openCypher explain 機能が提供する情報を使用するには、DFE クエリエンジンの仕組みについて、いくつかの詳細を理解する必要があります (DFE は Neptune が openCypher クエリの処理に使用するエンジン)。

DFE エンジンは、すべてのクエリを演算子のパイプラインに変換します。最初の演算子から始まり、中間ソリューションは、1 つの演算子から次の演算子へと、この演算子パイプラインを流れます。explain テーブルの各行は、評価ポイントまでの結果を表します。

DFE クエリプランで使用できる演算子は次のとおりです。

DFEApply — 指定した変数に保存されている値に対して、引数セクションで指定した関数を実行します。

DFEBindRelation – 指定された名前で変数をバインドします

DFEChunkLocalSubQuery – これは、実行されるサブクエリのラッパーとして機能するノンブロッキングオペレーションです。

DFEDistinctColumn – 指定された変数に基づいて、入力値の個別のサブセットを返します。

DFEDistinctRelation – 指定された変数に基づいて、入力ソリューションの個別のサブセットを返します。

DFEDrain — サブクエリの最後に表示され、そのサブクエリの終了ステップとして機能します。ソリューションの数は `Units In` として記録されます。`Units Out` は常に 0 です。

DFEForwardValue – すべての入力チャンクを出力チャンクとして直接コピーし、ダウンストリーム演算子に渡します。

DFEGroupByHashIndex – 以前に計算されたハッシュインデックスに基づいて、入力ソリューションに対してグループごとのオペレーションを実行します (`DFEHashIndexBuild`オペレーションを使用) 。出力として、指定された入力は、すべての入力ソリューションのグループキーを含む列によって拡張されます。

DFE HashIndexビルド — 一連の変数にハッシュインデックスを副作用として構築します。通常、このハッシュインデックスは後の操作で再利用されます。このハッシュインデックスの使用場所については、`DFEHashIndexJoin` または `DFEGroupByHashIndex` を参照してください。

DFE HashIndexJoin – 以前に構築されたハッシュインデックスに対して受信ソリューションの結合を実行します。このハッシュインデックスの構築場所については、`DFEHashIndexBuild` を参照してください。

DFEJoinExists – 左右の入力リレーションを取得し、指定された結合変数で定義された右側のリレーションに対応する値を持つ左側のリレーションの値を保持します。

— これはサブクエリのラッパーとして機能するノンブロッキング操作であり、繰り返し実行してループで使用できます。

DFEMergeChunks — これは、アップストリーム演算子のチャンクをソリューションの 1 つのチャンクに結合してダウンストリーム演算子に渡すブロッキングオペレーションです (の逆DFESplitChunks) 。

DFEMinus — 左側と右側の入力リレーションを取得し、指定された結合変数で定義されている右側のリレーションの値に対応しない左側のリレーションの値を保持します。両方のリレーションで変数に重複がない場合、この演算子は単に左側の入力リレーションを返します。

DFENotExists – 左右の入力リレーションを取得し、指定された結合変数で定義されているように、右側のリレーションに対応する値を持たない左側のリレーションの値を保持します。両方のリレーションで変数に重複がない場合、この演算子は空のリレーションを返します。

DFEOptionalJoin – 左外部結合 (オプション結合とも呼ばれます) を実行します。右側に少なくとも 1 つの結合パートナーがある左側からのソリューションは結合され、右側に結合パートナーがない左側からのソリューションはそのまま転送されます。これはブロッキング操作です。

DFEPipelineJoin – pattern引数で定義されたタプルパターンに対して入力を結合します。

DFE PipelineRangeCount – 特定のパターンに一致するソリューションの数をカウントし、カウント値を含む単一の 1 項ソリューションを返します。

DFEPipelineScan – 列に特定のフィルターの有無にかかわらず、指定されたpattern引数についてデータベースをスキャンします (複数可) 。

DFEProject — 複数の入力列を受け取り、必要な列のみを投影します。

DFEReduce — 指定された変数に対して指定された集計関数を実行します。

DFERelationalJoin – マージ結合を使用して、指定されたパターンキーに基づいて前の演算子の入力を結合します。これはブロッキング操作です。

DFERouteChunks – 単一の受信エッジから入力チャンクを取得し、それらのチャンクを複数の送信エッジに沿ってルーティングします。

DFESelectRows – この演算子は、左側の入力リレーションソリューションから行を選択してダウンストリーム演算子に転送します。行は、演算子の右側の入力リレーションで指定された行識別子に基づいて選択されます。

DFESerialize — クエリの最終結果を JSON 文字列としてシリアル化し、各入力ソリューションを適切な変数名にマッピングします。ノードとエッジの結果の場合、これらの結果はエンティティプロパティとメタデータのマップにシリアル化されます。

DFESort — 入力リレーションを受け取り、指定されたソートキーに基づいてソートされたリレーションを生成します。

DFE SplitByグループ – 1 つの入力エッジから各 1 つの入力チャンクを、他の入力エッジから対応する入力チャンクの行 IDs で識別される行グループに対応する小さな出力チャンクに分割します。

DFESplitChunks – 各単一の入力チャンクを小さな出力チャンクに分割します (の逆`DFEMergeChunks`)。

DFEStreamingHashIndexBuild – のストリーミングバージョン`DFEHashIndexBuild`。

DFE StreamingGroupByHashインデックス — のストリーミングバージョン`DFEGroupByHashIndex`。

DFESubquery — この演算子はすべてのプランの最初に表示され、openCypher のプラン全体である DFE エンジン上で実行されるプランの部分をカプセル化します。

DFE SymmetricHashJoin – ハッシュ結合を使用して、指定されたパターンキーに基づいて前の演算子の入力を結合します。これは非ブロッキング操作です。

DFESync — この演算子は、ノンブロッキングプランをサポートする同期演算子です。2 つの受信エッジからソリューションを受け取り、これらのソリューションを適切なダウンストリームエッジに転送します。同期の目的で、これらのエッジの 1 つに沿った入力は内部でバッファリングされる場合があります。

DFETee — これは、同じソリューションセットを複数の演算子に送信する分岐演算子です。

DFETermResolution – 入力に対してローカライズまたはグローバル化オペレーションを実行し、それぞれローカライズされた識別子またはグローバル化された識別子の列を作成します。

— 入力列の値のリストを個別の要素として出力列に展開します。

DFEunion — 2 つ以上の入力リレーションを受け取り、目的の出力スキーマを使用して、これらのリレーションの和集合を生成します。

SolutionInjection – 説明出力の他のすべての項目の前に表示され、単位出力列の値は 1 です。ただし、これはノーオペレーションとして機能し、実際には DFE エンジンにソリューションは挿入されません。

TermResolution – 計画の最後に表示され、Neptune エンジンから openCypher オブジェクトにオブジェクトを変換します。

openCypher **explain** 出力の列

Neptune が openCypher explain 出力として生成するクエリプラン情報には、1 行に 1 つの演算子を含むテーブルが含まれています。このテーブルには、次の列があります。

ID — プラン内のこの演算子の数値 ID。

Out #1 (および Out #2) — この演算子の下流にいる演算子の ID。ダウンストリーム演算子は最大で 2 つまで存在できます。

名前 — この演算子の名前。

引数 — 演算子に関連する詳細。これには、入力スキーマ、出力スキーマ、パターン (`PipelineScan` と`PipelineJoin`) などが含まれます。

モード — 基本的な演算子の動作を説明するラベル。この列はほとんど空白 (-) です。唯一の例外は `TermResolution` であり、この場合、モードは `id2value_opencypher` になり、ID から openCypher 値への解決を示します。

Units In — この演算子への入力として渡されるソリューションの数。`DFEPipelineScan`、`SolutionInjections`、`DFESubquery` など、静的な値が注入されていない上流演算子のない演算子は、ゼロの値を持ちます。

Units Out — この演算子の出力として生成されるソリューションの数。`DFEDrain` は特殊なケースであり、排出されるソリューションの数が `Units In` に記録され、`Units Out` は常にゼロになります。

比率 — `Units In` に対する `Units Out` の比率。

時間 (ms) — この演算子が消費した CPU 時間 (ミリ秒単位)。

openCypher explain 出力の基本的な例

openCypher `explain` 出力の基本的な例を次に示します。このクエリは、デフォルトの ASCII 出力形式の `details` モードを使用して `explain` を起動する空港コード ATL を含むノードを航路データセット内で単一ノードで検索します。

```
curl -d "query=MATCH (n {code: 'ATL'}) RETURN n" -k https://localhost:8182/openCypher -
d "explain=details"
                                    ~
Query:
MATCH (n {code: 'ATL'}) RETURN n
```

```
################################################################################################
# ID # Out #1 # Out #2 # Name              # Arguments          # Mode                #
 Units In # Units Out # Ratio # Time (ms) #
################################################################################################
# 0  # 1      # -      # SolutionInjection # solutions=[{}]     # -                   #
 0        # 1         # 0.00  # 0         #
################################################################################################
# 1  # 2      # -      # DFESubquery       # subQuery=subQuery1 # -                   #
 0        # 1         # 0.00  # 4.00      #
################################################################################################
# 2  # -      # -      # TermResolution    # vars=[?n]          # id2value_opencypher #
 1        # 1         # 1.00  # 2.00      #
################################################################################################



subQuery1
################################################################################################
# ID # Out #1 # Out #2 # Name                  # Arguments
                                                                  # Mode # Units
 In # Units Out # Ratio # Time (ms) #
################################################################################################
# 0  # 1      # -      # DFEPipelineScan       # pattern=Node(?n) with property 'code'
 as ?n_code2 and label 'ALL'                                       # -    # 0
   # 1         # 0.00  # 0.21      #
#    #        #        #                       # inlineFilters=[(?n_code2 IN
 ["ATL"^^xsd:string])]                                                  #
  #          #           #       #           #
#    #        #        #                       # patternEstimate=1
                                                                 #      #
  #           #       #           #
################################################################################################
# 1  # 2      # -      # DFEChunkLocalSubQuery # subQuery=http://aws.amazon.com/
neptune/vocab/v01/dfe/past/graph#9d84f97c-c3b0-459a-98d5-955a8726b159/graph_1 # -    #
 1        # 1         # 1.00  # 0.04      #
################################################################################################
# 2  # 3      # -      # DFEProject           # columns=[?n]
                                                                 # -    # 1
  # 1         # 1.00  # 0.04      #
################################################################################################
# 3  # -      # -      # DFEDrain             # -
                                                                 # -    # 1
  # 0         # 0.00  # 0.03      #
################################################################################################
```

```
subQuery=http://aws.amazon.com/neptune/vocab/v01/dfe/past/graph#9d84f97c-
c3b0-459a-98d5-955a8726b159/graph_1
#########################################################################################
# ID # Out #1 # Out #2 # Name                  # Arguments
                  # Mode # Units In # Units Out # Ratio # Time (ms) #
#########################################################################################
# 0  # 1      # -      # DFESolutionInjection # outSchema=[?n, ?n_code2]
                  # -    # 0        # 1         # 0.00  # 0.02      #
#########################################################################################
# 1  # 2      # 3      # DFETee               # -
                  # -    # 1        # 2         # 2.00  # 0.02      #
#########################################################################################
# 2  # 4      # -      # DFEDistinctColumn    # column=?n
                  # -    # 1        # 1         # 1.00  # 0.20      #
#    #        #        #                      # ordered=false
                  #      #          #         #       #           #
#########################################################################################
# 3  # 5      # -      # DFEHashIndexBuild    # vars=[?n]
                  # -    # 1        # 1         # 1.00  # 0.04      #
#########################################################################################
# 4  # 5      # -      # DFEPipelineJoin      # pattern=Node(?n) with property 'ALL'
 and label '?n_label1' # -    # 1        # 1         # 1.00  # 0.25      #
#    #        #        #                      # patternEstimate=3506
                  #      #          #         #       #           #
#########################################################################################
# 5  # 6      # 7      # DFESync              # -
                  # -    # 2        # 2         # 1.00  # 0.02      #
#########################################################################################
# 6  # 8      # -      # DFEForwardValue      # -
                  # -    # 1        # 1         # 1.00  # 0.01      #
#########################################################################################
# 7  # 8      # -      # DFEForwardValue      # -
                  # -    # 1        # 1         # 1.00  # 0.01      #
#########################################################################################
# 8  # 9      # -      # DFEHashIndexJoin     # -
                  # -    # 2        # 1         # 0.50  # 0.35      #
#########################################################################################
# 9  # -      # -      # DFEDrain             # -
                  # -    # 1        # 0         # 0.00  # 0.02      #
#########################################################################################
```

最上位では、他の何よりも先に SolutionInjection が表示され, ユニットアウトは 1 です。実際にはソリューションを注入しないことに注意してください。次の演算子 DFESubquery のユニットインは 0 であることがわかります。

最上位の SolutionInjection の後は、DFESubquery および TermResolution 演算子です。DFESubquery は、DFE エンジンにプッシュされるクエリ実行プランの一部をカプセル化します (openCypher クエリの場合、クエリプラン全体が DFE によって実行されます)。クエリプラン内のすべての演算子は、DFESubquery によって参照される subQuery1 の内部にネストされています。唯一の例外は TermResolution であり、内部 ID を完全にシリアル化された openCypher オブジェクトにマテリアライズします。

DFE エンジンにプッシュされるすべての演算子の名前は DFE プレフィックスで始まります。前述のように、openCypher クエリプラン全体が DFE によって実行されるため、結果として、最後の TermResolution 演算子を除くすべての演算子は DFE で始まります。

subQuery1 の内部には、メモリ制限のあるメカニズムで実行されるプッシュ実行プランの一部をカプセル化する DFEChunkLocalSubQuery または DFELoopSubQuery 演算子が 0 個以上存在する可能性があります。DFEChunkLocalSubQuery は、ここでは、サブクエリへの入力として使用される 1 つの SolutionInjection を含みます。出力でそのサブクエリのテーブルを見つけるには、DFEChunkLocalSubQuery または DFELoopSubQuery 演算子の Arguments 列で指定された subQuery=*graph URI* を検索します。

subQuery1 では、ID が 0 のDFEPipelineScan は、データベースをスキャンして、指定された pattern を探します。このパターンは、プロパティ code が変数 ?n_code2 として保存されているエンティティをすべてのラベルにわたってスキャンします (airport を n:airport に付加することによって、特定のラベルに絞り込むことができます)。inlineFilters 引数は、code プロパティが ATL に等しい場合のフィルタリングを示します。

次に、DFEChunkLocalSubQuery 演算子は DFEPipelineJoin を含むサブクエリの中間結果を結合します。これによって、前の DFEPipelineScan は code プロパティを持つすべてのエンティティをスキャンするため、?n が実際にノードであることが保証されます。

制限付きのリレーションシップルックアップの **explain** 出力例

このクエリは、タイプ route の 2 つの匿名ノード間のリレーションシップを検索し、最大 10 を返します。やはり、explain モードは details であり、出力形式はデフォルトの ASCII 形式です。explain 出力は次のとおりです。

ここでは、DFEPipelineScan は、匿名ノード ?anon_node7 で始まり、別の匿名ノード ?anon_node21 で終わり、?p_type1 として保存されたリレーションシップタイプを持つエッジを

スキャンします。?p_type1 が el://route であるフィルターがあり (el はエッジラベルを表します)、クエリ文字列の [p:route] に対応します。

DFEDrain は、Arguments 列に示されているように、出力ソリューションを 10 個に制限して収集します。DFEDrain は、制限に達するか、すべてのソリューションが生成されるか、いずれか早い方で終了します。

```
curl -d "query=MATCH ()-[p:route]->() RETURN p LIMIT 10" -k https://localhost:8182/
openCypher -d "explain=details"
                                      ~
Query:
MATCH ()-[p:route]->() RETURN p LIMIT 10

################################################################################################
# ID # Out #1 # Out #2 # Name              # Arguments          # Mode                 #
 Units In # Units Out # Ratio # Time (ms) #
################################################################################################
# 0  # 1      # -      # SolutionInjection # solutions=[{}]     # -                    #
 0        # 1         # 0.00  # 0         #
################################################################################################
# 1  # 2      # -      # DFESubquery       # subQuery=subQuery1 # -                    #
 0        # 10        # 0.00  # 5.00      #
################################################################################################
# 2  # -      # -      # TermResolution    # vars=[?p]          # id2value_opencypher #
 10       # 10        # 1.00  # 1.00      #
################################################################################################


subQuery1
################################################################################################
# ID # Out #1 # Out #2 # Name            # Arguments
              # Mode # Units In # Units Out # Ratio # Time (ms) #
################################################################################################
# 0  # 1      # -      # DFEPipelineScan # pattern=Edge((?anon_node7)-[?p:?p_type1]->(?
anon_node21)) # -    # 0        # 1000      # 0.00  # 0.66      #
#    #        #        #                 # inlineFilters=[(?p_type1 IN [<el://route>])]
              #      #          #           #       #           #
#    #        #        #                 # patternEstimate=26219
              #      #          #           #       #           #
################################################################################################
# 1  # 2      # -      # DFEProject      # columns=[?p]
              # -    # 1000     # 1000      # 1.00  # 0.14      #
################################################################################################
```

```
# 2  # -      # -      # DFEDrain        # limit=10
             # -    # 1000     # 0         # 0.00  # 0.11      #
##################################################################################################
```

値式関数の **explain** 出力例

関数:

```
MATCH (a) RETURN DISTINCT labels(a)
```

以下の explain 出力では、DFEPipelineScan (ID 0) はすべてのノードラベルをスキャンします。これは MATCH（a に対応します。

DFEChunkLocalSubquery (ID 1) は、それぞれの ?a のラベルを集約します。これは labels(a) に対応します。DFEApply と DFEReduce を通してそれが分かります。

BindRelation (ID 2) は、列の汎用名 ?__gen_labelsOfa2 を ?labels(a) に変更するために使用されます。

DFEDistinctRelation (ID 4) は、異なるラベルのみを取得します (複数の :airport ノードがあると、重複したラベル ["airport"] になります)。これは DISTINCT labels(a) に対応します。

```
curl -d "query=MATCH (a) RETURN DISTINCT labels(a)" -k https://localhost:8182/
openCypher -d "explain=details"
                                                ~
Query:
MATCH (a) RETURN DISTINCT labels(a)

##################################################################################################
# ID # Out #1 # Out #2 # Name              # Arguments          # Mode                #
 Units In # Units Out # Ratio # Time (ms) #
##################################################################################################
# 0  # 1      # -      # SolutionInjection # solutions=[{}]     # -                   #
 0        # 1         # 0.00  # 0         #
##################################################################################################
# 1  # 2      # -      # DFESubquery       # subQuery=subQuery1 # -                   #
 0        # 5         # 0.00  # 81.00     #
##################################################################################################
# 2  # -      # -      # TermResolution    # vars=[?labels(a)]  # id2value_opencypher #
 5        # 5         # 1.00  # 1.00      #
##################################################################################################
```

```
subQuery1
#######################################################################################################
# ID # Out #1 # Out #2 # Name                    # Arguments
                                                                                  # Mode # Units
 In # Units Out # Ratio # Time (ms) #
#######################################################################################################
# 0  # 1      # -      # DFEPipelineScan        # pattern=Node(?a) with property 'ALL'
 and label '?a_label1'                                                              # -    # 0
    # 3750      # 0.00  # 26.77     #
#    #        #        #                        # patternEstimate=3506
                                                                                 #      #
  #           #       #           #
#######################################################################################################
# 1  # 2      # -      # DFEChunkLocalSubQuery # subQuery=http://aws.amazon.com/
neptune/vocab/v01/dfe/past/graph#8b314f55-2cc7-456a-a48a-c76a0465cfab/graph_1 # -    #
 3750     # 3750      # 1.00  # 0.04      #
#######################################################################################################
# 2  # 3      # -      # DFEBindRelation        # inputVars=[?a, ?__gen_labelsOfa2, ?
__gen_labelsOfa2]                                                                  # -    # 3750
     # 3750      # 1.00  # 0.08      #
#    #        #        #                        # outputVars=[?a, ?__gen_labelsOfa2, ?
labels(a)]                                                                         #      #
    #           #       #           #
#######################################################################################################
# 3  # 4      # -      # DFEProject             # columns=[?labels(a)]
                                                                                 # -    # 3750
  # 3750      # 1.00  # 0.05      #
#######################################################################################################
# 4  # 5      # -      # DFEDistinctRelation    # -
                                                                                 # -    # 3750
  # 5         # 0.00  # 2.78      #
#######################################################################################################
# 5  # -      # -      # DFEDrain               # -
                                                                                 # -    # 5
  # 0         # 0.00  # 0.03      #
#######################################################################################################


subQuery=http://aws.amazon.com/neptune/vocab/v01/dfe/past/graph#8b314f55-2cc7-456a-
a48a-c76a0465cfab/graph_1
#######################################################################################################
# ID # Out #1 # Out #2 # Name                   # Arguments
                    # Mode     # Units In # Units Out # Ratio # Time (ms) #
```

```
#########################################################################################
# 0  # 1      # -      # DFESolutionInjection # outSchema=[?a]
                    # -        # 0        # 3750      # 0.00  # 0.02      #
#########################################################################################
# 1  # 2      # 3      # DFETee               # -
                    # -        # 3750     # 7500      # 2.00  # 0.02      #
#########################################################################################
# 2  # 4      # -      # DFEProject           # columns=[?a]
                    # -        # 3750     # 3750      # 1.00  # 0.04      #
#########################################################################################
# 3  # 17     # -      # DFEOptionalJoin      # -
                    # -        # 7500     # 3750      # 0.50  # 0.44      #
#########################################################################################
# 4  # 5      # -      # DFEDistinctRelation  # -
                    # -        # 3750     # 3750      # 1.00  # 2.23      #
#########################################################################################
# 5  # 6      # -      # DFEDistinctColumn    # column=?a
                    # -        # 3750     # 3750      # 1.00  # 1.50      #
#    #        #        #                      # ordered=false
                    #          #          #           #       #           #
#########################################################################################
# 6  # 7      # -      # DFEPipelineJoin      # pattern=Node(?a) with property 'ALL'
 and label '?a_label3' # -        # 3750     # 3750      # 1.00  # 10.58     #
#    #        #        #                      # patternEstimate=3506
                    #          #          #           #       #           #
#########################################################################################
# 7  # 8      # 9      # DFETee               # -
                    # -        # 3750     # 7500      # 2.00  # 0.02      #
#########################################################################################
# 8  # 10     # -      # DFEBindRelation      # inputVars=[?a_label3]
                    # -        # 3750     # 3750      # 1.00  # 0.04      #
#    #        #        #                      # outputVars=[?100]
                    #          #          #           #       #           #
#########################################################################################
# 9  # 11     # -      # DFEBindRelation      # inputVars=[?a, ?a_label3, ?100]
                    # -        # 7500     # 3750      # 0.50  # 0.07      #
#    #        #        #                      # outputVars=[?a, ?a_label3, ?100]
                    #          #          #           #       #           #
#########################################################################################
# 10 # 9      # -      # DFETermResolution    # column=?100
                    # id2value # 3750     # 3750      # 1.00  # 7.60      #
#########################################################################################
# 11 # 12     # -      # DFEBindRelation      # inputVars=[?a, ?a_label3, ?100]
                    # -        # 3750     # 3750      # 1.00  # 0.06      #
```

```
#    #        #        #                    # outputVars=[?a, ?100, ?a_label3]
                    #          #          #           #       #           #
#####################################################################################
# 12 # 13     # -      # DFEApply            # functor=nodeLabel(?a_label3)
                    # -        # 3750     # 3750      # 1.00  # 0.55      #
#####################################################################################
# 13 # 14     # -      # DFEProject          # columns=[?a, ?a_label3_alias4]
                    # -        # 3750     # 3750      # 1.00  # 0.05      #
#####################################################################################
# 14 # 15     # -      # DFEMergeChunks       # -
                    # -        # 3750     # 3750      # 1.00  # 0.02      #
#####################################################################################
# 15 # 16     # -      # DFEReduce            # functor=collect(?a_label3_alias4)
                    # -        # 3750     # 3750      # 1.00  # 6.37      #
#    #        #        #                    # segmentationKey=[?a]
                    #          #          #           #       #           #
#####################################################################################
# 16 # 3      # -      # DFEMergeChunks       # -
                    # -        # 3750     # 3750      # 1.00  # 0.03      #
#####################################################################################
# 17 # -      # -      # DFEDrain             # -
                    # -        # 3750     # 0         # 0.00  # 0.02      #
#####################################################################################
```

数学値式関数の **explain** 出力例

この例では、RETURN abs(-10) は定数 -10 の絶対値を取り、単純な評価を行います。

DFEChunkLocalSubQuery (ID 1) は、変数 ?100 に格納されている静的な値 -10 に対してソリューション注入を実行します。

DFEApply (ID 2) は、?100 変数に格納されている静的な値に対して絶対値関数 abs() を実行する演算子です。

クエリと結果の explain 出力は次のとおりです。

```
curl -d "query=RETURN abs(-10)" -k https://localhost:8182/openCypher  -d
 "explain=details"
                                                    ~
Query:
RETURN abs(-10)

#####################################################################################
```

```
# ID # Out #1 # Out #2 # Name              # Arguments             # Mode
  # Units In # Units Out # Ratio # Time (ms) #
#########################################################################################
# 0  # 1      # -      # SolutionInjection # solutions=[{}]        # -
  # 0        # 1        # 0.00  # 0         #
#########################################################################################
# 1  # 2      # -      # DFESubquery       # subQuery=subQuery1    # -
  # 0        # 1        # 0.00  # 4.00      #
#########################################################################################
# 2  # -      # -      # TermResolution    # vars=[?_internalVar1] #
 id2value_opencypher # 1        # 1         # 1.00  # 1.00      #
#########################################################################################


subQuery1
#########################################################################################
# ID # Out #1 # Out #2 # Name                  # Arguments
                                                                    # Mode # Units
 In # Units Out # Ratio # Time (ms) #
#########################################################################################
# 0  # 1      # -      # DFESolutionInjection  # outSchema=[]
                                                                  # -    # 0
  # 1         # 0.00  # 0.01      #
#########################################################################################
# 1  # 2      # -      # DFEChunkLocalSubQuery # subQuery=http://aws.amazon.com/
neptune/vocab/v01/dfe/past/graph#c4cc6148-cce3-4561-93c0-deb91f257356/graph_1 # -    #
 1        # 1         # 1.00  # 0.03      #
#########################################################################################
# 2  # 3      # -      # DFEApply             # functor=abs(?100)
                                                                  # -    # 1
  # 1         # 1.00  # 0.26      #
#########################################################################################
# 3  # 4      # -      # DFEBindRelation       # inputVars=[?_internalVar2, ?
_internalVar2]                                                                 # -
 # 1        # 1         # 1.00  # 0.04      #
#    #        #        #                      # outputVars=[?_internalVar2, ?
_internalVar1]                                                               #
 #          #           #       #           #
#########################################################################################
# 4  # 5      # -      # DFEProject           # columns=[?_internalVar1]
                                                                  # -    # 1
  # 1         # 1.00  # 0.06      #
#########################################################################################
```

```
# 5  # -      # -      # DFEDrain              # -
                                                                  # -    # 1
  # 0         # 0.00  # 0.05      #
#########################################################################################################

subQuery=http://aws.amazon.com/neptune/vocab/v01/dfe/past/graph#c4cc6148-
cce3-4561-93c0-deb91f257356/graph_1
#########################################################################################################
# ID # Out #1 # Out #2 # Name                 # Arguments                                  #
 Mode # Units In # Units Out # Ratio # Time (ms) #
#########################################################################################################
# 0  # 1      # -      # DFESolutionInjection # solutions=[?100 -> [-10^^<LONG>]]   # -
    # 0        # 1         # 0.00  # 0.01      #
#    #        #        #                      # outSchema=[?100]                           #
    #          #           #       #           #
#########################################################################################################
# 1  # 3      # -      # DFERelationalJoin    # joinVars=[]                                # -
    # 2        # 1         # 0.50  # 0.18      #
#########################################################################################################
# 2  # 1      # -      # DFESolutionInjection # outSchema=[]                             # -
    # 0        # 1         # 0.00  # 0.01      #
#########################################################################################################
# 3  # -      # -      # DFEDrain              # -                                          # -
    # 1        # 0         # 0.00  # 0.02      #
#########################################################################################################
```

可変長パス (VLP) クエリの **explain** 出力例

これは、可変長パスクエリを処理するための、より複雑なクエリプランの例です。この例では、わかりやすくするために、explain 出力の一部のみを示しています。

subQuery1 では、...graph_1 サブクエリを挿入する DFEPipelineScan (ID 0) と DFEChunkLocalSubQuery (ID 1) が、YPO コードを含むノードをスキャンします。

subQuery1 では、...graph_2 サブクエリを挿入する DFEChunkLocalSubQuery (ID 0) が、LAX コードを含むノードをスキャンします。

subQuery1 では、DFEChunkLocalSubQuery (ID 3) が ...graph3 サブクエリを挿入し、これが DFELoopSubQuery (ID 17) を含み、これが ...graph5 サブクエリを挿入します。この操作は、2 つのノード間のクエリ文字列の -[*2]-> 可変長パターンを解決する役割を果たします。

```
curl -d "query=MATCH p=(a {code: 'YPO'})-[*2]->(b{code: 'LAX'}) return p" -k https://
localhost:8182/openCypher -d "explain=details"
```

```
                                                ~
Query:
MATCH p=(a {code: 'YPO'})-[*2]->(b{code: 'LAX'}) return p

##########################################################################################
# ID # Out #1 # Out #2 # Name              # Arguments          # Mode                #
 Units In # Units Out # Ratio # Time (ms) #
##########################################################################################
# 0  # 1      # -      # SolutionInjection # solutions=[{}]     # -                   #
 0        # 1         # 0.00  # 0         #
##########################################################################################
# 1  # 2      # -      # DFESubquery       # subQuery=subQuery1 # -                   #
 0        # 0         # 0.00  # 84.00     #
##########################################################################################
# 2  # -      # -      # TermResolution    # vars=[?p]          # id2value_opencypher #
 0        # 0         # 0.00  # 0         #
##########################################################################################


subQuery1
##########################################################################################
# ID # Out #1 # Out #2 # Name                  # Arguments
                                                                  # Mode # Units
 In # Units Out # Ratio # Time (ms) #
##########################################################################################
# 0  # 1      # -      # DFEPipelineScan       # pattern=Node(?a) with property 'code'
 as ?a_code7 and label 'ALL'                                       # -    # 0
   # 1         # 0.00  # 0.68      #
#    #        #        #                       # inlineFilters=[(?a_code7 IN
 ["YPO"^^xsd:string])]                                                  #
  #          #           #       #           #
#    #        #        #                       # patternEstimate=1
                                                                 #      #
  #           #       #           #
##########################################################################################
# 1  # 2      # -      # DFEChunkLocalSubQuery # subQuery=http://aws.amazon.com/
neptune/vocab/v01/dfe/past/graph#cc05129f-d07e-4622-bbe3-9e99558eca46/graph_1 # -    #
 1        # 1         # 1.00  # 0.03      #
##########################################################################################
# 2  # 3      # -      # DFEChunkLocalSubQuery # subQuery=http://aws.amazon.com/
neptune/vocab/v01/dfe/past/graph#cc05129f-d07e-4622-bbe3-9e99558eca46/graph_2 # -    #
 1        # 1         # 1.00  # 0.02      #
##########################################################################################
```

```
# 3  # 4      # -      # DFEChunkLocalSubQuery # subQuery=http://aws.amazon.com/
neptune/vocab/v01/dfe/past/graph#cc05129f-d07e-4622-bbe3-9e99558eca46/graph_3 # -    #
 1        # 0         # 0.00  # 0.04      #
##########################################################################################
# 4  # 5      # -      # DFEBindRelation       # inputVars=[?__gen_path6, ?
anon_rel26, ?b_code8, ?b, ?a_code7, ?a, ?__gen_path6]                              # -
   # 0        # 0         # 0.00  # 0.10      #
#    #        #        #                       # outputVars=[?__gen_path6, ?
anon_rel26, ?b_code8, ?b, ?a_code7, ?a, ?p]                                      #
  #          #           #       #           #
##########################################################################################
# 5  # 6      # -      # DFEProject            # columns=[?p]
                                                                          # -    # 0
  # 0         # 0.00  # 0.05      #
##########################################################################################
# 6  # -      # -      # DFEDrain              # -
                                                                          # -    # 0
  # 0         # 0.00  # 0.02      #
##########################################################################################


subQuery=http://aws.amazon.com/neptune/vocab/v01/dfe/past/graph#cc05129f-d07e-4622-
bbe3-9e99558eca46/graph_1
##########################################################################################
# ID # Out #1 # Out #2 # Name                 # Arguments
                    # Mode # Units In # Units Out # Ratio # Time (ms) #
##########################################################################################
# 0  # 1      # -      # DFESolutionInjection # outSchema=[?a, ?a_code7]
                    # -    # 0        # 1         # 0.00  # 0.01      #
##########################################################################################
# 1  # 2      # 3      # DFETee                # -
                    # -    # 1        # 2         # 2.00  # 0.01      #
##########################################################################################
# 2  # 4      # -      # DFEDistinctColumn     # column=?a
                    # -    # 1        # 1         # 1.00  # 0.25      #
#    #        #        #                       # ordered=false
                    #      #          #           #       #           #
##########################################################################################
# 3  # 5      # -      # DFEHashIndexBuild     # vars=[?a]
                    # -    # 1        # 1         # 1.00  # 0.05      #
##########################################################################################
# 4  # 5      # -      # DFEPipelineJoin       # pattern=Node(?a) with property 'ALL'
 and label '?a_label1' # -    # 1        # 1         # 1.00  # 0.47      #
```

```
#    #        #        #                    # patternEstimate=3506
                     #      #          #           #       #           #
#################################################################################################
# 5  # 6      # 7      # DFESync              # -
                     # -    # 2        # 2         # 1.00  # 0.04      #
#################################################################################################
# 6  # 8      # -      # DFEForwardValue      # -
                     # -    # 1        # 1         # 1.00  # 0.01      #
#################################################################################################
# 7  # 8      # -      # DFEForwardValue      # -
                     # -    # 1        # 1         # 1.00  # 0.01      #
#################################################################################################
# 8  # 9      # -      # DFEHashIndexJoin     # -
                     # -    # 2        # 1         # 0.50  # 0.26      #
#################################################################################################
# 9  # -      # -      # DFEDrain             # -
                     # -    # 1        # 0         # 0.00  # 0.02      #
#################################################################################################


subQuery=http://aws.amazon.com/neptune/vocab/v01/dfe/past/graph#cc05129f-d07e-4622-
bbe3-9e99558eca46/graph_2
#################################################################################################
# ID # Out #1 # Out #2 # Name                 # Arguments
                           # Mode # Units In # Units Out # Ratio # Time (ms) #
#################################################################################################
# 0  # 1      # -      # DFEPipelineScan      # pattern=Node(?b) with property 'code'
 as ?b_code8 and label 'ALL' # -    # 0        # 1         # 0.00  # 0.38      #
#    #        #        #                    # inlineFilters=[(?b_code8 IN
 ["LAX"^^xsd:string])]                #      #          #           #       #
  #
#    #        #        #                    # patternEstimate=1
                           #      #          #           #       #           #
#################################################################################################
# 1  # 2      # -      # DFEMergeChunks       # -
                           # -    # 1        # 1         # 1.00  # 0.02      #
#################################################################################################
# 2  # 4      # -      # DFERelationalJoin    # joinVars=[]
                           # -    # 2        # 1         # 0.50  # 0.19      #
#################################################################################################
# 3  # 2      # -      # DFESolutionInjection # outSchema=[?a, ?a_code7]
                           # -    # 0        # 1         # 0.00  # 0         #
#################################################################################################
```

```
# 4  # -      # -      # DFEDrain             # -
                          # -    # 1        # 0         # 0.00  # 0.01      #
#########################################################################################


subQuery=http://aws.amazon.com/neptune/vocab/v01/dfe/past/graph#cc05129f-d07e-4622-
bbe3-9e99558eca46/graph_3
#########################################################################################
# ID # Out #1 # Out #2 # Name                  # Arguments
                                                                    # Mode     #
 Units In # Units Out # Ratio # Time (ms) #
#########################################################################################
...
# 17 # 18     # -      # DFELoopSubQuery       # subQuery=http://aws.amazon.com/
neptune/vocab/v01/dfe/past/graph#cc05129f-d07e-4622-bbe3-9e99558eca46/graph_5 # -
  # 1        # 2         # 2.00  # 0.31      #
...
```

Neptune openCypher でのトランザクション

Amazon Neptune の openCypher 実装は、Neptune によって定義されたトランザクションセマンティクスを使用します。ただし、Bolt ドライバーによって提供される分離レベルは、以下のセクションで説明するように、Bolt トランザクションセマンティクスに特定の影響を及ぼします。

読み取り専用の Bolt トランザクションクエリ

読み取り専用クエリの処理には、次のようなさまざまなトランザクションモデルと分離レベルがあります。

暗黙的な読み取り専用トランザクションクエリ

読み取り専用の暗黙的トランザクションの例を次に示します。

```
public void executeReadImplicitTransaction()
{
  // end point
  final String END_POINT = "(End Point URL)";

  // read query
  final String READ_QUERY = "MATCH (n) RETURN n limit 10";

  // create the driver
  final Driver driver = GraphDatabase.driver(END_POINT, AuthTokens.none(),
```

```
          Config.builder().withEncryption()
                          .withTrustStrategy(TrustStrategy.trustSystemCertificates())
                          .build());

  // create the session config
  SessionConfig sessionConfig = SessionConfig.builder()
                                             .withFetchSize(1000)
                                             .withDefaultAccessMode(AccessMode.READ)
                                             .build();

  // run the query as access mode read
  driver.session(sessionConfig).readTransaction(new TransactionWork<String>()
    {
      final StringBuilder resultCollector = new StringBuilder();

      @Override
      public String execute(final Transaction tx)
      {
        // execute the query
        Result queryResult = tx.run(READ_QUERY);

        // Read the result
        for (Record record : queryResult.list())
        {
          for (String key : record.keys())
          {
            resultCollector.append(key)
                           .append(":")
                           .append(record.get(key).asNode().toString());
          }
        }
        return resultCollector.toString();
      }

    }
  );

  // close the driver.
  driver.close();
}
```

リードレプリカは読み取り専用クエリしか受け付けないため、リードレプリカに対するすべてのクエリは、セッション構成に設定されているアクセスモードに関係なく、読み取り暗黙のトランザクショ

ンとして実行されます。Neptune は、SNAPSHOT 分離セマンティクスでは、読み取り暗黙のトランザクションを読み取り専用クエリとして評価します。

障害が発生すると、読み取り暗黙のトランザクションはデフォルトで再試行されます。

読み取り専用トランザクションクエリをオートコミットする

読み取り専用のオートコミットトランザクションの例を次に示します。

```
public void executeAutoCommitTransaction()
{
  // end point
  final String END_POINT = "(End Point URL)";

  // read query
  final String READ_QUERY = "MATCH (n) RETURN n limit 10";

  // Create the session config.
  final SessionConfig sessionConfig = SessionConfig
    .builder()
    .withFetchSize(1000)
    .withDefaultAccessMode(AccessMode.READ)
    .build();

  // create the driver
  final Driver driver = GraphDatabase.driver(END_POINT, AuthTokens.none(),
    Config.builder()
          .withEncryption()
          .withTrustStrategy(TrustStrategy.trustSystemCertificates())
          .build());

  // result collector
  final StringBuilder resultCollector = new StringBuilder();

  // create a session
  final Session session = driver.session(sessionConfig);

  // run the query
  final Result queryResult = session.run(READ_QUERY);
  for (final Record record : queryResult.list())
  {
    for (String key : record.keys())
    {
      resultCollector.append(key)
```

```
                    .append(":")
                    .append(record.get(key).asNode().toString());
    }
  }

  // close the session
  session.close();

  // close the driver
  driver.close();
}
```

セッション設定でアクセスモードが READ に設定されている場合、Neptune は、SNAPSHOT 分離セマンティクスではオートコミットトランザクションクエリを読み取り専用クエリとして評価します。リードレプリカは読み取り専用クエリしか受け付けないことに注意してください。

セッション設定を渡さない場合、オートコミットクエリはデフォルトでミューテーションクエリを分離して処理されるため、アクセスモードを明示的に READ に設定したセッション設定を渡すことが重要です。

失敗した場合、読み取り専用のオートコミットクエリは再試行されません。

暗黙的な読み取り専用トランザクションクエリ

以下は明示的な読み取り専用トランザクションの例です。

```
public void executeReadExplicitTransaction()
{
  // end point
  final String END_POINT = "(End Point URL)";

  // read query
  final String READ_QUERY = "MATCH (n) RETURN n limit 10";

  // Create the session config.
  final SessionConfig sessionConfig = SessionConfig
    .builder()
    .withFetchSize(1000)
    .withDefaultAccessMode(AccessMode.READ)
    .build();

  // create the driver
  final Driver driver = GraphDatabase.driver(END_POINT, AuthTokens.none(),
```

```
    Config.builder()
          .withEncryption()
          .withTrustStrategy(TrustStrategy.trustSystemCertificates())
          .build());

  // result collector
  final StringBuilder resultCollector = new StringBuilder();

  // create a session
  final Session session = driver.session(sessionConfig);

  // begin transaction
  final Transaction tx = session.beginTransaction();

  // run the query on transaction
  final List<Record> list = tx.run(READ_QUERY).list();

  // read the result
  for (final Record record : list)
  {
    for (String key : record.keys())
    {
      resultCollector
        .append(key)
        .append(":")
        .append(record.get(key).asNode().toString());
    }
  }

  // commit the transaction and for rollback we can use beginTransaction.rollback();
  tx.commit();

  // close the driver
  driver.close();
}
```

セッション設定でアクセスモードが READ に設定されている場合、Neptune は、SNAPSHOT 分離セマンティクスでは明示的な読み取り専用トランザクションを読み取り専用クエリとして評価します。リードレプリカは読み取り専用クエリしか受け付けないことに注意してください。

セッション設定を渡さない場合、明示的読み取り専用トランザクションは、デフォルトではミューテーションクエリ分離で処理されるため、アクセスモードを明示的に READ に設定したセッション設定を渡すことが重要です。

失敗した場合、読み取り専用の明示的なクエリはデフォルトで再試行されます。

ミューテーション Bolt トランザクションクエリ

読み取り専用クエリと同様、ミューテーションクエリの処理にはさまざまな方法があり、次のようなさまざまなトランザクションモデルと分離レベルがあります。

暗黙的なミューテーショントランザクションクエリ

暗黙的なミューテーショントランザクションの例を示します。

```
public void executeWriteImplicitTransaction()
{
  // end point
  final String END_POINT = "(End Point URL)";

  // create node with label as label and properties.
  final String WRITE_QUERY = "CREATE (n:label {name : 'foo'})";

  // Read the vertex created with label as label.
  final String READ_QUERY = "MATCH (n:label) RETURN n";

  // create the driver
  final Driver driver = GraphDatabase.driver(END_POINT, AuthTokens.none(),
    Config.builder()
          .withEncryption()
          .withTrustStrategy(TrustStrategy.trustSystemCertificates())
          .build());

  // create the session config
  SessionConfig sessionConfig = SessionConfig
    .builder()
    .withFetchSize(1000)
    .withDefaultAccessMode(AccessMode.WRITE)
    .build();

  final StringBuilder resultCollector = new StringBuilder();

  // run the query as access mode write
  driver.session(sessionConfig).writeTransaction(new TransactionWork<String>()
  {
    @Override
    public String execute(final Transaction tx)
```

```
    {
      // execute the write query and consume the result.
      tx.run(WRITE_QUERY).consume();

      // read the vertex written in the same transaction
      final List<Record> list = tx.run(READ_QUERY).list();

      // read the result
      for (final Record record : list)
      {
        for (String key : record.keys())
        {
          resultCollector
            .append(key)
            .append(":")
            .append(record.get(key).asNode().toString());
        }
      }
      return resultCollector.toString();
    }
  }); // at the end, the transaction is automatically committed.

  // close the driver.
  driver.close();
}
```

ミューテーションクエリの一部として行われる読み取りは、Neptune ミューテーショントランザクションでは通常の保証に従って READ COMMITTED 分離で実行されます。

セッション設定を具体的に渡すかどうかにかかわらず、そのトランザクションは常に書き込みトランザクションとして扱われます。

コンフリクトについては、「ロック待機タイムアウトを使用した競合の解決」を参照してください。

オートコミットミューテーショントランザクションクエリ

ミューテーションオートコミットクエリは、ミューテーション暗黙的トランザクションと同じ動作を継承します。

セッション設定を渡さなかった場合、トランザクションはデフォルトでは書き込みトランザクションとして扱われます。

失敗した場合、ミューテーションオートコミットクエリは、自動的に再試行されません。

明示的なミューテーショントランザクションクエリ

明示的なミューテーショントランザクションの例を示します。

```
public void executeWriteExplicitTransaction()
{
  // end point
  final String END_POINT = "(End Point URL)";

  // create node with label as label and properties.
  final String WRITE_QUERY = "CREATE (n:label {name : 'foo'})";

  // Read the vertex created with label as label.
  final String READ_QUERY = "MATCH (n:label) RETURN n";

  // create the driver
  final Driver driver = GraphDatabase.driver(END_POINT, AuthTokens.none(),
    Config.builder()
          .withEncryption()
          .withTrustStrategy(TrustStrategy.trustSystemCertificates())
          .build());

  // create the session config
  SessionConfig sessionConfig = SessionConfig
    .builder()
    .withFetchSize(1000)
    .withDefaultAccessMode(AccessMode.WRITE)
    .build();

  final StringBuilder resultCollector = new StringBuilder();

  final Session session = driver.session(sessionConfig);

  // run the query as access mode write
  final Transaction tx = driver.session(sessionConfig).beginTransaction();

  // execute the write query and consume the result.
  tx.run(WRITE_QUERY).consume();

  // read the result from the previous write query in a same transaction.
  final List<Record> list = tx.run(READ_QUERY).list();

  // read the result
  for (final Record record : list)
```

```
  {
    for (String key : record.keys())
    {
      resultCollector
        .append(key)
        .append(":")
        .append(record.get(key).asNode().toString());
    }
  }

  // commit the transaction and for rollback we can use tx.rollback();
  tx.commit();

  // close the session
  session.close();

  // close the driver.
  driver.close();
}
```

明示的なミューテーションクエリは、暗黙的なミューテーショントランザクションと同じ動作を継承します。

セッション設定を渡さなかった場合、トランザクションはデフォルトでは書き込みトランザクションとして扱われます。

コンフリクトについては、「ロック待機タイムアウトを使用した競合の解決」を参照してください。

Neptune openCypher の制限事項

Amazon Neptune リリースの openCypher では、openCypher 仕様コンプライアンス で説明されているように、「Cypher クエリ言語リファレンス、バージョン 9」で指定されているすべての内容がまだサポートされているわけではありません。将来のリリースでは、これらの制限の多くに対処する予定です。

Neptune openCypher の例外

Amazon Neptune で openCypher を操作する場合、さまざまな例外が発生する可能性があります。HTTPS エンドポイントまたは Bolt ドライバーから発生する可能性がある一般的な例外を以下に示します (Bolt ドライバーからの例外はすべてサーバー状態例外として報告されます)。

HTTP コード	エラーメッセージ	再取得可能?	回避策	
400	(openCypher パーサーから直接伝達される構文エラー)	いいえ	クエリ構文を修正して再試行してください。	
500	`Operation terminated (out of memory)`	はい	クエリを作り直してフィルタ条件を追加し、必要なメモリを削減してください。	
500	操作は終了しました (期限超過)	はい	DB クラスターパラメータグループのクエリタイムアウトを増やすか、リクエストを再試行してください。	
500	操作は終了しました (ユーザーによってキャンセルされました)	はい	リクエストを再試行します。	
500	データベースのリセットが進行中です。クラスターが使用可能になったら、クエリを再試行してください。	はい	リセットが完了したら再試行してください。	
500	同時操作が競合しているため操	はい	指数バックオフとリトライ戦	

HTTP コード	エラーメッセージ	再取得可能?	回避策	
	作が失敗しました (再試行してください)。トランザクションは現在ロールバック中です。		略を使用して再試行してください。	
400	*(###)* 操作/機能がサポートされていない例外	いいえ	指定されたオペレーションは、サポートされていません。	
400	読み取り専用レプリカに対して openCypher の更新が試みられました	いいえ	ターゲットエンドポイントをライターエンドポイントに変更します。	
400	Malformed QueryException (Neptune は内部パーサーの状態を表示しません)	いいえ	クエリ構文を修正して再試行してください。	
400	ノードにはまだリレーションシップがあるため削除できません。このノードを削除するには、まずリレーションシップを削除する必要があります。	いいえ	`MATCH (n) DELETE n` を使用する代わりに `MATCH(n) DETACH DELETE(n)` を使用してください。	

HTTP コード	エラーメッセージ	再取得可能?	回避策	
400	無効な操作: ノードの最後のラベルを削除しようとしています。ノードには少なくとも 1 つのラベルが必要です。	いいえ	Neptune では、すべてのノードに少なくとも 1 つのラベルが必要であり、明示的なラベルなしでノードを作成すると、デフォルトのラベル `vertex` が割り当てられます。最後のラベルが削除されないように、クエリやアプリケーションのロジックを変更してください。ノードのシングルトンラベルは、新しいラベルを設定してから古いラベルを削除することで更新できます。	
500	リクエストの最大数が超過しました。 ConfiguredQueueCapacity= connId の {} = {}	はい	現在、スタックやプロトコルに関係なく、処理できる同時リクエストは 8,192 件のみです。	

HTTP コード	エラーメッセージ	再取得可能?	回避策	
500	最大接続制限を超えました。	はい	1 つのインスタンスで許可される Bolt の同時接続数は 1000 件までです (HTTP には制限はありません)。	
400	[ノード、リレーションシップ、パスのいずれか] が必要で、リテラルを取得しました	いいえ	正しい引数、正しいクエリ構文を渡していることを確認して、再試行してください。	
400	プロパティ値は単純なリテラルでなければなりません。または:Set プロパティのマップが必要でしたが、見つかりませんでした。	いいえ	SET 句は単純なリテラルのみを受け入れ、複合型は受け付けません。	
400	見つかったエンティティは削除対象として渡されましたが、見つかりません。	いいえ	削除しようとしているエンティティがデータベースに存在することを確認してください。	

HTTP コード	エラーメッセージ	再取得可能?	回避策	
400	ユーザーにはデータベースへのアクセス権がない場合の確認	いいえ	使用中の IAM ロールのポリシーを確認してください。	
400	リクエストの一部としてトークンは渡されていません。	いいえ	IAM 対応クラスターでは、適切に署名されたトークンをクエリリクエストの一部として渡す必要があります。	
400	エラーメッセージは伝播されます。	いいえ	リクエスト ID を使用して AWS サポートにお問い合わせください。	
500	操作は終了しました (内部エラー)	はい	リクエスト ID を使用して AWS サポートにお問い合わせください。	

SPARQL を使用した Neptune グラフへのアクセス

SPARQL は、ウェブ用に設計されたグラフデータ形式であるリソース記述フレームワーク (RDF) のためのクエリ言語です。Amazon Neptune は、SPARQL 1.1 と互換性があります。つまり、Neptune DB インスタンス に接続して、SPARQL 1.1 クエリ言語仕様で記述されたクエリ言語を使用してグラフにクエリを実行できるということです。

SPARQL のクエリは、返す変数を指定する SELECT 句と、グラフで一致するデータを指定する WHERE 句で構成されます。SPARQL クエリに慣れていない場合は、SPARQL 1.1 クエリ言語にある「シンプルなクエリの書き込み」を参照してください。

> **Important**
>
> データをロードするには、SPARQL UPDATE INSERT は小さなデータセットに対してはうまく機能しますが、ファイルから大量のデータをロードする必要がある場合は、「Amazon Neptune 一括ローダーを使用したデータの取り込み」を参照してください。

Neptune の SPARQL 実装の仕様の詳細については、SPARQL 標準準拠を参照してください。

始めるには以下のものが必要です。

- Neptune DB インスタンス。Neptune DB インスタンスの作成については、新しい Neptune DB クラスターの作成 を参照してください。
- Neptune DB インスタンスと同じ Virtual Private Cloud (VPC) にある Amazon EC2; インスタンス。

トピック

RDF4J コンソールを使用して Neptune DB インスタンスに接続する

RDF4J コンソールでは、REPL (read-eval-print ループ) 環境で Resource Description Framework (RDF) グラフとクエリを試すことができます。

リポジトリとしてリモートグラフデータベースを追加して、RDF4J コンソールからクエリを実行できます。このセクションでは、Neptune DB インスタンス にリモートで接続するための RDF4J コンソールの設定について説明します。

RDF4J コンソールを使用して Neptune に接続するには

1. RDF4J ウェブサイトのダウンロードページから、RDF4J SDK をダウンロードしてください。
2. RDF4J SDK zip ファイルを解凍します。
3. ターミナルで RDF4J SDK ディレクトリに移動し、次のコマンドを入力して RDF4J コンソールを実行します。

   ```
   bin/console.sh
   ```

   次のような出力が表示されます:

   ```
   14:11:51.126 [main] DEBUG o.e.r.c.platform.PlatformFactory - os.name = linux
   14:11:51.130 [main] DEBUG o.e.r.c.platform.PlatformFactory - Detected Posix
    platform
   Connected to default data directory
   RDF4J Console 3.6.1

   3.6.1
   Type 'help' for help.
   >
   ```

   > プロンプトが表示されます。これは、RDF4J コンソールの一般プロンプトです。このプロンプトを使用して、リポジトリやその他の操作を設定します。リポジトリでは、クエリを実行するための独自のプロンプトが表示されます。
4. > プロンプトで、次のように入力して Neptune DB インスタンスに SPARQL リポジトリを作成します。

   ```
   create sparql
   ```
5. RDF4J コンソールで、SPARQL エンドポイントへの接続に必要な変数の値を求められます。

```
Please specify values for the following variables:
```

次の値を指定します。

変数名	値
SPARQL クエリのエンドポイント	`https://your-neptune-endpoint :port/sparql`
SPARQL 更新エンドポイント	`https://your-neptune-endpoint :port/sparql`
ローカルリポジトリ ID [endpoint@localhost]	`neptune`
リポジトリタイトル [SPARQL エンドポイントリポジトリ @localhost]	`Neptune DB instance`

Neptune DB インスタンスのアドレスを見つける方法については、Amazon Neptune エンドポイントに接続する セクションを参照してください。

このオペレーションが成功した場合は、次のメッセージが表示されます。

```
Repository created
```

6. > プロンプトで、次のように入力して Neptune DB インスタンスに接続します。

```
open neptune
```

このオペレーションが成功した場合は、次のメッセージが表示されます。

```
Opened repository 'neptune'
```

neptune> プロンプトが表示されます。このプロンプトで、Neptune グラフに対するクエリを実行することができます。

> **Note**
>
> リポジトリが追加されました。次に `bin/console.sh` を実行する場合は、直接 `open neptune` コマンドを実行して Neptune DB インスタンスに接続することができます。

7. `neptune>` プロンプトで、次のように入力して SPARQL クエリを実行します。このクエリは、10 の制限がある `?s ?p ?o` クエリを使用して、グラフのトリプル (主語 - 述語 - 目的語) のうち最大 10 個を返します。その他の対象にクエリを実行するには、`sparql` コマンドの後のテキストを別の SPARQL クエリで置き換えます。

```
sparql select ?s ?p ?o where {?s ?p ?o} limit 10
```

RDF4J Workbench を使用して Neptune DB インスタンスに接続する

このセクションでは、RDF4J Workbench および RDF4J Server を使用してAmazon Neptune DB インスタンスに接続する方法について説明します。RDF4J Server は、Neptune SPARQL HTTP REST エンドポイントと RDF4J Workbench 間のプロキシとして必要です。

RDF4J Workbench では、ローカルファイルのロードを含め、グラフの実験を行う簡単なインターフェイスを提供します。詳しくは、RDF4J ドキュメントにある「セクションを追加する」を参照してください。

前提条件

開始する前に、以下を実行します。

- Java 1.8 以降をインストールします。
- RDF4J Server および RDF4J Workbench をインストールします。詳細については、「RDF4J Server および RDF4J Workbench のインストール」を参照してください。

RDF4J Workbench を使用して Neptune に接続するには

1. ウェブブラウザで、RDF4J Workbench ウェブアプリケーションがデプロイされている URL に移動します。たとえば、Apache Tomcat を使用している場合の URL は次のようになります。https://*ec2_hostname*:8080/rdf4j-workbench/。

2. [Connect to RDF4J Server (RDF4J Server に接続)] と求められたら、RDF4J Server がインストールされて実行中であり、サーバーの URL が正しいことを確認します。そして、次のステップに進みます。
3. 左のペインの [New repository] (新しいレポジトリ) を選択します。

   [New repository(新しいレポジトリ)]

   - [Type (タイプ)] ドロップダウンリストで、[SPARQL endpoint proxy (SPARQL エンドポイントのプロキシ)] を選択します。
   - [ID] に [neptune] を入力します。
   - 役職の場合、タイプ Neptune DB インスタンス。

   [次へ] をクリックします。
4. [New repository(新しいレポジトリ)]

   - [SPARQL query endpoint URL (SPARQL クエリのエンドポイント URL)] については、「`https://`*`your-neptune-endpoint`*`:`*`port`*`/sparql`」と入力します。
   - [SPARQL update endpoint URL (SPARQL 更新のエンドポイント URL)] については、「`https://`*`your-neptune-endpoint`*`:`*`port`*`/sparql`」と入力します。

   Neptune DB インスタンスのアドレスを見つける方法については、Amazon Neptune エンドポイントに接続する セクションを参照してください。

   [Create] (作成) を選択します。
5. [neptune] リポジトリが、リポジトリのリストに表示されます。新しいリポジトリが使用できるようになるまで数分かかることがあります。
6. テーブルの [ID] 列で、[neptune] リンクをクリックします。
7. 左のペインの [Query] (クエリ) を選択します。

> **Note**
>
> [Explore (詳しく見る)] の下にあるメニュー項目が無効になっている場合は、RDF4J Server に再接続してもう一度 [neptune] リポジトリを選択する必要があるかもしれません。
> これを行うには、右上隅にある [change] (変更) リンクを使用します。

8. クエリフィールドで次の SPARQL クエリを入力し、[Execute] (実行) を選択します。

```
select ?s ?p ?o where {?s ?p ?o} limit 10
```

前述の例では、10 の制限がある ?s ?p ?o クエリを使用して、グラフのトリプル (主語 - 述語 - 目的語) のうち最大 10 個を返します。

Java を使用して Neptune DB インスタンスに接続する

このセクションでは、Amazon Neptune DB インスタンスに接続し、SPARQL クエリを実行する完全な Java サンプルを実行する方法について説明します。

Neptune DB インスタンスと同じ仮想プライベートクラウド (VPC) の Amazon EC2 インスタンスからこれらの手順を実行してください。

Java を使用して Neptune に接続するには

1. Apache Maven を EC2 インスタンスにインストールします。最初に、Maven パッケージを使用してリポジトリを追加するには、次のように入力します。

   ```
   sudo wget https://repos.fedorapeople.org/repos/dchen/apache-maven/epel-apache-maven.repo -O /etc/yum.repos.d/epel-apache-maven.repo
   ```

   パッケージのバージョン番号を設定するには、次のように入力します。

   ```
   sudo sed -i s/\$releasever/6/g /etc/yum.repos.d/epel-apache-maven.repo
   ```

   続いて、yum を使用して Maven をインストールできます。

   ```
   sudo yum install -y apache-maven
   ```

2. この例は Java 8 のみでテストされています。EC2 インスタンスで Java 8 をインストールするには、次のように入力します。

   ```
   sudo yum install java-1.8.0-devel
   ```

3. EC2 インスタンスで Java 8 をデフォルトランタイムとして設定するには、次のように入力します。

```
sudo /usr/sbin/alternatives --config java
```

プロンプトが表示されたら、Java 8 の数を入力します。

4. EC2 インスタンスで Java 8 をデフォルトコンパイラとして設定するには、次のように入力します。

```
sudo /usr/sbin/alternatives --config javac
```

プロンプトが表示されたら、Java 8 の数を入力します。

5. 新しいディレクトリで pom.xml ファイルを作成してから、テキストエディタで開きます。

6. 以下を pom.xml ファイルにコピーし保存します (通常、バージョン番号を最新の安定バージョンに調整できます)。

```
<project xmlns="https://maven.apache.org/POM/4.0.0" xmlns:xsi="https://
www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="https://maven.apache.org/POM/4.0.0 https://maven.apache.org/
maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.amazonaws</groupId>
  <artifactId>RDFExample</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>RDFExample</name>
  <url>https://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>org.eclipse.rdf4j</groupId>
      <artifactId>rdf4j-runtime</artifactId>
      <version>3.6</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
          <groupId>org.codehaus.mojo</groupId>
          <artifactId>exec-maven-plugin</artifactId>
          <version>1.2.1</version>
```

```
          <configuration>
            <mainClass>com.amazonaws.App</mainClass>
          </configuration>
        </plugin>
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-compiler-plugin</artifactId>
          <configuration>
            <source>1.8</source>
            <target>1.8</target>
          </configuration>
        </plugin>
      </plugins>
    </build>
</project>
```

> **Note**
>
> 既存の Maven プロジェクトを変更する場合、必要な依存関係が前述のコードにおいて強調表示されます。

7. ソースコード例 (src/main/java/com/amazonaws/) のサブディレクトリを作成するには、コマンドラインで次のように入力してください。

```
mkdir -p src/main/java/com/amazonaws/
```

8. src/main/java/com/amazonaws/ ディレクトリで App.java という名前のファイルを作成してから、テキストエディタで開きます。
9. App.java ファイルに次の内容をコピーします。*your-neptune-endpoint* を Neptune DB インスタンスのアドレスで置き換えます。

> **Note**
>
> Neptune DB インスタンスのホスト名を見つける方法については、Amazon Neptune エンドポイントに接続する セクションを参照してください。

```
package com.amazonaws;

import org.eclipse.rdf4j.repository.Repository;
```

```
import org.eclipse.rdf4j.repository.http.HTTPRepository;
import org.eclipse.rdf4j.repository.sparql.SPARQLRepository;

import java.util.List;
import org.eclipse.rdf4j.RDF4JException;
import org.eclipse.rdf4j.repository.RepositoryConnection;
import org.eclipse.rdf4j.query.TupleQuery;
import org.eclipse.rdf4j.query.TupleQueryResult;
import org.eclipse.rdf4j.query.BindingSet;
import org.eclipse.rdf4j.query.QueryLanguage;
import org.eclipse.rdf4j.model.Value;

public class App
{
    public static void main( String[] args )
    {
        String sparqlEndpoint = "https://your-neptune-endpoint:port/sparql";
        Repository repo = new SPARQLRepository(sparqlEndpoint);
        repo.initialize();

        try (RepositoryConnection conn = repo.getConnection()) {
           String queryString = "SELECT ?s ?p ?o WHERE { ?s ?p ?o } limit 10";

           TupleQuery tupleQuery = conn.prepareTupleQuery(QueryLanguage.SPARQL,
 queryString);

           try (TupleQueryResult result = tupleQuery.evaluate()) {
              while (result.hasNext()) {  // iterate over the result
                   BindingSet bindingSet = result.next();

                   Value s = bindingSet.getValue("s");
                   Value p = bindingSet.getValue("p");
                   Value o = bindingSet.getValue("o");

                   System.out.print(s);
                   System.out.print("\t");
                   System.out.print(p);
                   System.out.print("\t");
                   System.out.println(o);
              }
           }
        }
    }
```

```
}
```

10. 次の Maven コマンドを使用してサンプルをコンパイルおよび実行します。

```
mvn compile exec:java
```

前述の例では、10 の制限がある `?s ?p ?o` クエリを使用して、グラフのトリプル (主語 - 述語 - 目的語) のうち最大 10 個を返します。その他の対象にクエリを実行するには、クエリを別の SPARQL クエリで置き換えます。

この例では、結果の反復は返された各変数の値を出力します。`Value` オブジェクトは `String` に変換され、その後出力されます。クエリの `SELECT` 部分を変更した場合は、コードを変更する必要があります。

SPARQL HTTP API

SPARQL HTTP リクエストは、次のエンドポイントで受け付けられます。`https://`*`your-neptune-endpoint`*`:`*`port`*`/sparql`

SPARQL を使用した Amazon Neptune への接続の詳細については、SPARQL を使用した Neptune グラフへのアクセスを参照してください。

SPARQL プロトコルとクエリ言語の詳細については、「SPARQL 1.1 プロトコル」および「SPARQL 1.1 クエリ言語」仕様を参照してください。

以下のトピックでは、SPARQL RDF シリアル化形式、および Neptune で SPARQL HTTP API を使用する方法について説明します。

目次

- SPARQL UPDATE UNLOAD を使用して Neptune からデータを削除する

HTTP REST エンドポイントを使用して Neptune DB インスタンスに接続する

Amazon Neptune では、SPARQL クエリ用の HTTP エンドポイントが用意されています。REST インターフェイスは、SPARQL バージョン 1.1 と互換性があります。

> **Important**
>
> リリース： 1.0.4.0 (2020-10-12) は、Amazon Neptune へのすべての接続で TLS 1.2 と HTTPS を必須にしました。セキュリティで保護されていない HTTP や、1.2 より前のバージョンの TLS で HTTPS を使用して Neptune に接続することはできなくなりました。

次の手順は、curl コマンドを使用し、HTTPS を介した接続で HTTP 構文を使用して SPARQL エンドポイントに接続する方法について説明します。Neptune DB インスタンスと同じ仮想プライベートクラウド (VPC) の Amazon EC2 インスタンスからこれらの手順を実行してください。

Neptune DB インスタンスへの SPARQL クエリ用の HTTP エンドポイントは `https://`*`your-neptune-endpoint`*`:`*`port`*`/sparql` です。

> **Note**
>
> Neptune DB インスタンスのホスト名を見つける方法については、Amazon Neptune エンドポイントに接続する セクションを参照してください。

HTTP POST を使用したクエリ

次の例では、curl を使用して、HTTP POST を通じて SPARQL **`QUERY`** を送信します。

```
curl -X POST --data-binary 'query=select ?s ?p ?o where {?s ?p ?o} limit 10'
 https://your-neptune-endpoint:port/sparql
```

前述の例では、10 の制限がある `?s ?p ?o` クエリを使用して、グラフのトリプル (主語 - 述語 - 目的語) のうち最大 10 個を返します。その他の対象にクエリを実行するには、別の SPARQL クエリで置き換えます。

> **Note**
>
> SELECT および ASK クエリの場合、レスポンスのデフォルトの MIME メディアタイプは application/sparql-results+json です。
> CONSTRUCT および DESCRIBE クエリの場合、レスポンスのデフォルトの MIME タイプは application/n-quads です。
> Neptune によってシリアル化に使用されるメディアタイプのリストについては、Neptune SPARQL で使用される RDF シリアル化形式を参照してください。

HTTP POST を使用した更新

次の例では、curl を使用して、HTTP POST を通じて SPARQL **UPDATE** を送信します。

```
curl -X POST --data-binary 'update=INSERT DATA { <https://test.com/s> <https://
test.com/p> <https://test.com/o> . }' https://your-neptune-endpoint:port/sparql
```

前述の例では、次のトリプルを SPARQL デフォルトのグラフに挿入します。<https://test.com/s> <https://test.com/p> <https://test.com/o>

マルチパートの SPARQL レスポンスのオプションの HTTP 末尾ヘッダー

> **Note**
>
> この機能は、Neptune エンジンリリース 1.0.3.0 からアクセスできます。

SPARQL クエリと更新に対する HTTP 応答は、多くの場合、複数のパートまたはチャンクで返されます。クエリまたは更新がこれらのチャンクの送信を開始した後に発生する障害を診断するのは難しい場合があります。特に、最初のチャンクは HTTP ステータスコードが 200 であるからです。

末尾のヘッダーを明示的にリクエストしない限り、Neptune は、エラーメッセージをメッセージ本文に追加することによってのみ、そのような障害を報告しますが、通常これは破損しています。

この種の問題の検出と診断を容易にするために、転送エンコーディング (TE) トレーラーヘッダー (te: trailers) をリクエストに入れることができます (例としてTE リクエストヘッダーに関する MDN ページを参照)。これを行うと、Neptune はレスポンスチャンクの末尾ヘッダー内に 2 つの新しいヘッダーフィールドを含めます。

- `X-Neptune-Status`— 応答コードの後ろに短い名前が続きます。たとえば、成功した場合、末尾ヘッダーは次のようになります。`X-Neptune-Status: 200 OK`。失敗の場合、応答コードは、`X-Neptune-Status: 500 TimeLimitExceededException` といった Neptune エンジンのエラーコードとなる可能性があります。
- `X-Neptune-Detail`— 成功したリクエストでは空です。エラーの場合は、JSON エラーメッセージが含まれます。HTTP ヘッダー値には ASCII 文字しか使用できないため、JSON 文字列は URL 符号化されます。エラーメッセージは、レスポンスメッセージ本文にも追加されます。

Neptune の SPARQL で使用される RDF メディアタイプ。

Resource Description Framework (RDF) データはさまざまな方法でシリアル化することができ、そのほとんどは SPARQL が消費または出力することができます。

Neptune SPARQL で使用される RDF シリアル化形式

- RDF/XML – RDF の XML シリアル化。RDF 1.1 XML 構文で定義されています。メディアタイプ: `application/rdf+xml`。一般的なファイル拡張子: `.rdf`。
- N-Triples – RDF グラフをエンコードするための行ベースのプレーンテキスト形式。RDF 1.1 N-Triples で定義されています。メディアタイプ: `application/n-triples`、`text/turtle`、または `text/plain`。。一般的なファイル拡張子: `.nt`。
- N-Quads – グラフをエンコードするための行ベースのプレーンテキスト形式。RDF 1.1 N-Quads で定義されています。これは N-Triples の拡張子です。メディアタイプ: `application/n-quads`、または 7 ビットの US-ASCII でエンコードされている場合は `text/x-nquads`。一般的なファイル拡張子: `.nq`。
- Turtle – RDF 1.1 Turtle で定義されている RDF のテキスト構文で、RDF グラフをコンパクトで自然なテキスト形式で、一般的な使用パターンとデータ型の省略形で完全に記述できるようにします。Turtle は、N-Triples 形式および SPARQL の 3 つのパターン構文との互換性を提供します。メディアタイプ: `text/turtle`一般的なファイル形式: `.ttl`。
- TriG – RDF 1.1 TriG で定義されている RDF のテキスト構文で、RDF グラフをコンパクトで自然なテキスト形式で、一般的な使用パターンとデータ型の省略形で完全に記述できるようにします。TriG は Turtle 形式の拡張子です。メディアタイプ: `application/trig`。一般的なファイル拡張子: `.trig`。
- N3 (Notation3) - Notation3 (N3): 読み取り可能な RDF 構文 で定義されているアサーションと口ジック構文。N3 は、式 (グラフ自体であるリテラル)、変数、論理的意味、および機能的述部を追加することによって RDF データモデルを拡張し、RDF/XML に代わるテキスト形式の構文を提供します。メディアタイプ: `text/n3`。一般的なファイル拡張子: `.n3`。

- JSON-LD – JSON-LD 1.0で定義されているデータシリアル化およびメッセージング形式。メディアタイプ: `application/ld+json`。一般的なファイル拡張子: `.jsonld`。
- TriXX - TriX: XML での RDF トリプル で定義されている、XML での RDF のシリアル化。メディアタイプ: `application/trix`。一般的なファイル拡張子: `.trix`。
- SPARQL JSON 結果 – SPARQL 1.1 クエリ結果 JSON 形式を使用した、RDF のシリアル化。メディアタイプ: `application/sparql-results+json`。一般的なファイル拡張子: `.srj`。
- RDF4J バイナリ形式 - RDF4J バイナリ RDF 形式 で説明されている、RDF データをエンコードするためのバイナリ形式。メディアタイプ: `application/x-binary-rdf`。

Neptune SPARQL で使用される SPARQL 結果のシリアル化形式

- SPARQL XML 結果 – SPARQL クエリ結果 XML 形式 (Second Edition) で定義されている、SPARQL クエリ言語によって提供される変数バインディングおよびブール結果形式の XML 形式。メディアタイプ: `application/sparql-results+xml`。一般的なファイル拡張子: `.srx`。
- SPARQL CSVとTSVの結果 — カンマ区切り値とタブ区切り値を使用して、SPARQL 1.1 クエリ結果 CSV および TSV 形式で定義される `SELECT` クエリから SPARQL クエリの結果を表現します。メディアタイプ: カンマ区切り値の `text/csv`、およびタブ区切り値の `text/tab-separated-values`。一般的なファイル拡張子: カンマ区切り値の `.csv`、およびタブ区切り値の `.tsv`。
- バイナリ結果テーブル – SPARQL クエリの出力をエンコードするためのバイナリ形式。メディアタイプ: `application/x-binary-rdf-results-table`。
- SPARQL JSON 結果 – SPARQL 1.1 クエリ結果 JSON 形式を使用した、RDF のシリアル化。メディアタイプ: `application/sparql-results+json`。

Neptune が RDF データのインポートに使用できるメディアタイプ

Neptune bulk-loader によってサポートされるメディアタイプ

- N-Triples
- N-Quads
- RDF/XML
- Turtle

SPARQL UPDATE LOAD がインポートできるメディアタイプ

- N-Triples

- N-Quads
- RDF/XML
- Turtle
- TriG
- N3
- JSON-LD

Neptune がクエリ結果のエクスポートに使用できるメディアタイプ

SPARQL クエリ応答の出力形式を指定するには、クエリリクエストともに "`Accept: media-type`" ヘッダーを送信します。例:

```
curl -H "Accept: application/nquads" ...
```

SPARQL SELECT が Neptune から出力できる RDF メディアタイプ

- SPARQL JSON 結果 (これはデフォルトです)
- SPARQL XML 結果
- バイナリ結果テーブル (メディアタイプ: `application/x-binary-rdf-results-table`)
- カンマ区切り値 (CSV)
- タブ区切り値 (TSV)

SPARQL ASK が Neptune から出力できる RDF メディアタイプ

- SPARQL JSON 結果 (これはデフォルトです)
- SPARQL XML 結果
- Boolean (メディアタイプ: `text/boolean`、「true」または「false」を意味します)

SPARQL CONSTRUCT が Neptune から出力できる RDF メディアタイプ

- N-Quads (これはデフォルトです)
- RDF/XML
- JSON-LD
- N-Triples

- Turtle
- N3
- TriX
- TriG
- SPARQL JSON 結果
- RDF4J バイナリ RDF形式

SPARQL DESCRIBE が Neptune から出力できる RDF メディアタイプ

- N-Quads (これはデフォルトです)
- RDF/XML
- JSON-LD
- N-Triples
- Turtle
- N3
- TriX
- TriG
- SPARQL JSON 結果
- RDF4J バイナリ RDF形式

SPARQL UPDATE LOAD を使用して Neptune にデータをインポートする

SPARQL UPDATE LOAD コマンドの構文は、SPARQL 1.1 アップデートの推奨事項に記載されています。

```
LOAD SILENT (URL of data to be loaded) INTO GRAPH (named graph into which to load the
 data)
```

- **SILENT** — (オプション) 処理中にエラーが発生した場合でも、オペレーションは成功を返します。

  これは、単一のトランザクションに "LOAD ...; LOAD ...; UNLOAD ...; LOAD ...;" のような複数のステートメントが含まれている場合に便利です。また、リモートデータの一部が処理できない場合でも、トランザクションを完了させる必要があります。

- *######### URL* — (必須) グラフにロードするデータを含むリモートデータファイルを指定します。

  リモートファイルには、次のいずれかの拡張子を指定する必要があります。

  - NTriples 用 `.nt`。
  - NQuads 用 `.nq`。
  - Trig 用 `.trig`。
  - RDF/XML 用 `.rdf`。
  - Turtle 用 `.ttl`。
  - N3 用 `.n3`。
  - JSON-LD 用 `.jsonld`。
- **INTO GRAPH***(################)* — (オプション) データをロードするグラフを指定します。

  Neptune はすべてのトリプルを名前が付いたグラフに関連付けます。フォールバック名前付きグラフ URI、 `http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph`などを使用して、デフォルトの名前付きグラフを指定できます。次のようになります。

```
INTO GRAPH <http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph>
```

> **Note**
>
> 大量のデータをロードする必要がある場合は、UPDATE LOAD ではなく Neptune バルクローダーを使用することをお勧めします。バルクローダーについては、Amazon Neptune 一括ローダーを使用したデータの取り込みを参照してください。

SPARQL UPDATE LOAD を使用して、 Amazon S3 から直接データをロードすることも、セルフホストウェブサーバーから取得したファイルからデータをロードすることもできます。ロードするリソースは Neptune サーバーと同じリージョンに存在し、リソースのエンドポイントは VPC でホワイトリストに登録されている必要があります。Amazon S3 エンドポイントを作成する方法については、Amazon S3 VPC エンドポイントの作成 を参照してください。

すべてのSPARQL UPDATE LOAD URI は `https://` で始まる必要があります。これには Amazon S3 URL が含まれます。

バルクローダーとは対照的に、SPARQL UPDATE LOAD への呼び出しは完全にトランザクション型です。

SPARQL UPDATE LOAD を使用して Amazon S3 から Neptune にファイルを直接ロードする

Neptune では、SPARQL UPDATE LOAD を使用するときに Amazon S3 に IAM ロールを渡すことはできないため、問題の Amazon S3 バケットはパブリックであるか、LOAD クエリで署名付き Amazon S3 URL を使用する必要があります。

Amazon S3 ファイルの署名付き URL を生成するには、次のような AWS CLI コマンドを使用できます。

```
aws s3 presign --expires-in (number of seconds) s3://(bucket name)/(path to file of
 data to load)
```

次に、結果の署名付き URL を LOAD コマンドで使えます。

```
curl https://(a Neptune endpoint URL):8182/sparql \
  --data-urlencode 'update=load (pre-signed URL of the remote Amazon S3 file of data to
 be loaded) \
                           into graph (named graph)'
```

詳細については、「リクエストの認証: クエリパラメータの使用」を参照してください。Boto3 のドキュメントに、Python スクリプトを使用して署名付き URL を生成する方法が記載されています。

また、ロードするファイルのコンテンツタイプも正しく設定する必要があります。

1. 次のように -metadata パラメータを使用してファイルを Amazon S3 にアップロードするときにファイルのコンテンツタイプを設定します。

   ```
   aws s3 cp test.nt s3://bucket-name/my-plain-text-input/test.nt --metadata Content-
   Type=text/plain
   aws s3 cp test.rdf s3://bucket-name/my-rdf-input/test.rdf --metadata Content-
   Type=application/rdf+xml
   ```

2. メディアタイプの情報が実際に存在することを確認してください。以下を実行します:

   ```
   curl -v bucket-name/folder-name
   ```

   このコマンドの出力には、ファイルをアップロードするときに設定したメディアタイプ情報が表示されます。

3. その後、SPARQL UPDATE LOAD コマンドを使用してこれらのファイルを Neptune にインポートできます。

```
curl https://your-neptune-endpoint:port/sparql \
  -d "update=LOAD <https://s3.amazonaws.com/bucket-name/my-rdf-input/test.rdf>"
```

上記の手順は、パブリック バケット、または LOAD クエリで 署名付き Amazon S3 URL を使用してアクセスするバケットに対してのみ機能します。

以下に示すように、プライベート Amazon S3 バケットからロードするようにウェブプロキシサーバーを設定することもできます。

ウェブサーバーを使用して SPARQL UPDATE LOAD でファイルを Neptune にロードする

1. Neptune とロードするファイルをホストしている VPC 内で実行されているマシンにウェブサーバーをインストールします。たとえば、Amazon Linux を使用している場合は、次のように Apache をインストールします。

```
sudo yum install httpd mod_ssl
sudo /usr/sbin/apachectl start
```

2. ロードしようとしている RDF ファイルコンテンツの MIME タイプを定義します。SPARQL はウェブサーバーから送信された `Content-type` ヘッダーを使用してコンテンツの入力形式を決定するため、ウェブサーバーに関連する MIME タイプを定義する必要があります。

   たとえば、ファイル形式を識別するために次のファイル拡張子を使用するとします。

   - NTriples 用 `.nt`。
   - NQuads 用 `.nq`。
   - Trig 用 `.trig`。
   - RDF/XML 用 `.rdf`。
   - Turtle 用 `.ttl`。
   - N3 用 `.n3`。
   - JSON-LD 用 `.jsonld`。

   ウェブサーバーとして Apache 2 を使用している場合は、ファイル `/etc/mime.types` を編集して次のタイプを追加します。

```
text/plain nt
application/n-quads nq
application/trig trig
application/rdf+xml rdf
application/x-turtle ttl
text/rdf+n3 n3
application/ld+json jsonld
```

3. MIME タイプマッピングが機能することを確認します。ウェブサーバーを起動して実行し、選択した形式で RDFファイルをホストしたら、ローカルホストからウェブサーバーにリクエストを送信して設定をテストできます。

   たとえば、次のようなリクエストを送信します。

   ```
   curl -v http://localhost:80/test.rdf
   ```

   curl からの詳細な出力には、このような行が表示されるはずです。

   ```
   Content-Type: application/rdf+xml
   ```

   これは、コンテンツタイプマッピングが正常に定義されたことを示しています。

4. これで、SPARQL UPDATE コマンドを使用してデータをロードする準備が整いました。

   ```
   curl https://your-neptune-endpoint:port/sparql \
       -d "update=LOAD <http://web_server_private_ip:80/test.rdf>"
   ```

> **Note**
>
> SPARQL UPDATE LOAD を使用すると、ロードされるソースファイルが大きい場合にウェブサーバーでタイムアウトが発生する可能性があります。Neptune は、ストリーミング時と、サーバーで設定したタイムアウトよりも時間がかかる大きなファイルでは、ファイルデータを処理します。これにより、サーバーが接続を閉じることがあります。その場合、Neptune がストリームで予期しない EOF を検出すると、次のエラーメッセージが表示されることがあります。
>
> ```
> {
>   "detailedMessage":"Invalid syntax in the specified file",
> ```

```
  "code":"InvalidParameterException"
}
```

このメッセージが表示され、ソースファイルに無効な構文が含まれていると思われる場合は、ウェブサーバーのタイムアウト設定を大きくしてみてください。さらに、サーバーでデバッグログを有効にし、タイムアウトを探すことで、問題を診断することもできます。

SPARQL UPDATE UNLOAD を使用して Neptune からデータを削除する

Neptune はカスタム SPARQL オペレーション、UNLOAD、も提供し、リモートソースで指定されたデータを削除する場合に使用します。UNLOAD は LOAD オペレーションの対応物と見なすことができます。構文は次のとおりです。

> **Note**
>
> この機能は、Neptune エンジンリリース 1.0.4.1 からアクセスできます。

```
UNLOAD SILENT (URL of the remote data to be unloaded) FROM GRAPH (named graph from
 which to remove the data)
```

- **SILENT** — (オプション) データ処理中にエラーが発生した場合でも、オペレーションは成功を返します。

  これは、単一のトランザクションに "LOAD ...; LOAD ...; UNLOAD ...; LOAD ...;" のような複数のステートメントが含まれている場合に便利です。また、リモートデータの一部が処理できない場合でも、トランザクションを完了させる必要があります。
- *############### URL* — (必須) グラフにロードするデータを含むリモートデータファイルを指定します。

  リモートファイルには、次のいずれかの拡張子が必要です (UPDATE-LOAD がサポートする形式と同じです)。
  - NTriples 用 `.nt`。
  - NQuads 用 `.nq`。
  - Trig 用 `.trig`。
  - RDF/XML 用 `.rdf`。

- Turtle 用 `.ttl`。
- N3 用 `.n3`。
- JSON-LD 用 `.jsonld`。

このファイルに含まれるすべてのデータは、UNLOAD オペレーションによって DB クラスターから削除されます。

データをアンロードするには、Amazon S3 認証はいずれも URL に含める必要があります。Amazon S3 ファイルに事前署名し、結果の URL を使用して安全にアクセスできます。例:

```
aws s3 presign --expires-in (number of seconds) s3://(bucket name)/(path to file of
 data to unload)
```

次に:

```
curl https://(a Neptune endpoint URL):8182/sparql \
  --data-urlencode 'update=unload (pre-signed URL of the remote Amazon S3 data to be
 unloaded) \
                            from graph (named graph)'
```

詳細については、「リクエストの認証: クエリパラメータの使用」を参照してください。

- **FROM GRAPH** *(###############)* — (オプション) リモートデータのアンロード元となる名前付きグラフを指定します。

  Neptune はすべてのトリプルを名前が付いたグラフに関連付けます。フォールバック名前付きグラフ URI、`http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph` などを使用して、デフォルトの名前付きグラフを指定できます。次のようになります。

  ```
  FROM GRAPH <http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph>
  ```

それと同じ方法で LOAD は INSERT DATA { *(inline data)* } に、UNLOAD は DELETE DATA { *(inline data)* } に対応しています。DELETE DATA 同様に、UNLOAD は空白ノードを含むデータでは機能しません。

たとえば、ローカル Web サーバーが、`data.nt` という名前のファイルを提供しているとします。これには、次の 2 つのトリプルが含まれています。

```
<http://example.org/resource#a> <http://example.org/resource#p> <http://example.org/
resource#b> .
<http://example.org/resource#a> <http://example.org/resource#p> <http://example.org/
resource#c> .
```

以下の UNLOAD コマンドは名前の付いたグラフ <http://example.org/graph1> から、これら 2 つのトリプルを削除します。

```
UNLOAD <http://localhost:80/data.nt> FROM GRAPH <http://example.org/graph1>
```

これは、以下の DELETE DATA コマンドを使用する場合と同じ結果になります。

```
DELETE DATA {
  GRAPH <http://example.org/graph1> {
    <http://example.org/resource#a> <http://example.org/resource#p> <http://
example.org/resource#b> .
    <http://example.org/resource#a> <http://example.org/resource#p> <http://
example.org/resource#c> .
  }
}
```

UNLOAD コマンドによってスローされる例外

- **InvalidParameterException** — データに空のノードがありました。HTTP status: 400 Bad Request。

  メッセージ `Blank nodes are not allowed for UNLOAD`

- **InvalidParameterException** — データ内に壊れた構文がありました。HTTP status: 400 Bad Request。

  メッセージ`Invalid syntax in the specified file.`

- **UnloadUrlAccessDeniedException** – アクセスが拒否されました。HTTP status: 400 Bad Request。

  メッセージ`Update failure: Endpoint` *`(Neptune endpoint)`* `reported access denied error. Please verify access.`

- **`BadRequestException`** — リモートデータを取得できません。HTTP status: 400 Bad Request。

  メッセージ:(HTTP レスポンスによって決まります)。

SPARQL クエリヒント

クエリヒントを使用して Amazon Neptune の特定の SPARQL クエリの最適化と評価戦略を指定することができます。

クエリヒントは、以下の部分で SPARQL クエリに組み込まれている追加の 3 つのパターンで表されます。

```
scope hint value
```

- scope - クエリ内の特定のグループまたは完全なクエリなど、クエリヒントが適用されるクエリの部分を決定します。
- hint – 適用するヒントのタイプを特定します。
- value – 検討中のシステムアスペクトの動作を決めます。

クエリのヒントとスコープは、Amazon Neptune 名前空間 `http://aws.amazon.com/neptune/vocab/v01/QueryHints#` の事前定義された条件として表示されます。このセクションの例には、クエリで定義され、クエリに含まれている `hint` プレフィックスとして名前空間が含まれています。

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
```

たとえば、以下は `SELECT` クエリに `joinOrder` ヒントを含める方法を示します。

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
SELECT ... {
 hint:Query hint:joinOrder "Ordered" .
 ...
}
```

前述のクエリは、Neptune エンジンに、クエリ内の結合を評価するように指示します。指定された順序付けを行い、自動並べ替えを無効にします。

クエリヒントを使用するときは、以下について検討します。

- 単一のクエリでさまざまなクエリヒントを組み合わせることができます。たとえば、ボトムアップ評価のためにサブクエリに注釈を追加するには bottomUp クエリヒントを使用できます。また、サブクエリ内の結合の順序を修正するには joinOrder クエリヒントを使用できます。
- 別の重複していないスコープで同じクエリを複数回使用できます。
- クエリヒントはヒントです。クエリエンジンは通常、特定のクエリヒントを考慮することを目的としていますが、それらを無視することもあります。
- クエリヒントはセマンティクスを維持します。クエリヒントを追加しても、クエリの出力は変更されません (順序保証が指定されていない場合、つまり、ORDER BY を使用して結果の順序が明示的に適用されない場合を除く)。

以下のセクションでは、Neptune で使用可能なクエリヒントとその使用方法に関する詳しい情報が記載されています。

トピック

Neptune における SPARQL クエリヒントの範囲。

次の表は、Amazon Neptune で使用可能なスコープ、関連付けられたヒント、SPARQL クエリヒントの説明を示します。これらのエントリの hint プレフィックスは、ヒントの Neptune 名前空間を表します。

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
```

スコープ	サポートされるヒント	説明
`hint:Query`	joinOrder	クエリヒントはクエリ全体に適用されます。
`hint:Query`	queryTimeout	タイムアウト値はクエリ全体に適用されます。
`hint:Query`	RangeSafe	タイプ昇格は、クエリ全体に対して無効になっています。
`hint:Query`	queryId	クエリ ID 値はクエリ全体に適用されます。
`hint:Query`	useDFE	DFE の使用は、クエリ全体に対して無効になっています。
`hint:Group`	joinOrder	クエリヒントは指定されたグループの最上位の要素に適用されますが、ネストされた要素 (サブクエリなど) や親要素には適用されません。
`hint:SubQuery`	evaluationStrategy	ヒントは指定されて、ネストされた SELECT サブクエリに適用されます。サブクエリの前に計算されたソリューションを考慮せずに、サブクエリは個別に評価されます。

`joinOrder` SPARQL クエリヒント

SPARQL クエリを送信すると、Amazon Neptune クエリエンジンはクエリの構造を調査します。クエリの各パートの順序を変更し、評価に必要な作業の量とクエリレスポンス時間を最小限にしようとします。

たとえば、一連の接続された 3 つのパターンは通常、指定された順序では評価されません。ヒューリスティックと統計 (個々のパターンの選択性と共有変数で接続される方法など) を使用して順序が

変更されます。さらに、サブクエリ、FILTER、複雑な OPTIONAL や MINUS ブロックなど、より複雑なパターンがクエリに含まれている場合、Neptune クエリエンジンは評価の順序を効率的にするために、可能な限りそれらの順序を変更します。

より複雑なクエリでは、Neptune で選択されるクエリの評価順序は常に最適であるとは限りません。たとえば、Neptune では、クエリ評価中に発生するインスタンスデータ固有の特性 (グラフでの power ノードの発生など) を見逃す可能性があります。

データの正確な特性がわかっていて、クエリ実行の順序を手動で指定する場合は、Neptune `joinOrder` クエリヒントを使用して特定の順序でクエリが評価されるように指定します。

`joinOrder` SPARQL ヒント構文

`joinOrder` クエリヒントは、SPARQL クエリに含まれる 3 つのパターンとして指定されます。

わかりやすくするために、次の構文では、クエリに定義されて含まれる `hint` プレフィックスを使用して Neptune クエリヒント名前空間を指定します。

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
scope hint:joinOrder "Ordered" .
```

利用可能なスコープ

- `hint:Query`
- `hint:Group`

クエリヒントスコープの詳細については、「Neptune における SPARQL クエリヒントの範囲。」を参照してください。

`joinOrder`SPARQL ヒントの例

このセクションでは、`joinOrder` クエリヒントを使用した場合と使用していない場合に記述されたクエリ、および関連する最適化を示します。

この例では、データセットに次のものが含まれていることを前提としています。

- `Jane` を含む 1,000 人に `:likes` エッジを持つ (「いいね！」した) `John` という名前の 1 人の人物。
- `John` を含む 10 人に `:likes` エッジを持つ (「いいね！」した) `Jane` という名前の 1 人の人物。

クエリヒントなし

次の SPARQL クエリは、一連のソーシャルネットワーキングデータから相互に「いいね！」した John と Jane という名前のすべての人物のペアを抽出します。

```
PREFIX : <https://example.com/>
SELECT ?john ?jane {
  ?person1 :name "Jane" .
  ?person1 :likes ?person2 .
  ?person2 :name "John" .
  ?person2 :likes ?person1 .
}
```

Neptune クエリエンジンは、記述とは異なる順序でステートメントを評価する可能性があります。たとえば、次の順序で評価するように選択される場合があります。

1. John という名前の人物をすべて検索します。
2. :likes エッジによって John に接続されているすべての人物を検索します。
3. Jane という名前の人物でこのセットをフィルタリングします。
4. :likes エッジによって John に接続されている人物でこのセットをフィルタリングします。

データセットによると、この順序で評価すると 2 番目のステップで 1,000 のエンティティが抽出されます。これは 3 番目のステップで 1 つのノード Jane に絞り込まれます。そして最後のステップで、Jane も John ノードに対して :likes エッジを持っていると判断されます。

クエリヒント

Jane ノードには発信 :likes エッジが 10 個しかないため、このノードから開始する方が有利です。これにより、2 番目のステップで 1,000 のエンティティを抽出することを回避し、クエリの評価中の作業量を低減できます。

次の例では joinOrder クエリヒントを使用し、クエリの自動結合の順序変更をすべて無効にすることで、Jane ノードとその発信エッジが最初に処理されるようにします。

```
PREFIX : <https://example.com/>
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
SELECT ?john ?jane {
  hint:Query hint:joinOrder "Ordered" .
  ?person1 :name "Jane" .
  ?person1 :likes ?person2 .
```

```
  ?person2 :name "John" .
  ?person2 :likes ?person1 .
}
```

応用できる現実社会のシナリオとしては、ネットワーク内の人物を「コネクションが多いインフルエンサー」や「コネクションが少ない通常のユーザー」に分類する、ソーシャルネットワークへの応用などが挙げられます。このようなシナリオでは、前述の例のようなクエリで、インフルエンサー (John) の前に通常のユーザー (Jane) が処理されるようになっていることを確認します。

クエリヒントと順序変更

この例をさらに一歩進めることができます。:name 属性が単一のノードに固有であることがわかっている場合は、joinOrder クエリヒントの順序を変更して使用することで、クエリの時間を短縮することができます。このステップでは、最初に一意のノードが抽出されるようにします。

```
PREFIX : <https://example.com/>
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
SELECT ?john ?jane {
  hint:Query hint:joinOrder "Ordered" .
  ?person1 :name "Jane" .
  ?person2 :name "John" .
  ?person1 :likes ?person2 .
  ?person2 :likes ?person1 .
}
```

この場合、各ステップで以下の単一のアクションへのクエリを減らすことができます。

1. :name Jane を持つ 1 人の人物のノードを検索します。
2. :name John を持つ 1 人の人物のノードを検索します。
3. 最初のノードが 2 番目のノードに :likes エッジで接続されていることを確認します。
4. 2 番目のノードが最初のノードに :likes エッジで接続されていることを確認します。

> **Important**
>
> 間違った順序を選択した場合、joinOrder クエリヒントのパフォーマンスが大幅に低下する可能性があります。たとえば、前述の例は :name 属性が一意でない場合、非効率的です。100 ノードすべての名前が Jane で 1,000 ノードすべての名前が John の場合、クエリで 1,000 * 100 (100,000) ペアの :likes エッジを確認することになります。

evaluationStrategySPARQL クエリヒント

evaluationStrategy クエリヒントは、注釈が付けられたクエリのフラグメントを独立したユニットとして下部から評価するように Amazon Neptune クエリエンジンに指示します。つまり、前回の評価ステップからのソリューションは、クエリフラグメントの計算に使用されません。クエリフラグメントはスタンドアロンユニットとして評価され、生成されたソリューションは計算された後にクエリの残りの部分と結合されます。

evaluationStrategy クエリヒントを使用することは、(パイプライン化されていない) クエリプランをブロックすることを意味します。つまり、クエリヒントによって注釈が付けられたフラグメントのソリューションは、メインメモリでマテリアライズおよびバッファされます。特に注釈が付けられたクエリフラグメントが大量の結果を計算する場合、このクエリヒントを使用することによって、クエリの評価に必要なメインメモリの量が大幅に増加する可能性があります。

evaluationStrategy SPARQL ヒント構文

evaluationStrategy クエリヒントは、SPARQL クエリに含まれる 3 つのパターンとして指定されます。

わかりやすくするために、次の構文では、クエリに定義されて含まれる hint プレフィックスを使用して Neptune クエリヒント名前空間を指定します。

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
hint:SubQuery hint:evaluationStrategy "BottomUp" .
```

利用可能なスコープ

- hint:SubQuery

> **Note**
>
> このクエリヒントは、ネストされたサブクエリでのみサポートされています。

クエリヒントスコープの詳細については、「Neptune における SPARQL クエリヒントの範囲。」を参照してください。

evaluationStrategySPARQL ヒントの例

このセクションでは、evaluationStrategy クエリヒントを使用した場合と使用していない場合に記述されたクエリ、および関連する最適化を示します。

この例では、データセットに次の特性があることを前提としています。

- 1,000 のエッジラベル :connectedTo が含まれています。
- 各 component ノードは、平均 100 の他の component ノードに接続されています。
- ノード間の 4 つのホップの周期的な接続の標準的な数は、約 100 です。

クエリヒントなし

次の SPARQL クエリは、4 つのホップを通じて周期的に相互に接続される component ノードをすべて抽出します。

```
PREFIX : <https://example.com/>
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
SELECT * {
  ?component1 :connectedTo ?component2 .
  ?component2 :connectedTo ?component3 .
  ?component3 :connectedTo ?component4 .
  ?component4 :connectedTo ?component1 .
}
```

Neptune クエリエンジンのアプローチは、次のステップを使用してこのクエリを評価することです。

- グラフ内の 1,000 の connectedTo エッジをすべて抽出します。
- 100 倍で展開します (100 は component2 からの発信 connectedTo エッジの数)。

  中間結果: 100,000 ノード。
- 100 倍で展開します (100 は component3 からの発信 connectedTo エッジの数)。

  中間結果: 10,000,000 ノード。
- サイクルを閉じるために 10,000,000 ノードをスキャンします。

この結果、一定量のメインメモリを持つクエリプランがストリーミングされます。

クエリヒントとサブクエリ

メインメモリスペースと高速コンピューティングのトレードオフを実現できます。evaluationStrategy クエリヒントを使用してクエリを書き換えることで、2 つの小さなマテリアライズされたサブセット間の結合をエンジンに計算させることができます。

```
PREFIX : <https://example.com/>
          PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
SELECT * {
  {
    SELECT * WHERE {
      hint:SubQuery hint:evaluationStrategy "BottomUp" .
      ?component1 :connectedTo ?component2 .
      ?component2 :connectedTo ?component3 .
    }
  }
  {
    SELECT * WHERE {
      hint:SubQuery hint:evaluationStrategy "BottomUp" .
      ?component3 :connectedTo ?component4 .
      ?component4 :connectedTo ?component1 .
    }
  }
}
```

今後のパターンの入力として、前の 3 つのパターンの結果を反復的に使用しながら 3 つのパターンを順番に評価する代わりに、evaluationStrategy ヒントを使用すると、2 つのサブクエリを個別に評価することができます。この 2 つのサブクエリは、中間結果で 100,000 のノードを生成します。これは、最終出力を生成するために結合されます。

特に、より大規模なインスタンスタイプで Neptune を実行するとき、メインメモリにこれら 2 つの 100,000 サブセットを一時的に保管することにより、評価の時間を大幅に短縮するのと引き換えにメモリ使用量が増加します。

queryTimeout SPARQL クエリヒント

queryTimeout クエリヒントは、DB パラメータグループに設定されている neptune_query_timeout 値より短いタイムアウトを指定します。

このヒントの結果としてクエリが終了すると、 Operation terminated (deadline exceeded) メッセージとともに TimeLimitExceededException がスローされます。

queryTimeout SPARQL ヒント構文

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
SELECT ... WHERE {
    hint:Query hint:queryTimeout 10 .
    # OR
    hint:Query hint:queryTimeout "10" .
    # OR
    hint:Query hint:queryTimeout "10"^^xsd:integer .
 ...
}
```

タイムアウト値はミリ秒単位で表されます。

タイムアウト値は、DB neptune_query_timeout パラメーターグループで設定された値より小さくする必要があります。それ以外の場合は、`Malformed query: Query hint 'queryTimeout' must be less than neptune_query_timeout DB Parameter Group` メッセージとともに MalformedQueryException 例外がスローされます。

queryTimeout クエリヒントは、メインクエリの WHERE 句、または次の例に示すようにいずれかのサブクエリの WHERE 句に指定する必要があります。

すべてのクエリ / サブクエリおよび SPARQL 更新セクション (INSERT、DELETE など) で 1 回のみ設定する必要があります。それ以外の場合は、`Malformed query: Query hint 'queryTimeout' must be set only once` メッセージとともに MalformedQueryException 例外がスローされます。

利用可能なスコープ

queryTimeout ヒントは、SPARQL クエリと更新の両方に適用できます。

- SPARQL クエリでは、メインクエリまたはサブクエリの WHERE 句に表示されます。
- SPARQL 更新では、INSERT、DELETE、または WHERE 句で設定できます。複数の更新句がある場合は、そのうちの 1 つにのみ設定できます。

クエリヒントスコープの詳細については、「Neptune における SPARQL クエリヒントの範囲。」を参照してください。

queryTimeoutSPARQL ヒントの例

UPDATE クエリのメイン WHERE 句で hint:queryTimeout を使用する例を示します。

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
INSERT {
    ?s ?p ?o
} WHERE {
    hint:Query hint:queryTimeout 100 .
    ?s ?p ?o .
}
```

ここで、hint:queryTimeout はサブクエリの WHERE 句にあります。

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
SELECT * {
   ?s ?p ?o .
   {
      SELECT ?s WHERE {
         hint:Query hint:queryTimeout 100 .
         ?s ?p1 ?o1 .
      }
   }
}
```

rangeSafe SPARQL クエリヒント

SPARQL クエリのタイププロモーションをオフにするには、このクエリヒントを使用します。

数値または範囲を越える FILTER を含む SPARQL クエリを送信する場合、Neptune クエリエンジンは通常、クエリの実行時に型の上位変換を使用する必要があります。つまり、フィルタリングする値を保持できるすべてのタイプの値を調べる必要があります。

たとえば、55 に等しい値をフィルタリングする場合、エンジンは 55 に等しい整数、55L に等しい長整数、55.0 に等しい浮動小数点数などを探す必要があります。各型の上位変換では、ストレージに対する追加のルックアップが必要なため、単純なクエリの完了に予期せず長い時間がかかることがあります。

多くの場合、特定のタイプの値を見つけるだけで済むことが事前にわかっているため、型の上位変換は不要です。この場合、クエリを劇的に高速化するには、型の上位変換をオフにする rangeSafe クエリヒントを使います。

rangeSafe SPARQL ヒント構文

rangeSafe クエリヒントは、タイププロモーションをオフにする true の値を取ります。また、false (デフォルト) の値も受け入れます。

例。次の例は、1 より大きい o の整数値のフィルタリング時にタイプ昇格をオフにする方法を示しています。

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
SELECT * {
   ?s ?p ?o .
   hint:Prior hint:rangeSafe 'true' .
   FILTER (?o > '1'^^<http://www.w3.org/2001/XMLSchema#int>)
```

queryIdSPARQL クエリヒント

このクエリヒントを使用して、独自の queryId 値を SPARQL クエリに割り当てます。

例：

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
SELECT * WHERE {
  hint:Query hint:queryId "4d5c4fae-aa30-41cf-9e1f-91e6b7dd6f47"
  {?s ?p ?o}}
```

割り当てる値は、Neptune DB 内のすべてのクエリで一意である必要があります。

useDFE SPARQL クエリヒント

このクエリヒントを使用して、クエリを実行するための DFE の使用を有効にします。neptune_dfe_query_engine インスタンスパラメータはデフォルトでは viaQueryHint に設定されるため、デフォルトでは、このクエリヒントが true に設定されていないと、Neptune は DFE を使用しません。このインスタンスパラメータを enabled に設定した場合、useDFE クエリヒントが false に設定されているクエリを除き、すべてのクエリに DFE エンジンが使用されます。

DFE の使用をクエリに使用できるようにする例

```
PREFIX : <https://example.com/>
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
```

```
SELECT ?john ?jane
{
  hint:Query hint:useDFE true .
  ?person1 :name "Jane" .
  ?person1 :likes ?person2 .
  ?person2 :name "John" .
  ?person2 :likes ?person1 .
}
```

DESCRIBE で使用される SPARQL クエリヒント

SPARQL DESCRIBE クエリは、リソースの説明をリクエストするための柔軟なメカニズムを提供します。ただし、SPARQL 仕様では、DESCRIBE の正確なセマンティクスは定義されていません。

エンジンリリース 1.2.0.2 以降、Neptune はさまざまな状況に適したいくつかの異なる DESCRIBE モードとアルゴリズムをサポートしています。

このサンプルデータセットは、さまざまなモードを説明するのに役立ちます。

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix : <https://example.com/> .

:JaneDoe :firstName "Jane" .
:JaneDoe :knows :JohnDoe .
:JohnDoe :firstName "John" .
:JaneDoe :knows _:b1 .
_:b1 :knows :RichardRoe .

:RichardRoe :knows :JaneDoe .
:RichardRoe :firstName "Richard" .

_:s1 rdf:type rdf:Statement .
_:s1 rdf:subject :JaneDoe .
_:s1 rdf:predicate :knows .
_:s1 rdf:object :JohnDoe .
_:s1 :knowsFrom "Berlin" .

:ref_s2 rdf:type rdf:Statement .
:ref_s2 rdf:subject :JaneDoe .
:ref_s2 rdf:predicate :knows .
:ref_s2 rdf:object :JohnDoe .
:ref_s2 :knowsSince 1988 .
```

以下の例では、以下のような SPARQL クエリを使用してリソース :JaneDoe の説明が要求されていることを前提としています。

```
DESCRIBE <https://example.com/JaneDoe>
```

describeMode SPARQL クエリヒント

hint:describeMode SPARQL クエリヒントは、Neptune によってサポートされる次の SPARQL DESCRIBE モードのいずれかを選択するために使用されます。

ForwardOneStep DESCRIBE モード

次のような describeMode クエリヒントで ForwardOneStep モードを呼び出します。

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
DESCRIBE <https://example.com/JaneDoe>
{
  hint:Query hint:describeMode "ForwardOneStep"
}
```

ForwardOneStep モードは、記述されるリソースの属性と転送リンクのみを返します。この例では、これは、記述されるリソース :JaneDoe をサブジェクトとして持つトリプルを返すことを意味します。

```
:JaneDoe :firstName "Jane" .
:JaneDoe :knows :JohnDoe .
:JaneDoe :knows _:b301990159 .
```

DESCRIBE クエリは、入力データセットと比較して、_:b301990159 など、毎回異なる ID を持つ空白のノードのトリプルを返す場合があることに注意してください。

SymmetricOneStep DESCRIBE モード

SymmetricOneStep は、クエリヒントを指定しなかった場合のデフォルトの DESCRIBE モードです。また、次のような describeMode クエリヒントを使って明示的に呼び出すこともできます。

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
DESCRIBE <https://example.com/JaneDoe>
{
```

```
  hint:Query hint:describeMode "SymmetricOneStep"
}
```

SymmetricOneStep セマンティクスでは、DESCRIBE は、記述されるリソースの属性、送信リンク、逆リンクを返します。

```
:JaneDoe :firstName "Jane" .
:JaneDoe :knows :JohnDoe .
:JaneDoe :knows _:b318767375 .

_:b318767631 rdf:subject :JaneDoe .

:RichardRoe :knows :JaneDoe .

:ref_s2 rdf:subject :JaneDoe .
```

Concise Bounded Description (**CBD**) DESCRIBE モード

Concise Bounded Description (CBD) モードは、次のような describeMode クエリヒントを使用して呼び出されます。

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
DESCRIBE <https://example.com/JaneDoe>
{
  hint:Query hint:describeMode "CBD"
}
```

CBD セマンティクスでは、DESCRIBE は、記述されるリソースの Concise Bounded Description (W3C によって定義) を返します。

```
:JaneDoe :firstName "Jane" .
:JaneDoe :knows :JohnDoe .
:JaneDoe :knows _:b285212943 .
_:b285212943 :knows :RichardRoe .

_:b285213199 rdf:subject :JaneDoe .
_:b285213199 rdf:type rdf:Statement .
_:b285213199 rdf:predicate :knows .
_:b285213199 rdf:object :JohnDoe .
_:b285213199 :knowsFrom "Berlin" .
```

```
:ref_s2 rdf:subject :JaneDoe .
```

RDF リソース (つまり RDF グラフ内のノード) の Concise Bounded Description は、そのノードを中心として独立できる最小のサブグラフです。実際には、このグラフを、指定されたノードをルートとするツリーと考えると、そのツリーの葉のような空白のノード (bnode) は存在しないということです。bnode は外部からアドレス指定することも、後続のクエリで使用することもできないため、現在のノードから次のシングルホップを見つけるには、グラフをブラウズするだけでは不十分です。また、後続のクエリで使用できるもの (つまり、bnode 以外のもの) を見つけるにも、十分に調査する必要があります。

CBD の計算

ソース RDF グラフ内の特定のノード (開始ノードまたはルート) が指定されると、そのノードの CBD は次のように計算されます。

1. ステートメントのサブジェクトが開始ノードであるソースグラフ内のすべてのステートメントをサブグラフに含めます。
2. 再帰的に、サブグラフ内のこれまでに空白のノードオブジェクトを持つすべてのステートメントについて、ソースグラフ内の、ステートメントのサブジェクトがその空白のノードであり、サブグラフにまだ含まれていないすべてのステートメントをサブグラフに含めます。
3. 再帰的に、それまでにサブグラフに含まれていたすべてのステートメントについて、ソースグラフ内のこれらのステートメントのすべての具象化について、各具象化の `rdf:Statement` ノードから始まる CBD を含めます。

その結果、オブジェクトノードが IRI 参照またはリテラルのいずれかであるサブグラフ、または空白のノードがグラフ内のどのステートメントのサブジェクトにもなっていないサブグラフになります。CBD は、単一の SPARQL SELECT または CONSTRUCT クエリーでは計算できないことに注意してください。

Symmetric Concise Bounded Description (**SCBD**) DESCRIBE モード

Symmetric Concise Bounded Description (SCBD) モードは、次のような `describeMode` クエリヒントを使用して呼び出されます。

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
DESCRIBE <https://example.com/JaneDoe>
{
  hint:Query hint:describeMode "SCBD"
```

```
}
```

SCBD セマンティクスでは、DESCRIBE は、リソースの Symmetric Concise Bounded Description (W3C によって「リンクされたデータセットを VoID ボキャブラリで記述する」で定義) を返します 。

```
:JaneDoe :firstName "Jane" .
:JaneDoe :knows :JohnDoe .
:JaneDoe :knows _:b335544591 .
_:b335544591 :knows :RichardRoe .

:RichardRoe :knows :JaneDoe .

_:b335544847 rdf:subject :JaneDoe .
_:b335544847 rdf:type rdf:Statement .
_:b335544847 rdf:predicate :knows .
_:b335544847 rdf:object :JohnDoe .
_:b335544847 :knowsFrom "Berlin" .

:ref_s2 rdf:subject :JaneDoe .
```

CBD と SCBD が ForwardOneStep および SymmetricOneStep モードより優れている点は、空白のノードが常にその表現を含むように拡張される点です。SPARQL を使用して空白のノードをクエリすることはできないため、これは重要な利点かもしれません。さらに、CBD モードと SCBD モードでは具象化も考慮されます。

describeMode クエリヒントは WHERE 句の一部にもなることに注意してください。

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
DESCRIBE ?s
WHERE {
  hint:Query hint:describeMode "CBD" .
  ?s rdf:type <https://example.com/Person>
}
```

describeIterationLimit SPARQL クエリヒント

hint:describeIterationLimit SPARQL クエリヒントは、CBD や SCBD などの反復的な DESCRIBE アルゴリズムで実行される反復拡張の最大回数に関するオプション制約となります。

DESCRIBE 制限は AND 処理されます。したがって、反復制限とステートメント制限の両方が指定された場合、DESCRIBE クエリを切断する前に両方の制限を満たす必要があります。

この値のデフォルトは 5 です。ゼロ (0) に設定すると、反復拡張の回数に制限がないように指定できます。

`describeStatementLimit` SPARQL クエリヒント

`hint:describeStatementLimit` SPARQL クエリヒントは、DESCRIBE クエリレスポンスに含めることができるステートメントの最大数をオプションで制限できます。CBD や SCBD のような反復的な DESCRIBE アルゴリズムにのみ適用されます。

DESCRIBE 制限は AND 処理されます。したがって、反復制限とステートメント制限の両方が指定された場合、DESCRIBE クエリを切断する前に両方の制限を満たす必要があります。

この値のデフォルトは 5000 です。ゼロ (0) に設定すると、返されるステートメントの数に制限がないように指定できます。

SPARQL DESCRIBE のデフォルトグラフに対する動作

SPARQL `DESCRIBE` クエリフォームを使用すると、データの構造を知らなくても、またクエリを作成しなくても、リソースに関する情報を取得できます。この情報をどのように組み立てるかは、SPARQL の実装に任されています。Neptune には、`DESCRIBE` が使用できるさまざまなモードやアルゴリズムを呼び出すクエリヒントがいくつか用意されています。

Neptune の実装では、モードに関係なく、`DESCRIBE` は SPARQL デフォルトグラフにあるデータのみを使用します。これは SPARQL がデータセットを扱う方法と一致しています (SPARQL 仕様の「RDF データセットの指定」を参照) 。

Neptune では、`FROM` や `FROM NAMED` 句を使用して特定の名前付きグラフが指定されない限り、デフォルトのグラフには、データベース内のすべての名前付きグラフの和集合に含まれるすべてのユニークなトリプルが含まれます。Neptune のすべての RDF データは、名前付きのグラフに保存されます。名前付きグラフのコンテキストなしでトリプルが挿入された場合、Neptune は `http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph` という名前付きグラフに格納します。

`FROM` 句を使用して 1 つ以上の名前付きグラフが指定された場合、デフォルトのグラフは、それらの名前付きグラフに含まれるすべてのユニークなトリプルを結合したグラフになります。`FROM` 句がなく、`FROM NAMED` 句が 1 つ以上ある場合、デフォルトのグラフは空になります。

SPARQL **`DESCRIBE`** の例

以下のデータを考慮します。

```
PREFIX ex: <https://example.com/>

GRAPH ex:g1 {
    ex:s ex:p1 "a" .
    ex:s ex:p2 "c" .
}

GRAPH ex:g2 {
    ex:s ex:p3 "b" .
    ex:s ex:p2 "c" .
}

ex:s ex:p3 "d" .
```

このクエリでは:

```
PREFIX ex: <https://example.com/>
DESCRIBE ?s
FROM ex:g1
FROM NAMED ex:g2
WHERE {
  GRAPH ex:g2 { ?s ?p "b" . }
}
```

Neptune は以下を返します。

```
ex:s ex:p1 "a" .
ex:s ex:p2 "c" .
```

ここでは、グラフパターン GRAPH ex:g2 { ?s ?p "b" } が最初に評価され、その結果として ?s のバインディングが行われ、次に DESCRIBE 部分がデフォルトのグラフ (現在は ex:g1) に対して評価されます。

ただし、このクエリでは:

```
PREFIX ex: <https://example.com/>
DESCRIBE ?s
FROM NAMED ex:g1
WHERE {
  GRAPH ex:g1 { ?s ?p "a" . }
}
```

Neptune は何も返しません。なぜなら、FROM NAMED 句があり、FROM 句がない場合、デフォルトのグラフは空になるからです。

次のクエリでは、DESCRIBE が使用され、FROM または FROM NAMED 句はありません。

```
PREFIX ex: <https://example.com/>
DESCRIBE ?s
WHERE {
  GRAPH ex:g1 { ?s ?p "a" . }
}
```

この状況では、デフォルトのグラフは、データベース内のすべての名前付きグラフの和集合に含まれるすべてのユニークなトリプルで構成されます (正式にはRDFマージ) 。そのため、Neptune は次のように返します。

```
ex:s ex:p1 "a" .
ex:s ex:p2 "c" .
ex:s ex:p3 "b" .
ex:s ex:p3 "d" .
```

SPARQL クエリステータス API

SPARQL クエリのステータスを取得するには、HTTP GET または POST を使用して、https://*your-neptune-endpoint*:*port*/sparql/status エンドポイントにリクエストを送信します。

SPARQL クエリステータスのリクエストパラメータ

queryId (オプション)

実行中の SPARQL クエリの ID。指定したクエリのステータスのみを表示します。

SPARQL クエリステータスのレスポンスの構文

```
{
    "acceptedQueryCount": integer,
    "runningQueryCount": integer,
    "queries": [
      {
        "queryId":"guid",
```

```
        "queryEvalStats":
          {
            "subqueries": integer,
            "elapsed": integer,
            "cancelled": boolean
          },
        "queryString": "string"
      }
    ]
}
```

SPARQL クエリステータスのレスポンス値

受け入れられましたQueryCount

Neptune エンジンの最後の再起動以降に受け入れられたクエリの数。

実行中QueryCount

現在実行中の SPARQL クエリの数。

クエリ

現在の SPARQL クエリのリスト。

queryId

クエリの GUID ID。Neptune は、この ID 値を各クエリに自動的に割り当てます。または、独自の ID を割り当てることもできます (Neptune Gremlin または SPARQL クエリにカスタム ID を挿入するを参照)。

クエリEvalStats

このクエリの統計情報。

subqueries

このクエリのサブクエリの数。

elapsed

これまでクエリが実行されていた時間 (マイクロ秒)。

キャンセル済み

True はクエリがキャンセルされたことを示します。

queryString

送信されたクエリ。

SPARQL クエリステータスの例

以下は、curl と HTTP GET を使用したステータスコマンドの例です。

```
curl https://your-neptune-endpoint:port/sparql/status
```

この出力には、実行中の単一のクエリが表示されます。

```
{
    "acceptedQueryCount":9,
    "runningQueryCount":1,
    "queries": [
        {
            "queryId":"fb34cd3e-f37c-4d12-9cf2-03bb741bf54f",
            "queryEvalStats":
                {
                    "subqueries": 0,
                    "elapsed": 29256,
                    "cancelled": false
                },
            "queryString": "SELECT ?s ?p ?o WHERE {?s ?p ?o}"
        }
    ]
}
```

SPARQL クエリのキャンセル

SPARQL クエリのステータスを取得するには、HTTP GET または POST を使用して、https://*your-neptune-endpoint*:*port*/sparql/status エンドポイントにリクエストを送信します。

SPARQL クエリのキャンセルリクエストパラメータ

cancelQuery

(必須) クエリをキャンセルするようステータスコマンドに指示します。このパラメータは値を取りません。

queryId

(必須) キャンセルする実行中の SPARQL クエリの ID。

silent

(オプション) `silent=true` の場合、実行中のクエリはキャンセルされます。HTTP レスポンスコードは 200 です。`silent` が存在しない場合、または `silent=false` である場合、クエリは HTTP 500 ステータスコードでキャンセルされます。

SPARQL クエリのキャンセルの例

例 1: **`silent=false`** でのキャンセル

次に示すのは、`curl` を使用し、`silent` パラメータを `false` に設定してクエリをキャンセルするステータスコマンドの例です。

```
curl https://your-neptune-endpoint:port/sparql/status \
  -d "cancelQuery" \
  -d "queryId=4d5c4fae-aa30-41cf-9e1f-91e6b7dd6f47" \
  -d "silent=false"
```

クエリがすでに結果のストリーミングを開始していない限り、キャンセルされたクエリは、次のようなレスポンスを持つ HTTP 500 コードを返します。

```
{
  "code": "CancelledByUserException",
  "requestId": "4d5c4fae-aa30-41cf-9e1f-91e6b7dd6f47",
  "detailedMessage": "Operation terminated (cancelled by user)"
}
```

クエリがすでに HTTP 200 コード (OK) を返し、キャンセルされる前に結果のストリーミングを開始した場合、タイムアウト例外情報が通常の出力ストリームに送信されます。

例 2: **`silent=true`** でのキャンセル

次に示すのは、`silent` パラメータが `true` に設定されている場合を除いて、上記と同じステータスコマンドの例です。

```
curl https://your-neptune-endpoint:port/sparql/status \
  -d "cancelQuery" \
  -d "queryId=4d5c4fae-aa30-41cf-9e1f-91e6b7dd6f47" \
  -d "silent=true"
```

このコマンドは `silent=false` の場合と同じレスポンスを返しますが、キャンセルされたクエリは次のようなレスポンスを伴う HTTP 200 コードを返すようになります。

```
{
  "head" : {
    "vars" : [ "s", "p", "o" ]
  },
  "results" : {
    "bindings" : [ ]
  }
}
```

Amazon Neptune での SPARQL 1.1 グラフストア HTTP プロトコル (GSP) の使用

SPARQL 1.1 グラフストア HTTP プロトコルレコメンデーションでは、W3C は RDF グラフを管理するための HTTP プロトコルを定義しました。RDF グラフコンテンツの削除、作成、および置き換え、および既存のコンテンツに RDF ステートメントを追加する操作を定義します。

グラフストアプロトコル (GSP) は、複雑な SPARQL クエリを記述することなく、グラフ全体を操作する便利な方法を提供します。

リリース： 1.0.5.0 (2021-07-27) 現在、Neptuneはこのプロトコルを完全にサポートしています。

グラフストアプロトコル (GSP) のエンドポイントは次のとおりです。

```
https://your-neptune-cluster:port/sparql/gsp/
```

GSP でデフォルトのグラフにアクセスするには、以下を使用します。

```
https://your-neptune-cluster:port/sparql/gsp/?default
```

GSP で名前付きグラフにアクセスするには、以下を使用します。

```
https://your-neptune-cluster:port/sparql/gsp/?graph=named-graph-URI
```

Neptune GSP 実装の特別な詳細

Neptune はGSPを定義する W3C レコメンデーションを完全に実装しています。ただし、仕様が カバーしていない状況はいくつかあります。

その 1 つは、PUT または POST リクエストが、リクエスト本文に、リクエスト URL で指定されたグラフとは異なる名前付きグラフを 1 つ以上指定する場合です。これは、リクエスト本文 RDF 形式がたとえば、`Content-Type: application/n-quads` または `Content-Type: application/trig` を使って名前付きグラフをサポートしている場合にのみ発生します。

この状況では、Neptune は、URL で指定された名前付きグラフだけでなく、本文に存在するすべての名前付きグラフを追加または更新します。

たとえば、空のデータベースから開始して、3 つのグラフにアップサートするよう PUT リクエストを送ります。1つの名前付き `urn:votes` は、すべての選挙年度からの全得票を含みます。他の2つ、名前付き `urn:votes:2005` および `urn:votes:2019` は、特定の選挙年の投票を含めます。リクエストとそのペイロードは次のようになります。

```
PUT "http://your-Neptune-cluster:port/sparql/gsp/?graph=urn:votes"
  Host: example.com
  Content-Type: application/n-quads

  PAYLOAD:

  <urn:JohnDoe> <urn:votedFor> <urn:Labour> <urn:votes:2005>
  <urn:JohnDoe> <urn:votedFor> <urn:Conservative> <urn:votes:2019>
  <urn:JaneSmith> <urn:votedFor> <urn:LiberalDemocrats> <urn:votes:2005>
  <urn:JaneSmith> <urn:votedFor> <urn:Conservative> <urn:votes:2019>
```

リクエストが実行されると、データベース内のデータは次のようになります。

```
<urn:JohnDoe>   <urn:votedFor> <urn:Labour>           <urn:votes:2005>
<urn:JohnDoe>   <urn:votedFor> <urn:Conservative>     <urn:votes:2019>
<urn:JaneSmith> <urn:votedFor> <urn:LiberalDemocrats> <urn:votes:2005>
<urn:JaneSmith> <urn:votedFor> <urn:Conservative>     <urn:votes:2019>
<urn:JohnDoe>   <urn:votedFor> <urn:Labour>           <urn:votes>
<urn:JohnDoe>   <urn:votedFor> <urn:Conservative>     <urn:votes>
<urn:JaneSmith> <urn:votedFor> <urn:LiberalDemocrats> <urn:votes>
```

```
<urn:JaneSmith> <urn:votedFor> <urn:Conservative>     <urn:votes>
```

別のあいまいな状況は、PUT、POST、GET または DELETE を使って、リクエストURL自体に複数のグラフが指定されている場合です。例:

```
POST "http://your-Neptune-cluster:port/sparql/gsp/?
graph=urn:votes:2005&graph=urn:votes:2019"
```

または:

```
GET "http://your-Neptune-cluster:port/sparql/gsp/?default&graph=urn:votes:2019"
```

この状況では、Neptune はリクエスト URL に指定できるグラフが 1 つだけであることを示すメッセージを含む HTTP 400 を返します。

SPARQL **explain** を使用して Neptune クエリ実行を分析する

Amazon Neptune に explain という名前の SPARQL 機能が追加されました。この機能は、Neptune エンジンが使用する実行アプローチを理解するためのセルフサービスツールです。SPARQL クエリを送信する HTTP コールに `explain` パラメータを追加してこれを呼び出します。

`explain` 機能は、クエリ実行プランの論理構造に関する情報を提供します。この情報を使用して潜在的な評価と実行障害を明らかにします。次に、「クエリに関するヒント」を使用して、クエリ実行プランを改善できます。

トピック

- Neptune における SPARQL クエリエンジンの仕組み
- Neptune クエリ実行の分析に SPARQL explain を使用する方法
- Neptune の SPARQL explain を呼び出す例
- Neptune SPARQL explain 演算子
- Neptune での SPARQL explain の制約事項

Neptune における SPARQL クエリエンジンの仕組み

SPARQL `explain` 機能が提供する情報を使用するには、Amazon Neptune SPARQL クエリエンジンの仕組みについてのいくつかの詳細を理解する必要があります。

エンジンはすべての SPARQL クエリを演算子のパイプラインに変換します。最初の演算子から始まり、バインドリストとして知られる中間ソリューションはこの演算子パイプラインを介して進みます。バインドリストは、テーブルヘッダーがクエリ内で使用される変数のサブセットであるテーブルとして考えることができます。テーブルの各行は、評価ポイントまでの結果を表します。

使用するデータには 2 つの名前空間プレフィックスが定義されていると仮定します。

```
@prefix ex:   <http://example.com> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
```

以下に示しているのは、このコンテキストのシンプルなバインドの例です。

```
?person       | ?firstName
------------------------------------------------------
ex:JaneDoe    | "Jane"
ex:JohnDoe    | "John"
ex:RichardRoe | "Richard"
```

3 人ごとに、このリストは ?person 変数をこの人物の識別子にバインドし、?firstName 変数をこの人物の名前にバインドします。

一般的なケースでは、たとえばデータに値がないクエリに変数の OPTIONAL 選択がある場合には、変数はバインドしないままにできます。

PipelineJoin 演算子は、explain 出力にある Neptune クエリエンジン演算子の例です。ここでは、前の演算子からの一連の着信バインドを入力として使用し、これを 3 つのパターン (つまり、(?person, foaf:lastName, ?lastName)) に結合します。この演算では、入力ストリームに ?person 変数のバインドを使用し、これを 3 つのパターンに置き換えて、データベースから 3 つを検索します。

前のテーブルからの着信バインドのコンテキストで実行するとき、PipelineJoin は次に示す 3 つの検索を評価します。

```
(ex:JaneDoe,    foaf:lastName, ?lastName)
(ex:JohnDoe,    foaf:lastName, ?lastName)
(ex:RichardRoe, foaf:lastName, ?lastName)
```

このアプローチはアズバウンド評価と呼ばれます。この評価プロセスのソリューションは、検出された ?lastName を着信ソリューションに当てはめて、着信ソリューションに再び結合されます。3 人の人物のすべての姓が見つかったとすると、演算子は次のような送信バインドリストを生成します。

```
?person       | ?firstName | ?lastName
---------------------------------------
ex:JaneDoe    | "Jane"     | "Doe"
ex:JohnDoe    | "John"     | "Doe"
ex:RichardRoe | "Richard"  | "Roe"
```

次に、この送信バインドリストは、パイプラインの次の入力として機能します。最終的に、パイプラインの最後の演算子の出力はクエリの結果を定義します。

演算子パイプラインは多くの場合に直線的です。つまり、各演算子は単一に接続された演算子のソリューションを発行します。ただし、一部の場合、よる複雑な構造になることができます。たとえば、SPARQL クエリの UNION 演算子は Copy 演算にマッピングされます。この演算はバインドを複製し、このコピーを 2 つのサブプラン (1 つは UNION の左側、もう 1 つは右側) に転送します。

演算子についての詳細は、「Neptune SPARQL explain 演算子」を参照してください。

Neptune クエリ実行の分析に SPARQL **explain** を使用する方法

SPARQL の explain 機能は、Neptune エンジンが使用する実行アプローチを理解するために役立つ Amazon Neptune のセルフサービスツールです。explain を呼び出すには、パラメータを explain=*mode* フォームで HTTP あるいは HTTPS リクエストに渡します。

モードの値は static、dynamic、details のいずれかです。

- 静的モードでは、explain はクエリプランの静的構造のみを表示します。
- 動的モードでは、explain にはクエリプランの動的な要素も含まれます。以上には、演算子を介して通過する中間バインドの数、送信バインドに対する着信バインドの割合、演算子の所要合計時間が含まれる場合があります。
- 詳細モードの場合、explain は dynamic モードで表示される情報に加えて、実際の SPARQL クエリ文字列や、結合演算子が基づくパターンの推定範囲数などの詳細を出力します。

Neptune は explain を使用して W3C SPARQL 1.1 プロトコル仕様に挙げられている 3 つの SPARQL クエリアクセスプロトコルはすべてサポートしています。すなわち次です。

1. HTTP GET
2. URL エンコードパラメータを使用した HTTP POST
3. テキストパラメータを使用した HTTP POST

SPARQL クエリエンジンの詳細については、「Neptune における SPARQL クエリエンジンの仕組み」を参照してください。

SPARQL explain を呼び出すことで生成される出力の種類については、「Neptune の SPARQL explain を呼び出す例」を参照してください。

Neptune の SPARQL **explain** を呼び出す例

このセクションの例では、Amazon Neptune のクエリ実行を分析するために、SPARQL explain 機能を呼び出して生成できるさまざまな種類の出力を示しています。

トピック

Explain 出力を理解する

この例では、Jane Doe は 2 人の人 (John Doe と Richard Roe) を知っています。

```
@prefix ex: <http://example.com> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

ex:JaneDoe foaf:knows ex:JohnDoe .
ex:JohnDoe foaf:firstName "John" .
ex:JohnDoe foaf:lastName "Doe" .
ex:JaneDoe foaf:knows ex:RichardRoe .
ex:RichardRoe foaf:firstName "Richard" .
ex:RichardRoe foaf:lastName "Roe" .
.
```

Jane Doe が知っているすべての人々の名前を確認するには、次のクエリを記述できます。

```
 curl http(s)://your_server:your_port/sparql \
   -d "query=PREFIX foaf: <https://xmlns.com/foaf/0.1/> PREFIX ex: <https://
www.example.com/> \
```

```
      SELECT ?firstName WHERE { ex:JaneDoe foaf:knows ?person . ?person
 foaf:firstName ?firstName }" \
   -H "Accept: text/csv"
```

このシンプルなクエリは次を返します。

```
firstName
John
Richard
```

次に、-d "explain=dynamic" を追加し、text/csv の代わりにデフォルトの出力タイプを使用して curl コマンドを変更し、explain を呼び出します。

```
 curl http(s)://your_server:your_port/sparql \
   -d "query=PREFIX foaf: <https://xmlns.com/foaf/0.1/> PREFIX ex: <https://
www.example.com/> \
      SELECT ?firstName WHERE { ex:JaneDoe foaf:knows ?person . ?person
 foaf:firstName ?firstName }" \
   -d "explain=dynamic"
```

このクエリは ASCII 形式 (HTTP コンテンツタイプ text/plain) の適切な出力を変えすようになります。これはデフォルトの出力タイプです。

```
################################################################################################
# ID # Out #1 # Out #2 # Name              # Arguments
            # Mode     # Units In # Units Out # Ratio # Time (ms) #
################################################################################################
# 0  # 1      # -      # SolutionInjection # solutions=[{}]
            # -        # 0         # 1          # 0.00  # 0         #
################################################################################################
# 1  # 2      # -      # PipelineJoin      # pattern=distinct(ex:JaneDoe, foaf:knows, ?
person)     # -        # 1         # 2          # 2.00  # 1         #
#    #        #        #                   # joinType=join
            #          #          #           #       #           #
#    #        #        #                   # joinProjectionVars=[?person]
            #          #          #           #       #           #
################################################################################################
# 2  # 3      # -      # PipelineJoin      # pattern=distinct(?person,
 foaf:firstName, ?firstName) # -        # 2         # 2          # 1.00  # 1         #
#    #        #        #                   # joinType=join
            #          #          #           #       #           #
```

```
#    #        #        #                  # joinProjectionVars=[?person, ?firstName]
             #          #          #           #       #           #
##########################################################################################
# 3  # 4      # -      # Projection        # vars=[?firstName]
             # retain   # 2        # 2         # 1.00  # 0         #
##########################################################################################
# 4  # -      # -      # TermResolution    # vars=[?firstName]
             # id2value # 2        # 2         # 1.00  # 1         #
##########################################################################################
```

Name 列およびその引数における演算についての詳細は、「EXPLAIN 演算子」を参照してください。

以下に行ごとの出力行を説明します。

1. 主要なクエリの最初のステップでは常に、SolutionInjection 演算子を使用してソリューションを挿入します。次に、このソリューションは評価プロセスを介して最終結果へと拡大します。

   この場合、ユニバーサルソリューションと呼ばれる { } を挿入します。VALUES 句 あるいは BIND がある場合、このステップではより複雑な変数バインドを挿入して始めることもできます。

   Units Out 列は、この単一のソリューションが演算子から流出したことを示しています。Out #1 列は、この演算子が結果を出す先の演算子を指定します。この例では、すべての演算子はテーブル内に続く演算子に接続しています。

2. 2 番目のステップは PipelineJoin です。これは、前の演算子 (Units In := 1) によって生成された単一のユニバーサル (完全に制約がない) ソリューションを入力として受け取ります。これは、pattern 引数によって定義されるタプルパターンに結合します。これは、パターンのシンプルな探索に対応します。この場合、3 つのパターンは以下のように定義されます。

   ```
   distinct( ex:JaneDoe, foaf:knows, ?person )
   ```

   joinType := join 引数は、これが通常の結合であることを示します (他のタイプには optional 結合、existence check 結合などが含まれます)。

   distinct := true 引数は、データベースから重複を排除した一致のみ (重複なし) を抽出し、重複を除外した一致を 変数 joinProjectionVars := ?person に重複してバインドすることを示します。

Units Out 列の値が 2 であることは、2 つのソリューションが流出していることを示しています。具体的には、これらは ?person 変数のバインドであり、Jane Doe が知っているとデータが示す 2 人を反映します。

```
?person
-------------
ex:JohnDoe
ex:RichardRoe
```

3. ステージ 2 の 2 つのソリューションは、入力 (Units In := 2) として 2 番目の PipelineJoin になります。この演算子は、次の 3 つのパターンに前の 2 つのソリューションを結合します。

```
distinct(?person, foaf:firstName, ?firstName)
```

?person 変数は、演算子の着信ソリューションによって ex:JohnDoe あるいは ex:RichardRoe のいずれかにバインドされることになります。この結果、PipelineJoin は名前である John と Richard を抽出します。2 つの発信ソリューション (Units Out:=2) は、次のようになります。

```
?person       | ?firstName
---------------------------
ex:JohnDoe    | John
ex:RichardRoe | Richard
```

4. 次の射影演算子は入力としてステージ 3 の 2 つのソリューションを用い (Units In := 2) を用いて、?firstName 変数に射影します。これによって、マッピング内の他のすべての変数バインドが排除され、2 つのバインドに渡されます (Units Out := 2)。

```
?firstName
----------
John
Richard
```

5. パフォーマンスを向上させるため、Neptune は、文字列自身ではなく、URI や文字列リテラルなどの項目に割り当てることが可能な内部の識別子で機能します。最後の演算子 TermResolution は、これらの内部の識別子から対応する項目の文字列に戻ってマッピングを実行します。

通常の (explain ではない) クエリ評価では、最後の演算子によって計算された結果はリクエストされたシリアル化形式にシリアル化され、クライアントにストリームされます。

詳細モード出力の例

> **Note**
>
> SPARQL 詳細説明モードは、Neptune エンジンリリース 1.0.2.1 からご利用いただけます。

動的モードではなく詳細モードの前と同じクエリを実行するとします。

```
 curl http(s)://your_server:your_port/sparql \
   -d "query=PREFIX foaf: <https://xmlns.com/foaf/0.1/> PREFIX ex: <https://
www.example.com/> \
       SELECT ?firstName WHERE { ex:JaneDoe foaf:knows ?person . ?person
 foaf:firstName ?firstName }" \
   -d "explain=details"
```

この例に示すように、出力は同じですが、出力の上部にあるクエリ文字列や PipelineJoin 演算子の patternEstimate カウントなど、いくつかの詳細が追加されています。

```
Query:
PREFIX foaf: <https://xmlns.com/foaf/0.1/> PREFIX ex: <https://www.example.com/>
SELECT ?firstName WHERE { ex:JaneDoe foaf:knows ?person . ?person foaf:firstName ?
firstName }

################################################################################################
# ID # Out #1 # Out #2 # Name              # Arguments
            # Mode     # Units In # Units Out # Ratio # Time (ms) #
################################################################################################
# 0  # 1      # -      # SolutionInjection # solutions=[{}]
            # -        # 0         # 1         # 0.00  # 0         #
################################################################################################
# 1  # 2      # -      # PipelineJoin      # pattern=distinct(ex:JaneDoe, foaf:knows, ?
person)     # -        # 1         # 2         # 2.00  # 13        #
#    #        #        #                   # joinType=join
            #          #         #           #       #           #
#    #        #        #                   # joinProjectionVars=[?person]
            #          #         #           #       #           #
```

```
#    #        #        #                  # patternEstimate=2
            #          #          #           #       #           #
#################################################################################################
# 2  # 3      # -      # PipelineJoin      # pattern=distinct(?person,
 foaf:firstName, ?firstName) # -        # 2        # 2         # 1.00  # 3         #
#    #        #        #                  # joinType=join
            #          #          #           #       #           #
#    #        #        #                  # joinProjectionVars=[?person, ?firstName]
            #          #          #           #       #           #
#    #        #        #                  # patternEstimate=2
            #          #          #           #       #           #
#################################################################################################
# 3  # 4      # -      # Projection        # vars=[?firstName]
            # retain   # 2        # 2         # 1.00  # 1         #
#################################################################################################
# 4  # -      # -      # TermResolution    # vars=[?firstName]
            # id2value # 2        # 2         # 1.00  # 7         #
#################################################################################################
```

静的モード出力の例

静的モード (デフォルト) ではなく詳細モードの前と同じクエリを実行するとします。

```
 curl http(s)://your_server:your_port/sparql \
   -d "query=PREFIX foaf: <https://xmlns.com/foaf/0.1/> PREFIX ex: <https://
www.example.com/> \
       SELECT ?firstName WHERE { ex:JaneDoe foaf:knows ?person . ?person
 foaf:firstName ?firstName }" \
   -d "explain=static"
```

この例に示すように、出力は、最後の 3 つの列が除外されていることを除いて、同じになります。

```
#################################################################################################
# ID # Out #1 # Out #2 # Name              # Arguments
            # Mode     #
#################################################################################################
# 0  # 1      # -      # SolutionInjection # solutions=[{}]
            # -        #
#################################################################################################
# 1  # 2      # -      # PipelineJoin      # pattern=distinct(ex:JaneDoe, foaf:knows, ?
person)     # -        #
#    #        #        #                  # joinType=join
            #          #
```

```
#    #        #        #                  # joinProjectionVars=[?person]
            #          #
#############################################################################################
# 2  # 3      # -      # PipelineJoin      # pattern=distinct(?person,
 foaf:firstName, ?firstName) # -        #
#    #        #        #                  # joinType=join
            #          #
#    #        #        #                  # joinProjectionVars=[?person, ?firstName]
            #          #
#############################################################################################
# 3  # 4      # -      # Projection        # vars=[?firstName]
            # retain   #
#############################################################################################
# 4  # -      # -      # TermResolution    # vars=[?firstName]
            # id2value #
#############################################################################################
```

パラメータをエンコードするさまざま方法

次のクエリの例では、SPARQL explain を呼び出すときにパラメータをエンコードする 2 つの異なる方法を示しています。

URL エンコードを使用する – この例は、パラメータの URL エンコードを使用し、動的出力を指定します。

```
curl -XGET "http(s)://your_server:your_port/sparql?query=SELECT%20*%20WHERE%20%7B
%20%3Fs%20%3Fp%20%3Fo%20%7D%20LIMIT%20%31&explain=dynamic"
```

パラメータを直接指定する - これは、POST を介してパラメータを直接渡すことを除き、前のクエリと同じです。

```
curl http(s)://your_server:your_port/sparql \
  -d "query=SELECT * WHERE { ?s ?p ?o } LIMIT 1" \
  -d "explain=dynamic"
```

テクスト/プレーンの他の出力タイプ

上記の例では、デフォルト text/plain 出力タイプを使用しています。Neptune は SPARQL explain 出力もフォーマット他の 2 つの MIME タイプのフォーマット、すなわち text/csv および text/html に出力できます。HTTP Accept ヘッダーを設定することがこれらを呼び出します。次に示すように、curl で -H フラグを使用することでこれを実行できます。

```
  -H "Accept: output type"
```

次に例を示します。

text/csv 出力

このクエリは、-H "Accept: text/csv" を指定して CSV MIME タイプの出力を呼び出します。

```
curl http(s)://your_server:your_port/sparql \
  -d "query=SELECT * WHERE { ?s ?p ?o } LIMIT 1" \
  -d "explain=dynamic" \
  -H "Accept: text/csv"
```

スプレッドシートやデータベースへのインポートに便利な CSV 形式は、次に示すようにファイルを explain 行ごとにセミコロン (;) で分離します。

```
ID;Out #1;Out #2;Name;Arguments;Mode;Units In;Units Out;Ratio;Time (ms)
0;1;-;SolutionInjection;solutions=[{}];-;0;1;0.00;0
1;2;-;PipelineJoin;pattern=distinct(?s, ?p, ?o),joinType=join,joinProjectionVars=[?s, ?
p, ?o];-;1;6;6.00;1
2;3;-;Projection;vars=[?s, ?p, ?o];retain;6;6;1.00;2
3;-;-;Slice;limit=1;-;1;1;1.00;1
```

text/html 出力

-H "Accept: text/html" を指定すると、explain は HTML テーブルを生成します。

```
<!DOCTYPE html>
<html>
  <body>
    <table border="1px">
      <thead>
        <tr>
          <th>ID</th>
          <th>Out #1</th>
          <th>Out #2</th>
          <th>Name</th>
          <th>Arguments</th>
          <th>Mode</th>
          <th>Units In</th>
```

```
          <th>Units Out</th>
          <th>Ratio</th>
          <th>Time (ms)</th>
        </tr>
      </thead>

      <tbody>
        <tr>
          <td>0</td>
          <td>1</td>
          <td>-</td>
          <td>SolutionInjection</td>
          <td>solutions=[{}]</td>
          <td>-</td>
          <td>0</td>
          <td>1</td>
          <td>0.00</td>
          <td>0</td>
        </tr>

        <tr>
          <td>1</td>
          <td>2</td>
          <td>-</td>
          <td>PipelineJoin</td>
          <td>pattern=distinct(?s, ?p, ?o)<br>
              joinType=join<br>
              joinProjectionVars=[?s, ?p, ?o]</td>
          <td>-</td>
          <td>1</td>
          <td>6</td>
          <td>6.00</td>
          <td>1</td>
        </tr>

        <tr>
          <td>2</td>
          <td>3</td>
          <td>-</td>
          <td>Projection</td>
          <td>vars=[?s, ?p, ?o]</td>
          <td>retain</td>
          <td>6</td>
          <td>6</td>
```

```
          <td>1.00</td>
          <td>2</td>
        </tr>

        <tr>
          <td>3</td>
          <td>-</td>
          <td>-</td>
          <td>Slice</td>
          <td>limit=1</td>
          <td>-</td>
          <td>1</td>
          <td>1</td>
          <td>1.00</td>
          <td>1</td>
        </tr>
      </tbody>
    </table>
  </body>
</html>
```

HTML はブラウザに次のようにレンダリングされます。

ID	Out #1	Out #2	Name	Arguments	Mode	Units In	Units Out	Ratio	Time (ms)
0	1	-	SolutionInjection	solutions=[{}]	-	0	1	0.00	0
1	2	-	PipelineJoin	pattern=distinct(?s, ?p, ?o) joinType=join joinProjectionVars=[?s, ?p, ?o]	-	1	6	6.00	1
2	3	-	Projection	vars=[?s, ?p, ?o]	retain	6	6	1.00	2
3	-	-	Slice	limit=1	-	1	1	1.00	1

DFE が無効の場合の SPARQL **explain** 出力の例

DFE 代替クエリエンジンが有効な場合の SPARQL explain レポート例を次に示します。

```
################################################################################################
# ID # Out #1 # Out #2 # Name              # Arguments


 # Mode     # Units In # Units Out # Ratio # Time (ms) #
################################################################################################
# 0  # 1      # -      # SolutionInjection # solutions=[{}]
```

```
 # -        # 0        # 1         # 0.00  # 0         #
##########################################################################################
# 1  # 2      # -      # HashIndexBuild    # solutionSet=solutionSet1


 # -        # 1        # 1         # 1.00  # 22        #
#    #        #        #                   # joinVars=[]


 #          #          #           #       #           #
#    #        #        #                   # sourceType=pipeline


 #          #          #           #       #           #
##########################################################################################
# 2  # 3      # -      # DFENode            # DFE Stats=


      # -        # 101      # 100       # 0.99  # 32        #
#    #        #        #                   # ====> DFE execution time (measured by
 DFEQueryEngine)

     #          #          #           #       #           #
#    #        #        #                   # accepted [micros]=127


 #          #          #           #       #           #
#    #        #        #                   # ready [micros]=2


 #          #          #           #       #           #
#    #        #        #                   # running [micros]=5627


 #          #          #           #       #           #
#    #        #        #                   # finished [micros]=0


 #          #          #           #       #           #
#    #        #        #                   #


 #          #          #           #       #           #
```

```
#    #        #        #                  #


 #          #          #           #       #           #
#    #        #        #                  # ===> DFE execution time (measured in
 DFENode)

      #          #          #           #       #           #
#    #        #        #                  # -> setupTime [ms]=1


 #          #          #           #       #           #
#    #        #        #                  # -> executionTime [ms]=14


 #          #          #           #       #           #
#    #        #        #                  # -> resultReadTime [ms]=0


 #          #          #           #       #           #
#    #        #        #                  #


 #          #          #           #       #           #
#    #        #        #                  #


 #          #          #           #       #           #
#    #        #        #                  # ===> Static analysis statistics


 #          #          #           #       #           #
#    #        #        #                  # --> 35907 micros spent in parser.


 #          #          #           #       #           #
#    #        #        #                  # --> 7643 micros spent in range count
 estimation

      #          #          #           #       #           #
#    #        #        #                  # --> 2895 micros spent in value resolution


 #          #          #           #       #           #
```

```
#    #        #        #                  #


 #          #          #           #       #           #
#    #        #        #                  # --> 39974925 micros spent in optimizer
 loop

    #          #          #           #       #           #
#    #        #        #                  #


 #          #          #           #       #           #
#    #        #        #                  #


 #          #          #           #       #           #
#    #        #        #                  # DFEJoinGroupNode[ children={


  #          #          #           #       #           #
#    #        #        #                  #   DFEPatternNode[(?1, TERM[117442062], ?
2, ?3) . project DISTINCT[?1, ?2] {rangeCountEstimate=100},

  #          #          #           #       #           #
#    #        #        #                  #     OperatorInfoWithAlternative[


  #          #          #           #       #           #
#    #        #        #                  #       rec=OperatorInfo[


  #          #          #           #       #           #
#    #        #        #                  #         type=INCREMENTAL_PIPELINE_JOIN,


 #          #          #           #       #           #
#    #        #        #                  #
 costEstimates=OperatorCostEstimates[

                                   #          #          #           #       #
  #
#    #        #        #                  #
 costEstimate=OperatorCostEstimate[in=1.0000,out=100.0000,io=0.0002,comp=0.0000,mem=0],
```

```
                                    #           #           #           #       #
 #
#    #        #        #                    #
 worstCaseCostEstimate=OperatorCostEstimate[in=1.0000,out=100.0000,io=0.0002,comp=0.0000,mem=0]

                       #           #           #           #       #           #
#    #        #        #                    #       alt=OperatorInfo[


  #           #           #           #       #           #
#    #        #        #                    #         type=INCREMENTAL_HASH_JOIN,


 #           #           #           #       #           #
#    #        #        #                    #
 costEstimates=OperatorCostEstimates[

                                    #           #           #           #       #
  #
#    #        #        #                    #
 costEstimate=OperatorCostEstimate[in=1.0000,out=100.0000,io=0.0003,comp=0.0000,mem=3212],

                                #           #           #           #       #           #
#    #        #        #                    #
 worstCaseCostEstimate=OperatorCostEstimate[in=1.0000,out=100.0000,io=0.0003,comp=0.0000,mem=32

                  #           #           #           #       #           #
#    #        #        #                    #   DFEPatternNode[(?1, TERM[150997262], ?
4, ?5) . project DISTINCT[?1, ?4] {rangeCountEstimate=100},

  #           #           #           #       #           #
#    #        #        #                    #     OperatorInfoWithAlternative[


  #           #           #           #       #           #
#    #        #        #                    #       rec=OperatorInfo[


  #           #           #           #       #           #
#    #        #        #                    #         type=INCREMENTAL_HASH_JOIN,


 #           #           #           #       #           #
```

```
#    #        #        #                   #
 costEstimates=OperatorCostEstimates[

                                    #          #          #           #       #
  #
#    #        #        #                   #
 costEstimate=OperatorCostEstimate[in=100.0000,out=100.0000,io=0.0003,comp=0.0000,mem=6400],

                               #          #          #           #       #           #
#    #        #        #                   #
 worstCaseCostEstimate=OperatorCostEstimate[in=100.0000,out=100.0000,io=0.0003,comp=0.0000,mem=

                  #          #          #           #       #           #
#    #        #        #                   #       alt=OperatorInfo[


  #          #          #           #       #           #
#    #        #        #                   #         type=INCREMENTAL_PIPELINE_JOIN,


 #          #          #           #       #           #
#    #        #        #                   #
 costEstimates=OperatorCostEstimates[

                                    #          #          #           #       #
  #
#    #        #        #                   #
 costEstimate=OperatorCostEstimate[in=100.0000,out=100.0000,io=0.0010,comp=0.0000,mem=0],

                                 #          #          #           #       #           #
#    #        #        #                   #
 worstCaseCostEstimate=OperatorCostEstimate[in=100.0000,out=100.0000,io=0.0010,comp=0.0000,mem=

                    #          #          #           #       #           #
#    #        #        #                   # },


 #          #          #           #       #           #
#    #        #        #                   # ]


 #          #          #           #       #           #
#    #        #        #                   #
```

```
 #          #          #          #       #           #
#    #        #        #                  # ===> DFE configuration:


 #          #          #          #       #           #
#    #        #        #                  # solutionChunkSize=5000


 #          #          #          #       #           #
#    #        #        #                  # ouputQueueSize=20


 #          #          #          #       #           #
#    #        #        #                  # numComputeCores=3


 #          #          #          #       #           #
#    #        #        #                  # maxParallelIO=10


 #          #          #          #       #           #
#    #        #        #                  # numInitialPermits=12


 #          #          #          #       #           #
#    #        #        #                  #


 #          #          #          #       #           #
#    #        #        #                  #


 #          #          #          #       #           #
#    #        #        #                  # ====> DFE configuration (reported back)


 #          #          #          #       #           #
#    #        #        #                  # numComputeCores=3


 #          #          #          #       #           #
#    #        #        #                  # maxParallelIO=2
```

```
 #          #          #           #       #           #
#    #        #        #                   # numInitialPermits=12


 #          #          #           #       #           #
#    #        #        #                   #


 #          #          #           #       #           #
#    #        #        #                   # ===> Statistics & operator histogram


 #          #          #           #       #           #
#    #        #        #                   # ==> Statistics


 #          #          #           #       #           #
#    #        #        #                   # -> 3741 / 3668 micros total elapsed (incl.
 wait / excl. wait)

 #          #          #           #       #           #
#    #        #        #                   # -> 3741 / 3 millis total elapse (incl.
 wait / excl. wait)

    #          #          #           #       #           #
#    #        #        #                   # -> 3741 / 0 secs total elapsed (incl.
 wait / excl. wait)

     #          #          #           #       #           #
#    #        #        #                   # ==> Operator histogram


 #          #          #           #       #           #
#    #        #        #                   # -> 47.66% of total time (excl. wait):
 pipelineScan (2 instances)

     #          #          #           #       #           #
#    #        #        #                   # -> 10.99% of total time (excl. wait):
 merge (1 instances)

     #          #          #           #       #           #
#    #        #        #                   # -> 41.17% of total time (excl. wait):
 symmetricHashJoin (1 instances)
```

```
     #          #          #           #       #           #
#    #        #        #                  # -> 0.19% of total time (excl. wait): drain
 (1 instances)

 #          #          #           #       #           #
#    #        #        #                  #


 #          #          #           #       #           #
#    #        #        #                  # nodeId | out0   | out1 | opName
   | args                                          | rowsIn | rowsOut | chunksIn |
 chunksOut | elapsed* | outWait | outBlocked | ratio    | rate* [M/s] | rate [M/s] | %
    #          #          #           #       #           #
#    #        #        #                  # ------ | ------ | ---- | -----------------
 | ------------------------------------------------ | ------ | ------- | -------- |
 --------- | -------- | ------- | ---------- | -------- | ----------- | ---------- |
 ----- #          #          #           #       #           #
#    #        #        #                  # node_0 | node_2 | -    | pipelineScan
  | (?1, TERM[117442062], ?2, ?3) DISTINCT [?1, ?2]  | 0      | 100     | 0        | 1
        | 874      | 0       | 0          | Infinity | 0.1144      | 0.1144     | 23.83
 #          #          #           #       #           #
#    #        #        #                  # node_1 | node_2 | -    | pipelineScan
  | (?1, TERM[150997262], ?4, ?5) DISTINCT [?1, ?4]  | 0      | 100     | 0        | 1
        | 874      | 0       | 0          | Infinity | 0.1144      | 0.1144     | 23.83
 #          #          #           #       #           #
#    #        #        #                  # node_2 | node_4 | -    | symmetricHashJoin
 |                                                 | 200    | 100     | 2        | 2
        | 1510     | 73      | 0          | 0.50     | 0.0662      | 0.0632     | 41.17
 #          #          #           #       #           #
#    #        #        #                  # node_3 | -      | -    | drain
  |                                                 | 100    | 0       | 1        | 0
        | 7        | 0       | 0          | 0.00     | 0.0000      | 0.0000     | 0.19
 #          #          #           #       #           #
#    #        #        #                  # node_4 | node_3 | -    | merge
  |                                                 | 100    | 100     | 2        | 1
        | 403      | 0       | 0          | 1.00     | 0.2481      | 0.2481     | 10.99
 #          #          #           #       #           #
#############################################################################################
# 3  # 4      # -      # HashIndexJoin     # solutionSet=solutionSet1


 # -        # 100      # 100       # 1.00  # 4         #
```

```
#    #        #        #                  # joinType=join


 #          #          #           #       #           #
##########################################################################################
# 4  # 5      # -      # Distinct          # vars=[?s, ?o, ?o1]


 # -        # 100      # 100       # 1.00  # 9         #
##########################################################################################
# 5  # 6      # -      # Projection        # vars=[?s, ?o, ?o1]


 # retain   # 100      # 100       # 1.00  # 2         #
##########################################################################################
# 6  # -      # -      # TermResolution    # vars=[?s, ?o, ?o1]


 # id2value # 100      # 100       # 1.00  # 11        #
##########################################################################################
```

Neptune SPARQL **explain** 演算子

以下のセクションでは、Amazon Neptune で現在使用できる SPARQL `explain` 機能の演算子とパラメータについて説明しています。

> **Important**
>
> SPARQL `explain` 機能は今後も改良されます。ここに記載されている演算子とパラメータは、今後のバージョンで変更される可能性があります。

トピック

- Aggregationoperator
- ConditionalRoutingoperator
- Copyoperator
- DFENodeoperator
- Distinctoperator
- Federationoperator

- Filteroperator
- HashIndexBuildoperator
- HashIndexJoinoperator
- MergeJoinoperator
- NamedSubqueryoperator
- PipelineJoinoperator
- PipelineCountJoinoperator
- PipelinedHashIndexJoinoperator
- Projectionoperator
- PropertyPathoperator
- TermResolutionoperator
- Sliceoperator
- SolutionInjectionoperator
- Sortoperator
- VariableAlignmentoperator

Aggregationoperator

SPARQL 集約演算のセマンティクス (count、max、min、sum など) を実装した 1 つ以上の集計を実行します。

Aggregation には、オプションとして groupBy を使用したグループ化、およびオプションとして having 制約が用意されています。

引数

- groupBy – (オプション) 着信ソリューションのグループ化の基盤となる連続式を指定する groupBy 句を提供します。
- aggregates – (必須) 集約式の順序リストを指定します。
- having – (オプション) SPARQL クエリ内の having 句に指定されるように、グループに制約フィルターを追加します。

ConditionalRoutingoperator

指定された条件に基づいて、受信ソリューションをルーティングします。条件を満たすソリューションは、Out #1 に参照されるオペレーター ID にルーティングされます。条件を満たさないソリューションは、Out #2 に参照されるオペレーターにルーティングされません。

引数

- condition – (必須) ルーティング条件。

Copyoperator

指定されたモードに示されるように、ソリューションストームを委任します。

モード

- forward – ソリューションを Out #1 によって識別されるダウンストリーム演算子に転送します。
- duplicate – ソリューションを複製し、それぞれを Out #1 および Out #2 によって識別される 2 つの演算子に転送します。

Copy には引数がありません。

DFENodeoperator

この演算子は、DFE 代替クエリエンジンによって実行されるプランの抽象化です。詳細な DFE 計画は、この演算子の引数に概説されています。引数は現在、DFE プランの詳細なランタイム統計を含むようにオーバーロードされています。これには、DFE によるクエリ実行のさまざまなステップに費やされた時間が含まれます。

DFE クエリプランの論理最適化抽象構文ツリー (AST) には、計画中に考慮された演算子タイプに関する情報と、演算子の実行に関連する最良および最悪の場合のコストが出力されます。AST は、現時点で次のタイプのノードで構成されています。

- DFEJoinGroupNode – 1 つ以上の DFEPatternNodes の結合を表します。
- DFEPatternNode - 基礎となるデータベースから一致するタプルが投影される基になるパターンを括弧で囲みます。

サブセクション、Statistics & Operator histogram には、各オペレータが使用するCPU時間の DataflowOp 計画と内訳の実行時間に関する詳細が含まれています。この下に、DFE によって実行されるプランの詳細なランタイム統計を出力する表があります。

> **Note**
>
> DFE はラボモードでリリースされる実験的な機能なので、explain 出力は変更される可能性があります。

Distinctoperator

変数のサブセットにおける個別の射影を計算し、重複を排除します。その結果、流入するソリューションの数は流出するソリューションの数より大きいかあるいは等しくなります。

引数

- vars – (必須) Distinct 射影を適用する変数。

Federationoperator

指定されたクエリを指定されたリモート SPARQL エンドポイントに渡します。

引数

- endpoint – (必須) SPARQL SERVICE ステートメントのエンドポイント URL。これは定数文字列にすることができます。または、クエリエンドポイントが同じクエリ内の変数に基づいて決定される場合は、変数名にすることができます。
- query – (必須) リモートエンドポイントに送信される、再構築されたクエリ文字列。エンジンは、クライアントが指定されていない場合でも、このクエリにデフォルトのプレフィックスを追加します。
- silent – (必須) SILENT キーワードがキーワードの後に表示されるかどうかを示すブール値。SILENT は、リモート SERVICE 部分が失敗してもクエリ全体が失敗しないようにエンジンに指示します。

Filteroperator

着信ソリューションをフィルタリングします。フィルタリング条件を満たすソリューションのみがアップストリームに転送され、その他すべては削除されます。

引数

- condition – (必須) フィルタリング条件。

HashIndexBuildoperator

バインドのリストを取得し、solutionSet 引数で定義される名前を持つハッシュインデックスにこのリストをスプールします。通常は、以降の演算子の実行は、名前を参照して、この一連のソリューションに対して結合します。

引数

- solutionSet – (必須) 一連のハッシュインデックスソリューションの名前。
- sourceType – (必須) ハッシュインデックスに保存されるバインドが取得される元のソースのタイプ。
  - pipeline – 演算子パイプラインのダウンストリーム演算子から着信するソリューションをハッシュインデックスにスプールします。
  - binding set – sourceBindingSet 引数によって指定される一連の固定バインドをハッシュインデックスにスプールします。
- sourceBindingSet – (オプション) sourceType 引数の値が binding set の場合、この引数はハッシュインデックスにスプールされる一連の静的バインドを指定します。

HashIndexJoinoperator

solutionSet 引数によって識別されるハッシュインデックスに対する着信ソリューションを結合します。

引数

- solutionSet – (必須) 結合するソリューションの名前。これは、HashIndexBuild 演算子を使用して前のステップで作成したハッシュインデックスである必要があります。
- joinType – (必須) 実行する結合のタイプ。
  - join – 共有するすべての変数間における完全な一致を必要とする、通常結合です。
  - optional – SPARQL OPTIONAL 演算子セマンティクスを使用する optional 結合。
  - minus – SPARQL MINUS 演算子セマンティクスを使用した、結合パートナーが存在しないマッピングがある minus 演算。

  - existence check – 結合パートナーがあるかどうかを確認し、この確認の結果に existenceCheckResultVar 変数をバインドします。
- constraints – (オプション) 結合中に考慮される追加の結合制約。これらの制約を満たさない結合は、除外されます。
- existenceCheckResultVar – (オプション) joinType が existence check と等しくなる結合のみに使用されます (前の joinType 引数を参照)。

MergeJoinoperator

solutionSets 引数によって識別される、一連の複数のソリューション間のマージ結合。

引数

- solutionSets – (必須) 一緒に結合する一連のソリューション。

NamedSubqueryoperator

subQuery 引数によって識別されるサブクエリの評価をトリガーし、その結果を solutionSet 引数によって指定される一連のソリューション内にスプールします。演算子の着信ソリューションはサブクエリに転送され、その後次の演算子に転送されます。

引数

- subQuery – (必須) 評価するサブクエリの名前。サブクエリは出力で明示的にレンダリングされます。
- solutionSet – (必須) サブクエリの結果を保存する一連のソリューションの名前。

PipelineJoinoperator

前の演算子の出力を入力として受け取り、これを pattern 引数によって定義されるタプルパターンに結合します。

引数

- pattern– (必須) パターン。 の形式をとり subject-predicate-object、オプションで結合の背後にある -graph タプルを使用します。パターンに distinct が指定されている場合、この結合は、すべての一致するソリューションではなく、projectionVars 引数によって指定される射影変数からの重複排除ソリューションのみを抽出します。

- inlineFilters – (オプション) パターンで変数に適用される一連のフィルター。パターンはこれらのフィルターと組み合わせて評価されます。
- joinType – (必須) 実行する結合のタイプ。
  - join – 共有するすべての変数間における完全な一致を必要とする、通常結合です。
  - optional – SPARQL OPTIONAL 演算子セマンティクスを使用する optional 結合。
  - minus – SPARQL MINUS 演算子セマンティクスを使用した、結合パートナーが存在しないマッピングがある minus 演算。
  - existence check – 結合パートナーがあるかどうかを確認し、この確認の結果に existenceCheckResultVar 変数をバインドします。
- constraints – (オプション) 結合中に考慮される追加の結合制約。これらの制約を満たさない結合は、除外されます。
- projectionVars – (オプション) 射影変数。distinct := true と組み合わせて使用され、指定した一連の変数から重複を排除した射影の抽出を強制します。
- cutoffLimit – (オプション) 抽出された結合パートナーの数のカットオフ制限。デフォルトによる制限はありませんが、結合パートナーがあることを証明あるいは否定するために十分な場合に、FILTER (NOT) EXISTS 句を実装する結合を実行するときにこれを 1 に設定できます。

PipelineCountJoinoperator

PipelineJoin の変形。結合の代わりに、一致する結合パートナーを単にカウントし、countVar 引数によって指定される変数にこのカウントをバインドします。

引数

- countVar – (必須) カウントの結果の変数、つまり結合パートナーの数がバインドされる必要があります。
- pattern– (必須) パターン。 の形式をとり subject-predicate-object、オプションで結合の背後にある -graph タプルを取ります。パターンに distinct が指定されている場合、この結合は、すべての一致するソリューションではなく、projectionVars 引数によって指定される射影変数からの重複排除ソリューションのみを抽出します。
- inlineFilters – (オプション) パターンで変数に適用される一連のフィルター。パターンはこれらのフィルターと組み合わせて評価されます。
- joinType – (必須) 実行する結合のタイプ。
  - join – 共有するすべての変数間における完全な一致を必要とする、通常結合です。

  - optional – SPARQL OPTIONAL 演算子セマンティクスを使用する optional 結合。
  - minus – SPARQL MINUS 演算子セマンティクスを使用した、結合パートナーが存在しないマッピングがある minus 演算。
  - existence check – 結合パートナーがあるかどうかを確認し、この確認の結果に existenceCheckResultVar 変数をバインドします。
- constraints – (オプション) 結合中に考慮される追加の結合制約。これらの制約を満たさない結合は、除外されます。
- projectionVars – (オプション) 射影変数。distinct := true と組み合わせて使用され、指定した一連の変数から重複を排除した射影の抽出を強制します。
- cutoffLimit – (オプション) 抽出された結合パートナーの数のカットオフ制限。デフォルトによる制限はありませんが、結合パートナーがあることを証明あるいは否定するために十分な場合に、FILTER (NOT) EXISTS 句を実装する結合を実行するときにこれを 1 に設定できます。

PipelinedHashIndexJoinoperator

これは all-in-one ビルドハッシュインデックスおよび結合演算子です。バインディングのリストを取得してハッシュインデックスにスプールし、そのハッシュインデックスに対して受信ソリューションを結合します。

引数

- sourceType – (必須) ハッシュインデックスに保存されるバインドが取得される元のソースのタイプ。
  - pipeline – PipelinedHashIndexJoin は、演算子パイプラインのダウンストリームの演算子から着信するソリューションをハッシュインデックスにスプールします。
  - binding set – PipelinedHashIndexJoin は、sourceBindingSet 引数によって指定された固定バインドセットをハッシュインデックスにスプールします。
- sourceSubQuery — (オプション) sourceType 引数の値が pipeline の場合、この引数は、評価されてハッシュインデックスにスプールされるサブクエリを指定します。
- sourceBindingSet – (オプション) sourceType 引数の値が binding set の場合、この引数はハッシュインデックスにスプールされる静的バインドセットを指定します。
- joinType – (必須) 実行される結合のタイプ。
  - join – 共有するすべての変数間における完全な一致を必要とする、通常結合です。
  - optional – SPARQL OPTIONAL 演算子セマンティクスを使用する optional 結合。

  - minus – SPARQL MINUS 演算子セマンティクスを使用した、結合パートナーが存在しないマッピングがある minus 演算。
  - existence check – 結合パートナーがあるかどうかを確認し、この確認の結果に existenceCheckResultVar 変数をバインドします。
- existenceCheckResultVar – (オプション) joinType が existence check に等しい場合にのみ、結合に使用されます (前の joinType 引数を参照)。

Projectionoperator

変数のサブセット間で射影を行います。流入するソリューションの数は流出するソリューションの数と同じですが、ソリューションの形はモードの設定に応じて異なります。

モード

- retain – vars 引数によって指定される変数のみをソリューションに保持します。
- drop – vars 引数によって指定されるすべての変数を除外します。

引数

- vars – (必須) モード設定に応じて、変数を保持あるいは除外します。

PropertyPathoperator

次のような再帰的なプロパティパスを有効にします。+ または*。Neptune は、iterationTemplate 引数によって指定されるテンプレートに基づく固定小数点反復アプローチを実装します。既知の左側または右側の変数は、新しいソリューションが見つからなくなるまで、すべての固定小数点反復のテンプレートにバインドされます。

引数

- iterationTemplate – (必須) 固定小数点反復を実装するために使用されるサブクエリテンプレートの名前です。
- leftTerm – (必須) プロパティパスの左側の項目 (変数あるいは定数) です。
- rightTerm – (必須) プロパティパスの右側の項目 (変数あるいは定数) です。
- lowerBound – (必須) 固定小数点反復の下限 (* クエリでは 0、または + クエリでは 1)。

TermResolutionoperator

モードに応じて、内部文字列識別子の値を対応する外部の文字列に変換するか、または外部文字列を内部文字列識別子の値に変換します。

モード

- value2id – 内部 ID の値に対応するリテラルや URI などのマップ項目 (内部値にエンコード)。
- id2value – 内部 ID をリテラルや URI など対応する項目にマッピングします (内部値にデコード)。

引数

- vars – (必須) マップする必要がある文字列あるいは内部文字列 ID の変数を指定します。

Sliceoperator

SPARQL の LIMIT および OFFSET 句のセマンティクスを使用して、着信ソリューションストリームにスライスを実装します。

引数

- limit – (オプション) 転送されるソリューションの制限。
- offset – (オプション) ソリューションが転送のために評価されるオフセット。

SolutionInjectionoperator

入力を受け取りません。クエリプランにソリューションを静的に挿入し、これを solutions 引数に記録します。

クエリプランは常にこの静的挿入で始まります。静的バインドの複数のソースを組み合わせることで (VALUES や BIND 句からなど)、挿入する静的ソリューションがクエリ自身から配信されることができる場合、SolutionInjection 演算子は配信されたこれらの静的ソリューションを挿入します。シンプルなケースでは、これらは外部の VALUES 句によって示されるバインドを反映します。

クエリから配信される静的ソリューションがない場合、SolutionInjection はユニバーサルソリューションと呼ばれる空を挿入します。これは、クエリ評価プロセスを通して展開し、乗算されます。

引数

- solutions – (必須) 演算子によって挿入されるソリューションのシーケンス。

Sortoperator

指定されたソート条件を使用して、一連のソリューションをソートします。

引数

- sortOrder – (必須) 変数の注文リスト。各変数には ASC (昇順) または DESC (降順) 識別子が含まれ、一連のソリューションをソートするために順次使用されます。

VariableAlignmentoperator

ソリューションを 1 つ 1 つ検査し、2 つの変数 (指定された sourceVar および 指定された targetVar) でそれぞれに配列を実行します。

ソリューションの sourceVar および targetVar に同じ値がある場合、この変数は同調している考慮され、このソリューションは転送されます。重複した sourceVar は排除されます。

変数が異なる値にバインドされる場合、このソリューションは全体的に除外されます。

引数

- sourceVar – (必須) ターゲット変数と比較されるソース変数。ソリューションで同調に成功した場合、これは、2 つの変数に同じ値があり、ソース変数が排除されることになります。
- targetVar – (必須) ソース変数が比較されるターゲット変数。同調に成功した場合でも保持されます。

Neptune での SPARQL **explain** の制約事項

Neptune SPARQL explain 機能のリリースには以下の制限があります。

Neptune は現在 SPARQL SELECT クエリのみで Explain をサポートしています

他のクエリフォーム (ASK、CONSTRUCT、DESCRIBE、SPARQL UPDATE クエリなど) の評価プロセスに関する情報については、これらのクエリを SELECT クエリに変換できます。次に、explain を使用して対応する SELECT クエリを代わりに検査します。

たとえば、ASK WHERE {...} クエリに関する explain 情報を取得するには、対応する SELECT WHERE {...} LIMIT 1 クエリを explain を使用して実行します。

同様に、CONSTRUCT {...} WHERE {...} クエリについては、CONSTRUCT {...} 部分を除外し、2 番目の WHERE {...} 句で explain を使用して SELECT クエリを実行します。通常の場合、2 番目の WHERE 句を評価すると、CONSTRUCT クエリの処理の主要な課題が明らかになります。これは、2 番目の WHERE から CONSTRUCT テンプレートに流出するソリューションが通常簡単な置換のみを必要とするためです。

Explain 演算子は今後のリリースで変更される可能性があります

SPARQL explain 演算子とそのパラメータは、今後のリリースで変更される可能性があります。

Explain 出力は今後のリリースで変更される可能性があります

たとえば、列ヘッダーが変更され、より多くの列がテーブルに追加される可能性があります。

SERVICE 拡張を使用した Neptune での SPARQL フェデレーティッドクエリ

Amazon Neptune は、SERVICE キーワードを使用する SPARQL フェデレーティッドクエリ拡張を完全にサポートします。(詳細については、SPARQL 1.1 Federated Query を参照してください。)

> **Note**
>
> この機能は、リリース 1.0.1.0.200463.0 (2019-10-15) で始めることで使用できます。

SERVICE キーワードは、リモート SPARQL エンドポイントに対してクエリの一部を実行し、最終的なクエリ結果を構成するように SPARQL クエリエンジンに指示します。READ オペレーションのみ可能です。WRITE および DELETE オペレーションはサポートされていません。Neptune は、仮想プライベートクラウド (VPC) 内の SPARQL エンドポイントに対してのみフェデレーティッドクエリを実行できます。ただし、VPC のリバースプロキシを使用して、VPC 内で外部データソースにアクセスできるようにすることもできます。

> **Note**
>
> SPARQL SERVICE を使用して、同じ VPC 内の 2 つ以上の Neptune クラスターにクエリをフェデレートする場合、それらのすべての Neptune クラスターが相互に通信できるようにセキュリティグループを設定する必要があります。

> **Important**
>
> SPARQL 1.1 フェデレーションは、クエリとパラメータを外部 SPARQL エンドポイントに渡すときに、ユーザーに代わってサービスリクエストを行います。外部 SPARQL エンドポイントがアプリケーションのデータ処理とセキュリティ要件を満たしていることを確認するのはお客様の責任です。

Neptune フェデレーティッドクエリの例

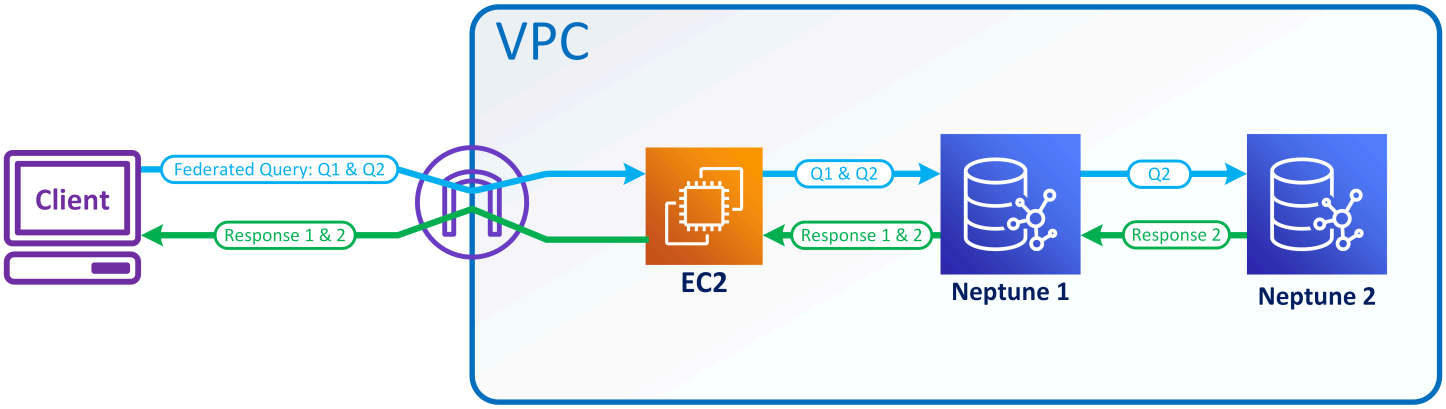
以下の簡単な例は、SPARQL フェデレーティッドクエリがどのように機能するかを示しています。

顧客が次のクエリを `http://neptune-1:8182/sparql` でNeptune-1に送信するとします。

```
SELECT * WHERE {
   ?person rdf:type foaf:Person .
   SERVICE <http://neptune-2:8182/sparql> {
       ?person foaf:knows ?friend .
    }
}
```

1. Neptune-1は最初のクエリパターン (`?person rdf:type foaf:Person`) を評価し (Q 1)、結果を使用して `?person` を Q-2 (`?person foaf:knows ?friend`) で解決します。それから結果のパターンを `http://neptune-2:8182/sparql` で Neptune-2 へ送ります。
2. Neptune-2 は Q-2 を評価し、結果を Neptune-1 に返します。
3. Neptune-1 は両方のパターンのソリューションに参加し、結果をお客様に返します。

このフローを、次の図に示します。



> **Note**
>
> 「デフォルトでは、オプティマイザは、クエリ実行のどの時点で SERVICE 命令が実行されるかを決定します。この配置は、joinOrder クエリのヒントを使って上書きできます。

Neptune でのフェデレーティッドクエリのアクセスコントロール

Neptune は認証と認可に AWS Identity and Access Management (IAM) を使用します。フェデレーティッドクエリのアクセスコントロールには、複数の Neptune DB インスタンスが含まれる場合があります。これらのインスタンスは、アクセスコントロールに関する要件が異なる場合があります。特定の状況では、これによってフェデレーティッドクエリの実行が制限される場合があります。

前のセクションで説明した簡単な例を考えてみましょう。Neptune-1 は呼び出されたのと同じクレデンシャルで Neptune 2 を呼び出します。

- Neptune-1 に IAM 認証と認可が必要であり、一方 Neptune-2 は不要であれば、フェデレーションクエリを作成するために必要なのは、Neptune-1 の適切な IAM 権限だけです。
- Neptune-1 と Neptune-2 の両方で IAM 認証と認可が必要な場合は、両方のデータベースに IAM 許可をアタッチしてフェデレーティッドクエリを作成する必要があります。両方のクラスターは、同じ AWS アカウントと同じリージョンに存在する必要があります。クロスリージョンおよび/またはクロスアカウントフェデレーティッドクエリアーキテクチャは現在サポートされていません。
- ただし、 Neptune-1 の IAM が有効ではなく、Neptune-2 が有効な場合は、フェデレーティッドクエリを作成することはできません。これは、Neptune-1 が IIAM 認証情報を取得できず、クエリの 2 番目の部分を認証するために Neptune-2 に渡すことができないためです。

Neptune のグラフ可視化ツール

Neptune グラフノートブックに組み込まれている視覚化機能に加えて、 AWS パートナーやサードパーティーベンダーによって構築されたソリューションを使用して、Neptune に保存されているデータを視覚化することもできます。

高度なグラフの可視化は、複雑なクエリの記述方法を知らなくても、組織内のデータサイエンティスト、マネージャー、その他の役割がグラフデータをインタラクティブに探索するのに役立ちます。

トピック

- オープンソースのグラフエクスプローラー
- Tom Sawyer ソフトウェア
- Cambridge Intelligence
- Graphistry
- metaphacts
- G.V( )
- リンク

オープンソースのグラフエクスプローラー

グラフエクスプローラーは、Apache-2.0 ライセンスの下で利用可能な、グラフデータ用のオープンソースのローコードビジュアル探索ツールです。グラフクエリを記述しなくても、グラフデータベース内のラベル付きプロパティグラフ (LPG) またはリソース記述フレームワーク (RDF) データを参照できます。Graph-Explorer は、組織内のデータサイエンティスト、ビジネスアナリスト、その他の役割がグラフクエリ言語を学ばなくても、グラフデータをインタラクティブに探索できるようにすることを目的としています。

Graph-Explorer は、グラフデータを可視化するコンテナとしてデプロイできる React ベースの Web アプリケーションを提供します。Amazon Neptune または Apache TinkerPop Gremlin または SPARQL 1.1 エンドポイントを提供する他のグラフデータベースに接続できます。

- ファセットフィルタを使用してデータの概要をすばやく表示したり、検索バーにテキストを入力してデータを検索したりできます。

- ノードとエッジの接続をインタラクティブに調べることもできます。隣接するノードを表示してオブジェクト同士の関係を確認したり、ドリルダウンしてエッジやプロパティを視覚的に調べたりできます。
- また、グラフのレイアウト、色、アイコン、およびノードとエッジに表示するデフォルトプロパティをカスタマイズすることもできます。RDF グラフでは、リソース URI の名前空間もカスタマイズできます。
- グラフデータを含むレポートやプレゼンテーションでは、作成したビューを高解像度の PNG 形式で構成して保存できます。関連データを CSV または JSON ファイルにダウンロードして、さらに処理することもできます。

Neptune グラフノートブックでのグラフエクスプローラーの使用

Neptune でグラフエクスプローラーを使用する最も簡単な方法は、Neptune グラフノートブックにあります。

Neptune ワークベンチを使用して Neptune ノートブックをホストすると、グラフエクスプローラーはノートブックと共に自動的にデプロイされ、Neptune に接続されます。

ノートブックを作成したら、Neptune コンソールに移動してグラフエクスプローラーを起動します。

1. Neptune に移動します。
2. [ノートブック] で、ノートブックを選択します。
3. [アクション] で [グラフエクスプローラーを開く] を選択します。

AWS Fargate 上の Amazon ECS でグラフエクスプローラーを実行して Neptune に接続する方法

また、グラフエクスプローラー Docker イメージを構築し、グラフエクスプローラー GitHub プロジェクト の read-me の開始方法セクションで説明されているように、ローカルマシンまたは Amazon Elastic Compute Cloud (Amazon EC2) や Amazon Elastic Container Service (Amazon ECS) などのホストサービスで実行することもできます。 https://github.com/aws/graph-explorer#getting-started

例として、このセクションでは、 で Amazon ECS で graph-explorer を実行する step-by-step 手順を示します AWS Fargate。

1. IAM ロールを作成して、それに新しいポリシーをアタッチします。

   - AmazonECSTaskExecutionRolePolicy
   - CloudWatchLogsFullアクセス

   ロール名は 1 分以内に使えるようにしておいてください。
2. インフラストラクチャを FARGATE に設定し、以下のネットワークオプションを使用して Amazon ECS クラスターを作成します。

   - VPC: Neptune データベースが配置されている VPC に設定します。
   - Subnets: その VPC のパブリックサブネットに設定します (その他はすべて削除)。
3. 次のように新しい JSON タスク定義を作成します。

```
{
  "family": "explorer-test",
  "containerDefinitions": [
    {
      "name": "graph-explorer",
      "image": "public.ecr.aws/neptune/graph-explorer:latest",
      "cpu": 0,
      "portMappings": [
        {
          "name": "graph-explorer-80-tcp",
          "containerPort": 80,
          "hostPort": 80,
          "protocol": "tcp",
          "appProtocol": "http"
        },
        {
          "name": "graph-explorer-443-tcp",
          "containerPort": 443,
          "hostPort": 443,
          "protocol": "tcp",
          "appProtocol": "http"
        }
      ],
      "essential": true,
      "environment": [
        {
          "name": "HOST",
          "value": "localhost"
```

```
        }
      ],
      "mountPoints": [],
      "volumesFrom": [],
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-create-group": "true",
          "awslogs-group": "/ecs/graph-explorer",
          "awslogs-region": "{region}",
          "awslogs-stream-prefix": "ecs"
        }
      }
    }
  ],
  "taskRoleArn": "arn:aws:iam::{account_no}:role/{role_name_from_step_1}",
  "executionRoleArn": "arn:aws:iam::{account_no}:role/{role_name_from_step_1}",
  "networkMode": "awsvpc",
  "requiresCompatibilities": [
    "FARGATE"
  ],
  "cpu": "1024",
  "memory": "3072",
  "runtimePlatform": {
    "cpuArchitecture": "X86_64",
    "operatingSystemFamily": "LINUX"
  }
}
```

4. 以下のフィールドを除き、デフォルト設定を使用して新しいタスクを開始します。

   - 環境
     - コンピューティングオプション => [起動タイプ]
   - Deployment configuration
     - アプリケーションタイプ => [タスク]
     - ファミリー => *(### JSON #####)*
     - リビジョン => *(##)*
   - ネットワーク
     - VPC => *(#### Neptune VPC)*
     - サブネット => *(VPC ############ – #########)*

- セキュリティグループ => [新しいセキュリティグループの作成]
- セキュリティグループ名 => graph-explorer
- セキュリティグループの説明 = graph-explorer にアクセスするためのセキュリティグループ
- セキュリティグループのインバウンドルール =>
  1. 80 Anywhere
  2. 443 Anywhere

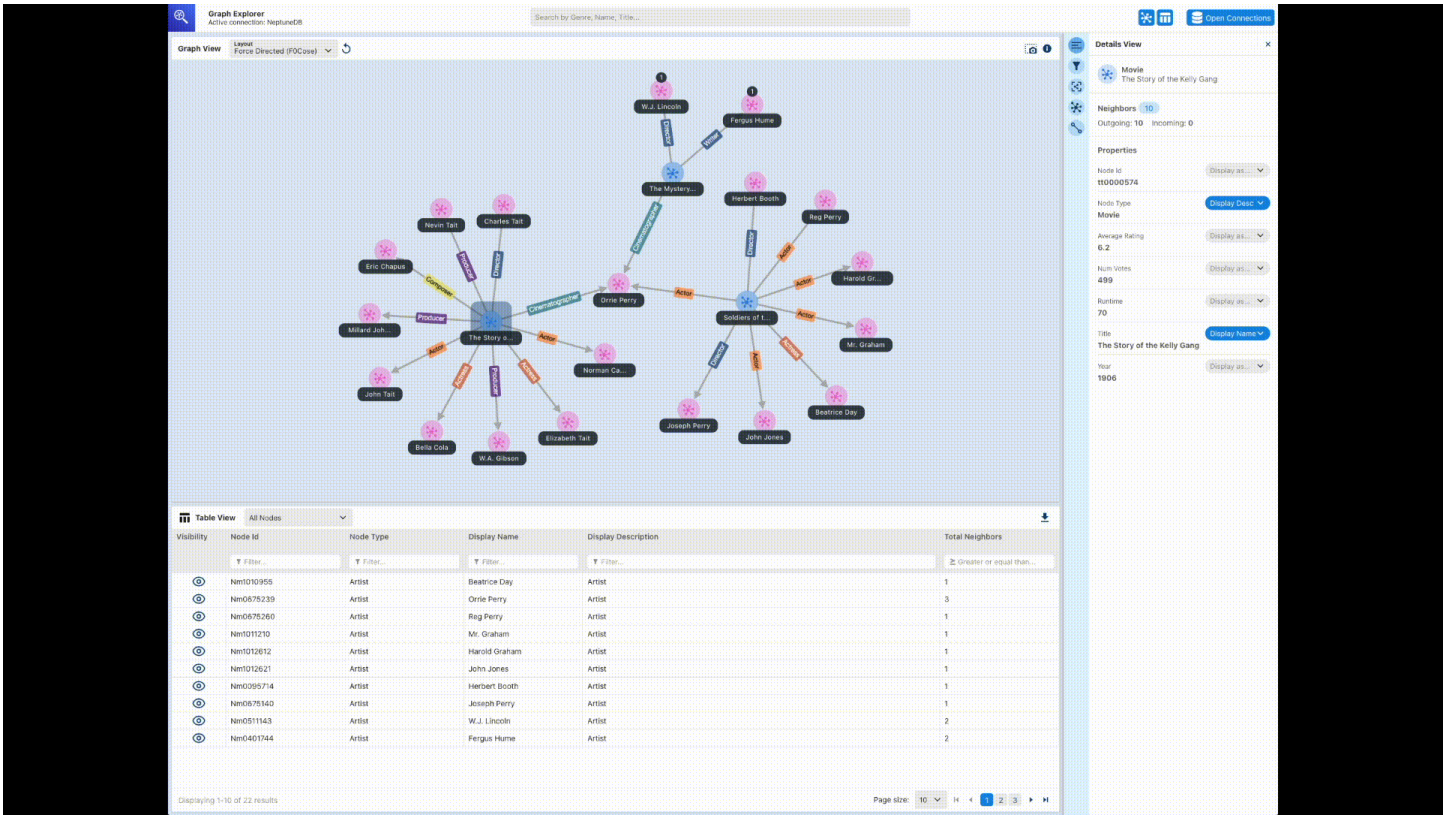
5. [作成] を選択します。
6. タスクが開始されたら、実行中のタスクのパブリック IP をコピーし、`https://(your public IP)/explorer` に移動します。
7. 生成された認識されない証明書を使用するリスクを冒すか、キーチェーンに追加してください。
8. これで、Neptune に接続を追加できるようになりました。プロパティグラフ (LPG) または RDF 用に新しい接続を作成し、以下のフィールドを設定します。

```
Using proxy server => true
Public or Proxy Endpoint => https://(your public IP address)
Graph connection URL => https://(your Neptune endpoint):8182
```

これで、接続されているはずです。

グラフエクスプローラーのデモンストレーション

この短いビデオでは、グラフエクスプローラーを使用してグラフデータを簡単に可視化する方法を紹介しています。



Tom Sawyer ソフトウェア

Tom Sawyer Perspectives は、Amazon Neptune に保存されているデータを対象とした、ローコードのグラフとデータの可視化および分析開発プラットフォームです。設計とプレビューの統合インターフェイスと豊富な API ライブラリにより、プロダクション品質のカスタムビジュアライゼーションアプリケーションを迅速に作成できます。 point-and-click デザイナーインターフェイスと 30 の組み込み分析アルゴリズムを使用すると、アプリケーションを設計および開発して、数十のソースからフェデレーションされたデータに関するインサイトを得ることができます。

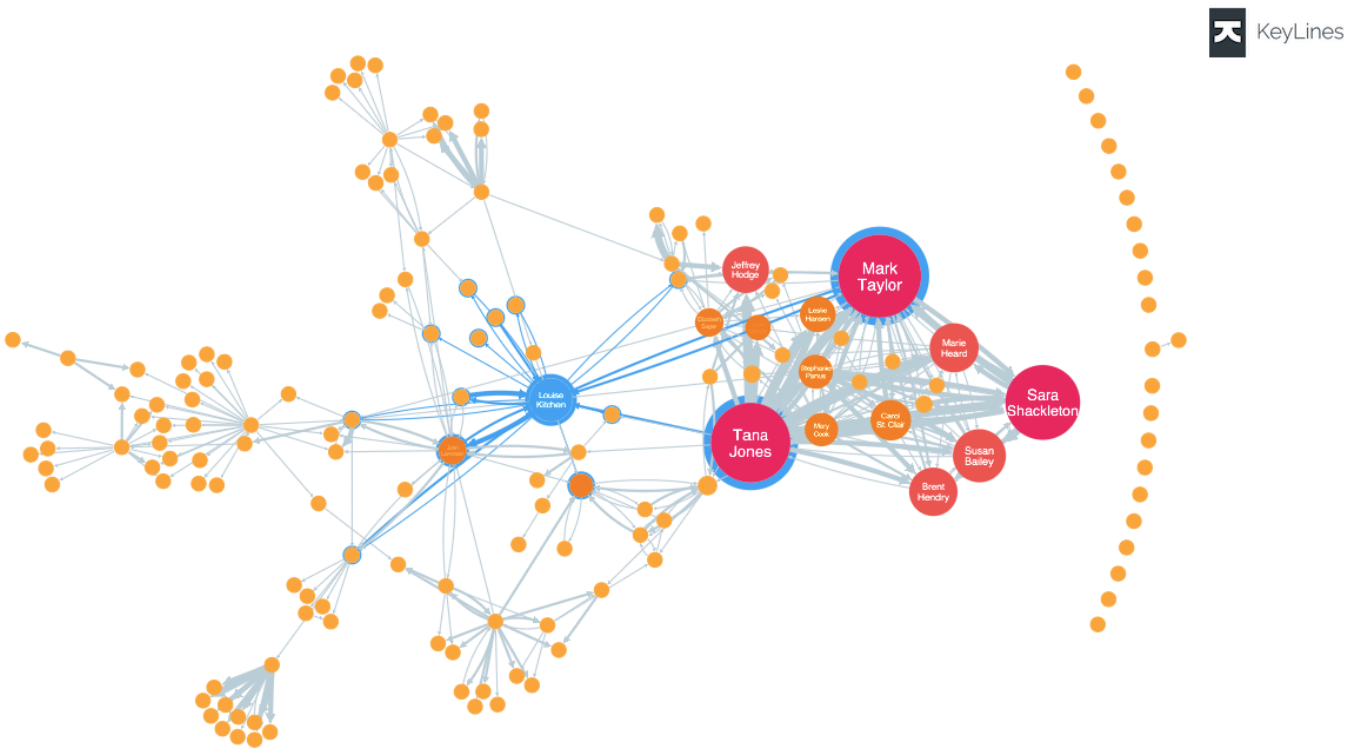
Tom Sawyer Graph Database Browser を使用すると、Amazon Neptune のデータを簡単に可視化して分析できます。クエリ言語やスキーマに関する幅広い知識がなくても、データ内の接続を確認して理解することができます。選択したノードの隣接ノードを読み込み、必要な方向にビジュアライゼーションを構築するだけで、技術的な知識がなくてもデータを操作できます。また、5 つのユニークなグラフレイアウトを活用して、最も意味のある方法でグラフを表示したり、中心性、クラスタリング、経路探索の分析を適用して、これまで見えなかったパターンを明らかにすることもできます。グラフデータベースブラウザと Neptune の統合の例については、こちらのブログ記事をご覧ください。Graph Database Browser の無料トライアルを開始するには、AWS Marketplace にアクセスしてください。

Cambridge Intelligence

Cambridge Intelligence は、Amazon Neptune データを探索して理解するためのデータ可視化テクノロジーを提供します。グラフ視覚化ツールキット (KeyLines JavaScript デベロッパー向けと React デベロッパーReGraph向け) は、ウェブアプリケーション用にインタラクティブでカスタマイズ可能なツールを簡単に構築できます。これらのツールキットは、WebGL と HTML5 Canvas を活用して高速なパフォーマンスを実現し、高度なグラフ分析機能をサポートし、柔軟性とスケーラビリティを安全で堅牢なアーキテクチャと組み合わせています。これらの SDK は、Neptune Gremlin と RDF データの両方で動作します。

Gremlin データ、SPARQL データ、および Neptune アーキテクチャに関するこれらの統合チュートリアルをご覧ください。

KeyLines 視覚化の例を次に示します。

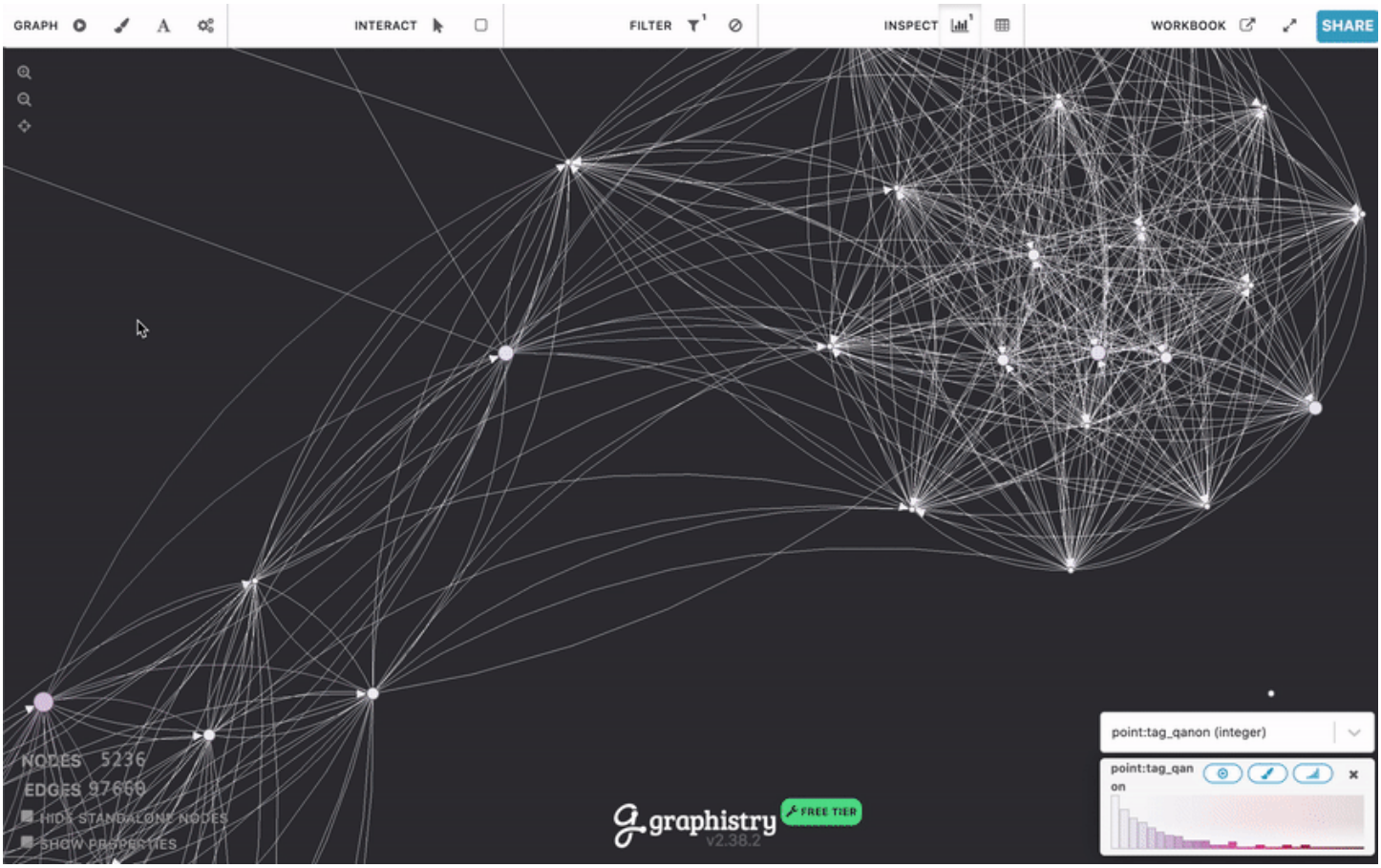


Graphistry

Graphistry は、GPU アクセラレーションを活用して豊かなビジュアル体験を実現するビジュアルグラフインテリジェンスプラットフォームです。ファイルやデータベースのコード不要の探索から、Jupyter ノートブックや Streamlit ダッシュボードの共有、独自のアプリでの埋め込み API の使用まで、さまざまな機能を使用して Graphistry でチームで共同作業を行うことができます。

graph-app-kit を設定して起動し、わずか数行のコードを変更するだけで、コーディングの手間がかからず、完全にインタラクティブなダッシュボードを使い始めることができます。Graphistry と Neptune を使用して初めてダッシュボードを作成する手順については、「こちらのブログ記事」をご覧ください。Neptune demo を試すこともできますPyGraphistry。 PyGraphistry はノートブック用の Python ビジュアルグラフ分析ライブラリです。Neptune PyGraphistry デモについては、このチュートリアルノートブックを参照してください。

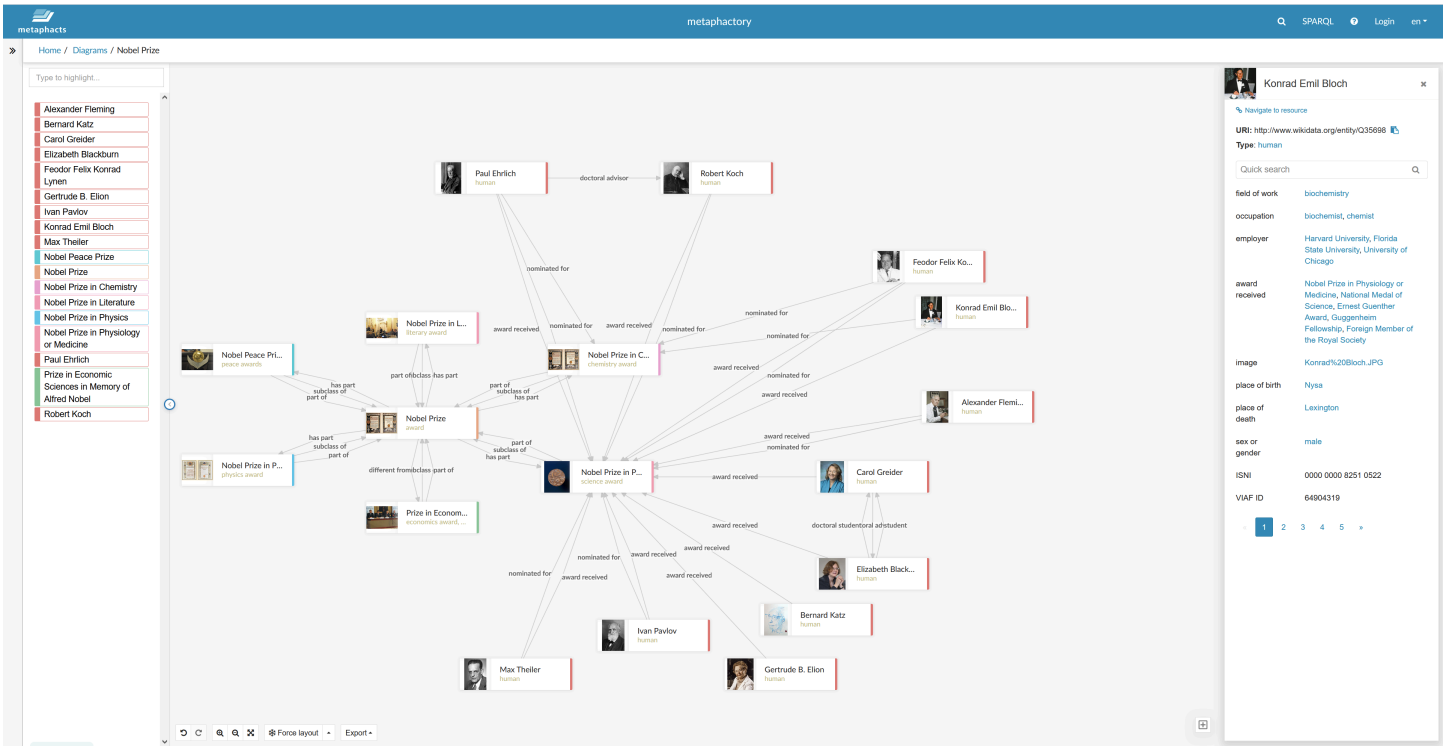
開始するには、 AWS Marketplace の「Graphistry」を参照してください。



metaphacts

metaphacts は、グラフデータの記述とクエリ、ナレッジグラフの可視化と操作のための柔軟でオープンなプラットフォームを提供します。metaphactory を使用すると、RDF データモデルを使用して、Neptune のナレッジグラフの上に可視化やダッシュボードなどのインタラクティブな Web アプリケーションを構築できます。metaphactory プラットフォームは、データローディング用のUI、OWL と SHACL をサポートするビジュアルオントロジーモデリングインターフェイス、SPARQL クエリ UI とクエリカタログ、およびグラフの探索、可視化、検索、オーサリングのためのリッチな Web コンポーネントのセットなどで、ローコード開発環境をサポートします。

metaphactory の可視化の例を次に示します。



このプラットフォームは、エンジニアリング、製造、製薬、ライフサイエンス、金融、保険などの分野向けに設計され、生産的に使用されています。ソリューションアーキテクチャのサンプルについては、こちらのブログ記事をご覧ください。

metaphactory の無料トライアルを始めるには、AWS Marketplace にアクセスしてください。

G.V( )

G.V() は、デベロッパーやデータアナリスト向けの強力な Gremlin 統合開発環境 (IDE) ツールです。これを使用すると、Neptune のグラフデータをインタラクティブにクエリ、可視化、更新できます。G.V() には Gremlin 言語オートコンプリート機能が組み込まれており、グラフデータモデルに基づいてクエリを入力する際の提案とドキュメントを提供します。

Gremlin のクエリデバッグ機能を使用して、グラフ探索プロセスを詳細に記述、デバッグ、テスト、分析することもできます。

OpenAI を搭載した自然言語処理を使用すると、G.V() はテキストプロンプトからグラフデータスキーマに正確な Gremlin クエリを生成し、自然言語でデータをクエリできます。

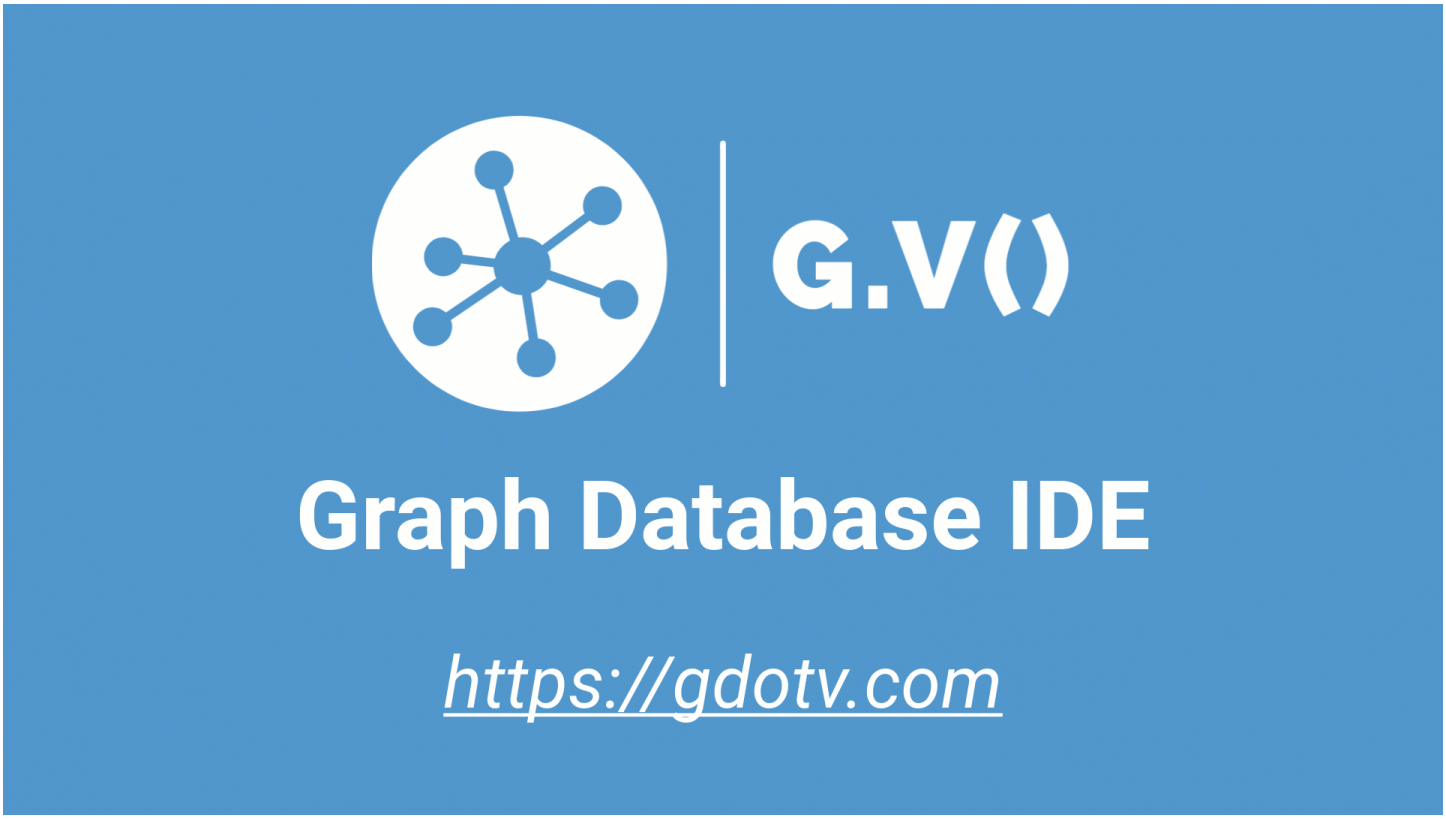
Graph Data Explorer を使用すると、グラフをナビゲートおよび変更して、新しいグラフ構造をすばやく設計し、既存のグラフ構造を維持できます。

G.V() には、クエリ出力を解釈し、グラフをインタラクティブにナビゲートするのに役立つクエリ結果用の複数の視覚化形式が用意されています。これらには、テーブル、グラフ、JSON、Gremlin コンソールの出力形式が含まれます。

G.V() は Amazon Neptune と完全に互換性があり、スロークエリや監査ログのインサイト、IAM 認証のサポートなど、Amazon Neptune 専用の多くの追加機能を提供します。詳細については、「」のドキュメントを参照してください。

G.V() は継続的に進化しており、毎月新機能を受け取ります。G.V() の詳細については、G.V() ウェブサイト にアクセスして、無料トライアルを開始してください。

以下の G.V() のデモを実際にご覧ください。



リンク

Linkurious は、技術系ユーザーと非技術系ユーザーの両方、およびさまざまなユースケースにさまざまなグラフインテリジェンスソリューションを提供します。

Linkurious Enterprise Explorer は、 day-to-day アクティビティの要求に追いついてデータ駆動型のプロフェッショナルが簡単に大きなことを行うのに役立つ、チーム向けに構築された off-the-shelf グラフの視覚化および分析ソフトウェアです。完全に設定可能で使いやすいため、ニーズに簡単に適応で

き、初心者や上級ユーザーが AWS Neptune でデータをすばやく視覚化したり、データのサイズや複雑さに関係なくデータセットを直感的に探索したり、チームレベルやエンタープライズレベルでシームレスにコラボレーションしたりできます。

Linkurious Enterprise Watchtower は、Linkurious Enterprise Explorer の機能を活用し、革新的な検出およびケース管理機能を追加して、グラフテクノロジーを活用した統合検出および調査ソフトウェアを提供します。一方、Neptune データベースと Neptune Analytics を活用するアラートを設定して、複雑な接続データに異常やパターンを自動的に表示することができます。一方、ケース管理機能とコラボレーション機能を組み合わせて、チームが調査ワークフローを効率的に管理できるようにします。

Ogma は、アプリケーションの強力で大規模なインタラクティブグラフの視覚化を開発するのに役立つ商用 JavaScript ライブラリです。WebGL レンダリングと高性能レイアウトを活用して、ユーザーは数千のノードやエッジを数秒で表示および操作できます。また、アプリケーションをカスタマイズし、豊富なユーザーエクスペリエンスを作成するためのさまざまな機能も提供します。最後に、チュートリアル 、多数の例https://doc.linkurious.com/ogma/latest/examples/transport-network.html、インタラクティブなプレイグラウンド などの包括的なドキュメントとツールが用意されています。

開始するには、Linkurious Enterprise または Ogma の 30 日間の無料トライアルをリクエストしてください。

Neptune DB クラスターからデータをエクスポートする

Neptune DB クラスターからデータをエクスポートするには、いくつかの良い方法があります。

- 少量のデータについては、1 つまたは複数のクエリの結果を使用するだけです。
- RDF データについては、グラフストアプロトコル (GSP) により簡単にエクスポートすることができます。例:

```
curl --request GET \
  'https://your-neptune-endpoint:port/sparql/gsp/?graph=http%3A//www.example.com/named/graph'
```

- Neptune データをエクスポートするための強力で柔軟なオープンソースツールもあります。すなわち `neptune-export`です。次のセクションでは、このツールの機能とその使い方について説明します。

トピック

- neptune-export を使用する
- Neptune-Export サービスを使用して Neptune データをエクスポートする
- neptune-export コマンドラインツールを使用して Neptune からデータをエクスポートする
- Neptune-Export と neptune-export でエクスポートされたファイル
- Neptune エクスポートプロセスを制御するために使用されるパラメーター
- Neptune エクスポートプロセスのトラブルシューティング

neptune-export を使用する

2つの異なる方法オープンソース neptune-export ツールを使えます。

- Neptune-Export サービスとして。 Neptune-Export サービスを使用して Neptune からデータをエクスポートする場合、REST API を使用してエクスポートジョブをトリガーおよび監視します。
- **neptune-export** Java コマンドラインユーティリティとして。 このコマンドラインツールを使用して Neptune データをエクスポートするには、Neptune DB クラスターがアクセス可能な環境で実行する必要があります。

Neptune-Exportサービスと neptune-export コマンドラインツールの両方は、Amazon S3 サーバー側の暗号化 (SSE-S3) により暗号化された Amazon Simple Storage Service (Amazon S3) にデータを公開します。

> **Note**
>
> すべての Amazon S3 バケットでアクセスログ記録を有効にして、それらのバケットへのすべてのアクセスを監査できるようにするのがベストプラクティスです。

エクスポート中にデータが変更されている Neptune DB クラスターからデータをエクスポートしようとすると、エクスポートされたデータの整合性は保証されません。つまり、エクスポートジョブの進行中にクラスターが書き込みトラフィックを処理している場合、エクスポートされたデータに不整合が生じる可能性があります。これは、クラスタ内のプライマリインスタンスからエクスポートするか、1 つ以上のリードレプリカからエクスポートするかにかかわらず当てはまります。

エクスポートされたデータの整合性を保証する最善策は、DB クラスターのクローンからエクスポートすることです。これにより、エクスポートツールにデータの静的バージョンが提供され、エクスポートジョブが元の DB クラスター内のクエリの速度を低下させないようにします。

これを簡単にするために、エクスポートジョブをトリガーするときにソース DB クラスターのクローンを作成することを指定できます。そうした場合、エクスポートプロセスによって自動的にクローンが作成され、エクスポートに使用され、エクスポートが完了すると削除されます。

Neptune-Export サービスを使用して Neptune データをエクスポートする

次の手順を使用して、Neptune-Export サービスを使用して Neptune DB クラスターから Amazon S3 にデータをエクスポートできます。

Neptune-Export サービスのインストール

AWS CloudFormation テンプレートを使用してスタックを作成します。

Neptune-Export サービスをインストールするには

1. AWS CloudFormation コンソールで AWS CloudFormation スタックを起動するには、以下の表でいずれかの [Launch Stack] (スタックの起動) ボタンを選択します。

リージョン	ビュー	デザイナーで表示	起動する
米国東部 (バージニア北部)	表示	デザイナーで表示	Launch Stack
米国東部 (オハイオ)	表示	デザイナーで表示	Launch Stack
米国西部 (北カリフォルニア)	表示	デザイナーで表示	Launch Stack
米国西部 (オレゴン)	表示	デザイナーで表示	Launch Stack
カナダ (中部)	表示	デザイナーで表示	Launch Stack
南米 (サンパウロ)	表示	デザイナーで表示	Launch Stack
欧州 (ストックホルム)	表示	デザイナーで表示	Launch Stack
欧州 (アイルランド)	表示	デザイナーで表示	Launch Stack

リージョン	ビュー	デザイナーで表示	起動する
欧州 (ロンドン)	表示	デザイナーで表示	
欧州 (パリ)	表示	デザイナーで表示	
欧州 (フランクフルト)	表示	デザイナーで表示	
中東 (バーレーン)	表示	デザイナーで表示	
中東 (アラブ首長国連邦)	表示	デザイナーで表示	
イスラエル (テルアビブ)	表示	デザイナーで表示	
アフリカ (ケープタウン)	表示	デザイナーで表示	
アジアパシフィック (香港)	表示	デザイナーで表示	
アジアパシフィック (東京)	表示	デザイナーで表示	
アジアパシフィック (ソウル)	表示	デザイナーで表示	
アジアパシフィック (シンガポール)	表示	デザイナーで表示	
アジアパシフィック (シドニー)	表示	デザイナーで表示	
アジアパシフィック (ムンバイ)	表示	デザイナーで表示	

リージョン	ビュー	デザイナーで表示	起動する
中国 (北京)	表示	デザイナーで表示	
中国 (寧夏)	表示	デザイナーで表示	
AWS GovCloud (米国西部)	表示	デザイナーで表示	
AWS GovCloud (米国東部)	表示	デザイナーで表示	

2. [Select Template] ページで、[Next] を選択します。
3. [Specify Details] (詳細指定)ページで、テンプレートでパラメータを次のように設定します。

   - **VPC** — Neptune-Export サービスをセットアップする最も簡単な方法は、Neptune データベースと同じ Amazon VPC にインストールすることです。別の VPC にインストールする場合は、VPC ピアリング接続を使用して Neptune DB クラスターの VPC と Neptune エクスポートサービス VPC 間の接続を確立します。
   - **Subnet1** — Neptune-Export サービスは、サブネットからインターネットへのアウトバウンド IPv4 HTTPS トラフィックを許可する VPC 内のサブネットにインストールする必要があります。これは、Neptune-Export サービスが AWS Batch API を呼び出して、エクスポートジョブを作成して実行します。

     Neptune ドキュメントの DB クラスターを作成する ページの CloudFormation テンプレートを使用して Neptune クラスターを作成した場合、このスタックからの `PrivateSubnet1` および `PrivateSubnet2` 出力を使用して、これと、次のパラメータを設定します。
   - **Subnet2** — そこからインターネットへのアウトバウンド IPv4 HTTPS トラフィックを許可する VPC 内の第 2 サブネット。
   - **EnableIAM** — これを `true` に設定して AWS Identity and Access Management (IAM) を使用して Neptune-Endpoint API を保護します。そのようにすることをお勧めします。

     IAM 認証を有効にする場合は、`Sigv4` エンドポイントへのすべての HTTPS リクエストに署名します。ユーザーに代わってリクエストに署名するには、awscurl のようなツールを使用できます。

- **`VPCOnly`** — これを `true` に設定すると、エクスポートエンドポイントを VPC 専用となり、これにより、Neptune-Export サービスがインストールされている VPC 内からしかアクセスできなくなります。これにより、Neptune-Export API がその VPC 内からしか使用できないように制限されます。

  `VPCOnly` を `true` に設定することをお勧めします。

- **`NumOfFilesULimit`** — `ulimits` コンテナプロパティで `nofile` に 10,000 から 1,000,000 までの値を指定します。デフォルトは 10,000 です。グラフに固有のラベルが多数含まれている場合を除いて、デフォルトのままにしておくことをお勧めします。

- **`PrivateDnsEnabled`** (ブール) - プライベートホストゾーンを指定された VPC に関連付けるかどうかを示します。デフォルト値は `true` です。

  このフラグを有効にして VPC エンドポイントを作成すると、すべての API ゲートウェイトラフィックは VPC エンドポイントを経由してルーティングされ、パブリック API ゲートウェイエンドポイントの呼び出しは無効になります。`PrivateDnsEnabled` を `false` に設定した場合、パブリック API ゲートウェイエンドポイントは有効になりますが、Neptune エクスポートサービスはプライベート DNS エンドポイント経由で接続できません。その場合、こちらで説明するように、VPC エンドポイントのパブリック DNS エンドポイントを使用してエクスポートサービスを呼び出すことができます。

4. [Next] (次へ) をクリックします。
5. [Options(オプション)] ページで、[Next(次へ)] を選択します。
6. [確認] ページで、AWS CloudFormation によって IAM リソースが作成されることを確認する最初のチェックボックスをオンにします。新しいスタックの `CAPABILITY_AUTO_EXPAND` を確認する 2 つ目のチェックボックスをオンにします。

> **Note**
>
> `CAPABILITY_AUTO_EXPAND` は、スタックの作成時に事前の確認なしにマクロが展開されることを明示的に確認します。ユーザーは、処理されたテンプレートから変更セットを作成することが多いため、実際にスタックを作成する前にマクロによって行われた変更を確認できます。詳細については、「AWS CloudFormation CreateStack API」を参照してください。

次に [作成] を選択します。

Neptune エクスポートからNeptune へのアクセスを有効にする

Neptune-Export のインストールが完了したら、Neptune VPC セキュリティグループを更新し、Neptune-Export からのアクセスを許可します。Neptune-Export AWS CloudFormation スタックが作成されると、出力タブには `NeptuneExportSecurityGroup` ID が含まれています。この Neptune-Export セキュリティグループからのアクセスを許可するように、Neptune VPC セキュリティグループを更新します。

VPC ベースの EC2 インスタンスから Neptune-Export エンドポイントへのアクセスを有効にする

Neptune-Export　エンドポイントを VPC 専用にすると、これにより、Neptune-Export サービスがインストールされている VPC 内からしかアクセスできなくなります。Neptune-Export API 呼び出しを実行できる VPC 内の Amazon EC2 インスタンスからの接続を許可するには、AWS CloudFormation スタックによって作成された `NeptuneExportSecurityGroup` を Amazon EC2 インスタンスにアタッチします。

Neptune-Export API を使用して Neptune-Export ジョブを実行する

AWS CloudFormation スタックの出力タブは `NeptuneExportApiUri` も含みます。Neptune-Export エンドポイントにリクエストを送信するたびに、この URI を使用します。

Export ジョブを実行する

- エクスポートを実行するユーザーまたはロールが `execute-api:Invoke` アクセス許可を付与されていることを確認してください。
- Neptune-Export をインストールしたときに AWS CloudFormation スタックの `EnableIAM` パラメータを `true` に設定すると、Neptune-Export API へのすべてのリクエストに `Sigv4` 署名しなければなりません。awscurl を使用して API にリクエストすることをお勧めします。ここでの例はすべて、IAM 認証が有効であることを前提としています。
- Neptune-Export をインストールしたときに AWS CloudFormation スタックの `VPCOnly` パラメータを `true` に設定すると、通常は VPC にある Amazon EC2 インスタンスから、VPC 内から Neptune-Export API を呼び出す必要があります。

データのエクスポートを開始するには、`command` および `outputS3Path` リクエストパラメータと `endpoint` エクスポートパラメータでリクエストを `NeptuneExportApiUri` エンドポイントに送信します。

以下は、Neptune からプロパティグラフデータをエクスポートし、それを Amazon S3 に発行するリクエストの例です。

```
curl \
  (your NeptuneExportApiUri) \
  -X POST \
  -H 'Content-Type: application/json' \
  -d '{
        "command": "export-pg",
        "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export",
        "params": { "endpoint": "(your Neptune endpoint DNS name)" }
      }'
```

同様に、Neptune から Amazon S3 に RDF データをエクスポートするリクエストの例を次に示します。

```
curl \
  (your NeptuneExportApiUri) \
  -X POST \
  -H 'Content-Type: application/json' \
  -d '{
        "command": "export-rdf",
        "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export",
        "params": { "endpoint": "(your Neptune endpoint DNS name)" }
      }'
```

command リクエストパラメーターを省略すると、デフォルトでは、Neptune-Export は Neptune からプロパティグラフデータをエクスポートしようとします。

前のコマンドが正常に実行された場合、出力は次のようになります。

```
{
  "jobName": "neptune-export-abc12345-1589808577790",
  "jobId": "c86258f7-a9c9-4f8c-8f4c-bbfe76d51c8f"
}
```

開始したばかりのエクスポートジョブを監視する

実行中のジョブを監視するには、次のようにその jobID を `NeptuneExportApiUri` に添付します。

```
curl \
```

```
  (your NeptuneExportApiUri)(the job ID)
```

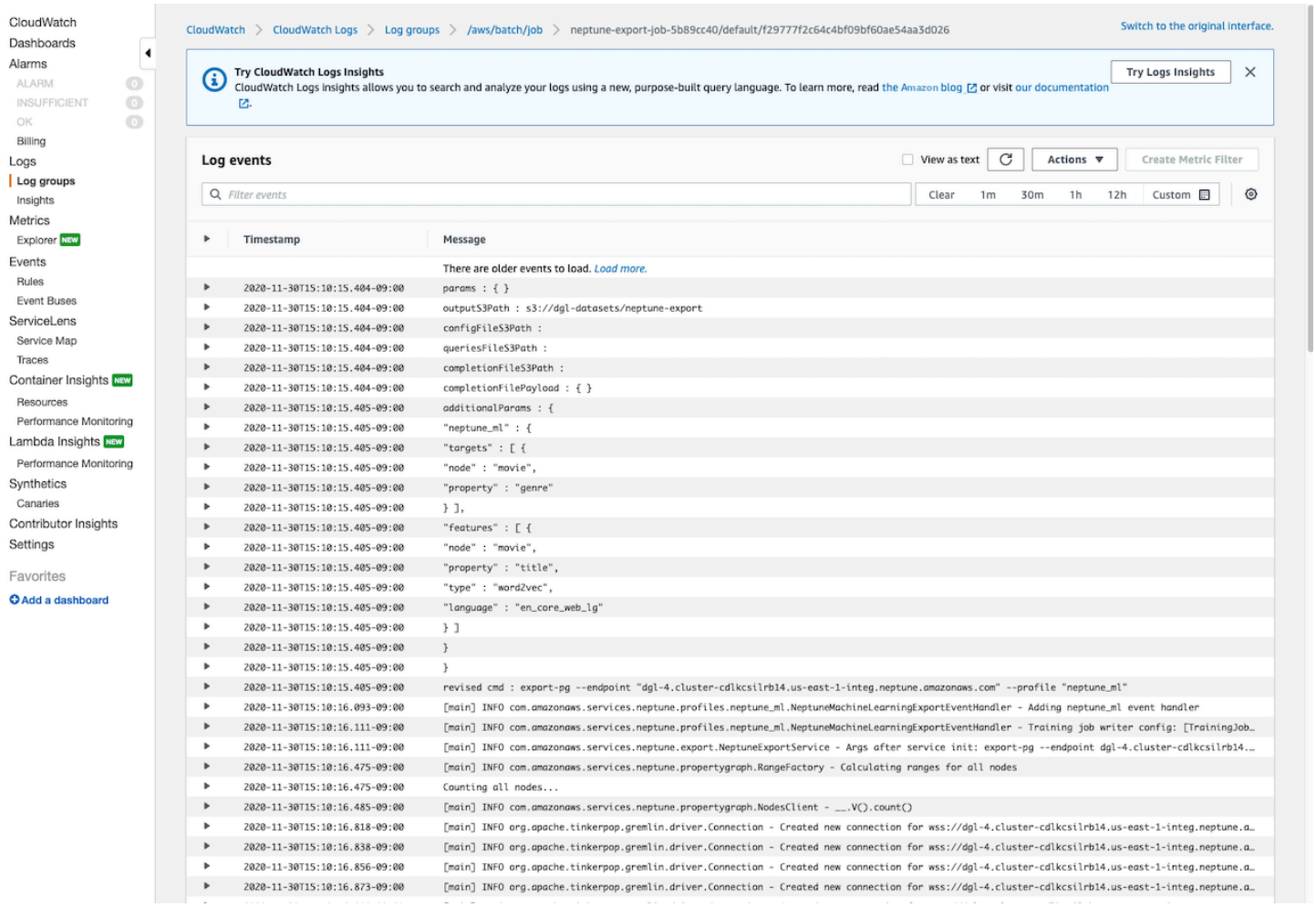
サービスがエクスポートジョブをまだ開始していない場合、応答は次のようになります。

```
{
  "jobId": "c86258f7-a9c9-4f8c-8f4c-bbfe76d51c8f",
  "status": "pending"
}
```

エクスポートジョブの開始後にコマンドを繰り返すと、応答は次のようになります。

```
{
  "jobId": "c86258f7-a9c9-4f8c-8f4c-bbfe76d51c8f",
  "status": "running",
  "logs": "https://us-east-1.console.aws.amazon.com/cloudwatch/home?..."
}
```

ステータスコールで提供される URI を使用して CloudWatch Logs でログを開くと、エクスポートの進行状況を詳細に監視できます。



実行中のエクスポートジョブをキャンセルする

AWS Management Console を使用して実行中のエクスポートジョブをキャンセルするには

1. AWS Batchコンソールをhttps://console.aws.amazon.com/batch/ で 開きます。
2. ジョブ を選択します。
3. `jobID` に基づいて、キャンセルする実行中のジョブを見つけます。
4. ジョブのキャンセルを選択します。

Neptune export API を使用して実行中のエクスポートジョブをキャンセルするには

次のように `HTTP DELETE` リクエストを `NeptuneExportApiUri` へ `jobID` を添付して送信します。

```
curl -X DELETE \
```

```
  (your NeptuneExportApiUri) (the job ID)
```

neptune-export コマンドラインツールを使用して Neptune からデータをエクスポートする

次の手順を使用して、neptune-export コマンドラインユーティリティを使用して Neptune DB クラスターから Amazon S3 にデータをエクスポートできます。

neptune-export コマンドラインユーティリティを使用するための前提条件

開始する前に

- JDK のバージョン 8 がある — Java SE Development Kit (JDK) バージョン 8 をインストールしておく必要があります。
- neptune-export ユーティリティをダウンロードする — neptune-export.jar ファイルをダウンロードしてインストールします。
- **neptune-export** が Neptune VPC にアクセスできることを確認してください — Neptune DB クラスターがある VPC にアクセスできる場所から neptune-export を実行します。

  たとえば、Neptune VPC 内の Amazon EC2 インスタンス、または Neptune VPC とピアリングされている別の VPC、または別の踏み台ホストで実行できます。
- VPC セキュリティグループが **neptune-export** にアクセス権を付与していることを確認します — Neptune VPC にアタッチされた VPC セキュリティグループが、neptune-export 環境に関連付けられている IP アドレスまたはセキュリティグループから DB クラスターへのアクセスを許可していることを確認します。
- 必要な IAM アクセス権限をセットアップする — データベースに有効な AWS Identity and Access Management (IAM) データベース認証がある場合、neptune-export を実行中のロールが Neptune への接続を許可する IAM ポリシーに関連付けられていることを確認してください。Neptune ポリシーの詳細については、IAM ポリシーの使用 を参照してください。

  クエリリクエストの clusterId エクスポートパラメータを使用したい場合、neptune-export を実行中のロールには、次の IAM アクセス権限が必要です。

  - rds:DescribeDBClusters
  - rds:DescribeDBInstances
  - rds:ListTagsForResource

クローンされたクラスターからエクスポートする場合は、neptune-export を実行中のロールには、次の IAM アクセス権限が必要です。

- rds:AddTagsToResource
- rds:DescribeDBClusters
- rds:DescribeDBInstances
- rds:ListTagsForResource
- rds:DescribeDBClusterParameters
- rds:DescribeDBParameters
- rds:ModifyDBParameterGroup
- rds:ModifyDBClusterParameterGroup
- rds:RestoreDBClusterToPointInTime
- rds:DeleteDBInstance
- rds:DeleteDBClusterParameterGroup
- rds:DeleteDBParameterGroup
- rds:DeleteDBCluster
- rds:CreateDBInstance
- rds:CreateDBClusterParameterGroup
- rds:CreateDBParameterGroup

エクスポートされたデータを Amazon S3 に発行するには、neptune-export を実行中のロールには、Amazon S3 ロケーションに対して次の IAM アクセス権限が必要です。

- s3:PutObject
- s3:PutObjectTagging
- s3:GetObject

- **SERVICE_REGION 環境変数を設定する** — SERVICE_REGION 環境変数を設定して DB クラスターが配置されているリージョンを識別します (リージョン識別子のリストについてはNeptune に接続するを参照)。

neptune-export ユーティリティを実行しエクスポート操作を開始する

次のコマンドを使用して、コマンドラインから neptune-export を実行し、エクスポート操作を開始します。

```
java -jar neptune-export.jar nesvc \
  --root-path (path to a local directory) \
  --json (the JSON file that defines the export)
```

コマンドには、2 つのパラメータがあります。

エクスポート開始時の neptune-Export のパラメータ

- **`--root-path`** — エクスポートファイルが Amazon S3 に発行される前に書き込まれるローカルディレクトリへのパス。
- **`--json`** — エクスポートを定義する JSON オブジェクト。

`neptune-export` コマンドラインユーティリティを使用したコマンド例

ソース DB クラスターから直接プロパティグラフデータをエクスポートするには、次の手順を実行します。

```
java -jar neptune-export.jar nesvc \
  --root-path /home/ec2-user/neptune-export \
  --json '{
            "command": "export-pg",
            "outputS3Path" : "s3://(your Amazon S3 bucket)/neptune-export",
            "params": {
              "endpoint" : "(your neptune DB cluster endpoint)"
            }
          }'
```

ソース DB クラスターから直接 RDF データをエクスポートするには、次の手順を実行します。

```
java -jar neptune-export.jar nesvc \
  --root-path /home/ec2-user/neptune-export \
  --json '{
            "command": "export-rdf",
            "outputS3Path" : "s3://(your Amazon S3 bucket)/neptune-export",
            "params": {
              "endpoint" : "(your neptune DB cluster endpoint)"
            }
          }'
```

command リクエストパラメーターを省略した場合、デフォルトでは、neptune-export ユーティリティは Neptune からプロパティグラフデータをエクスポートします。

DB クラスターのクローンからエクスポートするには、次の手順を実行します。

```
java -jar neptune-export.jar nesvc \
  --root-path /home/ec2-user/neptune-export \
  --json '{
            "command": "export-pg",
            "outputS3Path" : "s3://(your Amazon S3 bucket)/neptune-export",
            "params": {
              "endpoint" : "(your neptune DB cluster endpoint)",
              "cloneCluster" : true
            }
          }'
```

IAM 認証を使用して DB クラスターからエクスポートするには、次の手順を実行します。

```
java -jar neptune-export.jar nesvc \
  --root-path /home/ec2-user/neptune-export \
  --json '{
            "command": "export-pg",
            "outputS3Path" : "s3://(your Amazon S3 bucket)/neptune-export",
            "params": {
              "endpoint" : "(your neptune DB cluster endpoint)"
              "useIamAuth" : true
            }
          }'
```

Neptune-Export と **neptune-export** でエクスポートされたファイル

エクスポートが完了すると、エクスポートファイルは指定した Amazon S3 の場所に発行されます。Amazon S3 に発行されたすべてのファイルは、Amazon S3 のサーバー側の暗号化を使用して暗号化されています (SSE-S3)。Amazon S3 に発行されたフォルダとファイルは、プロパティグラフまたは RDF データのどちらをエクスポートするかによって異なります。ファイルが発行される Amazon S3 の場所を開くと、次の内容が表示されます。

Amazon S3 のエクスポートされたファイルの場所

- **nodes/** — このフォルダには、カンマ区切り値 (CSV) または JSON 形式のノードデータファイルが含まれています。

  Neptune では、ノードは 1 つ以上のラベルを持つことができます。個々のラベルが異なるノード (または複数のラベルの組み合わせが異なる) は、異なるファイルに書き込まれます。つまり、ラベルの組み合わせが異なるノードのデータは個々のファイルに含まれません。ノードに複数のラベルがある場合、これらのラベルはファイルに割り当てられる前にアルファベット順にソートされます。
- **edges/** — このフォルダには、カンマ区切り値 (CSV) または JSON 形式のエッジデータファイルが含まれています。

  ノードファイルと同様に、エッジデータはラベルの組み合わせに基づいて異なるファイルに書き込まれます。モデルトレーニングの目的で、エッジのラベルとエッジの開始ノードと終了ノードのラベルの組み合わせに基づいて、エッジデータが異なるファイルに割り当てられます。
- **statements/** — このフォルダには、タートル、N クワッド、N トリプル、または JSON 形式の RDF データファイルが含まれています。
- **config.json** — このファイルには、エクスポートプロセスで推測されるグラフのスキーマが含まれています。
- **lastEventId.json** — このファイルには、データベースの Neptune ストリームの最終イベントの `commitNum` および `opNum` が含まれています。エクスポートプロセスでは、`includeLastEventId` エクスポートパラメータを `true` に設定した場合にのみこのファイルが含まれ、データのエクスポート元のデータベースには有効化された Neptune Stream があります。

Neptune エクスポートプロセスを制御するために使用されるパラメーター

Neptune-Export サービスか neptune-export コマンドラインユーティリティを使用している場合は、エクスポートを制御するために使用するパラメータはほとんど同じです。これらのオブジェクトには、コマンドライン上で Neptune-Export エンドポイントまたは neptune-export に渡される JSON オブジェクトが含まれています。

エクスポートプロセスに渡されるオブジェクトには、最大 5 つの最上位フィールドがあります。

```
-d '{
      "command" : "(either export-pg or export-rdf)",
      "outputS3Path" : "s3:/(your Amazon S3 bucket)/(path to the folder for exported
 data)",
      "jobsize" : "(for Neptune-Export service only)",
      "params" : { (a JSON object that contains export-process parameters) },
      "additionalParams": { (a JSON object that contains parameters for training
 configuration) }
   }'
```

目次

    - gremlinEdgeFilter を使用する例
    - filter、gremlinNodeFilter、nodeLabels、edgeLabels、scopeを組み合わせる
- RDF データのエクスポートのフィルタリング
    - rdfExportScope および sparql を使用して特定のエッジをエクスポートする
    - namedGraph を使用して単一の名前付きグラフをエクスポートする

command パラメータ

command 最上位パラメータは、プロパティグラフデータまたは RDF データのどちらをエクスポートするかを決定します。command パラメータを省略すると、エクスポートプロセスはデフォルトでプロパティグラフデータをエクスポートします。

- **export-pg** — プロパティグラフデータをエクスポートします。
- **export-rdf** — RDF データをエクスポートします。

outputS3Path パラメータ

outputS3Path 最上位パラメータは必須で、エクスポートされたファイルを公開できる Amazon S3 ロケーションの URI が含まれている必要があります。

```
  "outputS3Path" : "s3://(your Amazon S3 bucket)/(path to output folder)"
```

値は s3:// で始まっている必要があります。その後に有効なバケット名と、オプションでバケット内のフォルダパスが続きます。

jobSize パラメータ

jobSize 最上位パラメータは、neptune-export コマンドラインユーティリティではなく Neptune-Export サービスでのみ使用され、オプションです。これにより、開始するエクスポートジョブのサイズを特徴づけ、ジョブに費やされるコンピューティングリソースの量とその最大同時実行レベルを決定するのに役立ちます。

```
  "jobsize" : "(one of four size descriptors)"
```

4 つの有効サイズ記述子は次のとおりです。

- `small` — 最大同時実行数: 8 10 GB までのストレージボリュームに適しています。
- `medium` — 最大同時実行数: 32 100 GB までのストレージボリュームに適しています。
- `large` — 最大同時実行数: 64 100 GB を超えるが 1 TB 未満のストレージボリュームに適しています。
- `xlarge` — 最大同時実行数: 96 1 TB を超えるストレージボリュームに適しています。

デフォルトでは、Neptune-Export サービスで開始されたエクスポートは、`small` ジョブとして実行されます。

エクスポートのパフォーマンスは、`jobSize` 設定だけでなく、エクスポート元のデータベースインスタンスの数、各インスタンスのサイズ、およびジョブの有効な同時実行レベルにより変わります。

プロパティグラフのエクスポートでは、データベースインスタンスの数を cloneClusterReplicaカウント パラメータを使用して設定でき、ジョブの有効な同時実行レベルは、concurrency パラメータを使用して設定できます。

`params` オブジェクト。

`params` 最上位パラメータは、params 最上位 JSON オブジェクトにパラメータフィールドをエクスポートする で説明されているように、エクスポートプロセス自体を制御するために使用するパラメータを含む JSON オブジェクトです。`params` オブジェクトのフィールドの一部は、プロパティグラフのエクスポートに固有で、一部は RDF に固有です。

`additionalParams` オブジェクト。

`additionalParams` 最上位パラメータは JSON オブジェクトで、エクスポート後にデータに適用されるアクションを制御するために使用できるパラメータを含みます。現在、`additionalParams` は Neptune ML のトレーニングデータのエクスポートのみに使用されます。

params 最上位 JSON オブジェクトにパラメータフィールドをエクスポートする

Neptune エクスポート params JSON オブジェクトを使用すると、エクスポートされるデータの型と形式など、エクスポートを制御できます。

エクスポートパラメータ params オブジェクトで使用可能なフィールドのリスト

以下に、params オブジェクトで使用可能なすべての最上位フィールドを一覧表示します。1 つのオブジェクトには、これらのフィールドのサブセットのみが表示されます。

すべてのタイプのエクスポートに共通のフィールドのリスト

- cloneCluster
- cloneClusterInstanceType
- cloneClusterReplicaCount
- clusterId
- endpoint
- endpoints
- profile
- useIamAuth
- includeLastEventId

プロパティグラフエクスポートのフィールドリスト

- concurrency
- edgeLabels
- filter
- filterConfigFile
- gremlinFilter
- gremlinNodeFilter
- gremlinEdgeFilter
- format

- nodeLabels
- scope

RDF エクスポートのフィールドリスト

- format
- rdfExportScope
- sparql
- namedGraph

すべてのタイプのエクスポートに共通のフィールド

params の cloneCluster フィールド

(オプション) デフォルト: `false`。

そのファイルに `cloneCluster` パラメータを `true` と設定する場合、エクスポートプロセスでは DB クラスターの高速クローンが使用されます。

```
  "cloneCluster" : true
```

デフォルトでは、エクスポートプロセスは、`endpoint`、`endpoints` または `clusterId` パラメータを使用して指定した DB クラスターからデータをエクスポートします。ただし、エクスポートの実行中に DB クラスターが使用中であり、データが変更されている場合、エクスポートプロセスでエクスポートされるデータの整合性を保証することはできません。

エクスポートされたデータの整合性を確保するには、代わりに `cloneCluster` パラメータを使用して DB クラスターの静的クローンからエクスポートします。

クローンされた DB クラスターは、ソース DB クラスターと同じ VPC 内に作成され、ソースのセキュリティグループ、サブネットグループ、IAM データベース認証設定を継承します。エクスポートが完了すると、Neptune はクローンされた DB クラスターを削除します。

デフォルトでは、クローンされた DB クラスターは、ソース DB クラスターのプライマリインスタンスと同じインスタンスタイプの単一のインスタンスで構成されます。クローン DB クラスターに使用されるインスタンスタイプを変更するには、`cloneClusterInstanceType` を使用して別のインスタンスタイプを指定します。

> **Note**
>
> cloneCluster オプションを使用せず、メイン DB クラスターから直接エクスポートする場合は、データのエクスポート元のインスタンスのタイムアウトを増やす必要がある場合があります。大規模なデータセットの場合、タイムアウトは数時間に設定する必要があります。

params の cloneClusterInstanceType フィールド

(オプション)

そのファイルに cloneCluster パラメータが存在すれば、true を用いることができます。cloneClusterInstanceType パラメータを使用して、クローン DB クラスターに使用するインスタンスタイプを指定します。

デフォルトでは、クローンされた DB クラスターは、ソース DB クラスターのプライマリインスタンスと同じインスタンスタイプの単一のインスタンスで構成されます。

```
  "cloneClusterInstanceType" : "(for example, r5.12xlarge)"
```

params の cloneClusterReplicaCount フィールド

(オプション)

cloneCluster パラメータが存在し true に設定されている場合、cloneClusterReplicaCount パラメータを使用して、クローン DB クラスター内に作成されるリードレプリカインスタンスの数を指定できます。

```
  "cloneClusterReplicaCount" : (for example, 3)
```

デフォルトでは、クローンされた DB クラスターは 1 つのプライマリインスタンスで構成されます。cloneClusterReplicaCount パラメータを使用して、作成する追加のリードレプリカインスタンスの数を指定できます。

params の clusterId フィールド

(オプション)

clusterId パラメータは、使用する DB クラスターの ID を指定します。

```
  "clusterId" : "(the ID of your DB cluster)"
```

clusterId パラメータを使用すると、エクスポートプロセスでは、その DB クラスターで使用可能なすべてのインスタンスを使用してデータを抽出します。

> **Note**
>
> パラメータ endpoint、endpoints および clusterId は相互に排他的です。そのうちの 1 つだけを使用します。

params の **endpoint** フィールド

(オプション)

endpoint を使用してエクスポートプロセスがデータを抽出するためにクエリを実行できる DB クラスター内の Neptune インスタンスのエンドポイントを指定します (エンドポイント接続 を参照)。これは DNS 名のみで、プロトコルやポートは含まれません。

```
  "endpoint" : "(a DNS endpoint of your DB cluster)"
```

クラスターまたはインスタンスエンドポイントを使用します。メインリーダーエンドポイントは使用しないでください。

> **Note**
>
> パラメータ endpoint、endpoints および clusterId は相互に排他的です。そのうちの 1 つだけを使用します。

params の **endpoints** フィールド

(オプション)

endpoints を使用してエクスポートプロセスでデータを抽出するためにクエリを実行できる DB クラスター内のエンドポイントの JSON 配列を指定します (エンドポイント接続 を参照)。これらは DNS 名のみであり、プロトコルやポートは含まれません。

```
  "endpoints": [
```

```
    "(one endpoint in your DB cluster)",
    "(another endpoint in your DB cluster)",
    "(a third endpoint in your DB cluster)"
    ]
```

クラスターに複数のインスタンス (プライマリと 1 つ以上のリードレプリカ) がある場合は、endpoints パラメータを使用して、これらのエンドポイントのリストにクエリを分散し、エクスポートパフォーマンスを向上することができます。

> **Note**
>
> パラメータ endpoint、endpoints および clusterId は相互に排他的です。そのうちの 1 つだけを使用します。

params の **profile** フィールド

(*neptune_ml* フィールドが *additionalParams* フィールドに存在しない限り、Neptune ML のトレーニングデータをエクスポートする必要があります)。

profile パラメータは、特定のワークロードに対して事前設定されたパラメータのセットを提供します。現在、エクスポートプロセスでは neptune_ml プロフィールのみをサポートしいています。

Neptune ML のトレーニングデータをエクスポートする場合は、次のパラメータを params オブジェクトに追加してください。

```
  "profile" : "neptune_ml"
```

params の **useIamAuth** フィールド

(オプション) デフォルト: false。

データのエクスポート元のデータベースに有効な IAM 認証がある場合、true に設定された useIamAuth パラメータを含める必要があります。

```
  "useIamAuth" : true
```

params の **includeLastEventId** フィールド

includeLastEventId を true 設定し、データのエクスポート元のデータベースに有効な Neptune Streams がある場合、エクスポートプロセスは lastEventId.json ファイルを指定したエクス

ポート場所に書き込みます。このファイルには、ストリーム内の最後のイベントの commitNum および opNum が含まれています。

```
"includeLastEventId" : true
```

エクスポートプロセスによって作成されたクローンデータベースは、親のストリーム設定を継承します。親でストリームが有効になっている場合、クローンでも同様にストリームが有効になります。クローン上のストリームのコンテンツには、クローンが作成された時点での親の内容 (同じイベント ID を含む) が反映されます。

property-graph エクスポートのフィールド

params の concurrency フィールド

(オプション) デフォルト: 4。

concurrency パラメータは、エクスポートプロセスで使用するパラレルクエリの数を指定します。

```
"concurrency" : (for example, 24)
```

データのエクスポート元となるすべてのインスタンスで、同時実行レベルを vCPUs の数の 2 倍に設定することが適切です。たとえば、r5.xlarge インスタンスには 4 つの vCPUs があります。3 つの r5.xlarge インスタンスのクラスターからエクスポートする場合は、同時実行レベルを 24 (= 3 x 2 x 4) に設定できます。

Neptune-Export サービスを使用している場合、同時実行レベルは jobSize 設定により制限されます。たとえば、小規模なジョブは 8 の同時実行レベルをサポートします。小さなジョブの同時実行レベルを 24 に指定しようとすると、concurrency パラメータの場合、有効レベルは 8 のままです。

クローンクラスターからエクスポートする場合、エクスポートプロセスは、クローンインスタンスのサイズとジョブサイズに基づいて適切な同時実行レベルを計算します。

params の edgeLabels フィールド

(オプション)

edgeLabels を使用して指定したラベルを持つエッジのみをエクスポートします。

```
"edgeLabels" : ["(a label)", "(another label"]
```

JSON 配列の各ラベルは、単一の単純なラベルでなければなりません。

`scope` パラメータが `edgeLabels` パラメータよりも優先されるため、`scope` 値にエッジが含まれない場合、`edgeLabels` パラメータは効果がありません。

`params` の `filter` フィールド

(オプション)

`filter` を使用して特定のラベルを持つノードまたはエッジのみをエクスポートするように指定し、各ノードまたはエッジに対してエクスポートされるプロパティをフィルタリングします。

`filter` オブジェクトの一般的な構造は、インラインまたはフィルター設定ファイルで、次のようになります。

```
"filter" : {
  "nodes": [ (array of node label and properties objects) ],
  "edges": [ (array of edge definition an properties objects) ]
}
```

- **`nodes`** — ノードとノードプロパティの JSON 配列が次の形式で含まれています。

  ```
  "nodes : [
    {
      "label": "(node label)",
      "properties": [ "(a property name)", "(another property name)", ( ... ) ]
    }
  ]
  ```

  - `label` — ノードの property-graph ラベル (1 つまたは複数)。

    単一の値、またはノードに複数のラベルがある場合は、値の配列を取ります。
  - `properties` — エクスポートするノードのプロパティの名前の配列が含まれています。
- **`edges`** — 次の形式のエッジ定義の JSON 配列が含まれます。

  ```
  "edges" : [
    {
      "label": "(edge label)",
      "properties": [ "(a property name)", "(another property name)", ( ... ) ]
    }
  ]
  ```

- `label` — エッジのプロパティグラフラベル。単一の値を取ります。
- `properties` -エクスポートするエッジのプロパティの名前の配列が含まれています。

`params` の `filterConfigFile` フィールド

(オプション)

`filterConfigFile` を使用して `filter` パラメータと同じ形式のフィルター設定を含む JSON ファイルを指定します。

```
 "filterConfigFile" : "s3://(your Amazon S3 bucket)/neptune-export/(the name of the
JSON file)"
```

`filterConfigFile` ファイルの形式については、フィルター を参照してください。

`params` の property-graph データに使用される `format` フィールド

(オプション) デフォルト: `csv` (カンマ区切り値)

`format` パラメータは、エクスポートされたプロパティグラフデータの出力形式を指定します。

```
 "format" : (one of: csv, csvNoHeaders, json, neptuneStreamsJson)
```

- **`csv`** — カンマ区切り値 (CSV) 形式の出力で、列見出しはGremlin ロードデータ形式に基づいています。
- **`csvNoHeaders`** — 列見出しを含まないCSV形式のデータ。
- **`json`** — JSON 形式のデータ。
- **`neptuneStreamsJson`** — GREMLIN_JSON 変更のシリアル化形式を使用する JSON 形式のデータ。

`params` の `gremlinFilter` フィールド

(オプション)

`gremlinFilter` パラメータを使用すると、`has()` ステップのように、Gremlin スニペットを指定でき、これはノードとエッジの両方をフィルタリングするために使用します。

```
 "gremlinFilter" : (a Gremlin snippet)
```

フィールド名と文字列値は、エスケープされた二重引用符で囲む必要があります。日付と時刻については、datetime メソッドを使用できます。

次の例では、日付作成プロパティの値が 2021-10-10 より大きいノードとエッジのみをエクスポートします。

```
  "gremlinFilter" : "has(\"created\", gt(datetime(\"2021-10-10\")))"
```

params の gremlinNodeFilter フィールド

(オプション)

gremlinNodeFilter パラメータを使用して、has() ステップのように、Gremlin スニペットを指定でき、これはノードをフィルタリングするために使用します。

```
  "gremlinNodeFilter" : (a Gremlin snippet)
```

フィールド名と文字列値は、エスケープされた二重引用符で囲む必要があります。日付と時刻については、datetime メソッドを使用できます。

次の例では、これらのノードのみをエクスポートしています。値が true である deleted ブールプロパティ。

```
  "gremlinNodeFilter" : "has(\"deleted\", true)"
```

params の gremlinEdgeFilter フィールド

(オプション)

gremlinEdgeFilter パラメータを使用すると、Gremlin スニペットを指定できます。エッジをフィルタリングする has() ステップを使用します。

```
  "gremlinEdgeFilter" : (a Gremlin snippet)
```

フィールド名と文字列値は、エスケープされた二重引用符で囲む必要があります。日付と時刻については、datetime メソッドを使用できます。

次の例では、strength 値が 5 の数値プロパティであるエッジのみをエクスポートしています。

```
  "gremlinEdgeFilter" : "has(\"strength\", 5)"
```

params の nodeLabels フィールド

(オプション)

nodeLabels を使用して指定したラベルを持つノードのみをエクスポートします。

```
  "nodeLabels" : ["(a label)", "(another label"]
```

JSON 配列の各ラベルは、単一の単純なラベルでなければなりません。

scope パラメータが nodeLabels パラメータよりも優先されるため、scope 値にノードが含まれない場合、nodeLabels パラメータは効果がありません。

params の scope フィールド

(オプション) デフォルト: all。

scope パラメータは、ノードのみ、エッジのみ、またはノードとエッジの両方をエクスポートするかどうかを指定します。

```
  "scope" : (one of: nodes, edges, or all)
```

- nodes — ノードとそのプロパティのみをエクスポートします。
- edges — エッジとそのプロパティのみをエクスポートします。
- all — ノードとエッジの両方とそのプロパティをエクスポートします (デフォルト)。

RDF エクスポート用のフィールド

params の RDF データに使用されるフィールド format

(オプション) デフォルト: turtle

format パラメータは、エクスポートされた RDF データの出力形式を指定します。

```
  "format" : (one of: turtle, nquads, ntriples, neptuneStreamsJson)
```

- **turtle** — Turtle 形式の出力。
- **nquads** — 列見出しのない N-Quads 形式のデータ。

- **ntriples** — N-Triples 形式のデータ。
- **neptuneStreamsJson** — SPARQL NQUADS 変更シリアル化フォーマットを使用する JSON 形式のデータ。

params の **rdfExportScope** フィールド

(オプション) デフォルト: graph。

rdfExportScope パラメータは、RDF エクスポートのスコープを指定します。

```
  "rdfExportScope" : (one of: graph, edges, or query)
```

- graph — すべての RDF データをエクスポートします。
- edges— エッジを表すトリプルだけをエクスポートします。
- query — sparql フィールドを使って SPARQL クエリによって取得されたデータをエクスポートします。。

params の **sparql** フィールド

(オプション)

sparql パラメータを使用すると、エクスポートするデータを取得するための SPARQL クエリを指定できます。

```
  "sparql" : (a SPARQL query)
```

sparql フィールドを使用してクエリを指定した場合、rdfExportScope フィールドを query に設定する必要があります。

params の **namedGraph** フィールド

(オプション)

namedGraph パラメータを使用すると、エクスポートを単一の名前付きグラフに制限する IRI を指定できます。

```
  "namedGraph" : (Named graph IRI)
```

namedGraph パラメータは、 rdfExportScopeフィールドを に設定した場合にのみ使用できますgraph。

エクスポートされるものをフィルタリングの例

エクスポートされるデータをフィルタリングする方法の例を次に示します。

property-graph データのエクスポートのフィルタリング

scope を使用してエッジのみをエクスポートする例

```
{
  "command": "export-pg",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "scope": "edges"
  },
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export"
}
```

nodeLabels および **edgeLabels** を使用して特定のラベルを持つノードとエッジのみをエクスポートする例

次の例の nodeLabels パラメータは、Person ラベルまたは Post ラベルを持つノードのみをエクスポートするように指定しています。edgeLabels パラメータは、likesラベルを持つエッジのみをエクスポートするように指定しています。

```
{
  "command": "export-pg",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "nodeLabels": ["Person", "Post"],
    "edgeLabels": ["likes"]
  },
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export"
}
```

filterを使用してノード、エッジ、およびプロパティのみをエクスポートする例

この例の filter オブジェクトはcountry ノードとそれらの type、code、desc プロパティ、および route エッジとその dist プロパティをエクスポートします。

```
{
  "command": "export-pg",
  "params": {
```

```
    "endpoint": "(your Neptune endpoint DNS name)",
    "filter": {
      "nodes": [
        {
          "label": "country",
          "properties": [
            "type",
            "code",
            "desc"
          ]
        }
      ],
      "edges": [
        {
          "label": "route",
          "properties": [
            "dist"
          ]
        }
      ]
    }
  },
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export"
}
```

`gremlinFilter` を使用する例

この例では gremlinFilter を使用して 2021-10-10 より後に作成されたノードとエッジ (つまり、2021-10-10より大きい値を持つ created プロパティ) のみをエクスポートします。

```
{
  "command": "export-pg",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "gremlinFilter" : "has(\"created\", gt(datetime(\"2021-10-10\")))"
  },
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export"
}
```

`gremlinNodeFilter` を使用する例

この例では gremlinNodeFilter を使用して削除されたノード (値が true のブール値 deleted プロパティを持つノード) のみをエクスポートします。

```
{
  "command": "export-pg",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "gremlinNodeFilter" : "has(\"deleted\", true)"
  },
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export"
}
```

gremlinEdgeFilter を使用する例

この例では gremlinEdgeFilter を使用して値が 5 の strength 数値プロパティを持つエッジのみをエクスポートします。

```
{
  "command": "export-pg",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "gremlinEdgeFilter" : "has(\"strength\", 5)"
  },
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export"
}
```

filter、gremlinNodeFilter、nodeLabels、edgeLabels、scopeを組み合わせる

この例の filter オブジェクトは次をエクスポートします。

- country ノードとそれらの type、code、desc プロパティ
- airport ノードとそれらの icao、code、runways プロパティ
- route エッジとその dist プロパティ

gremlinNodeFilter パラメータはノードをフィルタリングするため、値が A で始まる code プロパティのみがエクスポートされます。

nodeLabels および edgeLabels パラメータは出力をさらに制限し、airport ノードおよび route エッジのみがエクスポートされます。

最後に、scope パラメータはエクスポートからエッジを削除し、出力には指定した airport ノードだけが残されます。

```
{
  "command": "export-pg",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "filter": {
      "nodes": [
        {
          "label": "airport",
          "properties": [
            "code",
            "icao",
            "runways"
          ]
        },
        {
          "label": "country",
          "properties": [
            "type",
            "code",
            "desc"
          ]
        }
      ],
      "edges": [
        {
          "label": "route",
          "properties": [
            "dist"
          ]
        }
      ]
    },
    "gremlinNodeFilter": "has(\"code\", startingWith(\"A\"))",
    "nodeLabels": [
      "airport"
    ],
    "edgeLabels": [
      "route"
    ],
    "scope": "nodes"
  },
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export"
}
```

RDF データのエクスポートのフィルタリング

`rdfExportScope` および `sparql` を使用して特定のエッジをエクスポートする

この例では、述語が <http://kelvinlawrence.net/air-routes/objectProperty/route> で、オブジェクトがリテラルでないトリプルをエクスポートします。

```
{
  "command": "export-rdf",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "rdfExportScope": "query",
    "sparql": "CONSTRUCT { ?s <http://kelvinlawrence.net/air-routes/objectProperty/
route> ?o } WHERE { ?s ?p ?o . FILTER(!isLiteral(?o)) }"
  },
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export"
}
```

`namedGraph` を使用して単一の名前付きグラフをエクスポートする

この例では、名前付きグラフ <http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph> に属するトリプルをエクスポートします。

```
{
  "command": "export-rdf",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "rdfExportScope": "graph",
    "namedGraph": "http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph"
  },
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export"
}
```

Neptune エクスポートプロセスのトラブルシューティング

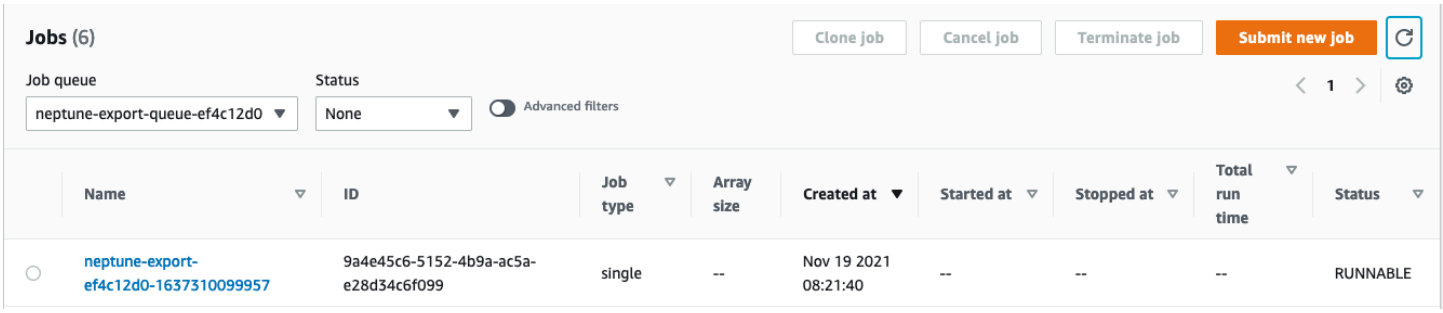
Amazon Neptune のエクスポートプロセスでは、AWS Batch を使用して Neptune データのエクスポートに必要なコンピューティングリソースとストレージリソースをプロビジョニングします。エクスポートが実行中である場合、`logs` フィールドのリンクを使用して、エクスポートジョブ用に CloudWatch ログにアクセスします。

ただし、エクスポートを実行する AWS Batch ジョブ用の CloudWatch はは、AWS Batch ジョブが実行中である場合にのみ利用できます。Neptune export がエクスポートが保留状態であることを報告した場合、CloudWatch Logs にアクセスするためのログリンクはありません。エクスポートジョブが数分以上 `pending` 状態になると、AWS Batch リソースの基礎となるプロビジョニングに問題がある可能性があります。

エクスポートジョブが保留中の状態を離れる場合、次のようにステータスを確認できます。

AWS Batch ジョブのステータスを確認するには

1. https://console.aws.amazon.com/batch/ で AWS Batch コンソールを開きます。
2. neptune-export ジョブキューを選択します。
3. エクスポートを開始すると Neptune のエクスポートによって返される `jobName` に名前が一致するジョブを探します。

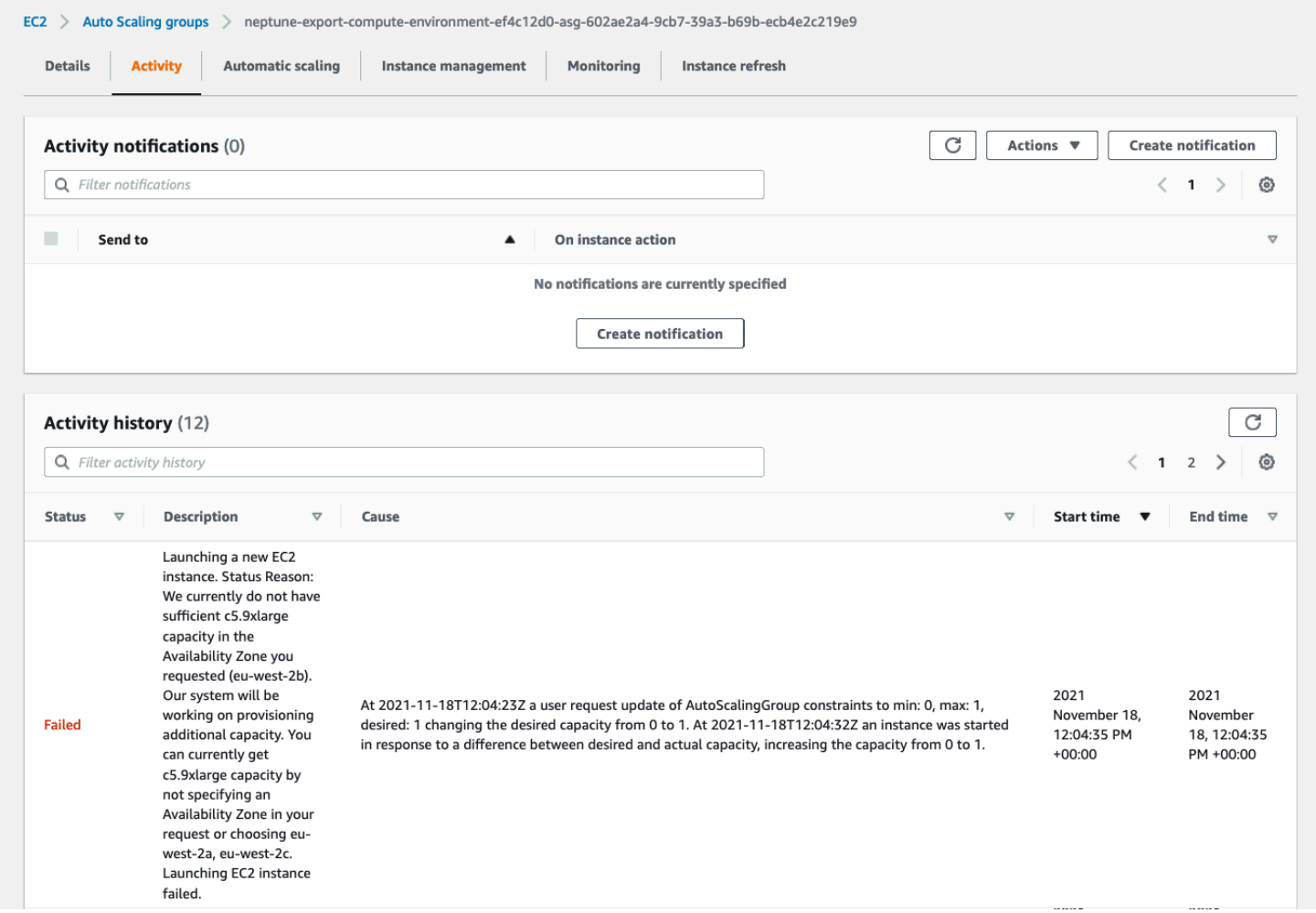


ジョブが `RUNNABLE` 状態で止まってしまった場合、ネットワークまたはセキュリティの問題により、コンテナインスタンスが基盤となる Amazon Elastic Container Service (Amazon ECS) クラスターに入れないことが原因である可能性があります。このサポート記事にあるコンピューティング環境のネットワークとセキュリティ設定の検証に関するセクションを参照してください。

もう 1 つ確認できるのは、auto-scaling の問題です。

Amazon EC2 auto-scaling グループで AWS Batch コンピューティング環境を確認するには

1. Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
2. neptune-export コンピューティング環境の Auto Scaling グループを選択します。
3. [Activity] (アクティビティ) タブを開き、失敗したイベントのアクティビティ履歴を確認します。



Neptune エクスポートの一般的なエラー

`org.eclipse.rdf4j.query.QueryEvaluationException: Tag mismatch!`

`export-rdf` ジョブが `Tag mismatch! QueryEvaluationException` で定期的に失敗する場合、Neptune インスタンスのサイズが、Neptune Export が使用する大規模で長時間実行されるクエリに対して小さすぎます。

以下のように大きな Neptune インスタンスにスケールアップするか、大規模なクローンクラスターからエクスポートするようにジョブを設定することで、このエラーの発生を回避できます。

```
'{
  "command": "export-rdf",
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "cloneCluster": True,
    "cloneClusterInstanceType" : "r5.24xlarge"
  }
}'
```

Amazon Neptune データベースの管理

このセクションでは、AWS Management Console と AWS CLI を使用して Neptune DB クラスターを管理、維持する方法について説明します。

Neptune は、レプリケーショントポロジに接続されているデータベースサーバーのクラスター上で動作します。そのため、Neptune を管理するには、複数のサーバーへの変更をデプロイし、すべての Neptune レプリカがプライマリサーバーで維持されていることを確認する必要があります。

Neptune では、データの増加に伴い、基本となるストレージを透過的にスケーリングしているため、Neptune の管理に必要なディスクストレージの管理は比較的わずかです。同様に、Neptune では、継続的バックアップが自動的に行われるため、Neptune クラスターでは、バックアップの実行に伴う過度な計画やダウンタイムは必要ありません。

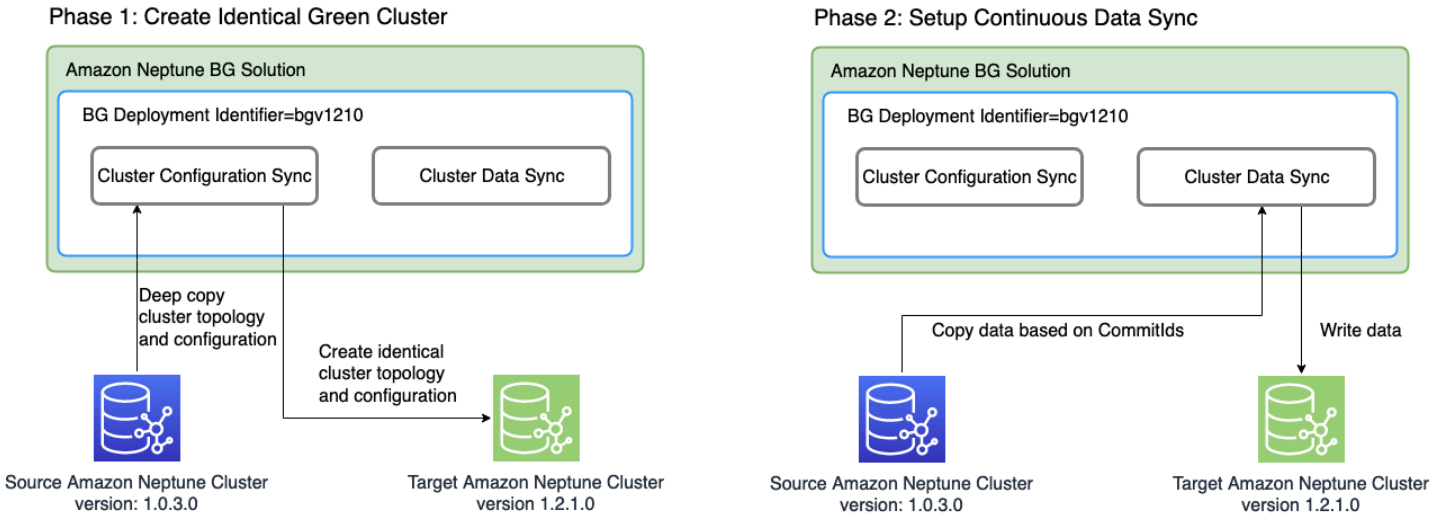
トピック

Neptune ブルー/グリーンソリューションを使用してブルーグリーンアップデートを実行する

Amazon Neptune エンジンのアップグレードでは、更新のインストールと検証中はデータベースが使用できないため、アプリケーションのダウンタイムが必要になる場合があります。これは、手動で開始されたか自動で開始されたかに関係なく当てはまります。

Neptune は、ブルー/グリーンデプロイソリューションを提供しています。これは、AWS CloudFormation スタックを使用して実行でき、このようなダウンタイムを大幅に削減できます。ブルー本番環境と同期したグリーンステージング環境が構築されます。その後、そのステージング環境を更新して、エンジンのマイナーまたはメジャーバージョンアップグレード、グラフデータモデルの変更、またはオペレーティングシステムの更新を実行し、結果をテストできます。最後に、ダウンタイムをほとんど発生させずに、すぐに本番環境に切り替えることができます。

Neptune ブルー/グリーンソリューションには、次の図に示すように 2 つのフェーズがあります。



フェーズ 1 では、本番クラスターと同じグリーン DB クラスターを作成します。

このソリューションでは、一意のブルー/グリーンデプロイ識別子と、本番クラスターと同じクラスタートポロジを使用して DB クラスターを作成します。つまり、DB インスタンスの数とサイズ、パラメータグループ、および設定は、本番 (ブルー) DB クラスターと同じですが、指定したターゲットエンジンバージョンにアップグレードされる点が異なります。ターゲットエンジンのバージョンは、現在の (ブルー) エンジンバージョンよりも高い必要があります。ターゲットのマイナーエンジンバージョンとメジャーエンジンバージョンを指定できます。必要に応じて、ソリューションは指定されたターゲットエンジンバージョンに到達するために必要な中間アップグレードを実行します。この新しいクラスターはグリーンステージング環境になります。

フェーズ 2 では、継続的なデータ同期を設定します。

グリーン環境が完全に準備されると、ソリューションは Neptune ストリームを使用してソース (ブルー) クラスターとターゲット (グリーン) クラスター間の連続レプリケーションを設定します。これらの間のレプリケーションの差がゼロになると、ステージング環境をテストできる状態になります。その時点で、レプリケーションの遅延がこれ以上発生しないように、ブルークラスターへの書き込みを一時停止する必要があります。

ターゲットエンジンのバージョンには、アプリケーションに影響する新しい機能や依存関係が含まれている可能性があります。「エンジンリリース」の下にあるターゲットエンジンリリースのページとそれに続くエンジンリリースのページをチェックして、現在のエンジンバージョンから何が変更されたかを確認してください。本番環境に昇格する前に、グリーンクラスターで統合テストを実行するか、アプリケーションを手動で検証するのが最善です。

グリーンクラスターで変更をテストして検証したら、アプリケーションのデータベースエンドポイントをブルークラスターからグリーンクラスターに切り替えるだけです。

スイッチオーバー後、Neptune ブルー/グリーンソリューションは古いブルー本番環境を削除しません。必要に応じて引き続きアクセスして、追加の検証やテストを行うことができます。削除しない限り、そのインスタンスには標準の請求料金が適用されます。ブルー/グリーンソリューションでは、他の AWS サービスも使用し、その費用は通常価格で請求されます。使い終わったソリューションを削除する方法の詳細については、クリーンアップセクションで説明しています。

Neptune ブルー/グリーンスタックを実行するための前提条件

Neptune ブルー/グリーンスタックを起動する前に:

- 本番 (ブルー) クラスターで Neptune ストリームを有効にします。
- ブルークラスターのすべてのインスタンスが使用可能状態になっている必要があります。インスタンスの状態は、Neptune コンソールで、または describe-db-instances API を使用して確認することができます。
- また、すべてのインスタンスは DB クラスターパラメータグループと同期している必要があります。
- Neptune ブルー/グリーンソリューションでは、ブルークラスターが配置されている VPC に DynamoDB VPC エンドポイントが必要です。「Amazon VPC エンドポイントを使用して DynamoDB にアクセスする」を参照してください。

- ブルー本番 DB クラスターの書き込みワークロードができるだけ軽くなる時間帯を選んで、ソリューションを実行します。例えば、一括ロードが行われるときや、その他の理由で大量の書き込み操作が行われる可能性があるときには、ソリューションを実行しないでください。

AWS CloudFormation テンプレートを使用して Neptune ブルー/グリーンソリューションを実行する

AWS CloudFormation を使用して Neptune ブルー/グリーンソリューションをデプロイできます。CloudFormation テンプレートは、ブルーソース Neptune データベースと同じ VPC に Amazon EC2 インスタンスを作成し、そこにソリューションをインストールして実行します。「進行状況のモニタリング」で説明されているように、CloudWatch ログで進行状況をモニタリングできます。

これらのリンクを使用して、ソリューションテンプレートを確認したり、[スタックの起動] ボタンを選択して AWS CloudFormation コンソールで起動したりできます。

表示	デザイナーで表示	

コンソールで、ウィンドウの右上にあるドロップダウンから、ソリューションを実行する AWS リージョンを選択します。

スタックパラメータを次のように設定します。

- **DeploymentID** — 各 Neptune ブルー/グリーンデプロイに固有の識別子。

  グリーン DB クラスター識別子として、またデプロイ時に作成された新しいリソースに名前を付けるためのプレフィックスとしても使用されます。
- **NeptuneSourceClusterId** - アップグレードしようとしているブルー DB クラスターの識別子。
- **NeptuneTargetClusterVersion:** — ブルー DB クラスターをアップグレードする Neptune エンジンのバージョン。

  これは、現在のブルー DB クラスターのエンジンバージョンよりも新しいバージョンである必要があります。
- **DeploymentMode** — これが新しいデプロイなのか、以前のデプロイを再開しようとしているのかを示します。以前のデプロイと同じ `DeploymentID` を使用している場合は、`DeploymentMode` を `resume` に設定します。

有効な値は `new` (デフォルト) と `resume` です。

- **`GraphQueryType`** — データベースのグラフデータタイプ。

  有効な値は `propertygraph` (デフォルト) と `rdf` です。

- **`SubnetId`** — ブルー DB クラスターが配置されているのと同じ VPC のサブネット ID。(「同じ VPC の Amazon EC2 インスタンスから Neptune DB クラスターに接続する」を参照)。

  EC2 Connect 経由でインスタンスに SSH 接続する場合は、パブリックサブネットの ID を指定します。

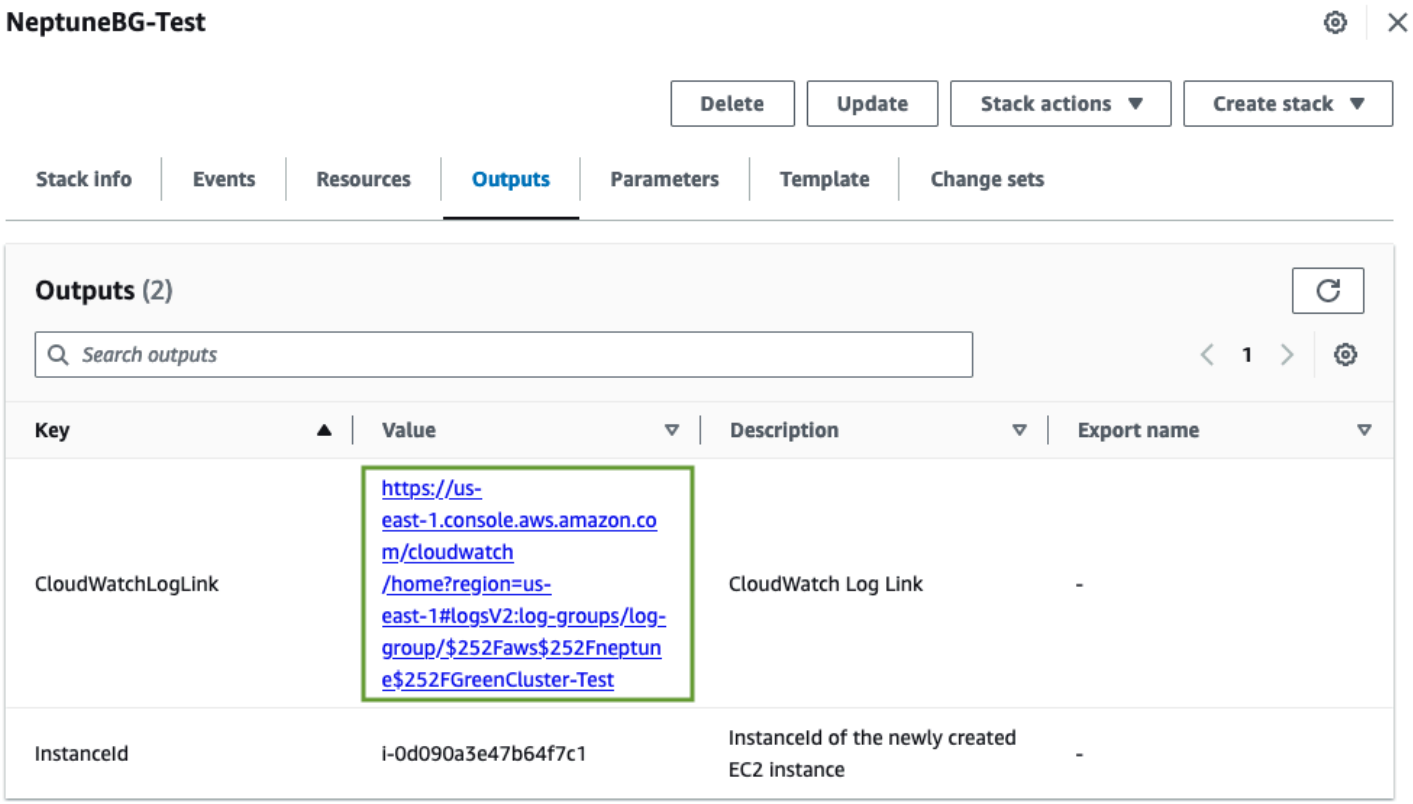
- **`InstanceSecurityGroup`** - Amazon EC2 インスタンス用のセキュリティグループ。

  セキュリティグループはブルー DB クラスターにアクセスできる必要があり、インスタンスに SSH 接続できる必要があります。「VPC コンソールを使用してセキュリティグループを作成する」を参照してください。

スタックが完了するまで待ちます。完了するとすぐにソリューションが開始されます。その後、次のセクションで説明するように、CloudWatch ログを使用してデプロイプロセスをモニタリングできます。

Neptune ブルー/グリーンデプロイの進捗状況の監視

Neptune ブルー/グリーンソリューションの進行状況をモニタリングするには、CloudWatch コンソールにアクセスして、`/aws/neptune/`*`(Neptune Blue/Green deployment ID)`* CloudWatch ロググループのログを確認します。CloudWatch ログへのリンクは、ソリューションの AWS CloudFormation スタックの出力にあります。



スタックパラメータとしてパブリックサブネットを指定した場合は、スタックの一部として作成された Amazon EC2 インスタンスに SSH 接続して、`/var/log/cloud-init-output.log` でログを参照することもできます。

ログには、次のスクリーンショットに示すように、Neptune ブルー/グリーンソリューションによって実行されたアクションが表示されます。

```
============================================================

Neptune Blue Green Deployment Solution Version: 0.1.06012023

============================================================

Checking whether cluster with id = bg-06-01-14-20-29test-bg1-bgInt already exists.

BlueGreen deployment_mode = new

Didn't find any cluster with id bg-06-01-14-20-29test-bg1-bgInt

Cloned_cluster_id: bg-06-01-14-20-29test-bg1-bgInt

Replication_stack_name: bg-06-01-14-20-29test-bg1-bgInt-replication

DescribeDbClusters response for test-bg1-bgIntegTest-06-01-14-20-29: {'AllocatedStorage': 1,
'AvailabilityZones': ['us-east-1b', 'us-east-1c', 'us-east-1f'], 'BackupRetentionPeriod': 1,
'DBClusterIdentifier': 'test-bg1-bgintegtest-06-01-14-20-29', 'DBClusterParameterGroup': 'green-     -blue-
green-deployment-test-123456789012345-pg-tes710', 'DBSubnetGroup': 'default', 'Status': 'available',
'EarliestRestorableTime': datetime.datetime(2023, 6, 1, 8, 51, 23, 394000, tzinfo=tzlocal()), 'Endpoint':
'test-bg1-bgintegtest-06-01-14-20-29.cluster-critvszpmydm.us-east-1.neptune.amazonaws.com', 'ReaderEndpoint':
'test-bg1-bgintegtest-06-01-14-20-29.cluster-ro-critvszpmydm.us-east-1.neptune.amazonaws.com', 'MultiAZ':
False, 'Engine': 'neptune', 'EngineVersion': '1.2.0.0', 'LatestRestorableTime': datetime.datetime(2023, 6, 1,
8, 51, 23, 394000, tzinfo=tzlocal()), 'Port': 8182, 'MasterUsername': 'admin', 'PreferredBackupWindow':
'06:33-07:03', 'PreferredMaintenanceWindow': 'fri:09:44-fri:10:14', 'ReadReplicaIdentifiers': [],
'DBClusterMembers': [{'DBInstanceIdentifier': 'test-bg1-bgintegtest-06-01-14-20-29i-1', 'IsClusterWriter':
True, 'DBClusterParameterGroupStatus': 'in-sync', 'PromotionTier': 1}], 'VpcSecurityGroups':
```

ログメッセージには、ブルーおよびグリーンクラスター間の同期ステータスが表示されます。

```
DDB checkpoint {'S': '1'}, {'S': '6'}

DDB Checkpoint: {'checkpointSubSequenceNumber': {'S': '6'}, 'lastUpdateTime': {'N': '1685611142127'},
'leaseOwner': {'S': 'nobody'}, 'checkpoint': {'S': '1'}, 'leaseKey': {'S': 'bg-anl   -234567899-replication'}}

Time difference for last checkpoint and last stream event: 5841351

Stream eventId difference for last replication checkpoint and last stream event on the Source cluster: 0:0

Found region : us-east-1

Cloudwatch Log Url for blue green solution is https://us-east-1.console.aws.amazon.com/cloudwatch
/home?region=us-east-1#logsV2:log-groups/log-group/aws/neptune/bg

Cloudwatch dashboard url for replication is https://console.aws.amazon.com/cloudwatch/home?region=us-
east-1#dashboards:name=neptune-stream-poller-bg-an   -234567899-replication

Replication poller lambda arn is arn:aws:lambda:us-east-1:451235071234:function:bg-an   -234567899-replic-
NeptuneStreamPollerLambd-B6V1ytULgmSP. Look for CW log the poller lambda for more troubleshooting.

Stream Last EventId {'commitNum': 1, 'opNum': 6} on cluster : database-d61852469-t        -experiment.cluster-
critvszpmydm.us-east-1.neptune.amazonaws.com:8182

DDB checkpoint {'S': '1'}, {'S': '6'}

DDB Checkpoint: {'checkpointSubSequenceNumber': {'S': '6'}, 'lastUpdateTime': {'N': '1685611207245'},
'leaseOwner': {'S': 'nobody'}, 'checkpoint': {'S': '1'}, 'leaseKey': {'S': 'bg-ankig-234567899-replication'}}
```

同期プロセスでは、ブルークラスターの最新のストリーム `eventID` と、Neptune-to-Neptune レプリケーションスタックによって作成された DynamoDB チェックポイントテーブルにあるレプリケーションチェックポイントとの差を計算することにより、レプリケーションラグがチェックされます。これらのメッセージを使用して、現在のレプリケーションの違いを監視できます。

本番環境のブルークラスターから更新されたグリーンクラスターへのカットオーバー

グリーンクラスターを本番環境に昇格させる前に、ブルークラスターとグリーンクラスターのコミットの差がゼロであることを確認し、ブルークラスターへの書き込みトラフィックをすべて無効にします。データベースエンドポイントをグリーンクラスターに切り替える際にブルークラスターへの書き込みが続行していると、両方のクラスターに部分的なデータを書き込むことによってデータが破損する可能性があります。まだ読み取りトラフィックを無効にする必要はないかもしれません。

ソース (ブルー) クラスターで IAM 認証を有効にしている場合は、アプリケーションで使用している IAM ポリシーをグリーンクラスターを指すように更新してください (このようなポリシーの例については、この「無制限のアクセスポリシー」を参照してください)。

書き込みトラフィックを無効にした後、レプリケーションが終了するのを待ってから、グリーンクラスターで (ブルークラスターではなく) 書き込みトラフィックを有効にします。また、読み取りトラフィックをブルークラスターからグリーンクラスターに切り替えます。

Neptune ブルー/グリーンソリューションが完了した後のクリーンアップ

ステージング (グリーン) クラスターを本番環境に昇格させたら、Neptune ブルー/グリーンソリューションによって作成されたリソースをクリーンアップします。

- ソリューションを実行するために作成された Amazon EC2 インスタンスを削除します。
- グリーンクラスターをブルークラスターと同期させていた Neptune ストリームベースのレプリケーションの AWS CloudFormation テンプレートを削除します。メインのテンプレートは以前に指定したスタック名であり、もう 1 つはデプロイ ID の後に「-replication」が続きます (つまり、*(DeploymentID)*-replication)。

AWS CloudFormation テンプレートを削除しても、クラスター自体は削除されません。グリーンクラスターが期待どおりに動作していることを確認したら、ブルークラスターを手動で削除する前に、オプションでスナップショットを取ることができます。

Neptune ブルー/グリーンソリューションのベストプラクティス

- グリーンクラスターを本番環境に切り替える前に、正常に機能していることを十分に検証する必要があります。データベースのデータと設定の一貫性をチェックしてください。新しいエンジンバージョンによっては、クライアントのアップグレードも必要になる場合があります。アップグレードする前に、エンジンのリリースノートを確認してください。本番環境でブルー/グリーンのアップグレードを開始する前に、開発、テスト、および実稼働前の環境で、これらすべてをテストする価値があります。
- ブルーサーバーからグリーンサーバーへの切り替えは、メンテナンス時間帯に行うのがベストです。
- アップグレードと同期後にすべてが正常に機能していることを確認するには、元のクラスターを一定期間保持してから削除することをお勧めします。予期しない問題が発生したときに役立つことがあります。
- Neptune ブルー/グリーンソリューションを実行するときは、一括読み込みなどの負荷の高い書き込み操作は避けてください。レプリケーションラグが発生し、大幅なダウンタイムが発生する可能性があるためです。ブルークラスターへの書き込みをオフにしてからグリーンクラスターでオンにするまでの時間は、ほんの数分であることが理想的です。

Neptune ブルー/グリーンソリューションのトラブルシューティング

Neptune ブルー/グリーンソリューションによって発生したエラー

- **`Cluster with id = (blue_green_deployment_id) already exists`** — 識別子 *`(blue_green_deployment_id)`* が付いた既存のクラスターがあります。

  クラスターが前回の Neptune ブルー/グリーンの実行で作成された場合は、新しいデプロイ ID を指定するか、デプロイモードを `resume` に設定します。
- **`Streams should be enabled on the source Cluster for Blue Green Deployment`** — ブルー (ソース) クラスターで Neptune ストリームを有効にします。
- **`No Bulkload should be in progress on source cluster: (cluster_id)`** — Neptune ブルー/グリーンソリューションは、進行中の一括ロードを確認すると終了します。

  これは、同期プロセスが書き込みに追いつくことができるようにするためです。Neptune ブルー/グリーンソリューションを開始する前に、進行中の一括ロードジョブを回避またはキャンセルしてください。
- **`Blue Green deployment requires instances to be in sync with db cluster parameter group`** — クラスターパラメータグループの変更は、DB クラスター全体で同期されている必要があります。「Amazon Neptune パラメータグループ」を参照してください。
- **`Invalid target engine version for Blue Green Deployment`** — ターゲットエンジンバージョンは Amazon Neptune のエンジンリリース にアクティブとしてリストされている必要があり、ソース (ブルー) クラスターの現在のエンジンリリースよりも新しい必要があります。

Neptune のアクセス許可を持つ IAM ユーザーの作成

Neptune コンソールにアクセスして Neptune DB クラスターを作成および管理するには、必要なすべてのアクセス許可を持つ IAM ユーザーを作成する必要があります。

最初のステップは、Neptune のサービスリンクロールポリシーを作成することです。

Amazon Neptune のサービスリンクロールポリシーを作成する

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左側のナビゲーションペインで、[ポリシー] を選択します。
3. [ポリシー] ページで、[ポリシーの作成] を選択します。
4. [ポリシーの作成] ページで [JSON] タブを選択し、以下のサービスリンクロールポリシーをコピーします。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "iam:CreateServiceLinkedRole",
      "Effect": "Allow",
      "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
AWSServiceRoleForRDS",
      "Condition": {
        "StringLike": {
            "iam:AWSServiceName":"rds.amazonaws.com"
        }
      }
    }
  ]
}
```

5. [次へ: タグ] を選択し、[タグの追加] ページで [次へ: レビュー] を選択します。
6. [ポリシーのレビュー] ページで、新しいポリシーに「NeptuneServiceLinked」という名前を付けます。

サービスにリンクされたロールの詳細については、「Neptune でのサービスにリンクされた IAM ロールの使用」を参照してください。

必要なすべてのアクセス許可を持つ新しい IAM ユーザーを作成する

次に、必要なアクセス許可を付与する、作成したサービスリンクロールポリシー (ここでは `NeptuneServiceLinked` という名前) とともに、適切な管理ポリシーがアタッチされた新しい IAM ユーザーを作成します。

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 左側のナビゲーションペインで [ユーザー] を選択し、[ユーザー] ページで [ユーザーの追加] を選択します。
3. [ユーザーの追加] ページで、新しい IAM ユーザーの名前を入力し、AWS 認証情報タイプとして [アクセスキー - プログラムによるアクセス] を選択し、[次へ: アクセス許可] を選択します。
4. [アクセス許可の設定] ページの [ポリシーのフィルタリング] ボックスに「Neptune」と入力します。次に、表示されるポリシーから以下を選択します。

   - NeptuneFullAccess
   - NeptuneConsoleFullAccess
   - NeptuneServiceLinked (以前に作成したサービスリンクロールポリシーにその名前を付けたと仮定します)。
5. 次に、[ポリシーのフィルタリング] ボックスに「Neptune」の代わりに「VPC」と入力します。リストされたポリシーから [AmazonVPCFullAccess] を選択します。
6. [次へ: タグ] を選択し、[タグの追加] ページで [次へ: レビュー] を選択します。
7. [レビュー] ページで、以下のすべてのポリシーが新しいユーザーにアタッチされていることを確認します。

   - NeptuneFullAccess
   - NeptuneConsoleFullAccess
   - NeptuneServiceLinked
   - AmazonVPCFullAccess

   次に、[ユーザーの作成] を選択します。
8. 最後に、新しいユーザーのアクセスキー ID とシークレットアクセスキーをダウンロードして保存します。

Amazon Simple Storage Service (Amazon S3) など、他のサービスとの相互運用には、さらにアクセス許可と信頼関係を追加する必要があります。

Amazon Neptune パラメータグループ

パラメータグループのパラメータを使用して、Amazon Neptune のデータベース設定を管理します。パラメータグループは、1 つ以上の DB インスタンスに適用されるエンジン設定値のコンテナとして機能します。

DB クラスターのパラメータグループと、いわゆる DB パラメータグループという、2 つのタイプの DB パラメータグループがあります。

- DB パラメータグループは、インスタンスレベルで適用され、通常 Neptune グラフエンジンに関連付けられています (例: `neptune_query_timeout` パラメータ)。
- DB クラスターのパラメータグループは、クラスター内のすべてのインスタンスに適用され、通常、より広範な設定があります。すべての Neptune クラスターは、DB クラスターパラメータグループに関連付けられます。そのクラスター内の各 DB インスタンスは、DB クラスターのパラメータグループに含まれるエンジン設定値を継承します。

DB クラスターパラメータグループで変更した設定値は、DB パラメータグループのデフォルト値を上書きします。DB パラメータグループ内の対応する値を編集すると、これらの値によって DB クラスターパラメータグループの設定が上書きされます。

カスタム DB パラメータグループを指定せずに DB インスタンスを作成した場合は、デフォルトの DB パラメータグループが使用されます。デフォルトの DB パラメータグループのパラメータ設定を変更することはできません。代わりに、デフォルトのパラメーター設定を変更するには、新しい DB パラメータグループを作成する必要があります。DB エンジンのすべてのパラメータを、作成した DB パラメータグループで変更できるわけではありません。

パラメータグループは、さまざまな Neptune エンジンバージョンと互換性のあるファミリーで作成されます。デフォルトのパラメータグループファミリーは `neptune1` であり、`1.2.0.0` より前のすべてのエンジンバージョンと互換性があります。リリース: 1.2.0.0 (2022-07-21) 以降では、代わりに `neptune1.2` パラメータグループファミリーを使用する必要があります。つまり、`1.2.0.0` 以上にアップグレードするときには、まず、すべてのカスタムパラメータグループを `neptune1.2` ファミリーで作成して、アップグレード時にアタッチできるようにする必要があります。

Neptune のパラメータには、静的なものと動的なものがあります。違いは次のとおりです。

静的パラメータ

- 静的パラメータは、DB インスタンスが再起動された後にのみ有効になるパラメータです。言い換えると、静的パラメータを変更してインスタンスの DB パラメータグループを保存したとき、

パラメータの変更を有効にするには、DB インスタンスを手動で再起動する必要があります。現在、Neptune インスタンスレベルのパラメータ (DB クラスターパラメータグループではなく DB パラメータグループ内) はすべて静的です。

- クラスターレベルの静的パラメータを変更して、DB クラスターのパラメータグループを保存したとき、パラメータの変更は、クラスター内のすべての DB インスタンスを手動で再起動した後に有効になります。

動的パラメータ

- 動的パラメータとは、パラメータグループ内のパラメータが更新されたほぼ直後に有効になるパラメータです。つまり、動的パラメータを更新した後、DB インスタンスを再起動しなくてもパラメータの変更が有効になります。
- 動的クラスターパラメータの変更がすべての DB インスタンスに適用されるまで、多少の遅延が予想されます。
- 更新された動的パラメータ値は、現在実行中のリクエストには適用されず、変更が行われた後に送信されたリクエストにのみ適用されます。
- クラスターレベルの動的パラメータを変更すると、デフォルトでは、パラメータの変更は直ちに DB クラスターに適用され、再起動を必要としません。パラメータの変更を、クラスター内の DB インスタンスが再起動されるまで延期するには、AWS CLI を使用して、パラメータの変更について `ApplyMethod` を `pending-reboot` に設定します。

現在、以下の新しいクラスターパラメータを除くすべてのパラメータは静的です。

- `neptune_enable_slow_query_log` (クラスターレベル)
- `neptune_slow_query_log_threshold` (クラスターレベル)

DB パラメータグループのパラメータを使用する際に、注意する必要がある重要な点を以下に示します。

- DB パラメータグループに不適切な設定のパラメータがあると、パフォーマンスが低下したりシステムが不安定になったり、予期しない悪影響が生じることがあります。データベースパラメータの変更時には常に注意が必要です。DB パラメータグループの変更前にデータをバックアップしてください。テスト DB インスタンスでパラメータグループの設定の変更を試してから、本稼働 DB インスタンスにそれらの変更を適用します。

- DB インスタンスに関連付けられている DB パラメータグループを変更する場合、DB インスタンスで新しい DB パラメータグループを使用する前に、インスタンスを手動で再起動する必要があります。

> **Note**
>
> リリース: 1.2.0.0 (2022-07-21) より前は、DB クラスター内のすべてのリードレプリカインスタンスは、プライマリ (ライター) インスタンスが再起動するたびに自動的に再起動されていました。
> リリース: 1.2.0.0 (2022-07-21) 以降では、プライマリインスタンスを再起動しても、レプリカインスタンスは再起動しません。つまり、クラスターレベルのパラメータを変更する場合、パラメータの変更を反映するには、各インスタンスを個別に再起動する必要があります。

DB クラスターパラメータグループまたは DB パラメータグループの編集

1. AWS マネジメントコンソールにサインインして Amazon Neptune コンソール (https://console.aws.amazon.com/neptune/home) を開きます。
2. ナビゲーションペインの [Parameter Groups] (パラメータグループ) を選択します。
3. 編集する DB パラメータグループの [Name] (名前) リンクを選択します。

   (オプション) [Create parameter group] (パラメータグループの作成) を選択して、新しいクラスターのパラメータグループを作成しその新しいグループを作成します。その後、その新しいパラメータグループの [Name] (名前) を選択します。

   > **Important**
   >
   > デフォルトの DB クラスターのパラメータグループしかない場合、このステップは必須です。デフォルトの DB クラスターのパラメータグループは変更できないためです。

4. パラメータを検索し、名前列の横にある値フィールドをクリックします。
5. 使用できる値を入力し、値フィールドの横にあるチェックを選択します。
6. [変更の保存] をクリックします。
7. DB クラスターパラメータを変更する場合は、Neptune クラスター内のすべての DB インスタンスを再起動し、DB インスタンスパラメータを変更する場合は 1 つ以上の特定のインスタンスを再起動します。

DB パラメータグループまたは DB クラスターパラメータグループの作成

Neptune コンソールを使用して、新しいパラメータグループを簡単に作成できます。

1. AWS マネジメントコンソールにサインインして Amazon Neptune コンソール (https://console.aws.amazon.com/neptune/home) を開きます。
2. 左のナビゲーションペインの [Parameter Groups] (パラメータグループ) を選択します。
3. [Create DB parameter group] (DB パラメータグループの作成) を選択します。

   [Create DB parameter group] (DB パラメータグループの作成) ページが表示されます。
4. [パラメータグループファミリー] リストで [neptune1] を選択するか、エンジンバージョン 1.2.0.0 以降をターゲットにする場合は [neptune1.2] を選択します。
5. [Type] (タイプ) リストで、[DB Parameter Group] (DB パラメータグループ) または [DB Cluster Parameter Group] (DB クラスターのパラメータグループ) を選択します。
6. [グループ名] ボックスに、新しい DB パラメータグループの名前を入力します。
7. [説明] ボックスに、新しい DB パラメータグループの説明を入力します。
8. [作成] を選択します。

また、AWS CLI を使用して、新しいパラメーターグループを作成することもできます。

```
aws neptune create-db-parameter-group \
  --db-parameter-group-name (a name for the new DB parameter group) \
  --db-parameter-group-family (either neptune1 or neptune1.2, depending on the engine
 version) \
  --description (a description for the new DB parameter group)
```

Amazon Neptune パラメータ

パラメータグループのパラメータを使用して Amazon Neptune のデータベース設定を管理します。Neptune データベースの設定には次のパラメータを利用できます。

クラスターレベルのパラメータ

- neptune_enable_audit_log
- neptune_enable_slow_query_log
- neptune_slow_query_log_threshold
- neptune_lab_mode
- neptune_query_timeout
- neptune_streams
- neptune_streams_expiry_days
- neptune_lookup_cache
- neptune_autoscaling_config
- neptune_ml_iam_role
- neptune_ml_endpoint

インスタンスレベルのパラメータ

- neptune_dfe_query_engine
- neptune_query_timeout
- neptune_result_cache

廃止されたパラメータ

- neptune_enforce_ssl

neptune_enable_audit_log (クラスターレベルパラメータ)

このパラメータは、Neptune の監査ログを切り替えます。

指定できる値は 0 (無効) と 1 (有効) です。デフォルト値は、「0」です。

このパラメータは静的です。つまり、このパラメータを変更しても、再起動するまでどのインスタンスにも反映されません。

CLI を使用して Neptune 監査ログを CloudWatch Logs に発行する で説明されているように、監査ログを Amazon CloudWatch に公開できます。

neptune_enable_slow_query_log (クラスターレベルパラメータ)

このパラメーターを使用して、Neptune のスロークエリロギング機能を有効または無効にします。

これは動的パラメータです。つまり、値を変更しても DB クラスターを再起動する必要はなく、再起動の原因にもなりません。

許可された値は次のとおりです:

- **info** — スロークエリロギングを有効にし、パフォーマンスの低下の原因となっている可能性のある特定の属性をログに記録します。
- **debug** — スロークエリロギングを有効にし、実行されたクエリで使用可能なすべての属性をログに記録します。
- **disable** — スロークエリロギングを無効にします。

デフォルト値は、「disable」です。

CLI を使用して Neptune スロークエリログを CloudWatch Logs に発行する で説明されているように、スロークエリログを Amazon CloudWatch に公開できます。

neptune_slow_query_log_threshold (クラスターレベルパラメータ)

このパラメータは、実行時間のしきい値をミリ秒単位で指定します。この値を過ぎると、クエリはスロークエリとみなされます。スロークエリロギングが有効になっている場合、このしきい値より長く実行されたクエリは、一部の属性とともにログに記録されます。

デフォルト値は 5,000 ミリ秒 (5 秒) です。

これは動的パラメータです。つまり、値を変更しても DB クラスターを再起動する必要はなく、再起動の原因にもなりません。

neptune_lab_mode (クラスターレベルパラメータ)

設定すると、このパラメータにより Neptune の特定の実験機能が有効になります。現在利用可能な実験機能については、「Neptune ラボモード」を参照してください。

このパラメータは静的です。つまり、このパラメータを変更しても、再起動するまでどのインスタンスにも反映されません。

実験機能を有効または無効にするには、このパラメータに *(###)*=enabled または *(### #)*=disabled を含めます。次のようにコンマで区切って、複数の機能を有効または無効にすることができます。

(## #1 #)=enabled,*(## #2 #)*=enabled

ラボモード機能は、通常、デフォルトで無効化されます。例外は DFEQueryEngine 機能で、Neptune エンジンリリース 1.0.5.0 からはクエリヒント (DFEQueryEngine=viaQueryHint) で使用するためにデフォルトで有効になりました。Neptune エンジンリリース 1.1.1.0 以降、DFE エンジンはラボモードではなくなり、インスタンスの DB パラメータグループの neptune_dfe_query_engine インスタンスパラメータを使用して制御されるようになりました。

neptune_query_timeout (クラスターレベルパラメータ)

グラフクエリの特定のタイムアウト時間をミリ秒単位で指定します。

指定できる値の範囲は 10 ~ 2,147,483,647 (2^{31} - 1) です。デフォルト値は 120,000 (2 分) です。

このパラメータは静的です。つまり、このパラメータを変更しても、再起動するまでどのインスタンスにも反映されません。

> **Note**
>
> 特にサーバーレスインスタンスでは、クエリのタイムアウト値を高く設定しすぎると、予期しないコストが発生する可能性があります。妥当なタイムアウト設定がないと、クエリは予想よりもはるかに長く実行され続け、予想もしなかったコストが発生する可能性があります。これは、クエリの実行中に大規模で高価なインスタンスタイプにスケールアップする可能性があるサーバーレスインスタンスに特に当てはまります。
> ほとんどのクエリに対応し、異常に長い実行でもタイムアウトが発生するだけのクエリタイムアウト値を使用することで、このような予期しない出費を回避できます。

neptune_streams (クラスターレベルパラメータ)

Neptune Streams を有効または無効にします。

このパラメータは静的です。つまり、このパラメータを変更しても、再起動するまでどのインスタンスにも反映されません。

指定できる値は 0 (無効、デフォルト)、および 1 (有効) です。

neptune_streams_expiry_days (クラスターレベルパラメータ)

サーバーがストリームレコードを削除するまでの日数を指定します。

有効な値は 1～90 です。デフォルトは 7 です。

このパラメータはエンジンバージョン 1.2.0.0 で導入されました。

このパラメータは静的です。つまり、このパラメータを変更しても、再起動するまでどのインスタンスにも反映されません。

neptune_lookup_cache (クラスターレベルパラメータ)

R5d インスタンスで Neptune lookup cache. を無効化または最有効化します。

このパラメータは静的です。つまり、このパラメータを変更しても、再起動するまでどのインスタンスにも反映されません。

指定できる値は enabled と disabled です。デフォルト値は disabled ですが、DB クラスター内に R5d インスタンスが作成される際は常に、neptune_lookup_cache パラメータは自動的に enabled に設定され、そのインスタンスにルックアップキャッシュが作成されます。

neptune_autoscaling_config (クラスターレベルパラメータ)

Neptune auto-scaling が作成し管理するリードレプリカインスタンスの構成パラメータを設定します。

このパラメータは静的です。つまり、このパラメータを変更しても、再起動するまでどのインスタンスにも反映されません。

neptune_autoscaling_config パラメータの値として設定した JSON 文字列を使用して以下を指定できます。

- 作成するすべての新しいリードレプリカインスタンスに対して Neptune auto-scaling が使用するインスタンスタイプ。
- これらのリードレプリカに割り当てられたメンテナンスウィンドウ。
- すべての新しいリードレプリカに関連付けるタグ。

JSON 文字列には、次のような構造があります。

```
"{
  \"tags\": [
    { \"key\" : \"reader tag-0 key\", \"value\" : \"reader tag-0 value\" },
    { \"key\" : \"reader tag-1 key\", \"value\" : \"reader tag-1 value\" },
  ],
  \"maintenanceWindow\" : \"wed:12:03-wed:12:33\",
  \"dbInstanceClass\" : \"db.r5.xlarge\"
}"
```

文字列内の引用符はバックスラッシュ文字 (\) を使ってエスケープする必要があります。

neptune_autoscaling_config パラメータで指定されていない 3 つの構成設定のいずれかは、DB クラスターのプライマリライターインスタンスの設定からコピーされます。

neptune_ml_iam_role (クラスターレベルパラメータ)

Neptune ML で使用される IAM ロール ARN を指定します。値には、有効な IAM ロール ARN を指定できます。

このパラメータは静的です。つまり、このパラメータを変更しても、再起動するまでどのインスタンスにも反映されません。

機械学習のためにグラフ上でデフォルトの IAM ロール ARN を指定できます。

neptune_ml_endpoint (クラスターレベルパラメータ)

Neptune ML に使用するエンドポイントを指定します。値には任意の有効な SageMaker エンドポイント名を指定できます。

このパラメータは静的です。つまり、このパラメータを変更しても、再起動するまでどのインスタンスにも反映されません。

機械学習のためにグラフ上でデフォルトの SageMaker エンドポイント を指定できます。

neptune_dfe_query_engine (インスタンスレベルパラメータ)

Neptune エンジンリリース 1.1.1.0 以降、この DB インスタンスパラメータは DFE クエリエンジンの使用方法を制御するために使用されます。許容値は、次のとおりです。

このパラメータは静的です。つまり、このパラメータを変更しても、再起動するまでどのインスタンスにも反映されません。

- **enabled** — useDFE クエリヒントが存在し、false に設定されている場合を除き、可能な限り DFE エンジンを使用するようにします。
- **viaQueryHint** (デフォルト) — DFE エンジンは、true に設定された useDFE クエリヒントを明示的に含むクエリにのみ使用されます。

このパラメータが明示的に設定されていない場合、インスタンスの起動時にデフォルト値の viaQueryHint が使用されます。

> **Note**
>
> すべての openCypher クエリは、このパラメータがどのように設定されているかにかかわらず, DFE エンジンによって実行されます。

リリース 1.1.1.0 より前のリリースでは、これは DB インスタンスパラメータではなくラボモードパラメータでした。

neptune_query_timeout (インスタンスレベルパラメータ)

この DB インスタンスパラメータでは、1 つのインスタンスのグラフクエリのタイムアウト時間をミリ秒単位で指定します。

このパラメータは静的です。つまり、このパラメータを変更しても、再起動するまでどのインスタンスにも反映されません。

指定できる値の範囲は 10 ~ 2,147,483,647 (2^{31} - 1) です。デフォルト値は 120,000 (2 分) です。

> **Note**
>
> 特にサーバーレスインスタンスでは、クエリのタイムアウト値を高く設定しすぎると、予期しないコストが発生する可能性があります。妥当なタイムアウト設定がないと、クエリは

予想よりもはるかに長く実行され続け、予想もしなかったコストが発生する可能性があります。これは、クエリの実行中に大規模で高価なインスタンスタイプにスケールアップする可能性があるサーバーレスインスタンスに特に当てはまります。
ほとんどのクエリに対応し、異常に長い実行でもタイムアウトが発生するだけのクエリタイムアウト値を使用することで、このような予期しない出費を回避できます。

`neptune_result_cache` (インスタンスレベルパラメータ)

`neptune_result_cache` — この DB インスタンスパラメータは、クエリ結果の使用 を有効化または無効化します。

このパラメータは静的です。つまり、このパラメータを変更しても、再起動するまでどのインスタンスにも反映されません。

指定できる値は `0` (無効、デフォルト)、および 1 (有効) です。

`neptune_enforce_ssl` (非推奨のクラスターレベルパラメータ)

(廃止) かつては Neptune への HTTP 接続を許可するリージョンがあり、このパラメータが 1 に設定されている場合、すべての接続に HTTPS を使用するよう強制するために使用されました。ただし、Neptune はすべてのリージョンで HTTPS 接続のみを受け入れるようになったため、このパラメータはもはや関連しません。

AWS Management Console を使用して Neptune DB クラスターを起動する

新しい Neptune DB クラスターを起動する最も簡単な方法は、DB クラスターを作成する で説明されているように、必要なリソースをすべて作成する AWS CloudFormation テンプレートを使用することです。

必要に応じて、ここで説明するように、Neptune コンソールを使用して新しい DB クラスターを手動で起動することもできます。

Neptune コンソールにアクセスして Neptune クラスターを作成するには、Neptune のアクセス許可を持つ IAM ユーザーの作成 で説明されているように、そのために必要なアクセス許可を持つ IAM ユーザーを作成します。

次に、その IAM ユーザーとして AWS Management Console にログインし、以下の手順に従って新しい DB クラスターを作成します。

コンソールを使用して Neptune DB クラスターを起動するには

1. AWS マネジメントコンソールにサインインして Amazon Neptune コンソール (https://console.aws.amazon.com/neptune/home) を開きます。
2. [データベース] ページに移動し、[データベースの作成] を選択すると、[データベースの作成] ページが開きます。
3. [エンジンオプション] で、エンジンタイプは `neptune` であり。特定のエンジンバージョンを選択するか、デフォルトを受け入れることができます。
4. [設定] で、新しい DB クラスターの名前を入力するか、そこに表示されているデフォルト名を受け入れます。この名前はインスタンスのエンドポイントアドレスで使用され、次の制約を満たす必要があります。

   - 1 ~ 63 文字の英数字またはハイフンを使用する必要があります。
   - 1 字目は文字である必要があります。
   - ハイフンを、文字列の最後に使用したり、2 つ続けて使用したりすることはできません。
   - 特定の AWS リージョン内の AWS アカウントのすべての DB インスタンスで一意である必要があります。
5. [テンプレート] で、[実稼働] または [開発とテスト] のいずれかを選択します。
6. [DB インスタンスサイズ] で、インスタンスサイズを選択します。これによって、新しい DB クラスターのプライマリ書き込みインスタンスの処理およびメモリ容量が決まります。

[実稼働] テンプレートを選択した場合、一覧に表示されている使用可能なメモリ最適化クラスからしか選択できませんが、[開発とテスト] を選択した場合は、より経済的なバースタブルクラスから選択することもできます (バースタブルクラスの説明については T3 バースト可能インスタンス を参照)。

> **Note**
>
> Neptune エンジンリリース 1.1.0.0 から、Neptune は R4 インスタンスタイプをサポートしなくなりました。

7. [可用性と耐久性] で、マルチアベイラビリティーゾーン (マルチ AZ) デプロイを有効にするかどうかを選択できます。実稼働テンプレートでは、デフォルトでマルチ AZ 配置が有効ですが、開発およびテストテンプレートでは有効になっていません。マルチ AZ 配置が有効になっている場合、Neptune は、可用性を高めるために、さまざまなアベイラビリティーゾーン (AZ) に作成したリードレプリカインスタンスを配置します。

8. [接続] で、新しい DB クラスターをホストする仮想プライベートクラウド (VPC) を使用可能な選択肢から選択します。Neptune に自動的に VPC を作成させる場合は、ここで [新しい VPC を作成] を選択できます。Neptune インスタンスにアクセスするには、この同じ VPC 内に Amazon EC2 インスタンスを作成する必要があります (詳細については、すべての Amazon Neptune DB クラスターは Amazon VPC に存在します を参照)。DB クラスターの作成後は VPC を変更できないことに注意してください。

   必要に応じて、[その他の接続設定] でクラスターの接続をさらに設定できます。

   a. [サブネットグループ] で、新しい DB クラスターで使用する Neptune DB サブネットグループを選択できます。VPC にサブネットグループがまだない場合は、Neptune によって DB サブネットグループが作成されます (すべての Amazon Neptune DB クラスターは Amazon VPC に存在します を参照)。

   b. [VPC セキュリティグループ] で、新しい DB クラスターへのネットワークアクセスを保護する既存の VPC セキュリティグループを 1 つ以上選択するか、Neptune に作成してもらいたい場合は [新規作成] を選択して、新しい VPC セキュリティグループの名前を指定します (VPC コンソールを使用してセキュリティグループを作成する を参照)。

   c. [データベースポート] で、データベースがアプリケーション接続に使用する TCP/IP ポートを入力します。Neptune は、デフォルトとしてポート番号 8182 を使用します。

9. Neptune ワークベンチで Jupyter Notebook を Neptune に自動的に作成させる場合は、[ノートブック設定] で [ノートブックの作成] を選択します (Neptune グラフノートブックを使用して、

すぐに使用を開始する および Neptune ワークベンチを使用して Neptune ノートブックをホストする を参照)。次に、新しいノートブックの設定方法を選択できます。

a. [ノートブックインスタンスタイプ] で、ノートブックで使用できるインスタンスクラスの中から選択します。

b. [ノートブック名] に、ノートブックの名前を入力します。

c. 必要に応じて、[説明 - オプション] にノートブックの説明を入力することもできます。

d. [IAM ロール名] で、Neptune にノートブック用の IAM ロールを作成させることを選択して、新しいロールの名前を入力するか、使用可能なロールの中から既存の IAM ロールを選択します。

e. 最後に、ノートブックをインターネットに直接接続するか、Amazon SageMaker 経由で接続するか、NAT ゲートウェイを備えた VPC 経由で接続するかを選択します。詳細については、「ノートブックインスタンスを VPC 内のリソースに接続する」を参照してください。

10. [タグ] で、最大 50 個のタグを新しい DB クラスターに関連付けることができます。

11. [追加設定] には、新しい DB クラスターに対して行うことができる設定が他にもあります (多くの場合、スキップしてデフォルト値をそのまま使用できます)。

オプション	できること
DB インスタンス識別子	クラスターのライターインスタンスの名前を指定できます。指定しない場合は、クラスター名に基づくデフォルトの識別子が使用されます。指定する場合は、AWS アカウントが現在のリージョンで所有しているすべての DB インスタンスについて一意な名前を指定します。DB インスタンス識別子では大文字と小文字が区別されませんが、すべて小文字で保存されます。
DB クラスターのパラメータグループ	DB クラスターパラメータグループを選択して、クラスター内のすべての DB インスタンスのデフォルト設定を定義します。特に選択しない限り、Neptune はデフォルトの DB クラスターパラメータグループを使用します。パラメータグループの詳細については、

オプション	できること
	「Amazon Neptune パラメータグループ」を参照してください。
DB パラメータグループ	DB パラメータグループを選択して、クラスター内のプライマリ DB インスタンスの設定を定義します。特に選択しない限り、Neptune はデフォルトのパラメータグループを使用します。パラメータグループの詳細については、「パラメータグループ」を参照してください。
IAM DB authentication	[IAM DB 認証を有効にする] をオンにした場合、データベースへのすべてのアクセスが AWS Identity and Access Management (IAM) を使用して認証されます。 **Important** その場合、AWS 署名バージョン 4 の署名を使用してすべてのリクエストに署名する必要があります。詳細については、「Amazon Neptune での AWS Identity and Access Management (IAM) の概要」を参照してください。
フェイルオーバー優先順位	`No preference` またはフェイルオーバーの優先階層を選択します。1 つの階層を選択し、階層内で競合がある場合、プライマリインスタンスと同じサイズのレプリカが選択されます。

オプション	できること
バックアップの保存期間	Neptune がこの DB インスタンスの自動バックアップを保持する期間を 1 ~ 35 日の範囲で選択します。ポイントインタイム復元 (PITR) は、バックアップ保持期間内の時点にのみ実行できます。
Copy tags to snapshots	(デフォルトで有効) このオプションにより、DB クラスターに関連付けられているすべてのタグがスナップショットにコピーされます。
Enable encryption	(デフォルトで有効) このオプションでは、DB クラスターのデータが保管時に暗号化されます。 その場合、このデータベースボリュームの暗号化に使用されるキーを保護するために使用されるマスターキーを選択します。デフォルトの `aws/rds` キーを選択するか、アカウントのマスターキーから選択するか、別のアカウントのキーの ARN を入力できます。IAM コンソールの [暗号化キー] タブで、新しいマスター暗号化キーを作成することができます。詳細については、「保管時の Neptune リソースの暗号化」を参照してください。
[監査ログ]	DB クラスターの監査ログを CloudWatch ログに公開したい場合は、これをチェックします。

オプション	できること
マイナーバージョン自動アップグレードの有効化	(デフォルトで有効) このオプションを使用すると、DB クラスターは、新しいマイナーエンジンバージョンのリリース後に自動的にアップグレードされます。自動アップグレードは、データベースのメンテナンス期間中に実行されます。「AutoMinorVersionUpgrade の使用」を参照してください。
メンテナンスウィンドウ	DB インスタンスクラスへの変更や自動エンジンパッチなど、保留中の変更を DB クラスターに適用したい特定の期間を選択できます。このようなメンテナンス操作はすべて、選択した期間内に開始され、完了します。期間を選択しない場合、Neptune はメンテナンス期間を任意に割り当てます。
削除保護の有効化	(デフォルトで有効) 削除保護は、DB クラスターの削除を防止します。DB クラスターを削除するには、明示的に無効にする必要があります。

12. [データベースの作成] を選択して、新しい Neptune DB クラスターとそのプライマリインスタンスを起動します。

    Amazon Neptune コンソールでは、新しい DB クラスターが DB クラスターのリストに表示されます。DB クラスターが作成されて使用できるようになるまで、DB クラスターのステータスは [作成中] となります。ステータスが [使用可能] に変わったら、DB クラスターのプライマリインスタンスに接続できます。DB インスタンスクラスと割り当てられたストレージによっては、新しいインスタンスを使用できるようになるまで数分かかることがあります。

    新しく作成したクラスターを表示するには、Neptune コンソールで [Databases] (データベース) ビューを選択します。

    > **Note**
    >
    > AWS Management Console を使用して、DB クラスター内のすべての Neptune DB インスタンスを削除した場合、コンソールは DB クラスターそのものを自動的に削除し

ます。AWS CLI または SDK を使用している場合は、最後のインスタンスを削除した後、DB クラスターを手動で削除する必要があります。

[クラスターエンドポイント] の値をメモしておきます。この値は、Neptune DB クラスターに接続する際に必要になります。

Amazon Neptune DB クラスターの停止と開始

Amazon Neptune クラスターの停止と開始は、開発とテスト環境のコスト管理に役立ちます。クラスターを使用するたびに、すべての DB インスタンスを設定および解放するのではなく、クラスターですべての DB インスタンスを一時的に停止することができます。

トピック

Neptune DB クラスターの停止と開始の概要

Neptune クラスターが必要ではない期間は、そのクラスターですべてのインスタンスを一度に停止することができます。クラスターを使用する必要がある時はいつでもクラスターを開始できます。開始と停止は、継続的な可用性を必要としない開発、テスト、または類似のアクティビティに使用されるクラスターのセットアップと解放のプロセスを簡素化します。クラスター内のインスタンス数に関係なく、1 回のアクションでこれを AWS Management Console で実行できます。

DB クラスターの停止中は、指定された保持期間内のクラスターストレージ、手動のスナップショット、および自動化されたバックアップストレージに対してのみ課金されます。DB インスタンス時間に対しては請求されません。

Neptune は、7 日後に DB クラスターを自動的に開始するため、必要なメンテナンスの更新が遅れ過ぎてしまうことはありません。

ライトにロードされた Neptune クラスターの料金を最小限に抑えるために、そのリードレプリカのすべてを削除するのはなくクラスターを停止することができます。2 つまたは 3 つ以上のインスタンスを持つクラスターの場合、頻繁に DB インスタンスを削除して再作成するのは、AWS CLI または Neptune API を使用することが唯一実用的です。また、適切な順序で削除を実行するのは困難な場合があります。たとえば、フェイルオーバーメカニズムのアクティブ化を避けるためには、プライマリインスタンスを削除する前にすべてのリードレプリカを削除する必要があります。

DB クラスターの実行を維持する必要があるが、容量を減らしたい場合は、開始と停止を使用しないでください。クラスターのコストが高すぎる、または非常にビジー状態でない場合は、1 つ以上の DB インスタンスを削除するか、より小さなインスタンスクラスを使用するように DB インスタンスを変更することはできますが、個別の DB インスタンスを停止することはできません。

Neptune DB クラスターの停止

しばらく使用しない場合は、実行中の Neptune DB クラスターを停止し、必要なときに再度開始できます。クラスターの停止中に、指定された保持期間内のクラスターストレージ、手動のスナップショット、および自動化されたバックアップストレージに対しては課金されますが、DB インスタンス時間に対しては課金されません。

停止オペレーションは、すべてのクラスターのリードレプリカインスタンスを停止してから、プライマリインスタンスを停止して、フェイルオーバーメカニズムのアクティブ化を回避します。

AWS Management Console の使用による DB クラスターの停止

AWS Management Console を使用して Neptune クラスターを停止するには

1. AWS マネジメントコンソールにサインインして Amazon Neptune コンソール (https://console.aws.amazon.com/neptune/home) を開きます。
2. ナビゲーションペインで、[クラスター] を選択し、クラスターを選択します。このページで停止オペレーションを実行するか、停止する DB クラスターの詳細ページに移動します。
3. [アクション] で [停止] を選択します。

AWS CLI の使用による DB クラスターの停止

AWS CLI を使用して DB インスタンスを停止するには、stop-db-cluster コマンドを呼び出し、`--db-cluster-identifier` パラメータを使用して、停止する DB クラスターを指定します。

Example

```
aws neptune stop-db-cluster --db-cluster-identifier mydbcluster
```

Neptune 管理 API の使用による DB クラスターの停止

管理 API を使用して DB インスタンスを停止するには、StopDBCluster API を呼び出し、`DBClusterIdentifier` パラメータを使用して、停止する DB クラスターを指定します。

DB クラスターが停止している間に発生する処理

- スナップショットからクラスターを復元できます (「DB クラスタースナップショットの復元」を参照)。
- DB クラスターまたはその DB インスタンスの設定を変更することはできません。

- クラスターに対して DB インスタンスを追加または削除することはできません。
- まだ DB インスタンスが関連付けられている場合は、クラスターを削除できません。
- 通常、ほとんどの管理アクションを実行するには、停止した DB クラスターを再び開始する必要があります。
- Neptune は、スケジュールされたメンテナンスが再び開始されるとすぐに、クラスターに適用します。7 日後、Neptune は停止したクラスターを自動的に再開始するため、そのメンテナンス状態より過剰に遅れることはありません。
- Neptune は、停止した DB クラスターの自動バックアップを実行しません。これは、クラスターの停止中に基になるデータを変更できないためです。
- Neptune は、DB クラスターの停止中にそのバックアップ保持期間を延長しません。

停止した Neptune DB クラスターの開始

開始できるのは、停止状態の Neptune DB クラスターのみです。クラスターを開始すると、すべての DB インスタンスが再び利用可能になります。クラスターは、エンドポイント、パラメータグループ、および VPC セキュリティグループなどの構成設定を保持します。

AWS Management Console を使用して、停止した DB クラスターを開始する

1. AWS マネジメントコンソールにサインインして Amazon Neptune コンソール (https://console.aws.amazon.com/neptune/home) を開きます。
2. ナビゲーションペインで、[クラスター] を選択し、クラスターを選択します。このページかで開始オペレーションを実行するか、DB クラスターの詳細ページに移動し、そこから開始します。
3. [アクション] で [開始] を選択します。

AWS CLI を使用して、停止した DB クラスターを開始する

AWS CLI を使用して停止した DB クラスターを開始するには、`--db-cluster-identifier` パラメータを使用して start-db-cluster コマンドを呼び出し、開始する DB クラスターを指定します。DB クラスターの作成時に選択したクラスター名を指定するか、最後に追加された `-cluster` で、選択した DB インスタンス名を使用します。

Example

```
aws neptune start-db-cluster --db-cluster-identifier mydbcluster
```

Neptune 管理 API を使用して、停止した DB クラスターを開始する

Neptune 管理 API を使用して Neptune DB クラスターを開始するには、`DBCluster` パラメータを使用して StartDBCluster API を呼び出し、開始する DB クラスターを指定します。DB クラスターの作成時に選択したクラスター名を指定するか、最後に追加された `-cluster` で、選択した DB インスタンス名を使用します。

高速リセット API を使用して Amazon Neptune DB クラスターを空にする

Neptune 高速リセット REST API を使用すると、Neptune グラフをすばやく簡単にリセットし、すべてのデータを削除できます。

これは、%db_reset ラインマジックを使用して、Neptune Notebook 内で行うことができます。

> **Note**
>
> この機能は、Neptune エンジンリリース 1.0.4.0 から利用できます。

- ほとんどの場合、高速リセット操作は数分以内に完了します。期間は、オペレーションの開始時のクラスターの負荷によって多少異なる場合があります。
- 高速リセット操作では、I/O は追加されません。
- 高速リセット後、ストレージボリュームのサイズは縮小しません。代わりに、新しいデータが挿入されると、ストレージが再利用されます。つまり、高速リセット操作の前後に作成されるスナップショットのボリュームサイズは同じになります。高速リセット操作の前後に作成されたスナップショットを使用して復元されたクラスターのボリュームサイズも同じになります。
- リセットオペレーションの一環として、DB クラスター内のすべてのインスタンスが再起動されます。

  > **Note**
  >
  > まれな状況では、これらのサーバーの再起動によってクラスターのフェイルオーバーが発生することもあります。

> **Important**
>
> 高速リセットを使用すると、Neptune DB クラスターと他のサービスの統合が壊れる可能性があります。例:
>
> - 高速リセットは、データベースからすべてのストリームデータを削除し、ストリームを完全にリセットします。つまり、ストリームコンシューマーは新たな設定をしなければ動作しなくなる可能性があります。

- 高速リセットは、Neptune ML によって使用されている SageMaker リソースに関するすべてのメタデータ (ジョブやエンドポイントを含む) を削除します。これらは SageMaker には引き続き存在し、Neptune ML 推論クエリに既存の SageMaker エンドポイントを引き続き使用できますが、Neptune ML 管理 API はそれらでは機能しなくなります。
- ElasticSearch とのフルテキスト検索統合などの統合も高速リセットによって消去され、再度使用する前に手動で確立する必要があります。

API を使用して Neptune DB クラスターからすべてのデータを削除するには

1. まず、データベースのリセットを実行するために使用できるトークンを生成します。この手順は、誰かが誤ってデータベースをリセットするのを防ぐためのものです。

   これを行うには、DB クラスターのライターインスタンス上の `/system` エンドポイントに HTTP `POST` リクエストを送信し、`initiateDatabaseReset` アクションを指定します。

   JSON コンテンツタイプを使用する `curl` コマンドは次のようになります。

   ```
   curl -X POST \
     -H 'Content-Type: application/json' \
         https://your_writer_instance_endpoint:8182/system \
     -d '{ "action" : "initiateDatabaseReset" }'
   ```

   または、`x-www-form-urlencoded` コンテンツタイプを使用します。

   ```
   curl -X POST \
     -H 'Content-Type: application/x-www-form-urlencoded' \
         https://your_writer_instance_endpoint:8182/system \
     -d 'action=initiateDatabaseReset '
   ```

   `initiateDatabaseReset` リクエストは JSON レスポンスで次のようにリセットトークンを返します。

   ```
   {
     "status" : "200 OK",
     "payload" : {
       "token" : "new_token_guid"
     }
   }
   ```

トークンは、発行後 1 時間 (60 分) 有効です。

リクエストをリーダーインスタンスまたはステータスエンドポイントに送信すると、Neptune は ReadOnlyViolationException をスローします。

initiateDatabaseReset リクエストを複数送信した場合、実際にリセットを実行する 2 番目のステップでは、生成された最新のトークンのみが有効になります。

サーバーが、initiateDatabaseReset リクエストの直後に再起動した場合、生成されたトークンが無効になるため、新しいトークンを取得するには新しいリクエストを送信する必要があります。

2. 次に、initiateDatabaseReset から戻ったトークンとともに performDatabaseReset リクエストを DB クラスターのライターインスタンス上の /system エンドポイントへ送ります。これにより、DB クラスターからすべてのデータが削除されます。

   JSON コンテンツタイプを使用する curl コマンドは次のようになります。

   ```
   curl -X POST \
     -H 'Content-Type: application/json' \
         https://your_writer_instance_endpoint:8182/system \
     -d '{
           "action" : "performDatabaseReset",
           "token" : "token_guid"
         }'
   ```

   または、x-www-form-urlencoded コンテンツタイプを使用します。

   ```
   curl -X POST \
     -H 'Content-Type: application/x-www-form-urlencoded' \
         https://your_writer_instance_endpoint:8182/system \
     -d 'action=performDatabaseReset&token=token_guid'
   ```

   リクエストは JSON レスポンスを返します。リクエストが受け入れられる場合、応答は次のようになります。

   ```
   {
     "status" : "200 OK"
   }
   ```

送信したトークンが発行されたトークンと一致しない場合、応答は次のようになります。

```
{
  "code" : "InvalidParameterException",
  "requestId":"token_guid",
  "detailedMessage" : "System command parameter 'token' : 'token_guid' does not
 match database reset token"
}
```

リクエストが受け入れられ、リセットが開始されると、サーバーは再起動してデータを削除します。DB クラスターのリセット中は、DB クラスターに他のリクエストを送信することはできません。

IAM-Auth での高速リセット API の使用

DB クラスターで Iam-Auth が有効になっている場合は、awscurl を使用し、IAM-auth 認証される高速リセットコマンドを送信します。

awscurl を使用して IAM-auth で高速リセットリクエストを送信する

1. AWS_ACCESS_KEY_ID および AWS_SECRET_ACCESS_KEY 環境変数を正しく設定します (一時的な資格情報を使用している場合は AWS_SECURITY_TOKEN も設定します) 。
2. `initiateDatabaseReset` リクエストは次のようになります。

```
awscurl -X POST --service neptune-db "$SYSTEM_ENDPOINT" \
  -H 'Content-Type: application/json' --region us-west-2 \
  -d '{ "action" : "initiateDatabaseReset" }'
```

3. `performDatabaseReset` リクエストは次のようになります。

```
awscurl -X POST --service neptune-db "$SYSTEM_ENDPOINT" \
  -H 'Content-Type: application/json' --region us-west-2 \
  -d '{ "action" : "performDatabaseReset" }'
```

Neptune Workbench **%db_reset** ラインマジックを使用して DB クラスターをリセットする

Neptune workbench は、Neptune ノートブックで高速なデータベースのリセットを実行できる `%db_reset` ラインマジックに対応しています。

パラメータなしでマジックを呼び出すと、クラスター内のすべてのデータを削除するかどうかを尋ねる画面が表示され、削除後にクラスターデータが利用できなくなることを確認するチェックボックスが表示されます。その時点で、先に進んでデータを削除するか、操作をキャンセルするかを選択できます。

もっと危険な選択肢は、`--yes` または `-y` オプションで `%db_reset` を呼び出すことで、これ以上のプロンプトを表示せずに削除が実行されます。

REST API と同様に、2 つのステップでリセットを実行することもできます。

```
%db_reset --generate-token
```

レスポンスは次のとおりです。

```
{
  "status" : "200 OK",
  "payload" : {
    "token" : "new_token_guid"
  }
}
```

次の操作を実行します。

```
%db_reset --token new_token_guid
```

レスポンスは次のとおりです。

```
{
  "status" : "200 OK"
}
```

高速リセット操作の一般的なエラーコード

Neptune エラーコード	HTTP status	Message	例
`InvalidParameterException`	400	システムコマンドパラメータ「*#####*」にはサポートされていない値「*XXX*」があります	無効なパラメータ
`InvalidParameterException`	400	次の値が多すぎます:*#####*	複数のアクションを含む高速リセットリクエストが「Content-type:application/x-www-form-urlencoded」のヘッダー付きで送信されました
`InvalidParameterException`	400	フィールド「action」が重複しています	複数のアクションを含む高速リセットリクエストが「Content-Type: application/json」のヘッダー付きで送信されました
`MethodNotAllowedException`	400	間違ったルート: /*bad_endpoint*	リクエストが間違ったエンドポイントに送信されました
`MissingParameterException`	400	必須パラメータがありません:[action]	高速リセットリクエストには、必要な「action」パラメータが含まれていません

Neptune エラーコード	HTTP status	Message	例
`ReadOnlyViolationException`	400	リードレプリカインスタンスでの書き込みは許可されていません	高速リセットリクエストがリーダーまたはステータスエンドポイントに送信されました
`AccessDeniedException`	403	認証トークンが見つかりません	IAM-Auth が有効になっている DB エンドポイントに、正しいシグニチャなしで高速リセットリクエストが送信されました
`ServerShutdownException`	500	データベースのリセットが進行中です。クラスターが使用可能になったら、クエリを再試行してください。	高速リセットが開始されると、既存および着信する Gremlin/Sparql クエリはエラーとなります。

DB クラスターに Neptune リーダーインスタンスを追加する

Neptune DB クラスターには、1 つのプライマリ DB インスタンスと最大 15 の Neptune リーダーインスタンスがあります。プライマリ DB インスタンスでは、読み書きオペレーションをサポートしており、クラスターボリュームに対してすべてのデータ変更を実行します。Neptune リーダーインスタンスは、プライマリ DB インスタンスと同じストレージボリュームに接続し、読み取りオペレーションのみサポートします。

リーダーインスタンスを使用して、プライマリ DB インスタンスから読み取りワークロードをオフロードします。

DB クラスターのプライマリインスタンスと Neptune リーダーを複数のアベイラビリティーゾーンに分散して、DB クラスターの可用性を改善することをお勧めします。

次のセクションでは、DB クラスターにリーダーインスタンスを作成する方法について説明します。

コンソールを使用して Neptune リーダーインスタンスを作成する

Neptune DB DB クラスターのプライマリインスタンスを作成した後、Neptune コンソールを使用して、さらに Neptune リーダーインスタンスを追加できます。

AWS Management Console を使用して Neptune リーダーインスタンスを作成するには

1. AWS マネジメントコンソールにサインインして Amazon Neptune コンソール (https://console.aws.amazon.com/neptune/home) を開きます。
2. ナビゲーションペインで、[Databases] (データベース) を選択します。
3. リーダーインスタンスを作成する DB クラスターを選択します。
4. [アクション] を選択し、[リーダーを追加] を選択します。
5. [Create replica DB instance] (レプリカ DB インスタンスの作成) ページで、Neptune レプリカのオプションを指定します。次の表は、Neptune リードレプリカの設定を示しています。

使用するオプション	操作
DB インスタンスクラス	Neptune レプリカの処理要件やメモリ要件を定義する DB インスタンスクラスを選択します。さまざまなリージョンで Neptune が提供する DB インスタンスクラスの最新のリストについては、Neptune の料金ページを参照してください。
アベイラビリティーゾーン	アベイラビリティーゾーンを指定します。プライマリ DB インスタンスとは異なるゾーンを選択します。リストには、DB クラスターの DB サブネットグループによってマッピングされたアベイラビリティーゾーンのみが含まれます。
暗号化	暗号化を有効または無効にします。
リードレプリカのソース	Neptune レプリカを作成するプライマリインスタンスの識別子を選択します。
DB インスタンス識別子	インスタンス名を入力します。選択した リージョン内で、自分のアカウントに対して一意であることが必要です。名前には、選択したアベイラビリティーゾー

使用するオプション	操作
	ンなどを含めると理解しやすくなります (neptune-us-east-1c など)。
データベースポート	データベースが接続を受け入れるポート番号。
DB パラメータグループ	このインスタンスのパラメータグループ。
ログのエクスポート	公開するログを選択します (ある場合)。
Auto Minor Version Upgrade	Neptune レプリカで Neptune DB エンジンのマイナーバージョンアップグレードをリリースと同時に自動的に受信するには、[Yes] (はい) を選択します。 マイナーバージョン自動アップグレードオプションは、マイナーアップグレードにのみ適用されます。エンジンメンテナンスパッチには適用されません。エンジンメンテナンスパッチは、システムの安定性を維持するために常に自動的に適用されます。

6. Neptune レプリカインスタンスを作成するには、[Create read replica] (リードレプリカの作成) を選択します。

DB クラスターから Neptune リーダーインスタンスを削除するには、「Amazon Neptune の DB インスタンスの削除」の説明に従います。

コンソールを使用した Neptune DB クラスターの変更

AWS Management Console を使用して DB インスタンスを変更する場合、[すぐに適用] を選択することで、変更内容をすぐに適用できます。変更の即時適用を選択した場合は、新しい変更と、保留中の変更キューにあるすべての変更が一度に適用されます。

変更の即時適用を選択しない場合、この変更は保留中の変更キューに保存されます。次のメンテナンスウィンドウ実行中に、キューのすべての保留中の変更が適用されます。

> **Important**
>
> ダウンタイムを必要とする保留中の変更がある場合、変更をすぐに適用するように選択すると、該当する DB インスタンスで予想外のダウンタイムが発生することがあります。DB クラスターの他の DB インスタンスではダウンタイムはありません。

> **Note**
>
> Neptune で DB クラスターを変更する場合、[Apply Immediately] (すぐに適用) 設定による影響を受けるのは、[DB cluster identifier]、[IAM DB authentication] への変更のみです。その他の変更はすべて、すぐに適用設定の値に関係なく、ただちに適用されます。

コンソールを使用してクラスターを変更するには

1. AWS マネジメントコンソールにサインインして Amazon Neptune コンソール (https://console.aws.amazon.com/neptune/home) を開きます。
2. ナビゲーションペインで、[Clusters (クラスター)] を選択して、変更する DB クラスターを選択します。
3. [Actions]、[Modify cluster] の順に選択します。[DB クラスターの変更] ページが表示されます。
4. 必要に応じて任意の設定を変更してください。

   > **Note**
   >
   > コンソールで、インスタンスレベルの変更は、現在の DB インスタンスにのみ適用される場合や、DB クラスター全体に適用される場合があります。インスタンスレベルで DB

クラスター全体を変更する設定をコンソールで変更するには、「DB クラスター内の DB インスタンスの変更」の手順に従います。

5. すべての変更が正しいことを確認したら、[Continue] を選択して概要を確認します。
6. 変更をすぐに適用するには、[すぐに適用] を選択します。
7. 確認ページで、変更内容を確認します。正しい場合は、[クラスターの変更] を選択して変更を保存します。

   変更を編集する場合は [Back] を選択し、変更をキャンセルする場合は [Cancel] を選択します。

DB クラスター内の DB インスタンスの変更

コンソールを使用して DB クラスター内の DB インスタンスを変更するには

1. AWS マネジメントコンソールにサインインして Amazon Neptune コンソール (https://console.aws.amazon.com/neptune/home) を開きます。
2. ナビゲーションペインで [Instances] を選択し、変更する DB インスタンスを選択します。
3. [Instance actions] を選択してから、[Modify] を選択します。[Modify DB Instance] ページが表示されます。
4. 必要に応じて任意の設定を変更してください。

   Note

   一部の設定は DB クラスター全体に適用されるため、クラスターレベルで変更する必要があります。これらの設定を変更するには、「コンソールを使用した Neptune DB クラスターの変更」の手順に従います。
   AWS Management Console で、インスタンスレベルの変更は、現在の DB インスタンスにのみ適用される場合や、DB クラスター全体に適用される場合があります。

5. すべての変更が正しいことを確認したら、[Continue] を選択して概要を確認します。
6. 変更をすぐに適用するには、[すぐに適用] を選択します。
7. 確認ページで、変更内容を確認します。変更内容が正しい場合は、[Modify DB Instance] (DB インスタンスを変更) を選択して変更を保存します。

   変更を編集する場合は [Back] を選択し、変更をキャンセルする場合は [Cancel] を選択します。

Amazon Neptune でのパフォーマンスとスケーリング

Neptune DB クラスターおよびインスタンスは、次の 3 つのレベルでスケーリングされます。

- ストレージのスケーリング
- インスタンススケーリング;
- 読み取りのスケーリング

Neptune でのストレージのスケーリング

Neptune ストレージは、クラスターボリューム内のデータに合わせて自動的にスケーリングします。データが増大するに従って、クラスターボリュームストレージはサポートされている全リージョンで最大 128 TiB まで拡張されます。ただし、中国と GovCloud では 64 TiB に制限されています。

クラスターボリュームのサイズは 1 時間ごとにチェックされ、ストレージコストが決定されます。

Neptune データベースによって消費されるストレージは、毎月 GB 単位で課金され、消費される I/O は 100 万リクエスト単位で課金されています。料金は Neptune データベースで使用したストレージおよび IO の分のみ発生し、事前にプロビジョニングする必要はありません。

料金の情報については、Neptune の製品ページを参照してください。

Neptune でのインスタンススケーリング

DB クラスター内の各 DB インスタンスの DB インスタンスクラスを変更することで、必要に応じて Neptune DB クラスターをスケーリングできます。Neptune は、いくつかの最適化された DB インスタンスクラスをサポートしています。

Neptune での読み込みのスケーリング

Neptune DB クラスターの読み取りのスケーリングは、最大 15 個の Neptune レプリカを DB クラスター内に作成することで実現できます。各 Neptune レプリカは、最小限のレプリカラグでクラスターボリュームから同じデータを返します (多くの場合、このラグは、プライマリインスタンスが更新を書き込んだ後、100 ミリ秒を大幅に下回ります)。読み取りトラフィックが増えたら、追加の Neptune レプリカを作成し、それらに直接接続することで DB クラスターの読み取りワークロードを分散できます。Neptune レプリカの DB インスタンスクラスは、プライマリイスタンスと同じものである必要はありません。

DB クラスターに Neptune レプリカを追加する方法については、リーダーインスタンスを追加する を参照してください。

Amazon Neptune DB クラスター内のレプリカの数の Auto-scaling

Neptune 自動スケーリングを使用すると、DB クラスター内の Neptune レプリカの数を自動的に調整して、接続およびワークロードの要件を満たすことができます。自動スケーリングを使用すると、Neptune DB クラスターでワークロードの増加を処理できます。その後、ワークロードが減少すると、自動スケーリングによって不要なレプリカが削除されるため、未使用の容量に対して料金が発生することはありません。

auto-scaling は、すでに 1 つのプライマリライターインスタンスと少なくとも 1 つのリードレプリカインスタンスを持つ Neptune DB クラスターでのみ使用できます (Amazon Neptune DB クラスターとインスタンス を参照)。また、クラスター内のすべての read-replica インスタンスが使用可能な状態である必要があります。read-replica が使用可能以外の状態にある場合、クラスター内のすべての read-replica が使用可能になるまで、Neptune 自動スケーリングは何もしません。

新しいクラスターを作成する必要がある場合は、DB クラスターを作成する を参照してください。

を使用して AWS CLI、スケーリングポリシーを定義し、DB クラスターに適用します。を使用して AWS CLI 、自動スケーリングポリシーを編集または削除することもできます。ポリシーでは、次の自動スケーリングパラメータを指定します。

- クラスター内に存在するレプリカの最小数と最大数。
- レプリカ (複数可) - 追加スケーリングアクティビティ間のScaleOutCooldown間隔、およびレプリカ (複数可) - 削除スケーリングアクティビティ間のScaleInCooldown間隔。
- スケールアップまたはスケールダウンの CloudWatch メトリクスとメトリクストリガー値。

Neptune 自動スケーリングアクションの頻度は、いくつかの方法で抑えられます。

- 最初に、自動スケーリングでリーダーを追加または削除するには、CPUUtilization 高アラームを少なくとも 3 分間超えるか、低アラームを少なくとも 15 分間超える必要があります。
- その最初の追加または削除の後、その後の Neptune 自動スケーリングアクションの頻度は、自動スケーリングポリシーの ScaleOutCooldown および ScaleInCooldown 設定によって制限されます。

使用している CloudWatch メトリクスがポリシーで指定した高いしきい値に達し、最後の自動スケーリングアクションからScaleOutCooldown間隔が経過し、DB クラスターに設定したレプリカの最大数がまだない場合、Neptune 自動スケーリングは DB クラスターのプライマリインスタンスと同じインスタンスタイプを使用して新しいレプリカを作成します。

同様に、メトリクスが指定した下限しきい値に達し、前回の自動スケーリングアクションから ScaleInCooldown 間隔が経過し、DB クラスターに指定した最小レプリカ数を超えるレプリカがある場合、Neptune 自動スケーリングはレプリカの 1 つを削除します。

> **Note**
>
> Neptune auto-scaling では、自身が作成したレプリカのみ削除されます。既存のレプリカは削除されません。

neptune_autoscaling_config DB クラスターパラメータを使用して、Neptune auto-scaling が作成する新しい read-replica のインスタンスタイプ、それらの read-replica のメンテナンスウィンドウ、および各新しい read-replica に関連付けるタグを指定することもできます。これらの構成設定は、neptune_autoscaling_config パラメータの JSON 文字列の値として次のように指定します。

```
"{
  \"tags\": [
    { \"key\" : \"reader tag-0 key\", \"value\" : \"reader tag-0 value\" },
    { \"key\" : \"reader tag-1 key\", \"value\" : \"reader tag-1 value\" },
  ],
  \"maintenanceWindow\" : \"wed:12:03-wed:12:33\",
  \"dbInstanceClass\" : \"db.r5.xlarge\"
}"
```

JSON 文字列内の引用符はすべてバックスラッシュ文字 (\) で エスケープする必要があります。文字列内のすべての空白は、通常どおり、任意です。

neptune_autoscaling_config パラメータで指定されていない 3 つの構成設定のいずれかは、DB クラスターのプライマリライターインスタンスの設定からコピーされます。

自動スケーリングが新しいリードレプリカインスタンスを追加する際、DB インスタンス ID の接頭句に autoscaled-reader (例えば、autoscaled-reader-7r7t7z3lbd-20210828) を付けます。また、作成するすべてのリードレプリカに、キー autoscaled-reader と TRUE の値でタグを追加します。このタグは AWS Management Consoleの DB インスタンスの詳細ページの [Tags] (タグ) タブで確認できます。

```
 "key" : "autoscaled-reader",  "value" : "TRUE"
```

auto-scaling によって作成されたすべての read-replica インスタンスのプロモーション層は、15 デフォルトでは優先順位が最も低くなります。つまり、フェイルオーバー時には、手動で作成されたものなど、優先順位の高いレプリカが最初に昇格されます。Neptune DB クラスターの耐障害性 を参照してください。

Neptune 自動スケーリングは、Neptune `CPUUtilization` CloudWatch メトリクスを事前定義されたメトリクスとして使用するターゲット追跡スケーリングポリシーを持つ Application Auto Scaling を使用して実装されます。

Neptune サーバーレス DB クラスターでの自動スケーリングの使用

Neptune サーバーレスは、需要がインスタンスの容量を超えたとき、Neptune 自動スケーリングよりもはるかに迅速に対応し、別のインスタンスを追加する代わりにインスタンスをスケールアップします。自動スケーリングが比較的安定したワークロードの増減に対応するように設計されているのに対して、サーバーレスは需要の急激な増加や変動への対応に優れています。

両者の長所を理解すれば、自動スケーリングとサーバーレスを組み合わせて、ワークロードの変化を効率的に処理し、コストを最小限に抑えながら需要を満たす柔軟なインフラストラクチャを構築できます。

自動スケーリングをサーバーレスと効果的に連携させるには、需要の急増や短期的な変化に対応できるように、サーバーレスクラスターの `maxNCU` 設定を十分に高く設定することが重要です。そうしないと、一時的な変更によってサーバーレススケーリングがトリガーされず、自動スケーリングによって不要な追加インスタンスが大量に発生する可能性があります。`maxNCU` を十分に高く設定すると、サーバーレススケーリングはそれらの変更をより迅速かつ低コストで処理できます。

Amazon Neptune auto-scaling を有効にする方法

自動スケーリングは、 AWS CLIを使用する Neptune DB クラスターでのみ有効にできます。 AWS Management Consoleを使用して自動スケーリングを有効にすることはできません。

また、自動スケーリングは、次の Amazon リージョンではサポートされていません。

- アフリカ (ケープタウン): `af-south-1`
- 中東 (アラブ首長国連邦): `me-central-1`
- AWS GovCloud （米国東部): `us-gov-east-1`
- AWS GovCloud （米国西部): `us-gov-west-1`

Neptune DB クラスターauto-scaling を有効にするには、次の 3 つのステップが必要です。

1. Application Auto Scaling を使用して DB クラスターを登録する

Neptune DB クラスターに対して auto-scaling を有効にする最初のステップは、Application Auto Scaling を使用して、クラスターをApplication Auto Scaling に登録することです。このとき、 AWS CLI または Application Auto Scaling SDK の 1 つを使用します。クラスターには、すでに 1 つのプライマリインスタンスと少なくとも 1 つの read-replica インスタンスが必要です。

例えば、1 つから 8 つの追加のレプリカで自動スケーリングするクラスターを登録するには、次のように コマンドを使用できます AWS CLI `register-scalable-target`。

```
aws application-autoscaling register-scalable-target \
  --service-namespace neptune \
  --resource-id cluster:(your DB cluster name) \
  --scalable-dimension neptune:cluster:ReadReplicaCount \
  --min-capacity 1 \
  --max-capacity 8
```

これは、`RegisterScalableTarget` Application Auto Scaling API オペレーションを使用することと同じです。

AWS CLI `register-scalable-target` コマンドでは、以下のパラメータを使用します。

- **`service-namespace`** – `neptune` に設定します。

  このパラメータは、Application Auto Scaling API にある `ServiceNamespace` パラメータと同じです。
- **`resource-id`** — これを Neptune DB クラスターのリソース識別子に設定します。リソースタイプは `cluster` で、コロン (':') が、それから DB クラスターの名前が続きます。

  このパラメータは、Application Auto Scaling API にある `ResourceID` パラメータと同じです。
- **`scalable-dimension`** — この場合のスケーラブルディメンションは、DB クラスター内のレプリカインスタンスの数であるため、このパラメータを `neptune:cluster:ReadReplicaCount` に設定します。

  このパラメータは、Application Auto Scaling API にある `ScalableDimension` パラメータと同じです。
- **`min-capacity`** – アプリケーションの Auto Scaling で管理するリーダー DB レプリカの最小数。この値は 0 から 15 までの範囲に設定される必要があり、`max-capacity` の Neptune レプリカの

最大数として指定された値以下である必要があります。自動スケーリングが機能するには、DB クラスターに少なくとも 1 つのリーダーが必要です。

このパラメータは、Application Auto Scaling API にある `MinCapacity` パラメータと同じです。

- **`max-capacity`** — アプリケーションの自動スケーリングによって管理される既存のインスタンスと新しいインスタンスを含めて、DB クラスター内のリーダー DB レプリカインスタンスの最大数。この値は 0 から 15 までの範囲に設定される必要があり、`min-capacity` の Neptune レプリカの最小数として指定された値以上である必要があります。

  `max-capacity` AWS CLI パラメータは、Application Auto Scaling API の `MaxCapacity`パラメータと同等です。

DB クラスターを登録すると、Application Auto Scaling によって `AWSServiceRoleForApplicationAutoScaling_NeptuneCluster` サービスにリンクされたロールが生成されます。詳細については、Application Auto Scaling ユーザーガイドのApplication auto-scaling のサービスにリンクされたロールを参照してください。

2. DB クラスターで使用するAuto Scaling ポリシーを定義します。

ターゲット追跡スケーリングポリシーは、テキストファイルに保存できる JSON テキストオブジェクトとして定義されます。Neptune の場合、このポリシーは現在、 という名前の事前定義されたCPUUtilization CloudWatch メトリクスとしてのみ Neptune メトリクスを使用できますNeptuneReaderAverageCPUUtilization。

次に、Neptune のターゲット追跡スケーリング設定ポリシーの例を示します。

```
{
  "PredefinedMetricSpecification": { "PredefinedMetricType":
 "NeptuneReaderAverageCPUUtilization" },
  "TargetValue": 60.0,
  "ScaleOutCooldown" : 600,
  "ScaleInCooldown" : 600
}
```

ここで **`TargetValue`** 要素には、スケールアウト (つまり、より多くのレプリカを追加します)、その下でスケールイン (つまり、レプリカを削除します) をオートスケーリングする CPU 使用率のパーセンテージがこの上に含まれています。この場合、スケーリングをトリガーする目標パーセンテージは 60.0% です。

ScaleInCooldown 要素はスケールインアクティビティが完了してから別のスケールインが開始されるまでの時間 (秒単位) を指定します。デフォルトは 300 秒です。ここで、値 600 は、あるレプリカの削除が完了してから別のレプリカの開始までの間に 10 分以上経過する必要があることを指定します。

ScaleOutCooldown 要素はスケールアウトアクティビティが完了してから別のスケールアウトが開始されるまでの時間 (秒単位) を指定します。デフォルトは 300 秒です。ここで、値 600 は、あるレプリカの追加が完了してから別のレプリカの開始までの間に 10 分以上経過する必要があることを指定します。

-**DisableScaleIn** 要素はブール値であり、存在しており `true` に設定いれば、スケールインを完全に無効にします。つまり、auto-scaling ではレプリカが追加される可能性がありますが、レプリカは削除されません。デフォルトでは、スケールインは有効になっており、`DisableScaleIn` は `false` です。

Neptune DB クラスターを アプリケーションの Auto Scaling に登録し、テキストファイルに JSON スケーリングポリシーを定義した後、登録された DB クラスターにスケーリングポリシーを適用します。コマンドを使用して AWS CLI `put-scaling-policy`、次のようなパラメータでこれを行うことができます。

```
aws application-autoscaling put-scaling-policy \
  --policy-name (name of the scaling policy) \
  --policy-type TargetTrackingScaling \
  --resource-id cluster:(name of your Neptune DB cluster) \
  --service-namespace neptune \
  --scalable-dimension neptune:cluster:ReadReplicaCount \
  --target-tracking-scaling-policy-configuration file://(path to the JSON configuration
 file)
```

auto-scaling ポリシーを適用すると、DB クラスターで auto-scaling が有効になります。

コマンドを使用して AWS CLI `put-scaling-policy`、既存の自動スケーリングポリシーを更新することもできます。

Application Auto Scaling API リファレンスのPutScaling「ポリシー」も参照してください。 Auto Scaling

Neptune DB クラスターから auto-scaling を削除する

Neptune DB クラスターから自動スケーリングを削除するには、 AWS CLI delete-scaling-policy コマンドと deregister-scalable-target コマンドを使用します。

Amazon Neptune DB クラスターのメンテナンス

Neptune は、使用するすべてのリソースに対して、以下のようなメンテナンスを定期的に実施します。

- 必要に応じた基盤となるハードウェアの交換。これはバックグラウンドで処理され、ユーザーは何もする必要がなく、通常、ユーザーの操作には影響しません。
- 基盤となるオペレーティングシステムの更新。DB クラスター内のインスタンスのオペレーティングシステムのアップグレードは、パフォーマンスとセキュリティを向上させるために行われるため、通常はできるだけ早く完了する必要があります。通常、アップデートには約 10 分かかります。オペレーティングシステムのアップデートでは、DB インスタンスの DB エンジンのバージョンまたは DB インスタンスクラスは変更されません。

  一般的に、最初に DB クラスターのリーダーインスタンスを更新し、次にライターインスタンスを更新するのが最善です。リーダーとライターを同時に更新すると、フェイルオーバーが発生してダウンタイムが生じる可能性があります。DB インスタンスはオペレーティングシステムの更新前に自動的にバックアップされないため、オペレーティングシステムの更新を適用する前に必ず手動でバックアップしてください。
- Neptune データベースエンジンの更新。Neptune は、新機能や機能強化の導入、バグの修正を目的として、さまざまなエンジンアップデートを定期的にリリースします。

エンジンバージョン番号

エンジンリリース 1.3.0.0 より前のバージョン番号

2019 年 11 月までは、Neptune がサポートするエンジンバージョンは一度に 1 つのみで、エンジンのバージョン番号はすべて `1.0.1.0.200<xxx>` という形式でした。ここで `xxx` はパッチ番号を示します。すべての新しいエンジンバージョンは、以前のバージョンへのパッチとしてリリースされました。

2019 年 11 月に、Neptune は複数のバージョンのサポートを開始し、お客様がアップグレードパスをより詳細に制御できるようになりました。これに伴って、エンジンのリリースの番号付けも変更されました。

2019 年 11 月からエンジンリリース 1.3.0.0 までのエンジンバージョン番号は 5 つの部分で構成されています。バージョン番号 `1.0.2.0.R2` を例として取り上げます。

- 最初の部分は常に 1 でした。

- 2 番目の部分 (1.0.2.0.R2 の 0) は、データベースのメジャーバージョン番号でした。
- 3 番目と 4 番目の部分 (1.0.2.0.R2 の 2.0) はどちらもマイナーバージョン番号でした。
- 5 番目の部分 (1.0.2.0.R2 の R2) はパッチ番号でした。

ほとんどのアップデートはパッチアップデートであり、パッチとマイナーバージョンアップデートの区別は必ずしも明確ではありませんでした。

エンジンリリース 1.3.0.0 以降のバージョン番号

エンジンリリース 1.3.0.0 以降、Neptune はエンジンアップデートの番号付けと管理の方法を変更しました。

エンジンのバージョン番号は、次の 4 つの部分で構成され、それぞれがリリースのタイプに対応しています。

```
#######.#########.#########.########
```

以前はパッチとしてリリースされていた非破壊的変更が、現在はマイナーバージョンとしてリリースされ、AutoMinorVersionUpgrade インスタンス設定を使用して管理できるようになりました。

これにより、新しいマイナーバージョンがリリースされるたびに通知を受け取ることができます。そのためには、RDS-EVENT-0156 イベントにサブスクライブします (「Neptune イベント通知にサブスクライブする」を参照してください)。

現在、パッチリリースは緊急のターゲット修正に限定されており、バージョン番号の最後の部分 (*.*.*.1、*.*.*.2 など) を使用して番号が付けられます。

Amazon Neptune のさまざまなタイプのエンジンリリース

エンジンバージョン番号の 4 つの部分に対応するエンジンリリースの 4 つのタイプは次のとおりです。

- 製品バージョン — 製品の機能やインターフェースに抜本的かつ根本的な変更が加えられた場合にのみ変更されます。現在の Neptune 製品バージョンは 1 です。
- メジャーバージョン — メジャーバージョンでは重要な新機能や互換性を破る変更が導入されます。通常 2 年以上の有効期間があります。
- マイナーバージョン — マイナーバージョンには新機能、機能強化、バグ修正が含まれますが、互換性を破るような変更は含まれません。次回のメンテナンスウィンドウ中に自動的に適用するかどうかを選択できます。また、リリースされるたびに通知を受けるように選択することもできます。

- パッチバージョン — パッチバージョンは、緊急のバグ修正または重大なセキュリティアップデートに対応するためにのみリリースされます。互換性を破る変更が含まれることはめったになく、リリース後の次回のメンテナンスウィンドウ中に自動的に適用されます。

Amazon Neptune のメジャーバージョンアップデート

メジャーバージョンアップデートでは、通常 1 つ以上の重要な新機能が導入され、多くの場合、互換性を破る変更が含まれます。通常、サポート期間は約 2 年間です。Neptune のメジャーバージョンは、リリース日および推定サポート期間と共に、エンジンリリースに記載されます。

使用しているメジャーバージョンのサポートが終了するまでは、メジャーバージョンの更新は完全に任意です。新しいメジャーバージョンにアップグレードする場合は、「メジャーバージョンのアップグレード」で説明しているように、AWS CLI または Neptune コンソールを使用して新しいバージョンを自分でインストールする必要があります。

ただし、使用しているメジャーバージョンがサポート終了になると、より新しいメジャーバージョンへのアップグレードが必要であることが通知されます。通知後の猶予期間内にアップグレードしない場合、次回のメンテナンスウィンドウ中に最新のメジャーバージョンへのアップグレードが自動的にスケジュールされます。詳細については、「エンジンバージョンの有効期間」を参照してください。

Amazon Neptune のマイナーバージョンアップデート

Neptune エンジンのほとんどの更新はマイナーバージョンアップデートです。これらは頻繁に発生し、互換性を破る変更は含まれません。

DB クラスターのライター (プライマリ) インスタンスで `AutoMinorVersionUpgrade` フィールドを `true` に設定している場合、マイナーバージョンアップデートは、リリース後の次回のメンテナンスウィンドウ中に DB クラスター内のすべてのインスタンスに自動的に適用されます。

DB クラスターのライターインスタンスで `AutoMinorVersionUpgrade` フィールドを `false` に設定している場合は、明示的にインストールした場合にのみ更新が適用されます。

> **Note**
>
> マイナーバージョンアップデートは、自己完結型 (同じメジャーバージョンへの以前のマイナーバージョンアップデートには依存しない) および累積的 (以前のマイナーバージョンアップデートで導入されたすべての機能と修正を含む) です。つまり、以前のマイナーバージョンアップデートをインストールしたかどうかに関係なく、どのマイナーバージョンアップデートでもインストールできます。

マイナーバージョンのリリースは、RDS-EVENT-0156 イベントにサブスクライブすることで簡単に追跡できます (「Neptune イベント通知にサブスクライブする」を参照)。これにより、新しいマイナーバージョンがリリースされるたびに通知されます。

また、通知にサブスクライブしているかどうかに関係なく、いつでも保留中のアップデートを確認できます。

Amazon Neptune のパッチバージョンアップデート

インスタンスの信頼性に影響するセキュリティ上の問題やその他の重大な不具合が発生した場合、Neptune は必須のパッチをデプロイします。これらは次回のメンテナンスウィンドウ中に DB クラスター内のすべてのインスタンスに適用されます。ユーザーが介入する必要はありません。

パッチリリースは、デプロイしない場合のリスクが、デプロイした場合のリスクやダウンタイムを上回る場合にのみデプロイされます。パッチリリースは頻繁に発生するものではありません (通常数か月に 1 回程度です)。また、メンテナンスウィンドウのごく一部を使用するだけです。

Amazon Neptune のメジャーエンジンバージョンの有効期間の計画

Neptune のエンジンバージョンは、ほとんどの場合、暦四半期の終わりに有効期限を迎えます。例外が発生するのは、セキュリティや可用性に関する重要な問題が発生した場合のみです。

エンジンバージョンの有効期間が終了すると、Neptune データベースを新しいバージョンにアップグレードする必要があります。

一般的に、Neptune エンジンバージョンは以下のように引き続きご利用いただけます。

- マイナーエンジンバージョン: マイナーエンジンバージョンは、リリース後少なくとも 6 か月間はご利用いただけます。
- メジャーエンジンバージョン: メジャーエンジンバージョンは、リリース後少なくとも 12 か月間はご利用いただけます。

エンジンバージョンの有効期限の少なくとも 3 か月前に、AWS は、AWS アカウントに登録されているメールアドレスに自動メール通知を送信し、同じメッセージが AWS ヘルスダッシュボードに投稿されます。これにより、アップグレードを計画し、準備する時間を確保できます。

エンジンバージョンの有効期間が終了すると、そのバージョンを使用して新しいクラスターやインスタンスを作成できなくなり、オートスケーリングでもそのバージョンを使用してインスタンスを作成できなくなります。

実際に有効期間が終了したエンジンバージョンは、メンテナンス期間中に自動的にアップグレードされます。エンジンバージョンの有効期限の 3 か月前に送信されるメッセージには、自動的にアップグレードされるバージョン、DB クラスターへの影響、推奨アクションなど、この自動更新の内容に関する詳細が記載されています。

> **Important**
>
> データベースエンジンのバージョンを最新の状態に保つ責任はお客様にあります。AWS は、セキュリティ、プライバシー、および可用性に関する最新の保護措置の恩恵を受けるために、すべてのお客様にデータベースを最新のエンジンバージョンにアップグレードするよう促します。廃止日を過ぎたサポートされていないエンジンまたはソフトウェア (「レガシーエンジン」) でデータベースを運用すると、セキュリティ、プライバシー、およびダウンタイムイベントを含む運用上のリスクにさらされる可能性が高くなります。
> どのエンジンでもデータベースの運用には、AWS サービスの利用を規定する契約が適用されます。レガシーエンジンは一般公開されていません。AWS は、レガシーエンジンについてはサポートを提供せず、AWS は、レガシーエンジンがサービス、AWS、その関連会社、または第三者にセキュリティまたは賠償責任のリスク、または危害のリスクをもたらすと AWS が判断した場合は、いつでもレガシーエンジンへのアクセスまたは使用を制限することができます。レガシーエンジンでコンテンツを引き続き実行すると、コンテンツが利用できなくなったり、破損したり、回復できなくなったりする可能性があります。レガシーエンジンで実行されているデータベースは、サービスレベルアグリーメント (SLA) の例外の対象となります。
> レガシーエンジンで実行されているデータベースおよび関連ソフトウェアには、バグ、エラー、欠陥、または有害なコンポーネントが含まれています。したがって、本契約またはサービス条件にこれと異なる定めがある場合でも、AWS はレガシーエンジンを「現状のまま」提供しています。

Neptune DB クラスターのエンジン更新の管理

> **Note**
>
> 更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、インスタンスでデータベースを再起動する必要があるため、20 ~ 30 秒から数秒のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。インスタンスでは、メンテナンス更新を完了するためにマルチ AZ フェイルオーバーが必要になる場合がまれにあります。

適用に時間がかかるメジャーバージョンアップグレードの場合は、ブルー/グリーンデプロイ戦略を使用してダウンタイムを最小限に抑えることができます。

現在使用しているエンジンバージョンの確認

AWS CLI get-engine-status コマンドを使用して、DB クラスターが現在使用しているエンジンのリリースバージョンを確認できます。

```
aws neptunedata get-engine-status
```

JSON 出力には、次のような "dbEngineVersion" フィールドが含まれています。

```
  "dbEngineVersion": "1.3.0.0",
```

保留中および利用可能な更新を確認する

DB クラスターの保留中の更新は、Neptune コンソールを使用して確認できます。左側の列で [データベース] を選択し、データベースペインで DB クラスターを選択します。保留中の更新は [メンテナンス] 列に表示されます。[アクション]、[メンテナンス] の順に選択すると、次の 3 つの選択肢が表示されます。

- 今すぐアップグレード
- 次回のウィンドウでアップグレード
- アップグレードを延期

AWS CLI を使用して、次のように保留中のエンジンアップデートを一覧表示できます。

```
aws neptune describe-pending-maintenance-actions \
  --resource-identifier (ARN of your DB cluster)
  --region (your region) \
  --engine neptune
```

AWS CLI を使用して、次のように利用可能なエンジンアップデートを一覧表示することもできます。

```
aws neptune describe-db-engine-versions \
  --region (your region) \
```

```
  --engine neptune
```

利用可能なエンジンリリースのリストには、バージョン番号が現在のものよりも高く、アップグレードパスが定義されているリリースのみが含まれます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでは、互換性を破る変更がなくても、コードに影響するような新しい機能や動作が導入される場合があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に、新しいバージョンをテストする最善の方法は、Neptune ブルー/グリーンデプロイソリューションを使用することです。これにより、本番 DB クラスターに影響を与えずに、新しいバージョンでアプリケーションやクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

Neptune メンテナンスウィンドウ

毎週のメンテナンスウィンドウは 30 分間で、その間に、予定されたエンジンの更新やその他のシステム変更が適用されます。ほとんどのメンテナンスイベントは 30 分のウィンドウ中に完了しますが、大規模なメンテナンスイベントは 30 分以上かかる場合があります。

各 DB クラスターには、毎週 30 分のメンテナンスウィンドウがあります。DB クラスターの作成時に時間を指定しないと、Neptune は、ランダムに曜日を選択し、リージョンによって異なる 8 時間の時間ブロックから 30 分をその曜日内に割り当てます。

例えば、いくつかの AWS リージョンで使用されているメンテナンスウィンドウの 8 時間の時間ブロックは次のとおりです。

リージョン	時間ブロック
米国西部 (オレゴン) リージョン	06:00 ~ 14:00 UTC
US West (N. California) リージョン	06:00 ~ 14:00 UTC
米国東部 (オハイオ) リージョン	03:00 ~ 11:00 UTC
欧州 (アイルランド) リージョン	22:00 ~ 06:00 UTC

メンテナンスウィンドウにより、保留中の操作がいつ開始されるかが決まり、ほとんどのメンテナンス操作はそのウィンドウ内に完了します。ただし、大規模なメンテナンスタスクはウィンドウの終了時間を超えて続く場合があります。

DB クラスターのメンテナンスウィンドウの移動

理想的には、メンテナンスウィンドウは、クラスターの使用率が最も低い時間帯に設定する必要があります。現在のウィンドウが該当しない場合は、次のように、より適切な時間に移動できます。

DB クラスターのメンテナンスウィンドウを変更するには

1. AWS マネジメントコンソールにサインインして Amazon Neptune コンソール (https://console.aws.amazon.com/neptune/home) を開きます。
2. ナビゲーションペインで、[データベース] を選択します。
3. メンテナンスウィンドウを変更する DB クラスターを選択します。
4. [Modify] (変更) を選択します。
5. [クラスターを変更] ページの下部にある [さらに表示] を選択します。

6. [優先メンテナンスウィンドウ] セクションで、希望するメンテナンスウィンドウの曜日、時間、期間を設定します。
7. [Next] (次へ) をクリックします。

   確認ページで、変更内容を確認します。
8. 直ちにメンテナンスウィンドウに変更を適用するには、[すぐに適用] を選択します。
9. [送信] を選択して変更を適用します。

   変更を編集する場合は、[戻る] を選択します。変更をキャンセルする場合は [キャンセル] を選択します。

`AutoMinorVersionUpgrade` を使用してマイナーバージョンの自動更新を制御する

> **Important**
>
> `AutoMinorVersionUpgrade` は、エンジンリリース 1.3.0.0 以降のマイナーバージョンアップグレードにのみ有効です。

DB クラスターのライター (プライマリ) インスタンスで `AutoMinorVersionUpgrade` フィールドを `true` に設定している場合、マイナーバージョンの更新は、リリース後の次回のメンテナンスウィンドウで DB クラスター内のすべてのインスタンスに自動的に適用されます。

DB クラスターのライターインスタンスで `AutoMinorVersionUpgrade` フィールドを `false` に設定している場合は、明示的にインストールした場合にのみ更新が適用されます。

> **Note**
>
> パッチリリース (`*.*.*.1`、`*.*.*.2` など) は、`AutoMinorVersionUpgrade` パラメータの設定に関係なく、常に次回のメンテナンスウィンドウ中に自動的にインストールされます。

AWS Management Consoleを使用して、次のように `AutoMinorVersionUpgrade` を設定できます。

Neptune コンソールを使用して **AutoMinorVersionUpgrade** を設定するには

1. AWS マネジメントコンソールにサインインして Amazon Neptune コンソール (https://console.aws.amazon.com/neptune/home) を開きます。
2. ナビゲーションペインで、[Databases] (データベース) を選択します。
3. AutoMinorVersionUpgrade を設定する対象の DB クラスターのプライマリ (ライター) インスタンスを選択します。
4. [Modify] (変更) を選択します。
5. [クラスターを変更] ページの下部にある [さらに表示] を選択します。
6. 展開されたページの下部で、[マイナーバージョン自動アップグレードをオンにする] または [マイナーバージョン自動アップグレードをオフにする] を選択します。
7. [Next] (次へ) をクリックします。

   確認ページで、変更内容を確認します。
8. マイナーバージョン自動アップグレードに変更を適用するには、[すぐに適用] を選択します。
9. [送信] を選択して変更を適用します。

   変更を編集する場合は、[戻る] を選択します。変更をキャンセルする場合は [キャンセル] を選択します。

AWS CLI を使用して AutoMinorVersionUpgrade フィールドを設定することもできます。例えば、true に設定するには、次のようなコマンドを使用できます。

```
aws neptune modify-db-instance \
  --db-instance-identifier (the ID of your cluster's writer instance) \
  --auto-minor-version-upgrade \
  --apply-immediately
```

同様に、false に設定するには、次のようなコマンドを使用します。

```
aws neptune modify-db-instance \
  --db-instance-identifier (the ID of your cluster's writer instance) \
  --no-auto-minor-version-upgrade \
  --apply-immediately
```

Neptune エンジンへの更新の手動インストール

メジャーバージョンのエンジンアップグレードのインストール

メジャーエンジンリリースは常に手動でインストールする必要があります。ダウンタイムを最小限に抑え、テストと検証に十分な時間を確保するために、新しいメジャーバージョンをインストールする最善の方法は、一般的に Neptune ブルー/グリーンデプロイソリューションを使用することです。

場合によっては、DB クラスターの作成時に使用した AWS CloudFormation テンプレートを使用してメジャーバージョンアップグレードをインストールすることもできます (「AWS CloudFormation テンプレートを使用して Neptune DB クラスターのエンジンバージョンを更新する」を参照)。

メジャーバージョンアップデートをすぐにインストールする場合は、次のような CLI コマンドを使用できます。

```
aws neptune modify-db-cluster \
  --db-cluster-identifier (identifier for your neptune cluster) \
  --engine neptune \
  --engine-version (the new engine version) \
  --apply-immediately
```

アップグレード先のエンジンのバージョンを必ず指定します。指定しないと、エンジンは最新バージョンではないか、期待するバージョンとは異なるバージョンにアップグレードされる可能性があります。

`--apply-immediately` の代わりに `--no-apply-immediately` と指定することができます。

クラスターでカスタムクラスターパラメータグループを使用している場合は、必ずこのパラメータを使用することを指定します。

```
  --db-cluster-parameter-group-name (name of the custom DB cluster parameter group)
```

同様に、クラスター内のインスタンスがカスタム DB パラメータグループを使用している場合は、必ずこのパラメータを使用して指定します。

```
  ---db-instance-parameter-group-name (name of the custom instance parameter group)
```

AWS Management Consoleを使用したマイナーバージョンのエンジンアップグレードのインストール

Neptune コンソールを使用してマイナーバージョンアップグレードを実行するには

1. AWS マネジメントコンソールにサインインして Amazon Neptune コンソール (https://console.aws.amazon.com/neptune/home) を開きます。
2. ナビゲーションペインで、[データベース] を選択し、変更する DB クラスターを選択します。
3. [Modify] (変更) を選択します。
4. [インスタンスの仕様] で、アップグレード先の新しいバージョンを選択します。
5. [Next] (次へ) をクリックします。
6. 変更をすぐに適用するには、[すぐに適用] を選択します。
7. [送信] を選択して DB クラスターを更新します。

AWS CLIを使用したマイナーバージョンのエンジンアップグレードのインストール

次のようなコマンドを使用すると、次回のメンテナンスウィンドウを待たずにマイナーバージョンアップグレードを実行できます。

```
aws neptune modify-db-cluster \
  --db-cluster-identifier (your-neptune-cluster) \
  --engine-version (new-engine-version) \
  --apply-immediately
```

AWS CLI を使用して手動でアップグレードする場合は、アップグレード先のエンジンバージョンを必ず含めます。そうしないと、エンジンが最新バージョンまたは期待するバージョンではないバージョンにアップグレードされる可能性があります。

1.2.0.0 より前のバージョンからエンジンバージョン 1.2.0.0 以降へのアップグレード

エンジンリリース 1.2.0.0 では、重要な変更がいくつか導入され、以前のバージョンからのアップグレードが通常よりも複雑になる可能性があります。

- エンジンリリース 1.2.0.0 では、カスタムパラメータグループとカスタムクラスターパラメータグループに新しい形式が導入されました。そのため、1.2.0.0 より前のエンジンバージョンからエンジンバージョン 1.2.0.0 以降にアップグレードする場合は、パラメータグループファミリー

neptune1.2 を使用している既存のカスタムパラメータグループとカスタムクラスターパラメータグループをすべて再作成する必要があります。以前のリリースではパラメータグループファミリー neptune1 が使用されていましたが、それらのパラメータグループはリリース 1.2.0.0 以降では動作しません。詳細については、「Amazon Neptune パラメータグループ」を参照してください。

- エンジンリリース 1.2.0.0 では、UNDO ログの新しい形式も導入されました。そのため、1.2.0.0より前のバージョンからのアップグレードを開始する前に、以前のエンジンバージョンで作成された元に戻すログをすべてパージし、`UndoLogsListSize` CloudWatch メトリクスをゼロにする必要があります。更新を開始しようとしたときに UNDO ログレコードが多すぎる (200,000 以上) 場合、UNDO ログのパージが完了するのを待っている間にアップグレードがタイムアウトすることがあります。

  パージが行われるクラスターのライターインスタンスをアップグレードすることで、パージの速度を上げることができます。アップグレードを試みる前にこれを行うと、開始前に UNDO ログの数を減らすことができます。ライターのサイズを 24XL インスタンスタイプに増やすと、パージ率が 1 時間あたり 100 万レコードを超えることがあります。

  UndoLogsListSize CloudWatch メトリクスが非常に大きい場合、サポートケースを開くと、それを停止するための追加の戦略を検討するのに役立つ場合があります。
- 最後に、リリース 1.2.0.0 には、IAM 認証で Bolt プロトコルを使用していた以前のコードに影響する重大な変更がありました。リリース 1.2.0.0 以降、Bolt には IAM 署名用のリソースパスが必要です。Java では、リソースパスの設定は以下のようになります: `request.setResourcePath("/openCypher"));`。その他の言語では、`/openCypher` をエンドポイント URI に追加できます。例については、「Bolt プロトコルの使用」を参照してください。

AWS CloudFormation テンプレートを使用して Neptune DB クラスターのエンジンバージョンを更新する

Neptune DB クラスターの作成に使用した Neptune AWS CloudFormation テンプレートを再利用して、エンジンバージョンを更新できます。

Neptune エンジンバージョンのアップグレードは、マイナーとメジャーがあります。 AWS CloudFormation テンプレートを使用すると、大きな変更が含まれることが多いメジャーバージョンのアップグレードに役立ちます。メジャーバージョンアップグレードには、既存のアプリケーションとの下位互換性のないデータベースの変更が含まれる場合があるため、アップグレード時にアプリケーションに変更を加える必要がある場合もあります。必ず、アップグレードの前にテストしてください。また、アップグレードの前に、DB クラスターの手動スナップショットを常に作成することを強くお勧めします。

メジャーバージョンごとに個別のエンジンアップグレードを行う必要があることに注意してください。メジャーバージョンをスキップして、次のメジャーバージョンに直接アップグレードすることはできません。

2023 年 5 月 17 日より前は、Neptune AWS CloudFormation スタックを使用してエンジンバージョンをアップグレードした場合、新しい空の DB クラスターが現在のクラスターとしてのみ作成されていました。ただし、2023 年 5 月 17 日以降、Neptune AWS CloudFormation スタックは既存のデータを保持するインプレースエンジンアップグレードをサポートするようになりました。

メジャーバージョンアップグレードの場合、テンプレートで `DBCluster` に次のプロパティを設定する必要があります。

- `DBClusterParameterGroup` (カスタムまたはデフォルト)
- `DBInstanceParameterGroupName`
- `EngineVersion`

同様に、DBCluster にアタッチされた DBInstance に対しても以下を設定する必要があります。

- `DBParameterGroup` (カスタム/デフォルト)

デフォルトかカスタムかにかかわらず、すべてのパラメータグループがテンプレートで定義されていることを確認してください。

カスタムパラメータグループの場合は、既存のカスタムパラメータグループのファミリーが新しいエンジンバージョンと互換性があることを確認してください。1.2.0.0 より前のエンジンバージョンでは、パラメータグループファミリー neptune1 が使用されていましたが、1.2.0.0 以降のエンジンリリースでは、パラメータグループファミリー neptune1.2 が必要です。詳細については、「Amazon Neptune パラメータグループ」を参照してください。

メジャーエンジンバージョンアップグレードの場合、適切なファミリーのパラメータグループを DBCluster DBInstanceParameterGroupName フィールドで指定します。

デフォルトのパラメータグループは、新しいエンジンバージョンと互換性のあるものにアップグレードする必要があります。

Neptune はエンジンのアップグレード後に DB インスタンスを自動的に再起動することに注意してください。

トピック

- 例: 1.2.0.1 から 1.2.0.2 へのマイナーエンジンアップグレード
- 例: デフォルトのパラメータグループによる 1.1.1.0 から 1.2.0.2 へのメジャーバージョンアップグレード
- 例: カスタムパラメータグループによる 1.1.1.0 から 1.2.0.2 へのメジャーバージョンアップグレード
- 例: デフォルトパラメータグループとカスタムパラメータグループを組み合わせた 1.1.1.0 から 1.2.0.2 へのメジャーバージョンアップグレード

例: 1.2.0.1 から 1.2.0.2 へのマイナーエンジンアップグレード

アップグレードする DB クラスターと、そのクラスターの作成に使用したテンプレートを選択します。例:

```
Description: Base Template to create Neptune Stack with Engine Version 1.2.0.1 using
 custom Parameter Groups
Parameters:
  DbInstanceType:
    Description: Neptune DB instance type
    Type: String
    Default: db.r5.large
Resources:
  NeptuneDBClusterParameterGroup:
    Type: 'AWS::Neptune::DBClusterParameterGroup'
```

```
    Properties:
      Family: neptune1.2
      Description: test-cfn-neptune-db-cluster-parameter-group-description
      Parameters:
        neptune_enable_audit_log: 0
  NeptuneDBParameterGroup:
    Type: 'AWS::Neptune::DBParameterGroup'
    Properties:
      Family: neptune1.2
      Description: test-cfn-neptune-db-parameter-group-description
      Parameters:
        neptune_query_timeout: 20000
  NeptuneDBCluster:
    Type: 'AWS::Neptune::DBCluster'
    Properties:
      EngineVersion: 1.2.0.1
      DBClusterParameterGroupName:
        Ref: NeptuneDBClusterParameterGroup
    DependsOn:
      - NeptuneDBClusterParameterGroup
  NeptuneDBInstance:
    Type: 'AWS::Neptune::DBInstance'
    Properties:
      DBClusterIdentifier:
        Ref: NeptuneDBCluster
      DBInstanceClass:
        Ref: DbInstanceType
      DBParameterGroupName:
        Ref: NeptuneDBParameterGroup
    DependsOn:
      - NeptuneDBCluster
      - NeptuneDBParameterGroup
Outputs:
  DBClusterId:
    Description: Neptune Cluster Identifier
    Value:
      Ref: NeptuneDBCluster
```

EngineVersion プロパティを 1.2.0.1 から 1.2.0.2 に更新します。

```
Description: Template to upgrade minor engine version to 1.2.0.2
Parameters:
  DbInstanceType:
```

```
    Description: Neptune DB instance type
    Type: String
    Default: db.r5.large
Resources:
  NeptuneDBClusterParameterGroup:
    Type: 'AWS::Neptune::DBClusterParameterGroup'
    Properties:
      Family: neptune1.2
      Description: test-cfn-neptune-db-cluster-parameter-group-description
      Parameters:
        neptune_enable_audit_log: 0
  NeptuneDBParameterGroup:
    Type: 'AWS::Neptune::DBParameterGroup'
    Properties:
      Family: neptune1.2
      Description: test-cfn-neptune-db-parameter-group-description
      Parameters:
        neptune_query_timeout: 20000
  NeptuneDBCluster:
    Type: 'AWS::Neptune::DBCluster'
    Properties:
      EngineVersion: 1.2.0.2
      DBClusterParameterGroupName:
        Ref: NeptuneDBClusterParameterGroup
    DependsOn:
      - NeptuneDBClusterParameterGroup
  NeptuneDBInstance:
    Type: 'AWS::Neptune::DBInstance'
    Properties:
      DBClusterIdentifier:
        Ref: NeptuneDBCluster
      DBInstanceClass:
        Ref: DbInstanceType
      DBParameterGroupName:
        Ref: NeptuneDBParameterGroup
    DependsOn:
      - NeptuneDBCluster
      - NeptuneDBParameterGroup
Outputs:
  DBClusterId:
    Description: Neptune Cluster Identifier
    Value:
      Ref: NeptuneDBCluster
```

次に、 AWS CloudFormation を使用して、改訂されたテンプレートを実行します。

例: デフォルトのパラメータグループによる 1.1.1.0 から 1.2.0.2 へのメジャーバージョンアップグレード

アップグレードする DBCluster と、その作成に使用したテンプレートを見つけます。例:

```
Description: Base Template to create Neptune Stack with Engine Version 1.1.1.0 using
 default Parameter Groups
Parameters:
  DbInstanceType:
    Description: Neptune DB instance type
    Type: String
    Default: db.r5.large
Resources:
  NeptuneDBCluster:
    Type: 'AWS::Neptune::DBCluster'
    Properties:
      EngineVersion: 1.1.1.0
  NeptuneDBInstance:
    Type: 'AWS::Neptune::DBInstance'
    Properties:
      DBClusterIdentifier:
        Ref: NeptuneDBCluster
      DBInstanceClass:
        Ref: DbInstanceType
    DependsOn:
      - NeptuneDBCluster
Outputs:
  DBClusterId:
    Description: Neptune Cluster Identifier
    Value:
      Ref: NeptuneDBCluster
```

- DBClusterParameterGroupデフォルトを、新しいエンジンバージョンで使用されているパラメータグループファミリーのものに更新します (こちらdefault.neptune1.2)。
- DBCluster にアタッチされた各 DBInstance について、デフォルトの DBParameterGroup を新しいエンジンバージョンで使用されているファミリーのものに更新します (こちら default.neptune1.2)。
- DBInstanceParameterGroupName プロパティをそのファミリーのデフォルトパラメータグループに設定します (こちら default.neptune1.2)。

- EngineVersion プロパティを 1.1.0.0 から 1.2.0.2 に更新します。

テンプレートは次のようになります。

```
Description: Template to upgrade major engine version to 1.2.0.2 by using upgraded
 default parameter groups
Parameters:
  DbInstanceType:
    Description: Neptune DB instance type
    Type: String
    Default: db.r5.large
Resources:
  NeptuneDBCluster:
    Type: 'AWS::Neptune::DBCluster'
    Properties:
      EngineVersion: 1.2.0.2
      DBClusterParameterGroupName: default.neptune1.2
      DBInstanceParameterGroupName: default.neptune1.2
  NeptuneDBInstance:
    Type: 'AWS::Neptune::DBInstance'
    Properties:
      DBClusterIdentifier:
        Ref: NeptuneDBCluster
      DBInstanceClass:
        Ref: DbInstanceType
      DBParameterGroupName: default.neptune1.2
    DependsOn:
      - NeptuneDBCluster
Outputs:
  DBClusterId:
    Description: Neptune Cluster Identifier
    Value:
```

次に、 AWS CloudFormation を使用して、改訂されたテンプレートを実行します。

例: カスタムパラメータグループによる 1.1.1.0 から 1.2.0.2 へのメジャーバージョンアップグレード

アップグレードする DBCluster と、その作成に使用したテンプレートを見つけます。例:

```
Description: Base Template to create Neptune Stack with Engine Version 1.1.1.0 using
 custom Parameter Groups
```

```
Parameters:
  DbInstanceType:
    Description: Neptune DB instance type
    Type: String
    Default: db.r5.large
Resources:
  NeptuneDBClusterParameterGroup:
    Type: 'AWS::Neptune::DBClusterParameterGroup'
    Properties:
      Name: engineupgradetestcpg
      Family: neptune1
      Description: 'NeptuneDBClusterParameterGroup with family neptune1'
      Parameters:
        neptune_enable_audit_log: 0
  NeptuneDBParameterGroup:
    Type: 'AWS::Neptune::DBParameterGroup'
    Properties:
      Name: engineupgradetestpg
      Family: neptune1
      Description: 'NeptuneDBParameterGroup1 with family neptune1'
      Parameters:
        neptune_query_timeout: 20000
  NeptuneDBCluster:
    Type: 'AWS::Neptune::DBCluster'
    Properties:
      EngineVersion: 1.1.1.0
      DBClusterParameterGroupName:
        Ref: NeptuneDBClusterParameterGroup
    DependsOn:
      - NeptuneDBClusterParameterGroup
  NeptuneDBInstance:
    Type: 'AWS::Neptune::DBInstance'
    Properties:
      DBClusterIdentifier:
        Ref: NeptuneDBCluster
      DBInstanceClass:
        Ref: DbInstanceType
      DBParameterGroupName:
        Ref: NeptuneDBParameterGroup
    DependsOn:
      - NeptuneDBCluster
      - NeptuneDBParameterGroup
Outputs:
  DBClusterId:
```

```
    Description: Neptune Cluster Identifier
    Value:
      Ref: NeptuneDBCluster
```

- カスタムDBClusterParameterGroupファミリーを、新しいエンジンバージョンで使用されているものに更新します。) default.neptune1.2。
- にDBInstanceアタッチされている各 についてDBCluster、カスタムDBParameterGroupファミリーを新しいエンジンバージョン (ここでは) で使用されるものに更新しますdefault.neptune1.2。
- DBInstanceParameterGroupName プロパティをそのファミリーのパラメータグループに設定します (こちら default.neptune1.2)。
- EngineVersion プロパティを 1.1.0.0 から 1.2.0.2 に更新します。

テンプレートは次のようになります。

```
Description: Template to upgrade major engine version to 1.2.0.2 by modifying existing
 custom parameter groups
Parameters:
  DbInstanceType:
    Description: Neptune DB instance type
    Type: String
    Default: db.r5.large
Resources:
  NeptuneDBClusterParameterGroup:
    Type: 'AWS::Neptune::DBClusterParameterGroup'
    Properties:
      Name: engineupgradetestcpgnew
      Family: neptune1.2
      Description: 'NeptuneDBClusterParameterGroup with family neptune1.2'
      Parameters:
        neptune_enable_audit_log: 0
  NeptuneDBParameterGroup:
    Type: 'AWS::Neptune::DBParameterGroup'
    Properties:
      Name: engineupgradetestpgnew
      Family: neptune1.2
      Description: 'NeptuneDBParameterGroup1 with family neptune1.2'
      Parameters:
        neptune_query_timeout: 20000
  NeptuneDBCluster:
```

```
    Type: 'AWS::Neptune::DBCluster'
    Properties:
      EngineVersion: 1.2.0.2
      DBClusterParameterGroupName:
        Ref: NeptuneDBClusterParameterGroup
      DBInstanceParameterGroupName:
        Ref: NeptuneDBParameterGroup
    DependsOn:
      - NeptuneDBClusterParameterGroup
  NeptuneDBInstance:
    Type: 'AWS::Neptune::DBInstance'
    Properties:
      DBClusterIdentifier:
        Ref: NeptuneDBCluster
      DBInstanceClass:
        Ref: DbInstanceType
      DBParameterGroupName:
        Ref: NeptuneDBParameterGroup
    DependsOn:
      - NeptuneDBCluster
      - NeptuneDBParameterGroup
Outputs:
  DBClusterId:
    Description: Neptune Cluster Identifier
    Value:
      Ref: NeptuneDBCluster
```

次に、 AWS CloudFormation を使用して、改訂されたテンプレートを実行します。

例: デフォルトパラメータグループとカスタムパラメータグループを組み合わせた 1.1.1.0 から 1.2.0.2 へのメジャーバージョンアップグレード

アップグレードする DBCluster と、その作成に使用したテンプレートを見つけます。例:

```
Description: Base Template to create Neptune Stack with Engine Version 1.1.1.0 using
 custom Parameter Groups
Parameters:
  DbInstanceType:
    Description: Neptune DB instance type
    Type: String
    Default: db.r5.large
Resources:
  NeptuneDBClusterParameterGroup:
```

```
    Type: 'AWS::Neptune::DBClusterParameterGroup'
    Properties:
      Family: neptune1
      Description: 'NeptuneDBClusterParameterGroup with family neptune1'
      Parameters:
        neptune_enable_audit_log: 0
  NeptuneDBParameterGroup:
    Type: 'AWS::Neptune::DBParameterGroup'
    Properties:
      Family: neptune1
      Description: 'NeptuneDBParameterGroup with family neptune1'
      Parameters:
        neptune_query_timeout: 20000
  NeptuneDBCluster:
    Type: 'AWS::Neptune::DBCluster'
    Properties:
      EngineVersion: 1.1.1.0
      DBClusterParameterGroupName:
        Ref: NeptuneDBClusterParameterGroup
    DependsOn:
      - NeptuneDBClusterParameterGroup
  CustomNeptuneDBInstance:
    Type: 'AWS::Neptune::DBInstance'
    Properties:
      DBClusterIdentifier:
        Ref: NeptuneDBCluster
      DBInstanceClass:
        Ref: DbInstanceType
      DBParameterGroupName:
        Ref: NeptuneDBParameterGroup
    DependsOn:
      - NeptuneDBCluster
      - NeptuneDBParameterGroup
  DefaultNeptuneDBInstance:
    Type: 'AWS::Neptune::DBInstance'
    Properties:
      DBClusterIdentifier:
        Ref: NeptuneDBCluster
      DBInstanceClass:
        Ref: DbInstanceType
    DependsOn:
      - NeptuneDBCluster
Outputs:
  DBClusterId:
```

```
    Description: Neptune Cluster Identifier
    Value:
      Ref: NeptuneDBCluster
```

- カスタムクラスターパラメータグループの場合は、DBClusterParameterGroup ファミリーを新しいエンジンバージョンに対応するもの、つまり neptune1.2 に更新します。
- デフォルトのクラスターパラメータグループの場合、DBClusterParameterGroup を新しいエンジンバージョンに対応するもの、つまり default.neptune1.2 に更新します。
- DBCluster にアタッチされる各 DBInstance について、デフォルトの DBParameterGroup を新しいエンジンバージョンで使用されるファミリーのもの (こちら default.neptune1.2) に更新し、カスタムパラメータグループを新しいエンジンバージョンによってサポートされるファミリーを使用するもの (こちら neptune1.2) に更新します。
- DBInstanceParameterGroupName プロパティを、新しいエンジンバージョンによってサポートされているファミリーのパラメータグループに設定します。

テンプレートは次のようになります。

```
Description: Template to update Neptune Stack to Engine Version 1.2.0.1 using custom
 and default Parameter Groups
Parameters:
  DbInstanceType:
    Description: Neptune DB instance type
    Type: String
    Default: db.r5.large
Resources:
  NeptuneDBClusterParameterGroup:
    Type: 'AWS::Neptune::DBClusterParameterGroup'
    Properties:
      Family: neptune1.2
      Description: 'NeptuneDBClusterParameterGroup with family neptune1.2'
      Parameters:
        neptune_enable_audit_log: 0
  NeptuneDBParameterGroup:
    Type: 'AWS::Neptune::DBParameterGroup'
    Properties:
      Family: neptune1.2
      Description: 'NeptuneDBParameterGroup1 with family neptune1.2'
      Parameters:
        neptune_query_timeout: 20000
  NeptuneDBCluster:
```

```
    Type: 'AWS::Neptune::DBCluster'
    Properties:
      EngineVersion: 1.2.0.2
      DBClusterParameterGroupName:
        Ref: NeptuneDBClusterParameterGroup
      DBInstanceParameterGroupName: default.neptune1.2
    DependsOn:
      - NeptuneDBClusterParameterGroup
  CustomNeptuneDBInstance:
    Type: 'AWS::Neptune::DBInstance'
    Properties:
      DBClusterIdentifier:
        Ref: NeptuneDBCluster
      DBInstanceClass:
        Ref: DbInstanceType
      DBParameterGroupName:
        Ref: NeptuneDBParameterGroup
    DependsOn:
      - NeptuneDBCluster
      - NeptuneDBParameterGroup
  DefaultNeptuneDBInstance:
    Type: 'AWS::Neptune::DBInstance'
    Properties:
      DBClusterIdentifier:
        Ref: NeptuneDBCluster
      DBInstanceClass:
        Ref: DbInstanceType
      DBParameterGroupName: default.neptune1.2
    DependsOn:
      - NeptuneDBCluster
Outputs:
  DBClusterId:
    Description: Neptune Cluster Identifier
    Value:
      Ref: NeptuneDBCluster
```

次に、 AWS CloudFormation を使用して、改訂されたテンプレートを実行します。

Neptune のデータベースのクローン化

DB のクローン作成を使用すると、Amazon Neptune ですべてのデータベースのクローンを迅速かつ高いコスト効率で作成できます。クローンデータベースの初回作成時に必要な追加スペースは最小限です。データベースのクローン作成では、 コピーオンライトプロトコルが使用されます。データは、ソースデータベースまたはクローンデータベースのいずれかで、変更時にコピーされます。同じ DB クラスターから複数のクローンを作成できます。また、そのほかのクローンから追加のクローンを作成することもできます。Neptune ストレージのコンテキストにおけるコピーオンライトプロトコルの詳しい動作については、コピーオンライトプロトコル を参照してください。

DB クローン作成はさまざまなユースケースで使用でき、特に、次のような稼働環境で影響を及ぼさないために役立ちます。

- スキーマの変更やパラメータグループの変更など、変更の影響を試したり評価したりする場合。
- データのエクスポートや分析クエリの実行など、大量のワークロードを扱うオペレーションを実行する場合。
- 開発やテストの目的で試験的な環境で本番稼働用の DB クラスターのコピーを作成する場合

AWS Management Console を使用して DB クラスターのクローンを作成するには

1. AWS マネジメントコンソールにサインインして Amazon Neptune コンソール (https://console.aws.amazon.com/neptune/home) を開きます。
2. ナビゲーションペインで、[インスタンス] を選択します。クローンを作成する DB クラスターのプライマリインスタンスを選択します。
3. [Instance actions] を選択してから、[Create clone] を選択します。
4. [Create Clone (クローンの作成)] ページで、[DB instance identifier (DB インスタンス識別子)] としてクローン DB クラスターのプライマリインスタンスの名前を入力します。

   希望する場合には、クローン DB クラスターのそのほかの設定を行います。各種の DB クラスターの設定についての詳細は、「コンソールを使用して起動する」を参照してください。
5. [Create Clone] を選択してクローン DB クラスターを起動します。

AWS CLI を使用して DB クラスターのクローンを作成するには

- Neptune restore-db-cluster-to-point-in-time AWS CLI コマンドを呼び出し、以下の値を指定します。

- `--source-db-cluster-identifier` – クローンを作成するソース DB クラスターの名前。
- `--db-cluster-identifier` – クローン DB クラスターの名前。
- `--restore-type copy-on-write` – `copy-on-write` 値は、クローン DB クラスターを作成する必要があることを示します。
- `--use-latest-restorable-time` – これにより、復元可能な最新のバックアップ時間が使用されることが指定されます。

> **Note**
>
> restore-db-cluster-to-point-in-time AWS CLI コマンドは、その DB クラスターの DB インスタンスではなく、DB クラスターのみをクローン作成します。

次の Linux/UNIX の例では、`source-db-cluster-id` DB クラスターからクローンを作成し、クローンに `db-clone-cluster-id` という名前を付けます。

```
aws neptune restore-db-cluster-to-point-in-time \
  --region us-east-1 \
  --source-db-cluster-identifier source-db-cluster-id \
  --db-cluster-identifier db-clone-cluster-id \
  --restore-type copy-on-write \
  --use-latest-restorable-time
```

\ 行末のエスケープ文字が Windows の ^ と同等のものに置き換えられている場合、同じ例が Windows でも機能します。

```
aws neptune restore-db-cluster-to-point-in-time ^
  --region us-east-1 ^
  --source-db-cluster-identifier source-db-cluster-id ^
  --db-cluster-identifier db-clone-cluster-id ^
  --restore-type copy-on-write ^
  --use-latest-restorable-time
```

制限事項

Neptune での DB クローン作成には以下の制限があります。

- AWS リージョンをまたぐクローンデータベースは作成できません。クローンデータベースはソース データベースと同じリージョンで作成する必要があります。
- クローンされたデータベースは、クローンされたデータベースで使用されている Neptune エンジンバージョンの最新のパッチを常に使用します。これは、ソースデータベースがそのパッチバージョンにアップグレードされていない場合でも当てはまります。ただし、エンジンのバージョン自体は変更されません。
- 現在、他のクローンに基づくクローンを含め、Neptune DB クラスターの 1 つのコピーについて、15 クローンに制限されます。この制限に達したら、データベースを複製するのではなく、データベースのコピーをもう 1 つ作成する必要があります。ただし、新しいコピーから最大で 15 クローンまで作成できます。
- アカウント間での DB クローン作成は現在サポートされていません。
- クローンに異なる virtual private cloud (VPC) を提供できます。ただし、この VPCのサブネットは同じアベイラビリティゾーンにマッピングする必要があります。

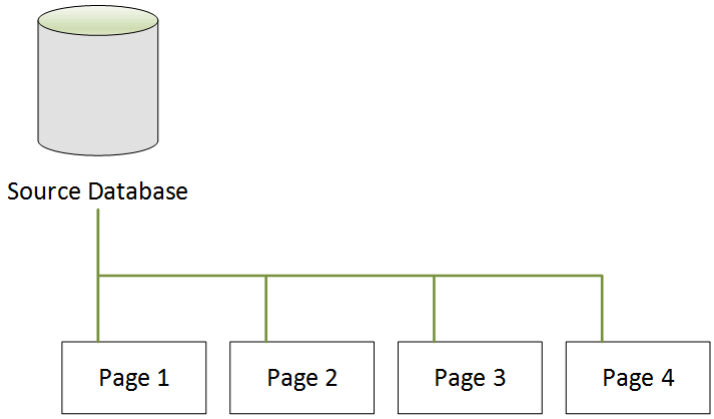
DB クローン作成のためのコピーオンライトプロトコル

以下の例で、コピーオンライトプロトコルの動作について説明します。

- クローン作成前の Neptune データベース
- クローン作成後の Neptune データベース
- ソースデータベースが変更されたとき
- クローンデータベースが変更されたとき

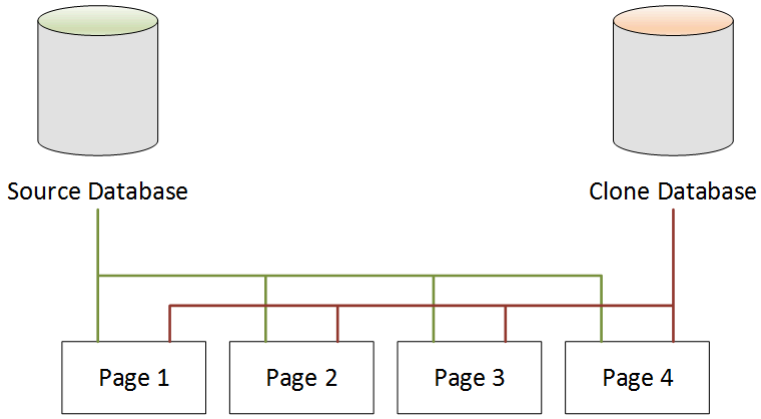
クローン作成前の Neptune データベース

ソースデータベースのデータはページに保存されています。次の図では、ソースデータベースに 4 つのページがあります。



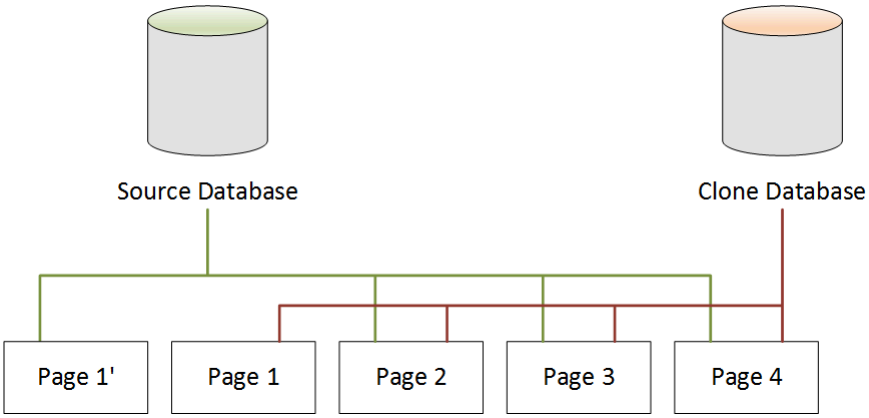
クローン作成後の Neptune データベース

次の図に示すように、DB クローン作成後、ソースデータベースに変更はありません。ソースデータベースとクローンデータベースの両方が同じ 4 つのページを指しています。どのページも物理的にコピーされていないため、追加のストレージは不要です。



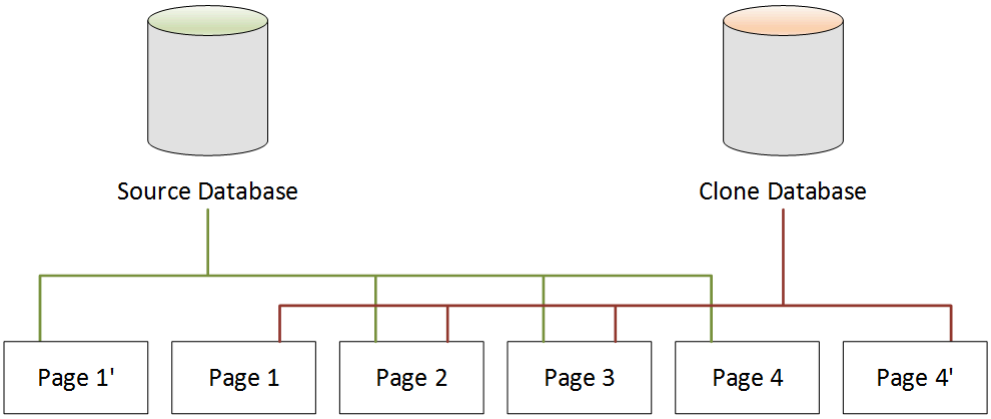
ソースデータベースが変更されたとき

次の例では、ソースデータベースの Page 1 のデータに変更があります。元の Page 1 に書き込む代わりに、追加のストレージを使用して Page 1' という新しいページが作成されます。ソースデータベースはこれで新しい Page 1'、および Page 2、Page 3 と Page 4 を指すようになります。クローンデータベースは、これまでと同じく Page 1 から Page 4 を指します。



クローンデータベースが変更されたとき

次の図では、クローンデータベースにも変更があります。今回の変更は Page 4 です。元の Page 4 に書き込む代わりに、追加のストレージを使用して Page 4' という新しいページが作成されます。ソースデータベースはこれまで同様、Page 1' および Page 2 から Page 4 を指しますが、クローンデータベースは Page 1 から Page 3 までと Page 4' を指すようになります。



2 番目の例で示すように、DB クローン作成時点では追加のストレージは不要です。ただし、例 3 と例 4 で示すように、ソースデータベースとクローンデータベースで変更があると、変更があったページだけが作成されます。時間が経過してソースデータベースとクローンデータベースの両方で追加の変更があると、その変更をキャプチャして保存するために増分のストレージが必要になります。

ソースデータベースの削除

ソースデータベースを削除しても、それに関連付けられているクローンデータベースには影響しません。クローンデータベースは、ソースデータベースが前に所有していたページを指し続けます。

Amazon Neptune インスタンスの管理

以下のセクションには、インスタンスレベルのオペレーションに関する情報を含みます。

トピック

Neptune T3 バースト可能インスタンスクラス

R5 や R6 などの固定パフォーマンスインスタンスクラスに加えて、Amazon Neptune にはバースト可能パフォーマンス T3 インスタンスを使用するオプションがあります。グラフアプリケーションを開発しているときは、速度と応答性の高いデータベースが必要ですが、常に使用する必要はありません。Neptune の db.t3.medium インスタンスクラスは、まさにそのような状況に適したものであり、最も安価な固定パフォーマンスインスタンスクラスよりも大幅にコストが低くなります。

バースト可能インスタンスは、ワークロードから必要とされるまで CPU パフォーマンスのベースラインレベルで実行され、ワークロードが必要としている間はベースラインをかなり上回ってバーストします。平均 CPU 使用率が 24 時間ベースラインを超えなければ、1 時間あたりの料金でバーストがカバーされます。ほとんどの開発およびテスト状況では、低コストで高いパフォーマンスが実現します。

T3 インスタンスクラスから始めた場合、本稼働の準備ができたら、AWS Management Console、AWS CLI、またはいずれかの AWS SDK を使用して、後で固定パフォーマンスインスタンスクラスに簡単に切り替えることができます。

T3 バーストは CPU クレジットによって管理される

CPU クレジットは、1 つの仮想 CPU コア (vCPU) の使用率が 1 分間 100% であることを表します。また、1 つの vCPU の使用率が 2 分間 50%、2 つの vCPU の使用率が 2 分間 25% などとも言い換えることができます。

T3 インスタンスは、アイドル状態のときに CPU クレジットを蓄積し、アクティブなときに消費します。どちらもミリ秒単位で測定されます。db.t3.medium インスタンスクラスには 2 つの vCPU があり、それぞれアイドル状態のときに 1 時間あたり 12 CPU クレジットを獲得します。つまり、各 vCPU の使用率が 20% の場合、CPU クレジットバランスがゼロになります。獲得した 12 CPU クレジットは、vCPU の使用率が 20% になると消費されます (60 分の 20% も 12 であるため)。したがって、この 20% の使用率は、CPU クレジット残高がプラスにもマイナスにもならないベースライン使用率です。

アイドル時間 (CPU 使用率が使用可能な合計量の 20% を下回る) が発生すると、CPU クレジットがクレジットバランスバケットに保存されます。 db.t3.medium インスタンスクラスの上限は 576 です (24 時間で累積できる CPU クレジットの最大数、つまり 2 x 12 x 24)。この制限を超えると、CPU クレジットはそのまま破棄されます。

CPU クレジットバランスがゼロを下回っても、必要に応じて、ワークロードが必要とする限り CPU 使用率は 100% までバーストできます。インスタンスのバランスが 24 時間継続してマイナスの場

合、その期間中 -60 CPU クレジット蓄積されるたびに 0.05 USD の追加料金が発生します。ただし、ほとんどの開発およびテストワークロードでは、バーストは通常、バースト前後のアイドル時間により力バーされます。

> **Note**
>
> Neptune の T3 インスタンスクラスは、Amazon EC2 unlimited モードと同じように設定されます。

AWS Management Console を使用した T3 バースト可能インスタンスの作成

AWS Management Console では、`db.t3.medium` インスタンスクラスを使用するプライマリ DB クラスターインスタンスまたはリードレプリカインスタンスを作成することも、`db.t3.medium` インスタンスクラスを使用するように既存のインスタンスを変更することもできます。

たとえば、Neptune コンソールで新しい DB クラスタープライマリインスタンスを作成するには、次のようにします。

- [Create Database] (データベースの作成) を選択します。
- DB エンジンバージョン `1.0.2.2` と同等以降を選びます。
- [目的] で、[開発とテスト] を選択します。
- [DB インスタンスクラス] としてデフォルト値 `db.t3.medium — 2 vCPU, 4 GiB RAM` を受け入れます。

AWS CLI を使用した T3 バースト可能インスタンスの作成

AWS CLI を使用して同じ作業を行うこともできます。

```
aws neptune create-db-cluster \
    --db-cluster-identifier (name for a new DB cluster) \
    --engine neptune \
    --engine-version "1.0.2.2"

aws neptune create-db-instance \
    --db-cluster-identifier (name of the new DB cluster) \
    --db-instance-identifier (name for the primary writer instance in the cluster) \
    --engine neptune \
    --db-instance-class db.t3.medium
```

Neptune DB インスタンスを変更する (その後すぐに適用する)

Amazon Neptune DB インスタンスへのほとんどの変更は、すぐに適用するか、または次回のメンテナンスの期間まで延期できます。一部の変更 (パラメータグループの変更など) を適用するには、手動による DB インスタンスの再起動が必要になる場合があります。

> **Important**
>
> 変更を適用するために Neptune が DB インスタンスを再起動する必要がある場合、変更が機能停止につながることがあります。DB インスタンスの設定を変更する前に、データベースとアプリケーションに対する影響を考慮してください。

一般的な設定とダウンタイムに関する意義

次の表には、変更ができる設定、変更が適用される時間、変更により DB インスタンスのダウンタイムが生じるかどうかに関する詳細が含まれます。

DB インスタンス設定	ダウンタイムに関する注意
DB インスタンスクラス	すぐに適用されるか、次のメンテナンスウィンドウ時に適用されるかにかかわらず、この変更時に機能停止が発生します。
DB インスタンス識別子	すぐに適用されるか、次のメンテナンスウィンドウ時に適用されるかにかかわらず、この変更時に DB インスタンスは再起動され、機能停止が発生します。
サブネットグループ	すぐに適用されるか、次のメンテナンスウィンドウ時に適用されるかにかかわらず、この変更時に DB インスタンス

DB インスタンス設定	ダウンタイムに関する注意	
	は再起動され、機能停止が発生します。	
セキュリティグループ	変更がいつ行われるかを指定しても、変更はできるだけ早く非同期的に適用され、停止は発生しません。	–
認証機関	デフォルトでは、新しい認証局を割り当てると, DB インスタンスは再起動されます。	
Database Port	変更は常に即時に行われるため、DB インスタンスが再起動され、機能停止が発生します。	

DB インスタンス設定	ダウンタイムに関する注意	
DB パラメータグループ	この設定を変更しても機能は停止しません。パラメータグループ名自体は即時に変更されますが、フェイルオーバーなしでインスタンスを再起動するまで実際のパラメータの変更は適用されません。この場合、DB インスタンスは自動的には再起動されず、次のメンテナンスウィンドウ時にパラメータの変更は適用されません。ただし、新しく関連付けられた DB パラメータグループの動的パラメータを変更すると、これらの変更は再起動せずに直ちに適用されます。 詳細については、「Amazon Neptune における DB インスタンスの再起動」を参照してください。	
DB クラスターのパラメータグループ	DB パラメータグループ名は直ちに変更されます。	

DB インスタンス設定	ダウンタイムに関する注意	
バックアップの保存期間	変更をすぐに実行するように指定した場合、この変更はすぐに実行されます。そうでない場合、設定を 0 以外の値から別の 0 以外の値に変更した場合、変更は可能な限り早く非同期的に適用されます。その他の変更は、次のメンテナンスウィンドウ時に行われます。0 から 0 以外の値、0 以外の値から 0 に変更した場合、機能停止が発生します。	
[監査ログ]	CloudWatch ログを通じて監査ログを使用する場合は、[監査ログ] を選択します。監査ログを有効にするには、DB クラスターパラメータグループの `neptune_enable_audit_log` パラメータを `enable` (1) に設定する必要もあります。	

DB インスタンス設定	ダウンタイムに関する注意	
マイナーバージョン自動アップグレード	Neptune DB クラスターに DB エンジンのマイナーバージョンアップグレードがリリースと同時に自動的に適用されるようにする場合は、[Enable auto minor version upgrade] (マイナーバージョン自動アップグレードの有効化) を選択します。 [Auto minor version upgrade] (自動マイナーバージョンアップグレード) オプションは、Amazon Neptune DB クラスターのマイナーエンジンバージョンに対するアップグレードにのみ適用されます。システム安定性を維持するために適用される定期的なパッチは適用されません。	

Neptune DB インスタンスの名前変更

AWS Management Console を使用して Amazon Neptune DB インスタンスの名前を変更します。DB インスタンスの名前を変更すると、広範囲に影響が及びます。DB インスタンスの名前を変更する前に知っておくべき事柄の一覧を次に示します。

- DB インスタンスの名前を変更すると、DB インスタンスに割り当てた名前が URL に含まれているため、DB インスタンスのエンドポイントが変更されます。必ず古い URL のトラフィックを新しい URL にリダイレクトする必要があります。
- DB インスタンスの名前を変更すると、DB インスタンスで使用されていた古い DNS 名はすぐに削除されますが、数分間キャッシュされたままになる場合があります。名前を変更した DB インスタンスの新しい DNS 名は、約 10 分後に有効になります。名前を変更した DB インスタンスは、新しい名前が有効になるまで使用できません。
- インスタンスの名前を変更する際に、既存の DB インスタンス名を使用することはできません。
- DB インスタンスに関連付けられたすべてのリードレプリカは、名前を変更した後もそのインスタンスに関連付けられたままになります。たとえば、本番データベースを提供する DB インスタンスがあり、そのインスタンスに複数の関連するリードレプリカがあるとします。DB インスタンスの名前を変更して本番環境で DB スナップショットに置き換えた場合、名前を変更した DB インスタンスにはリードレプリカが関連付けられたままになります。
- DB インスタンスの名前に関連付けられたメトリクスとイベントは、DB インスタンス名を再利用する場合も維持されます。例えば、リードレプリカを昇格させ、前のプライマリインスタンスの名前となるように名前を変更した場合、プライマリインスタンスに関連付けられたイベントとメトリクスは、名前が変更されたインスタンスに関連付けられます。
- DB インスタンスのタグは、名前を変更するかどうかにかかわらず DB インスタンスに残ります。
- 名前を変更した DB インスタンスの DB スナップショットは保持されます。

Neptune コンソールを使用して DB インスタンスの名前を変更するには

1. AWS マネジメントコンソールにサインインして Amazon Neptune コンソール (https://console.aws.amazon.com/neptune/home) を開きます。
2. ナビゲーションペインで、[Databases] (データベース) を選択します。
3. 名前を変更する DB インスタンスの横にあるラジオボタンを選択します。
4. [Instance actions] メニューで [Modify] を選択します。
5. [DB instance identifier] テキストボックスに新しい名前を入力します。[Apply immediately] を選択し、次に [Continue] を選択します。

6. [Modify DB instance] を選択して変更を完了します。

Amazon Neptune における DB インスタンスの再起動

場合によっては、Amazon Neptune DB インスタンスを変更したり、インスタンスに関連付けられている DB パラメータグループを変更したり、インスタンスによって使用されるパラメータグループの静的データベースパラメータを変更したりすると、その変更を適用するためにインスタンスを再起動する必要があります。

DB インスタンスを再起動すると、データベースエンジンサービスが再起動されます。再起動により、関連付けられている DB パラメータグループに対して保留中になっていた変更も DB インスタンスに適用されます。DB インスタンスを再起動すると、そのインスタンスは一時的に機能停止になります。その間、DB インスタンスのステータスは [rebooting] に設定されます。Neptune インスタンスがマルチ AZ 用に構成されている場合、再起動はフェイルオーバーにより実行される可能性があります。再起動が完了すると、Neptune イベントが作成されます。

DB インスタンスがマルチ AZ 配置の場合は、[Reboot (再起動)] オプションを選択すると、1 つのアベイラビリティーゾーンから別のアベイラビリティーゾーンへのフェイルオーバーを強制的に実行できます。DB インスタンスのフェイルオーバーを強制すると、Neptune は別のアベイラビリティーゾーンでスタンバイレプリカに自動的に切り替わります。次に、DB インスタンスの DNS レコードが更新され、スタンバイ DB インスタンスを指します。その結果、DB インスタンスへの既存の接続のクリーンアップと再確立が必要になります。

[Reboot with failover] (フェイルオーバーによる再起動) が便利なのは、テスト用の DB インスタンスで障害をシミュレートするときや、フェイルオーバーの実行後にオペレーションを元のアベイラビリティーゾーンに復元するときです。詳細については、Amazon RDS ユーザーガイドの高可用性 (マルチ AZ) を参照してください。DB クラスターを再起動すると、そのスタンバイレプリカにフェイルオーバーします。Neptune レプリカを再起動しても、フェイルオーバーは開始されません。

再起動にかかる時間は、クラッシュ回復プロセスによって異なります。再起動時間を短くするには、再起動プロセス中のデータベースアクティビティをできる限り減らして、未完了のトランザクションのロールバックアクティビティが少なくなるようにすることをお勧めします。

コンソールでは、DB インスタンスが [Available] (利用可能) 状態ではない場合、[Reboot] (再起動) オプションが無効になる場合があります。その原因としては、進行中のバックアップ、お客様の要求による変更、メンテナンスウィンドウのアクションなどが考えられます。

> **Note**
>
> リリース: 1.2.0.0 (2022-07-21) より前は、DB クラスター内のすべてのリードレプリカインスタンスは、プライマリ (ライター) インスタンスが再起動するたびに自動的に再起動されていました。
> リリース: 1.2.0.0 (2022-07-21) 以降では、プライマリインスタンスを再起動しても、レプリカは再起動しません。つまり、クラスターパラメータを変更する場合、パラメータの変更を反映するには、各インスタンスを個別に再起動する必要があります (「パラメータグループ」を参照)。

Neptune コンソールを使用して DB インスタンスを再起動するには

1. AWS マネジメントコンソールにサインインして Amazon Neptune コンソール (https://console.aws.amazon.com/neptune/home) を開きます。
2. ナビゲーションペインで、[Databases] (データベース) を選択します。
3. 再起動する DB インスタンスを選択します。
4. [Instance actions]、[Reboot] の順に選択します。
5. アベイラビリティーゾーン間でフェイルオーバーを強制的に実行するには、DB インスタンスの再起動ダイアログボックスからフェイルオーバーによる再起動を選択します。
6. [Reboot] を選択します。逆に、再起動をキャンセルするには、[Cancel] (キャンセル) を選択します。

Amazon Neptune の DB インスタンスを削除する

インスタンスでインスタンスに対する削除保護が無効になっている限り、どの状態の Amazon Neptune DB インスタンスでも随時削除できます。

削除保護が有効になっている場合、DB インスタンスを削除できません

削除保護が有効になっていない DB インスタンスのみ削除できます。ただし、コンソール、AWS CLI、または API を使用して DB インスタンスを削除する場合、Neptune で削除保護が強制的に適用されます。

AWS Management Console を使用して本稼働 DB クラスターを作成する場合は、削除保護はデフォルトで有効です。

AWS CLI または API コマンドを使用して DB インスタンスを作成すると、削除保護はデフォルトで無効になります。

削除保護が有効になっている DB インスタンスを削除するには、まず `DeletionProtection` フィールドを `false` に設定するようにインスタンスを変更します。

削除保護を有効または無効にしても、インスタンスが停止することはありません。

DB インスタンスを削除する前に DB インスタンスの最終スナップショットを作成する

DB インスタンスを削除するには、インスタンスの名前と、そのインスタンスの最終 DB スナップショットを作成するかどうかを指定する必要があります。削除する DB インスタンスのステータスが Creating (作成中) の場合、最終 DB スナップショットを作成することはできません。DB インスタンスが failed (失敗)、incompatible-restore (互換性のない復元)、または incompatible-network (互換性のないネットワーク) というステータスのエラー状態にある場合は、`SkipFinalSnapshot` パラメータを `true` に設定しているときにのみ、インスタンスを削除できます。

DB クラスターで、AWS Management Console を使用して、DB クラスター内のすべての Neptune DB インスタンスを削除した場合、DB クラスターが自動的に削除されます。AWS CLI または SDK を使用している場合は、最後のインスタンスを削除した後、DB クラスターを手動で削除する必要があります。

> **Important**
>
> DB クラスター全体を削除すると、自動バックアップはすべて同時に削除され、復旧できません。つまり、最終的な DB スナップショットを手動で作成しない限り、後で DB インスタ

ンスを最終状態に復元することはできません。インスタンスの手動スナップショットは、クラスターを削除したときに削除されません。

削除する DB インスタンスにリードレプリカがある場合は、そのリードレプリカを昇格させるか、削除する必要があります。

以下の例では、DB インスタンスを削除する際に、最終的な DB スナップショットを作成する場合と作成しない場合の両方を示しています。

最終スナップショットを作成しない DB インスタンスの削除

DB インスタンスを迅速に削除する必要がある場合は、最終 DB スナップショットの作成をスキップできます。DB インスタンスを削除すると、自動バックアップはすべて削除され、復旧できません。手動スナップショットは削除されません。

Neptune コンソールを使用して最終的な DB スナップショットを作成せずに DB インスタンスを削除するには

1. AWS マネジメントコンソールにサインインして Amazon Neptune コンソール (https://console.aws.amazon.com/neptune/home) を開きます。
2. ナビゲーションペインで、[Databases] (データベース) を選択します。
3. [Instances] (インスタンス) リストで、削除する DB インスタンスの横にあるラジオボタンを選択します。
4. [Instance actions] を選択し、[Delete] を選択します。
5. [Create final snapshot?] (最終スナップショットを作成しますか) ボックスで [No] (いいえ) を選択します。
6. [Delete] (削除) をクリックします。

最終スナップショットを作成する DB インスタンスの削除

削除した DB インスタンスを後で復元可能にする必要がある場合、最終 DB スナップショットを作成できます。自動バックアップもすべて削除され、復旧できません。手動スナップショットは削除されません。

Neptune コンソールを使用して最終的な DB スナップショットを作成せずに DB インスタンスを削除するには

1. AWS マネジメントコンソールにサインインして Amazon Neptune コンソール (https://console.aws.amazon.com/neptune/home) を開きます。
2. ナビゲーションペインで、[Databases] (データベース) を選択します。
3. [Instances] (インスタンス) リストで、削除する DB インスタンスの横にあるラジオボタンを選択します。
4. [Instance actions] を選択し、[Delete] を選択します。
5. [Create final snapshot?] (最終スナップショットを作成しますか) ボックスで [Yes] (はい) を選択します。
6. [Final snapshot name] (最終スナップショット名) ボックスに、最終 DB スナップショットの名前を入力します。
7. [Delete] (削除) をクリックします。

instance-status API を使用して、インスタンスの状態の確認、インスタンスの種類の特定、現在インストールされているエンジンリリースバージョンの確認、インスタンスに関するその他の情報の取得を行うことができます。

Amazon Neptune サーバーレス

Amazon Neptune サーバーレスは、非常に大きな処理需要の増加も致すために、必要に応じて DB クラスターをスケールアップし、需要が減少すると再びスケールダウンするオンデマンド自動スケーリング設定です。Neptune データベースのワークロードを監視し、容量を調整するプロセスを自動化するのに役立ちます。容量はアプリケーションの需要に基づいて自動的に調整されるため、課金されるのは、アプリケーションが実際に必要とするリソースに対してのみです。

Neptune サーバーレスのユースケース

Neptune サーバーレスは、多くのタイプのワークロードをサポートしています。要求が厳しく変動の激しいワークロードに適しており、データベースの使用負荷が短時間の間だけ増大し、その後に軽いアクティビティが長時間続くか、またはアクティビティがまったく発生しなくなるケースに非常に役立ちます。Neptune サーバーレスは、以下のユースケースに特に役立ちます。

- 変動するワークロード - 突然で予測不可能な CPU アクティビティの増加が発生するワークロード。Neptune サーバーレスでは、グラフデータベースはワークロードのニーズに合わせて自動的にスケーリングされ、アクティビティのサージが終了した時点でスケールダウンして元に戻ります。ピーク容量や平均容量に合わせてプロビジョニングする必要はなくなります。ピークワークロードに対応するために容量の上限を指定でき、その容量は必要な場合以外使用されません。

  Neptune サーバーレスのスケーリングの詳細度は、ワークロードのニーズに合わせて容量を細かく調整しやすくします。Neptune サーバーレスでは、必要に応じてきめ細かい単位で容量を追加または削除できます。ほんの少しの容量が必要なときには、Neptune 容量ユニット (NCU) の半分だけ追加できます。
- マルチテナントアプリケーション — Neptune サーバーレスを活用することで、テナントクラスターを個別に管理しなくても、実行する必要のあるアプリケーションごとに個別の DB クラスターを作成できます。各テナントクラスターは、複数の要因によってビジー期間とアイドル期間が異なる場合がありますが、Neptune サーバーレスでは、ユーザーが操作しなくても効率的にスケーリングできます。
- 新しいアプリケーション — 新しいアプリケーションをデプロイする際、そのアプリケーションに必要なデータベース容量がわからないことがよくあります。Neptune サーバーレスを使用すると、新しいアプリケーションの容量要件に合わせて自動的にスケーリングできる DB クラスターを設定できます。
- 容量計画 – 通常、クラスター内のすべての DB インスタンスの DB インスタンスクラスを変更することによって、データベース容量を調整するか、ワークロードに最適なデータベース容量を確認す

るとします。Neptune サーバーレスでは、この管理オーバーヘッドを回避できます。代わりに、新しい DB クラスターやインスタンスを作成しなくても、既存の DB インスタンスをプロビジョニング済みからサーバーレスに、またはサーバーレスからプロビジョニング済みに変更できます。

- 開発とテスト — Neptune サーバーレスは、開発環境やテスト環境にも最適です。Neptune サーバーレスを使用すると、最も要求の厳しいアプリケーションをテストするのに十分な最大容量の DB インスタンスを作成でき、テストの合間にシステムがアイドル状態になる可能性があるその他の時間のために最小容量を低く設定した DB インスタンスを作成できます。

Neptune サーバーレスはコンピューティング容量をスケーリングするだけです。ストレージ容量は変わらず、サーバーレススケーリングの影響を受けません。

> **Note**
>
> また、Neptune サーバーレスで Neptune 自動スケーリングを使用して、さまざまな種類のワークロードの変動に対応することもできます。

Amazon Neptune サーバーレスの制約

- すべてのリージョンで利用できるわけではない - Neptune サーバーレスは、次のリージョンでのみ利用できます。
  - 米国東部 (バージニア北部): `us-east-1`
  - 米国東部 (オハイオ): `us-east-2`
  - 米国西部 (北カリフォルニア): `us-west-1`
  - 米国西部 (オレゴン): `us-west-2`
  - カナダ (中部): `ca-central-1`
  - 欧州 (ストックホルム): `eu-north-1`
  - 欧州 (アイルランド): `eu-west-1`
  - 欧州 (ロンドン): `eu-west-2`
  - 欧州 (フランクフルト): `eu-central-1`
  - アジアパシフィック (東京): `ap-northeast-1`
  - アジアパシフィック (シンガポール): `ap-southeast-1`
  - アジアパシフィック (シドニー): `ap-southeast-2`

- 初期のエンジンバージョンでは使用不可 — Neptune サーバーレスはエンジンリリース 1.2.0.1 以降でのみ利用可能です。
- Neptune ルックアップキャッシュとの互換性がない — ルックアップキャッシュはサーバーレス DB インスタンスでは機能しません。
- サーバーレスインスタンスの最大メモリは 256 GB — `MaxCapacity` を 128 NCU (サポートされている最大設定) に設定すると、Neptune サーバーレスインスタンスは 256 GB のメモリまでスケーリングでき、これは `R6g.8XL` プロビジョニングされたインスタンスタイプと同等です。

Neptune サーバーレス DB クラスターの容量スケーリング

Neptune サーバーレス DB クラスターのセットアップは、通常のプロビジョニングクラスターの設定と似ていますが、スケーリングの最小単位と最大単位を追加設定し、インスタンスタイプを `db.serverless` に設定します。スケーリング構成は Neptune 容量ユニット (NCU) で定義され、それぞれ 2 GiB (ギビバイト) のメモリ (RAM) と、関連する仮想プロセッサ容量 (vCPU) とネットワークで構成されています。これは `ServerlessV2ScalingConfiguration` オブジェクトの一部として設定され、次のように JSON で表されます。

```
"ServerlessV2ScalingConfiguration": {
  "MinCapacity": (minimum NCUs, a floating-point number such as   1.0),
  "MaxCapacity": (maximum NCUs, a floating-point number such as 128.0)
}
```

特定の時点で、各 Neptune ライターまたはリーダーインスタンスには、そのインスタンスによって現在使用されている NCU の数を表す浮動小数点数で測定される容量があります。インスタンスレベルで CloudWatch ServerlessDatabaseキャパシティメトリクスを使用して、特定の DB インスタンスが現在使用している NCUsの数を調べることができます。また、NCUUtilization メトリクスを使用して、インスタンスが使用している最大キャパシティの割合を調べることができます。これらのメトリクスは両方とも DB クラスターレベルでも使用でき、DB クラスター全体の平均リソース使用率を示します。

Neptune サーバーレス DB クラスターを作成するときは、すべてのサーバーレスインスタンスの Neptune 容量ユニット (NCU) の最小数と最大数の両方を設定します。

指定する最小 NCU 値は、DB クラスター内のサーバーレスインスタンスを縮小できる最小サイズを設定します。同様に、最大 NCU 値は、サーバーレスインスタンスを拡張できる最大サイズを設定します。設定できる最高の最大 NCU 値は 128.0 NCU であり、最低の最小値は 1.0 NCU です。

Neptune は、CPU、メモリ、ネットワークなどのリソースの使用状況を監視することによって、各 Neptune サーバーレスインスタンスの負荷を継続的に追跡します。負荷は、アプリケーションのデータベース操作、サーバーのバックグラウンド処理、その他の管理タスクによって発生します。

サーバーレスインスタンスの負荷が現在の容量の制限に達するか、Neptune が他のパフォーマンス問題を検出すると、インスタンスは自動的にスケールアップします。インスタンスの負荷が低下すると、設定された最小容量単位に向かって容量がスケールダウンし、CPU 容量がメモリより先に解放されます。このアーキテクチャでは、制御されたステップダウン方式でリソースを解放でき、需要の変動に効果的に対処できます。

リーダーインスタンスはライターインスタンスと一緒にスケーリングすることも、プロモーション層を設定して個別にスケーリングすることもできます。プロモーション層 0 と 1 のリーダーインスタンスは、ライターと同時にスケーリングされるため、フェイルオーバー時にライターからワークロードを迅速に引き継ぐために、適切な容量に合わせてサイズ調整されます。プロモーション層 2 ~ 15 のリーダーは、ライターインスタンスとは無関係に、また互いに独立してスケーリングします。

高可用性を確保するために Neptune DB クラスターをマルチ AZ クラスターとして作成した場合、Neptune サーバーレスはデータベースの負荷に応じて、すべての AZ のインスタンスをスケールアップおよびスケールダウンします。セカンダリ AZ のリーダーインスタンスのプロモーション層を 0 または 1 に設定すると、プライマリ AZ のライターインスタンスの容量に合わせてスケールアップまたはスケールダウンできるため、いつでも現在のワークロードを引き継ぐことができます。

> **Note**
>
> Neptune DB クラスターのストレージは、クラスターをマルチ AZ クラスターとして作成したかどうかにかかわらず、3 つの AZ に分散したすべてのデータの 6 つのコピーで構成されます。ストレージレプリケーションはストレージサブシステムによって処理され、Neptune サーバーレスの影響を受けません。

Neptune サーバーレス DB クラスターの最小容量値の選択

最小容量に設定できる最小値は `1.0` NCU です。

最小値は、アプリケーションが効率的に動作するために必要な値よりも低く設定しないようにしてください。この値を低く設定しすぎると、メモリ集約的な特定のワークロードでタイムアウト率が高くなる可能性があります。

最小値をできるだけ低く設定すると、需要が少ないときにはクラスターが最小限のリソースしか使用しないため、コストを節約できます。ただし、ワークロードが非常に低いものから非常に高いものへ

と大きく変動する傾向がある場合は、最小値を高く設定することをお勧めします。最小値を高くすると、Neptune サーバーレスインスタンスのスケールアップが速くなるためです。

これは、Neptune が現在の容量に基づいてスケーリング単位を選択するためです。現在の容量が少ない場合、Neptune は最初はゆっくりとスケールアップします。最小値が高い場合、Neptune はより大きなスケーリングインクリメントから開始するため、ワークロードの急激な増加にもより速くスケールアップできます。

Neptune サーバーレス DB クラスターの最小容量値の選択

最大容量に設定できる最大値は `128.0` NCU であり、最大容量に設定できる最小値は `2.5` NCU です。設定する最大容量値は、設定した最小容量値より大きくなければなりません。

原則として、最大値は、アプリケーションで発生する可能性のあるピーク負荷に対応できる十分な大きさに設定してください。この値を低く設定しすぎると、メモリ集約的な特定のワークロードでタイムアウト率が高くなる可能性があります。

最大値をできるだけ高く設定すると、予想外のワークロードでもアプリケーションが処理できる可能性が高くなるという利点があります。デメリットは、リソースコストを予測して制御する能力をある程度失うことです。需要が予想外に急増すると、予算の予想をはるかに超えるコストがかかる可能性があります。

最大値を注意深く設定することの利点は、ピーク需要に対応できると同時に、Neptune のコンピューティングコストに上限を設けることができることです。

> **Note**
>
> Neptune サーバーレス DB クラスターの容量範囲を変更すると、一部の設定パラメータのデフォルト値が変更されます。Neptune は、一部の新しいデフォルトを直ちに適用できますが、一部の動的パラメータの変更は、再起動後に有効になります。`pending-reboot` ステータスは、一部のパラメータの変更を適用するために再起動が必要であることを示しています。

既存の設定を使用してサーバーレス要件を見積もる

通常、例外的に高いか低いワークロードを満たすために、プロビジョニングした DB インスタンスの DB インスタンスクラスを変更する場合、その経験を活かして、同等の Neptune サーバーレス容量範囲を概算で見積もることができます。

最適な最小容量設定を見積もる

既存の Neptune DB クラスターについてわかっていることを応用して、最適なサーバーレスの最小容量設定を見積もることができます。

例えば、プロビジョン済みワークロードのメモリ要件が、T3 や T4g などの小さな DB インスタンスクラスに対して大きすぎる場合、R5 または R6g DB インスタンスに相当するメモリを提供する最小 ACU 設定を選択します。

または、クラスターのワークロードが低い場合、db.r6g.xlarge DB インスタンスクラスを使用するとします。この DB インスタンスクラスには 32 GiB のメモリがあるため、NCU の最小設定を 16 に指定すると、ほぼ同じ容量にスケールダウンできるサーバーレスインスタンスを作成できます (各 NCU は、約 2 GiB のメモリに対応します)。db.r6g.xlarge インスタンスの使用率が低い場合は、より小さい値を指定できる可能性があります。

DB インスタンスが一定量のデータをメモリやバッファキャッシュに保持できるときにアプリケーションが最も効率的に動作する場合は、そのために十分なメモリを提供できる最小の NCU 設定を指定することを検討してください。そうしないと、サーバーレスインスタンスがスケールダウンしたときにバッファーキャッシュからデータが削除され、インスタンスがスケールアップしたときに時間をかけてバッファーキャッシュに読み戻さなければならなくなる可能性があります。データをバッファキャッシュに戻すための I/O 量が大きい場合は、最小 NCU 値を大きくする方が効果的な場合があります。

サーバーレスインスタンスがほとんどの時間特定の容量で実行されていることがわかった場合は、最小容量をそれより少しだけ小さく設定するとよいでしょう。Neptune サーバーレスは、現在の容量が必要容量より極端に小さくない場合、スケールアップする規模と速度を最も効果的に見積もることができます。

プロビジョン済みライターと Neptune サーバーレスリーダーの混合設定では、リーダーはライターと一緒にスケーリングしません。これらは個別にスケーリングするため、最小容量を小さく設定すると、レプリケーションの遅延が大きくなる場合があります。書き込み集約的なワークロードがある場合、ライターが行う変更に対応できるだけの十分な容量がない可能性があります。このような場合は、ライター容量と同等の最小容量を設定してください。特に、プロモーション層 2 ~ 15 のリーダーでレプリカのラグが発生した場合は、クラスターの最小容量設定を増やしてください。

最適な最大容量設定を見積もる

既存の Neptune DB クラスターについてわかっていることを応用して、最適なサーバーレス最大容量設定を見積もることができます。

例えば、クラスターのワークロードが高い場合に db.r6g.4xlarge DB インスタンスクラスを使用するとします。その DB インスタンスクラスには 128 GiB のメモリがあるため、最大 NCU 設定を 64 に指定して、同等の Neptune サーバーレスインスタンスを設定できます (各 NCU は約 2 GiB のメモリに対応)。db.r6g.4xlarge インスタンスが常にワークロードを処理できるとは限らない場合には、DB インスタンスをさらにスケールアップさせるため、より高い値を指定することができます。

ワークロードの予期しない急増がまれな場合は、その急増時でもアプリケーションのパフォーマンスを維持できるよう、最大容量を十分に高く設定するとよいでしょう。一方、異常な急上昇時にスループットを低下させることができるように、最大容量を低く設定したい場合もありますが、これにより Neptune は予想されるワークロードを問題なく処理でき、コストも抑えられます。

Neptune サーバーレス DB クラスターとインスタンスの追加設定

Neptune サーバーレス DB クラスターの最小容量と最大容量の設定に加えて、考慮すべき設定上の選択肢がいくつかあります。

DB クラスター内のサーバーレスインスタンスとプロビジョンドインスタンスを組み合わせる

DB クラスターはサーバーレス専用である必要はありません。サーバーレスインスタンスとプロビジョニングされたインスタンスの組み合わせ (混合構成) を作成できます。

例えば、サーバーレスインスタンスで利用可能な容量よりも多くの書き込み容量が必要だとします。この場合、非常に大きいプロビジョン済みライターを持つクラスターをセットアップしても、リーダーにはサーバーレスインスタンスを使用できます。

または、クラスターの書き込みワークロードは変化するが、読み取りワークロードは安定しているとします。この場合、サーバーレスライターと 1 つまたは複数のプロビジョン済みリーダーを持つクラスターをセットアップできます。

混合構成 DB クラスターの作成方法については、「Amazon Neptune のサーバーレスの使用」を参照してください。

Neptune サーバーレスインスタンスのプロモーション層の設定

複数のサーバーレスインスタンスを含むクラスター、またはプロビジョン済みインスタンスとサーバーレスインスタンスの混在するクラスターでは、各サーバーレスインスタンスのプロモーション層

の設定に注意してください。この設定は、プロビジョン済み DB インスタンスよりも、サーバーレスインスタンスの多くの動作を制御します。

では AWS Management Console、データベースの作成、インスタンスの変更、リーダーの追加ページの「追加設定」の「フェイルオーバー優先度」を使用してこの設定を指定します。既存のインスタンスのこのプロパティは、[データベース] ページのオプションの [優先階層] 列に表示されます。このプロパティは、DB クラスターまたはインスタンスの詳細ページにも表示されます。

プロビジョン済みインスタンスの場合、0 ~ 15 層の選択肢は、フェイルオーバー操作時に Neptune がどのリーダーインスタンスをライターに昇格させるかを選択する順序のみを決定します。Neptune サーバーレスリーダーインスタンスの場合、層番号によって、インスタンスがライターインスタンスの容量に合わせてスケールアップするのか、それとも独自のワークロードのみに基づいて個別にスケールアップするのかが決まります。

層 0 または 1 の Neptune サーバーレスリーダーインスタンスは、フェイルオーバーが発生した場合にライターから引き継ぐことができるように、少なくともライターインスタンスと同じ大きさの最小容量に保たれます。ライターがプロビジョン済みインスタンスの場合、Neptune は同等のサーバーレス容量を見積もり、その推定値をサーバーレスリーダーインスタンスの最小容量として使用します。

層 2 ~ 15 の Neptune サーバーレスリーダーインスタンスには、最小容量に関する同じ制約はなく、ライターとは独立してスケーリングできます。アイドル状態の場合、クラスターの容量範囲で指定された 最小 NCU 値までスケールダウンします。ただし、読み取りワークロードが急激に増加すると、問題が発生する可能性があります。

リーダー容量をライター容量と同じに保つ

覚えておくべき重要なことの 1 つは、過度のレプリケーションラグを防ぐには、リーダーインスタンスがライターインスタンスに遅れないようにする必要があるということです。これは、サーバーレスのリーダーインスタンスがライターインスタンスと同期して自動的にスケーリングされない次の 2 つの状況で特に懸念されます。

- ライターがプロビジョニングされていて、リーダーがサーバーレスの場合。
- ライターがサーバーレスであり、サーバーレスリーダーがプロモーション層 2 ~ 15 にある場合。

いずれの場合も、リーダー操作がタイムアウトして再起動を引き起こす可能性がないように、サーバーレスの最小容量を予想されるライター容量と一致するように設定してください。プロビジョニングされたライターインスタンスの場合は、プロビジョニングされたインスタンスの最小容量と一致す

るように最小容量を設定します。サーバーレスライターの場合、最適な設定を予測するのは難しいかもしれません。

インスタンス容量の範囲はクラスターレベルで設定されるため、すべてのサーバーレスインスタンスは同じ最小容量設定と最大容量設定によって制御されます。層 0 と 1 のリーダーインスタンスはライターインスタンスと同期してスケーリングされますが、プロモーション層 2 ~ 15 のインスタンスは、ワークロードに応じて互いに、またライターインスタンスとは独立してスケーリングされます。最小容量を低く設定しすぎると、層 2 ~ 15 のアイドル状態のインスタンスは、ライターアクティビティの急増に対応できるほど速くスケールダウンしてスケールアップできない可能性があります。

高すぎるタイムアウト値の設定を避ける

サーバーレスインスタンスでクエリのタイムアウト値を高く設定しすぎると、予期しないコストが発生する可能性があります。

妥当なタイムアウト設定がないと、強力で高価なインスタンスタイプを必要とするクエリをうっかり発行してしまい、長時間実行し続け、予想もしなかったコストが発生する可能性があります。ほとんどのクエリに対応し、異常に長い実行でもタイムアウトが発生するだけのクエリタイムアウト値を使用することで、このような予期しない出費を回避できます。

これは、パラメータを使用して設定された一般的なクエリタイムアウト値と、クエリヒントを使用して設定されたクエリごとのタイムアウト値の両方に当てはまります。

Neptune サーバーレス構成の最適化

Neptune サーバーレス DB クラスターが実行中のワークロードに合わせて調整されていない場合、最適に動作しないことに気付くかもしれません。メモリの問題に遭遇することなくスケーリングできるように、最小または最大容量設定を調整できます。

- クラスターの最小容量設定を増加します。これにより、アイドル状態のインスタンスが、アプリケーションや有効になっている機能が必要とするよりも少ないメモリ容量にスケールバックする状況を修正できます。
- クラスターの最大容量設定を増加します。これにより、ワークロードや有効になっているメモリ集約的機能を処理するのに十分なメモリがある容量まで、びいーなデータベースがスケールアップできない状況を修正できます。
- 問題のインスタンスのワークロードを変更します。例えば、クラスターにリーダーインスタンスを追加して、読み取り負荷をより多くのインスタンスに分散できます。
- アプリケーションのクエリを調整して、使用するリソースを減らします。

- Neptune サーバーレス内で使用可能な最大 NCU よりも大きいプロビジョニングされたインスタンスを使用みて、ワークロードのメモリと CPU の要件により適しているかどうかを確認してください。

Amazon Neptune のサーバーレスの使用

新しい Neptune DB クラスターをサーバーレスクラスターとして作成することも、場合によっては既存の DB クラスターをサーバーレスを使用するように変換することもできます。サーバーレス DB クラスター内の DB インスタンスをサーバーレスインスタンスに変換したり、サーバーレスインスタンスから変換したりすることもできます。Neptune Serverless は、サポートされている の 1 つでのみ使用できますが、他のいくつかの制限があります (AWS リージョン 「」を参照Amazon Neptune サーバーレスの制約) 。

Neptune AWS CloudFormation スタックを使用して Neptune サーバーレス DB クラスターを作成することもできます。

サーバーレスを使用する新しい DB クラスターの作成

サーバーレスを使用する Neptune DB クラスターを作成するには、AWS Management Consoleを使用して、プロビジョニングされたクラスターを作成するのと同じ方法で作成できます。違いは、DB インスタンスサイズ未満では、DB インスタンスクラスをサーバーレスに設定する必要がある点です。その場合は、クラスターのサーバーレス容量範囲を設定する必要があります。

次のようなコマンド AWS CLI で を使用してサーバーレス DB クラスターを作成することもできます (Windows では、「\」を「^」に置き換えます) 。

```
aws neptune create-db-cluster \
  --region (an AWS ##### region that supports serverless) \
  --db-cluster-identifier (ID for the new serverless DB cluster) \
  --engine neptune \
  --engine-version (optional: 1.2.0.1 or above) \
  --serverless-v2-scaling-configuration "MinCapacity=1.0, MaxCapacity=128.0"
```

次のように serverless-v2-scaling-configuration パラメータを指定することもできます。

```
  --serverless-v2-scaling-configuration '{"MinCapacity":1.0, "MaxCapacity":128.0}'
```

その後、ServerlessV2ScalingConfiguration 属性に対して describe-db-clusters コマンドを実行すると、指定した容量範囲設定が返されます。

```
"ServerlessV2ScalingConfiguration": {
    "MinCapacity": (the specified minimum number of NCUs),
    "MaxCapacity": (the specified maximum number of NCUs)
}
```

既存の DB クラスターまたはインスタンスをサーバーレスに変換する

エンジンバージョン 1.2.0.1 以降を使用している Neptune DB クラスターがある場合は、サーバーレスに変換できます。このプロセスでは、ある程度のダウンタイムが発生します。

最初のステップは、既存のクラスターに容量範囲を追加することです。これを行うには AWS Management Console、 を使用するか、次のような AWS CLI コマンドを使用します (Windows では、「\」を「^」に置き換えます)。

```
aws neptune modify-db-cluster \
  --db-cluster-identifier (your DB cluster ID) \
  --serverless-v2-scaling-configuration \
      MinCapacity=(minimum number of NCUs, such as  2.0), \
      MaxCapacity=(maximum number of NCUs, such as 24.0)
```

次のステップは、クラスター内の既存のプライマリインスタンス (ライター) を置き換える新しいサーバーレス DB インスタンスを作成することです。ここでも、 AWS Management Console または を使用して、これ以降のすべてのステップを実行できます AWS CLI。どちらの場合も、DB インスタンスクラスをサーバーレスとして指定します。 AWS CLI コマンドは次のようになります (Windows では、「\」を「^」に置き換えてください)。

```
aws neptune create-db-instance \
  --db-instance-identifier (an instance ID for the new writer instance) \
  --db-cluster-identifier (ID of the DB cluster) \
  --db-instance-class db.serverless
  --engine neptune
```

新しいライターインスタンスが使用可能になったら、フェイルオーバーを実行してクラスターのライターインスタンスにします。

```
aws neptune failover-db-cluster \
  --db-cluster-identifier (ID of the DB cluster) \
  --target-db-instance-identifier (instance ID of the new serverless instance)
```

次に、古いライターインスタンスを削除します。

```
aws neptune delete-db-instance \
  --db-instance-identifier (instance ID of the old writer instance) \
  --skip-final-snapshot
```

最後に、同じ操作を行って、サーバーレスインスタンスにしたい既存のプロビジョニング済みリーダーインスタンスの代わりになる新しいサーバーレスインスタンスを作成し、既存のプロビジョニング済みインスタンスを削除します (リーダーインスタンスにはフェイルオーバーは必要ありません)。

既存のサーバーレス DB クラスターの容量範囲の変更

Neptune サーバーレス DB クラスターの容量範囲は、次のように AWS CLI を使用して変更できます (Windows では、「\」を「^」に置き換えます)。

```
aws neptune modify-db-cluster \
  --region (an AWS region that supports serverless) \
  --db-cluster-identifier (ID of the serverless DB cluster) \
  --apply-immediately \
  --serverless-v2-scaling-configuration MinCapacity=4.0, MaxCapacity=32
```

容量範囲を変更することで、一部の設定パラメータのデフォルト値が変更されます。Neptune は、一部の新しいデフォルトを直ちに適用できますが、一部の動的パラメータの変更は、再起動後に有効になります。pending-reboot ステータスは、一部のパラメータの変更を適用するために再起動が必要であることを示しています。

サーバーレス DB インスタンスをプロビジョニング済みに変更する

Neptune サーバーレスインスタンスをプロビジョニングされたインスタンスに変換するには、そのインスタンスクラスをプロビジョニングされたインスタンスクラスのいずれかに変更するだけです。Neptune DB インスタンスを変更する (その後すぐに適用する) を参照してください。

Amazon によるサーバーレス容量のモニタリング CloudWatch

CloudWatch を使用して、DB クラスター内の Neptune サーバーレスインスタンスの容量と使用率をモニタリングできます。クラスターレベルとインスタンスレベルの両方で現在のサーバーレス容量を追跡できる CloudWatch メトリクスは 2 つあります。

- **ServerlessDatabaseCapacity** — インスタンスレベルのメトリクスとして、ServerlessDatabaseCapacity は現在のインスタンス容量を NCU 単位で報告しま

す。クラスターレベルのメトリクスとして、クラスター内のすべての DB インスタンスの ServerlessDatabaseCapacity 値の平均を報告します。

- **NCUUtilization** — このメトリクスは、使用中の容量のパーセンテージを報告します。これは、現在の ServerlessDatabaseCapacity (インスタンスレベルまたはクラスターレベルで) を DB クラスターの最大容量設定で割って計算されます。

  このメトリクスがクラスターレベルで 100% に近づいた場合、つまりクラスターが可能な限り大きくスケーリングされた場合は、最大容量設定を増やすことを検討してください。

  ライターインスタンスが最大容量に近づいていないのに、リーダーインスタンスが 100% に近づいている場合は、リーダーインスタンスをさらに追加して読み取りワークロードを分散することを検討してください。

サーバーレスインスタンスとプロビジョニングされたインスタンスでは、CPUUtilization および FreeableMemory メトリクスの意味が少し異なることに注意してください。サーバーレスでは、CPUUtilization は、現在の CPU の使用量を、最大容量で使用可能な CPU の量で割った割合です。同様に、FreeableMemory は、インスタンスの容量が最大になった場合に使用可能な空きメモリの量を報告します。

次の例は、Linux AWS CLI で を使用して、1 時間にわたって 10 分ごとに測定される特定の DB インスタンスの最小、最大、平均容量値を取得する方法を示しています。Linux の date コマンドでは、現在の日付と時刻を基準にして開始時刻と終了時刻を指定します。--query パラメータの sort_by 関数は、Timestamp フィールドに基づいて結果を時系列でソートします。

```
aws cloudwatch get-metric-statistics \
  --metric-name "ServerlessDatabaseCapacity" \
  --start-time "$(date -d '1 hour ago')" \
  --end-time "$(date -d 'now')" \
  --period 600 \
  --namespace "AWS/Neptune"
  --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=(instance ID) \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' \
  --output table
```

Neptune Streams を使用してグラフ変更をリアルタイムでキャプチャする

Neptune Streams は、グラフへのすべての変更を、発生した順序で、フルマネージド型の方法でログに記録します。Streams を有効にすると、Neptune は可用性、バックアップ、セキュリティ、有効期限を処理します。

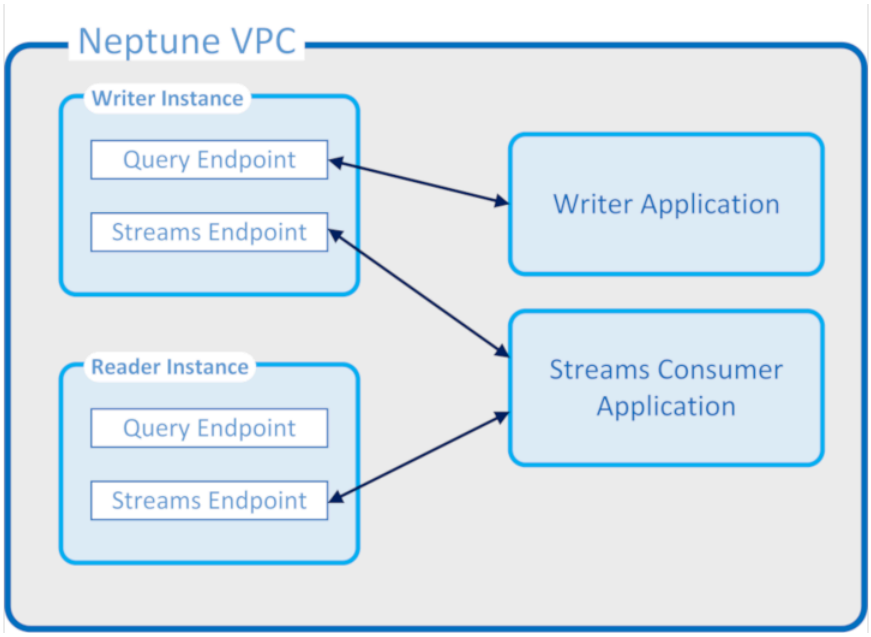
> **Note**
>
> この機能は リリース 1.0.1.0.200463.0 (2019-10-15) でラボモードから提供開始となり、Neptune エンジンリリース 1.0.2.2.R2 を皮切りに本稼働環境用に公開されます。

以下は、発生したグラフへの変更をキャプチャするユースケースの例です。

- 特定の変更が行われたときに、アプリケーションがユーザーに自動的に通知するようにする場合があります。
- グラフデータの最新バージョンを Amazon OpenSearch Service、Amazon 、 ElastiCacheAmazon Simple Storage Service (Amazon S3) などの別のデータストアに保持することもできます。

Neptune は、グラフデータと同じネイティブストレージを変更ログストリームに使用します。変更ログエントリを、その変更を行うトランザクションと共に同期的に書き込みます。HTTP REST API を使用して、ログストリームからこれらの変更レコードを取得します。(詳細については、「Streams API の呼び出し」を参照してください。)

次の図は、Neptune Streams から変更ログデータを取得する方法を示しています。



Neptune Streams 保証

- トランザクションによって行われた変更は、トランザクションが完了するとすぐにライターとリーダーの両方から読み取ることができます (リーダーの通常のレプリケーション遅延は除く)。
- 変更レコードは、発生した順序 (トランザクション内で行われた変更を含む) で厳密に連続して表示されます。
- 変更ストリームに重複は含まれません。各変更は 1 回のみ記録されます。
- 変更ストリームは完全です。変更が失われたり省略されたりすることはありません。
- 変更ストリームには、開始状態がわかっている限り、任意の時点でデータベース自体の完全な状態を判断するために必要なすべての情報が含まれます。
- Streams はいつでもオン/オフを切り替えることができます。

Neptune Streams 操作のプロパティ

- 変更ログストリームはフルマネージド型です。
- 変更ログデータは、変更を行う同じトランザクションの一部として同期的に書き込まれます。
- Neptune Streams を有効にすると、変更ログデータに関連付けられた I/O およびストレージ料金が発生します。
- デフォルトでは、変更レコードは、作成されてから 1 週間後に自動的に消去されます。エンジンリリース 1.2.0.0 以降、この保持期間は neptune_streams_expiry_days DB クラスターパラメータを使用して 1 から 90 までの任意の日数に変更できます。

- ストリームの読み取りパフォーマンスは、インスタンスに応じてスケールされます。
- リードレプリカを使用して、高可用性と高い読み込みスループットを達成できます。同時に作成して使用できるストリームリーダーの数に制限はありません。
- 変更ログデータは、複数のアベイラビリティーゾーンにまたがってレプリケートされるため、高い耐久性を実現できます。
- ログデータは、グラフデータ自体と同じくらい安全です。保管時および転送時に暗号化できます。アクセスは、IAM、Amazon VPC、および AWS Key Management Service () を使用して制御できますAWS KMS。グラフデータと同様に、復元 (PITR) を使用してバックアップおよび後で point-in-time 復元できます。
- 各トランザクションの一部としてストリームデータを同期的に書き込むと、全体的な書き込みパフォーマンスがわずかに低下します。
- Neptune は設計上シングルシャードされるため、ストリームデータはシャードされません。
- ログストリーム `GetRecords` API は、他のすべての Neptune グラフオペレーションと同じリソースを使用します。つまり、クライアントはストリームリクエストと他の DB リクエスト間で負荷を分散する必要があります。
- ストリームが無効になると、すべてのログデータにはすぐにアクセスできなくなります。つまり、ログ記録を無効にする前に、必要なすべてのログデータを読み取る必要があります。
- 現在、 とのネイティブ統合はありません AWS Lambda。ログストリームは、Lambda 関数をトリガーできるイベントを生成しません。

トピック

Neptune Streams の使用

Neptune Streams 機能を使用すると、グラフデータに加えられたすべての変更を記録する、変更ログエントリの完全なシーケンスを生成できます。この機能の概要については、「Neptune Streams を使用してグラフ変更をリアルタイムでキャプチャする」を参照してください。

トピック

Neptune Streams の有効化

neptune_streams DB クラスターパラメータを設定することで、いつでも Neptune Streams を有効または無効にできます。パラメータを 1 に設定すると Streams が有効になり、0 に設定すると Streams が無効になります。

> **Note**
>
> neptune_streams DB クラスターパラメータを変更した後、変更を有効にするには、クラスター内のすべての DB インスタンスを再起動する必要があります。

neptune_streams_expiry_days DB クラスターパラメータを設定して、ストリームレコードが削除されるまでにサーバー上に残る日数 (1 日から 90 日まで) を制御できます。デフォルトは 7 です。

Neptune Streams は、当初、DB クラスター neptune_lab_mode パラメータを使用してラボモードで有効または無効にする実験機能として導入されました (「Neptune ラボモード」を参照)。現在、ラボモードを使用した Streams の有効化は非推奨で、将来無効化される予定です。

Neptune Streams の無効化

Neptune Streams は、実行中であればいつでも無効にできます。

Streams をオフにするには、neptune_streams パラメータの値が 0 に設定されるように DB クラスターパラメータグループを更新します。

> **Important**
>
> Streams がオフになるとすぐに、変更ログデータにアクセスできなくなります。Streams をオフにする前に、関心のある内容を必ずお読みください。

Neptune Streams REST API の呼び出し

Neptune Streams にアクセスするには、次のいずれかのローカルエンドポイントに HTTP GET リクエストを送信する REST API を使用します。

- SPARQL グラフ DB の場合: `https://Neptune-DNS:8182/sparql/stream`。
- Gremlin または openCypher グラフ DB の場合: `https://Neptune-DNS:8182/propertygraph/stream` または `https://Neptune-DNS:8182/pg/stream`。

> **Note**
>
> エンジンリリース 1.1.0.0 現在、Gremlin ストリームエンドポイント (`https://Neptune-DNS:8182/gremlin/stream`) は、関連する出力形式 (`GREMLIN_JSON`) とともに非推奨です。下位互換性のために引き続きサポートされていますが、将来のリリースで削除される可能性があります。

HTTP `GET` オペレーションのみが許可されます。

Neptune は、レスポンスの `gzip` 圧縮をサポートします。ただし、HTTP リクエストに、受け入れられた圧縮形式として `gzip` を指定する `Accept-Encoding` ヘッダーが含まれていることが条件です (つまり、`"Accept-Encoding: gzip"`)。

パラメータ

- `limit` - long、オプション。範囲:1 ~ 100,000。デフォルト: 10

  返すレコードの最大数を指定します。また、レスポンスのサイズ制限は 10 MB であり、これは変更できず、`limit` パラメータで指定されたレコード数よりも優先されます。10 MB の制限に達した場合、レスポンスにはしきい値超過レコードが含まれます。

- `iteratorType` - 文字列、オプション。

  このパラメータには以下の値のいずれかがあります。

  - `AT_SEQUENCE_NUMBER` (デフォルト) - `commitNum` および `opNum` パラメータで一緒に指定されたイベントシーケンス番号から読み取りを開始することを示します。
  - `AFTER_SEQUENCE_NUMBER` - `commitNum` および `opNum` パラメータで一緒に指定されたイベントシーケンス番号の直後に読み取りが開始されることを示します。

- TRIM_HORIZON - 読み取りは、システム内の最後のトリミングされていないレコードから開始することを示します。これは、変更ログストリームで最も古い (まだ削除されていない) レコードであることを示しています。このモードは、特定の開始イベントシーケンス番号がないアプリケーションの起動時に便利です。
- LATEST - 読み取りは、システム内の最新のレコードから開始することを示します。これは、変更ログストリームで最近の (まだ削除されていない) レコードであることを示しています。これは、災害対策時やダウンタイムゼロのアップグレード時など、古いレコードを処理しないように、ストリームの現在の上位からレコードを読み取る必要がある場合に便利です。このモードでは、返されるレコードは最大 1 つだけであることに注意してください。

- commitNum - long、iteratorType が AT_SEQUENCE_NUMBER または AFTER_SEQUENCE_NUMBER のときは必須。

  変更ログストリームから読み取る開始レコードのコミット番号。

  iteratorType が TRIM_HORIZON または LATEST の場合、このパラメータは無視されます。

- opNum - long、オプション (デフォルトは 1)。

  変更ログストリームデータからの読み取りを開始するための、指定されたコミット内のオペレーションシーケンス番号。

通常、SPARQL グラフデータを変更するオペレーションでは、オペレーションごとに 1 つの変更レコードしか生成されません。ただし、Gremlin グラフデータを変更するオペレーションでは、次の例のように、オペレーションごとに複数の変更レコードを生成できます。

- INSERT - Gremlin 頂点は複数のラベルを持つことができ、Gremlin 要素は複数のプロパティを持つことができます。要素が挿入されると、ラベルとプロパティごとに個別の変更レコードが生成されます。
- UPDATE - Gremlin 要素プロパティが変更されると、2 つの変更レコードが生成されます。1 つ目は前の値の削除で、2 つ目は新しい値の挿入です。
- DELETE - 削除される要素プロパティごとに個別の変更レコードが生成されます。たとえば、プロパティを持つ Gremlin エッジが削除されると、プロパティごとに 1 つの変更レコードが生成されます。その後、エッジラベルの削除用に 1 つの変更レコードが生成されます。

  Gremlin 頂点が削除されると、すべての受信エッジプロパティと送信エッジプロパティが最初に削除され、次にエッジラベル、頂点プロパティ、最後に頂点ラベルが削除されます。これらの削除はそれぞれ、変更レコードを生成します。

Neptune Streams API レスポンスの形式

Neptune Streams REST API リクエストに対するレスポンスには、以下のフィールドがあります。

- `lastEventId` - ストリームレスポンスの最後の変更のシーケンス識別子。イベント ID は 2 つのフィールドで構成されます。`commitNum` はグラフを変更したトランザクションを識別し、`opNum` はそのトランザクション内の特定のオペレーションを識別します。以下の例ではこれを示しています。

```
"eventId": {
  "commitNum": 12,
  "opNum": 1
}
```

- `lastTrxTimestamp` - トランザクションのコミットがリクエストされた時間 (Unix エポックからのミリ秒単位)。
- `format` - 返される変更レコードのシリアル化形式。指定できる値は、Gremlin または openCypher 変更レコードの場合は `PG_JSON`、SPARQL 変更レコードの場合は `NQUADS` です。
- `records` - レスポンスに含まれるシリアル化された変更ログストリームレコードの配列。`records` 配列内の各レコードには、次のフィールドが含まれます。
  - `commitTimestamp` - トランザクションのコミットがリクエストされた時間 (Unix エポックからのミリ秒単位)。
  - `eventId` - ストリームレスポンスの最後の変更のシーケンス識別子。
  - `data` – シリアル化された Gremlin、SPARQL、または OpenCypher 変更レコード。各レコードのシリアル化形式については、次のセクション Neptune Streams のシリアル化形式 で詳しく説明します。
  - op — 変更を作成した操作。
  - `isLastOp` - この操作がトランザクションの最後の操作である場合にのみ表示されます。存在する場合は、`true` に設定されます。トランザクション全体が確実に消費されるようにする場合に便利です。
- `totalRecords` - レスポンスのレコードの総数。

例えば、次のレスポンスは、複数の操作を含むトランザクションの Gremlin 変更データを返します。

```
{
```

```
  "lastEventId": {
    "commitNum": 12,
    "opNum": 1
  },
  "lastTrxTimestamp": 1560011610678,
  "format": "PG_JSON",
  "records": [
    {
      "commitTimestamp": 1560011610678,
      "eventId": {
        "commitNum": 1,
        "opNum": 1
      },
      "data": {
        "id": "d2b59bf8-0d0f-218b-f68b-2aa7b0b1904a",
        "type": "vl",
        "key": "label",
        "value": {
          "value": "vertex",
          "dataType": "String"
        }
      },
      "op": "ADD"
    }
  ],
  "totalRecords": 1
}
```

次のレスポンスは、トランザクションの最後の操作 (トランザクション番号 97 の `EventId(97, 1)` によって識別される操作) の SPARQL 変更データを返します。

```
{
  "lastEventId": {
    "commitNum": 97,
    "opNum": 1
  },
  "lastTrxTimestamp": 1561489355102,
  "format": "NQUADS",
  "records": [
    {
      "commitTimestamp": 1561489355102,
      "eventId": {
        "commitNum": 97,
```

```
        "opNum": 1
      },
      "data": {
        "stmt": "<https://test.com/s> <https://test.com/p> <https://test.com/o> .\n"
      },
      "op": "ADD",
      "isLastOp": true
    }
  ],
  "totalRecords": 1
}
```

Neptune Streams API の例外

次の表では、Neptune Streams の例外について説明します。

エラーコード	HTTP コード	再試行してもいいですか。	メッセージ
InvalidParameterException	400	いいえ	無効な または out-of-range 値が入力パラメータとして指定されました。
ExpiredStreamException	400	いいえ	リクエストされたすべてのレコードが許容される最大有効期間を超え、有効期限が切れています。
ThrottlingException	500	はい	リクエストの速度が、最大スループットを超えています。
StreamRecordsNotFoundException	404	いいえ	リクエストされたリソースが見つかりませんでした。ストリームが正しく指定さ

エラーコード	HTTP コード	再試行してもいいですか。	メッセージ
			れていない可能性があります。
`MemoryLimitExceededException`	500	はい	メモリ不足のため、リクエストの処理は成功しませんでしたが、サーバーがビジー状態でなくなったら再試行できます。

Neptune Streams のシリアル化形式

Amazon Neptune では、2 つの異なる形式を使用して、グラフ変更データをログストリームにシリアル化します。これは、グラフが Gremlin と SPARQL のどちらを使用して作成されたかによって異なります。

両方のフォーマットは、Neptune Streams API レスポンスの形式 で説明されているように、共通のレコードシリアル化形式を共有します。これには、以下のフィールドが含まれています。

- `commitTimestamp` - トランザクションのコミットがリクエストされた時間 (Unix エポックからのミリ秒単位)。
- `eventId` - ストリームレスポンスの最後の変更のシーケンス識別子。
- `data` – シリアル化された Gremlin、SPARQL、または OpenCypher 変更レコード。各レコードのシリアル化形式については、次のセクションで詳しく説明します。
- op — 変更を作成した操作。

トピック

- PG_JSON 変更のシリアル化形式
- SPARQL NQUADS 変更のシリアル化形式

PG_JSON 変更のシリアル化形式

> **Note**
>
> エンジンリリース 1.1.0.0 現在、Gremlin ストリームエンドポイント (`https://Neptune-DNS:8182/gremlin/stream`) が出力した Gremlin ストリームの出力形式 (`GREMLIN_JSON`) は非推奨です。これは PG_JSON に置き換えられます。PG_JSON は現在 `GREMLIN_JSON` と同じです。

ログストリームレスポンスの `data` フィールドに含まれる Gremlin または openCypher 変更レコードには、以下のフィールドが含まれます。

- `id` - 文字列、必須。

  Gremlin または openCypher 要素の ID。
- `type` - 文字列、必須。

  この Gremlin または openCypher 要素のタイプ。以下のいずれかである必要があります。
  - `vl` — Gremlin の頂点ラベル、openCypher のノードラベル。
  - `vp` — Gremlin の頂点プロパティ、openCypher のノードプロパティ。
  - `e` — Gremlin のエッジとエッジラベル、openCypher のリレーションシップとリレーションシップタイプ。
  - `ep` — Gremlin のエッジプロパティ、openCypher のリレーションシッププロパティ。
- `key` - 文字列、必須。

  プロパティ名。要素ラベルの場合、これは "label" です。
- `value` – `value` オブジェクト、必須。

  これは、値自体の `value` フィールドと、その値の JSON データ型の `datatype` フィールドを含む JSON オブジェクトです。

  ```
  "value": {
    "value": "the new value",
    "dataType": "the JSON datatype of the new value"
  }
  ```
- `from` - 文字列、オプション。

これがエッジ (type="e") である場合、対応する from 頂点または始点ノードの ID。

- to - 文字列、オプション。

  これがエッジ (type="e") である場合、対応する to 頂点またはターゲットノードの ID。

Gremlin の例

- Gremlin 頂点ラベルの例を次に示します。

```
{
  "id": "an ID string",
  "type": "vl",
  "key": "label",
  "value": {
    "value": "the new value of the vertex label",
    "dataType": "String"
  }
}
```

- Gremlin 頂点プロパティの例を次に示します。

```
{
  "id": "an ID string",
  "type": "vp",
  "key": "the property name",
  "value": {
    "value": "the new value of the vertex property",
    "dataType": "the datatype of the vertex property"
  }
}
```

- Gremlin エッジの例を次に示します。

```
{
  "id": "an ID string",
  "type": "e",
  "key": "label",
  "value": {
    "value": "the new value of the edge",
    "dataType": "String"
  },
```

```
  "from": "the ID of the corresponding "from" vertex",
  "to": "the ID of the corresponding "to" vertex"
}
```

opencyPher の例

- 以下は、openCypher ノードラベルの例です。

```
{
  "id": "an ID string",
  "type": "vl",
  "key": "label",
  "value": {
    "value": "the new value of the node label",
    "dataType": "String"
  }
}
```

- 以下は、openCypher ノードプロパティの例です。

```
{
  "id": "an ID string",
  "type": "vp",
  "key": "the property name",
  "value": {
    "value": "the new value of the node property",
    "dataType": "the datatype of the node property"
  }
}
```

- 以下は、openCypher リレーションシップの例です。

```
{
  "id": "an ID string",
  "type": "e",
  "key": "label",
  "value": {
    "value": "the new value of the relationship",
    "dataType": "String"
  },
  "from": "the ID of the corresponding source node",
  "to": "the ID of the corresponding target node"
```

```
}
```

SPARQL NQUADS 変更のシリアル化形式

Neptune は W3C RDF 1.1 N-Quads 仕様で定義されているリソース記述フレームワーク (RDF) N-QUADS 言語を使用して、グラフ内の SPARQL クワッドの変更を記録します。

次の例のように、変更レコードの `data` フィールドには、変更されたクアッドを表す N-QUADS ステートメントを保持する `stmt` フィールドが含まれます。

```
  "stmt" : "<https://test.com/s> <https://test.com/p> <https://test.com/o> .\n"
```

Neptune Streams の例

次の例では、Amazon Neptune の変更ログストリームデータにアクセスする方法を示します。

トピック

- AT_SEQUENCE_NUMBER 変更ログ
- AFTER_SEQUENCE_NUMBER 変更ログ
- TRIM_HORIZON 変更ログ
- LATEST 変更ログ
- 圧縮変更ログ

AT_SEQUENCE_NUMBER 変更ログ

次の例は、Gremlin または openCypher `AT_SEQUENCE_NUMBER` 変更ログを示しています。

```
curl -s "https://Neptune-DNS:8182/propertygraph/stream?
limit=1&commitNum=1&opNum=1&iteratorType=AT_SEQUENCE_NUMBER" |jq
{
  "lastEventId": {
    "commitNum": 1,
    "opNum": 1
  },
  "lastTrxTimestamp": 1560011610678,
  "format": "PG_JSON",
  "records": [
```

```
    {
      "eventId": {
        "commitNum": 1,
        "opNum": 1
      },
      "commitTimestamp": 1560011610678,
      "data": {
        "id": "d2b59bf8-0d0f-218b-f68b-2aa7b0b1904a",
        "type": "vl",
        "key": "label",
        "value": {
          "value": "vertex",
          "dataType": "String"
        }
      },
      "op": "ADD",
      "isLastOp": true
    }
  ],
  "totalRecords": 1
}
```

これは、AT_SEQUENCE_NUMBER 変更ログの SPARQL の例を示しています。

```
curl -s "https://localhost:8182/sparql/stream?
limit=1&commitNum=1&opNum=1&iteratorType=AT_SEQUENCE_NUMBER" |jq
{
  "lastEventId": {
    "commitNum": 1,
    "opNum": 1
  },
  "lastTrxTimestamp": 1571252030566,
  "format": "NQUADS",
  "records": [
    {
      "eventId": {
        "commitNum": 1,
        "opNum": 1
      },
      "commitTimestamp": 1571252030566,
      "data": {
        "stmt": "<https://test.com/s> <https://test.com/p> <https://test.com/o> .\n"
      },
```

```
      "op": "ADD",
      "isLastOp": true
    }
  ],
  "totalRecords": 1
}
```

AFTER_SEQUENCE_NUMBER 変更ログ

次の例は、Gremlin または openCypher AFTER_SEQUENCE_NUMBER 変更ログを示しています。

```
curl -s "https://Neptune-DNS:8182/propertygraph/stream?
limit=1&commitNum=1&opNum=1&iteratorType=AFTER_SEQUENCE_NUMBER" |jq
{
  "lastEventId": {
    "commitNum": 2,
    "opNum": 1
  },
  "lastTrxTimestamp": 1560011633768,
  "format": "PG_JSON",
  "records": [
    {
      "commitTimestamp": 1560011633768,
      "eventId": {
        "commitNum": 2,
        "opNum": 1
      },
      "data": {
        "id": "d2b59bf8-0d0f-218b-f68b-2aa7b0b1904a",
        "type": "vl",
        "key": "label",
        "value": {
          "value": "vertex",
          "dataType": "String"
        }
      },
      "op": "REMOVE",
      "isLastOp": true
    }
  ],
  "totalRecords": 1
}
```

TRIM_HORIZON 変更ログ

次の例は、Gremlin または openCypher TRIM_HORIZON 変更ログを示しています。

```
curl -s "https://Neptune-DNS:8182/propertygraph/stream?
limit=1&iteratorType=TRIM_HORIZON" |jq
{
  "lastEventId": {
    "commitNum": 1,
    "opNum": 1
  },
  "lastTrxTimestamp": 1560011610678,
  "format": "PG_JSON",
  "records": [
    {
      "commitTimestamp": 1560011610678,
      "eventId": {
        "commitNum": 1,
        "opNum": 1
      },
      "data": {
        "id": "d2b59bf8-0d0f-218b-f68b-2aa7b0b1904a",
        "type": "vl",
        "key": "label",
        "value": {
          "value": "vertex",
          "dataType": "String"
        }
      },
      "op": "ADD",
      "isLastOp": true
    }
  ],
  "totalRecords": 1
}
```

LATEST 変更ログ

次の例は、Gremlin または openCypher LATEST 変更ログを示しています。API パラメータ limit、commitNum および opNum は完全に任意であることに注意してください。

```
curl -s "https://Neptune-DNS:8182/propertygraph/stream?iteratorType=LATEST" | jq
```

```
{
  "lastEventId": {
    "commitNum": 21,
    "opNum": 4
  },
  "lastTrxTimestamp": 1634710497743,
  "format": "PG_JSON",
  "records": [
    {
      "commitTimestamp": 1634710497743,
      "eventId": {
        "commitNum": 21,
        "opNum": 4
      },
      "data": {
        "id": "24be4e2b-53b9-b195-56ba-3f48fa2b60ac",
        "type": "e",
        "key": "label",
        "value": {
          "value": "created",
          "dataType": "String"
        },
        "from": "4",
        "to": "5"
      },
      "op": "REMOVE",
      "isLastOp": true
    }
  ],
  "totalRecords": 1
}
```

圧縮変更ログ

次の例は、Gremlin または openCypher の圧縮変更ログを示しています。

```
curl -sH \
  "Accept-Encoding: gzip" \
  "https://Neptune-DNS:8182/propertygraph/stream?limit=1&commitNum=1" \
  -H "Accept-Encoding: gzip" \
  -v |gunzip -|jq
> GET /propertygraph/stream?limit=1 HTTP/1.1
> Host: localhost:8182
```

```
> User-Agent: curl/7.64.0
> Accept: /
> Accept-Encoding: gzip
*> Accept-Encoding: gzip*
>
< HTTP/1.1 200 OK
< Content-Type: application/json; charset=UTF-8
< Connection: keep-alive
*< content-encoding: gzip*
< content-length: 191
<
{ [191 bytes data]
Connection #0 to host localhost left intact
{
  "lastEventId": "1:1",
  "lastTrxTimestamp": 1558942160603,
  "format": "PG_JSON",
  "records": [
    {
      "commitTimestamp": 1558942160603,
      "eventId": "1:1",
      "data": {
        "id": "v1",
        "type": "vl",
        "key": "label",
        "value": {
          "value": "person",
          "dataType": "String"
        }
      },
      "op": "ADD",
      "isLastOp": true
    }
  ],
  "totalRecords": 1
}
```

AWS CloudFormation を使用して Streams コンシューマーアプリケーションで Neptune から Neptune へのレプリケーションを設定する

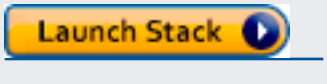
AWS CloudFormation テンプレートを使用して、Neptune から Neptune へのレプリケーションをサポートするように Neptune ストリームコンシューマーアプリケーションをセットアップできます。

トピック

- リージョンの AWS CloudFormation テンプレートを選択する
- 作成中の Neptune Streams コンシューマースタックの詳細の追加
- AWS CloudFormation テンプレートを実行する
- 最新の Lambda アーティファクトでストリームポーラーを更新するには

リージョンの AWS CloudFormation テンプレートを選択する

AWS CloudFormation コンソールで適切な AWS CloudFormation スタックを起動するには、使用するリージョンに応じて AWS 、次の表の「スタック起動」ボタンのいずれかを選択します。

リージョン	ビュー	デザイナーで表示	起動する
米国東部 (バージニア北部)	表示	デザイナーで表示	
米国東部 (オハイオ)	表示	デザイナーで表示	
米国西部 (北カリフォルニア)	表示	デザイナーで表示	
米国西部 (オレゴン)	表示	デザイナーで表示	
カナダ (中部)	表示	デザイナーで表示	
南米 (サンパウロ)	表示	デザイナーで表示	

リージョン	ビュー	デザイナーで表示	起動する
欧州 (ストックホルム)	表示	デザイナーで表示	Launch Stack
欧州 (アイルランド)	表示	デザイナーで表示	Launch Stack
欧州 (ロンドン)	表示	デザイナーで表示	Launch Stack
欧州 (パリ)	表示	デザイナーで表示	Launch Stack
欧州 (フランクフルト)	表示	デザイナーで表示	Launch Stack
中東 (バーレーン)	表示	デザイナーで表示	Launch Stack
中東 (アラブ首長国連邦)	表示	デザイナーで表示	Launch Stack
イスラエル (テルアビブ)	表示	デザイナーで表示	Launch Stack
アフリカ (ケープタウン)	表示	デザイナーで表示	Launch Stack
アジアパシフィック (東京)	表示	デザイナーで表示	Launch Stack
アジアパシフィック (香港)	表示	デザイナーで表示	Launch Stack
アジアパシフィック (ソウル)	表示	デザイナーで表示	Launch Stack
アジアパシフィック (シンガポール)	表示	デザイナーで表示	Launch Stack

リージョン	ビュー	デザイナーで表示	起動する
アジアパシフィック (シドニー)	表示	デザイナーで表示	Launch Stack
アジアパシフィック（ムンバイ）	表示	デザイナーで表示	Launch Stack
中国 (北京)	表示	デザイナーで表示	Launch Stack
中国 (寧夏)	表示	デザイナーで表示	Launch Stack
AWS GovCloud（米国西部）	表示	デザイナーで表示	Launch Stack
AWS GovCloud（米国東部）	表示	デザイナーで表示	Launch Stack

[Create Stack (スタックの作成)] ページで、[Next (次へ)] を選択します。

作成中の Neptune Streams コンシューマースタックの詳細の追加

[Specify Stack Details (スタック詳細の指定)] ページには、アプリケーションの設定を制御するために使用できるプロパティとパラメータがあります。

スタック名 – 作成する新しい AWS CloudFormation スタックの名前。通常は、デフォルト値、`NeptuneStreamPoller` を使用できます。

[Parameters (パラメータ)] で、以下を指定します。

Streams コンシューマーが実行される VPC のネットワーク設定

- **`VPC`** – ポーリング Lambda 関数を実行する VPC の名前を指定します。
- **`SubnetIDs`** - ネットワークインターフェースが確立されるサブネット。Neptune クラスターに対応するサブネットを追加します。
- **`SecurityGroupIds`** –ソース Neptune DB クラスターへの書き込みインバウンドアクセスを許可するセキュリティグループの ID を指定します。

- **RouteTableIds** – これは、Neptune VPC に Amazon DynamoDB エンドポイントを作成するために必要です (まだエンドポイントを持っていない場合)。サブネットに関連付けられたルートテーブル ID のコンマ区切りリストを指定する必要があります。
- **CreateDDBVPCEndPoint** — ブール値。デフォルトは `true` であり、Dynamo DB VPC エンドポイントを作成する必要があるかどうかを示します。この値を `false` に変更する必要があるのは、VPC に作成済みの DynamoDB エンドポイントがある場合のみです。
- **CreateMonitoringEndPoint** — ブール値。デフォルトは `true` であり、モニタリング VPC エンドポイントを作成する必要があるかどうかを示します。この値を `false` に変更する必要があるのは、VPC に作成済みのモニタリングエンドポイントがある場合のみです。

ストリームポーラー

- **ApplicationName** – 通常は、この設定をデフォルト (`NeptuneStream`) のままにしておくことができます。別の名前を使用する場合は、一意である必要があります。
- **LambdaMemorySize** - Lambda ポーラー関数で使用可能なメモリサイズを設定するために使用されます。デフォルト値は 2,048 メガバイトです。
- **LambdaRuntime** – ストリームから項目を取得する Lambda 関数で使用される言語。これは `python3.9` または `java8` どちらにも設定できます。
- **LambdaS3Bucket** — Lambda コードのアーティファクトを含む Amazon S3 バケット。別の Amazon S3 バケットからロードするカスタム Lambda ポーリング関数を使用している場合を除き、空白のままにしておきます。
- **LambdaS3Key** — Lambda コードのアーティファクトに対応する Amazon S3 キー。カスタム Lambda ポーリング関数を使用している場合を除き、空白のままにしておきます。
- **LambdaLoggingLevel** – 通常、デフォルト値 (`INFO`) のままにしておきます。
- **ManagedPolicies** — Lambda 関数の実行に使用する管理ポリシーを一覧表示します。カスタム Lambda ポーリング関数を使用している場合を除き、通常は空白のままにしておきます。
- **StreamRecordsHandler** – Neptune ストリーム内のレコードにカスタムハンドラを使用している場合を除き、通常は空白のままにしておきます。
- **StreamRecordsBatchSize** — ストリームからフェッチされるレコードの最大数。このパラメータを使用して、パフォーマンスを調整できます。デフォルト (`5000`) は、開始に適しています。最大許容値は 10,000 です。数値が大きいほど、ストリームからレコードを読み取るために必要なネットワークの呼び出しは少なくなりますが、レコードを処理するために必要なメモリは多くなります。このパラメータの値を小さくすると、スループットが低下します。

- **`MaxPollingWaitTime`** — 2 回のポーリング間の最大待機時間 (秒単位)。Neptune ストリームをポーリングするために Lambda ポーラーが呼び出される頻度を決定します。連続ポーリングの場合は、この値を 0 に設定します。最大値は 3,600 秒 (60 分) です。グラフデータの変化速度に応じて、デフォルト値 (60 秒) は開始に適しています。
- **`MaxPollingInterval`** — 最大連続ポーリング時間 (秒単位)。これを使用して、Lambda ポーリング関数のタイムアウトを設定します。値は 5 秒から 900 秒の範囲でなければなりません。デフォルト値 (600 秒) は、開始に適しています。
- **`StepFunctionFallbackPeriod`** – ポーラーを待機 step-function-fallback-period する のユニット数。その後、Amazon CloudWatch Events を介してステップ関数が呼び出され、障害から回復します。デフォルト (5 分) は、開始に適しています。
- **`StepFunctionFallbackPeriodUnit`** - 直前の `StepFunctionFallbackPeriodUnit` の測定に使用される時間単位 (`minutes`、`hours`、または `days`)。通常は、デフォルト (`minutes`) で十分です。

Neptune Stream

- **`NeptuneStreamEndpoint`** — (必須) Neptune ソースストリームのエンドポイント。: これは次の 2 つの形式のいずれかになります。
  - **`https://`*`your DB cluster`*`:`*`port`*`/propertygraph/stream`** (またはそのエイリアス `https://`*`your DB cluster`*`:`*`port`*`/pg/stream`)。
  - **`https://`*`your DB cluster`*`:`*`port`*`/sparql/stream`**.
- **`Neptune Query Engine`** — Gremlin、openCypher、または SPARQL を選択します。
- **`IAMAuthEnabledOnSourceStream`** – Neptune DB クラスターで IAM 認証を使用している場合は、このパラメータを `true` に設定します。
- **`StreamDBClusterResourceId`** – Neptune DB クラスターで IAM 認証を使用している場合は、このパラメータをクラスターのリソース ID に設定します。リソース ID はクラスター ID と同じではありません。正確には、28 文字の英数字が続く `cluster-` です。Neptune コンソールの [Cluster Details] (クラスターの詳細) で確認できます。

ターゲットの Neptune DB クラスター

- **`TargetNeptuneClusterEndpoint`** — ターゲットのバックアップクラスターのクラスターエンドポイント (ホスト名のみ)。

TargetNeptuneClusterEndpoint を指定した場合、TargetSPARQLUpdateEndpoint も指定することはできません。

- **TargetNeptuneClusterPort** — ターゲットクラスターのポート番号。

  TargetSPARQLUpdateEndpoint を指定した場合、TargetNeptuneClusterPort の設定は無視されることに注意してください。

- **IAMAuthEnabledOnTargetCluster** — ターゲットクラスターで IAM 認証を有効にする場合は true に設定します。

- **TargetAWSRegion** – などのターゲットバックアップクラスターの AWS リージョンus-east-1 ）。このパラメータは、クロス AWS リージョンレプリケーションの場合と同様に、ターゲットバックアップクラスターのリージョンが Neptune ソースクラスターのリージョンと異なる場合にのみ指定する必要があります。ソースリージョンとターゲットリージョンが同じ場合、このパラメータはオプションです。

  TargetAWSRegion 値が Neptune がサポートする有効な AWS リージョンではない場合、プロセスは失敗することに注意してください。

- **TargetNeptuneDBClusterResourceId** — オプション: これはターゲット DB クラスターで IAM 認証が有効になっている場合にのみ必要です。ターゲットクラスターのリソース ID に設定します。

- **SPARQLTripleOnlyMode** — トリプルオンリーモードを有効にするかどうかを決定するブール型フラグ。トリプルオンリーモードでは、名前付きグラフの複製は行われません。デフォルト値は、falseです。

- **TargetSPARQLUpdateEndpoint** — https://abc.com/xyz など、SPARQL 更新のターゲットエンドポイントの URL。このエンドポイントは、クアッドまたはトリプルをサポートする任意の SPARQL ストアでもかまいません。

  TargetNeptuneClusterEndpoint を指定した場合、TargetNeptuneClusterPort も指定することはできず、TargetSPARQLUpdateEndpoint の設定は無視されることに注意してください。

- **BlockSparqlReplicationOnBlankNode** – ブールフラグ。 に設定するとtrue、SPARQL (RDF) データ BlankNode 内の のレプリケーションが停止します。デフォルト値は、falseです。

アラーム

- **`Required to create Cloud watch Alarm`** – 新しいスタックの CloudWatch アラームを作成するtrue場合は、これを に設定します。
- **`SNS Topic ARN for Cloudwatch Alarm Notifications`** – CloudWatch アラーム通知を送信する SNS トピック ARN (アラームが有効になっている場合にのみ必要) 。
- **`Email for Alarm Notifications`** - アラーム通知の送信先の E メールアドレス (アラームが有効な場合のみ必要)。

アラーム通知の宛先には、SNS のみ、E メールのみ、または SNS と E メールの両方を追加できます。

AWS CloudFormation テンプレートを実行する

これで、次のように、Neptune ストリームコンシューマーアプリケーションインスタンスをプロビジョニングするプロセスを完了できます。

1. で AWS CloudFormation、スタックの詳細の指定 ページで、次へ を選択します。
2. [Options(オプション)] ページで、[Next(次へ)] を選択します。
3. [確認] ページで、 AWS CloudFormation によって IAM リソースが作成されることを確認する最初のチェックボックスをオンにします。新しいスタックの `CAPABILITY_AUTO_EXPAND` を確認する 2 つ目のチェックボックスをオンにします。

   > **Note**
   >
   > `CAPABILITY_AUTO_EXPAND` は、スタックの作成時に事前の確認なしにマクロが展開されることを明示的に確認します。ユーザーは、処理されたテンプレートから変更セットを作成することが多いため、実際にスタックを作成する前にマクロによって行われた変更を確認できます。詳細については、「 API リファレンス」の AWS CloudFormation CreateStack「 API」を参照してください。 AWS CloudFormation

   次に [作成] を選択します。

最新の Lambda アーティファクトでストリームポーラーを更新するには

最新の Lambda アーティファクトで次のようにストリームポーラーを更新できます。

1. で AWS Management Console、メインの親 AWS CloudFormation スタックに移動 AWS CloudFormation して選択します。
2. スタックの[Update] (更新) オプションを選択します。
3. [Replace current template] (現在のテンプレートを置換) を選択します。
4. テンプレートソースで、Amazon S3 URL を選択し、次の S3 URL を入力します。

```
https://aws-neptune-customer-samples.s3.amazonaws.com/neptune-stream/
neptune_to_neptune.json
```

5. AWS CloudFormation パラメータを変更せずに次へを選択します。
6. [Update Stack] を選択します。

これで、スタックは Lambda アーティファクトを最新のアーティファクトで更新します。

災害対策に Neptune ストリームクロスリージョンレプリケーションを使用する

Neptune は、クロスリージョンフェイルオーバー機能を実装する 2 つの方法を提供します。

- クロスリージョンスナップショットのコピーと復元
- Neptune ストリームを使用して、2 つの異なるリージョンの 2 つのクラスター間でデータをレプリケートします。

クロスリージョンスナップショットのコピーと復元では、異なるリージョンの Neptune クラスターを復旧する際の運用オーバーヘッドが最小限に抑えられます。ただし、リージョン間でスナップショットをコピーするには、スナップショットが Neptune クラスターのフルバックアップであるため、データ転送にかなりの時間がかかることがあります。その結果、クロスリージョンスナップショットのコピーと復元は、目標復旧時点 (RPO) の時間と目標復旧時間 (RTO) の時間のみを必要とするシナリオに使用できます。

目標復旧時点 (RPO) は、バックアップ間の時間によって測定されます。これは、最後のバックアップが作成されてからデータベースをリカバリする時点までの間に損失する可能性のあるデータの量を定義します。

目標復旧時間 (RTO) は、リカバリオペレーションの実行にかかる時間によって測定されます。これは、障害発生後に DB クラスターが復旧されたデータベースにフェイルオーバーするのにかかる時間です。

Neptune ストリームは、バックアップ Neptune クラスターをプライマリ本番クラスターと常に同期させる方法を提供します。障害が発生すると、データベースはバックアップクラスターにフェイルオーバーします。これにより、データは常にバックアップクラスターにコピーされるため、RPO と RTO が数分に短縮されます。これは、いつでもフェイルオーバーターゲットとしてすぐに利用できるためです。

この方法で Neptune ストリームを使用する場合の欠点は、レプリケーションコンポーネントの維持に必要な運用オーバーヘッドと、2 番目の Neptune DB クラスターを常にオンラインにするコストの両方が大きな場合があることです。

Neptune から Neptune へのレプリケーションセットアップ

プライマリの本番 DB クラスターは、特定のソースリージョンの VPC に存在します。災害対策の目的で、別のリカバリリージョンでレプリケートまたはエミュレートする必要があるのは、主に次の 3 つです。

- クラスターに格納されているデータ。
- プライマリクラスターの設定。これには、IAM 認証を使用するかどうか、暗号化されているかどうか、DB クラスターパラメータ、インスタンスパラメータ、インスタンスサイズなどが含まれます)。
- ターゲット VPC、セキュリティグループなど、使用するネットワークトポロジ。

次のような Neptune 管理 API を使用して、その情報を収集できます。

- DescribeDBClusters
- DescribeDBInstances
- DescribeDBClusterParameters
- DescribeDBParameters
- DescribeVpcs

収集した情報を使用して、次の手順を使用して、障害が発生した場合に本番クラスターをフェイルオーバーできる別のリージョンにバックアップクラスターをセットアップできます。

1: Neptune ストリームを有効にします。

ModifyDBClusterParameterGroup を使用して neptune_streams パラメータを 1 に設定します。次に、DB クラスター内のすべてのインスタンスを再起動して、変更を有効にします。

Neptune ストリームを有効にした後に、ソース DB クラスターで少なくとも 1 つの追加または更新操作を実行することをお勧めします。これにより、本番クラスタをバックアップクラスタに再同期するときに後で参照できるデータポイントが変更ストリームに追加されます。

2: バックアップクラスターを設定するリージョンに新しい VPC を作成します。

プライマリクラスターとは異なるリージョンに新しい Neptune DB クラスターを作成する前に、クラスターをホストするターゲットリージョンに新しい VPC を確立する必要があります。プライマリクラスターとバックアップクラスター間の接続は、異なる VPC 内のプライベートサブネット間のトラフィックを使用する VPC ピアリングによって確立されます。ただし、2 つの VPC 間で VPC ピアリングを確立するには、重複する CIDR ブロックまたは IP アドレススペースがあってはなりません。これは、デフォルトの VPC の CIDR ブロックが常に同じであるため、両方のリージョンでデフォルト VPC を使用することはできません (172.31.0.0/16)。

以下の条件を満たしている限り、ターゲットリージョンで既存の VPC を使用できます。

- プライマリクラスターが配置されている VPC の CIDR ブロックと重複する CIDR ブロックはありません。
- プライマリクラスターがある VPC と同じ CIDR ブロックを持つ別の VPC とピアリングされていません。

ターゲットリージョンに適切な VPC がない場合は、Amazon EC2 CreateVpc API を使用して作成します。

3: プライマリクラスターのスナップショットを作成し、それをターゲットバックアップリージョンとして復元します。

次に、本番クラスターのコピーであるターゲットバックアップリージョン内の適切な VPC に新しい Neptune クラスターを作成します。

バックアップリージョンに本番クラスターのコピーを作成します。

1. ターゲットのバックアップリージョンで、本番 DB クラスターで使用されるパラメータとパラメータグループを再作成します。これを、CreateDBClusterParameterGroup

および CreateDBParameterGroup、ModifyDBClusterParameterGroup および ModifyDBParameterGroup を使用して行うことができます。

CopyDBClusterParameterGroup および CopyDBParameterGroup API は現在クロスリージョンコピーをサポートしていないという点に注意してください。

2. CreateDBClusterSnapshot を使用して、本番リージョンの VPC に本番クラスターのスナップショットを作成します。
3. CopyDBClusterSnapshot を使用して、ターゲットバックアップリージョンの VPC にスナップショットをコピーします。
4. RestoreDBClusterFromSnapshot を使用して、コピーしたスナップショットを使用して、ターゲットバックアップリージョンの VPC 内に新しい DB クラスターを作成します。プライマリ本番クラスターからコピーした構成設定とパラメータを使用します。
5. 新しい Neptune クラスターは現在存在しますが、インスタンスは一切含まれていません。CreateDBInstance を使用して、本番クラスターのライターインスタンスと同じインスタンスタイプとサイズを持つ新しいプライマリ/ライターインスタンスを作成します。フェイルオーバーの前に、バックアップインスタンスを使用してターゲットリージョンでの読み取り I/O の処理を行わない限り、この時点で追加のリードレプリカを作成する必要はありません。

4: プライマリクラスターの VPC と新しいバックアップクラスターの VPC 間の VPC ピアリングを確立します。

VPC ピアリングを設定することで、プライマリクラスターの VPC が単一のプライベートネットワークであるかのようにバックアップクラスターの VPC と通信できるようになります。これを行うには、以下のステップを行います：

1. 本番クラスターの VPC から、CreateVpcPeeringConnection API を呼び出し、ピアリング接続を確立します。
2. ターゲットバックアップクラスターの VPC から、AcceptVpcPeeringConnection API を呼び出し、ピアリング接続を受け入れます。
3. 本番クラスターの VPC から、CreateRoute API を使用して、VPC ピアリングプレフィクスリストを使用するように、すべてのトラフィックをターゲット VPC の CIDR ブロックにリダイレクトする VPC のルートテーブルにルートを追加します。
4. 同様に、ターゲットバックアップクラスターの VPC から、CreateRoute API を使用して、プライマリクラスターの VPC にトラフィックをルーティングする VPC のルートテーブルにルートを追加します。

5: Neptune ストリームレプリケーションインフラストラクチャをセットアップします。

両方のクラスターがデプロイされ、両方のリージョン間のネットワーク通信が確立されたら、Neptune-to-Neptune AWS CloudFormation テンプレートを使用して、データレプリケーションをサポートする追加のインフラストラクチャで Neptune ストリームコンシューマー Lambda 関数をデプロイします。これは、プライマリ本番クラスターの VPC で実行します。

この AWS CloudFormation スタックに提供する必要があるパラメータは次のとおりです。

- **`NeptuneStreamEndpoint`** — プライマリクラスターのストリームエンドポイント (URL 形式)。例: `https://(cluster name):8182/pg/stream`。
- **`QueryEngine`** – これは `gremlin`、`sparql`、または `openCypher` のいずれかである必要があります。
- **`RouteTableIds`** — DynamoDB VPC エンドポイントとモニタリング VPC エンドポイントの両方にルートを追加できます。

  2 つの追加パラメータ、すなわち `CreateMonitoringEndpoint` および `CreateDynamoDBEndpoint` は、プライマリクラスターの VPC にまだ存在しない場合は true に設定する必要があります。既に存在する場合は、false に設定されていることを確認してください。設定されていない場合、 AWS CloudFormation 作成は失敗します。
- **`SecurityGroupIds`** — Lambda コンシューマーがプライマリクラスターの Neptune ストリームエンドポイントと通信するために使用するセキュリティグループを指定します。

  ターゲットバックアップクラスターで、このセキュリティグループから発信されるトラフィックを許可するセキュリティグループをアタッチします。
- **`SubnetIds`** — Lambda コンシューマーがプライマリクラスターと通信するために使用できる、プライマリクラスターの VPC 内のサブネット ID のリスト。
- **`TargetNeptuneClusterEndpoint`** — ターゲットのバックアップクラスターのクラスターエンドポイント (ホスト名のみ)。
- **`TargetAWSRegion`** – などのターゲットバックアップクラスターの AWS リージョン`us-east-1`)。このパラメータは、クロス AWS リージョンレプリケーションの場合と同様に、ターゲットバックアップクラスターのリージョンが Neptune ソースクラスターのリージョンと異なる場合にのみ指定する必要があります。ソースリージョンとターゲットリージョンが同じ場合、このパラメータはオプションです。

`TargetAWSRegion` 値が Neptune がサポートする有効な AWS リージョンではない場合、プロセスは失敗することに注意してください。

- **VPC** — プライマリクラスターの VPC の ID。

その他のパラメータはすべて、デフォルト値のままにしておくことができます。

AWS CloudFormation テンプレートがデプロイされると、Neptune はプライマリクラスターからバックアップクラスターへの変更のレプリケーションを開始します。このレプリケーションは、Lambda コンシューマー関数によって生成された CloudWatch ログでモニタリングできます。

その他の考慮事項

- プライマリクラスターとバックアップクラスター間で IAM 認証を使用する必要がある場合は、AWS CloudFormation テンプレートを呼び出すときにセットアップすることもできます。
- プライマリクラスターで保存時の暗号化が有効になっている場合は、スナップショットをターゲットリージョンにコピーするときに、関連する KMS キーを管理し、ターゲットリージョンに新しい KMS キーを関連付ける方法を検討します。
- ベストプラクティスは、アプリケーションで使用される Neptune エンドポイントの前で DNS CNAME を使用することです。その後、ターゲットのバックアップクラスタに手動でフェイルオーバーする必要がある場合は、これらの CNAME をターゲットクラスタまたはインスタンスのエンドポイントを指すように変更できます。

アマゾンサービスを使ったAmazon Neptune での全文検索 OpenSearch

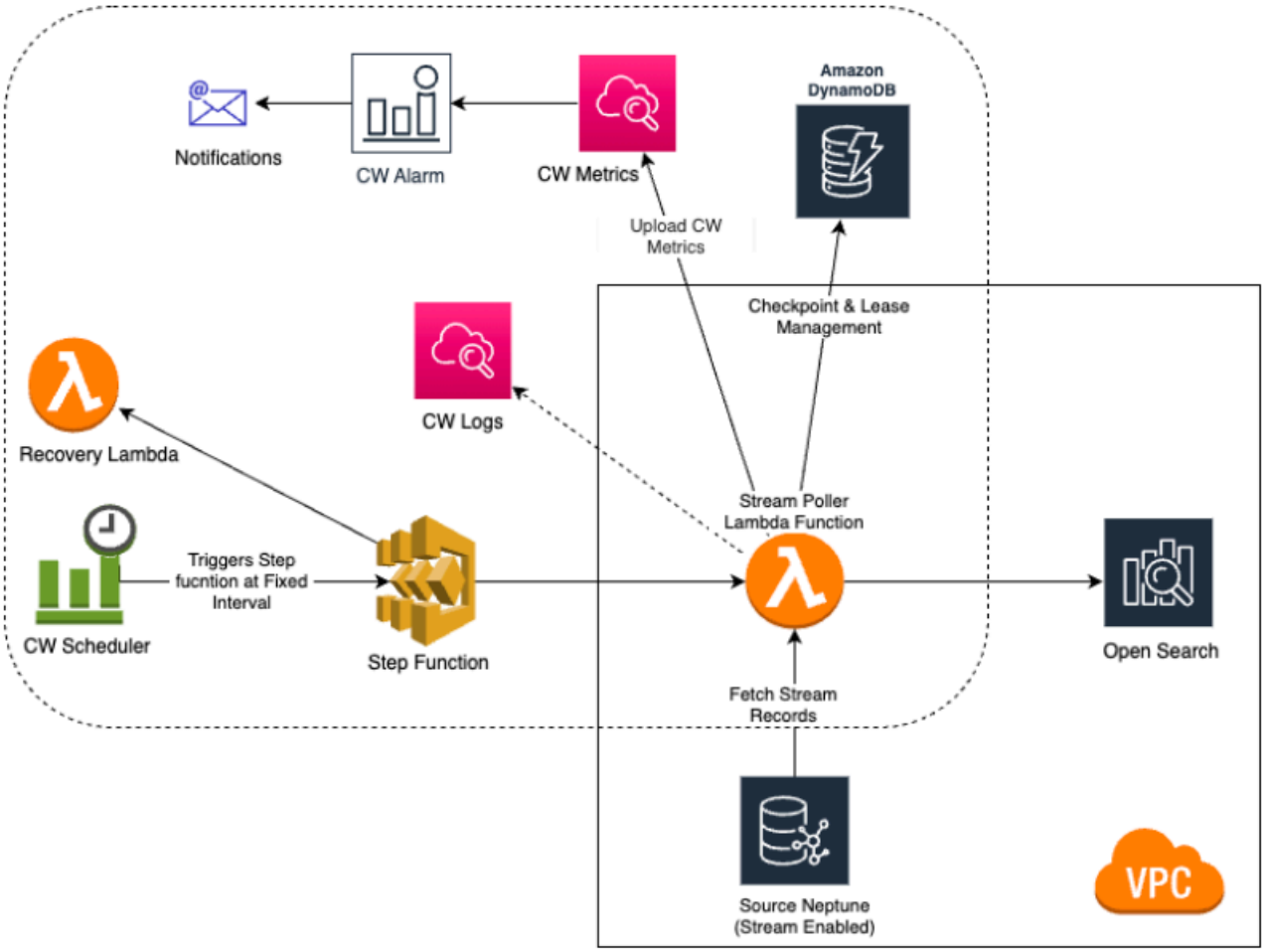
Neptune はAmazon OpenSearch サービス (OpenSearch サービス) と統合されており、Gremlin と SPARQL の両方のクエリで全文検索をサポートします。この機能は、Neptune エンジンリリース 1.0.2.1 以降で使用できます。`1.0.4.2` 以降のエンジンリリースで使用することをお勧めしますが、最新の修正を利用できます。

エンジンリリース 1.3.0.0 以降、Amazon Neptune は Gremlin クエリと SPARQL クエリでのフルテキスト検索に Amazon OpenSearch サービスサーバーレスを使用することをサポートしています。

> **Note**
>
> Amazon OpenSearch サービスと統合する場合、Neptune ンはElasticsearchのバージョン7.1以降を必要とし、2.3、 OpenSearch 2.5以降で動作します。Neptune はサーバーレスでも動作します。OpenSearch

Neptune は、 OpenSearch に従ってデータが入力された既存のサービスクラスタで使用できます。OpenSearch データの Neptune データモデルまたは、スタックを使用して Neptune OpenSearch にリンクされたサービスドメインを作成することもできます。 AWS CloudFormation

> **Important**
>
> ここで説明する Neptune OpenSearch からレプリケーションへのプロセスでは、ブランクノードはレプリケートされません。これは注意すべき重要な制限です。
> また、 OpenSearch クラスターで詳細なアクセス制御を有効にする場合は、Neptune データベースでも IAM 認証を有効にする必要があります。



トピック

- アマゾンネプチューンからレプリケーションへ OpenSearch
- OpenSearch サーバーレスへのレプリケーション
- きめ細かいアクセスコントロール (FGAC) が有効になっている OpenSearch クラスターからのクエリ
- Neptune フルテキスト検索クエリでの Apache Lucene クエリ構文
- OpenSearch データの Neptune データモデル
- Neptune フルテキスト検索パラメータ
- Amazon Neptune での文字列以外の OpenSearch インデックス作成
- Amazon Neptune でのフルテキスト検索クエリの実行
- Neptune のフルテキスト検索を使用した SPARQL クエリの例
- Gremlin クエリでの Neptune フルテキスト検索の使用
- Neptune フルテキスト検索のトラブルシューティング

アマゾンネプチューンからレプリケーションへ OpenSearch

Amazon Neptune は、Amazon サービス (サービス) を使用した Gremlin クエリと SPARQL クエリでの全文検索をサポートしています。 OpenSearch OpenSearch AWS CloudFormation スタックを使用して、 OpenSearch サービスドメインを Neptune にリンクできます。 AWS CloudFormation このテンプレートは、NepTune-to-レプリケーションを提供するストリームコンシューマーアプリケーションインスタンスを作成します。OpenSearch

開始する前に、ソースとして機能するストリームが有効になっている既存の Neptune DB クラスターと、 OpenSearch レプリケーションターゲットとして機能するサービスドメインが必要です。

Neptune DB クラスターが配置されている VPC に Lambda OpenSearch からアクセスできる既存のターゲットサービスドメインがある場合、テンプレートはそのサービスドメインを使用できます。それ以外の場合は、新しいものを作成する必要があります。

> **Note**
>
> OpenSearch 作成するクラスターと Lambda 関数は Neptune DB クラスターと同じ VPC に配置する必要があり、クラスターは VPC モード (インターネットモードではなく) で設定する必要があります。 OpenSearch

新しく作成した Neptune インスタンスを使用して Service で使用することをお勧めします。OpenSearch すでにデータが含まれている既存のインスタンスを使用する場合は、クエリを実行する前に OpenSearch Service データ同期を実行する必要があります。そうしないと、データに不整合が生じる可能性があります。 GitHub このプロジェクトでは、同期の実行方法の例を示します。Neptune を OpenSearch (https://github.com/awslabs/ amazon-neptune-tools export-neptune-to-elasticsearch /tree/master/) にエクスポートします。

> **Important**
>
> Amazon OpenSearch サービスと統合する場合、Neptune はElasticsearchバージョン7.1以降を必要とし、 OpenSearch 2.3、2.5およびfuture 互換性のあるOpensearchバージョンで動作します。

> **Note**
>
> エンジンリリース 1.3.0.0 以降、Amazon Neptune は Gremlin クエリと SPARQL クエリでのフルテキスト検索に Amazon OpenSearch サービスサーバーレスを使用することをサポートしています。

トピック

AWS CloudFormation テンプレートを使用して Neptune 間のレプリケーションを開始する OpenSearch

AWS CloudFormation リージョン固有のスタックを起動する

AWS CloudFormation 以下の各テンプレートは、 AWS 特定のリージョンにストリームコンシューマーアプリケーションインスタンスを作成します。 AWS CloudFormation コンソールを使用して対応するスタックを起動するには、 AWS 使用するリージョンに応じて、以下の表の「Launch Stack」ボタンのいずれかを選択します。

リージョン	ビュー	デザイナーで表示	起動する
米国東部 (バージニア北部)	表示	デザイナーで表示	
米国東部 (オハイオ)	表示	デザイナーで表示	
米国西部 (北カリフォルニア)	表示	デザイナーで表示	

リージョン	ビュー	デザイナーで表示	起動する
米国西部 (オレゴン)	表示	デザイナーで表示	Launch Stack
カナダ (中部)	表示	デザイナーで表示	Launch Stack
南米 (サンパウロ)	表示	デザイナーで表示	Launch Stack
欧州 (ストックホルム)	表示	デザイナーで表示	Launch Stack
欧州 (アイルランド)	表示	デザイナーで表示	Launch Stack
欧州 (ロンドン)	表示	デザイナーで表示	Launch Stack
欧州 (パリ)	表示	デザイナーで表示	Launch Stack
欧州 (フランクフルト)	表示	デザイナーで表示	Launch Stack
中東 (バーレーン)	表示	デザイナーで表示	Launch Stack
中東 (アラブ首長国連邦)	表示	デザイナーで表示	Launch Stack
イスラエル (テルアビブ)	表示	デザイナーで表示	Launch Stack
アフリカ (ケープタウン)	表示	デザイナーで表示	Launch Stack
アジアパシフィック (香港)	表示	デザイナーで表示	Launch Stack
アジアパシフィック (東京)	表示	デザイナーで表示	Launch Stack

リージョン	ビュー	デザイナーで表示	起動する
アジアパシフィック（ソウル）	表示	デザイナーで表示	Launch Stack
アジアパシフィック（シンガポール）	表示	デザイナーで表示	Launch Stack
アジアパシフィック（ムンバイ）	表示	デザイナーで表示	Launch Stack
中国 (北京)	表示	デザイナーで表示	Launch Stack
中国 (寧夏)	表示	デザイナーで表示	Launch Stack
AWS GovCloud (米国西部)	表示	デザイナーで表示	Launch Stack
AWS GovCloud (米国東部)	表示	デザイナーで表示	Launch Stack

[Create Stack (スタックの作成)] ページで、[Next (次へ)] を選択します。

OpenSearch作成している新しいスタックに関する詳細を追加します。

[Specify Stack Details] (スタック詳細の指定) ページには、フルテキスト検索の設定を制御するために使用できるプロパティとパラメータがあります。

スタック名 — AWS CloudFormation 作成する新しいスタックの名前。通常は、デフォルト値、`NeptuneStreamPoller` を使用できます。

[Parameters (パラメータ)] で、以下を指定します。

Streams コンシューマーが実行される VPC のネットワーク設定

- **`VPC`** – ポーリング Lambda 関数を実行する VPC の名前を指定します。
- **`List of Subnet IDs`** – ネットワークインターフェースが確立されるサブネット。Neptune クラスターに対応するサブネットを追加します。

- **List of Security Group Ids** – ソース Neptune DB クラスターへの書き込みインバウンドアクセスを許可するセキュリティグループの ID を指定します。
- **List of Route Table Ids** – これは、Neptune VPC に Amazon DynamoDB エンドポイントを作成するために必要です (まだエンドポイントを持っていない場合)。サブネットに関連付けられたルートテーブル ID のコンマ区切りリストを指定する必要があります。
- **Require to create Dynamo DB VPC Endpoint** – デフォルトが `true` のブール値です。この値を `false` に変更する必要があるのは、VPC に作成済みの DynamoDB エンドポイントがある場合のみです。
- **Require to create Monitoring VPC Endpoint** – デフォルトが `true` のブール値です。この値を `false` に変更する必要があるのは、VPC に作成済みのモニタリングエンドポイントがある場合のみです。

ストリームポーラー

- **Application Name** – 通常は、この設定をデフォルト (`NeptuneStream`) のままにしておくことができます。別の名前を使用する場合は、一意である必要があります。
- **Memory size for Lambda Poller** – Lambda ポーラー関数で使用可能なメモリサイズを設定するために使用されます。デフォルト値は 2,048 メガバイトです。
- **Lambda Runtime** – ストリームから項目を取得する Lambda 関数で使用される言語。これは `python3.9` または `java8` どちらにも設定できます。
- **S3 Bucket having Lambda code artifacts** – 別の S3 バケットからロードするカスタム Lambda ポーリング関数を使用している場合を除き、空白のままにしておきます。
- **S3 Key corresponding to Lambda Code artifacts** – カスタム Lambda ポーリング関数を使用している場合を除き、この空白のままにしておきます。
- **StartingCheckpoint** — ストリームポーラーの開始チェックポイント。デフォルトは `0:0` であり、Neptune ストリームの始めから開始することを意味します。
- **StreamPollerInitialState** - ポーラーの初期状態。デフォルトは `ENABLED` であり、スタック全体の作成が完了するとすぐにストリームの複製が開始されることを意味します。
- **Logging level for Lambda** – 通常は、デフォルト値、`INFO` のままにしておきます。
- **Managed Policies for Lambda Execution** – カスタム Lambda ポーリング関数を使用しない限り、通常は空白のままにしておきます。
- **Stream Records Handler** – Neptune ストリーム内のレコードにカスタムハンドラを使用している場合を除き、通常は空白のままにしておきます。

- **Maximum records Fetched from Stream** – このパラメータを使用して、パフォーマンスを調整できます。デフォルト (100) は、開始に適しています。最大許容値は 10,000 です。数値が大きいほど、ストリームからレコードを読み取るために必要なネットワークの呼び出しは少なくなりますが、レコードを処理するために必要なメモリは多くなります。
- **Max wait time between two Polls (in Seconds)** – Neptune ストリームを ポーリングするために Lambda ポーラーが呼び出される頻度を決定します。連続ポーリングの場合は、この値を 0 に設定します。最大値は 3,600 秒 (60 分) です。グラフデータの変化速度に応じて、デフォルト値 (60 秒) は開始に適しています。
- **Maximum Continuous polling period (in Seconds)** – Lambda ポーリング関数のタイムアウトを設定するために使用されます。これは、5 秒から 900 秒の間でなければなりません。デフォルト値 (600 秒) は、開始に適しています。
- **Step Function Fallback Period**— step-function-fallback-period ポーラーを待機するユニット数。その後、障害から回復するために Amazon CloudWatch Events を通じて step 関数が呼び出されます。デフォルト (5 分) は、開始に適しています。
- **Step Function Fallback Period Unit** – 直前の `Step Function Fallback Period` 測定に使用される時間単位 (分、時間、日)。通常は、デフォルト (分) で十分です。
- **Data replication scope**— ノードとエッジの両方を複製するか、ノードのみを複製するかを決定します OpenSearch (これは Gremlin エンジンのデータにのみ適用されます)。一般的なケースであれば、まずはデフォルト値 (All) を使用することをお勧めします。
- **Ignore OpenSearch missing document error**— OpenSearch でのドキュメントが見つからないというエラーを無視できるかどうかを判断するフラグ。ドキュメントの欠落エラーはまれに発生しますが、無視しない場合は手動による介入が必要になります。通常、デフォルト値 (`True`) は開始に適しています。
- **Enable Non-String Indexing** – 文字列の内容を持たないフィールドのインデックス作成を有効または無効にするフラグ。このフラグをに設定すると`true`、文字列以外のフィールドにインデックスが付けられ OpenSearch`false`、その場合は文字列フィールドのみがインデックスされます。デフォルトは `true` です。
- **Properties to exclude from being inserted into OpenSearch**— インデックスから除外するプロパティキーまたは述語キーをカンマで区切ったリスト。 OpenSearch この CFN パラメータ値を空白のままにすると、すべてのプロパティキーにインデックスが付けられます。
- **Datatypes to exclude from being inserted into OpenSearch**— インデックス作成から除外するプロパティまたは述語のデータ型をカンマで区切ったリスト。 OpenSearch この CFN パラメータ値を空白のままにすると、データ型に安全に変換できるすべてのプロパティ値がインデックス化されます。 OpenSearch

Neptune Stream

- **Endpoint of source Neptune Stream** – (必須) これは次の 2 つの形式のいずれかになります。
  - **https://*your DB cluster*:*port*/propertygraph/stream** (またはそのエイリアス https://*your DB cluster*:*port*/pg/stream)。
  - https://*your DB cluster*:*port*/sparql/stream
- **Neptune Query Engine** – Gremlin または SPARQL を選択します。
- **Is IAM Auth Enabled?** – Neptune DB クラスターで IAM 認証を使用している場合は、このパラメータを `true` に設定します。
- **Neptune Cluster Resource Id** – Neptune DB クラスターで IAM 認証を使用している場合は、このパラメータをクラスターのリソース ID に設定します。リソース ID はクラスター ID と同じではありません。正確には、28 文字の英数字が続く `cluster-` です。Neptune コンソールの [Cluster Details] (クラスターの詳細) で確認できます。

OpenSearch ターゲットクラスター

- **Endpoint for OpenSearch service**— (必須) VPC OpenSearch 内のサービスのエンドポイントを指定します。
- **Number of Shards for OpenSearch Index** – 通常、デフォルト値 (5) は開始に適しています。
- **Number of Replicas for OpenSearch Index** – 通常、デフォルト値 (1) は開始に適しています。
- **Geo Location Fields for Mapping** – 位置情報フィールドを使用している場合は、ここにプロパティキーをリストします。

アラーム

- **Require to create Cloud watch Alarm**— `true` CloudWatch 新しいスタックのアラームを作成する場合は、これをに設定します。
- **SNS Topic ARN for Cloudwatch Alarm Notifications**— CloudWatch アラーム通知を送信する必要がある SNS トピック ARN (アラームが有効な場合にのみ必要) 。
- **Email for Alarm Notifications** – アラーム通知の送信先の E メールアドレス (アラームが有効な場合のみ必要)。

アラーム通知の宛先には、SNS のみ、E メールのみ、または SNS と E メールの両方を追加できます。

テンプレートを実行します。 AWS CloudFormation

これで、次のように、Neptune ストリームコンシューマーアプリケーションインスタンスをプロビジョニングするプロセスを完了できます。

1. の AWS CloudFormation「スタックの詳細を指定」ページで、「次へ」を選択します。
2. [Options(オプション)] ページで、[Next(次へ)] を選択します。
3. [確認] ページで、 AWS CloudFormation によって IAM リソースが作成されることを確認する最初のチェックボックスをオンにします。新しいスタックの `CAPABILITY_AUTO_EXPAND` を確認する 2 つ目のチェックボックスをオンにします。

   > **Note**
   >
   > `CAPABILITY_AUTO_EXPAND` は、スタックの作成時に事前の確認なしにマクロが展開されることを明示的に確認します。ユーザーは、処理されたテンプレートから変更セットを作成することが多いため、実際にスタックを作成する前にマクロによって行われた変更を確認できます。詳細については、『 AWS CloudFormation CreateStackAPI リファレンス』のAWS CloudFormation API オペレーションを参照してください。

   次に [作成] を選択します。

既存の Neptune データベースでフルテキスト検索を有効にする

書き込みワークロードを一時停止できる場合

既存の Neptune データベースでフルテキスト検索を有効にする最良の方法は、書き込みワークロードを一時停止できれば、一般的に次の方法です。クローンを作成し、クラスターパラメータを使用してストリームを有効にし、すべてのインスタンスを再起動する必要があります。クローンの作成は比較的速いので、必要なダウンタイムは限られています。

必要な手順は次のとおりです。

1. データベース上のすべての書き込みワークロードを停止します。
2. データベースのストリームを有効にします (「Neptune ストリームの有効化」を参照)。

3. データベースのクローンを作成します (「Neptune でのデータベースクローニング」を参照)。
4. 書き込みワークロードを再開します。
5. github export-neptune-to-elasticsearchのツールを使用して、クローニングされたデータベースからドメインへの 1 回限りの同期を実行します。 OpenSearch
6. リージョン用のAWS CloudFormation テンプレートを使用して、元のデータベースから同期して継続的な更新を行います (テンプレートで設定を変更する必要はありません)。
7. クローニングされたデータベースと、 AWS CloudFormation ツール用に作成されたスタックを削除します。`export-neptune-to-elasticsearch`

書き込みワークロードを一時停止できない場合

データベースで書き込みワークロードを中断する余裕がない場合は、上記の推奨アプローチよりも少ないダウンタイムを必要とするアプローチを以下に示します。ただし、慎重に実行する必要があります。

1. データベースのストリームを有効にします (「Neptune ストリームの有効化」を参照)。
2. データベースのクローンを作成します (「Neptune でのデータベースクローニング」を参照)。
3. Streams API エンドポイントに対してこの種のコマンドを実行することによって、クローンされたデータベース上のストリームの最新の `eventID` を取得します (詳細については、「Neptune Streams REST API の呼び出し」を参照)。

   ```
   curl "https://(your neptune endpoint):(port)/(propertygraph or sparql)/stream?
   iteratorType=LATEST"
   ```

   レスポンス内の `lastEventId` オブジェクト内の `commitNum` および `opNum` フィールドの値を書き留めておきます。
4. github export-neptune-to-elasticsearchのツールを使用して、クローニングされたデータベースからドメインへの 1 回限りの同期を実行します。 OpenSearch
5. リージョン用のAWS CloudFormation テンプレートを使用して、元のデータベースから同期して継続的な更新を行います。

   スタックの作成時に次の変更を行います。スタックの詳細ページの [Parameters] セクションで、上記で記録した `commitNum` および `opNum` の値を使用して、`StartingCheckpoint` フィールドの値を *`CommitNum`*:*`opnum`* に設定します。

6. クローニングされたデータベースと、 AWS CloudFormation ツール用に作成されたスタックを削除します。`export-neptune-to-elasticsearch`

ストリームポーラーの更新

最新の Lambda アーティファクトでストリームポーラーを更新するには

最新の Lambda アーティファクトで次のようにストリームポーラーを更新できます。

1. で AWS Management Console、 AWS CloudFormation AWS CloudFormation メインの親スタックに移動して選択します。
2. スタックの[Update] (更新) オプションを選択します。
3. [Replace current template] (現在のテンプレートを置換) を選択します。
4. テンプレートソースで、Amazon S3 URL を選択し、次の S3 URL を入力します。

```
https://aws-neptune-customer-samples.s3.amazonaws.com/neptune-stream/
neptune_to_elastic_search.json
```

5. AWS CloudFormation パラメータを変更せずに [Next] を選択します。
6. [Update Stack] を選択します。

これで、スタックは Lambda アーティファクトを最新のアーティファクトで更新します。

ストリームポーラーを拡張してカスタムフィールドをサポートする

現在のストリームポーラーは、ブログ記事「Neptune Streams を使用してグラフの変化を捕捉する」で詳しく説明されているように、カスタムフィールドを処理するためのカスタムコードを記述するように簡単に拡張できます。

> **Note**
>
> にカスタムフィールドを追加するときは OpenSearch、必ず新しいフィールドを述部の内部オブジェクトとして追加してください (「」を参照Neptune フルテキスト検索データモデル)。

ストリームポーラープロセスの無効化と再有効化

> **Warning**
>
> ストリームポーラープロセスを無効にするときは注意してください。ストリームの有効期限を超えてプロセスを一時停止すると、データが失われる可能性があります。デフォルトの期間は 7 日間ですが、エンジンバージョン 1.2.0.0 以降では、カスタムストリームの有効期限を最大 90 日まで設定できます。

ストリームポーラープロセスの無効化 (一時停止)

1. AWS Management Console にサインインし、https://console.aws.amazon.com/events/ にある Amazon EventBridge コンソールを開きます。
2. ナビゲーションペインで、[Rules] を選択します。
3. AWS CloudFormation ストリームポーラーの設定に使用したテンプレートの [アプリケーション名] として指定した名前を含む名前のルールを選択します。
4. [無効化] を選択します。
5. https://console.aws.amazon.com/states/ で Step Functions コンソールを開きます。
6. ストリームポーラープロセスに対応するステップ実行関数を選択します。繰り返しますが、そのステップ関数の名前には、 AWS CloudFormation ストリームポーラーの設定に使用したテンプレートの [アプリケーション名] として指定した名前が含まれます。関数の実行ステータスでフィルタリングすると、実行中の関数だけを表示できます。
7. [Stop] (停止) を選択します。

ストリームポーラープロセスの再有効化

1. AWS Management Console にサインインし、https://console.aws.amazon.com/events/ にある Amazon EventBridge コンソールを開きます。
2. ナビゲーションペインで、[Rules] を選択します。
3. AWS CloudFormation ストリームポーラーの設定に使用したテンプレートの [アプリケーション名] として指定した名前を含む名前のルールを選択します。
4. [無効化] を選択します。指定したスケジュールされた間隔に基づくイベントルールが、ステップ関数の新しい実行をトリガーします。

OpenSearch サーバーレスへのレプリケーション

エンジンリリース 1.3.0.0 以降、Amazon Neptune は Gremlin クエリと SPARQL クエリでのフルテキスト検索において Amazon OpenSearch Service サーバーレスの使用をサポートしています。

OpenSearch サーバーレスにレプリケートする場合は、Lambda ストリームポーリング実行ロールを OpenSearch サーバーレスコレクションのデータアクセスポリシーに追加します。Lambda ストリームポーリング実行ロールの ARN の形式は次のとおりです。

```
arn:aws:iam::(account ID):role/stack-name-NeptuneOSReplication-
NeptuneStreamPollerExecu-(uuid)
```

詳細については、「Amazon OpenSearch Serverless のデータアクセスコントロール」を参照してください。

OpenSearch クラスターで詳細なアクセスコントロールを有効にした場合は、Neptune データベースでも IAM 認証を有効にする必要があります。

Neptune データベースへの接続に使用する IAM エンティティ (ユーザーまたはロール) には、Neptune と OpenSearch サーバーレスコレクションの両方に対するアクセス許可が必要です。つまり、ユーザーまたはロールには、次のような OpenSearch サーバーレスポリシーをアタッチする必要があります。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::(account ID):root"
      },
      "Action": "aoss:APIAccessAll",
      "Resource": "arn:aws:aoss:(region):(account ID):collection/(collection ID)"
    }
  ]
}
```

詳細については、「Amazon Neptune のカスタム IAM データアクセスポリシーステートメント」を参照してください。

きめ細かいアクセスコントロール (FGAC) が有効になっている OpenSearch クラスターからのクエリ

OpenSearch クラスターで詳細なアクセスコントロールを有効にしている場合、Neptune データベースでも IAM 認証を有効にする必要があります。

Neptune データベースへの接続に使用される IAM エンティティ (ユーザーまたはロール) には、Neptune と OpenSearch クラスターの両方に対するアクセス許可が必要です。つまり、ユーザーまたはロールには、次のような OpenSearch Service ポリシーがアタッチされている必要があります。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::account-id:root"
      },
      "Action": "es:*",
      "Resource": "arn:aws:es:region:account-id:es-resource-id/*"
    }
  ]
}
```

詳細については、「Amazon Neptune のカスタム IAM データアクセスポリシーステートメント」を参照してください。

Neptune フルテキスト検索クエリでの Apache Lucene クエリ構文

OpenSearch は query_string クエリに関して Apache Lucene 構文の使用をサポートしています。これは、クエリで複数のフィルタを渡す場合に特に便利です。

Neptune は、ネストされた構造を使用して、プロパティを OpenSearch ドキュメントに格納します (Neptune フルテキスト検索データモデル を参照)。Lucene 構文を使用する場合は、次のモデルのプロパティへのフルパスを使用する必要があります。

Gremlin の例を次に示します。

```
g.withSideEffect("Neptune#fts.endpoint", "es_endpoint")
```

```
  .withSideEffect("Neptune#fts.queryType", "query_string")
  .V()
  .has("*", "Neptune#fts predicates.name.value:\"Jane Austin\" AND entity_type:Book")
```

SPARQL の例を次に示します。

```
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
  neptune-fts:config neptune-fts:endpoint 'http://localhost:9200 (http://
localhost:9200/)' .
  neptune-fts:config neptune-fts:queryType 'query_string' .
  neptune-fts:config neptune-fts:query "predicates.\\*foaf\\*name.value:Ronak AND
 predicates.\\*foaf\\*surname.value:Sh*" .
  neptune-fts:config neptune-fts:field '*' .
  neptune-fts:config neptune-fts:return ?res .
}
```

OpenSearch データの Neptune データモデル

Amazon Neptune は、統一された JSON ドキュメント構造を使用して、SPARQL と Gremlin の両方のデータを OpenSearch Service に保存します。OpenSearch の各ドキュメントは 1 つのエンティティに対応し、そのエンティティに関連するすべての情報を格納します。Gremlin の場合、頂点とエッジはエンティティと見なされるため、対応する OpenSearch ドキュメントには頂点、ラベル、およびプロパティに関する情報があります。SPARQL の場合、サブジェクトはエンティティと見なすことができるため、対応する OpenSearch ドキュメントには、1 つのドキュメント内のすべての述語とオブジェクトのペアに関する情報があります。

> **Note**
>
> Neptune から OpenSearch へのレプリケーション実装では、文字列データのみが保存されます。ただし、他のデータ型を保存するように変更することはできます。

統一された JSON ドキュメント構造は次のようになります。

```
{
  "entity_id": "Vertex Id/Edge Id/Subject URI",
  "entity_type": [List of Labels/rdf:type object value],
  "document_type": "vertex/edge/rdf-resource"
```

```
  "predicates": {
    "Property name or predicate URI": [
      {
        "value": "Property Value or Object Value",
        "graph": "(Only for Sparql) Named Graph Quad is present"
        "language": "(Only for Sparql) rdf:langString"
      },
      {
        "value": "Property Value 2/ Object Value 2",
      }
    ]
  }
}
```

- `entity_id` – ドキュメントを表すエンティティ固有の ID。
  - SPARQL の場合、これはサブジェクトの URI です。
  - Gremlin の場合、これは `Vertex_ID` または `Edge_ID` です。
- `entity_type` – 頂点またはエッジの 1 つや複数のラベル、またはサブジェクトの 0 個以上の `rdf:type` 述語値を表します。
- `document_type` – 現在のドキュメントが頂点、エッジ、RDF リソースを表すかどうかを指定するために使用されます。
- `predicates` – Gremlin では、頂点またはエッジのプロパティと値が保存されます。SPARQL の場合、述語オブジェクトのペアを保存します。

  プロパティ名は、OpenSearch では `properties.name.value` の形式になります。クエリを実行するには、その形式で名前を付ける必要があります。
- `value` – Gremlin のプロパティ値、または SPARQL のオブジェクト値。
- `graph` – SPARQL の名前付きグラフ。
- `language` – SPARQL の `rdf:langString` リテラルの言語タグ。

サンプルの SPARQL OpenSearch ドキュメント

データ

```
@prefix dt:   <http://example.org/datatype#> .
@prefix ex:   <http://example.org/> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .
```

```
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

ex:simone   rdf:type     ex:Person                    ex:g1
ex:michael  rdf:type     ex:Person                    ex:g1
ex:simone   ex:likes     "spaghetti"                  ex:g1

ex:simone   ex:knows     ex:michael                   ex:g2   # Not stored in ES
ex:simone   ex:likes     "spaghetti"                  ex:g2
ex:simone   ex:status    "La vita è un sogno"@it      ex:g2

ex:simone   ex:age       "40"^^xsd:int                DG      # Not stored in ES
ex:simone   ex:dummy     "testData"^^dt:newDataType   DG
ex:simone   ex:hates     _:bnode                              # Not stored in ES
_:bnode     ex:means     "coding"                     DG      # Not stored in ES
```

ドキュメント

```
{
  "entity_id": "http://example.org/simone",
  "entity_type": ["http://example.org/Person"],
  "document_type": "rdf-resource"
  "predicates": {
    "http://example.org/likes": [
      {
        "value": "spaghetti",
        "graph": "http://example.org/g1"
      },
      {
        "value": "spaghetti",
        "graph": "http://example.org/g2"
      }
    ]
    "http://example.org/status": [
      {
        "value": "La vita è un sogno",
        "language": "it"       // Only present for rdf:langString
      }
    ]
  }
}
```

```
{
  "entity_id" : "http://example.org/michael",
```

```
  "entity_type" : ["http://example.org/Person"],
  "document_type": "rdf-resource"
}
```

サンプルの Gremlin OpenSearch ドキュメント

データ

```
# Vertex 1
simone   label    Person       <== Label
simone   likes    "spaghetti"  <== Property
simone   likes    "rice"       <== Property
simone   age      40           <== Property

# Vertex 2
michael  label    Person       <== Label

# Edge 1
simone  knows     michael      <== Edge
e1      updated  "2019-07-03"  <== Edge Property
e1      through  "company"     <== Edge Property
e1      since     10           <== Edge Property
```

ドキュメント

```
{
  "entity_id": "simone",
  "entity_type": ["Person"],
  "document_type": "vertex",
  "predicates": {
    "likes": [
      {
        "value": "spaghetti"
      },
      {
        "value": "rice"
      }
    ]
  }
}
```

```
{
```

```
  "entity_id" : "michael",
  "entity_type" : ["Person"],
  "document_type": "vertex"
}
```

```
{
  "entity_id": "e1",
  "entity_type": ["knows"],
  "document_type": "edge"
  "predicates": {
    "through": [
      {
        "value": "company"
      }
    ]
  }
}
```

Neptune フルテキスト検索パラメータ

Amazon Neptune は、以下のパラメータを使用して、Gremlin と SPARQL の両方でフルテキストの OpenSearch クエリを指定します。

- **`queryType`** – (必須) OpenSearch クエリのタイプ。(クエリタイプのリストについては、「OpenSearch ドキュメンテーション」を参照してください。) Neptune は以下の OpenSearch クエリタイプをサポートしています。
  - simple_query_string – 指定されたクエリ文字列に基づいてドキュメントを返し、限定されているが耐障害性の Lucene 構文のパーサーを使用します。これがデフォルトのクエリタイプです。

    このクエリは、単純な構文を使用して、指定されたクエリ文字列を特殊な演算子に基づいて単語に分割します。クエリは、一致するドキュメントを返す前に、各用語を個別に分析します。

    構文は `query_string` クエリよりも制限されていますが、`simple_query_string` クエリでは無効な構文のエラーは返されません。代わりに、クエリ文字列の無効な部分は無視されます。
  - match – `match`クエリは、ファジーマッチングのオプションを含む、フルテキスト検索を実行するための標準クエリです。
  - prefix – 指定されたフィールドに特定のプレフィックスを含むドキュメントを返します。
  - fuzzy – Levenshtein 編集距離で測定された、検索語に類似する語を含むドキュメントを返します。

編集距離とは、ある用語を別の用語に変換するのに必要な 1 文字の変更の数です。これらの変更には、次のものがあります。

- キャラクターの変更 (box から fox へ)。
- 文字を削除する (black から lack へ)。
- 文字を挿入する (sic から sick へ)。
- 隣接する 2 つの文字を転置する (act から cat へ)。

類似する用語を見つけるために、fuzzy クエリは、指定された編集距離内に検索用語のすべての可能なバリエーションと拡張のセットを作成し、それらのバリアントごとに完全一致を返します。

- term – 指定したフィールドの 1 つに、指定した語句と完全に一致するドキュメントを返します。

  `term` クエリを使用すると、価格、製品 ID、ユーザー名などの正確な値に基づいてドキュメントを検索できます。

  > **Warning**
  >
  > テキストフィールドにはクエリという用語を使用しないでください。デフォルトでは、OpenSearch は分析の一部としてテキストフィールドの値を変更するため、テキストフィールド値の完全一致を見つけることが困難になります。
  > テキストフィールド値を検索するには、代わりに match クエリを使用します。

- query_string – 厳密な構文 (Lucene 構文) のパーサーを使用して、指定されたクエリ文字列に基づいてドキュメントを返します。

  このクエリは、構文を使用して、AND や NOT などの演算子に基づいて、指定されたクエリ文字列を解析して分割します。クエリは、一致するドキュメントを返す前に、分割された各テキストを個別に分析します。

  `query_string` クエリを使用して、ワイルドカード文字、複数のフィールドにわたる検索などを含む複雑な検索を作成できます。汎用性がありますが、クエリは厳密であり、クエリ文字列に無効な構文が含まれている場合はエラーを返します。

> **Warning**
>
> 無効な構文に対してエラーが返されるため、検索ボックスに query_string クエリを使用することはお勧めしません。
> クエリ構文をサポートする必要がない場合は、match クエリの使用を検討してください。クエリ構文の機能が必要な場合は、simple_query_string クエリを使用します。これはあまり厳密ではありません。

- **field** – 検索を実行する対象の OpenSearch のフィールド。simple_query_string や query_string など、このフィールドの省略を許可する queryType では、これを省略できます。省略すると、検索はすべてのフィールドを対象に実行されます。Gremlin では、これは暗黙的です。

  simple_query_string および query_string など、複数のフィールドを指定できるクエリでは、複数を指定できます。

- **query** – (必須) OpenSearch に対して実行するクエリ。このフィールドの内容は、QueryType によって異なる場合があります。たとえば、異なる QueryType (Regexp など) は異なる構文を受け入れます。Gremlin では、query は暗黙的です。

- **maxResults** – 返される結果の最大数。デフォルトは index.max_result_window OpenSearch 設定であり、それ自体のデフォルトは 10,000 です。maxResults パラメータには、それよりも小さい任意の数を指定できます。

  > **Important**
  >
  > maxResults を OpenSearch の index.max_result_window の値より大きい値に設定して、index.max_result_window より多くの結果を取得しようとすると、OpenSearch は Result window is too large エラーで失敗します。ただし、Neptune はエラーを伝播せずにこれを正常に処理します。index.max_result_window 結果以上のものを取得しようとしている場合は、これを考慮してください。

- **minScore** – 検索結果が返されるために必要な最小限のスコア。結果スコアリングの説明については、「OpenSearch の関連ドキュメント」を参照してください。

- **batchSize** – Neptune は常にデータをバッチで取得します (デフォルトのバッチサイズは 100 です)。このパラメータを使用して、パフォーマンスを調整できます。バッチサイズは

`index.max_result_window` OpenSearch 設定を超えることはできません。デフォルトは 10,000 です。

- **`sortBy`** – OpenSearch が返した結果を以下のいずれかで並べ替えることができるオプションのパラメータ。
  - ドキュメント内の特定のフィールド –

    たとえば、SPARQL クエリでは、次のように指定できます。

    ```
    neptune-fts:config neptune-fts:sortBy foaf:name .
    ```

    同様の Gremlin クエリでは、次のように指定できます。

    ```
    .withSideEffect('Neptune#fts.sortBy', 'name')
    ```

  - ドキュメント内の特定のフィールド (*`long`*、*`double`*、など) –

    文字列以外のフィールドでソートする場合は、文字列フィールドと区別するために `.value` をフィールド名に追加する必要があります。

    たとえば、SPARQL クエリでは、次のように指定できます。

    ```
    neptune-fts:config neptune-fts:sortBy foaf:name.value .
    ```

    同様の Gremlin クエリでは、次のように指定できます。

    ```
    .withSideEffect('Neptune#fts.sortBy', 'name.value')
    ```

  - `score` – マッチスコアで並べ替えます (デフォルト)。

    `sortOrder` パラメータが存在しても `sortBy` が存在しない場合、結果は `sortOrder` によって指定された順序で `score` によって並べ替えられます。

  - `id` – ID (SPARQL URI、Gremlin Vertex またはエッジ ID) で並べ替えます。

    たとえば、SPARQL クエリでは、次のように指定できます。

    ```
    neptune-fts:config neptune-fts:sortBy 'Neptune#fts.entity_id' .
    ```

    同様の Gremlin クエリでは、次のように指定できます。

```
    .withSideEffect('Neptune#fts.sortBy', 'Neptune#fts.entity_id')
```

- `label` – ラベルで並べ替えます。

  たとえば、SPARQL クエリでは、次のように指定できます。

  ```
      neptune-fts:config neptune-fts:sortBy 'Neptune#fts.entity_type' .
  ```

  同様の Gremlin クエリでは、次のように指定できます。

  ```
      .withSideEffect('Neptune#fts.sortBy', 'Neptune#fts.entity_type')
  ```

- `doc_type` – ドキュメントタイプ (SPARQL または Gremlin) で並べ替えます。

  たとえば、SPARQL クエリでは、次のように指定できます。

  ```
      neptune-fts:config neptune-fts:sortBy 'Neptune#fts.document_type' .
  ```

  同様の Gremlin クエリでは、次のように指定できます。

  ```
      .withSideEffect('Neptune#fts.sortBy', 'Neptune#fts.document_type')
  ```

  デフォルトでは、OpenSearch の結果はソートされず、その順序は非決定的です。つまり、同じクエリが実行されるたびに異なる順序で項目を返す可能性があります。このため、結果セットが `max_result_window` より大きい場合、クエリが実行されるたびに、合計結果のかなり異なるサブセットが返される可能性があります。ただし、ソートすることで、異なる実行の結果をより直接的に比較することができます。

  `sortBy` に `sortOrder` パラメータが伴わない場合は、最大から最小までの降順 (DESC) が使用されます。

- **`sortOrder`** – OpenSearch の結果を最小から最大に並べ替えるか、最大から最小 (デフォルト) に並べ替えるかを指定できるオプションのパラメータ。

  - `ASC` – 昇順、最小から最大。
  - `DESC` – 降順、最大から最小。

このパラメータはデフォルト値で、sortBy パラメータが存在するものの sortOrder が指定されていない場合に使用されます。

sortBy も sortOrder も存在しない場合、OpenSearch の結果はデフォルトではソートされません。

Amazon Neptune での文字列以外の OpenSearch インデックス作成

Amazon Neptune での文字列以外の OpenSearch インデックス作成により、ストリームポーラーを使用して述語の非文字列値を OpenSearch へレプリケートできます。対応する OpenSearch マッピングまたはデータ型に安全に変換できるすべての述語値が OpenSearch へレプリケートされます。

新しいスタックで非文字列インデックスを有効にするには、AWS CloudFormation テンプレート内の Enable Non-String Indexing フラグを true に設定する必要があります。これはデフォルトの設定です。非文字列インデックスをサポートするように既存のスタックを更新するには、以下の 既存のスタックの更新する を参照してください。

> **Note**
>
> - **1.0.4.2** 以前のエンジンバージョンでは、非文字列インデックス作成を有効にしないことをおすすめします。
> - 文字列値を含むものもあれば、文字列以外の値を含むフィールド名を含むクエリもある、複数のフィールドに一致するフィールド名に正規表現を使用する OpenSearch クエリはエラーになります。Neptune のフルテキスト検索クエリがその型の場合にも同じことが起こります。
> - 文字列以外のフィールドでソートする場合は、文字列フィールドと区別するために「.value」をフィールド名に追加する必要があります。

目次

- SPARQL および Gremlin のデータ型を OpenSearch にマッピングする
- データマッピングの検証
- Neptune での非文字列 OpenSearch クエリの例
  - 1. 年齢が 30 より大きく、名前が「Si」で始まるすべての頂点を取得します。
  - 2. 年齢が 10 から 50 までのノードをすべて取得し、「Ronka」とあいまい一致の名前を取得する
  - 3. 過去 25 日以内のタイムスタンプを持つすべてのノードを取得する
  - 4. 特定の年と月以内のタイムスタンプを持つすべてのノードを取得する

既存の Neptune フルテキスト検索スタックを更新して、非文字列インデックスをサポートする

既に Neptune 全文検索を使用している場合は、非文字列インデックスをサポートするために必要な手順は次のとおりです。

1. ストリームポーラーの Lambda 関数を停止します。これにより、エクスポート中に新しい更新がコピーされなくなります。これを行うには、Lambda 関数を呼び出すクラウドイベントルールを無効にします。

   - AWS Management Console で CloudWatch に移動します。
   - [Rules] (ルール) を選択します。
   - Lambda ストリームポーラー名のルールを選択します。
   - [disable] (無効) を選択して、ルールを一時的に無効にします。

2. OpenSearch で現在の Neptune インデックスを削除します。次の `curl` クエリを使用して、OpenSearch クラスターから `amazon_neptune` インデックスを削除します。

   ```
   curl -X DELETE "your OpenSearch endpoint/amazon_neptune"
   ```

3. Neptune から OpenSearch へ 1 回限りのエクスポートを開始します。この時点で新しい OpenSearch スタックをセットアップするのが最善です。そうすれば、エクスポートを実行するポーラーに対して新しいアーティファクトが取得されるようになります。

   GitHub のここで一覧表示されている手順に従って、Neptune から OpenSearch へのデータの 1 回限りのエクスポートを開始します。

4. 既存のストリームポーラーの Lambda アーティファクトを更新します。Neptune のデータを OpenSearch へのエクスポートが正常に完了した後、次のステップを実行します。

   - AWS Management Console で、AWS CloudFormation に移動します。
   - メインの親 AWS CloudFormation スタックを選択します。
   - スタックの[Update] (更新) オプションを選択します。
   - [Replace current template] (現在のテンプレートを置換) オプションを選択します。
   - テンプレートソースの場合、Amazon S3 URL を選択します。
   - Amazon S3 URL に、次のように入力します。

     ```
     https://aws-neptune-customer-samples.s3.amazonaws.com/neptune-stream/
     neptune_to_elastic_search.json
     ```

   - AWS CloudFormation パラメータを何も変更せずに [Next] (次へ) を選択します。
   - [Update stack] (スタックの更新) を選択すると、AWS CloudFormation がストリームポーラーの Lambda コードアーティファクトを最新のアーティファクトに置き換えます。
5. ストリームポーラーを再度開始します。これを行うには、適切な CloudWatch ルールを有効にします。

   - AWS Management Console で CloudWatch に移動します。
   - [Rules] (ルール) を選択します。
   - Lambda ストリームポーラー名のルールを選択します。
   - [enable] (有効化) を選択します。

Neptune 全文検索でインデックス付けされるフィールドのフィルタリング

AWS CloudFormation テンプレートには 2 つのフィールドがあり、OpenSearch インデックス作成から除外するプロパティキー、述語キー、またはデータ型を指定できる詳細があります。

プロパティまたは述語名でフィルタリングする

OpenSearch インデックス作成から除外するプロパティキーまたは述語キーのコンマ区切りリストを提供するための `Properties to exclude from being inserted into Elastic Search Index` という名前のオプションの AWS CloudFormation テンプレートパラメータを使用できます。

たとえば、このパラメータを bob に設定したとします。

```
"Properties to exclude from being inserted into Elastic Search Index" : bob
```

その場合、次の Gremlin 更新クエリのストリームレコードは、インデックスに入るのではなく削除されます。

```
g.V("1").property("bob", "test")
```

同様に、パラメータを `http://my/example#bob` に設定することもできます。

```
"Properties to exclude from being inserted into Elastic Search Index" : http://my/
example#bob
```

その場合、次の SPARQL 更新クエリのストリームレコードは、インデックスに入るのではなく削除されます。

```
PREFIX ex: <http://my/example#>
INSERT DATA { ex:s1 ex:bob "test"}.
```

この AWS CloudFormation テンプレートパラメータに何も入力しなければ、除外されない限りすべてのプロパティキーにインデックスが付けられます。

プロパティまたは述語値型でフィルタリングする

OpenSearch インデックス作成から除外するプロパティまたは述語値データ型のコンマ区切りリストを指定するための `Datatypes to exclude from being inserted into Elastic Search Index` という名前のオプションの AWS CloudFormation テンプレートパラメータを使用できます。

SPARQL の場合、完全な XSD 型 URI を一覧表示する必要はなく、データ型トークンを一覧表示するだけで済みます。一覧表示できる有効なデータ型トークンは次のとおりです。

- `string`
- `boolean`
- `float`
- `double`
- `dateTime`
- `date`

- time
- byte
- short
- int
- long
- decimal
- integer
- nonNegativeInteger
- nonPositiveInteger
- negativeInteger
- unsignedByte
- unsignedShort
- unsignedInt
- unsignedLong

Gremlin の場合、一覧表示する有効なデータ型は次のとおりです。

- string
- date
- bool
- byte
- short
- int
- long
- float
- double

たとえば、このパラメータを string に設定したとします。

```
"Datatypes to exclude from being inserted into Elastic Search Index" : string
```

その場合、次の Gremlin 更新クエリのストリームレコードは、インデックスに入るのではなく削除されます。

```
g.V("1").property("myStringval", "testvalue")
```

同様に、パラメータを int に設定することもできます。

```
"Datatypes to exclude from being inserted into Elastic Search Index" : int
```

その場合、次の SPARQL 更新クエリのストリームレコードは、インデックスに入るのではなく削除されます。

```
PREFIX ex: <http://my/example#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
INSERT DATA { ex:s1 ex:bob "11"^^xsd:int }.
```

この AWS CloudFormation テンプレートパラメータに何も入力しなかった場合、値を安全に OpenSearch に変換できるすべてのプロパティにインデックスが付けられます。クエリ言語でサポートされていないリストされた型は無視されます。

SPARQL および Gremlin のデータ型を OpenSearch にマッピングする

OpenSearch での新しいデータ型マッピングは、プロパティまたはオブジェクトで使用されているデータ型に基づいて作成されます。一部のフィールドには異なる種類の値が含まれているため、初期マッピングではフィールドの一部の値が除外される場合があります。

Neptune データ型は OpenSearch データ型に次のようにマッピングされます。

SPARQL 型	Gremlin 型	OpenSearch の型
XSD:int	byte	long
XSD:unsignedInt	short	
XSD:integer	int	
XSD:byte	long	
XSD:unsignedByte		

SPARQL 型	Gremlin 型	OpenSearch の型
XSD:short XSD:unsignedShort XSD:long XSD:unsignedLong		
XSD:float XSD:double XSD:decimal	float double	double
XSD:boolean	bool	boolean
XSD:datetime XSD:date	date	date
XSD:string XSD:time	string	text
カスタムデータ型	該当なし	text
その他のデータ型	該当なし	text

たとえば、次の Gremlin 更新クエリでは、「newField」の新しいマッピングが OpenSearch に追加され、すなわち、`{ "type" : "double" }` になります。

```
g.V("1").property("newField" 10.5)
```

同様に、次の SPARQL 更新クエリでは、「ex:byte」の新しいマッピングが OpenSearch に追加され、すなわち、`{ "type" : "long" }` になります。

```
PREFIX ex: <http://my/example#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
```

```
INSERT DATA { ex:test ex:byte "123"^^xsd:byte }.
```

> **Note**
>
> ご覧のとおり、Neptune から OpenSearch にマッピングされた項目は、OpenSearch では Neptune にあるものとは別のデータ型になる可能性があります。ただし、OpenSearch には明示的なテキストフィールド、「datatype」があり、Neptune でその項目が持つデータ型を記録します。

データマッピングの検証

このプロセスを使用して、データは Neptune から OpenSearch にレプリケートされます。

- 問題のフィールドのマッピングが既に OpenSearch に存在する場合:
  - データ検証ルールを使用してデータを既存のマッピングに安全に変換できる場合は、フィールドを OpenSearch に格納します。
  - そうでない場合は、対応するストリーム更新レコードを削除します。
- 問題のフィールドに既存のマッピングがない場合は、Neptune のフィールドのデータ型に対応する OpenSearch データ型を見つけます。
  - データ検証ルールを使用してそのフィールドデータを OpenSearch データ型に安全に変換できる場合は、新しいマッピングおよびフィールドデータを OpenSearch に格納します。
  - そうでない場合は、対応するストリーム更新レコードを削除します。

値は Neptune 型ではなく同等の OpenSearch 型または既存の OpenSearch マッピングに対して検証されます。たとえば、`"123"^^xsd:int` の値 `"123"` の検証が `int` 型ではなく `long` 型に対して行われます。

Neptune はすべてのデータを OpenSearch にレプリケートしようとしますが、OpenSearch のデータ型が Neptune のものとは全く異なっている場合があり、そのような場合、レコードは OpenSearch でインデックス化されるのではなく、スキップされます。

たとえば、Neptune では、1 つのプロパティが異なる型の複数の値を持つことがありますが、OpenSearch では、フィールドは、インデックス全体で同じ型である必要があります。

デバッグログを有効にすると、Neptune から OpenSearch へのエクスポート中に削除されたレコードを表示できます。デバッグログエントリの例を次に示します。

```
Dropping Record : Data type not a valid Gremlin type
<Record>
```

データ型は、次のように検証されます。

- **`text`** — Neptune のすべての値は、OpenSearch のテキストに安全にマッピングできます。
- **`long`** — 次の Neptune データ型のルールは、OpenSearch マッピング型が long の場合に当てはまります (以下の例では、"`testLong`" は `long` マッピング型を持つとみなされています) 。
  - `boolean` — 無効で、変換できない、対応するストリーム更新レコードは削除されます。

    無効な Gremlin の例は次のとおりです。

    ```
    "testLong" : true.
    "testLong" : false.
    ```

    無効な SPARQL の例は次のとおりです。

    ```
    ":testLong" : "true"^^xsd:boolean
    ":testLong" : "false"^^xsd:boolean
    ```

  - `datetime` — 無効で、変換できない、対応するストリーム更新レコードは削除されます。

    無効な Gremlin の例は次のとおりです。

    ```
    ":testLong" :  datetime('2018-11-04T00:00:00').
    ```

    無効な SPARQL の例は次のとおりです。

    ```
    ":testLong" : "2016-01-01"^^xsd:date
    ```

  - `float`、`double`、または `decimal` — Neptune での値が 64 ビットに収まる整数である場合、その値は有効であり、長整数として OpenSearch に格納されますが、分数部を持つ場合、または `NaN` あるいは `INF` である場合、または、9,223,372,036,854,775,807 より大きいか、-9,223,372,036,854,775,808 より小さい場合、その値は無効であり、対応するストリーム更新レコードは削除されます。

    有効な Gremlin の例は次のとおりです。

    ```
    "testLong" :  145.0.
    ```

```
":testLong" :  123
":testLong" :  -9223372036854775807
```

有効な SPARQL の例は次のとおりです。

```
":testLong" : "145.0"^^xsd:float
":testLong" :  145.0
":testLong" : "145.0"^^xsd:double
":testLong" : "145.0"^^xsd:decimal
":testLong" : "-9223372036854775807"
```

無効な Gremlin の例は次のとおりです。

```
"testLong" :  123.45
":testLong" :  9223372036854775900
```

無効な SPARQL の例は次のとおりです。

```
":testLong" :  123.45
":testLong" :  9223372036854775900
":testLong" : "123.45"^^xsd:float
":testLong" : "123.45"^^xsd:double
":testLong" : "123.45"^^xsd:decimal
```

- `string` — Neptune での値が 64 ビット整数に含まれる可能性がある整数の文字列表現である場合、その値は有効であり、OpenSearch で `long` に変換されます。他の文字列値は `long` マッピングには無効であり、対応するストリーム更新レコードは削除されます。

  有効な Gremlin の例は次のとおりです。

  ```
  "testLong" :  "123".
  ":testLong" :  "145.0"
  ":testLong" :  "-9223372036854775807"
  ```

  有効な SPARQL の例は次のとおりです。

  ```
  ":testLong" : "145.0"^^xsd:string
  ":testLong" : "-9223372036854775807"^^xsd:string
  ```

  無効な Gremlin の例は次のとおりです。

```
"testLong" :  "123.45"
":testLong" :  "9223372036854775900"
":testLong" :  "abc"
```

無効な SPARQL の例は次のとおりです。

```
":testLong" : "123.45"^^xsd:string
":testLong" : "abc"
":testLong" : "9223372036854775900"^^xsd:string
```

- **`double`** — OpenSearch マッピング型が `double` の場合、以下の規則が適用されます (ここで「testDouble」フィールドは、OpenSearch で `double` のマッピングを持つとみなされます)。
  - `boolean` — 無効で、変換できない、対応するストリーム更新レコードは削除されます。

    無効な Gremlin の例は次のとおりです。

    ```
    "testDouble" : true.
    "testDouble" : false.
    ```

    無効な SPARQL の例は次のとおりです。

    ```
    ":testDouble" : "true"^^xsd:boolean
    ":testDouble" : "false"^^xsd:boolean
    ```

  - `datetime` — 無効で、変換できない、対応するストリーム更新レコードは削除されます。

    無効な Gremlin の例は次のとおりです。

    ```
    ":testDouble" :  datetime('2018-11-04T00:00:00').
    ```

    無効な SPARQL の例は次のとおりです。

    ```
    ":testDouble" : "2016-01-01"^^xsd:date
    ```

  - 浮動小数点型 `NaN` または `INF` — SPARQL の値が浮動小数点 `NaN` または `INF` の場合、これは有効ではなく、対応するストリーム更新レコードは削除されます。

    無効な SPARQL の例は次のとおりです。

```
"  :testDouble" : "NaN"^^xsd:float
  ":testDouble" : "NaN"^^double
  ":testDouble" : "INF"^^double
  ":testDouble" : "-INF"^^double
```

- 数値または数値文字列 — Neptune での値が、double として安全に表現できる他の数値または数値文字列表現である場合、それは有効であり、OpenSearch で double に変換されます。その他の文字列値は、OpenSearch double マッピングでは無効であり、対応するストリーム更新レコードは削除されます。

  有効な Gremlin の例は次のとおりです。

  ```
  "testDouble" :  123
  ":testDouble" :  "123"
  ":testDouble" :  145.67
  ":testDouble" :  "145.67"
  ```

  有効な SPARQL の例は次のとおりです。

  ```
  ":testDouble" :  123.45
  ":testDouble" :  145.0
  ":testDouble" : "123.45"^^xsd:float
  ":testDouble" : "123.45"^^xsd:double
  ":testDouble" : "123.45"^^xsd:decimal
  ":testDouble" : "123.45"^^xsd:string
  ```

  無効な Gremlin の例は次のとおりです。

  ```
  ":testDouble" :  "abc"
  ```

  無効な SPARQL の例は次のとおりです。

  ```
  ":testDouble" : "abc"
  ```

- **date** — OpenSearch マッピング型が date である場合、Neptune の date および dateTime 値は有効であり、dateTime 形式に正常に解析できる任意の文字列値も同様です。

  Gremlin または SPARQL の有効な例は次のとおりです。

```
Date(2016-01-01)
"2016-01-01" "
2003-09-25T10:49:41"
"2003-09-25T10:49"
"2003-09-25T10"
"20030925T104941-0300"
"20030925T104941"
"2003-Sep-25" "
Sep-25-2003"
"2003.Sep.25"
"2003/09/25"
"2003 Sep 25" "
Wed, July 10, '96"
"Tuesday, April 12, 1952 AD 3:30:42pm PST"
"123"
"-123"
"0"
"-0"
"123.00"
"-123.00"
```

無効な例は次のとおりです。

```
123.45
True
"abc"
```

Neptune での非文字列 OpenSearch クエリの例

Neptune は現在、OpenSearch 範囲クエリを直接はサポートしていません。ただし、次のサンプルクエリに示すように、Lucene 構文と query-type="query_string" を使用して同じ効果を得ることができます。

1. 年齢が 30 より大きく、名前が「Si」で始まるすべての頂点を取得します。

Gremlin の場合:

```
g.withSideEffect('Neptune#fts.endpoint', 'http://your-es-endpoint')
 .withSideEffect("Neptune#fts.queryType", "query_string")
```

```
  .V().has('*', 'Neptune#fts predicates.age.value:>30 && predicates.name.value:Si*');
```

SPARQL の場合:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://localhost:9200' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:query "predicates.\\*foaf\\*age.value:>30 AND
 predicates.\\*foaf\\*name.value:Si*" .
    neptune-fts:config neptune-fts:field '*' .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

ここで、簡潔にするために、"*foaf*age は完全な URI の代わりに使用されます。この正規表現では、URI に foaf および age の両方があるすべてのフィールドが取得されます。

2. 年齢が 10 から 50 までのノードをすべて取得し、「Ronka」とあいまい一致の名前を取得する

Gremlin の場合:

```
g.withSideEffect('Neptune#fts.endpoint', 'http://your-es-endpoint')
 .withSideEffect("Neptune#fts.queryType", "query_string")
 .V().has('*', 'Neptune#fts predicates.age.value:[10 TO 50] AND
 predicates.name.value:Ronka~');
```

SPARQL の場合:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://localhost:9200' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:query "predicates.\\*foaf\\*age.value:[10 TO 50] AND
 predicates.\\*foaf\\*name.value:Ronka~" .
    neptune-fts:config neptune-fts:field '*' .
```

```
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

3. 過去 25 日以内のタイムスタンプを持つすべてのノードを取得する

Gremlin の場合:

```
g.withSideEffect('Neptune#fts.endpoint', 'http://your-es-endpoint')
 .withSideEffect("Neptune#fts.queryType", "query_string")
 .V().has('*', 'Neptune#fts predicates.timestamp.value:>now-25d');
```

SPARQL の場合:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://localhost:9200' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:query "predicates.\\*foaf\
\*timestamp.value:>now-25d~" .
    neptune-fts:config neptune-fts:field '*' .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

4. 特定の年と月以内のタイムスタンプを持つすべてのノードを取得する

Gremlin では、Lucene構文の日付の数式、2020年 12 月は次のようになります。

```
g.withSideEffect('Neptune#fts.endpoint', 'http://your-es-endpoint')
 .withSideEffect("Neptune#fts.queryType", "query_string")
 .V().has('*', 'Neptune#fts predicates.timestamp.value:>2020-12');
```

グレムリンの代替案:

```
g.withSideEffect('Neptune#fts.endpoint', 'http://your-es-endpoint')
 .withSideEffect("Neptune#fts.queryType", "query_string")
 .V().has('*', 'Neptune#fts predicates.timestamp.value:[2020-12 TO 2021-01]');
```

Amazon Neptune でのフルテキスト検索クエリの実行

フルテキスト検索を含むクエリでは、Neptune は、フルテキスト検索呼び出しを、クエリの他の部分より前となる、最初に配置しようとします。これにより、OpenSearch への呼び出し回数が減り、ほとんどの場合、パフォーマンスが大幅に向上します。しかし、これは決して厳格なルールではありません。たとえば、次のような状況があります。PatternNode または UnionNode がフルテキスト検索コールの前に表示されることがあります。

次の Gremlin クエリを 100,000 個の Person インスタンスを含むデータベースに対して行うとします。

```
g.withSideEffect('Neptune#fts.endpoint', 'your-es-endpoint-URL')
 .hasLabel('Person')
 .has('name', 'Neptune#fts marcello~');
```

このクエリがステップが表示される順序で実行された場合、100,000 のソリューションが OpenSearch に流れ、大量の OpenSearch 呼び出しが発生します。実際、Neptune は最初に OpenSearch を呼び出し、結果を Neptune 結果と結合します。ほとんどの場合、これは元の順序でクエリを実行するよりもはるかに高速です。

noReordering クエリヒントを使用すると、このクエリーステップ実行の順序変更を防ぐことができます。

```
g.withSideEffect('Neptune#fts.endpoint', 'your-es-endpoint-URL')
 .withSideEffect('Neptune#noReordering', true)
 .hasLabel('Person')
 .has('name', 'Neptune#fts marcello~');
```

この 2 番目のケースでは、.hasLabel ステップが最初に実行され、.has('name', 'Neptune#fts marcello~') ステップは 2 番目に実行されます。

別の例として、同じ種類のデータに対する SPARQL クエリを考えてみます。

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT ?person WHERE {
  ?person rdf:type foaf:Person .
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint.com' .
```

```
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:query 'mike' .
    neptune-fts:config neptune-fts:return ?person .
  }
}
```

ここでも、Neptune がクエリの SERVICE 部分を最初に実行してから、 Person データと結果を結合します。joinOrder クエリヒントを使用して、この動作を抑制できます。

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
SELECT ?person WHERE {
  hint:Query hint:joinOrder "Ordered" .
  ?person rdf:type foaf:Person .
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint.com' .
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:query 'mike' .
    neptune-fts:config neptune-fts:return ?person .
  }
}
```

ここでも、2 番目のクエリでは、パーツはクエリに表示される順序で実行されます。

Neptune のフルテキスト検索を使用した SPARQL クエリの例

Amazon Neptune でフルテキスト検索を使用した SPARQL クエリの例は次のとおりです。

SPARQL の match クエリの例

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint.com' .
    neptune-fts:config neptune-fts:queryType 'match' .
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:query 'michael' .
    neptune-fts:config neptune-fts:return ?res .
  }
```

```
}
```

SPARQL の prefix クエリの例

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint.com' .
    neptune-fts:config neptune-fts:queryType 'prefix' .
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:query 'mich' .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

SPARQL の fuzzy クエリの例

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint.com' .
    neptune-fts:config neptune-fts:queryType 'fuzzy' .
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:query 'mikael' .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

SPARQL の term クエリの例

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint.com' .
    neptune-fts:config neptune-fts:queryType 'term' .
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:query 'Dr. Kunal' .
    neptune-fts:config neptune-fts:return ?res .
```

```
  }
}
```

SPARQL の query_string クエリの例

このクエリは、複数のフィールドを指定します。

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint.com' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:query 'mikael~ OR rondelli' .
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:field foaf:surname .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

SPARQL の simple_query_string クエリの例

次のクエリでは、ワイルドカード ('*') 文字を使用してフィールドを指定します。

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint.com' .
    neptune-fts:config neptune-fts:queryType 'simple_query_string' .
    neptune-fts:config neptune-fts:query 'mikael~ | rondelli' .
    neptune-fts:config neptune-fts:field '*' .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

SPARQL 文字列フィールドによる並べ替えのクエリの例

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
```

```
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:query 'mikael~ | rondelli' .
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:sortOrder 'asc' .
    neptune-fts:config neptune-fts:sortBy foaf:name .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

SPARQL 非文字列フィールドによる並べ替えのクエリの例

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:query 'mikael~ | rondelli' .
    neptune-fts:config neptune-fts:field foaf:name.value .
    neptune-fts:config neptune-fts:sortOrder 'asc' .
    neptune-fts:config neptune-fts:sortBy dc:date.value .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

SPARQL ID によるソートのクエリの例

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:query 'mikael~ | rondelli' .
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:sortOrder 'asc' .
    neptune-fts:config neptune-fts:sortBy 'Neptune#fts.entity_id' .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

SPARQL ラベル によるソートのクエリの例

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:query 'mikael~ | rondelli' .
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:sortOrder 'asc' .
    neptune-fts:config neptune-fts:sortBy 'Neptune#fts.entity_type' .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

SPARQL doc_type によるソートのクエリの例

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:query 'mikael~ | rondelli' .
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:sortOrder 'asc' .
    neptune-fts:config neptune-fts:sortBy 'Neptune#fts.document_type' .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

SPARQL で Lucene 構文を使用する例

Lucene 構文は OpenSearch の `query_string` クエリでのみサポートされています。

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
```

```
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:query 'predicates.\\foaf\\name.value:micheal AND
 predicates.\\foaf\\surname.value:sh' .
    neptune-fts:config neptune-fts:field '' .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

Gremlin クエリでの Neptune フルテキスト検索の使用

NeptuneSearchStep は、Neptune ステップに変換されない Gremlin トラバーサル部分に対してフルテキスト検索クエリを有効にします。たとえば、次のようなクエリを考えてみます。

```
g.withSideEffect("Neptune#fts.endpoint", "your-es-endpoint-URL")
  .V()
      .tail(100)
      .has("name", "Neptune#fts mark*")            <== # Limit the search on name
```

このクエリは、Neptune で次の最適化されたトラバーサルに変換されます。

```
Neptune steps:
[
    NeptuneGraphQueryStep(Vertex) {
        JoinGroupNode {
            PatternNode[(?1, <~label>, ?2, <~>) . project distinct ?1 .],
 {estimatedCardinality=INFINITY}
        }, annotations={path=[Vertex(?1):GraphStep], maxVarId=4}
    },
    NeptuneTraverserConverterStep
]
+ not converted into Neptune steps: [NeptuneTailGlobalStep(100),
 NeptuneTinkerpopTraverserConverterStep, NeptuneSearchStep {
    JoinGroupNode {
        SearchNode[(idVar=?3, query=mark*, field=name) . project ask .],
 {endpoint=your-OpenSearch-endpoint-URL}
    }
    JoinGroupNode {
        SearchNode[(idVar=?3, query=mark*, field=name) . project ask .],
 {endpoint=your-OpenSearch-endpoint-URL}
    }
}]
```

以下に、エアルートデータに対する Gremlin クエリの例を示します。

Gremlin の基本的な大文字と小文字を区別しない **match** クエリ

```
g.withSideEffect("Neptune#fts.endpoint",
                 "your-OpenSearch-endpoint-URL")
  .withSideEffect('Neptune#fts.queryType', 'match')
  .V().has("city","Neptune#fts dallas")

==>v[186]
==>v[8]
```

Gremlin **match** クエリ

```
g.withSideEffect("Neptune#fts.endpoint",
                 "your-OpenSearch-endpoint-URL")
  .withSideEffect('Neptune#fts.queryType', 'match')
  .V().has("city","Neptune#fts southampton")
     .local(values('code','city').fold())
     .limit(5)

==>[SOU, Southampton]
```

Gremlin **fuzzy** クエリ

```
g.withSideEffect("Neptune#fts.endpoint",
                 "your-OpenSearch-endpoint-URL")
  .V().has("city","Neptune#fts allas~").values('city').limit(5)

==>Dallas
==>Dallas
==>Walla Walla
==>Velas
==>Altai
```

Gremlin **query_string** fuzzy クエリ

```
g.withSideEffect("Neptune#fts.endpoint",
                 "your-OpenSearch-endpoint-URL")
  .withSideEffect('Neptune#fts.queryType', 'query_string')
```

```
  .V().has("city","Neptune#fts allas~").values('city').limit(5)

==>Dallas
==>Dallas
```

Gremlin **query_string** 正規表現クエリ

```
g.withSideEffect("Neptune#fts.endpoint",
                 "your-OpenSearch-endpoint-URL")
  .withSideEffect('Neptune#fts.queryType', 'query_string')
  .V().has("city","Neptune#fts /[dp]allas/").values('city').limit(5)

==>Dallas
==>Dallas
```

Gremlin ハイブリッドクエリ

このクエリでは、Neptune 内部インデックスと OpenSearch インデックスが同じクエリで使用されます。

```
g.withSideEffect("Neptune#fts.endpoint",
                 "your-OpenSearch-endpoint-URL")
  .V().has("region","GB-ENG")
      .has('city','Neptune#fts L*')
      .values('city')
      .dedup()
      .limit(10)

==>London
==>Leeds
==>Liverpool
==>Land's End
```

シンプルな Gremlin フルテキスト検索の例

```
g.withSideEffect("Neptune#fts.endpoint",
                 "your-OpenSearch-endpoint-URL")
  .V().has('desc','Neptune#fts regional municipal')
      .local(values('code','desc').fold())
      .limit(100)
```

```
==>[HYA, Barnstable Municipal Boardman Polando Field]
==>[SPS, Sheppard Air Force Base-Wichita Falls Municipal Airport]
==>[ABR, Aberdeen Regional Airport]
==>[SLK, Adirondack Regional Airport]
==>[BFD, Bradford Regional Airport]
==>[EAR, Kearney Regional Airport]
==>[ROT, Rotorua Regional Airport]
==>[YHD, Dryden Regional Airport]
==>[TEX, Telluride Regional Airport]
==>[WOL, Illawarra Regional Airport]
==>[TUP, Tupelo Regional Airport]
==>[COU, Columbia Regional Airport]
==>[MHK, Manhattan Regional Airport]
==>[BJI, Bemidji Regional Airport]
==>[HAS, Hail Regional Airport]
==>[ALO, Waterloo Regional Airport]
==>[SHV, Shreveport Regional Airport]
==>[ABI, Abilene Regional Airport]
==>[GIZ, Jizan Regional Airport]
==>[USA, Concord Regional Airport]
==>[JMS, Jamestown Regional Airport]
==>[COS, City of Colorado Springs Municipal Airport]
==>[PKB, Mid Ohio Valley Regional Airport]
```

'+' と '-' 演算子で **query_string** を使用した Gremlin クエリ

query_string クエリタイプはデフォルトの simple_query_string タイプのように寛容ではありませんが、より正確なクエリが可能です。以下の最初のクエリは query_string を使用し、2 番目のクエリはデフォルト simple_query_stringを使用します。

```
g.withSideEffect("Neptune#fts.endpoint",
                 "your-OpenSearch-endpoint-URL")
  .withSideEffect('Neptune#fts.queryType', 'query_string')
 . V().has('desc','Neptune#fts +London -(Stansted|Gatwick)')
      .local(values('code','desc').fold())
      .limit(10)

==>[LHR, London Heathrow]
==>[YXU, London Airport]
==>[LTN, London Luton Airport]
==>[SEN, London Southend Airport]
==>[LCY, London City Airport]
```

以下の例では、simple_query_string は '+' と '-' 演算子を静かに無視しています。

```
g.withSideEffect("Neptune#fts.endpoint",
                 "your-OpenSearch-endpoint-URL")
  .V().has('desc','Neptune#fts +London -(Stansted|Gatwick)')
      .local(values('code','desc').fold())
      .limit(10)

==>[LHR, London Heathrow]
==>[YXU, London Airport]
==>[LGW, London Gatwick]
==>[STN, London Stansted Airport]
==>[LTN, London Luton Airport]
==>[SEN, London Southend Airport]
==>[LCY, London City Airport]
==>[SKG, Thessaloniki Macedonia International Airport]
==>[ADB, Adnan Menderes International Airport]
==>[BTV, Burlington International Airport]
```

```
g.withSideEffect("Neptune#fts.endpoint",
                 "your-OpenSearch-endpoint-URL")
  .withSideEffect('Neptune#fts.queryType', 'query_string')
  .V().has('desc','Neptune#fts +(regional|municipal) -(international|bradford)')
      .local(values('code','desc').fold())
      .limit(10)

==>[CZH, Corozal Municipal Airport]
==>[MMU, Morristown Municipal Airport]
==>[YBR, Brandon Municipal Airport]
==>[RDD, Redding Municipal Airport]
==>[VIS, Visalia Municipal Airport]
==>[AIA, Alliance Municipal Airport]
==>[CDR, Chadron Municipal Airport]
==>[CVN, Clovis Municipal Airport]
==>[SDY, Sidney Richland Municipal Airport]
==>[SGU, St George Municipal Airport]
```

AND および OR 演算子を使用した Gremlin query_string クエリ

```
g.withSideEffect("Neptune#fts.endpoint",
                 "your-OpenSearch-endpoint-URL")
  .withSideEffect('Neptune#fts.queryType', 'query_string')
```

```
  .V().has('desc','Neptune#fts (St AND George) OR (St AND Augustin)')
      .local(values('code','desc').fold())
      .limit(10)

==>[YIF, St Augustin Airport]
==>[STG, St George Airport]
==>[SGO, St George Airport]
==>[SGU, St George Municipal Airport]
```

Gremlin **term** クエリ

```
g.withSideEffect("Neptune#fts.endpoint",
                 "your-OpenSearch-endpoint-URL")
  .withSideEffect('Neptune#fts.queryType', 'term')
  .V().has("SKU","Neptune#fts ABC123DEF9")
      .local(values('code','city').fold())
      .limit(5)

==>[AUS, Austin]
```

Gremlin **prefix** クエリ

```
g.withSideEffect("Neptune#fts.endpoint",
                 "your-OpenSearch-endpoint-URL")
  .withSideEffect('Neptune#fts.queryType', 'prefix')
  .V().has("icao","Neptune#fts ka")
      .local(values('code','icao','city').fold())
      .limit(5)

==>[AZO, KAZO, Kalamazoo]
==>[APN, KAPN, Alpena]
==>[ACK, KACK, Nantucket]
==>[ALO, KALO, Waterloo]
==>[ABI, KABI, Abilene]
```

Neptune Gremlinでの Lucene 構文の使用

Neptune Gremlin では、Lucene クエリ構文を使用して非常に強力なクエリを作成することもできます。Lucene 構文は OpenSearch の `query_string` クエリでのみサポートされていることに注意してください。

次のデータを想定します。

```
g.addV("person")
        .property(T.id, "p1")
        .property("name", "simone")
        .property("surname", "rondelli")

g.addV("person")
        .property(T.id, "p2")
        .property("name", "simone")
        .property("surname", "sengupta")

g.addV("developer")
        .property(T.id, "p3")
        .property("name", "simone")
        .property("surname", "rondelli")
```

queryType が query_string のときに呼び出される Lucene 構文を使用すると、名前と姓で以下のようにこのデータを検索できます。

```
g.withSideEffect("Neptune#fts.endpoint", "es_endpoint")
    .withSideEffect("Neptune#fts.queryType", "query_string")
    .V()
    .has("*", "Neptune#fts predicates.name.value:simone AND
 predicates.surname.value:rondelli")

==> v[p1], v[p3]
```

上記の has( ) ステップでは、フィールドが "*" に置き換えられていることに注意してください)。実際には、そこに配置された値は、クエリ内でアクセスするフィールドによって上書きされます。データモデルが構造化されているため、predicates.name.value, を使用して名前フィールドにアクセスします。

次のように名前、姓、ラベルで検索できます。

```
g.withSideEffect("Neptune#fts.endpoint", getEsEndpoint())
    .withSideEffect("Neptune#fts.queryType", "query_string")
    .V()
    .has("*", "Neptune#fts predicates.name.value:simone AND
 predicates.surname.value:rondelli AND entity_type:person")
```

```
==> v[p1]
```

データモデルが構造化されているため、再び entity_type を使用してラベルにアクセスします。

ネスト条件を含めることもできます。

```
g.withSideEffect("Neptune#fts.endpoint", getEsEndpoint())
    .withSideEffect("Neptune#fts.queryType", "query_string")
    .V()
    .has("*", "Neptune#fts (predicates.name.value:simone AND
 predicates.surname.value:rondelli AND entity_type:person) OR
 predicates.surname.value:sengupta")

==> v[p1], v[p2]
```

Modern TinkerPop グラフを挿入する

```
g.addV('person').property(T.id, '1').property('name', 'marko').property('age', 29)
 .addV('personr').property(T.id, '2').property('name', 'vadas').property('age', 27)
 .addV('software').property(T.id, '3').property('name', 'lop').property('lang', 'java')
 .addV('person').property(T.id, '4').property('name', 'josh').property('age', 32)
 .addV('software').property(T.id, '5').property('name', 'ripple').property('lang',
 'java')
 .addV('person').property(T.id, '6').property('name', 'peter').property('age', 35)

g.V('1').as('a').V('2').as('b').addE('knows').from('a').to('b').property('weight',
 0.5f).property(T.id, '7')
 .V('1').as('a').V('3').as('b').addE('created').from('a').to('b').property('weight',
 0.4f).property(T.id, '9')
 .V('4').as('a').V('3').as('b').addE('created').from('a').to('b').property('weight',
 0.4f).property(T.id, '11')
 .V('4').as('a').V('5').as('b').addE('created').from('a').to('b').property('weight',
 1.0f).property(T.id, '10')
 .V('6').as('a').V('3').as('b').addE('created').from('a').to('b').property('weight',
 0.2f).property(T.id, '12')
 .V('1').as('a').V('4').as('b').addE('knows').from('a').to('b').property('weight',
 1.0f).property(T.id, '8')
```

文字列フィールド値で並べ替える例

```
g.withSideEffect("Neptune#fts.endpoint", "your-OpenSearch-endpoint-URL")
```

```
  .withSideEffect('Neptune#fts.queryType', 'query_string')
  .withSideEffect('Neptune#fts.sortOrder', 'asc')
  .withSideEffect('Neptune#fts.sortBy', 'name')
  .V().has('name', 'Neptune#fts marko OR vadas OR ripple')
```

非文字列フィールド値で並べ替える例

```
g.withSideEffect("Neptune#fts.endpoint", "your-OpenSearch-endpoint-URL")
  .withSideEffect('Neptune#fts.queryType', 'query_string')
  .withSideEffect('Neptune#fts.sortOrder', 'asc')
  .withSideEffect('Neptune#fts.sortBy', 'age.value')
  .V().has('name', 'Neptune#fts marko OR vadas OR ripple')
```

ID フィールド値で並べ替える例

```
g.withSideEffect("Neptune#fts.endpoint", "your-OpenSearch-endpoint-URL")
.withSideEffect('Neptune#fts.queryType', 'query_string')
.withSideEffect('Neptune#fts.sortOrder', 'asc')
.withSideEffect('Neptune#fts.sortBy', 'Neptune#fts.entity_id')
.V().has('name', 'Neptune#fts marko OR vadas OR ripple')
```

ラベル付けフィールド値で並べ替える例

```
g.withSideEffect("Neptune#fts.endpoint", "your-OpenSearch-endpoint-URL")
  .withSideEffect('Neptune#fts.queryType', 'query_string')
  .withSideEffect('Neptune#fts.sortOrder', 'asc')
  .withSideEffect('Neptune#fts.sortBy', 'Neptune#fts.entity_type')
  .V().has('name', 'Neptune#fts marko OR vadas OR ripple')
```

`document_type` フィールド値による並べ替えの例

```
g.withSideEffect("Neptune#fts.endpoint", "your-OpenSearch-endpoint-URL")
  .withSideEffect('Neptune#fts.queryType', 'query_string')
  .withSideEffect('Neptune#fts.sortOrder', 'asc')
  .withSideEffect('Neptune#fts.sortBy', 'Neptune#fts.document_type')
  .V().has('name', 'Neptune#fts marko OR vadas OR ripple')
```

Neptune フルテキスト検索のトラブルシューティング

> **Note**
>
> OpenSearch クラスターで詳細なアクセスコントロールを有効にした場合は、Neptune データベースでも IAM 認証を有効にする必要があります。

Neptune から OpenSearch へのレプリケーションに関する問題を診断するには、ポーラー Lambda 関数の CloudWatch Logs を参照してください。これらのログには、ストリームから読み取られたレコードの数と、OpenSearch に正常にレプリケートされたレコードの数に関する詳細が記載されています。

`LoggingLevel` 環境変数を変更することで、Lambda 関数の LOGGING レベルを変更することもできます。

> **Note**
>
> `LoggingLevel` を `DEBUG` に設定すると、StreamPoller によって Neptune から OpenSearch へデータがレプリケートされるときに削除されたストリームレコードや各レコードが削除された理由など、追加の詳細を表示できます。これは、レコードが欠落している場合に便利です。

Neptune ストリームコンシューマーアプリケーションは、問題の診断にも役立つ、CloudWatch 上の 2 つのメトリクスを公開します。

- `StreamRecordsProcessed` – 単位時間当たりのアプリケーションによって処理されたレコード数。アプリケーションの実行率を追跡するのに役立ちます。
- `StreamLagTime` – 現在時刻と処理中のストリームレコードのコミット時刻との間の時間差 (ミリ秒単位)。このメトリクスは、コンシューマーアプリケーションがどれだけ遅れているかを示します。

さらに、レプリケーションプロセスに関連するすべてのメトリクスは、CloudWatch テンプレートを使用してアプリケーションをインスタンス化したときに `ApplicationName` によって提供されたものと同じ名前で CloudWatch のダッシュボードに表示されます。

また、ポーリングが 2 回以上連続して失敗したときにトリガーされる CloudWatch アラームを作成することもできます。これを行うには、アプリケーションをインスタンス化するときに、`CreateCloudWatchAlarm` フィールドを `true` に設定します。次に、アラームがトリガーされたときに通知する E メールアドレスを指定します。

ストリームからのレコードの読み取り中に失敗するプロセスのトラブルシューティング

ストリームからのレコードの読み取り中にプロセスが失敗した場合は、次のことを確認してください。

- ストリームはクラスターで有効になっています。
- Neptune ストリームのエンドポイントは正しい形式です。
  - Gremlin または openCypher の場合: `https://`*`your cluster endpoint`*`:`*`your cluster port`*`/propertygraph/stream` またはそのエイリアスの `https://`*`your cluster endpoint`*`:`*`your cluster port`*`/pg/stream`
  - SPARQL の場合 : `https://`*`your cluster endpoint`*`:`*`your cluster port`*`/sparql/stream`
- DynamoDB エンドポイントが VPC 用に設定されています。
- モニタリングエンドポイントは VPC サブネットに設定されます。

OpenSearch へのデータの書き込み中に失敗するプロセスのトラブルシューティング

OpenSearch へのレコードの書き込み中にプロセスが失敗した場合は、次のことを確認してください。

- お使いの Elasticsearch のバージョンが 7.1 以上、または Opensearch 2.3 以上であること。
- VPC のポーラー Lambda 関数から OpenSearch にアクセスできること。
- OpenSearch にアタッチされたセキュリティポリシーにより、インバウンド HTTP/HTTPS リクエストが許可されること。

既存のレプリケーション設定で Neptune と OpenSearch 間の同期が取れない問題の修正

`ExpiredStreamException` またはデータの破損により、Neptune データベースと OpenSearch ドメイン間で同期が取れない問題が発生した場合に備えて、以下の手順を使用して Neptune データベースと OpenSearch ドメインを最新のデータと同期させることができます。

この方法では、OpenSearch ドメイン内のすべてのデータが削除され、Neptune データベースの現在の状態から再同期されるため、Neptune データベースにデータを再ロードする必要がないことに注意してください。

1. 「ストリームポーラープロセスの無効化 (一時停止)」で説明されているように、レプリケーションプロセスを無効にします。
2. 以下のコマンドを使用して、OpenSearch ドメインの Neptune インデックスを削除します。

   ```
   curl -X DELETE "(your OpenSearch endpoint)/amazon_neptune"
   ```

3. データベースのクローンを作成します (「Neptune でのデータベースクローニング」を参照)。
4. Streams API エンドポイントに対してこの種のコマンドを実行することによって、クローンされたデータベース上のストリームの最新の `eventID` を取得します (詳細については、「Neptune Streams REST API の呼び出し」を参照)。

   ```
   curl "https://(your neptune endpoint):(port)/(propertygraph or sparql)/stream?
   iteratorType=LATEST"
   ```

   レスポンス内の `lastEventId` オブジェクト内の `commitNum` および `opNum` フィールドの値を書き留めておきます。

5. github の export-neptune-to-elasticsearch ツールを使用して、クローンされたデータベースから OpenSearch ドメインへの 1 回限りの同期を実行します。
6. レプリケーションスタックの DynamoDB テーブルに移動します。テーブルの名前は、AWS CloudFormation テンプレートで指定したアプリケーション名 (デフォルトは`NeptuneStream`) に `-LeaseTable` サフィックスを付けたものです。つまり、デフォルトのテーブル名は `NeptuneStream-LeaseTable` です。

   テーブルには行が 1 つしかないはずなので、スキャンすることでテーブルの行を調べることができます。上記で記録した `commitNum` と `opNum` の値を使用して、次の変更を行います。

- テーブル内の `checkpoint` フィールドの値を、`commitNum` について書き留めた値に変更します。
- テーブル内の `checkpointSubSequenceNumber` フィールドの値を、`opNum` について書き留めた値に変更します。

7. 「ストリームポーラープロセスの再有効化」の説明に従って、レプリケーションプロセスを再び有効にします。
8. クローンされたデータベースと、`export-neptune-to-elasticsearch` ツール用に作成された AWS CloudFormation スタックを削除します。

Amazon Neptune で AWS Lambda 関数を使用する

AWS Lambda 関数は Amazon Neptune アプリケーションで多くの用途があります。ここでは、一般的な Gremlin ドライバーと言語バリアントで Lambda 関数を使用するための一般的なガイダンス、および Java、JavaScript、および Python で記述された Lambda 関数の具体的な例を紹介します。

> **Note**
>
> Neptune で Lambda 関数を使用する最良の方法は、最近のエンジンリリースで変更されました。Neptune では、Lambda 実行コンテキストがリサイクルされた後接続をずっとアイドル状態で開いたままとなり、サーバー上のリソースリークを引き起こす可能性がありました。これを軽減するために、以前は各 Lambda 呼び出しで接続を開いたり閉じたりすることを推奨していました。ただし、エンジンバージョン 1.0.3.0 以降では、非アクティブな Lambda 実行コンテキストがリサイクルされた後で接続がリークしないように、アイドル接続タイムアウトが減少しました。そのため、実行コンテキストの間は 1 つの接続を使用することをお勧めします。これには、予期せずクローズされた接続を処理するためのエラー処理とバックオフと再試行の定型コードを含める必要があります。

AWS Lambda 関数で Gremlin WebSocket 接続を管理する

Gremlin 言語バリアントを使用して Neptune を照会する場合、ドライバは WebSocket 接続を使用してデータベースに接続します。WebSockets は、長寿命のクライアント/サーバー接続シナリオをサポートするように設計されています。一方で、AWS Lambda は比較的短命でステートレスな実行をサポートするように設計されています。設計哲学におけるこの不一致は、Lambda を使用して Neptune を照会するときに予期しない問題が発生する可能性があります。

ある AWS Lambda 関数は、関数を他の関数から分離する実行コンテキストです。実行コンテキストは、関数が最初に呼び出されたときに作成され、同じ関数の後続の呼び出しで再利用できます。

ただし、1 つの実行コンテキストは、関数の複数の同時呼び出しを処理するために使用されることはありません。関数が複数のクライアントによって同時に呼び出された場合、Lambda は関数のインスタンスごとに追加の実行コンテキストをスピンアップします。これらの新しい実行コンテキストはすべて、関数のそれ以降の呼び出しに対して再利用されます。

ある時点で、Lambda は実行コンテキストをリサイクルします。特に、しばらくの間非アクティブになっている場合は実行コンテキストがリサイクルされます。AWS Lambda は実行コンテキストのラ

イフサイクルを公開し、これには Lambda の拡張による Init、Invoke、Shutdown フェーズが含まれます。これらの拡張機能を使用すると、実行コンテキストがリサイクルされたときにデータベース接続などの外部リソースをクリーンアップするコードを記述できます。

一般的なベストプラクティスは、ハンドラ呼び出しごとに再利用できるようにLambda ハンドラー関数の外部でデータベース接続を開きます。ある時点でデータベース接続が切断された場合は、ハンドラ内から再接続できます。ただし、この方法では接続リークの危険性があります。実行コンテキストが破棄されてから接続がアイドル状態で長く開いたままの場合、断続的またはバースト的な Lambda 呼び出しシナリオでは、徐々に接続がリークされ、データベースリソースが枯渇する可能性があります。

Neptune の接続制限と接続タイムアウトは、新しいエンジンリリースで変更されました。以前は、すべてのインスタンスが最大 60,000 の WebSocket 接続をサポートしていました。Neptune インスタンスごとの WebSocket の同時接続の最大数はインスタンスタイプによって異なります。。

また、エンジンリリース 1.0.3.0 以降、Neptune は接続のアイドルタイムアウトを 1 時間から約 20 分に短縮しました。クライアントが接続を閉じない場合、20～25 分のアイドルタイムアウト後に接続は自動的に閉じられます。AWS Lambda は実行コンテキストのライフタイムを文書化しませんが、実験では、新しい Neptune 接続タイムアウトが非アクティブな Lambda 実行コンテキストのタイムアウトと適切に一致することが示されています。非アクティブな実行コンテキストがリサイクルされるまでに、その接続が Neptune によって既に閉じられているか、その後すぐに閉じられる可能性があります。

AWS Lambda と Amazon Neptune Gremlinを一緒に使用する際の推奨事項

ここでは、関数呼び出しごとに 1 つではなく、Lambda 実行コンテキストのライフタイム全体に 1 つの接続とグラフトラバーサルソースを使用することをお勧めします (すべての関数の呼び出しは 1 つのクライアントリクエストのみを処理します)。同時クライアント要求は、別々の実行コンテキストで実行される異なる関数インスタンスによって処理されるため、関数インスタンス内の同時リクエストを処理するために、接続のプールを維持する必要はありません。使用している Gremlin ドライバに接続プールがある場合は、接続を 1 つだけ使用するように構成します。

接続エラーを処理するには、各クエリで再試行ロジックを使用します。実行コンテキストの存続期間中、単一の接続を維持することが目的ですが、予期しないネットワークイベントによってその接続が突然終了する可能性があります。このような接続障害は、使用しているドライバによって異なるエラーとして現れます。Lambda 関数をコーディングして、これらの接続の問題を処理し、必要に応じて再接続を試みる必要があります。

一部の Gremlin ドライバは再接続を自動的に処理します。たとえば、Java ドライバは、クライアントコードの代わりに Neptune への接続を自動的に再確立しようとします。このドライバでは、関数コードを戻してクエリを再試行するだけで済みます。これに対し、JavaScript および Python ドライバは自動再接続ロジックを実装しないため、これらのドライバでは、関数コードがバックオフ後に再接続を試み、接続が再確立された後にのみクエリを再試行する必要があります。

ここでのコード例には、クライアントがそれを処理していると仮定するのではなく、再接続ロジックが含まれています。

Lambda で Gremlin 書き込みリクエストを使用するための推奨事項

Lambda 関数がグラフデータを変更する場合は、バックオフと再試行戦略を採用して、次の例外を処理することを検討してください。

- **`ConcurrentModificationException`** — Neptune トランザクションセマンティクスは、書き込み要求が `ConcurrentModificationException` で失敗することがあるということを意味しています。このような状況では、指数バックオフベースの再試行メカニズムを試してください。
- **`ReadOnlyViolationException`** — クラスタートポロジは、計画されたイベントまたは計画外のイベントによっていつでも変更される可能性があるため、書き込み責任がクラスタ内のインスタンスから別のインスタンスに移行することがあります。関数コードが、プライマリ (ライター) インスタンスでなくなったインスタンスに書き込みリクエストを送信しようとすると、リクエストは `ReadOnlyViolationException` で失敗します。この場合、既存の接続を閉じて、クラスターエンドポイントに再接続してから、リクエストを再試行します。

また、バックオフと再試行戦略を使用して書き込みリクエストの問題を処理する場合は、作成リクエストと更新リクエストの冪等クエリを実装することを検討してください (たとえば、次を使用。fold().coalesce().unfold()。

Lambda で Gremlin 読み取りリクエストを使用するための推奨事項

クラスター内に 1 つ以上のリードレプリカがある場合は、これらのレプリカ間で読み取りリクエストのバランスを取ることをお勧めします。1 つのオプションは、リーダーエンドポイントを使うことです。リーダーエンドポイントは、レプリカを追加または削除する、またはレプリカを新しいプライマリインスタンスにするときにクラスタトポロジが変更された場合でも、レプリカ間の接続のバランスをとります。

ただし、リーダーエンドポイントを使用すると、状況によってはクラスタリソースの不均等な使用が発生する可能性があります。リーダーエンドポイントのラウンドロビンルーティングを実行するに

は、DNS エントリがポイントするホストを変更します。DNS エントリが変更される前にクライアントが多くの接続を開くと、すべての接続要求が単一の Neptune インスタンスに送信されます。これは、Lambda 関数への多数の同時リクエストによって複数の実行コンテキストが作成され、それぞれが独自の接続を持つ高スループットの Lambda シナリオの場合です。これらの接続がすべてほぼ同時に作成されている場合、接続はすべてクラスタ内の同じレプリカを指し、実行コンテキストがリサイクルされるまでそのレプリカを指し続けます。

インスタンス間でリクエストを分散する方法の 1 つは、リーダーエンドポイントではなく、レプリカインスタンスのエンドポイントのリストからランダムに選択されたインスタンスエンドポイントに接続するように Lambda 関数を設定することです。この方法の欠点は、クラスターを監視し、クラスターのメンバーシップが変更されるたびにエンドポイントリストを更新することで、クラスタートポロジの変更を処理する Lambda コードが必要であることです。

クラスター内のインスタンス間で読み取りリクエストのバランスを取る Java Lambda 関数を作成する場合は、クラスタートポロジを認識しており、Neptune クラスター内のインスタンスのセット間で接続とリクエストを公平に分散する Java Gremlin クライアントである、Amazon Neptune の Gremlin クライアントを使用できます。このブログ投稿には、Amazon Neptune 用の Gremlin クライアントを使用するサンプル Java Lambda 関数が含まれています。

Neptune Gremlin Lambda 関数のコールドスタートを遅らせる可能性がある要因

初めて AWS Lambda 関数が呼び出されることを、コールドスタートと呼びます。コールドスタートのレイテンシーを増加させる要因はいくつかあります。

- 必ず Lambda 関数に十分なメモリを割り当ててください。 — コールドスタート中のコンパイルは、Lambda 関数の場合、EC2 よりも大幅に遅くなる可能性があります。関数に割り当てるメモリに比例して直線的にAWS Lambda が CPU サイクルを割り当てるからです。1,792 MB では、関数は 1 つのフル vCPU (1 秒あたりのクレジットの 1 vCPU 秒) 相当量を受信します。十分な CPU サイクルを受信するのに十分なメモリを割り当てないことの影響は、Java で記述された大規模な Lambda 関数で特に顕著です。
- IAM データベース認証の有効化によりコールドスタートが遅くなる可能性があることに注意する – AWS Identity and Access Management (IAM) データベース認証では、特に Lambda 関数で新しい署名キーを生成する必要がある場合に、コールドスタートが遅くなる可能性があります。このレイテンシーはコールドスタートにのみ影響し、後続のリクエストには影響しません。これは、IAM DB 認証が接続認証情報を確立すると、Neptune が定期的にそれらの認証情報がまだ有効であることを検証するだけだからです。

AWS CloudFormation を使用して Neptune で使用する Lambda 関数を作成するには

AWS CloudFormation テンプレートを使用して、Neptune にアクセスできる AWS Lambda 関数を作成できます。

1. AWS CloudFormation コンソールで Lambda 関数スタックを起動するには、以下の表でいずれかの [Launch Stack] (スタックの起動) ボタンを選択します。

リージョン	ビュー	デザイナーで表示	起動する
米国東部 (バージニア北部)	表示	デザイナーで表示	
米国東部 (オハイオ)	表示	デザイナーで表示	
米国西部 (北カリフォルニア)	表示	デザイナーで表示	
米国西部 (オレゴン)	表示	デザイナーで表示	
カナダ (中部)	表示	デザイナーで表示	
南米 (サンパウロ)	表示	デザイナーで表示	
欧州 (ストックホルム)	表示	デザイナーで表示	
欧州 (アイルランド)	表示	デザイナーで表示	
欧州 (ロンドン)	表示	デザイナーで表示	
欧州 (パリ)	表示	デザイナーで表示	

リージョン	ビュー	デザイナーで表示	起動する
欧州 (フランクフルト)	表示	デザイナーで表示	
中東 (バーレーン)	表示	デザイナーで表示	
中東 (アラブ首長国連邦)	表示	デザイナーで表示	
イスラエル (テルアビブ)	表示	デザイナーで表示	
アフリカ (ケープタウン)	表示	デザイナーで表示	
アジアパシフィック (香港)	表示	デザイナーで表示	
アジアパシフィック (東京)	表示	デザイナーで表示	
アジアパシフィック (ソウル)	表示	デザイナーで表示	
アジアパシフィック (シンガポール)	表示	デザイナーで表示	
アジアパシフィック (シドニー)	表示	デザイナーで表示	
アジアパシフィック (ムンバイ)	表示	デザイナーで表示	
中国 (北京)	表示	デザイナーで表示	
中国 (寧夏)	表示	デザイナーで表示	

リージョン	ビュー	デザイナーで表示	起動する
AWS GovCloud (米国西部)	表示	デザイナーで表示	
AWS GovCloud (米国東部)	表示	デザイナーで表示	

2. [Select Template] ページで、[Next] を選択します。
3. [詳細の指定] ページで、以下のオプションを設定します。

   a. Lambda 関数で使用したい言語に応じて、Lambda ランタイムを選択してください。これらの AWS CloudFormation テンプレートは現在、次の言語をサポートしています。

      - Python 3.9 (Amazon S3 URL の `python39` にマップ)
      - NodeJS 18 (Amazon S3 URL の `nodejs18x` にマップ)
      - Ruby 2.5 (Amazon S3 URL の `ruby25` にマップ)

   b. 適切な Neptune クラスターのエンドポイントとポート番号を入力します。

   c. 適切な Neptune セキュリティグループを入力します。

   d. 適切な Neptune サブネットパラメータを指定します。
4. [Next] (次へ) をクリックします。
5. [Options(オプション)] ページで、[Next(次へ)] を選択します。
6. [Review] (確認) ページで、AWS CloudFormation によって IAM リソースが作成されることを確認する最初のチェックボックスをオンにします。

   次に [作成] を選択します。

Lambda ランタイムに独自の変更を加える必要がある場合は、リージョンの Amazon S3 ロケーションから一般的なものをダウンロードできます。

```
https://s3.Amazon region.amazonaws.com/aws-neptune-customer-samples-Amazon region/
lambda/runtime-language/lambda_function.zip.
```

例:

```
https://s3.us-west-2.amazonaws.com/aws-neptune-customer-samples-us-west-2/lambda/
python36/lambda_function.zip
```

AWS Lambda Amazon Neptune 関数の例

以下の Java、JavaScript、および Python で記述された AWS Lambda 関数の例は、`fold().coalesce().unfold()` イディオムを使用してランダムに生成された ID で単一の頂点をアップサートしていることを示しています。

各関数のコードの多くは定型コードであり、接続を管理し、エラーが発生した場合に接続とクエリを再試行します。実際のアプリケーションロジックと Gremlin クエリは、`doQuery()` および `query()` メソッドのそれぞれを使用します。これらの例を独自の Lambda 関数の基礎として使用すると、`doQuery()` および `query()` の変更に集中できます。

関数は、クエリがうまくいかなければ 5 回再試行し、再試行の間に 1 秒待機するように構成されています。

関数では、次の Lambda 環境変数に値が存在する必要があります。

- **`NEPTUNE_ENDPOINT`** — Neptune DB クラスターエンドポイント。Python の場合、これは `neptuneEndpoint` のはずです。
- **`NEPTUNE_PORT`** — Neptune ポート。Python の場合、これは `neptunePort` のはずです。
- **`USE_IAM`** — (`true` または `false`) データベースに有効な AWS Identity and Access Management (IAM) データベース認証がある場合、`USE_IAM` 環境変数を `true` に設定します。これにより、Lambda 関数は Neptune への接続リクエストを SIGV4 署名します。このような IAM DB 認証リクエストについては、Lambda 関数の実行ロールに、関数が Neptune DB クラスターに接続できるようにする適切な IAM ポリシーがアタッチされていることを確認してください (IAM ポリシーのタイプ を参照)。

Amazon Neptune の Java Lambda 関数の例

Java AWS Lambda 関数について留意すべき点をいくつかご紹介します。

- Java ドライバーは独自の接続プールを保持しますが、これは必要ありませんので、`Cluster` のオブジェクトを `minConnectionPoolSize(1)` および `maxConnectionPoolSize(1)` で構成します。

- Cluster オブジェクトは、1 つ以上のシリアライザ (デフォルトで Gyro、binary のような追加の出力フォーマット用に構成している場合は別のシリアライザ) を作成するため、構築に時間がかかることがあります。インスタンス化には時間がかかる場合があります。
- 接続プールは、最初の要求で初期化されます。この時点で、ドライバーは Netty スタックを設定し、バイトバッファを割り当て、IAM DB 認証を使用している場合は、署名キーを作成します。これらすべてがコールドスタートのレイテンシーに追加される可能性があります。
- Java ドライバーの接続プールは、サーバーホストの可用性を監視し、接続に失敗すると自動的に再接続を試みます。バックグラウンドタスクが開始され、接続の再確立が試行されます。reconnectInterval( ) を使用して、再接続の試行間隔を設定します。ドライバーが再接続しようとしている間、Lambda 関数はクエリの再試行のみができます。

  再試行間隔が再接続試行の間隔より短い場合、ホストが使用できないと見なされるため、失敗した接続に対する再試行は再度失敗します。これは、ConcurrentModificationException の再試行には該当しません。
- Java 11ではなくJava 8を使用してください。Java 11 では、Netty 最適化はデフォルトで有効になっていません。
- この例では再試行に retry4J を使用します。
- Java Lambda 関数で Sigv4 署名ドライバーを使用するには、依存関係の要件について、Java と Gremlin でバージョン 4 署名を使用して Neptune に接続する を参照してください。

> **⚠ Warning**
>
> Retry4J からの CallExecutor はスレッドセーフではないことがあります。各スレッドで独自の CallExecutor インスタンスを使用することを検討してください。

```
package com.amazonaws.examples.social;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestStreamHandler;
import com.evanlennick.retry4j.CallExecutor;
import com.evanlennick.retry4j.CallExecutorBuilder;
import com.evanlennick.retry4j.Status;
import com.evanlennick.retry4j.config.RetryConfig;
import com.evanlennick.retry4j.config.RetryConfigBuilder;
import org.apache.tinkerpop.gremlin.driver.Cluster;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
```

```
import com.amazonaws.neptune.auth.NeptuneNettyHttpSigV4Signer;
import org.apache.tinkerpop.gremlin.driver.remote.DriverRemoteConnection;
import org.apache.tinkerpop.gremlin.driver.ser.Serializers;
import org.apache.tinkerpop.gremlin.process.traversal.AnonymousTraversalSource;
import org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.GraphTraversalSource;
import org.apache.tinkerpop.gremlin.structure.T;

import java.io.*;
import java.time.temporal.ChronoUnit;
import java.util.HashMap;
import java.util.Map;
import java.util.Random;
import java.util.concurrent.Callable;
import java.util.function.Function;

import static java.nio.charset.StandardCharsets.UTF_8;
import static org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.__.addV;
import static org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.__.unfold;

public class MyHandler implements RequestStreamHandler {

  private final GraphTraversalSource g;
  private final CallExecutor<Object> executor;
  private final Random idGenerator = new Random();

  public MyHandler() {

    this.g = AnonymousTraversalSource
            .traversal()
            .withRemote(DriverRemoteConnection.using(createCluster()));


    this.executor = new CallExecutorBuilder<Object>()
            .config(createRetryConfig())
            .build();

  }

  @Override
  public void handleRequest(InputStream input,
                            OutputStream output,
                            Context context) throws IOException {

    doQuery(input, output);
```

```
  }

  private void doQuery(InputStream input, OutputStream output) throws IOException {
    try {

      Map<String, Object> args = new HashMap<>();
      args.put("id", idGenerator.nextInt());

      String result = query(args);

      try (Writer writer = new BufferedWriter(new OutputStreamWriter(output, UTF_8))) {
          writer.write(result);
      }

    } finally {
        input.close();
        output.close();
    }
  }

  private String query(Map<String, Object> args) {
    int id = (int) args.get("id");

    @SuppressWarnings("unchecked")
    Callable<Object> query = () -> g.V(id)
      .fold()
      .coalesce(
        unfold(),
        addV("Person").property(T.id, id))
      .id().next();

    Status<Object> status = executor.execute(query);

    return status.getResult().toString();
  }

  private Cluster createCluster() {
    Cluster.Builder builder = Cluster.build()

.addContactPoint(System.getenv("NEPTUNE_ENDPOINT"))

.port(Integer.parseInt(System.getenv("NEPTUNE_PORT")))
                                    .enableSsl(true)
                                    .minConnectionPoolSize(1)
```

```
                                    .maxConnectionPoolSize(1)
                                    .serializer(Serializers.GRAPHBINARY_V1D0)
                                    .reconnectInterval(2000);

 if (Boolean.parseBoolean(getOptionalEnv("USE_IAM", "true"))) {
   // For versions of TinkerPop 3.4.11 or higher:
   builder.handshakeInterceptor( r ->
     {
       NeptuneNettyHttpSigV4Signer sigV4Signer = new
NeptuneNettyHttpSigV4Signer(region, new DefaultAWSCredentialsProviderChain());
       sigV4Signer.signRequest(r);
       return r;
     }
   )

   // Versions of TinkerPop prior to 3.4.11 should use the following approach.
   // Be sure to adjust the  imports to include:
   //   import org.apache.tinkerpop.gremlin.driver.SigV4WebSocketChannelizer;
   //   builder = builder.channelizer(SigV4WebSocketChannelizer.class);

   return builder.create();
 }

 private RetryConfig createRetryConfig() {
   return new RetryConfigBuilder().retryOnCustomExceptionLogic(retryLogic())
                                  .withDelayBetweenTries(1000, ChronoUnit.MILLIS)
                                  .withMaxNumberOfTries(5)
                                  .withFixedBackoff()
                                  .build();
 }

 private Function<Exception, Boolean> retryLogic() {
   return e -> {
     StringWriter stringWriter = new StringWriter();
     e.printStackTrace(new PrintWriter(stringWriter));
     String message = stringWriter.toString();

     // Check for connection issues
     if ( message.contains("Timed out while waiting for an available host") ||
          message.contains("Timed-out waiting for connection on Host") ||
          message.contains("Connection to server is no longer active") ||
          message.contains("Connection reset by peer") ||
          message.contains("SSLEngine closed already") ||
          message.contains("Pool is shutdown") ||
```

```
            message.contains("ExtendedClosedChannelException") ||
            message.contains("Broken pipe")) {
          return true;
        }

        // Concurrent writes can sometimes trigger a ConcurrentModificationException.
        // In these circumstances you may want to backoff and retry.
        if (message.contains("ConcurrentModificationException")) {
            return true;
        }

        // If the primary fails over to a new instance, existing connections to the old
 primary will
        // throw a ReadOnlyViolationException. You may want to back and retry.
        if (message.contains("ReadOnlyViolationException")) {
            return true;
        }

        return false;
      };
    }

    private String getOptionalEnv(String name, String defaultValue) {
      String value = System.getenv(name);
      if (value != null && value.length() > 0) {
        return value;
      } else {
        return defaultValue;
      }
    }
}
```

関数に再接続ロジックを含める場合は、Java 再接続のサンプル を参照してください。

Amazon Neptune の JavaScript Lambda 関数の例

この例についての注意

- JavaScript ドライバーは接続プールを維持しません。常に 1 つの接続を開きます。
- このサンプル関数は、IAM 認証が有効なデータベースへのリクエストに署名するため、gremlin-aws-sigv4 の Sigv4 署名ユーティリティを使用します。

- これは、オープンソース非同期ユーティリティモジュールからの retry( ) 関数を使用してバックオフと再試行を処理します。
- Gremlin ターミナルステップは JavaScript `promise` を返します (TinkerPop ドキュメントを参照してください)。`next()` の場合、これは `{value, done}` タプルです。
- 接続エラーはハンドラ内で発生し、ここで概説した推奨事項に沿ったバックオフと再試行ロジックを使用して処理されます。ただし、例外が 1 つあります。ドライバーが例外として扱わないため、このバックオフと再試行ロジックでは対応できないある種の接続の問題があります。

  問題は、ドライバーが要求を送信した後、ドライバーが応答を受信する前に接続が閉じられた場合、クエリは完了しているように見え、NULL 値を返すことです。Lambda 関数クライアントに関する限り、関数は正常に完了しているように見えますが、レスポンスは空です。

  この問題の影響は、アプリケーションが空のレスポンスをどのように扱うかによって異なります。一部のアプリケーションでは、読み取りリクエストからの空のレスポンスをエラーとして扱うことがありますが、他のアプリケーションでは誤って空の結果として扱われる場合があります。

  この接続の問題に遭遇した書き込み要求も、空の応答を返します。空の応答で成功した呼び出しは、成功または失敗を示しますか？ 書き込み関数を呼び出すクライアントが、応答の本文を調べるのではなく、データベースへの書き込みがコミットされたことを意味する関数の正常な呼び出しを処理すると、システムがデータを失ったように見える場合があります。

  この問題は、基になるソケットによって発生したイベントをドライバが処理する方法に起因します。基盤となるネットワークソケットが `ECONNRESET` エラーで閉じると、ドライバーが使用する WebSocket が閉じられ、`'ws close'` イベントの発生となります。ただし、ドライバーには例外としてそのイベントを処理するものが含まれていません。その結果、クエリは単に消えます。

  この問題を回避するために、この例では Lambda 関数を使用して、リモート接続の作成時にドライバーに例外をスローする `'ws close'` イベントハンドラーを追加しています。ただし、この例外は Gremlin クエリの要求応答パスに沿って発生しないため、Lambda 関数自体のバックオフと再試行ロジックをトリガーするために使用することはできません。代わりに、`'ws close'` イベントハンドラーによってスローされる例外は、Lambda 呼び出しが失敗する原因となる、未処理の例外となってしまいます。これにより、関数を呼び出すクライアントはエラーを処理し、必要に応じて Lambda 呼び出しを再試行できます。

  クライアントを断続的な接続の問題から保護するために、Lambda 関数自体にバックオフと再試行ロジックを実装することをお勧めします。ただし、上記の問題の回避策では、この特定の接続の問題に起因する障害を処理するために、クライアントが再試行ロジックを実装する必要があります。

JavaScript コード

```
const gremlin = require('gremlin');
const async = require('async');
const {getUrlAndHeaders} = require('gremlin-aws-sigv4/lib/utils');

const traversal = gremlin.process.AnonymousTraversalSource.traversal;
const DriverRemoteConnection = gremlin.driver.DriverRemoteConnection;
const t = gremlin.process.t;
const __ = gremlin.process.statics;

let conn = null;
let g = null;

async function query(context) {

  const id = context.id;

  return g.V(id)
    .fold()
    .coalesce(
      __.unfold(),
      __.addV('User').property(t.id, id)
    )
    .id().next();
}

async function doQuery() {
  const id = Math.floor(Math.random() * 10000).toString();

  let result = await query({id: id});
  return result['value'];
}

exports.handler = async (event, context) => {

  const getConnectionDetails = () => {
    if (process.env['USE_IAM'] == 'true'){
       return getUrlAndHeaders(
         process.env['NEPTUNE_ENDPOINT'],
         process.env['NEPTUNE_PORT'],
         {},
         '/gremlin',
```

```
        'wss');
    } else {
      const database_url = 'wss://' + process.env['NEPTUNE_ENDPOINT'] + ':' +
process.env['NEPTUNE_PORT'] + '/gremlin';
      return { url: database_url, headers: {}};
    }
 };


 const createRemoteConnection = () => {
   const { url, headers } = getConnectionDetails();

   const c = new DriverRemoteConnection(
     url,
     {
       mimeType: 'application/vnd.gremlin-v2.0+json',
       headers: headers
     });

    c._client._connection.on('close', (code, message) => {
        console.info(`close - ${code} ${message}`);
        if (code == 1006){
          console.error('Connection closed prematurely');
          throw new Error('Connection closed prematurely');
        }
      });

    return c;
 };

 const createGraphTraversalSource = (conn) => {
   return traversal().withRemote(conn);
 };

 if (conn == null){
   console.info("Initializing connection")
   conn = createRemoteConnection();
   g = createGraphTraversalSource(conn);
 }

 return async.retry(
   {
     times: 5,
     interval: 1000,
```

```
      errorFilter: function (err) {

        // Add filters here to determine whether error can be retried
        console.warn('Determining whether retriable error: ' + err.message);

        // Check for connection issues
        if (err.message.startsWith('WebSocket is not open')){
          console.warn('Reopening connection');
          conn.close();
          conn = createRemoteConnection();
          g = createGraphTraversalSource(conn);
          return true;
        }

        // Check for ConcurrentModificationException
        if (err.message.includes('ConcurrentModificationException')){
          console.warn('Retrying query because of ConcurrentModificationException');
          return true;
        }

        // Check for ReadOnlyViolationException
        if (err.message.includes('ReadOnlyViolationException')){
          console.warn('Retrying query because of ReadOnlyViolationException');
          return true;
        }

        return false;
      }

    },
    doQuery);
};
```

Amazon Neptune の Python Lambda 関数の例

次の Python AWS Lambda 関数の例について注意すべき点をいくつかご紹介します。

- これは、バックオフモジュールを使用しています。
- `pool_size=1` を設定して不要な接続プールを作成しないようにします。
- `message_serializer=serializer.GraphSONSerializersV2d0()` を設定します。

```
import os, sys, backoff, math
from random import randint
from gremlin_python import statics
from gremlin_python.driver.driver_remote_connection import DriverRemoteConnection
from gremlin_python.driver.protocol import GremlinServerError
from gremlin_python.driver import serializer
from gremlin_python.process.anonymous_traversal import traversal
from gremlin_python.process.graph_traversal import __
from gremlin_python.process.strategies import *
from gremlin_python.process.traversal import T
from aiohttp.client_exceptions import ClientConnectorError
from botocore.auth import SigV4Auth
from botocore.awsrequest import AWSRequest
from botocore.credentials import ReadOnlyCredentials
from types import SimpleNamespace

import logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)


reconnectable_err_msgs = [
    'ReadOnlyViolationException',
    'Server disconnected',
    'Connection refused',
    'Connection was already closed',
    'Connection was closed by server',
    'Failed to connect to server: HTTP Error code 403 - Forbidden'
]

retriable_err_msgs = ['ConcurrentModificationException'] + reconnectable_err_msgs

network_errors = [OSError, ClientConnectorError]

retriable_errors = [GremlinServerError, RuntimeError, Exception] + network_errors

def prepare_iamdb_request(database_url):

    service = 'neptune-db'
    method = 'GET'

    access_key = os.environ['AWS_ACCESS_KEY_ID']
    secret_key = os.environ['AWS_SECRET_ACCESS_KEY']
```

```
    region = os.environ['AWS_REGION']
    session_token = os.environ['AWS_SESSION_TOKEN']

    creds = SimpleNamespace(
        access_key=access_key, secret_key=secret_key, token=session_token,
 region=region,
    )

    request = AWSRequest(method=method, url=database_url, data=None)
    SigV4Auth(creds, service, region).add_auth(request)

    return (database_url, request.headers.items())

def is_retriable_error(e):

    is_retriable = False
    err_msg = str(e)

    if isinstance(e, tuple(network_errors)):
        is_retriable = True
    else:
        is_retriable = any(retriable_err_msg in err_msg for retriable_err_msg in
 retriable_err_msgs)

    logger.error('error: [{}] {}'.format(type(e), err_msg))
    logger.info('is_retriable: {}'.format(is_retriable))

    return is_retriable

def is_non_retriable_error(e):
    return not is_retriable_error(e)

def reset_connection_if_connection_issue(params):

    is_reconnectable = False

    e = sys.exc_info()[1]
    err_msg = str(e)

    if isinstance(e, tuple(network_errors)):
        is_reconnectable = True
    else:
        is_reconnectable = any(reconnectable_err_msg in err_msg for
 reconnectable_err_msg in reconnectable_err_msgs)
```

```
    logger.info('is_reconnectable: {}'.format(is_reconnectable))

    if is_reconnectable:
        global conn
        global g
        conn.close()
        conn = create_remote_connection()
        g = create_graph_traversal_source(conn)

@backoff.on_exception(backoff.constant,
    tuple(retriable_errors),
    max_tries=5,
    jitter=None,
    giveup=is_non_retriable_error,
    on_backoff=reset_connection_if_connection_issue,
    interval=1)
def query(**kwargs):

    id = kwargs['id']

    return (g.V(id)
        .fold()
        .coalesce(
            __.unfold(),
            __.addV('User').property(T.id, id)
        )
        .id().next())

def doQuery(event):
    return query(id=str(randint(0, 10000)))

def lambda_handler(event, context):
    result = doQuery(event)
    logger.info('result - {}'.format(result))
    return result

def create_graph_traversal_source(conn):
    return traversal().withRemote(conn)

def create_remote_connection():
    logger.info('Creating remote connection')

    (database_url, headers) = connection_info()
```

```
    return DriverRemoteConnection(
        database_url,
        'g',
        pool_size=1,
        message_serializer=serializer.GraphSONSerializersV2d0(),
        headers=headers)

def connection_info():

    database_url = 'wss://{}:{}/gremlin'.format(os.environ['neptuneEndpoint'],
 os.environ['neptunePort'])

    if 'USE_IAM' in os.environ and os.environ['USE_IAM'] == 'true':
        return prepare_iamdb_request(database_url)
    else:
        return (database_url, {})

conn = create_remote_connection()
g = create_graph_traversal_source(conn)
```

以下はサンプルの結果であり、負荷が高い期間と軽い負荷が交互に繰り返されることを示しています。


GremlinWebSocketOpenConnections
Count
261
131
0
06:10 06:15 06:20 06:25 06:30 06:35 06:40 06:45 06:50 06:55 07:00 07:05
neptunedbinstance-l22qsahgvgvl
replica-python-lambdas-neptunebasestack-5rzhq9jvp2ri
GremlinRequestsPreSecond
Count/Second
861
430
0
06:10 06:15 06:20 06:25 06:30 06:35 06:40 06:45 06:50 06:55 07:00 07:05
neptunedbinstance-l22qsahgvgvl
replica-python-lambdas-neptunebasestack-5rzhq9jvp2ri
Lambda-ConcurrentExecutions
Count
86
43.5
1
06:10 06:15 06:20 06:25 06:30 06:35 06:40 06:45 06:50 06:55 07:00 07:05
social-b6501310_add-follows-relationship social-b6501310_add-follows-relationship
social-b6501310_create-user social-b6501310_create-user
social-b6501310_get-recent-follows social-b6501310_get-recent-follows
social-b6501310_get-user-details social-b6501310_get-user-details
Lambda-P99-Latency
Milliseconds
2.10k
1.05k
5.46
06:05 06:10 06:15 06:20 06:25 06:30 06:35 06:40 06:45 06:50 06:55 07:00 07:05
social-b6501310_add-follows-relationship
social-b6501310_create-user
social-b6501310_get-recent-follows
social-b6501310_get-user-details
social-b6501310_recommend-new-follows
Workload
Various units
Count
2.17k
1.09k
9e-3
147k
73.7k
2
06:10 06:15 06:20 06:25 06:30 06:35 06:40 06:45 06:50 06:55 07:00 07:05
P99Latency
AverageThroughput
Succeeded
Errors
Count
1
0.5
0
06:10 06:15 06:20 06:25 06:30 06:35 06:40 06:45 06:50 06:55 07:00 07:05
social-b6501310_add-follows-relationship
social-b6501310_create-user
social-b6501310_get-recent-follows
social-b6501310_get-user-details
social-b6501310_recommend-new-follows

グラフ上の機械学習のための Amazon Neptune ML

多くの場合、大きな接続データセットには、人間の直感だけに基づくクエリを使用して抽出するのが難しい貴重な情報があります。機械学習 (ML) 手法は、数十億もの関係を持つグラフの隠れた相関を見つけるのに役立ちます。これらの相関関係は、製品の推奨、信用力の予測、詐欺の特定、その他多くの事柄に役立ちます。

Neptune ML 機能を使用すると、数週間ではなく数時間で、大きなグラフで便利な機械学習モデルを構築し、トレーニングすることができます。これを達成するために、Neptune MLは、Amazon SageMaker とディープグラフライブラリ (DGL) (これはオープンソースです) を活用したグラフニューラルネットワーク (GNN) 技術を使用しています。グラフニューラルネットワークは、人工知能における新たな分野です (例えば、グラフニューラルネットワークに関する包括的な調査を参照してください)。DGL での GNN の使用に関する実践的なチュートリアルについては、ディープグラフライブラリを使用したグラフニューラルネットワークの学習を参照してください。

> **Note**
>
> グラフの頂点は、Neptune ML モデルでは「ノード」として識別されます。たとえば、頂点分類ではノード分類機械学習モデルを使用し、頂点回帰はノード回帰モデルを使用します。

Neptune MLができること

Neptune は、トレーニング時に事前に計算された予測をその時点のグラフデータに基づいて返すトランスダクティブ推論と、現在のデータに基づいて適用データ処理とモデル評価をリアルタイムで返す帰納的推論の両方をサポートしています。「帰納的推論とトランスダクティブ推論の違い」を参照してください。

Neptune ML は、次の 5 つの異なるカテゴリの推論をサポートするように機械学習モデルをトレーニングできます。

Neptune ML で現在サポートされている推論タスクのタイプ

- ノード分類 — 頂点プロパティのカテゴリカル特徴を予測します。

  たとえば、ショーシャンクの空にという映画について、Neptune ML はその `genre` プロパティを `[story, crime, action, fantasy, drama, family, ...]` の候補集合から `story` として予測できます。

ノード分類タスクには 2 つのタイプがあります。

- 単一クラス分類: この種のタスクでは、各ノードにはターゲットフィーチャが 1 つしかありません。たとえば、プロパティ、Alan Turing の Place_of_birth には UK 値があります。
- 複数クラス分類: この種のタスクでは、各ノードにはターゲットフィーチャが 1 つしかありません。たとえば、映画ゴッドファーザーのプロパティ genre には crime および story の値があります。

- ノード回帰 — 頂点の数値プロパティを予測します。

  たとえば、映画[アベンジャーズ/エンドゲーム]では、Neptune MLはそのプロパティ popularity は 5.0 の値を有すると予測できます。

- ノード分類 — 頂点プロパティのカテゴリカル特徴を予測します。

  ノード分類タスクには 2 つのタイプがあります。

  - 単一クラス分類: この種のタスクでは、各ノードにはターゲットフィーチャが 1 つしかありません。たとえば、ユーザーと映画の間の評価エッジには、「はい」または「いいえ」のいずれかの値を持つ、プロパティ liked が含まれる場合があります。
  - 複数クラス分類: この種のタスクでは、各ノードにはターゲットフィーチャが 1 つしかありません。たとえば、ユーザーと映画の間の評価には、「面白い」、「心温まる」、「リラックスできる」などのプロパティタグに対する複数の値が含まれる場合があります。

- エッジ回帰 — エッジの数値プロパティを予測します。

  たとえば、ユーザーと映画の間の評価エッジには、数値プロパティ score が含まれる場合があります。このためにNeptune MLは、ユーザーと映画を特定する値を予測できます。

- リンク予測 — 特定のソースノードと発信エッジの最も可能性の高いデスティネーションノード、または特定のデスティネーションノードと着信エッジの最も可能性の高いソースノードを予測します。

  例えば、薬物疾患の知識グラフでは、ソースノードとして Aspirin、出力エッジとして treatsがあり、Neptune ML は最も関連性の高い宛先ノードを heart disease、fever、などと予測できます。

  または、ウィキメディアのナレッジグラフで与えられたエッジ President-of またはデスティネーションノードとしてリレーションおよび United-Statesの場合、Neptune MLは、最も関連性の高いヘッドを George Washington、Abraham Lincoln、Franklin D. Roosevelt、などのように予測できます。

> **Note**
>
> ノード分類とエッジ分類は文字列値のみをサポートします。つまり、`0` や `1` などの数値プロパティ値はサポートされませんが、該当する文字列 `"0"` および `"1"` はサポートされます。同様に、ブールのプロパティ値 `true` および `false` は機能しませんが、`"true"` および `"false"` は機能します。

Neptune ML では、次の 2 つの一般的なカテゴリに分類される機械学習モデルを使用できます。

Neptune ML で現在サポートされている機械学習モデルの種類

- グラフニューラルネットワーク (GNN) モデル — ここにはリレーショナルグラフ畳み込みネットワーク (R-GCNS) が含まれます。GNN モデルは、上記の 3 種類のタスクすべてに対して機能します。
- ナレッジグラフ埋め込み (KGE) モデル — ここには `TransE`、`DistMult`、`RotatE` モデルが含まれます。リンク予測のみに機能します。

ユーザー定義のモデル — Neptune ML では、上記のすべてのタイプのタスクに対して独自のカスタムモデル実装を提供することもできます。Neptune ML ツールキットで Neptune ML トレーニング API を使用する前に、Python ベースのカスタムモデル実装を開発およびテストします。Neptune ML のトレーニングインフラストラクチャと互換性があるように、実装を構造化して編成する方法の詳細については、Neptune ML のカスタムモデル を参照してください。

Neptune ML の設定

Neptune ML の使用を開始する最も簡単な方法は、 AWS CloudFormation クイックスタートテンプレートを使用することです。このテンプレートは、新しい Neptune DB クラスター、必要なすべての IAM ロール、新しい Neptune グラフノートブックなど、Neptune ML での作業を容易にするために必要なすべてのコンポーネントをインストールします。

クイックスタート AWS CloudFormation テンプレートを使わずに Neptune ML をセットアップする で説明されているように、Neptune ML を手動でインストールすることもできます。

新しい DB クラスターをすぐに使用し始めるための Neptune ML AWS CloudFormation テンプレートを使う

Neptune MLの使用を開始する最も簡単な方法は、AWS CloudFormation クイックスタートテンプレートを使用することです。このテンプレートは、新しい Neptune DB クラスター、すべての必要な IAM ロール、新しい Neptune グラフノートブックなど、Neptune ML での作業を容易にするために必要なすべてのコンポーネントをインストールします。

Neptune ML クイックスタートスタックを作成するには

1. AWS CloudFormation コンソールで AWS CloudFormation スタックを起動するには、以下の表でいずれかの [Launch Stack (スタックの起動)] ボタンを選択します。

リージョン	ビュー	デザイナーで表示	起動する
米国東部 (バージニア北部)	表示	デザイナーで表示	Launch Stack
米国東部 (オハイオ)	表示	デザイナーで表示	Launch Stack
米国西部 (北カリフォルニア)	表示	デザイナーで表示	Launch Stack
米国西部 (オレゴン)	表示	デザイナーで表示	Launch Stack
カナダ (中部)	表示	デザイナーで表示	Launch Stack
南米 (サンパウロ)	表示	デザイナーで表示	Launch Stack
欧州 (ストックホルム)	表示	デザイナーで表示	Launch Stack
欧州 (アイルランド)	表示	デザイナーで表示	Launch Stack

リージョン	ビュー	デザイナーで表示	起動する
欧州 (ロンドン)	表示	デザイナーで表示	Launch Stack
欧州 (パリ)	表示	デザイナーで表示	Launch Stack
欧州 (フランクフルト)	表示	デザイナーで表示	Launch Stack
中東 (バーレーン)	表示	デザイナーで表示	Launch Stack
中東 (アラブ首長国連邦)	表示	デザイナーで表示	Launch Stack
イスラエル (テルアビブ)	表示	デザイナーで表示	Launch Stack
アフリカ (ケープタウン)	表示	デザイナーで表示	Launch Stack
アジアパシフィック (香港)	表示	デザイナーで表示	Launch Stack
アジアパシフィック (東京)	表示	デザイナーで表示	Launch Stack
アジアパシフィック (ソウル)	表示	デザイナーで表示	Launch Stack
アジアパシフィック (シンガポール)	表示	デザイナーで表示	Launch Stack
アジアパシフィック (シドニー)	表示	デザイナーで表示	Launch Stack
アジアパシフィック (ムンバイ)	表示	デザイナーで表示	Launch Stack

リージョン	ビュー	デザイナーで表示	起動する
中国 (北京)	表示	デザイナーで表示	Launch Stack
中国 (寧夏)	表示	デザイナーで表示	Launch Stack
AWS GovCloud (米国西部)	表示	デザイナーで表示	Launch Stack

2. [Select Template] ページで、[Next] を選択します。
3. [詳細の指定] ページで [次へ] を選択します。
4. [Options(オプション)] ページで、[Next(次へ)] を選択します。
5. [Review] (確認) ページでは、次の 2 つのチェックボックスをオンにする必要があります。

   - 最初のもの AWS CloudFormation は、カスタム名で IAM リソースを作成する場合があることを認めます。
   - 2 つ目は、AWS CloudFormation には新しいスタックの `CAPABILITY_AUTO_EXPAND` が必要になるかもしれないことを認識します。`CAPABILITY_AUTO_EXPAND` は明示的に AWS CloudFormation がスタックの作成時に事前の確認なしにマクロを自動的に展開できるようにします。

     カスタマーは、処理されたテンプレートから変更セットを作成することが多いため、実際にスタックを作成する前にマクロによって行われた変更を確認できます。詳細については、「AWS CloudFormation CreateStack API」を参照してください。

   次に [作成] を選択します。

クイックスタートテンプレートは以下を作成し、設定します。

- Neptune DB クラスター
- 必要な IAM ロール (およびそれらをアタッチする)。
- 必要な Amazon EC2 セキュリティグループ。
- 必要な SageMaker VPC エンドポイント。
- Neptune ML の DB クラスターパラメータグループ。
- そのパラメータグループ内の必要なパラメータ。

- Neptune ML 用のノートブックサンプルが事前に入力された SageMaker ノートブック。すべてのリージョンですべてのインスタンスサイズが使用できるわけではないため、選択したノートブックインスタンスサイズが、リージョンでサポートされているサイズであることを確認する必要があります。
- Neptune-Export サービスとして。

クイックスタートスタックの準備ができたら、テンプレートが作成した SageMaker ノートブックに移動し、あらかじめ入力された例をチェックしてください。これらは、Neptune ML 機能の実験に使用するサンプルデータセットをダウンロードするのに役立ちます。

また、Neptune MLを使用しているときに多くの時間を節約できます。例えば、%neptune_ml ラインマジック、およびこれらのノートブックがサポートする %%neptune_ml セルマジックを参照してください。

また、クイックスタート AWS CloudFormation テンプレートを実行する次の AWS CLI コマンドテンプレートを使用することもできます。

```
aws cloudformation create-stack \
  --stack-name neptune-ml-fullstack-$(date '+%Y-%m-%d-%H-%M') \
  --template-url https://aws-neptune-customer-samples.s3.amazonaws.com/v2/
cloudformation-templates/neptune-ml-nested-stack.json \
  --parameters ParameterKey=EnableIAMAuthOnExportAPI,ParameterValue=(true if you have
 IAM auth enabled, or false otherwise) \
               ParameterKey=Env,ParameterValue=test$(date '+%H%M')\
  --capabilities CAPABILITY_IAM \
  --region (the AWS region, like us-east-1) \
  --disable-rollback \
  --profile (optionally, a named CLI profile of yours)
```

クイックスタート AWS CloudFormation テンプレートを使わずに Neptune ML をセットアップする

1. 動作中の Neptune DB クラスターから始める

AWS CloudFormation クイックスタートテンプレートを使用して Neptune ML を設定しない場合は、既存の Neptune DB クラスターと連携する必要があります。必要な場合は、既存のものを使用することも、すでに使用しているものを複製することも、新しいものを作成することもできます (「DB クラスターを作成する」を参照)。

2. Neptune-Export サービスをインストールする

まだの場合は、Neptune-Export サービスを使用して Neptune データをエクスポートする で説明されているように、Neptune-Export サービスをインストールしてください。

インストールによって作成される `NeptuneExportSecurityGroup` セキュリティグループに、以下の設定でインバウンドルールを追加します。

- タイプ: `Custom TCP`
- プロトコル: `TCP`
- [Port range] (ポート範囲): `80 - 443`
- ソース: *`(Neptune DB ############### ID)`*

3. カスタム **`NeptuneLoadFromS3`** IAM ロールを作成する

まだ作成していない場合は、Amazon S3 にアクセスするための IAM ロールの作成 で説明されているように、カスタム `NeptuneLoadFromS3` IAM ロールを作成します。

カスタム **`NeptuneSageMakerIAMRole`** ロールを作成する

以下のポリシーを使用して、IAM コンソールを使用し、カスタム `NeptuneSageMakerIAMRole` を作成します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ec2:CreateNetworkInterface",
```

```
        "ec2:CreateNetworkInterfacePermission",
        "ec2:CreateVpcEndpoint",
        "ec2:DeleteNetworkInterface",
        "ec2:DeleteNetworkInterfacePermission",
        "ec2:DescribeDhcpOptions",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeRouteTables",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcEndpoints",
        "ec2:DescribeVpcs"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "ecr:BatchCheckLayerAvailability"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "iam:PassRole"
      ],
      "Resource": [
        "arn:aws:iam::*:role/*"
      ],
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": [
            "sagemaker.amazonaws.com"
          ]
        }
      },
      "Effect": "Allow"
    },
    {
      "Action": [
        "kms:CreateGrant",
```

```
        "kms:Decrypt",
        "kms:GenerateDataKey*"
      ],
      "Resource": "arn:aws:kms:*:*:key/*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogGroups",
        "logs:DescribeLogStreams",
        "logs:GetLogEvents"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:/aws/sagemaker/*"
      ],
      "Effect": "Allow"
    },
    {
      "Action": [
        "sagemaker:AddTags",
        "sagemaker:CreateEndpoint",
        "sagemaker:CreateEndpointConfig",
        "sagemaker:CreateHyperParameterTuningJob",
        "sagemaker:CreateModel",
        "sagemaker:CreateProcessingJob",
        "sagemaker:CreateTrainingJob",
        "sagemaker:CreateTransformJob",
        "sagemaker:DeleteEndpoint",
        "sagemaker:DeleteEndpointConfig",
        "sagemaker:DeleteModel",
        "sagemaker:DescribeEndpoint",
        "sagemaker:DescribeEndpointConfig",
        "sagemaker:DescribeHyperParameterTuningJob",
        "sagemaker:DescribeModel",
        "sagemaker:DescribeProcessingJob",
        "sagemaker:DescribeTrainingJob",
        "sagemaker:DescribeTransformJob",
        "sagemaker:InvokeEndpoint",
        "sagemaker:ListTags",
        "sagemaker:ListTrainingJobsForHyperParameterTuningJob",
        "sagemaker:StopHyperParameterTuningJob",
```

```
        "sagemaker:StopProcessingJob",
        "sagemaker:StopTrainingJob",
        "sagemaker:StopTransformJob",
        "sagemaker:UpdateEndpoint",
        "sagemaker:UpdateEndpointWeightsAndCapacities"
      ],
      "Resource": [
        "arn:aws:sagemaker:*:*:*"
      ],
      "Effect": "Allow"
    },
    {
      "Action": [
        "sagemaker:ListEndpointConfigs",
        "sagemaker:ListEndpoints",
        "sagemaker:ListHyperParameterTuningJobs",
        "sagemaker:ListModels",
        "sagemaker:ListProcessingJobs",
        "sagemaker:ListTrainingJobs",
        "sagemaker:ListTransformJobs"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject",
        "s3:AbortMultipartUpload",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

このロールを作成するときに、以下のように信頼関係を編集します。

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "ec2.amazonaws.com",
          "rds.amazonaws.com",
          "sagemaker.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

最後に、この新しい NeptuneSageMakerIAMRole ロールに割り当てられた ARN をコピーします。

> **Important**
>
> - NeptuneSageMakerIAMRole 内の Amazon S3 アクセス許可が上記のものと一致していることを確認してください。
> - ユニバーサル ARN arn:aws:s3:::* は、上記のポリシーの Amazon S3 リソースに使用されます。何らかの理由でユニバーサル ARN を使用できない場合は、arn:aws:s3:::graphlytics* と、NeptuneML コマンドが使用する他のカスタマー Amazon S3 リソースの ARN をリソースセクションに追加する必要があります。

Neptune ML を有効にするように DB クラスターを設定する

Neptune ML の DB クラスターを設定するには

1. Neptune コンソールで、[パラメータグループ] に移動し、次に、使用する DB クラスターに関連付けられた DB クラスターパラメータグループに移動します。neptune_ml_iam_role パラメータを、作成した NeptuneSageMakerIAMRole ロールに割り当てられた ARN に設定します。
2. [データベース] に移動し、Neptune ML に使用する DB クラスターを選択します。[アクション]、[IAM ロールの管理] の順に選択します。

3. [IAM ロールの管理] ページで、[ロールを追加] を選択し、NeptuneSageMakerIAMRole を追加します。次に、NeptuneLoadFromS3 ロールを追加します。
4. DB クラスターのライターインスタンスを再起動します。

Neptune VPC に SageMaker の 2 つのエンドポイントを作成します。

最後に、Neptune エンジンに必要な SageMaker 管理 API へのアクセスを与えるために、Neptune VPC で SageMaker の 2 つのエンドポイントを作成します。 で説明されているように、Neptune VPC に 2 つの SageMaker エンドポイントを作成する必要があります。

Neptune ML 用の Neptune ノートブックの手動設定

Neptune SageMaker ノートブックには、Neptune ML 用のさまざまなサンプルノートブックがプリロードされています。これらのサンプルは、オープンソースのグラフノートブック GitHub リポジトリでプレビューできます。

既存の Neptune ノートブックを使用することも、必要に応じて独自のノートブックを作成することもできます。手順については、Neptune ワークベンチを使用して Neptune ノートブックをホストする を参照してください。

次の手順に従って、デフォルトの Neptune ノートブックを Neptune ML で使用するように設定することもできます。

Neptune ML 用にノートブックを変更する

1. Amazon SageMaker コンソール (https://console.aws.amazon.com/sagemaker/) を開きます。
2. 左側のナビゲーションペインで [ノートブック] を選択し、[ノートブックインスタンス] を選択します。Neptune ML に使用する Neptune ノートブックの名前を探して選択し、詳細ページに移動します。
3. ノートブックインスタンスが実行中の場合は、ノートブックの詳細ページの右上にある [停止] ボタンを選択します。
4. [ノートブックインスタンス設定] の [ライフサイクル設定] で、ノートブックのライフサイクルのページを開くリンクを選択します。
5. 右上の [編集] を選択し、[続行] を選択します。
6. [ノートブックの開始] タブで、スクリプトを変更して追加のエクスポートコマンドを含め、Neptune ML IAM ロールとエクスポートサービス URI のフィールドに入力します。シェルによっては以下のようになります。

```
echo "export NEPTUNE_ML_ROLE_ARN=(your Neptune ML IAM role ARN)" >> ~/.bashrc
echo "export NEPTUNE_EXPORT_API_URI=(your export service URI)" >> ~/.bashrc
```

7. [Update] (更新) を選択します。
8. ノートブックインスタンスページに戻ります。[アクセス許可と暗号化] に、[IAM ロール ARN] のフィールドがあります。このフィールドのリンクを選択して、このノートブックインスタンスが実行される IAM ロールに移動します。
9. 次のような新しいインラインポリシーを作成します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Resource": "arn:aws:cloudwatch:[AWS_REGION]:[AWS_ACCOUNT_ID]:*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:DescribeLogStreams",
        "logs:PutLogEvents",
        "logs:GetLogEvents"
      ],
      "Resource": "arn:aws:logs:[AWS_REGION]:[AWS_ACCOUNT_ID]:*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "s3:Put*",
        "s3:Get*",
        "s3:List*"
      ],
      "Resource": "arn:aws:s3:::*",
      "Effect": "Allow"
    },
    {
      "Action": "execute-api:Invoke",
```

```
      "Resource": "arn:aws:execute-api:[AWS_REGION]:[AWS_ACCOUNT_ID]:*/*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "sagemaker:CreateModel",
        "sagemaker:CreateEndpointConfig",
        "sagemaker:CreateEndpoint",
        "sagemaker:DescribeModel",
        "sagemaker:DescribeEndpointConfig",
        "sagemaker:DescribeEndpoint",
        "sagemaker:DeleteModel",
        "sagemaker:DeleteEndpointConfig",
        "sagemaker:DeleteEndpoint"
      ],
      "Resource": "arn:aws:sagemaker:[AWS_REGION]:[AWS_ACCOUNT_ID]:*/*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "iam:PassRole"
      ],
      "Resource": "[YOUR_NEPTUNE_ML_IAM_ROLE_ARN]"",
      "Effect": "Allow"
    }
  ]
}
```

10. この新しいポリシーを保存し、ステップ 8 で IAM ロールにアタッチします。
11. SageMaker ノートブックインスタンスの詳細ページの右上にある [開始] を選択して、ノートブックインスタンスを起動します。

AWS CLI を使用して DB クラスターに Neptune ML をセットアップする

AWS CloudFormation クイックスタートテンプレートと AWS Management Console に加えて、AWS CLI を使用して Neptune ML を設定することもできます。

1. 新しい Neptune ML クラスター用の DB クラスターパラメータグループを作成します。

以下の AWS CLI コマンドは、新しい DB クラスターパラメータグループを作成し、Neptune ML で動作するように設定します。

Neptune ML 用の DB クラスターパラメータグループを作成して設定するには

1. 新しい DB クラスターのパラメータグループを作成します。

```
aws neptune create-db-cluster-parameter-group \
  --db-cluster-parameter-group-name (name of the new DB cluster parameter group) \
  --db-parameter-group-family neptune1
  --description "(description of your machine learning project)" \
  --region (AWS region, such as us-east-1)
```

2. neptune_ml_iam_role DB クラスターパラメータを作成し、SageMakerExcecutionIAMRole のARN に設定します。SageMaker を呼び出しホストされた ML モデルからジョブを作成し、予測を取得する際に DB クラスターを使用するためです。

```
aws neptune modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name (name of the new DB cluster parameter group) \
  --parameters "ParameterName=neptune_ml_iam_role, \
                ParameterValue=ARN of the SageMakerExcecutionIAMRole, \
                Description=NeptuneMLRole, \
                ApplyMethod=pending-reboot" \
  --region (AWS region, such as us-east-1)
```

   このパラメータを設定すると、Neptune はコールごとにロールを渡す必要なく SageMaker にアクセスできます。

   SageMakerExcecutionIAMRole を作成する方法については、カスタム NeptuneSageMakerIAMRole ロールを作成する を参照してください。

3. 最後に、新しい DB クラスターパラメーターグループのすべてのパラメータが希望通りに設定されていることを確認するには、describe-db-cluster-parameters を実行します。

```
aws neptune describe-db-cluster-parameters \
  --db-cluster-parameter-group-name (name of the new DB cluster parameter group) \
  --region (AWS region, such as us-east-1)
```

Neptune ML で使用する DB クラスターに新しい DB クラスターパラメータグループをアタッチします。

これで、次のコマンドを使用して、作成した新しい DB クラスターパラメータグループを既存の DB クラスターにアタッチできます。

```
aws neptune modify-db-cluster \
  --db-cluster-identifier (the name of your existing DB cluster) \
  --apply-immediately
  --db-cluster-parameter-group-name (name of your new DB cluster parameter group) \
  --region (AWS region, such as us-east-1)
```

すべてのパラメータを有効にするために、DB クラスターを再起動します。

```
aws neptune reboot-db-instance
  --db-instance-identifier (name of the primary instance of your DB cluster) \
  --profile (name of your AWS profile to use) \
  --region (AWS region, such as us-east-1)
```

または、Neptune ML で使用する新しい DB クラスターを作成する場合は、次のコマンドを使用して新しいパラメータグループがアタッチされたクラスターを作成し、新しいプライマリ (ライター) インスタンスを作成できます。

```
cluster-name=(the name of the new DB cluster)
aws neptune create-db-cluster
  --db-cluster-identifier ${cluster-name}
  --engine graphdb \
  --engine-version 1.0.4.1 \
  --db-cluster-parameter-group-name (name of your new DB cluster parameter group) \
  --db-subnet-group-name (name of the subnet to use) \
  --region (AWS region, such as us-east-1)

aws neptune create-db-instance
  --db-cluster-identifier ${cluster-name}
  --db-instance-identifier ${cluster-name}-i \
```

```
  --db-instance-class (the instance class to use, such as db.r5.xlarge)
  --engine graphdb \
  --region (AWS region, such as us-east-1)
```

SageMaker および Amazon S3 リソースにアクセスできるように DB クラスターに **NeptuneSageMakerIAMRole** をアタッチする

最後に、カスタム NeptuneSageMakerIAMRole ロールを作成する に記載されている手順に従って、DB クラスターが SageMaker および Amazon S3 と通信できるようにする IAM ロールを作成します。次に、次のコマンドを使用して、作成した NeptuneSageMakerIAMRole ロールを DB クラスターにアタッチします。

```
aws neptune add-role-to-db-cluster
  --db-cluster-identifier ${cluster-name}
  --role-arn arn:aws:iam::(the ARN number of the role's ARN):role/NeptuneMLRole \
  --region (AWS region, such as us-east-1)
```

Neptune VPC で SageMaker の 2 つのエンドポイントを作成します。

Neptune ML は、Neptune DB クラスターの VPC に 2 つの SageMaker エンドポイントを必要とします。

- com.amazonaws.*(AWS region, like us-east-1)*.sagemaker.runtime
- com.amazonaws.*(AWS region, like us-east-1)*.sagemaker.api

クイックスタート AWS CloudFormation テンプレートを使用したことがない場合は自動的に作成されます。それらを作成する次の AWS CLI コマンドを使用できます。

これは、sagemaker.runtime エンドポイントを作成します。

```
create-vpc-endpoint
  --vpc-id (the ID of your Neptune DB cluster's VPC)
  --service-name com.amazonaws.(AWS region, like us-east-1).sagemaker.runtime
  --subnet-ids (the subnet ID or IDs that you want to use)
  --security-group-ids (the security group for the endpoint network interface, or omit
 to use the default)
  --private-dns-enabled
```

そしてこれは、sagemaker.api エンドポイントを作成します。

```
aws create-vpc-endpoint
  --vpc-id (the ID of your Neptune DB cluster's VPC)
  --service-name com.amazonaws.(AWS region, like us-east-1).sagemaker.api
  --subnet-ids (the subnet ID or IDs that you want to use)
  --security-group-ids (the security group for the endpoint network interface, or omit
 to use the default)
  --private-dns-enabled
```

また、VPC コンソール を使用して、これらのエンドポイントを作成することもできます。Amazon SageMaker での AWS PrivateLink によるセキュアな予測コールおよびすべての Amazon SageMaker API コールを AWS PrivateLink で保護するを参照してください。

DB クラスターパラメータグループに SageMaker 推論エンドポイントパラメータを作成します。

すべてのクエリで使用しているモデルの SageMaker 推論エンドポイントを指定する必要がないようにするには、Neptune ML の DB クラスターパラメータグループで neptune_ml_endpoint という名前の DB クラスターパラメータを作成します。パラメータを、対象のインスタンスエンドポイントの id に設定します。

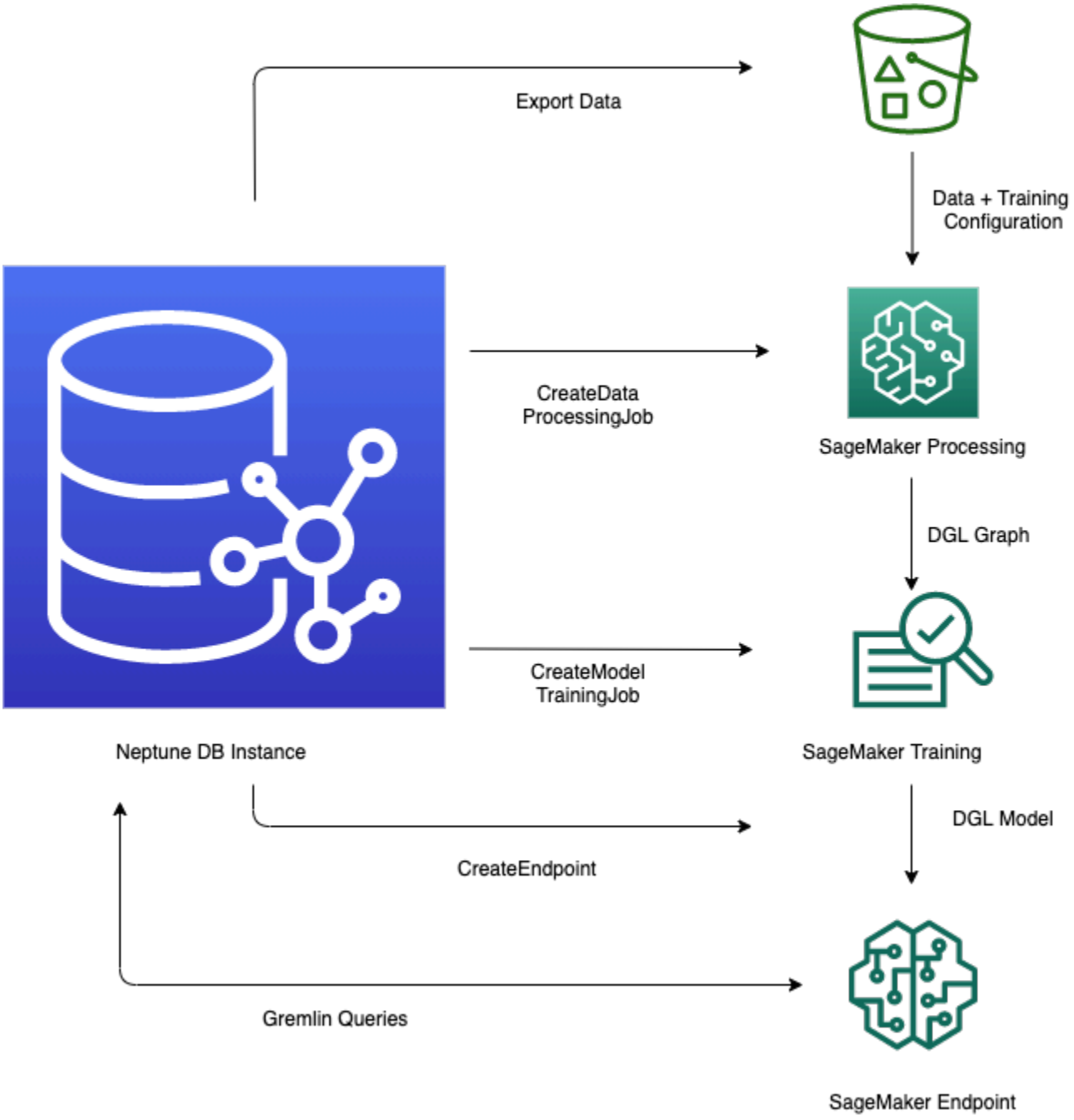
これを行うには、次の AWS CLI コマンドを使用できます。

```
aws neptune modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name neptune-ml-demo \
  --parameters "ParameterName=neptune_ml_endpoint, \
                ParameterValue=(the name of the SageMaker inference endpoint you want
 to query), \
                Description=NeptuneMLEndpoint, \
                ApplyMethod=pending-reboot" \
  --region (AWS region, such as us-east-1)
```

Neptune ML 特徴の使い方の概要

Neptune ML を使用するためのワークフローの開始

Amazon Neptune で Neptune ML 特徴を使用するには、通常、まず次の 5 つの手順が必要です。



1. データのエクスポートと設定 — データエクスポートステップでは、Neptune-Export サービスまたはNeptune から Amazon Simple Storage Service (Amazon S3) にデータを CSV 形式でエクスポートするための `neptune-export` コマンドラインツールを使用します。`training-data-`

`configuration.json` という構成ファイルが同時に自動的に生成されます。これにより、エクスポートされたデータをトレーニング可能なグラフにロードする方法が指定されます。

2. データの前処理 — このステップでは、エクスポートされたデータセットは標準的な手法を使用して前処理され、モデルトレーニング用に準備されます。数値データに対して特徴の正規化を実行でき、テキスト特徴は `word2vec` を使用して符号化できます。この手順の最後に、エクスポートされたデータセットから DGL (ディープグラフライブラリ) グラフが生成され、モデルトレーニングステップで使用できるようになります。

   この手順は、アカウントの SageMaker 処理ジョブを使用して実装され、結果のデータは指定した Amazon S3 ロケーションに保存されます。

3. モデルトレーニング — モデルトレーニングステップは、予測に使用される機械学習モデルをトレーニングします。

   モデルトレーニングは、次の2つの段階で行われます。

   - 最初のステージでは、SageMaker 処理ジョブを使用して、モデルトレーニングに使用するモデルおよびモデルのハイパーパラメータ範囲のタイプを指定するモデルトレーニング戦略構成セットを生成します。
   - 次に、第 2 段階では SageMaker モデルチューニングジョブを使用して、さまざまなハイパーパラメータ構成を試し、最もパフォーマンスの高いモデルを生成したトレーニングジョブを選択します。チューニングジョブは、処理されたデータに対して事前に指定された数のモデルトレーニングジョブ試行を実行します。このステージの最後に、最適なトレーニングジョブのトレーニング済みモデルパラメータを使用して、推論のためのモデルアーティファクトを生成します。

4. Amazon SageMaker で推論エンドポイントを作成する — 推論エンドポイントは、最適なトレーニングジョブによって生成されたモデルアーティファクトで起動される SageMaker エンドポイントインスタンスです。各モデルは 1 つのエンドポイントに関連付けられています。エンドポイントは、グラフデータベースからの受信リクエストを受け入れ、リクエストの入力に対するモデル予測を返すことができます。エンドポイントを作成した後も、削除するまでアクティブなままになります。

5. Gremlin を使用して機械学習モデルを照会する — Gremlin クエリ言語の拡張機能を使用して、推論エンドポイントから予測を照会できます。

> **Note**
>
> Neptune workbench ラインマジックとセルマジックが含まれており、これらの手順の管理に多大な時間を節約できます。

- %neptune_ml
- %%neptune_ml

変化するグラフデータに基づいて予測を行う

継続的に変化するグラフでは、新しいデータを使用して、定期的に新しいバッチ予測を作成する必要があるかもしれません。最新のデータに基づいて新しい予測をその場で生成する (帰納的推論) よりも、事前に計算された予測 (トランスダクティブ推論) をクエリするほうがはるかに高速です。どちらの方法にも、データの変化の速さやパフォーマンス要件に応じて弱点があります。

帰納的推論とトランスダクティブ推論の違い

トランスダクティブ推論を実行する場合、Neptune はトレーニング時に事前に計算された予測を検索して返します。

帰納的推論を実行する場合、Neptune は関連するサブグラフを作成し、そのプロパティを取得します。次に、DGL GNN モデルはデータ処理とモデル評価をリアルタイムで適用します。

そのため、帰納的推論では、トレーニング時には存在しなかったノードやエッジに関する予測や、グラフの現在の状態を反映した予測を生成できます。ただし、これにはレイテンシが大きくなるという代償が伴います。

グラフが動的な場合は、最新のデータを考慮に入れるために帰納的推論を使用することもできますが、グラフが静的な場合は、トランスダクティブ推論の方が速くて効率的です。

帰納的推論は、デフォルトでは無効です。クエリで Gremlin Neptune#ml.inductiveInference 述語を次のように使用することで、クエリで有効にできます。

```
.with( "Neptune#ml.inductiveInference")
```

インクリメンタルトランスダクティブワークフロー

ステップ 1 ~ 3 (データのエクスポートと設定からモデル変換まで) を再実行するだけで、モデルアーティファクトを更新できますが、Neptune ML は、新しいデータを使用してバッチ ML 予測を更新する、より簡単な方法をサポートしています。1 つは、インクリメンタルモデルワークフローを使用する方法で、もう 1 つはウォームスタートによるモデルの再トレーニングです。

インクリメンタルモデルワークフロー

このワークフローでは、ML モデルを再トレーニングせずに ML 予測を更新します。

> **Note**
>
> これは、グラフデータが新しいノードや エッジで更新された場合にのみ実行できます。ノードが削除されると、今のところ動作しません。

1. データのエクスポートと設定 — このステップは、メインワークフローと同じです。
2. 増分データの前処理 — このステップは、メインワークフローのデータ前処理ステップと似ていますが、特定のトレーニング済みモデルに対応する、以前に使用したのと同じ処理構成を使用します。
3. モデルの変換 — モデルトレーニングステップではなく、このモデル変換ステップは、メインワークフローと増分データの前処理ステップの結果からトレーニング済みモデルを取得し、推論に使用する新しいモデルアーティファクトを生成します。モデル変換ステップは SageMaker 処理ジョブを起動して、更新されたモデルアーティファクトを生成する計算を実行します。
4. Amazon SageMaker 推論エンドポイントを更新する — オプションで、既存の推論エンドポイントがある場合、このステップでは、モデル変換ステップによって生成された新しいモデルアーティファクトでエンドポイントを更新します。または、新しいモデルのアーティファクトを使用して新しい推論エンドポイントを作成することもできます。

ウォームスタートでモデル再トレーニング

このワークフローを使用して、増分グラフデータを使用して予測を行うための新しい ML モデルをトレーニングおよび展開できますが、メインワークフローを使用して生成された既存のモデルから開始します。

1. データのエクスポートと設定 — このステップは、メインワークフローと同じです。
2. 増分データの前処理 — この手順は、増分モデル推論ワークフローの場合と同じです。新しいグラフデータは、以前にモデルトレーニングに使用したのと同じ処理方法で処理する必要があります。
3. ウォームスタートでモデルトレーニング — モデルトレーニングはメインワークフローで発生するものと似ていますが、前のモデルトレーニングタスクの情報を活用して、モデルのハイパーパラメータ検索を高速化できます。

4. Amazon SageMaker 推論エンドポイントを更新する — この手順は、増分モデル推論ワークフローの場合と同じです。

Neptune ML のカスタムモデルのワークフロー

Neptune ML では、Neptune ML がサポートする任意のタスクに対して、独自のカスタムモデルを実装、トレーニング、デプロイできます。カスタムモデルの開発およびデプロイのワークフローは、組み込みモデルの場合と基本的に同じですが、カスタムモデルのワークフロー で説明するように、いくつかの違いがあります。

Neptune ML ステージのインスタンス選択

Neptune ML 処理のさまざまなステージでは、異なる SageMaker インスタンスが使用されます。ここでは、ステージごとに適切なインスタンスタイプを選択する方法について説明します。SageMaker インスタンスタイプと価格については、Amazon SageMaker 料金表を参照してください。

データ処理用のインスタンスの選択

SageMaker データ処理ステップには入力、中間、および出力データを格納するための十分なメモリとディスクストレージがある処理インスタンスを要します。必要なメモリとディスクストレージの特定の量は、Neptune ML グラフの特性とエクスポートされた機能によって異なります。

デフォルトでは、Neptune ML は、ディスク上にエクスポートされたグラフデータのサイズの 10 倍のメモリを持つ最も小さい `ml.r5` インスタンスを選択します。。

モデルトレーニングとモデル変換のインスタンスの選択

タスクタイプ、グラフサイズ、およびターンアラウンド要件によって異なる、モデルトレーニングまたはモデルの変換の適切なインスタンスタイプの選択。GPU インスタンスは、最高のパフォーマンスを提供します。一般的に `p3` および `g4dn` シリアルインスタンスを推奨します。また、`p2` または `p4d` インスタンスもご使用いただけます。

デフォルトでは、Neptune ML はモデルトレーニングとモデル変換が必要とするよりも多くのメモリを持つ最小の GPU インスタンスを選択します。その選択内容については、Amazon S3 データ処理の出力場所に格納されている `train_instance_recommendation.json` ファイルに説明されています。この `train_instance_recommendation.json` ファイルの内容の例を示します。

```
{
  "instance":     "(the recommended instance type for model training and transform)",
  "cpu_instance": "(the recommended instance type for base processing instance)",
  "disk_size":    "(the estimated disk space required)",
  "mem_size":     "(the estimated memory required)"
}
```

推論エンドポイントのインスタンスを選択する

タスクタイプ、グラフサイズ、予算に応じた推論エンドポイントに適切なインスタンスタイプの選択。デフォルトでは、Neptune ML は推論エンドポイントが必要とするメモリが多い最小の `ml.m5d` インスタンスを選択します。

> **Note**
>
> 384 GB 以上のメモリが必要な場合は、Neptune MLは `ml.r5d.24xlarge` インスタンスを使用します。

Neptune ML が推奨するインスタンスタイプは、モデルトレーニングに使用している Amazon S3 の場所にある `infer_instance_recommendation.json` ファイルに記載されています。このファイルの内容例を示します。

```
{
  "instance" :   "(the recommended instance type for an inference endpoint)",
  "disk_size" :  "(the estimated disk space required)",
  "mem_size" :   "(the estimated memory required)"
}
```

neptune-export ツールまたは Neptune-Export サービスを使用して Neptune ML 用に Neptune からデータをエクスポートする

Neptune ML では、ディープグラフライブラリ (DGL) にトレーニングデータを提供して、モデルを作成および評価する必要があります。

Neptune-Export サービス または neptune-export ユーティリティのいずれかの方法で、Neptune からデータをエクスポートできます。。サービスとコマンドラインツールの両方は、Amazon S3 サーバー側の暗号化 (SSE-S3) により暗号化された Amazon Simple Storage Service (Amazon S3) に CSV 形式でデータを公開します。「Neptune-Export と neptune-export でエクスポートされたファイル」を参照してください。

さらに、Neptune ML のトレーニングデータのエクスポートを構成すると、エクスポートジョブは、エクスポートされたデータとともに暗号化されたモデルトレーニング構成ファイルを作成し、発行します。デフォルトでは、このファイルには training-data-configuration.json という名前が付けられます。

Neptune-Export サービスを使用してトレーニングデータをNeptune ML へエクスポートする例

このリクエストは、ノード分類タスクのプロパティグラフトレーニングデータをエクスポートします。

```
curl \
  (your NeptuneExportApiUri) \
  -X POST \
  -H 'Content-Type: application/json' \
  -d '{
        "command": "export-pg",
        "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export",
        "params": {
          "endpoint": "(your Neptune endpoint DNS name)",
          "profile": "neptune_ml"
        },
        "additionalParams": {
          "neptune_ml": {
            "version": "v2.0",
            "targets": [
              {
                "node": "Movie",
```

```
                "property": "genre",
                "type": "classification"
              }
            ]
          }
        }
      }'
```

このリクエストは、ノード分類タスクの RDF トレーニングデータをエクスポートします。

```
curl \
  (your NeptuneExportApiUri) \
  -X POST \
  -H 'Content-Type: application/json' \
  -d '{
        "command": "export-rdf",
        "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export",
        "params": {
          "endpoint": "(your Neptune endpoint DNS name)",
          "profile": "neptune_ml"
        },
        "additionalParams": {
          "neptune_ml": {
            "version": "v2.0",
            "targets": [
              {
                "node": "http://aws.amazon.com/neptune/csv2rdf/class/Movie",
                "predicate": "http://aws.amazon.com/neptune/csv2rdf/datatypeProperty/
genre",
                "type": "classification"
              }
            ]
          }
        }
      }'
```

トレーニングデータをエクスポートするとき **params** オブジェクトで設定するフィールド

エクスポートリクエストの params オブジェクトには、params ドキュメンテーションで説明されているように、さまざまなフィールドを含めることができます。次のものは、機械学習トレーニングデータのエクスポートに最も関連性があります。

- **endpoint** — endpoint を使用してエクスポートプロセスでクエリを実行してデータを抽出できる DB クラスター内の Neptune インスタンスのエンドポイントを指定します。
- **profile** — params オブジェクト内の profile フィールドは **neptune-ml** に設定する必要があります。

  これにより、エクスポートプロセスでは、エクスポートされたデータが Neptune ML モデルトレーニング、プロパティグラフデータ用の CSV 形式、または RDF データの場合は N トリプルとして適切にフォーマットされます。また、エクスポートされたトレーニングデータと同じ Amazon S3 の場所にファイル training-data-configuration.json が作成されび書き込まれます。
- **cloneCluster** — true に設定されている場合、エクスポートプロセスによって DB クラスターのクローンが作成され、クローンからエクスポートされ、完了するとクローンが削除されます。
- **useIamAuth** — DB クラスターが IAM 認証有効となっている場合は、このフィールドを true に設定する必要があります。

エクスポートプロセスでは、エクスポートするデータをフィルタリングする方法もいくつかあります(これらの例を参照)。

モデルトレーニング情報のエクスポートを調整する **additionalParams** オブジェクトの使用

additionalParams オブジェクトには、トレーニング目的で機械学習クラスのラベルと特徴を指定し、トレーニングデータ設定ファイルの作成をガイドするために使用できるフィールドが含まれています。

エクスポートプロセスでは、トレーニング用の例として使用するために、機械学習クラスラベルにするノードとエッジプロパティを自動的に推論することはできません。また、数値、カテゴリ、およびテキストプロパティの最適な特徴エンコーディングを自動的に推論することもできないため、additionalParams オブジェクトのフィールドを使用してこれらを指定するか、デフォルトのエンコーディングを上書きします。

プロパティグラフデータの場合、エクスポートリクエストにおける additionalParams の最上位構造は次のとおりです。

```
{
  "command": "export-pg",
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export",
  "params": {
```

```
    "endpoint": "(your Neptune endpoint DNS name)",
    "profile": "neptune_ml"
  },
  "additionalParams": {
      "neptune_ml": {
        "version": "v2.0",
        "targets": [ (an array of node and edge class label targets) ],
        "features": [ (an array of node feature hints) ]
    }
  }
}
```

RDF データの場合、最上位の構造は次のようになります。

```
{
  "command": "export-rdf",
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "profile": "neptune_ml"
  },
  "additionalParams": {
      "neptune_ml": {
        "version": "v2.0",
        "targets": [ (an array of node and edge class label targets) ]
    }
  }
}
```

また、jobs フィールドを使用して複数のエクスポート設定を指定することもできます。

```
{
  "command": "export-pg",
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "profile": "neptune_ml"
  },
  "additionalParams" : {
    "neptune_ml" : {
      "version": "v2.0",
      "jobs": [
        {
```

```
          "name" : "(training data configuration name)",
          "targets": [ (an array of node and edge class label targets) ],
          "features": [ (an array of node feature hints) ]
        },
        {
          "name" : "(another training data configuration name)",
          "targets": [ (an array of node and edge class label targets) ],
          "features": [ (an array of node feature hints) ]
        }
      ]
    }
  }
}
```

additionalParams の neptune_ml フィールド内の最上位要素

neptune_ml の version 要素

生成するトレーニングデータ設定のバージョンを指定します。

(オプション)、タイプ: 文字列、デフォルト値:"v2.0"。

version を含める場合、それを v2.0 に設定します。

neptune_ml の jobs フィールド

トレーニングデータ構成オブジェクトの配列を格納し、それぞれがデータ処理ジョブを定義し、以下を含みます。

- **name** — 作成するトレーニングデータ構成の名前。

  たとえば、「job-number-1」という名前のトレーニングデータ構成では、job-number-1.json という名前のトレーニングデータ構成ファイルが作成されます。
- **targets** — トレーニング用の機械学習クラスラベルを表すノードおよびエッジクラスラベルターゲットの JSON 配列。「neptune_ml オブジェクトの targets フィールド」を参照してください。
- **features** — ノードプロパティ特徴の JSON 配列。「neptune_ml の features フィールド」を参照してください。

neptune_ml オブジェクトの targets フィールド

JSON トレーニングデータエクスポート設定の targets フィールドには、トレーニングタスク、およびこのタスクを学習するための機械学習クラスラベルを指定するターゲットオブジェクトの配列が含まれます。ターゲットオブジェクトの内容は、プロパティグラフデータと RDF データのどちらについてトレーニングしているかによって異なります。

プロパティグラフノードの分類と回帰タスクでは、配列内のターゲットオブジェクトは次のようになります。

```
{
  "node": "(node property-graph label)",
  "property": "(property name)",
  "type" : "(used to specify classification or regression)",
  "split_rate": [0.8,0.2,0.0],
  "separator": ","
}
```

プロパティグラフのエッジ分類、回帰、またはリンク予測タスクでは、次のようになります。

```
{
  "edge": "(edge property-graph label)",
  "property": "(property name)",
  "type" : "(used to specify classification, regression or link_prediction)",
  "split_rate": [0.8,0.2,0.0],
  "separator": ","
}
```

RDF 分類と回帰タスクでは、配列内のターゲットオブジェクトは次のようになります。

```
{
  "node": "(node type of an RDF node)",
  "predicate": "(predicate IRI)",
  "type" : "(used to specify classification or regression)",
  "split_rate": [0.8,0.2,0.0]
}
```

RDF リンク予測タスクの場合、配列内のターゲットオブジェクトは次のようになります。

```
{
```

```
  "subject": "(source node type of an edge)",
  "predicate": "(relation type of an edge)",
  "object": "(destination node type of an edge)",
  "type" : "link_prediction",
  "split_rate": [0.8,0.2,0.0]
}
```

ターゲットオブジェクトには、次のフィールドを含めることができます。

目次

プロパティグラフターゲットオブジェクトのフィールド

ターゲットオブジェクトの **node** (頂点) フィールド

ターゲットノード (頂点) のプロパティグラフラベル。ターゲットオブジェクトには、node 要素または edge 要素を含めますが、両方ではありません。

ある node は次のように、単一の値を取ることができます。

```
  "node": "Movie"
```

または、マルチラベル頂点の場合、次のように値の配列をとることができます。

```
  "node": ["Content", "Movie"]
```

プロパティグラフターゲットオブジェクトの **edge** フィールド

ターゲットエッジを、開始ノードラベル、独自のラベル、および終了ノードラベルで指定します。ターゲットオブジェクトには、edge 要素または node 要素を含めますが、両方ではありません。

edge フィールドの値は、開始ノードのプロパティグラフラベル、エッジ自体のプロパティグラフラベル、および終了ノードのプロパティグラフラベルを表す 3 つの文字列からなる JSON 配列です。

```
  "edge": ["Person_A", "knows", "Person_B"]
```

開始ノードや終了ノードに複数のラベルがある場合は、次のように配列で囲みます。

```
  "edge": [ ["Admin", Person_A"], "knows", ["Admin", "Person_B"] ]
```

プロパティグラフターゲットオブジェクトの **property** フィールド

ターゲットの頂点またはエッジのプロパティを次のように指定します。

```
  "property" : "rating"
```

このフィールドは、ターゲットタスクがリンク予測の場合を除き、必須です。

プロパティグラフターゲットオブジェクトの **type** フィールド

次のように、node または edge で実行されるターゲットタスクのタイプを示します。

```
  "type" : "regression"
```

ノードでサポートされているタスクタイプは次のとおりです。

- `classification`
- `regression`

エッジでサポートされているタスクタイプは次のとおりです。

- classification
- regression
- link_prediction

このフィールドは必須です。

プロパティグラフターゲットオブジェクトの **split_rate** フィールド

(オプション) トレーニング、検証、テストの各段階でそれぞれ使用するノードまたはエッジの比率の推定。これらの比率は、ゼロから 1 までの間の 3 つの数値の JSON 配列で表されます。

```
"split_rate": [0.7, 0.1, 0.2]
```

オプション split_rate フィールドを指定しない場合、デフォルトの推定値は [0.9, 0.1, 0.0] です。

プロパティグラフターゲットオブジェクトの **separator** フィールド

(オプション) 分類タスクで使用します。

separator フィールドは、文字列に複数のカテゴリ値を格納する場合に、ターゲットプロパティ値を複数のカテゴリ値に分割するために使用する文字を指定します。例:

```
"separator": "|"
```

separator フィールドの存在は、そのタスクがマルチターゲット分類タスクであることを示します。

RDF ターゲットオブジェクトのフィールド

RDF ターゲットオブジェクトの **node** フィールド

ターゲットノードのノードタイプを定義します。ノード分類タスクまたはノード回帰タスクで使用されます。RDF のノードのノードタイプは、以下で定義されます。

```
  node_id, <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>, node_type
```

RDF node は次のように、単一の値を取ることができます。

```
  "node": "http://aws.amazon.com/neptune/csv2rdf/class/Movie"
```

RDF ターゲットオブジェクトの **subject** フィールド

リンク予測タスクの場合、subject はターゲットエッジの始点ノードタイプを定義します。

```
  "subject": "http://aws.amazon.com/neptune/csv2rdf/class/Director"
```

> **Note**
>
> リンク予測タスクの場合、subject は predicate および object とともに使用する必要があります。これら 3 つのいずれかが指定されていない場合、すべてのエッジがトレーニングターゲットとして扱われます。

RDF ターゲットオブジェクトの **predicate** フィールド

ノード分類およびノード回帰タスクの場合、predicate はターゲットノードのターゲットノード特徴として使用されるリテラルデータを定義します。

```
  "predicate": "http://aws.amazon.com/neptune/csv2rdf/datatypeProperty/genre"
```

> **Note**
>
> ターゲットノードにターゲットノードの特徴を定義する述語が 1 つしかない場合は、predicate フィールドは省略できます。

リンク予測タスクの場合、predicate はターゲットエッジの関連タイプを定義します。

```
"predicate": "http://aws.amazon.com/neptune/csv2rdf/datatypeProperty/direct"
```

> **Note**
>
> リンク予測タスクの場合、predicate は subject および object とともに使用する必要があります。これら 3 つのいずれかが指定されていない場合、すべてのエッジがトレーニングターゲットとして扱われます。

RDF ターゲットオブジェクトの **object** フィールド

リンク予測タスクの場合、object はターゲットエッジの終点ノードタイプを定義します。

```
  "object": "http://aws.amazon.com/neptune/csv2rdf/class/Movie"
```

> **Note**
>
> リンク予測タスクの場合、object は subject および predicate とともに使用する必要があります。これら 3 つのいずれかが指定されていない場合、すべてのエッジがトレーニングターゲットとして扱われます。

RDF ターゲットオブジェクトの **type** フィールド

次のように、実行するターゲットタスクのタイプを示します。

```
  "type" : "regression"
```

RDF データでサポートされているタスクタイプは次のとおりです。

- link_prediction
- classification
- regression

このフィールドは必須です。

プロパティグラフターゲットオブジェクトの **split_rate** フィールド

(オプション) トレーニング、検証、テストの各段階でそれぞれ使用するノードまたはエッジの比率の推定。これらの比率は、ゼロから 1 までの間の 3 つの数値の JSON 配列で表されます。

```
"split_rate": [0.7, 0.1, 0.2]
```

オプション split_rate フィールドを指定しない場合、デフォルトの推定値は [0.9, 0.1, 0.0] です。

neptune_ml の features フィールド

プロパティ値と RDF リテラルには、さまざまな形式とデータ型があります。機械学習で優れたパフォーマンスを実現するには、これらの値を特徴として知られる数値エンコーディングに変換することが不可欠です。

Neptune ML は、Neptune ML での特徴エンコーディング で説明されているように、データのエクスポートおよびデータ処理手順の一部として、特徴抽出とエンコーディングを実行します。

プロパティグラフデータセットの場合、エクスポートプロセスにより、文字列プロパティおよび複数の値を含む数値プロパティの auto 特徴が自動的に推論されます。単一の値を含む数値プロパティの場合、numerical 特徴を推論します。します。日付のプロパティについては、datetime 特徴を推論します。

自動推論特徴仕様をオーバーライドする場合、またはプロパティにバケット数値、TF-IDF、FastText、または SBERT 仕様を追加する場合は、特徴フィールドを使用して特徴のエンコーディングを制御できます。

> **Note**
>
> 使用できるのは features フィールドのみで、RDF データではなく、プロパティグラフデータの特徴仕様を制御します。

自由形式のテキストの場合、Neptune ML はいくつかの異なるモデルを使用して、文字列プロパティ値内のトークンのシーケンスを固定サイズの実数値ベクトルに変換できます。

- text_fasttext — fastText エンコーディングを使用します。これは、fastText がサポートする 5 つの言語のうちの 1 つだけを使用する機能に推奨されるエンコーディングです。
- text_sbert — センテンス BERT (SBERT) エンコーディングモデルを使用します。これは、text_fasttext がサポートしていないテキストについて推奨されるエンコーディングです。
- text_word2vec — Google が最初に公開した Word2Vec アルゴリズムを使用して、テキストをエンコードします。Word2Vec は英語のみをサポートしています。
- text_tfidf — term frequency–inverse document frequency (TF-IDF) ベクタライザを使用して、テキストのエンコードを行います。TF-IDF エンコーディングは、他のエンコーディングにはない統計機能をサポートします。

`features` フィールドには、ノードプロパティ特徴の JSON 配列が含まれます。ターゲットオブジェクトには、次のフィールドを含めることができます。

目次

`features` の `node` フィールド

node フィールドは、特徴頂点のプロパティグラフラベルを指定します。例:

```
  "node": "Person"
```

頂点に複数のラベルがある場合は、配列を使用してそれらを含めます。例:

```
  "node": ["Admin", "Person"]
```

`features` の `edge` フィールド

edge フィールドは、特徴エッジのエッジタイプを指定します。エッジタイプは、始点頂点のプロパティグラフラベル、エッジのプロパティグラフラベル、および終点頂点のプロパティグラフラベルを含む配列で構成されます。エッジ特徴を指定するときは、3 つの値すべてを指定する必要があります。例:

```
  "edge": ["User", "reviewed", "Movie"]
```

エッジタイプの始点または終点頂点に複数のラベルがある場合は、別の配列を使用してラベルを格納します。例:

```
  "edge": [["Admin", "Person"]. "edited", "Post"]
```

`features` の `property` フィールド

プロパティパラメータを使用して、node パラメータで識別される頂点のプロパティを指定します。例:

```
  "property" : "age"
```

特徴の `type` フィールドで使用できる値

type パラメータは、定義するフィーチャのタイプを指定します。例:

```
  "type": "bucket_numerical"
```

`type` パラメータの使用できる値

- **`"auto"`** — Neptune ML がプロパティタイプを自動的に検出し、適切な特徴エンコーディングを適用するように指定します。auto 特徴には、オプションで separator フィールドも使用できます。

  「Neptune ML での自動特徴エンコーディング」を参照してください。
- **`"category"`** — この特徴エンコーディングは、プロパティ値をいくつかのカテゴリの 1 つとして表します。つまり、特徴は 1 つ以上の離散値を取ることができます。category 特徴には、オプションで separator フィールドも使用できます。

  「Neptune MLのカテゴリ別特徴」を参照してください。

- **"numerical"** — この特徴エンコーディングは、数値プロパティ値を「より大きい」と「より小さい」が意味を持つ連続間隔の数値として表します。

  numerical 特徴には、オプションで norm、imputer、separator フィールドも使用できます。

  「Neptune MLのカテゴリ別特徴」を参照してください。
- **"bucket_numerical"** — この特徴エンコーディングは、数値のプロパティ値をバケットまたはカテゴリのセットに分割します。

  たとえば、人の年齢を、子供 (0 ~20 歳)、若年大人 (20 ~ 40 歳)、中年 (40 ~ 60 歳)、および高齢者 (60歳以上) の 4 つのバケットで符号化できます。

  bucket_numerical 特徴には range と bucket_cnt フィールドが必須で、オプションで、imputer および/または slide_window_size フィールドを使用できます。

  「Neptune MLの Bucket-numerical 特徴」を参照してください。
- **"datetime"** — この特徴エンコーディングは、datetime プロパティ値を、年、月、曜日、および時間の分類的特徴の配列として表します。

  これら 4 つのカテゴリのうち 1 つ以上は、datetime_parts パラメータを使って排除できます。

  「Neptune ML のDatetime 特徴」を参照してください。
- **"text_fasttext"** — この特徴エンコーディングは、fastText モデルを使用して、文または自由形式のテキストで構成されるプロパティ値を数値ベクトルに変換します。英語 (en)、中国語 (zh)、ヒンディー語 (hi)、スペイン語 (es)、フランス語 (fr) の 5 つの言語をサポートしています。この 5 つの言語のいずれかのテキストプロパティ値の場合、text_fasttextが推奨されるエンコーディングです。ただし、同じ文に複数の言語の単語が含まれている場合は処理できません。

  fastText がサポートする言語以外の言語では、text_sbert エンコーディングを使用してください。

  例えば、120 トークンを超えるプロパティ値のテキスト文字列が多数ある場合は、max_length フィールドを使用して、"text_fasttext" がエンコードする各文字列のトークンの数を制限します。

  「Neptune ML でのテキストプロパティ値の fastText エンコーディング」を参照してください。
- **"text_sbert"** — このエンコーディングは、Sententh BERT (SBERT) モデルを使用してテキストプロパティ値を数値ベクトルに変換します。Neptune は 2 つの SBERT メソッドをサ

ポートしています。すなわち、text_sbert128 (text_sbert とだけ指定した場合のデフォルト) と text_sbert512 です。両者の違いは、エンコードされるテキストプロパティ内のトークンの最大数です。text_sbert128 エンコーディングでは最初の 128 トークンのみがエンコードされるのに対し、text_sbert512 は最大 512 トークンをエンコードします。その結果、text_sbert512 を使用する場合は、text_sbert128 よりも処理時間が長くなる可能性があります。どちらの方法も、text_fasttext より遅くなります。

text_sbert* の方法は、多くの言語をサポートしており、複数の言語を含む文をエンコードできます。

「Neptune MLにおけるテキストフィーチャの Sentence BERT (SBERT) エンコーディング」を参照してください。

- **"text_word2vec"** — このエンコーディングは、Word2Vec アルゴリズムを使用して、テキストプロパティ値を数値ベクトルに変換します。英語のみをサポートしています。

  「Neptune ML でのテキストフィーチャの Word2Vec エンコーディング」を参照してください。

- **"text_tfidf"** — このエンコーディングは、term frequency–inverse document frequency (TF-IDF) ベクタライザーを使用して、テキストプロパティ値を数値ベクトルに変換します。

  text_tfidf 機能エンコーディングのパラメータは、ngram_range フィールド、min_df フィールド、および max_features フィールドを使用して定義します。

  「Neptune ML でのテキストフィーチャの TF-IDF エンコーディング」を参照してください。

- **"none"** — none タイプを使用すると、特徴エンコーディングは実行されません。代わりに生プロパティ値が解析され、保存されます。

  カスタムモデルトレーニングの一部として独自のカスタム特徴エンコーディングを実行する予定の場合のみ none を使用します。

norm フィールド

このフィールドは数値特徴に必須です。数値に使用する正規化方法を指定します。

```
"norm": "min-max"
```

次の正規化メソッドがサポートされています。

- "min-max" — 最小値を減算し、最大値と最小値の差で除算して、各値を正規化します。

- "standard" — すべての値の合計で割って、各値を正規化します。
- "none" — エンコーディング中に数値を正規化しないでください。

「Neptune MLのカテゴリ別特徴」を参照してください。

`language` フィールド

言語フィールドは、テキストプロパティ値に使用される言語を指定します。その使用法は、テキストのエンコード方法によって異なります。

- `text_fasttext` エンコーディングの場合、このフィールドは必須で、以下の言語のいずれかを指定する必要があります。
  - `en` (英語)
  - `zh` (中国語)
  - `hi` (ヒンディー語)
  - `es` (スペイン語)
  - `fr` (フランス語)
- `text_sbert` エンコーディングの場合、SBERT エンコーディングは多言語であるため、このフィールドは使用されません。
- `text_word2vec` エンコーディングの場合、`text_word2vec` は英語のみをサポートするため、このフィールドはオプションです。存在する場合は、英語言語モデルの名前を指定する必要があります。

```
"language" : "en_core_web_lg"
```

- `text_tfidf` エンコーディングの場合、このフィールドは使用されません。

`max_length` フィールド

`max_length` フィールドは、`text_fasttext` 機能についてはオプションであり、入力テキストフィーチャ内のエンコードされるトークンの最大数を指定します。`max_length` より長い入力テキストは切り捨てられます。例えば、max_length を 128 に設定すると、テキストシーケンス内の 128 番目より後のトークンは無視されます。

```
"max_length": 128
```

`separator` フィールド

このフィールドはオプションで category、numerical および auto と使用されます。プロパティ値を複数のカテゴリ値または数値に分割するために使用できる文字を指定します。

```
"separator": ";"
```

たとえば、"Actor;Director"または"0.1;0.2"といった、プロパティが複数の区切り値を 1 つの文字列に格納する場合にのみ、この separator フィールドを使用してください。

- カテゴリ別特徴、数値特徴、自動エンコーディング フィールド

`range` フィールド

このフィールドは bucket_numerical 特徴に必須です。バケットに分割する数値の範囲を、[*lower-bound*, *upper-bound*] の形式で指定します。

```
"range" : [20, 100]
```

プロパティ値が下限より小さい場合は、最初のバケットに割り当てられます。上限よりも大きい場合は最後のバケットに割り当てられます。

「Neptune MLの Bucket-numerical 特徴」を参照してください。

- `bucket_cnt` フィールド

このフィールドは bucket_numerical 特徴に必須です。これは、range パラメータにより定義される数値範囲が分割されるバケットの数を指定します。

```
"bucket_cnt": 10
```

「Neptune MLの Bucket-numerical 特徴」を参照してください。

`slide_window_size` フィールド

このフィールドはオプションで bucket_numerical 特徴と使用して複数のバケットに値を割り当てます。

```
"slide_window_size": 5
```

スライドウィンドウの仕組みでは、Neptune ML はウィンドウサイズ **s** を取り、プロパティの各数値 **v** を `v - s/2` から `v + s/2` の範囲内へ変換します。この値は、範囲が重なるすべてのバケットに割り当てられます。

「Neptune MLの Bucket-numerical 特徴」を参照してください。

`imputer` フィールド

このフィールドはオプションで `numerical` および `bucket_numerical` 特徴と使用して欠損値を埋めるためのインプテーション手法を提供します。

```
"imputer": "mean"
```

サポートされているインプテーション手法は次のとおりです。

- `"mean"`
- `"median"`
- `"most-frequent"`

imputer パラメータを含めない場合、欠落した値が見つかったときにデータの前処理が停止し、終了します。

「Neptune MLのカテゴリ別特徴」および「Neptune MLの Bucket-numerical 特徴」を参照してください。

`max_features` フィールド

このフィールドはオプションで `text_tfidf` 特徴と使用して符号化する項の最大数を指定します。

```
"max_features": 100
```

100 に設定すると、TF-IDF ベクタライザーは最も一般的な項を 100 個だけ符号化します。指定しない場合、デフォルト値は `max_features` 5,000 です。

「Neptune ML でのテキストフィーチャの TF-IDF エンコーディング」を参照してください。

`min_df` フィールド

このフィールドはオプションで `text_tfidf` 特徴と使用して符号化する項の最低ドキュメント頻度を指定します。

```
"min_df": 5
```

5 に設定されている場合、符号化されるためには、少なくとも 5 つの異なるプロパティ値に項が含まれている必要があります。

この min_df パラメータを使用しない場合、デフォルト値は 2 です。

「Neptune ML でのテキストフィーチャの TF-IDF エンコーディング」を参照してください。

ngram_range フィールド

このフィールドはオプションで text_tfidf 特徴と使用して、符号化する可能性のある個々の項として考慮すべき単語またはトークンのサイズを指定します。

```
"ngram_range": [2, 4]
```

値 [2, 4] は、2、3、4 語のシーケンスが潜在的な個別項として考慮されることを指定します。

ngram_range と明示的に設定しない場合のデフォルトは [1, 1] で、符号化する項として単一の単語またはトークンのみを考慮することを意味します。

「Neptune ML でのテキストフィーチャの TF-IDF エンコーディング」を参照してください。

datetime_parts フィールド

このフィールドはオプションで datetime 特徴と使用して datetime 値のどの部分をカテゴリ別に符号化するかを指定します。

```
"datetime_parts": ["weekday", "hour"]
```

datetime_parts を含めない場合、デフォルトでは、Neptune ML は datetime 値の年、月、曜日、および時間の部分を符号化します。値 ["weekday", "hour"] は、datetime 値の曜日と時間のみを特徴内でカテゴリ的に符号化する必要があることを示します。

パートの 1 つがトレーニングセットに複数の固有値を持たない場合、そのパートは符号化されません。

「Neptune ML のDatetime 特徴」を参照してください。

モデルトレーニング構成のチューニング用に `additionalParams` 内でパラメータを使用する例

目次

additionalParams を使用したプロパティグラフの例

モデルトレーニングコン構成のデフォルトの分割レートの指定

次の例で、split_rate パラメーターは、モデルトレーニングの既定の分割率を設定します。デフォルトの分割レートが指定されていない場合、トレーニングでは [0.9, 0.1, 0.0] の値が使用されます。ターゲット単位でデフォルト値を上書きするには、各ターゲットに対して split_rate を指定します。

次の例は、default split_rate フィールドは、ターゲット単位で上書きしない限り [0.7,0.1,0.2] の分割レートが使用されることを示します。

```
  "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "split_rate": [0.7,0.1,0.2],
    "targets": [
      (...)
    ],
    "features": [
      (...)
    ]
  }
}
```

モデルトレーニング構成のノード分類タスクの指定

トレーニング目的でラベル付きの例が含まれているノードプロパティがどれかを示すには、ノード分類要素を targets 配列に追加し、"type" : "classification" を使用します。デフォルトの分割レートを上書きする場合は、split_rate フィールドをの追加します。

次の例で、node ターゲットは、各 Movie ノードの genre プロパティはノードクラスラベルとして扱われることを示しています。split_rate 値は、デフォルトの分割レートを上書きします。

```
  "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "node": "Movie",
        "property": "genre",
```

```
        "type": "classification",
        "split_rate": [0.7,0.1,0.2]
      }
    ],
    "features": [
      (...)
    ]
  }
}
```

モデルトレーニング構成のマルチクラスノード分類タスクの指定

トレーニング目的で複数のラベル付き例が含まれているノードプロパティがどれかを示すには、次のようにしてターゲット配列にノード分類要素を追加します。"type" : "classification"、および separator を使用し、ターゲットプロパティ値を複数のカテゴリ値に分割するために使用できる文字を指定します。デフォルトの分割レートを上書きする場合は、split_rate フィールドをの追加します。

次の例で、node ターゲットは、各 Movie ノードの genre プロパティはノードクラスラベルとして扱われることを示しています。separator フィールドは、各ジャンルプロパティに複数のセミコロン区切り値が含まれていることを示します。

```
  "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "node": "Movie",
        "property": "genre",
        "type": "classification",
        "separator": ";"
      }
    ],
    "features": [
      (...)
    ]
  }
}
```

モデルトレーニング構成のノード回帰タスクの指定

トレーニング目的でラベル付きの例が含まれているノードプロパティがどれかを示すには、"type" : "regression" を使用してノード分類要素をターゲット配列に追加します。デフォルトの分割レートを上書きする場合は、split_rate フィールドを追加します。

次の node ターゲットは、各 rating ノードの Movie プロパティは回帰ラベルとして扱われることを示しています。

```
    "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "node": "Movie",
        "property": "rating",
        "type" : "regression",
        "split_rate": [0.7,0.1,0.2]
      }
    ],
    "features": [
      ...
    ]
  }
}
```

モデルトレーニング構成のエッジ分類タスクの指定

トレーニング目的のラベル付きの例が含まれているエッジプロパティがどれかを示すには、エッジ要素を targets 配列に追加し、"type" : "regression" を使用します。デフォルトの分割レートを上書きする場合は、split_rate フィールドを追加します。

次の edge ターゲットは、各 knows エッジの metAtLocation プロパティはエッジクラスラベルとして扱われることを示しています。

```
  "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "edge": ["Person", "knows", "Person"],
        "property": "metAtLocation",
```

```
        "type": "classification"
      }
    ],
    "features": [
      (...)
    ]
  }
}
```

モデルトレーニング構成のマルチクラスエッジ分類タスクの指定

トレーニング目的で複数のラベル付き例が含まれているエッジプロパティを指定するには、エッジ要素を targets 配列に追加し、"type" : "classification"、および separator フィールドを使用して、ターゲットプロパティ値を複数のカテゴリ値に分割するために使用される文字を指定します。デフォルトの分割レートを上書きする場合は、split_rate フィールドをの追加します。

次の edge ターゲットは、各 repliedTo エッジの sentiment プロパティはエッジクラスラベルとして扱われることを示しています。セパレータフィールドは、各センチメントプロパティに複数のカンマ区切り値が含まれていることを示します。

```
  "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "edge": ["Person", "repliedTo", "Message"],
        "property": "sentiment",
        "type": "classification",
        "separator": ","
      }
    ],
    "features": [
      (...)
    ]
  }
}
```

モデルトレーニング構成のエッジ回帰の指定

トレーニング目的のラベル付き回帰の例が含まれているエッジプロパティがどれかを示すには、edge 要素を targets 配列に追加し、"type" : "regression" を使用します。デフォルトの分割レートを上書きする場合は、split_rate フィールドをの追加します。

次の edge ターゲットは、各 rating エッジの reviewed プロパティはエッジ回帰として扱われることを示しています。

```
  "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "edge": ["Person", "reviewed", "Movie"],
        "property": "rating",
        "type" : "regression"
      }
    ],
    "features": [
      (...)
    ]
  }
}
```

モデルトレーニング構成のノード回帰タスクの指定

リンク予測トレーニングの目的でどのエッジが使用されるかを示すには、"type" : "link_prediction" を使用してターゲット配列にエッジ要素を追加します。デフォルトの分割レートを上書きする場合は、split_rate フィールドをの追加します。

次の edge ターゲットは、cites エッジがリンク予測に使用されることを示しています。

```
  "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "edge": ["Article", "cites", "Article"],
        "type" : "link_prediction"
      }
    ],
    "features": [
      (...)
    ]
  }
}
```

数値バケット特徴の指定

ノードプロパティの数値データ特徴を指定するには、"type": "bucket_numerical" を features 配列に追加します。

次の node 特徴は、それぞれの Person ノード の age プロパティが数値バケット特徴として処理されることを示しています。

```
  "additionalParams": {
  "neptune_ml": {
    "targets": [
      ...
    ],
    "features": [
      {
        "node": "Person",
        "property": "age",
        "type": "bucket_numerical",
        "range": [1, 100],
        "bucket_cnt": 5,
        "slide_window_size": 3,
        "imputer": "median"
      }
    ]
  }
}
```

Word2Vec 特徴の指定

"type": "text_word2vec" を features 配列に追加することでノードプロパティの Word2Vec 特徴を指定できます。

次の node 特徴は、それぞれの Movie ノード の description プロパティが Word2Vec 特徴として処理されることを示しています。

```
"additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      ...
    ],
    "features": [
```

```
      {
        "node": "Movie",
        "property": "description",
        "type": "text_word2vec",
        "language": "en_core_web_lg"
      }
    ]
  }
}
```

FastText 特徴の指定

"type": "text_fasttext" を features 配列に追加することでノードプロパティの FastText 特徴を指定できます。language フィールドは必須であり、次のいずれかの言語コードを指定する必要があります。

- en (英語)
- zh (中国語)
- hi (ヒンディー語)
- es (スペイン語)
- fr (フランス語)

text_fasttext エンコーディングでは、1 つの機能で同時に複数の言語を処理できないことに注意してください。

次の node 機能は、それぞれの Movie ノード のフランス語の description プロパティが FastText 機能として処理されることを示しています。

```
"additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      ...
    ],
    "features": [
      {
        "node": "Movie",
        "property": "description",
        "type": "text_fasttext",
```

```
        "language": "fr",
        "max_length": 1024
      }
    ]
  }
}
```

Sentence BERT 特徴の指定

"type": "text_sbert" を features 配列に追加することでノードプロパティの Sentence BERT 特徴を指定できます。このメソッドは多言語言語モデルを使用してテキストフィーチャを自動的にエンコードするため、言語を指定する必要はありません。

次の node 特徴は、それぞれの Movie ノード の description プロパティが Sentence BERT 特徴として処理されることを示しています。

```
"additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      ...
    ],
    "features": [
      {
        "node": "Movie",
        "property": "description",
        "type": "text_sbert128",
      }
    ]
  }
}
```

TF-IDF 特徴の指定

"type": "text_tfidf" を features 配列に追加することでノードプロパティの TF-IDF 特徴を指定できます。

次の node 特徴は、それぞれの Person ノード の bio プロパティが TF-IDF 特徴として処理されることを示しています。

```
  "additionalParams": {
```

```
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      ...
    ],
    "features": [
      {
        "node": "Movie",
        "property": "bio",
        "type": "text_tfidf",
        "ngram_range": [1, 2],
        "min_df": 5,
        "max_features": 1000
      }
    ]
  }
}
```

datetime 特徴の指定

エクスポートプロセスにより、datetime 日付プロパティの特徴が自動的に推論されます。ただし、通常は auto 特徴として処理されるプロパティが明示的に datetime 特徴として処理されるように datetime 特徴に使用される datetime_parts を制限する、または特徴仕様を上書きする場合は、"type": "datetime" 特徴を追加します。

次の node 特徴は、それぞれの Post ノード の createdAt プロパティが datetime 特徴として処理されることを示しています。

```
  "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      ...
    ],
    "features": [
      {
        "node": "Post",
        "property": "createdAt",
        "type": "datetime",
        "datetime_parts": ["month", "weekday", "hour"]
      }
    ]
  }
```

```
}
```

category 特徴の指定

エクスポートプロセスにより、文字列プロパティおよび複数の値を含む数値プロパティの auto 特徴が自動的に推論されます。単一の値を含む数値プロパティの場合、numerical 特徴を推論します。します。日付のプロパティについては、datetime 特徴を推論します。

プロパティがカテゴリ特徴として扱われるように、特徴仕様を上書きする場合は、"type": "category" を特徴配列に追加します。プロパティに複数の値が含まれている場合は、separator フィールドを含めます。例:

```
  "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      ...
    ],
    "features": [
      {
        "node": "Post",
        "property": "tag",
        "type": "category",
        "separator": "|"
      }
    ]
  }
}
```

numerical 特徴の指定

エクスポートプロセスにより、文字列プロパティおよび複数の値を含む数値プロパティの auto 特徴が自動的に推論されます。単一の値を含む数値プロパティの場合、numerical 特徴を推論します。します。日付のプロパティについては、datetime 特徴を推論します。

プロパティを numerical 特徴として処理するために特徴仕様を上書きしたい場合は、"type": "numerical" を特徴配列に追加します。プロパティに複数の値が含まれている場合は、separator フィールドを含めます。例:

```
  "additionalParams": {
```

```
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      ...
    ],
    "features": [
      {
        "node": "Recording",
        "property": "duration",
        "type": "numerical",
        "separator": ","
      }
    ]
  }
}
```

auto 特徴の指定

エクスポートプロセスにより、文字列プロパティおよび複数の値を含む数値プロパティの auto 特徴が自動的に推論されます。単一の値を含む数値プロパティの場合、numerical 特徴を推論します。します。日付のプロパティについては、datetime 特徴を推論します。

プロパティを auto 特徴として処理するために特徴仕様を上書きしたい場合は、"type": "auto" を特徴配列に追加します。プロパティに複数の値が含まれている場合は、separator フィールドを含めます。例:

```
  "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      ...
    ],
    "features": [
      {
        "node": "User",
        "property": "role",
        "type": "auto",
        "separator": ","
      }
    ]
  }
}
```

`additionalParams` を使用した RDF の例

モデルトレーニングコン構成のデフォルトの分割レートの指定

次の例で、split_rate パラメーターは、モデルトレーニングの既定の分割率を設定します。デフォルトの分割レートが指定されていない場合、トレーニングでは [0.9, 0.1, 0.0] の値が使用されます。ターゲット単位でデフォルト値を上書きするには、各ターゲットに対して split_rate を指定します。

次の例は、default split_rate フィールドは、ターゲット単位で上書きしない限り [0.7,0.1,0.2] の分割レートが使用されることを示します。

```
  "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "split_rate": [0.7,0.1,0.2],
    "targets": [
      (...)
    ]
  }
}
```

モデルトレーニング構成のノード分類タスクの指定

トレーニング目的のラベル付きの例が含まれているノードプロパティがどれかを示すには、ノード分類要素を targets 配列に追加し、"type" : "classification" を使用します。ノードフィールドを追加して、ターゲットノードのノードタイプを指定します。predicate フィールドを追加して、ターゲットノードのターゲットノード特徴として使用されるリテラルデータを定義します。デフォルトの分割レートを上書きする場合は、split_rate フィールドをの追加します。

次の例で、node ターゲットは、各 Movie ノードの genre プロパティはノードクラスラベルとして扱われることを示しています。split_rate 値は、デフォルトの分割レートを上書きします。

```
  "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "node": "http://aws.amazon.com/neptune/csv2rdf/class/Movie",
        "predicate": "http://aws.amazon.com/neptune/csv2rdf/datatypeProperty/genre",
        "type": "classification",
        "split_rate": [0.7,0.1,0.2]
```

```
      }
    ]
  }
}
```

モデルトレーニング構成のノード回帰タスクの指定

トレーニング目的でラベル付きの例が含まれているノードプロパティがどれかを示すには、"type" : "regression" を使用してノード分類要素をターゲット配列に追加します。node フィールドを追加して、ターゲットノードのノードタイプを指定します。predicate フィールドを追加して、ターゲットノードのターゲットノード特徴として使用されるリテラルデータを定義します。デフォルトの分割レートを上書きする場合は、split_rate フィールドをの追加します。

次の node ターゲットは、各 rating ノードの Movie プロパティは回帰ラベルとして扱われることを示しています。

```
    "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "node": "http://aws.amazon.com/neptune/csv2rdf/class/Movie",
        "predicate": "http://aws.amazon.com/neptune/csv2rdf/datatypeProperty/rating",
        "type": "regression",
        "split_rate": [0.7,0.1,0.2]
      }
    ]
  }
}
```

特定のエッジのリンク予測タスクの指定

リンク予測トレーニングの目的に使用するエッジを指定するには、"type" : "link_prediction" を使用してターゲット配列にエッジ要素を追加します。subject、predicate および object フィールドを追加して、エッジタイプを指定します。デフォルトの分割レートを上書きする場合は、split_rate フィールドをの追加します。

次の edge ターゲットは、Directors を Movies に接続する directed エッジがリンク予測に使用されることを示しています。

```
  "additionalParams": {
```

```
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "subject": "http://aws.amazon.com/neptune/csv2rdf/class/Director",
        "predicate": "http://aws.amazon.com/neptune/csv2rdf/datatypeProperty/directed",
        "object": "http://aws.amazon.com/neptune/csv2rdf/class/Movie",
        "type" : "link_prediction"
      }
    ]
  }
}
```

すべてのエッジのリンク予測タスクの指定

リンク予測トレーニングの目的に使用するエッジを指定するには、"type" : "link_prediction" を使用してターゲット配列に edge 要素を追加します。subject、predicate または object フィールドは追加しません。デフォルトの分割レートを上書きする場合は、split_rate フィールドをの追加します。

```
  "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "type" : "link_prediction"
      }
    ]
  }
}
```

Neptune からエクスポートされたグラフデータをトレーニング用に処理する

データ処理ステップでは、エクスポートプロセスによって Neptune グラフデータが取得され、トレーニング中にディープグラフライブラリ (DGL) が使用する情報が作成されます。これには、さまざまなデータマッピングと変換の実行が含まれます。

- ノードとエッジを解析して、DGL で必要とされるグラフおよび ID マッピングファイルを構築する。
- ノードとエッジプロパティを DGL で必要なノードおよびエッジ特徴に変換する。
- データをトレーニング、検証、およびテストセットに分割します。

Neptune ML のデータ処理ステップを管理する

モデルトレーニングに使用するデータを Neptune からエクスポートした後、`curl` (または `awscurl`) コマンドを使用してデータ処理ジョブを開始できます。

```
curl \
  -X POST https://(your Neptune endpoint)/ml/dataprocessing \
  -H 'Content-Type: application/json' \
  -d '{
        "inputDataS3Location" : "s3://(Amazon S3 bucket name)/(path to your input
 folder)",
        "id" : "(a job ID for the new job)",
        "processedDataS3Location" : "s3://(S3 bucket name)/(path to your output
 folder)",
        "configFileName" : "training-job-configuration.json"
      }'
```

このコマンドの使用方法の詳細については、データ処理コマンド を参照してください。また、実行中のジョブのステータスの取得方法、実行中のジョブの停止方法、実行中のすべてのジョブの一覧表示方法について説明した情報もご覧ください。

Neptune ML の更新されたグラフデータの処理

また、`previousDataProcessingJobId` をAPI に追加して、新しいデータ処理ジョブが前のジョブと同じ処理方法を使用するようにします。これは、新しいデータで古いモデルを再学習するか、

新しいデータのモデルアーティファクトを再計算することによって、Neptune で更新されたグラフデータの予測を取得する場合に必要です。

これを行うには、次のような curl (または awscurl) コマンドを使います。

```
curl \
  -X POST https://(your Neptune endpoint)/ml/dataprocessing \
  -H 'Content-Type: application/json' \
  -d '{ "inputDataS3Location" : "s3://(Amazon S3 bucket name)/(path to your input
 folder)",
        "id" : "(a job ID for the new job)",
        "processedDataS3Location" : "s3://(Amazon S3 bucket name)/(path to your output
 folder)",
        "previousDataProcessingJobId", "(the job ID of the previous data-processing
 job)"}'
```

トレーニング済みモデルに対応する前のデータ処理ジョブのジョブ ID に previousDataProcessingJobId パラメータの値を設定します。

> **Note**
>
> 更新されたグラフでのノードの削除は、現在サポートされていません。更新されたグラフでノードが削除されている場合は、previousDataProcessingJobId を使用するのではなく、まったく新しいデータ処理ジョブを開始する必要があります。

Neptune ML での特徴エンコーディング

プロパティ値と RDF リテラルには、さまざまな形式とデータ型があります。機械学習で優れたパフォーマンスを実現するには、これらの値を特徴として知られる数値エンコーディングに変換することが不可欠です。

Neptune ML は、ここで説明されているように、データのエクスポートおよびデータ処理手順の一部として、特徴抽出とエンコーディングを実行します。

> **Note**
>
> カスタムモデルの実装で独自の特徴エンコーディングを実装する場合は、データの前処理段階で none を特徴エンコーディングタイプとして選択することで自動特徴エンコーディングを無効にすることができます。その後、そのノードまたはエッジプロパティで特徴エンコーディングは発生せず、代わりに生プロパティ値が解析され、ディクショナリに保存されます。データの前処理では、エクスポートされたデータセットから DGL グラフが作成されますが、構築された DGL グラフにはトレーニング用の前処理特徴がありません。
> カスタムモデルトレーニングの一部として独自のカスタム特徴エンコーディングを実行する予定の場合のみこの選択肢を使用します。詳細については、「Neptune ML のカスタムモデル」を参照してください。

Neptune MLのカテゴリ別特徴

可能な値の固定リストから 1 つ以上の異なる値を取得できるプロパティは、カテゴリ別特徴です。Neptune ML では、カテゴリ別特徴は one-hot エンコーディングを使用して符号化されます。次の例は、さまざまな食品のプロパティ名がそのカテゴリに従って one-hot 符号化される方法を示しています。

```
 Food        Veg.   Meat   Fruit    Encoding
---------    ----   ----   -----    --------
 Apple         0      0      1         001
 Chicken       0      1      0         010
 Broccoli      1      0      0         100
```

> **Note**
>
> カテゴリ別特徴の最大数は 100 です。プロパティの値のカテゴリが 100 を超える場合、最も一般的なカテゴリの 99 のみが個別のカテゴリに配置され、残りは OTHER という名前の特別なカテゴリに配置されます。

Neptune MLのカテゴリ別特徴

値が実数であるプロパティは、Neptune ML で数値特徴として符号化できます。数値特徴は、浮動小数点数を使用して符号化されます。

次のように、数値特徴を符号化するときに使用するデータ正規化方法を指定できます。`"norm": "normalization technique"`。次の正規化手法がサポートされています。

- "none" — エンコーディング中に数値を正規化しないでください。
- "min-max" — 最小値を減算し、最大値と最小値の差で除算して、各値を正規化します。
- "standard" — すべての値の合計で割って、各値を正規化します。

Neptune MLの Bucket-numerical 特徴

生の数値を使用して数値プロパティを表すのではなく、数値をカテゴリに集約できます。たとえば、人の年齢を、子供 (0 ~20 歳)、若年大人 (20 ~ 40 歳)、中年 (40 ~ 60 歳)、および高齢者 (60歳以上) などのカテゴリに分類できます。これらの数値バケットを使用すると、数値プロパティを一種のカテゴリ特徴に変換することになります。

Neptune ML では、数値プロパティをバケット数値特徴として符号化できます。次の 2 つを指定する必要があります。

- 数値範囲、 `"range": [a, b]` 形式では、a および b は整数です。
- バケットカウント、 `"bucket_cnt": c` 形式では、c はバケットの数であり、整数でもあります。

次に、Neptune ML は各バケットのサイズを `( b - a ) / c` と計算し、各数値を格納するバケットの数として符号化します。a 未満の値は最初のバケットに属していると見なされ、b より大きい任意の値は最後のバケットに属していると見なされます。

オプションで、次のようにスライドウィンドウサイズを `"slide_window_size": s` のように (s は数字) 指定することで、数値を複数のバケットに分類することもできます。Neptune ML はプロパティの各数値 v を `v - s/2` から `v + s/2` 範囲内へ変換し、範囲がカバーするすべてのバケットに値 v を代入します。

最後に、オプションで、数値特徴と bucket-numerical 特徴の欠損値を埋める方法を提供することもできます。これは、 `"imputer": "imputation technique "` を使用して行います。ここで、帰属技法は`"mean"`、`"median"` または `"most-frequent"` のうちの 1 つです。中央値を指定しなかった場合、値が欠落していると、処理が停止する可能性があります。

Neptune ML でのテキストフィーチャエンコーディング

自由形式のテキストの場合、Neptune ML はいくつかの異なるモデルを使用して、プロパティ値文字列内のトークンのシーケンスを固定サイズの実数値ベクトルに変換できます。

- `text_fasttext` — fastText エンコーディングを使用します。これは、fastText がサポートする 5 つの言語のうちの 1 つだけを使用するフィーチャに推奨されるエンコーディングです。
- `text_sbert` — Sentence BERT (SBERT) エンコーディングモデルを使用します。これは、`text_fasttext` がサポートしていないテキストについて推奨されるエンコーディングです。
- `text_word2vec` — Google が最初に公開した Word2Vec アルゴリズムを使用して、テキストをエンコードします。Word2Vec は英語のみをサポートしています。
- `text_tfidf` — term frequency–inverse document frequency (TF-IDF) ベクタライザを使用して、テキストのエンコードを行います。TF-IDF エンコーディングは、他のエンコーディングにはない統計機能をサポートします。

Neptune ML でのテキストプロパティ値の fastText エンコーディング

Neptune ML は fastText モデルを使用して、テキストプロパティ値を固定サイズの実数値ベクトルに変換できます。これは、FastText がサポートする 5 つの言語のいずれかのテキストプロパティ値の推奨エンコーディング方法です。

- `en` (英語)
- `zh` (中国語)
- `hi` (ヒンディー語)
- `es` (スペイン語)

- `fr` (フランス語)

fastText は複数の言語の単語を含む文を処理できないことに注意してください。

`text_fasttext` メソッドは、エンコードされるテキストプロパティ値のトークンの最大数を指定する `max_length` フィールドをオプションで取得できます。この数を超えると、文字列は切り捨てられます。これによって、テキストプロパティ値に長い文字列が含まれる場合のパフォーマンスが向上します。`max_length` が指定されなかった場合、fastText は文字列の長さに関係なく、すべてのトークンをエンコードするためです。

次の例では、フランス語の映画タイトルが fastText を使用してエンコードされるように指定しています。

```
{
    "file_name" : "nodes/movie.csv",
    "separator" : ",",
    "node" : ["~id", "movie"],
    "features" : [
      {
        "feature": ["title", "title", "text_fasttext"],
        "language": "fr",
        "max_length": 1024
      }
    ]
  }
```

Neptune MLにおけるテキストフィーチャの Sentence BERT (SBERT) エンコーディング

Neptune ML は、Sententh BERT (SBERT) モデルを使用して、文字列プロパティ値内のトークンのシーケンスを固定サイズの実数値ベクトルに変換できます。Neptune は 2 つの SBERT メソッドをサポートしています。すなわち、`text_sbert128` (`text_sbert` を指定した場合のデフォルト) と `text_sbert512` です。この 2 つの違いは、エンコードされるテキストプロパティ値文字列の最大長です。`text_sbert128` エンコーディングでは、128 トークンをエンコードするとテキスト文字列が切り捨てられますが、`text_sbert512` では、512 トークンをエンコードした後にテキスト文字列が切り捨てられます。その結果、`text_sbert512` を使用する場合は、`text_sbert128` よりも処理時間が長くなる可能性があります。どちらの方法も、`text_fasttext` より遅くなります。

SBERT エンコーディングは多言語対応なので、エンコードするプロパティ値のテキストについて言語を指定する必要はありません。SBERT は多くの言語をサポートしており、複数の言語を含む文を

エンコードできます。fastText がサポートしていない言語のテキストを含むプロパティ値をエンコードする場合は、SBERT が推奨されるエンコード方法です。

次の例では、映画のタイトルを最大 128 トークンまで SBERT としてエンコードするように指定しています。

```
{
    "file_name" : "nodes/movie.csv",
    "separator" : ",",
    "node" : ["~id", "movie"],
    "features" : [
      { "feature": ["title", "title", "text_sbert128"] }
    ]
  }
```

Neptune ML でのテキストフィーチャの Word2Vec エンコーディング

Neptune ML は、文字列プロパティ値を Word2Vec 機能としてエンコードできます (Word2Vec アルゴリズムは元々 Google によって公開されました)。`text_word2vec` メソッドは、spaCy でトレーニング済みモデルのいずれかを使用して、文字列内のトークンを密なベクトルとしてエンコードします。これは en_core_web_lg モデルを使用する英語のみをサポートしています。

次の例では、映画のタイトルを Word2Vec を使用してエンコードするように指定しています。

```
{
    "file_name" : "nodes/movie.csv",
    "separator" : ",",
    "node" : ["~id", "movie"],
    "features" : [
      {
        "feature": ["title", "title", "text_word2vec"],
        "language": "en_core_web_lg"
      }
    ]
  }
```

Neptune がサポートしているのは英語 `en_core_web_lg` モデルだけなので、言語フィールドはオプションであることに注意してください。

Neptune ML でのテキストフィーチャの TF-IDF エンコーディング

Neptune ML は、テキストプロパティ値を `text_tfidf` フィーチャとしてエンコードできます。このエンコーディングは、term frequency–inverse document frequency (TF-IDF) ベクタライザを使用して、テキスト内の単語のシーケンスを数値ベクトルに変換し、その後に次元削減操作を実行します。

TF-IDF (項の頻度 — 逆ドキュメントの頻度) は、文書セット内の単語の重要性を測定するための数値です。これは、特定のプロパティ値に単語が表示される回数をそのプロパティ値の合計数で割って計算されます。

例えば、ある映画のタイトルに「キス」という単語が2回登場し (例えば「キス· キス· バン· バン」)、4本の映画のタイトルすべてに「キス」が登場する場合、「キス· バンバン」タイトルの「キス」の TF-IDF 値は、 `2 / 4` となります。

最初に作成されるベクトルは d ディメンションで、ここで d は、そのタイプのすべてのプロパティ値の一意の項の数です。次元削減オペレーションでは、ランダムなスパース投影を使用して、その数値を最大 100 に減らします。グラフのボキャブラリーは、その中の `text_tfidf` 特徴をすべて合わせて生成されます。

TF-IDF ベクタライザーは、いくつかの方法で制御できます。

- **`max_features`** — `max_features` パラメータを使用して、最も一般的な特徴に `text_tfidf` 特徴の項数を制限できます。たとえば、`max_features` を 100 に設定した場合、最も一般的に使用される項の上位 100 のみが含まれます。`max_features` のデフォルト値を明示的に設定しなかった場合、5,000 となります。
- **`min_df`** — `min_df` パラメータを使用して、少なくとも指定されたドキュメント頻度を持つ特徴へ `text_tfidf` 特徴の項数を制限できます。たとえば、`min_df` を 5 に設定した場合、少なくとも 5 つの異なるプロパティ値に含まれる項のみが使用されます。`min_df` のデフォルト値を明示的に設定しなかった場合、2 となります。
- **`ngram_range`** — `ngram_range` パラメータは、項として扱われる単語の組み合わせを決定します。たとえば、`ngram_range` を `[2, 4]` に設定した場合、「キス· キス· バン· バン」のタイトルで以下の 6 項が見つかることになります。
  - 2 単語項：「キス· キス」、「キス· バン」、「バン· バン」。
  - 3 単語項：「キス· キス· バン」と「キス· バン· バン」。
  - 4 単語項：「キス· キス· バン· バン」。

  `ngram_range` のデフォルトの設定は `[1, 1]` です。

Neptune ML のDatetime 特徴

Neptune ML は `datetime` プロパティ値の一部を、one-hot 配列として符号化してカテゴリ別特徴に変換できます `datetime_parts` パラメータを使用して、符号化する次のパートの `["year", "month", "weekday", "hour"]` のうち 1 つまたは複数を指定します。`datetime_parts` を設定しなければで、デフォルトで 4 つの部分すべてが符号化されます。

たとえば、datetime 値の範囲が 2010 年から 2012 年にわたる場合、datetime 入力 `2011-04-22 01:16:34` の 4 つの部分は次のとおりです。

- year – `[0, 1, 0]`。

  スパンには 3 年 (2010、2011、2012) しかないため、one-hot 配列には各年に 1 つずつ、3 つの入力値があります。
- month – `[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]`。

  ここでは、one-hot 配列には、その年の各月の入力値があります。
- weekday – `[0, 0, 0, 0, 1, 0, 0]`。

  ISO 8601 規格では、月曜日が週の最初の曜日であると定められており、2011 年 4 月 22 日は金曜日であったため、対応するone-hot weekday 配列は 5 番目の位置で hot です。
- hour – `[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]`。

  午前 1 時の時間は 24-member one-hot 配列で設定されます。

月、分、秒の曜日はカテゴリ的に符号化されません。

総 `datetime` 対象範囲に 1 年以内の日付のみが含まれている場合、`year` 配列はどれも符号化されません。

`imputer` パラメータと、数値特徴に使用できる戦略の 1 つを使用して、欠落した`datetime`値を保管するためのインピュテーション戦略を指定できます。

Neptune ML での自動特徴エンコーディング

グラフのプロパティに使用する特徴エンコーディングメソッドを手動で指定する代わりに、特徴エンコーディングメソッドとして `auto` を設定できます。次に、Neptune ML は、基礎となるデータ型に基づいて、各プロパティの最適な特徴エンコーディングを推論しようとします。

Neptune ML が適切な特徴エンコーディングを選択する際に使用する発見的手法をいくつか紹介します。

- プロパティが数値のみを持ち、数値データ型にキャストできる場合、Neptune ML は通常、それを数値として符号化します。ただし、プロパティの一意の値の数が値の合計数の 10% 未満で、それらの一意の値のカーディナリティが 100 未満の場合、Neptune ML はカテゴリ別エンコーディングを使用します。
- プロパティ値を `datetime` 型にキャストできる場合、Neptune ML はそれらを `datetime` 特徴に符号化します。
- プロパティ値をブール値 (1/0 または True/False) に強制できる場合、Neptune ML はカテゴリエンコーディングを使用します。
- プロパティがその値の 10% を超える一意の文字列であり、値あたりの平均トークン数が 3 以上の場合、Neptune ML はプロパティタイプをテキストとして推論し、使用されている言語を自動的に検出します。検出された言語が fastText によってサポートされている言語、つまり、英語、中国語、ヒンディー語、スペイン語、フランス語のいずれかである場合、Neptune ML は `text_fasttext` を使用してテキストをエンコードします。それ以外の場合、Neptune ML は `text_sbert` を使用します。
- プロパティがテキスト特徴として分類されない文字列の場合、Neptune ML はカテゴリ別特徴であると仮定し、カテゴリエンコーディングを使用します。
- 各ノードがカテゴリ特徴であると推論されるプロパティに固有の値がある場合、Neptune ML は学習に有益ではない ID であるため、トレーニンググラフからプロパティを削除します。
- プロパティにセミコロン (「;」) などの有効な Neptune 区切り文字が含まれていることがわかっている場合、Neptune ML はプロパティを `MultiNumerical` または `MultiCategorical` とのみ扱えます。
  - Neptune ML は、まず値を数値特徴として符号化しようとします。これが成功すると、Neptune ML は数値エンコーディングを使用して数値ベクトル特徴を作成します。
  - それ以外の場合、Neptune ML は値をマルチカテゴリとして符号化します。
- Neptune ML がプロパティの値のデータ型を推論できない場合、Neptune MLはトレーニンググラフからプロパティを削除します。

トレーニングデータ設定ファイルの編集

Neptune エクスポートプロセスは、Neptune DB クラスターから S3 バケットに Neptune ML データをエクスポートします。ノードとエッジを nodes/ と edges/ の別々のフォルダにエクスポーします。また、デフォルトでは training-data-configuration.json という名前の JSON トレーニングデータ設定ファイルも作成します。このファイルには、グラフのスキーマ、特徴の型、特徴変換と正規化操作、分類または回帰タスクのターゲット特徴に関する情報が含まれています。

設定ファイルを直接変更したい場合があります。このようなケースの 1 つは、解析する機械学習タスクの仕様を変更するたびにエクスポートを再実行しなくても、特徴の処理方法やグラフの作成方法を変更する場合です。

トレーニングデータ設定ファイルを編集するには

1. ファイルをローカルマシンにダウンロードします。

   エクスポートプロセスで渡された additionalParams/neptune_ml パラメータで 1 つ以上の名前付きジョブを指定していない限り、ファイルはデフォルトの名前 training-data-configuration.json になります。次のような AWS CLI コマンドを使用siteファイルをダウンロードします。

   ```
   aws s3 cp \
     s3://(your Amazon S3 bucket)/(path to your export folder)/training-data-
   configuration.json \
     ./
   ```

2. テキストエディタを使用してファイルを編集します。
3. 変更したファイルをアップロードします。変更したファイルを、ダウンロードした Amazon S3 の同じ場所にアップロードします。次のような AWS CLI コマンドを使用します。

   ```
   aws s3 cp \
     training-data-configuration.json \
     s3://(your Amazon S3 bucket)/(path to your export folder)/training-data-
   configuration.json
   ```

JSON トレーニングデータ設定ファイルの例

ノード分類タスクのグラフを説明するトレーニングデータ設定ファイルの例を次に示します。

```
{
  "version" : "v2.0",
  "query_engine" : "gremlin",
  "graph" : [
    {
      "edges" : [
        {
          "file_name" : "edges/(movie)-included_in-(genre).csv",
          "separator" : ",",
          "source" : ["~from", "movie"],
          "relation" : ["", "included_in"],
          "dest" : [ "~to", "genre" ]
        },
        {
          "file_name" : "edges/(user)-rated-(movie).csv",
          "separator" : ",",
          "source" : ["~from", "movie"],
          "relation" : ["rating", "prefixname"], # [prefixname#value]
          "dest" : ["~to", "genre"],
          "features" : [
            {
              "feature" : ["rating", "rating", "numerical"],
              "norm" : "min-max"
            }
          ]
        }
      ],
      "nodes" : [
        {
          "file_name" : "nodes/genre.csv",
          "separator" : ",",
          "node" : ["~id", "genre"],
          "features" : [
            {
              "feature": ["name", "genre", "category"],
              "separator": ";"
            }
          ]
        },
        {
          "file_name" : "nodes/movie.csv",
          "separator" : ",",
          "node" : ["~id", "movie"],
```

```
          "features" : [
            {
              "feature": ["title", "title", "word2vec"],
              "language": ["en_core_web_lg"]
            }
          ]
        },
        {
          "file_name" : "nodes/user.csv",
          "separator" : ",",
          "node" : ["~id", "user"],
          "features" : [
            {
              "feature": ["age", "age", "numerical"],
              "norm" : "min-max",
              "imputation": "median",
            },
            {
              "feature": ["occupation", "occupation", "category"],
            }
          ],
          "labels" : [
            {
              "label": ["gender", "classification"],
              "split_rate" : [0.8, 0.2, 0.0]
            }
          ]
        }
      ]
    },
    "warnings" : [ ]
  ]
}
```

JSON トレーニングデータ設定ファイルの構造

トレーニング設定ファイルは、エクスポートプロセスによって nodes/ および edges/ フォルダに保存された CSV ファイルを参照します。

nodes/ の下にある各ファイルは、同じプロパティグラフノードラベルを持つノードに関する情報を格納します。ノードファイルの各列には、ノード ID またはノードプロパティが格納されます。ファイルの最初の行には、~id または、各列のプロパティ名を指定するヘッダーが含まれます。

edges/ の下にある各ファイルは、同じプロパティグラフエッジラベルを持つノードに関する情報を格納します。ノードファイルの各列には、始点ノード ID、終点ノード ID、またはエッジプロパティが格納されます。ファイルの最初の行には、~from、~to または各列のプロパティ名を指定するヘッダーが含まれます。

トレーニングデータ設定ファイルには、次の 3 つの最上位要素があります。

```
{
  "version" : "v2.0",
  "query_engine" : "gremlin",
  "graph" : [ ... ]
}
```

- version — (文字列) 使用されているコンフィギュレーションファイルのバージョン。
- query_engine — (文字列) グラフデータのエクスポートに使用するクエリ言語。現在、有効なのは「gremlin」のみです。
- graph — (JSON 配列) 使用される各ノードとエッジのモデルパラメーターを含む 1 つ以上の設定オブジェクトを一覧表示します。

  グラフ配列の構成オブジェクトは、次のセクションで説明する構造です。

graph 配列に一覧表示されている構成オブジェクトの内容

graph 配列内の構成オブジェクトには 3 つの最上位ノードを含めることができます。

```
    {
      "edges"    : [ ... ],
      "nodes"    : [ ... ],
      "warnings" : [ ... ],
    }
```

- edges — (JSON オブジェクトの配列) 各 JSON オブジェクトは、モデルの処理およびトレーニング中にグラフ内のエッジがどのように処理されるかを定義するパラメータのセットを指定します。これは Gremlin エンジンでのみ使用されます。
- nodes — (JSON オブジェクトの配列) 各 JSON オブジェクトは、モデルの処理およびトレーニング中にグラフ内のノードがどのように処理されるかを定義するパラメータのセットを指定します。これは Gremlin エンジンでのみ使用されます。

- warnings — (JSON オブジェクトの配列) 各オブジェクトには、データのエクスポートプロセス中に生成された警告が含まれています。

edges 配列に一覧表示されているエッジ構成オブジェクトの内容

edges 配列に一覧表示されているエッジ構成オブジェクトは、次の最上位のフィールドを含めることができます。

```
    {
      "file_name" : "(path to a CSV file)",
      "separator" : "(separator character)",
      "source"    : ["(column label for starting node ID)", "(starting node type)"],
      "relation"  : ["(column label for the relationship name)", "(the prefix name
for the relationship name)"],
      "dest"      : ["(column label for ending node ID)", "(ending node type)"],
      "features"  : [(array of feature objects)],
      "labels"    : [(array of label objects)]
    }
```

- **file_name** — 同じプロパティグラフラベルを持つエッジに関する情報を格納する CSV ファイルへのパスを指定する文字列。

  そのファイルの最初の行には、列ラベルのヘッダー行が含まれています。

  最初の 2 つの列ラベルは ~from および ~to です。最初の列 (~from 列) は、エッジの開始ノードの ID を格納し、2 番目 (~to列) は、エッジの終了ノードの ID を格納します。

  ヘッダー行の残りの列ラベルは、残りの列ごとに、その列に値がエクスポートされたエッジプロパティの名前を指定します。
- **separator** — その CSV ファイル内の列を区切る区切り文字を含む文字列。
- **source** — エッジの開始ノードを指定する 2 つの文字列を含む JSON 配列。最初の文字列には、開始ノード ID が格納されているカラムのヘッダー名が含まれます。2 番目の文字列は、ノードタイプを指定します。
- **relation** — エッジのリレーションタイプを指定する 2 つの文字列を含む JSON 配列。最初の文字列には、リレーション名 (relname) が格納されているカラムのヘッダー名が含まれます。2 番目の文字列には、リレーション名 (prefixname) のプレフィックスが含まれています。

  完全なリレーションタイプは、*prefixname*-*relname* のように 2 つの文字列を結合し、それらの間にハイフン文字を組み合わせて構成します。

最初の文字列が空の場合、すべてのエッジが同じリレーションタイプになります。つまり、`prefixname` 文字列です。

- **`dest`** — エッジの終了ノードを指定する 2 つの文字列を含む JSON 配列。最初の文字列には、ノード ID が格納されているカラムのヘッダー名が含まれます。2 番目の文字列は、ノードタイプを指定します。
- **`features`** — プロパティ値特徴オブジェクトの JSON 配列。各プロパティ値特徴オブジェクトには、次のフィールドが含まれています。
  - feature — 3 つの文字列の JSON 配列。最初の文字列には、プロパティ値を含むカラムのヘッダー名が含まれます。2 番目の文字列には、特徴名が含まれます。3 番目の文字列には、特徴型が含まれます。
  - norm — (オプション) プロパティ値に適用する正規化方法を指定します。

- **`labels`** — オブジェクトの JSON 配列。各オブジェクトは、エッジのターゲット特徴を定義し、トレーニングおよび検証段階で取るエッジの比率を指定します。各オブジェクトには、以下のフィールドが含まれています。
  - label — 2 つの文字列の JSON 配列。最初の文字列には、ターゲット特徴のプロパティ値を格納する列のヘッダー名が含まれます。2 番目の文字列は、以下のいずれかのターゲットタスク型を指定します。
    - `"classification"` — エッジ分類タスク。列に表示されるプロパティ値は、`label` 配列の最初の文字列で識別され、カテゴリ別値として扱われます。エッジ分類タスクの場合、`label` 配列の最初の文字列を空にすることはできません。
    - `"regression"` — エッジ回帰タスク。列に表示されるプロパティ値は、`label` 配列の最初の文字列で識別され、数値として扱われます。エッジ回帰タスクの場合、`label` 配列の最初の文字列を空にすることはできません。
    - `"link_prediction"` — リンク予測タスク。プロパティ値は必要ありません。リンク予測タスクの場合、`label` 配列の最初の文字列は無視されます。
  - **`split_rate`** — ゼロから 1 までの、足して 1 になる 3 つの数値を含む JSON 配列。トレーニング、検証、およびテストの各ステージで使用されるノードの比率の推定値を表します。このフィールドまたは `custom_split_filenames` のいずれかを定義できますが、両方は定義できません。split_rate を参照してください。
  - **`custom_split_filenames`** — トレーニング、検証、およびテストの母集団を定義するファイルのファイル名を指定する JSON オブジェクト。このフィールドまたは `split_rate` のいずれかを定義できますが、両方は定義できません。詳細については、「カスタムのトレイン、検証、テストの比率」を参照してください。

nodes 配列に一覧表示されているノード構成オブジェクトの内容

nodes 配列に一覧表示されているノード設定オブジェクトには、次のフィールドを含めることができます。

```
    {
      "file_name" : "(path to a CSV file)",
      "separator" : "(separator character)",
      "node"      : ["(column label for the node ID)", "(node type)"],
      "features"  : [(feature array)],
      "labels"    : [(label array)],
    }
```

- **file_name** — 同じプロパティグラフラベルを持つノードに関する情報を格納する CSV ファイルへのパスを指定する文字列。

  そのファイルの最初の行には、列ラベルのヘッダー行が含まれています。

  最初の列ラベルは ~id であり、最初の列 (~id 列) には、ノード ID が保存されます。

  ヘッダー行の残りの列ラベルは、残りの列ごとに、その列に値がエクスポートされたノードプロパティの名前を指定します。
- **separator** — その CSV ファイル内の列を区切る区切り文字を含む文字列。
- **node** — 2 つの文字列を含む JSON 配列。最初の文字列には、ノード ID が格納されているカラムのヘッダー名が含まれます。2 番目の文字列は、グラフ内のノード型を指定します。これは、ノードのプロパティグラフラベルに対応します。
- **features** — ノード特徴オブジェクトの JSON 配列。「ノードまたはエッジの features 配列に一覧表示されている特徴オブジェクトの内容」を参照してください。
- **labels** — ノードラベルオブジェクトの JSON 配列。「ノード labels 配列に一覧表示されたノードラベルオブジェクトの内容」を参照してください。

ノードまたはエッジの **features** 配列に一覧表示されている特徴オブジェクトの内容

ノード features 配列に一覧表示されているノード特徴オブジェクトには、次の最上位フィールドを含めることができます。

- **feature** — 3 つの文字列の JSON 配列。最初の文字列には、特徴のプロパティ値を格納する列のヘッダー名が含まれます。2 番目の文字列には、特徴名が含まれます。

3 番目の文字列には、特徴型が含まれます。有効な特徴型の一覧を 特徴の type フィールドで使用できる値 に示します。

- **norm** - このフィールドは数値特徴に必須です。数値に使用する正規化方法を指定します。有効な値は、"none"、"min-max"、「標準」です。詳細については、「norm フィールド」を参照してください。
- **language** - 言語フィールドは、テキストプロパティ値に使用されている言語を指定します。その使用法は、テキストのエンコード方法によって異なります。
  - text_fasttext エンコーディングの場合、このフィールドは必須であり、以下の言語のいずれかを指定する必要があります。
    - en (英語)
    - zh (中国語)
    - hi (ヒンディー語)
    - es (スペイン語)
    - fr (フランス語)

    ただし、text_fasttext は、一度に複数の言語を処理することはできません。
  - text_sbert エンコーディングの場合、SBERT エンコーディングは多言語対応であるため、このフィールドは使用されません。
  - text_word2vec エンコーディングの場合、text_word2vec は英語のみをサポートするため、このフィールドはオプションです。存在する場合は、英語言語モデルの名前を指定する必要があります。

    ```
    "language" : "en_core_web_lg"
    ```
  - tfidf エンコーディングの場合、このフィールドは使用されません。
- **max_length** - このフィールドは、text_fasttext 機能についてはオプションであり、エンコードされる入力テキストフィーチャ内のトークンの最大数を指定します。max_length に達した後の入力テキストは無視されます。例えば、max_length を 128 に設定すると、テキストシーケンス内の 128 番目より後のトークンは無視されます。
- **separator** - このフィールドはオプションで 、category、numerical および auto と使用されます。プロパティ値を複数のカテゴリ値または数値に分割するために使用できる文字を指定します。

  「separator フィールド」を参照してください。

- **range** - このフィールドは bucket_numerical 特徴に必須です。バケットに分割する数値の範囲を指定します。

  「range フィールド」を参照してください。

- **bucket_cnt** - このフィールドは bucket_numerical 特徴に必須です。これは、range パラメータにより定義される数値範囲が分割されるバケットの数を指定します。

  「Neptune MLの Bucket-numerical 特徴」を参照してください。

- **slide_window_size** - このフィールドはオプションで bucket_numerical 特徴と使用して複数のバケットに値を割り当てます。

  「slide_window_size フィールド」を参照してください。

- **imputer** - このフィールドはオプションで numerical、bucket_numerical および datetime 特徴と使用して欠損値を埋めるためのインプテーション手法を提供します。サポートされているインプテーションテクニックは、"mean"、"median" および "most_frequent" です。

  「imputer フィールド」を参照してください。

- **max_features** - このフィールドはオプションで text_tfidf 特徴と使用して符号化する項の最大数を指定します。

  「max_features フィールド」を参照してください。

- **min_df** - このフィールドはオプションで text_tfidf 特徴と使用して符号化する項の最低ドキュメント頻度を指定します。

  「min_df フィールド」を参照してください。

- **ngram_range** — このフィールドは、text_tfidf 特徴にオプションで使用され、符号化する可能性のある個々の項と見なされる単語またはトークンの数の範囲を指定します。

  「ngram_range フィールド」を参照してください。

- **datetime_parts** - このフィールドはオプションで datetime 特徴と使用して datetime 値のどの部分をカテゴリ別に符号化するかを指定します。

  「datetime_parts フィールド」を参照してください。

ノード **labels** 配列に一覧表示されたノードラベルオブジェクトの内容

ノード labels 配列に一覧表示されたラベルオブジェクトは、ノードターゲット特徴を定義し、トレーニング、検証、およびテストの各段階で使用するノードの比率を指定します。各オブジェクトには次のフィールドの要素を含めることができます。

```
    {
      "label"      : ["(column label for the target feature property value)", "(task
type)"],
      "split_rate" : [(training proportion), (validation proportion), (test
proportion)],
      "custom_split_filenames" : {"train": "(training file name)", "valid":
"(validation file name)", "test": "(test file name)"},
      "separator"  : "(separator character for node-classification category values)",
    }
```

- **label** — 2 つの文字列を含む JSON 配列。最初の文字列には、特徴のプロパティ値を格納する列のヘッダー名が含まれます。2 番目の文字列は、ターゲットタスクの種類を指定します。次のようになります。
  - "classification" — ノード分類タスク。指定した列のプロパティ値は、カテゴリ特徴の作成に使用されます。
  - "regression" — ノード回帰タスク。指定した列のプロパティ値は、数値特徴の作成に使用されます。
- **split_rate** — ゼロから 1 の間の 3 つの数値を含む JSON 配列。最大 1 を足し、トレーニング、検証、およびテストの各ステージで使用されるノードの比率の推定値を表します。「split_rate」を参照してください。
- **custom_split_filenames** — トレーニング、検証、およびテストの母集団を定義するファイルのファイル名を指定する JSON オブジェクト。このフィールドまたは split_rate のいずれかを定義できますが、両方は定義できません。詳細については、「カスタムのトレイン、検証、テストの比率」を参照してください。
- **separator** — 分類タスクのカテゴリ別特徴値を区切る区切り文字を含む文字列。

> **Note**
>
> エッジとノードの両方にラベルオブジェクトを指定しない場合、タスクは自動的にリンク予測と見なされ、エッジはトレーニングのために 90%、検証のために 10% にランダムに分割されます。

カスタムのトレイン、検証、テストの比率

デフォルトでは、`split_rate` パラメータは Neptune ML によって、このパラメータで定義された比率を使用してグラフをトレーニング、検証、およびテストの母集団にランダムに分割するために使用されます。これらの異なる母集団でどのエンティティを使用するかをより正確に制御するには、それらを明示的に定義するファイルを作成し、トレーニングデータ設定ファイルを編集して、これらのインデックスファイルを母集団にマッピングすることができます。このマッピングは、トレーニング設定ファイルの `custom_split_filesnames` キーの JSON オブジェクトによって指定されます。このオプションを使用する場合、`train` および `validation` キーにはファイル名を指定する必要がありますが、`test` キーでは省略可能です。

これらのファイルの形式は Gremlin データ形式に一致する必要があります。具体的には、ノードレベルのタスクでは、各ファイルにノード ID を一覧表示する `~id` ヘッダー付きの列が含まれている必要があります。エッジレベルのタスクでは、ファイルは `~from` および `~to` を指定して、エッジのソースノードとデスティネーションノードをそれぞれ示す必要があります。これらのファイルは、データ処理に使用されるエクスポートデータと同じ Amazon S3 の場所に配置する必要があります (「`outputS3Path`」を参照)。

プロパティの分類や回帰タスクでは、これらのファイルで機械学習タスクのラベルをオプションで定義できます。その場合、ファイルには、トレーニングデータ設定ファイルで定義されているのと同じヘッダー名のプロパティ列が必要です。エクスポートされたノードおよびエッジファイルとカスタム分割ファイルの両方でプロパティラベルが定義されている場合、カスタム分割ファイルが優先されます。

Neptune ML を使用したモデルのトレーニング

モデルトレーニングのために Neptune からエクスポートするデータを処理した後、次のように curl (または awscurl) コマンドを使用してモデルトレーニングジョブを開始できます。

```
curl \
  -X POST https://(your Neptune endpoint)/ml/modeltraining
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique model-training job ID)",
        "dataProcessingJobId" : "(the data-processing job-id of a completed job)",
        "trainModelS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-graph-
autotrainer"
      }'
```

このコマンドの使用方法の詳細については、モデルトレーニングコマンド を参照してください。また、実行中のジョブのステータスの取得方法、実行中のジョブの停止方法、実行中のすべてのジョブの一覧表示方法について説明する情報も併せてご確認ください。

完了した Neptune ML モデルトレーニングジョブの情報を使用して、新しいトレーニングジョブでハイパーパラメータ検索を高速化するために previousModelTrainingJobId を指定することもできます。これは、新しいグラフデータのモデルの再トレーニングであり、同じグラフデータに対するインクリメンタルトレーニングでもあります。次のようなコマンドを使用します。

```
curl \
  -X POST https://(your Neptune endpoint)/ml/modeltraining
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique model-training job ID)",
        "dataProcessingJobId" : "(the data-processing job-id of a completed job)",
        "trainModelS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-graph-
autotrainer"
        "previousModelTrainingJobId" : "(the model-training job-id of a completed job)"
      }'
```

Neptune ML トレーニングインフラストラクチャで、次のように customModelTrainingParameters オブジェクトを指定して独自のモデル実装をトレーニングできます。

```
curl \
```

```
  -X POST https://(your Neptune endpoint)/ml/modeltraining
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique model-training job ID)",
        "dataProcessingJobId" : "(the data-processing job-id of a completed job)",
        "trainModelS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-graph-
autotrainer"
        "modelName": "custom",
        "customModelTrainingParameters" : {
          "sourceS3DirectoryPath": "s3://(your Amazon S3 bucket)/(path to your Python
 module)",
          "trainingEntryPointScript": "(your training script entry-point name in the
 Python module)",
          "transformEntryPointScript": "(your transform script entry-point name in the
 Python module)"
        }
      }'
```

実行中のジョブのステータスの取得方法、実行中のジョブの停止方法、実行中のすべてのジョブの一覧表示方法などの詳細については、モデルトレーニングコマンド を参照してください。カスタムモデルの実装と使用方法の詳細については、Neptune ML のカスタムモデル を参照してください。

トピック

- Amazon Neptune ML でのモデルとモデルトレーニング
- Neptune ML でのモデルのハイパーパラメータ設定のカスタマイズ
- モデルトレーニングのベストプラクティス

Amazon Neptune ML でのモデルとモデルトレーニング

Neptune ML は、グラフニューラルネットワーク (GNN) を使用して、さまざまな機械学習タスクのモデルを作成します。グラフニューラルネットワークは、グラフ機械学習タスクに関する最先端の結果を得るため、グラフ構造化データから情報パターンを抽出するのに優れています。

Neptune ML のグラフニューラルネットワーク (GNN)

グラフニューラルネットワーク (GNN) は、近傍のノードの構造と特徴を考慮してノード表現を計算するニューラルネットワークのファミリーに属します。GNN は、グラフデータには適していない他の従来の機械学習とニューラルネットワーク手法を補完します。

GNN は、ノード分類と回帰 (ノードの特性の予測)、エッジ分類と回帰 (エッジの特性の予測) やリンク予測 (グラフ内の 2 つのノードを接続すべきかどうかを予測) などの機械学習タスクを実行するために使用されます。

一般に、GNN を機械学習タスクに使用するには、次の 2 つの段階があります。

- 符号化段階。GNN がグラフ内の各ノードの d 次元ベクトルを計算します。これらのベクトルは、表現または埋め込みです。
- 符号化された表現に基づいて予測を行う復号化ステージ。

ノードの分類と回帰では、ノード表現が分類および回帰タスクに直接使用されます。エッジ分類と回帰では、エッジ上のインシデントノードのノード表現が分類または回帰の入力として使用されます。リンク予測の場合、エッジ尤度スコアは、ノード表現とエッジタイプ表現のペアを使用して計算されます。

ディープグラフライブラリ (DGL) は、これらのタスクの GNN の効率的な定義とトレーニングを容易にします。

メッセージパッシングの定式化の下で異なる GNN モデルが統一されます。このビューでは、グラフ内のノードの表現は、ノードの近傍の表現 (メッセージ) とノードの初期表現を使用して計算されます。NeptuneML では、ノードの初期表現は、ノードのプロパティから抽出された特徴から導出されるか、または学習可能であり、ノードのアイデンティティに依存します。

Neptune ML には、ノード特徴と学習可能なノード表現を連結して、元のノード表現として機能するオプションも用意されています。

ノードプロパティを持つグラフを含む Neptune ML のさまざまなタスクについては、リレーショナルグラフ畳み込みネットワーク (R-GCN)) で符号化段階を行います。R-GCN は、複数のノードタイ

プとエッジタイプを持つグラフに適した GNN アーキテクチャです (これらは異種グラフと呼ばれます)。

R-GCN ネットワークは固定数のレイヤーで構成され、次々に積み重ねられます。R-GCN の各レイヤーは、学習可能なモデルパラメータを使用して、ノードの直近 1 ホップ近傍からの情報を集約します。後続のレイヤーは前のレイヤーの出力表現を入力として使用するため、ノードの最終的な埋め込みに影響するグラフ近傍の半径は、R-GCN ネットワークのレイヤーの数 (`num-layer`) によって異なります。

たとえば、これは、2 層ネットワークが 2 ホップ離れたノードからの情報を使用することを意味します。

GNN の詳細については、グラフニューラルネットワークに関する包括的な調査を参照してください。Deep Graph Library (DGL) の詳細については、DGL ウェブページを参照してください。DGL での GNN の使用に関する実践的なチュートリアルについては、ディープグラフライブラリを使用したグラフニューラルネットワークの学習を参照してください。

トレーニンググラフニューラルネットワーク

機械学習では、タスクに対して適切な予測を行う方法を学ぶためにモデルを取得するプロセスをモデルトレーニングと呼びます。これは、通常、最適化する具体的な目的と、この最適化を実行するために使用するアルゴリズムを指定することによって実行されます。

このプロセスは、下流のタスクに対する適切な表現を学ぶための GNN のトレーニングにも使用されます。モデルトレーニング中に最小化される、そのタスクの目的関数を作成します。たとえば、ノード分類の場合、CrossentropyLoss を目標とし、誤分類にペナルティを科し、ノード回帰では SquareError を最小限に抑えます。

目的は通常、特定のデータポイントのモデル予測を取得し、そのデータポイントのグラウンドトゥルース値と比較する損失関数です。これは、モデルの予測がどれくらい離れているかを示す損失値を返します。トレーニングプロセスの目標は、損失を最小限に抑え、モデルの予測がグラウンドトゥルースに近いことを確認することです。

トレーニングプロセスのディープラーニングで使用される最適化アルゴリズムは、通常、勾配降下の一種です。Neptune MLでは、Adam を使用します。これは、低次モーメントの適応的推定に基づいて、確率的目的関数を一次勾配に基づいて最適化するアルゴリズムです。

モデルトレーニングプロセスでは、学習したモデルパラメータが目的関数の最小値に近いことを確認しようとしますが、モデルの全体的なパフォーマンスは、モデルのハイパーパラメータに依存し、これは、トレーニングアルゴリズムで学習されないモデル設定です。たとえば、学習したノード表

現、num-hidden、の次元性は、モデルのパフォーマンスに影響するハイパーパラメータです。したがって、機械学習では、ハイパーパラメータ最適化 (HPO) を実行して適切なハイパーパラメータを選択することが一般的です。

Neptune ML は SageMaker ハイパーパラメータチューニングジョブを使用して、さまざまなハイパーパラメータ設定によりモデルトレーニングの複数のインスタンスを起動し、ハイパーパラメータ設定範囲に最適なモデルを見つけようとします。「Neptune ML でのモデルのハイパーパラメータ設定のカスタマイズ」を参照してください。

Neptune ML にモデルを埋め込むナレッジグラフ

ナレッジグラフ (KG) は、異なるエンティティ (ノード) とその関係 (エッジ) に関する情報をエンコードするグラフです。Neptune ML では、グラフにノードプロパティが含まれず、他のノードとの関係のみが含まれている場合に、リンク予測を実行するためにデフォルトでナレッジグラフ埋め込みモデルが適用されます。ただし、学習可能な埋め込みを持つ R-GCN モデルは、モデル型を "rgcn" のように指定することで、これらのグラフにも使用できます。一方、知識グラフ埋め込みモデルはより単純であり、大規模な知識グラフの表現を学ぶのに有効になるように設計されています。

ナレッジグラフ埋め込みモデルは、リンク予測タスクで使用され、**h** が始点ノードであり、**r** がリレーションタイプ、**t** が終点ノードであるトリプル (**h**, **r**, **t**) を完了するノードまたはリレーションを予測します。

Neptune MLで実装されたナレッジグラフ埋め込みモデルは distmult、transE および rotatE です。ナレッジグラフの埋め込みモデルの詳細については、「DGL-KE」を参照してください。

Neptune ML のカスタムモデルのトレーニング

Neptune ML では、特定のシナリオに対して、独自のカスタムモデルを定義して実装できます。カスタムモデルの実装方法と、Neptune ML インフラストラクチャを使用してトレーニングする方法については、Neptune ML のカスタムモデル を参照してください。

Neptune ML でのモデルのハイパーパラメータ設定のカスタマイズ

Neptune ML モデルトレーニングジョブを開始すると、Neptune ML は先行するデータ処理ジョブから推測された情報を自動的に使用します。この情報を使用して、ハイパーパラメータ設定範囲を生成し、これは SageMaker のハイパーパラメータ調整ジョブタスクのために複数のモデルをトレーニングするために使用されます。そうすれば、トレーニングの対象となるモデルのハイパーパラメータ値の長いリストを指定する必要はありません。代わりに、モデルのハイパーパラメータの範囲とデフォルトは、タスクタイプ、グラフタイプ、およびチューニングジョブ設定に基づいて選択されます。

ただし、データ処理ジョブが生成する JSON 設定ファイルを変更して、デフォルトのハイパーパラメータ設定を上書きし、カスタムハイパーパラメータを指定することもできます。

Neptune MLを使う modelTraining API では、maxHPONumberOfTrainingJobs、maxHPOParallelTrainingJobs および trainingInstanceType のようなハイレベルハイパーパラメータチューニングジョブ設定をいくつか制御できます。モデルのハイパーパラメータをよりきめ細かく制御するには、データ処理ジョブが生成する model-HPO-configuration.json ファイルをカスタマイズします。ファイルは、処理ジョブの出力用に指定した Amazon S3 の場所に保存されます。

ファイルをダウンロードし、デフォルトのハイパーパラメータ設定を上書きするように編集し、同じ Amazon S3 の場所にアップロードし直すことができます。ファイルの名前を変更しないでください。編集の際は、次の手順に従うよう注意してください。

Amazon S3 からファイルをダウンロードするには:

```
aws s3 cp \
  s3://(bucket name)/(path to output folder)/model-HPO-configuration.json \
  ./
```

編集が終わったら、ファイルを元の場所にアップロードし直します。

```
aws s3 cp \
  model-HPO-configuration.json \
  s3://(bucket name)/(path to output folder)/model-HPO-configuration.json
```

model-HPO-configuration.json ファイルの構造

model-HPO-configuration.json ファイルは、学習するモデル、機械学習 task_type およびモデルトレーニングのさまざまな実行に対して変更または固定する必要のあるハイパーパラメータを指定します。

ハイパーパラメータは、ハイパーパラメータチューニングジョブが呼び出されたときにハイパーパラメータに与えられる優先順位を示すさまざまな階層に属するものとして分類されます。

- Tier-1 ハイパーパラメータの優先順位は最優先されます。maxHPONumberOfTrainingJobs を10未満の値に設定すると、Tier-1 ハイパーパラメータのみが調整され、残りはデフォルト値になります。
- Tier-2 ハイパーパラメータの優先順位は低いため、チューニングジョブの合計トレーニングジョブが 10 以上あっても 50 未満の場合は、Tier-1 と Tier-2 の両方のハイパーパラメータが調整されます。
- Tier 3 のハイパーパラメータは、合計 50 を超えるトレーニングジョブがある場合にのみ、Tier-1 および Tier-2 とともに調整されます。
- ハイパーパラメータを特定の層に配置し、その範囲を編集し、そのデフォルト値が適切に設定されていることを確認して、ハイパーパラメータの優先順位を変更します。

model-HPO-configuration.json ファイルの例

次に、model-HPO-configuration.json ファイルの例を示します。

```
{
  "models": [
    {
      "model": "rgcn",
      "task_type": "node_class",
      "eval_metric": {
        "metric": "acc"
      },
      "eval_frequency": {
          "type":  "evaluate_every_epoch",
          "value":  1
      },
      "1-tier-param": [
        {
            "param": "num-hidden",
            "range": [16, 128],
            "type": "int",
            "inc_strategy": "power2"
        },
        {
          "param": "num-epochs",
          "range": [3,30],
```

```
          "inc_strategy": "linear",
          "inc_val": 1,
          "type": "int",
          "node_strategy": "perM"
        },
        {
          "param": "lr",
          "range": [0.001,0.01],
          "type": "float",
          "inc_strategy": "log"
        }
      ],
      "2-tier-param": [
        {
          "param": "dropout",
          "range": [0.0,0.5],
          "inc_strategy": "linear",
          "type": "float",
          "default": 0.3
        },
        {
          "param": "layer-norm",
          "type": "bool",
          "default": true
        }
      ],
      "3-tier-param": [
        {
          "param": "batch-size",
          "range": [128, 4096],
          "inc_strategy": "power2",
          "type": "int",
          "default": 1024
        },
        {
          "param": "fanout",
          "type": "int",
          "options": [[10, 30],[15, 30], [15, 30]],
          "default": [10, 15, 15]
        },
        {
          "param": "num-layer",
          "range": [1, 3],
          "inc_strategy": "linear",
```

```
          "inc_val": 1,
          "type": "int",
          "default": 2
        },
        {
          "param": "num-bases",
          "range": [0, 8],
          "inc_strategy": "linear",
          "inc_val": 2,
          "type": "int",
          "default": 0
        }
      ],
      "fixed-param": [
        {
          "param": "concat-node-embed",
          "type": "bool",
          "default": true
        },
        {
          "param": "use-self-loop",
          "type": "bool",
          "default": true
        },
        {
          "param": "low-mem",
          "type": "bool",
          "default": true
        },
        {
          "param": "l2norm",
          "type": "float",
          "default": 0
        }
      ]
    }
  ]
}
```

model-HPO-configuration.json 要素のリスト。

このファイルには、モデル設定オブジェクトが 1 つ含まれる `models` という名前のトップレベル配列が 1 つある JSON オブジェクトが含まれています。ファイルをカスタマイズするときは、`models`

配列にはモデル設定オブジェクトが 1 つしかないことを確認してください。ファイルに複数のモデル設定オブジェクトが含まれている場合、チューニングジョブは警告とともに失敗します。

モデル設定オブジェクトには、次の最上位要素が含まれます。

- **`model`** — (文字列) トレーニングするモデルタイプ (変更しない)。有効な 値は次のとおりです。
  - `"rgcn"` — これは、ノード分類および回帰タスク、および異種リンク予測タスクのデフォルトです。
  - `"transe"` — これは、KGE リンク予測タスクのデフォルトです。
  - `"distmult"` — これは KGE リンク予測タスクの代替モデルタイプです。
  - `"rotate"` — これは KGE リンク予測タスクの代替モデルタイプです。

  原則として、`model` 値は直接変更しないでください。モデルタイプによって適用可能なハイパーパラメータが実質的に異なることが多く、トレーニングジョブの開始後に解析エラーが発生する可能性があるためです。

  モデルタイプを変更するには、`model-HPO-configuration.json` ファイルで変えるのではなく、モデルトレーニング API で `modelName` パラメータを使用します。

  モデルタイプを変更し、細粒度のハイパーパラメータを変更するには、使用するモデルのデフォルトのモデル設定テンプレートをコピーして、`model-HPO-configuration.json` ファイルにペーストします。推論タスクタイプが複数のモデルをサポートしている場合は、`model-HPO-configuration.json` ファイルと同じ Amazon S3 の場所に `hpo-configuration-templates` という名前のフォルダがあります。このフォルダには、タスクに適用可能な他のモデルのデフォルトのハイパーパラメータ設定がすべて含まれています。

  たとえば、KGE リンク予測タスクのためにモデルとハイパーパラメータ設定をデフォルトの `transe` モデルから `distmult` モデルに変更する場合、`hpo-configuration-templates/distmult.json` ファイルの内容を `model-HPO-configuration.json` ファイルに貼り付けるだけで、それから必要に応じてハイパーパラメータを編集します。

  > **Note**
  >
  > `modelTraining` API で `modelName` のパラメータを設定し、また `model` とハイパーパラメータの仕様を `model-HPO-configuration.json` ファイルで変更すると、これらは異なり、`model-HPO-configuration.json` ファイル内の `model` 値が優先され、`modelName` 値は無視されます。

- **task_type** — (文字列) データ処理ジョブによって推測される、またはデータ処理ジョブに直接渡される機械学習タスクタイプ (変更しない)。有効な 値は次のとおりです。
  - "node_class"
  - "node_regression"
  - "link_prediction"

  データ処理ジョブは、エクスポートされたデータセットと生成されたトレーニングジョブ設定ファイルでデータセットのプロパティを調べて、タスクタイプを推論します。

  この値は変更しないでください。別のタスクをトレーニングしたいなら、新しいデータ処理ジョブを実行する必要があります。task_type 値が予測と異なる場合は、データ処理ジョブへの入力が正しいか確認する必要があります。これには、modelTraining API に対するパラメータ、およびデータエクスポートプロセスによって生成されたトレーニングジョブ設定ファイル内があります。
- **eval_metric** — (文字列) 評価指標は、モデルのパフォーマンスを評価し、HPO 実行全体で最もパフォーマンスの高いモデルを選択するために使用する必要があります。有効な 値は次のとおりです。
  - "acc" — 標準の分類精度。これは、データ処理中に不均衡なラベルが検出されない限り、単一ラベル分類タスクのデフォルトです。この場合、デフォルトは "F1" です。
  - "acc_topk" — **k** 予測の中で正しいラベルが一番上にある回数。追加のキーとして topk で受け渡すことで **k** 値も設定できます。
  - "F1" — F1 スコア。
  - "mse" – 平均二乗誤差メトリクス。回帰タスクの場合。
  - "mrr" – 平均逆数ランクメトリクス。
  - "precision" — 予測された陽性に対する真陽性の比率として計算されたモデルの精度。= true-positives / (true-positives + false-positives)。
  - "recall" — 実際の陽性に対する真陽性の比率として計算されたモデルのリコール。= true-positives / (true-positives + false-negatives)。
  - "roc_auc" — ROC カーブ下のエリア。これは、マルチラベル分類のデフォルトです。

  たとえば、メトリクスを F1 に変更するには、eval_metric 値を次のように変更します。

```
"  eval_metric": {
    "metric": "F1",
  },
```

または、メトリクスを topk 精度スコアに変更するには、eval_metric 値を次のように変更します。

```
"eval_metric": {
  "metric": "acc_topk",
  "topk": 2
},
```

- **eval_frequency** — (オブジェクト) トレーニング中に検証セット上のモデルのパフォーマンスをチェックする頻度を指定します。検証のパフォーマンスに基づいて、早期停止を開始し、最適なモデルを保存できます。

  eval_frequency オブジェクトには いわゆる "type" および "value" の、2 つの要素が含まれています。例:

```
"eval_frequency": {
  "type":  "evaluate_every_pct",
  "value":  0.1
},
```

  有効な type 値は次のとおりです。

  - **evaluate_every_pct** — 各評価で完了するトレーニングの割合を指定します。

    evaluate_every_pct の場合、"value" フィールドには、そのパーセンテージを表すゼロから 1 までの浮動小数点数が含まれます。

  - **evaluate_every_batch** — 各評価で完了するトレーニングバッチの数を指定します。

    evaluate_every_batch の場合、"value" フィールドには、そのバッチ数を表す整数が含まれます。

  - **evaluate_every_epoch** — 評価ごとのエポック数を指定します。新しいエポックは午前 0 時に開始されます。

    evaluate_every_epoch の場合、"value" フィールドには、そのエポック数を表す整数が含まれます。

  eval_frequency のデフォルトの設定は次のとおりです。

```
"eval_frequency": {
```

```
    "type":  "evaluate_every_epoch",
    "value":  1
  },
```

- **1-tier-param** — (必須) Tier-1 ハイパーパラメータの配列。

  ハイパーパラメータを調整しない場合は、これを空の配列に設定できます。これは SageMaker ハイパーパラメータのチューニングジョブによって起動されるトレーニングジョブの総数には影響しません。これは、すべてのトレーニングジョブが 1 より多いが 10 未満の場合、同じハイパーパラメータのセットで実行されることを意味します。

  一方、すべての調整可能なハイパーパラメータを同じ有意で扱う場合は、すべてのハイパーパラメータをこの配列に入れることができます。

- **2-tier-param** — (必須) Tier-2 ハイパーパラメータの配列。

  これらのパラメータは、`maxHPONumberOfTrainingJobs` が 10 より大きい値を持つ場合にのみ調整されます。そうしない場合は、デフォルト値が固定されます。

  最大で 10 のトレーニングジョブ用のトレーニング予算がある場合、または他の理由で Tier-2 ハイパーパラメータを必要とせず、すべての調整可能なハイパーパラメータを調整する場合は、これを空の配列に設定できます。

- **3-tier-param** — (必須) Tier-3 ハイパーパラメータの配列。

  これらのパラメータは、`maxHPONumberOfTrainingJobs` が 50 より大きい値を持つ場合にのみ調整されます。そうしない場合は、デフォルト値が固定されます。

  Tier-3 ハイパーパラメータを調整しない場合は、これを空の配列に設定できます。

- **fixed-param** — (必須) 既定値のみを取り、さまざまなトレーニングジョブによって変化しない固定ハイパーパラメータの配列。

  すべてのハイパーパラメータを変更する場合は、これを空の配列に設定し、すべての階層を変えるか、すべてのハイパーパラメータを Tier-1 にするのに十分な大きさ `maxHPONumberOfTrainingJobs` の値に設定します。

`1-tier-param`、`2-tier-param`、`3-tier-param` および `fixed-param` の各ハイパーパラメータを表す JSON オブジェクトには、次の要素が含まれます。

- **param** — (文字列) ハイパーパラメータの名前 (変わらない)。

フレームワークの使用の詳細については、Neptune ML の有効なハイパーパラメータ名のリストを参照してください。

- **type** — (文字列) ハイパーパラメータタイプ (変わらない)。

  有効なタイプは、bool、int および float です。

- **default** — (文字列) ハイパーパラメータのデフォルト値。

  新しいデフォルト値を設定できます。

調整可能なハイパーパラメータには、次の要素を含めることができます。

- **range** — (配列) 連続調整可能なハイパーパラメータの範囲。

  これは、2つの値、すなわち範囲の最小値と最大値を持つ配列でなければなりません ([min, max])。

- **options** — (配列) カテゴリカル調整可能なハイパーパラメータのオプション。

  この配列には、考慮すべきすべてのオプションが含まれている必要があります。

  ```
  "options" : [value1, value2, ... valuen]
  ```

- **inc_strategy** — (文字列) 連続調整可能なハイパーパラメータ範囲に対する増分変更のタイプ (変わらない)。

  有効な値は、log、linear、power2 です。これは、範囲キーが設定されている場合にだけ適用されます。

  これを変更すると、チューニングにハイパーパラメータの全範囲を使用しないことがあります。

- **inc_val** — (浮動小数点) 連続調整可能ハイパーパラメータで連続する増分が異なる量 (変わらない)。

  これは、範囲キーが設定されている場合にだけ適用されます。

  これを変更すると、チューニングにハイパーパラメータの全範囲を使用しないことがあります。

- **node_strategy** — (文字列) このハイパーパラメータの有効範囲は、グラフ内のノード数に基づいて変化することを示します (変わらない)。

有効な値は、"perM" (100 万当たり)、"per10M" (1000 万当たり)、"per100M" (1 億当たり) です。

この値を変更するのではなく、代わりに range を変更します。

- **edge_strategy** — (文字列) このハイパーパラメータの有効範囲は、グラフ内のエッジ数に基づいて変化することを示します (変わらない)。

  有効な値は、"perM" (100 万当たり)、"per10M" (1000 万当たり)、"per100M" (1 億当たり) です。

  この値を変更するのではなく、代わりに range を変更します。

Neptune ML のすべてのハイパーパラメータのリスト

次のリストには、Neptune ML の任意のモデルタイプおよびタスクに対して設定できるすべてのハイパーパラメータが含まれています。これらはすべて全部のモデルタイプに適用できるわけではないので、ハイパーパラメータは使用しているモデルのテンプレートに表示される model-HPO-configuration.json ファイルでのみ設定します。

- **batch-size** — 1 つのフォワードパスで使用するターゲットノードのバッチのサイズ。タイプ: int。

  これをはるかに大きな値に設定すると、GPU インスタンスのトレーニングでメモリの問題が発生する可能性があります。
- **concat-node-embed** — モデルの表現度を高めるために、処理された特徴を学習可能な最初のノード埋め込みと連結して、ノードの初期表現を取得するかどうかを示します。タイプ: bool。
- **dropout** — ドロップアウトレイヤーに適用されるドロップアウトの確率。タイプ: float。
- **edge-num-hidden** — エッジフィーチャモジュールの非表示のレイヤーサイズまたはユニット数。use-edge-features が True に設定されている場合にのみ使用されます。タイプ: 浮動小数点。
- **enable-early-stop** — 早期停止機能を使用するかどうかを切り替えます。タイプ: bool。デフォルト: true。

  このブール値パラメータを使用して、早期停止機能をオフにします。
- **fanout** — ネイバーサンプリング中にターゲットノードについてサンプリングするネイバーの数。タイプ: int。

この値は、num-layers と緊密に結合されています。また、常に同じハイパーパラメータ層内に配置する必要があります。これは、潜在的な GNN レイヤーごとにファンアウトを指定できるためです。

このハイパーパラメータによってモデルのパフォーマンスが大きく変わる可能性があるため、固定するか、Tier-2 または Tier-3 ハイパーパラメータとして設定する必要があります。これを大きな値に設定すると、GPU インスタンスのトレーニングでメモリの問題が発生する可能性があります。

- **gamma** — スコア関数のマージン値。タイプ: float。

  これは、KGE リンク予測モデルのみに当てはまります。

- **l2norm** — オプティマイザで使用される荷重減衰値。荷重に L2 正規化ペナルティを課します。タイプ: bool。
- **layer-norm** — rgcn モデルにレイヤー正規化を使用するかどうかを示します。タイプ: bool。
- **low-mem** — 速度を犠牲にしてリレーションメッセージパッシング関数のメモリ不足実装を使用するかどうかを示します。タイプ: bool。
- **lr** — 学習レート。タイプ: float。

  これは Tier-1 ハイパーパラメータとして設定する必要があります。

- **neg-share** — リンク予測では、正のサンプリングされたエッジが負のエッジサンプルを共有できるかどうかを示します。タイプ: bool。
- **num-bases** — rgcn モデルにおける基底分解のベース数。グラフ内のエッジタイプの数よりも小さい num-bases の値を使用すると、グラフは rgcn モデルの正則化手段として機能します。タイプ: int。
- **num-epochs** - 実行するトレーニングエポックの数。タイプ: int。

  エポックは、グラフを通る完全なトレーニングパスです。

- **num-hidden** — 非表示のレイヤーのサイズまたは単位数。タイプ: int。

  これにより、特徴がないノードの初期埋め込みサイズも設定されます。

  これを batch-size を減らすことなくはるかに大きな値に設定すると、GPU インスタンスのトレーニングでメモリの問題が発生する可能性があります。

- **num-layer** — モデル内の GNN レイヤーの数。タイプ: int。

この値は、ファンアウトパラメータと緊密に結合されています。また、常に同じハイパーパラメータ層内にファンアウトを設定した後にこれが来る必要があります。

これによってモデルのパフォーマンスが大きく変わる可能性があるため、固定するか、Tier-2 または Tier-3 ハイパーパラメータとして設定する必要があります。

- **num-negs** — リンク予測では、正のサンプルあたりの負のサンプルの数。タイプ: int。
- **per-feat-name-embed** — 特徴を組み合わせる前に個別に変換することにより、各特徴を埋め込むかどうかを示します。タイプ: bool。

  true に設定すると、ノードごとの各フィーチャが独立して固定次元サイズに変換され、その後、ノードのすべての変換されたフィーチャが連結され、さらに num_hidden 次元に変換されます。

  false に設定すると、フィーチャ固有の変換を行わずにフィーチャが連結されます。
- **regularization-coef** — リンク予測では、正則化損失の係数。タイプ: float。
- **rel-part** — KGE リンク予測に対してリレーションパーティションを使用するかどうかを示します。タイプ: bool。
- **sparse-lr** — 学習可能なノード埋め込みの学習率。タイプ: float。

  学習可能な初期ノード埋め込みは、特徴がないノードや、concat-node-embed が設定されたときに使用されます。スパース学習可能ノード埋め込みレイヤーのパラメータは、個別の学習率を持つことができる別のオプティマイザを使用してトレーニングされます。
- **use-class-weight** — 不均衡な分類タスクにクラス荷重を適用するかどうかを示します。true に設定すると、ラベル数を使用して、各クラスラベルの荷重が設定されます。タイプ: bool。
- **use-edge-features** — メッセージの受け渡し時にエッジ機能を使用するかどうかを示します。true に設定すると、特徴を持つエッジタイプのカスタムエッジ機能モジュールが RGCN レイヤーに追加されます。タイプ: bool。
- **use-self-loop** — rgcn モデルのトレーニングにセルフループを含めるかどうかを示します。タイプ: bool。
- **window-for-early-stop** - 早期停止を決定するために平均する最新の検証スコアの数を制御します。デフォルトは 3. type=int です。「Neptune ML でのモデルトレーニングプロセスの早期停止」も参照してください。タイプ: int。デフォルト: 3。

  「」を参照してください。

Neptune ML でのハイパーパラメータのカスタマイズ

model-HPO-configuration.json ファイルを編集しているときでは、最も一般的な変更の種類を以下に示します。

- range ハイパーパラメータの最小値および/または最大値を編集します。
- ハイパーパラメータを固定値に設定するには、fixed-param セクションを開き、デフォルト値を設定したい固定値に設定します。
- ハイパーパラメータを特定の層に配置し、その範囲を編集し、そのデフォルト値が適切に設定されていることを確認して、ハイパーパラメータの優先順位を変更します。

モデルトレーニングのベストプラクティス

Neptune ML モデルのパフォーマンスを向上させるためにできることはあります。

適切なノードプロパティを選択する

グラフ内のすべてのプロパティが機械学習タスクに有意義である、または関連性があるとは限りません。無関係なプロパティは、データのエクスポート時に除外する必要があります。

ベストプラクティスを以下に示します。

- ドメインのエキスパートを使用して、特徴の重要性とそれらを予測に使用することの実現可能性を評価します。
- データのノイズや重要でない相関を減らすために、冗長または無関係であると判断した特徴を削除します。
- モデルの構築中に反復処理を行います。特徴、特徴の組み合わせ、調整目標を調整できます。

Amazon Machine Learning Developer Guide の特徴処理には、Neptune ML に関連する特徴処理に関する追加のガイドラインが記載されています。

外れ値のデータポイントの処理

外れ値は、残りのデータとは大きく異なるデータ点です。データの外れ値は、トレーニングプロセスを台無しにしたり誤解を招いたりして、トレーニング時間が長くなったり、モデルの精度が低下したりする可能性があります。これらが本当に重要でない限り、データをエクスポートする前に外れ値を排除する必要があります。

重複するノードとエッジを削除する

Neptune に格納されているグラフには、重複したノードまたはエッジがある場合があります。これらの冗長要素は、ML モデルトレーニングにノイズを発生させます。データをエクスポートする前に、重複するノードまたはエッジを削除してください。

グラフ構造の微調整

グラフをエクスポートするときに、特徴の処理方法やグラフの作成方法を変更して、モデルのパフォーマンスを向上させることができます。

ベストプラクティスを以下に示します。

- エッジプロパティがエッジのカテゴリの意味を持つ場合、場合によってはエッジタイプに変換する価値があります。
- 数値プロパティに使用されるデフォルトの正規化ポリシーは min-max ですが、場合によっては他の正規化ポリシーがうまくいく場合もある。model-HPO-configuration.json 要素のリスト。 の説明に従って、プロパティを前処理し、正規化ポリシーを変更できます。
- エクスポートプロセスでは、プロパティタイプに基づいて特徴タイプが自動的に生成されます。例えば、String プロパティをカテゴリ別特徴とし、Float および Int プロパティを数値特徴として処理します。必要に応じて、エクスポート後にとしタイプを変更できます (model-HPO-configuration.json 要素のリスト。)。

ハイパーパラメータの範囲とデフォルトを調整する

データ処理オペレーションは、グラフからハイパーパラメータ設定範囲を推測します。生成されたモデルのハイパーパラメータの範囲とデフォルトがグラフデータでうまく機能しない場合は、HPO 設定ファイルを編集して、独自のハイパーパラメータ調整戦略を作成できます。

ベストプラクティスを以下に示します。

- グラフが大きくなると、デフォルトの非表示次元サイズがすべての情報を含むのに十分な大きさではない場合があります。num-hidden ハイパーパラメータを変更して非表示の寸法サイズを制御することもできます。
- Knowledge Graph Embedding (KGE) モデルの場合、グラフの構造と予算に応じて、使用されている特定のモデルを変更したい場合があります。

  TrainsE モデルは、 1 対多 (1-N)、多対 1 (N-1)、多対多 (N-N) の関係を扱うのが困難です。DistMult モデルは対称関係を扱うのが困難です。RotatE はあらゆる種類のリレーションのモデリングが得意ですが、トレーニング中は TrainsE および DistMult よりも高価です。
- 場合によっては、ノード識別とノード特徴情報の両方が重要な場合は、`concat-node-embed` を使用してNeptune ML モデルに、その特徴と初期埋め込みを連結して、ノードの初期表現を取得するように指示します。
- 一部のハイパーパラメータに対して適度に良好なパフォーマンスが得られる場合は、それらの結果に応じてハイパーパラメータサーチスペースを調整できます。

Neptune ML でのモデルトレーニングプロセスの早期停止

早期停止により、モデルのパフォーマンスを低下させることなく、モデルトレーニングのランタイムと関連コストを大幅に削減できます。また、モデルがトレーニングデータに過適合するのを防ぎます。

早期停止は、検証セットのパフォーマンスの定期的な測定値に依存します。最初は、トレーニングが進むにつれてパフォーマンスは向上しますが、モデルが過適合となると、再び低下し始めます。早期停止機能は、モデルが過適合を開始し、その時点でモデルトレーニングを停止する点を特定します。

Neptune ML は検証メトリクスの呼び出しを監視し、最新の検証メトリクスを最後の **n** 評価 (**n** は `window-for-early-stop` パラメータを使用して設定した数字) を検証メトリックの平均と比較します。検証メトリクスがその平均よりも悪くなるとすぐに、Neptune ML はモデルトレーニングを停止し、それまでに最適なモデルを保存します。

次のパラメータを使用して、早期停止を制御できます。

- **`window-for-early-stop`** — このパラメータの値は、早期停止を決定する際に平均する最近の検証スコアの数を指定する整数です。デフォルト値は、「3」です。
- **`enable-early-stop`** — このブール値パラメータを使用して、早期停止機能をオフにします。デフォルトでは、この値は `true` です。

Neptune ML での HPO プロセスの早期停止

Neptune ML の早期停止機能は、SageMaker HPO ウォームスタート機能を使用して、他のトレーニングジョブと比較してうまく機能しないトレーニングジョブを停止します。これにより、コストを削減し、HPO の品質を向上させることができます。

この仕組みについては、ウォームスタートのハイパーパラメータチューニングジョブを実行するを参照してください。

ウォームスタートは、以前のトレーニングジョブから学習した情報を後続のトレーニングジョブに渡す機能を提供し、次の 2 つの異なる利点があります。

- まず、以前の調整ジョブの結果は、新しいトレーニングジョブで検索するハイパーパラメータの良い組み合わせを選択する目的で使用されます。
- 第 2 に、早期停止により多くのモデル実行にアクセスできるため、チューニング時間が短縮されます。

この機能は Neptune ML で自動的に有効になり、モデルトレーニング時間とパフォーマンスのバランスを取ることができます。現在のモデルのパフォーマンスに満足すれば、そのモデルを使用できます。そうでない場合は、以前の実行の結果でウォームスタートされるより多くの HPO を実行して、より良いモデルを発見します。

プロフェッショナルサポートサービスを受ける

AWS は、Neptune プロジェクトでの機械学習における問題に対処するための専門的なサポートサービスを提供します。行き詰まったら、AWS サポートをご利用ください。

トレーニング済みモデルを使用して新しいモデルのアーティファクトを生成する

Neptune ML モデル変換コマンドを使用すると、事前にトレーニングしたモデルパラメータを使用して、処理されたグラフデータに対するノード埋め込みなどのモデルアーティファクトを計算できます。

増分推論のためのモデル変換

増分モデル推論ワークフローで、Neptune からエクスポートした更新されたグラフデータを処理した後、次のような curl (または awscurl) コマンドを使用してモデル変換ジョブを開始できます。

```
curl \
  -X POST https://(your Neptune endpoint)/ml/modeltransform
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique model-training job ID)",
        "dataProcessingJobId" : "(the data-processing job-id of a completed job)",
        "mlModelTrainingJobId": "(the ML model training job-id)",
        "modelTransformOutputS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-
transform/"
      }'
```

その後、このジョブの ID を create-endpoints API 呼び出しに渡して、新しいエンドポイントを作成するか、このジョブによって生成された新しいモデルアーティファクトで既存のエンドポイントを更新できます。これにより、新しいエンドポイントまたは更新されたエンドポイントが、更新されたグラフデータのモデル予測を提供できるようになります。

あらゆるトレーニングジョブのモデル変換

Neptune ML モデルトレーニング中に起動された任意の SageMaker トレーニングジョブのモデルアーティファクトを生成する `trainingJobName` パラメータを指定することもできます。Neptune ML モデルトレーニングジョブは潜在的に多くの SageMaker トレーニングジョブを起動できるため、これらのいずれかの SageMaker トレーニングジョブに基づいて推論エンドポイントを柔軟に作成できます。

例:

```
curl \
```

```
  -X POST https://(your Neptune endpoint)/ml/modeltransform
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique model-training job ID)",
        "trainingJobName" : "(name a completed SageMaker training job)",
        "modelTransformOutputS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-
transform/"
      }'
```

元のトレーニングジョブがユーザー指定のカスタムモデルを対象とした場合は、モデル変換を呼び出す際に customModelTransformParameters オブジェクトを含める必要があります。カスタムモデルの実装と使用方法の詳細については、Neptune ML のカスタムモデル を参照してください。

> **Note**
>
> modeltransform コマンドは常に、そのトレーニングに最適な SageMaker トレーニングジョブでモデル変換を実行します。

モデル変換ジョブについては、モデルトランスフォームコマンド を参照してください。

Neptune MLのモデルトレーニングによって生成されたアーティファクト

モデルトレーニングの後、Neptune ML は最適なトレーニング済みモデルパラメータを使用して、推論エンドポイントの起動とモデル予測の提供に必要なモデルアーティファクトを生成します。これらのアーティファクトは、トレーニングジョブによってパッケージ化され、最適な SageMaker トレーニングジョブの Amazon S3 出力場所に保存されます。

次のセクションでは、さまざまなタスクのモデルアーティファクトに含まれる内容と、モデル変換コマンドで既存のトレーニング済みモデルを使用して、新しいグラフデータでもアーティファクトを生成する方法について説明します。

異なるタスクに対して生成されたアーティファクト

トレーニングプロセスによって生成されるモデルアーティファクトの内容は、対象とする機械学習タスクによって異なります。

- ノードの分類と回帰 — ノードプロパティ予測の場合、アーティファクトには、モデルのパラメータ、GNN エンコーダからのノード埋め込み、トレーニンググラフのノードのモデル予測、および推論エンドポイントの設定ファイルが含まれます。ノードの分類およびノード回帰タスクでは、モデル予測は、トレーニング時に存在するノードについては事前に計算されて、クエリのレイテンシーが削減されます。
- エッジ分類と回帰 — エッジプロパティ予測の場合、アーティファクトにはモデルパラメータとノードの埋め込みも含まれます。エッジの始点および終点の頂点の埋め込みにモデルデコーダを適用することにより、エッジ分類またはエッジ回帰予測を計算するため、モデルデコーダのパラメータは推論において特に重要です。
- リンク予測 — リンク予測では、エッジプロパティ予測で生成されたアーティファクトに加えて、トレーニンググラフが予測を実行する必要があるため、DGL グラフもアーティファクトとして含まれます。リンク予測の目的は、始点頂点と結合してグラフ内の特定のタイプのエッジを形成する可能性が高い終点頂点を予測することです。これを行うために、始点頂点のノード埋め込みとエッジタイプの学習された表現を、可能なすべての終点頂点のノード埋め込みと結合して、各終点頂点のエッジ尤度スコアを生成します。次に、スコアがソートされ、潜在的な終点頂点がランク付けされ、上位候補が返されます。

タスクタイプごとに、DGL の Graph Neural Network モデルの重みがモデルアーティファクトに保存されます。これにより、Neptune ML は、事前に計算された予測と埋め込み(トランスダクティブ推

論) を使用してレイテンシーを短縮するだけでなく、グラフの変化に応じて新しいモデル出力を計算できます (帰納的推論)。

新しいモデルアーティファクトの生成

Neptune ML でのモデルトレーニング後に生成されたモデルアーティファクトは、トレーニンググラフに直接結びつけられます。つまり、事前に計算された埋め込みと予測は、元のトレーニンググラフにあったエンティティについてのみ存在します。Neptune ML エンドポイントの帰納的推論モードでは、新しいエンティティの予測をリアルタイムで計算できますが、エンドポイントにクエリを実行せずに新しいエンティティのバッチ予測を生成したい場合があります。

グラフに追加された新しいエンティティのバッチモデル予測を取得するには、新しいグラフデータに対して新しいモデルアーティファクトを再計算する必要があります。これは、`modeltransform` コマンドを使用して行われます。エンドポイントを設定せずにバッチ予測のみを行いたい場合や、すべての予測を生成してグラフに書き戻せるようにする場合には、`modeltransform` コマンドを使用します。

モデルトレーニングは学習プロセスの最後にモデル変換を暗黙的に実行するため、モデルのアーティファクトはトレーニングジョブによってトレーニンググラフデータ上で常に再計算されます。ただし、`modeltransform` コマンドは、モデルのトレーニングに使用されなかったグラフデータのモデルアーティファクトを計算することもできます。そのためには、新しいグラフデータを元のグラフデータと同じ特徴エンコーディングを使用して処理し、同じグラフスキーマに従う必要があります。

これを行うには、まず、元のトレーニンググラフデータで実行されるデータ処理ジョブのクローンである新しいデータ処理ジョブを作成し、新しいグラフデータで実行します (Neptune ML の更新されたグラフデータの処理 を参照)。次に、新 `dataProcessingJobId` および旧 `modelTrainingJobId` で `modeltransform` コマンドを呼び出し、更新されたグラフデータのモデルアーティファクトを再計算します。

ノードプロパティ予測では、元のトレーニンググラフに存在していたノードであっても、ノードの埋め込みと予測は新しいグラフデータで再計算されます。

エッジプロパティ予測とリンク予測では、ノードの埋め込みも再計算され、既存のノード埋め込みも同様に上書きされます。ノードの埋め込みを再計算するために、Neptune ML は、前のトレーニング済みモデルから学習済み GNN エンコーダを新しい特徴を持つ新しいグラフデータのノードに適用します。

特徴を持たないノードの場合、元のモデルトレーニングから学習した初期表現が再使用されます。特徴を持たず、元のトレーニンググラフに存在しなかった新しいノードの場合、Neptune ML は、元の

トレーニンググラフに存在するそのノードタイプの学習済み初期ノード表現の平均としてそれらの表現を初期化します。これにより、特徴を持たない新しいノードが多数ある場合、モデル予測のパフォーマンスが低下する可能性があります。これは、すべてそのノードタイプの平均初期埋め込みに初期化されるためです。

`concat-node-embed` を true に設定してモデルがトレーニングされている場合、ノード特徴と学習可能な初期表現を連結して、初期ノード表現が作成されます。したがって、更新されたグラフでは、新しいノードの初期ノード表現では、新しいノード特徴と連結された平均初期ノード埋め込みも使用されます。

さらに、ノードの削除は現在サポートされていません。更新されたグラフでノードが削除されている場合は、更新されたグラフデータでモデルを再トレーニングする必要があります。

モデルのアーティファクトを再計算すると、新しいグラフで学習したモデルパラメータが再利用され、新しいグラフが古いグラフと非常によく似ている場合にのみ実行されます。新しいグラフが十分に類似していない場合は、新しいグラフデータで同様のモデル性能を得るために、モデルを再トレーニングする必要があります。十分に類似した構成は、グラフデータの構造によって異なりますが、経験則として、新しいデータと元のトレーニンググラフデータの差異が 10 ~ 20% を超える場合は、モデルを再トレーニングする必要があります。

すべてのノードに特徴があるグラフの場合、しきい値の上端 (20% の差異) が適用されますが、多くのノードに特徴がなく、グラフに追加された新しいノードにプロパティがないグラフでは、下端 (10% の差異) が高すぎることがあります。

モデル変換ジョブについては、モデルトランスフォームコマンド を参照してください。

Neptune ML のカスタムモデル

Neptune ML では、Python を使用して独自のカスタムモデルの実装を定義できます。組み込みモデルの場合と同様に、Neptune ML インフラストラクチャを使用してカスタムモデルをトレーニングおよびデプロイし、グラフクエリを使用して予測を取得できます。

> **Note**
>
> リアルタイムの帰納的推論は、現在、カスタムモデルではサポートされていません。

Python で独自のカスタムモデルの実装を開始するには、Neptune ML ツールキットの例に従い、また Neptune ML ツールキットで提供されているモデルコンポーネントを使用します。詳細については次のセクションで説明します。

目次

Neptune ML のカスタムモデルの概要

Neptune ML でカスタムモデルを使用する時期

Neptune ML の組み込みモデルは、Neptune ML でサポートされているすべての標準タスクを処理しますが、特定のタスクのモデルをより細かく制御したい場合や、モデルトレーニングプロセスをカスタマイズしなければならない場合があります。たとえば、次のような場合は、カスタムモデルが適しています。

- 非常に大きなテキストモデルのテキスト機能の特徴エンコーディングは GPU で実行する必要があります。
- ディープグラフライブラリ (DGL) で開発された独自のカスタムグラフニューラルネットワーク (GNN) モデルを使いたい場合。
- ノード分類と回帰に表形式モデルまたはアンサンブルモデルを使用したい場合。

Neptune ML でカスタムモデルを開発して使用するためのワークフロー

Neptune ML でのカスタムモデルのサポートは、既存の Neptune ML ワークフローにシームレスに統合するように設計されています。Neptune ML のインフラストラクチャでソースモジュールでカスタムコードを実行してモデルをトレーニングすることで機能します。組み込みモードの場合と同様に、Neptune ML は自動的に SageMaker HyperParameter 調整ジョブを起動し、評価メトリクスに従って最適なモデルを選択します。次に、ソースモジュールで提供されている実装を使用して、デプロイ用のモデルアーティファクトを生成します。

カスタムモデルの場合、データのエクスポート、トレーニング設定、およびデータの前処理は、組み込みモデルの場合と同じです。

データの前処理の後は、Python を使用してカスタムモデルの実装を反復的かつ対話的に開発およびテストできるときです。モデルが実稼働状態になったら、作成された Python モジュールを次のようにして Amazon S3 にアップロードできます。

```
aws s3 cp --recursive (source path to module) s3://(bucket name)/(destination path for
 your module)
```

次に、モデルを本番環境にデプロイするためのノーマルなデフォルトまたはインクリメンタルデータワークフローを使うことができ、いくつかの違いがあります。

カスタムモデルを使用したモデルトレーニングの場合、カスタムコードが使用されていることを確認するために、Neptune ML モデルトレーニング API へ `customModelTrainingParameters` JSON

オブジェクトを提供する必要があります。customModelTrainingParameters オブジェクトのフィールドは次のとおりです。

- **sourceS3DirectoryPath** — (必須) モデルを実装する Python モジュールがある Amazon S3 の場所へのパス。これは、少なくともトレーニングスクリプト、変換スクリプト、および model-hpo-configuration.json ファイルを含む有効な既存の Amazon S3 の場所を指定しなければなりません。
- **trainingEntryPointScript** — (オプション) モデルトレーニングを実行し、固定ハイパーパラメーターを含むコマンドライン引数としてハイパーパラメーターを取るスクリプトのモジュール内のエントリポイントの名前。

  デフォルト: training.py。
- **transformEntryPointScript** — (オプション) モデルのデプロイに必要なモデルアーティファクトを計算するために、ハイパーパラメーター検索の最適なモデルが特定された後に実行されるスクリプトのモジュール内のエントリポイントの名前。コマンドライン引数なしで実行できるはずです。

  デフォルト: transform.py。

例:

```
curl \
  -X POST https://(your Neptune endpoint)/ml/modeltraining
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique model-training job ID)",
        "dataProcessingJobId" : "(the data-processing job-id of a completed job)",
        "trainModelS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-graph-
autotrainer"
        "modelName": "custom",
        "customModelTrainingParameters" : {
          "sourceS3DirectoryPath": "s3://(your Amazon S3 bucket)/(path to your Python
 module)",
          "trainingEntryPointScript": "(your training script entry-point name in the
 Python module)",
          "transformEntryPointScript": "(your transform script entry-point name in the
 Python module)"
        }
      }'
```

同様に、カスタムモデル変換を有効にするには、トレーニングジョブの保存されたモデルパラメータと互換性のあるフィールド値を持つ Neptune ML モデル変換 API へcustomModelTransformParameters JSON オブジェクトを提供する必要があります。customModelTransformParameters オブジェクトには、次のフィールドが含まれます。

- **sourceS3DirectoryPath** — (必須) モデルを実装する Python モジュールがある Amazon S3 の場所へのパス。これは、少なくともトレーニングスクリプト、変換スクリプト、および model-hpo-configuration.json ファイルを含む有効な既存の Amazon S3 の場所を指定しなければなりません。
- **transformEntryPointScript** — (オプション) モデルのデプロイに必要なモデルアーティファクトを計算するために、ハイパーパラメーター検索の最適なモデルが特定された後に実行されるスクリプトのモジュール内のエントリポイントの名前。コマンドライン引数なしで実行できるはずです。

  デフォルト: transform.py。

例:

```
curl \
  -X POST https://(your Neptune endpoint)/ml/modeltransform
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique model-training job ID)",
        "trainingJobName" : "(name of a completed SageMaker training job)",
        "modelTransformOutputS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-
transform/"
        "customModelTransformParameters" : {
          "sourceS3DirectoryPath": "s3://(your Amazon S3 bucket)/(path to your Python
 module)",
          "transformEntryPointScript": "(your transform script entry-point name in the
 Python module)"
        }
     }'
```

Neptune ML でのカスタムモデル開発

カスタムモデルの開発を開始する良い方法は、以下を実行することです。Neptune ML ツールキットの例に従ってトレーニングモジュールを構造化して記述します。Neptune ML ツールキットは、モジュール化されたグラフ ML モデルコンポーネントをスタックしてカスタムモデルを作成するために使用できるモデルズーに実装します。

さらに、ツールキットには、モデルトレーニングおよびモデル変換時に必要なアーティファクトを生成するのに役立つユーティリティ関数が用意されています。この Python パッケージは、カスタム実装にインポートできます。ツールキットで提供されている関数またはモジュールは、Neptune ML トレーニング環境でも使用できます。

Python モジュールに外部依存関係が追加されている場合は、モジュールのディレクトリにある `requirements.txt` ファイルを指定してこれらの追加依存関係を含めることができます。`requirements.txt` ファイルに一覧表示されているこのパッケージは、トレーニングスクリプトが実行される前にインストールされます。

少なくとも、カスタムモデルを実装する Python モジュールには、次のものが含まれている必要があります。

- トレーニングスクリプトのエントリポイント
- トランスフォームスクリプトのエントリポイント
- `model-hpo-configuration.json` ファイル

Neptune ML でのカスタムモデルトレーニングスクリプト開発

カスタムモデルトレーニングスクリプトは、Neptune ML ツールキット`train.py`の例のような実行可能な Python スクリプトである必要があります。ハイパーパラメータの名前と値をコマンドライン引数として受け入れる必要があります。モデルトレーニング中、ハイパーパラメータ名は `model-hpo-configuration.json` ファイルから取得します。ハイパーパラメータの値は、ハイパーパラメータが調整可能な場合は有効なハイパーパラメータの範囲内に入るか、調整可能でない場合はデフォルトのハイパーパラメータ値を使用します。

トレーニングスクリプトは、次のような構文を使用して SageMaker トレーニングインスタンスで実行されます。

```
python3 (script entry point) --(1st parameter) (1st value) --(2nd parameter) (2nd
 value) (...)
```

すべてのタスクについて、Neptune ML AutoTrainer は、指定したハイパーパラメータに加えて、いくつかの必須パラメータをトレーニングスクリプトに送信しますが、適切に動作させるには、スクリプトがこれらの追加パラメータを処理できる必要があります。

これらの追加必須パラメータは、タスクによって多少異なります。

ノード分類またはノード回帰の場合

- **task** — Neptune ML によって内部的に使用されるタスクタイプ。ノード分類の場合、これは node_class であり、ノード回帰については node_regression です。
- **model** — Neptune ML によって内部的に使用されるモデル名。この場合、custom です。
- **name** — Neptune ML によって内部的に使用されるタスクの名前。この場合、ノード分類の場合 node_class-custom で、ノード回帰の場合 node_regression-custom です。
- **target_ntype** — 分類または回帰のノードタイプの名前。
- **property** — 分類または回帰のノードプロパティの名前。

リンク予測の場合

- **task** — Neptune ML によって内部的に使用されるタスクタイプ。リンク予測の場合、これは link_predict です。
- **model** — Neptune ML によって内部的に使用されるモデル名。この場合、custom です。
- **name** — Neptune ML によって内部的に使用されるタスク名。この場合、link_predict-custom です。

エッジ分類またはエッジ回帰の場合

- **task** — Neptune ML によって内部的に使用されるタスクタイプ。エッジ分類の場合、これは edge_class であり、エッジ回帰については edge_regression です。
- **model** — Neptune ML によって内部的に使用されるモデル名。この場合、custom です。
- **name** — Neptune ML によって内部的に使用されるタスクの名前。この場合、エッジ分類の場合 edge_class-custom で、エッジ回帰の場合 edge_regression-custom です。
- **target_etype** — 分類または回帰のエッジタイプの名前。
- **property** — 分類または回帰のエッジプロパティの名前。

スクリプトは、モデルのパラメータと、トレーニングの終了時に必要となるその他のアーティファクトを保存する必要があります。

Neptune ML Toolkit ユーティリティ関数を使用して、処理されたグラフデータの場所、モデルパラメータを保存する場所、およびトレーニングインスタンスで使用可能な GPU デバイスを決定できます。フレームワークの使用の詳細については、これらのユーティリティ関数の使用方法の例を示す train.py サンプルトレーニングスクリプトを参照してください。

Neptune ML でのカスタムモデル変換スクリプト開発

モデルの再トレーニングを行わずに進化するグラフのモデル推論のための Neptune ML 増分ワークフローを利用するには、トランスフォームスクリプトが必要です。モデルのデプロイに必要なすべてのアーティファクトがトレーニングスクリプトによって生成された場合でも、モデルを再トレーニングせずに更新されたモデルを生成する場合は、変換スクリプトを提供する必要があります。

> **Note**
>
> リアルタイムの帰納的推論は、現在、カスタムモデルではサポートされていません。

カスタムモデルトレーニングスクリプトは、Neptune ML ツールキット transform.py の例スクリプトのような実行可能な Python スクリプトである必要があります。このスクリプトは、コマンドライン引数なしでモデルトレーニング中に呼び出されるため、スクリプトが受け入れるコマンドライン引数はデフォルトである必要があります。

このスクリプトは、次のような構文で SageMaker トレーニングインスタンスで実行されます。

```
python3 (your transform script entry point)
```

トランスフォームスクリプトには、次のようなさまざまな情報が必要です。

- 処理されたグラフデータの場所。
- モデルパラメータが保存され、新しいモデルアーティファクトが保存される場所。
- そのインスタンスで使用可能なデバイス。
- 最適なモデルを生成したハイパーパラメータ。

これらの入力は、スクリプトが呼び出すことができる Neptune ML ユーティリティ関数を使用して取得されます。その方法の例としてツールキットのサンプル transform.py スクリプトを参照してください。

スクリプトは、ノード埋め込み、ノード ID マッピング、および各タスクのモデル展開に必要なその他のアーティファクトを保存する必要があります。さまざまな Neptune ML タスクに必要なモデルアーティファクトの詳細については、モデルアーティファクトのドキュメントを参照してください。

Neptune MLのカスタム **model-hpo-configuration.json** ファイル

model-hpo-configuration.json ファイルは、カスタムモデルのハイパーパラメータを定義します。それは Neptune ML 組み込みモデルで使用される model-hpo-configuration.json ファイルと同じフォーマットで、Neptune ML によって自動生成され、処理されたデータの場所にアップロードされるバージョンよりも優先されます。

新しいハイパーパラメータをモデルに追加するときは、ハイパーパラメータがトレーニングスクリプトに渡されるように、このファイルにハイパーパラメータのエントリを追加する必要があります。

ハイパーパラメータを調整できるようにするには、ハイパーパラメータの範囲を指定し、tier-1、tier-2 または tier-3 param に設定します。ハイパーパラメータは、構成されたトレーニングジョブの総数で、階層内のハイパーパラメータの調整ができる場合に調整されます。調整不可能なパラメータの場合は、デフォルト値を指定し、ハイパーパラメータをファイルの fixed-param セクションに追加します。その方法の例についてはツールキットのサンプルサンプル model-hpo-configuration.json ファイルを参照してください。

また、SageMaker HyperParameter Optimization ジョブがトレーニングした候補モデルの評価に使用するメトリクス定義も指定する必要があります。これを行うには、次のように eval_metric JSON オブジェクトを model-hpo-configuration.json ファイルに追加します。

```
"eval_metric": {
  "tuning_objective": {
      "MetricName": "(metric_name)",
      "Type": "Maximize"
  },
  "metric_definitions": [
    {
      "Name": "(metric_name)",
      "Regex": "(metric regular expression)"
    }
  ]
```

```
},
```

eval_metric オブジェクトの metric_definitions 配列は、SageMaker がトレーニングインスタンスから抽出する各指標のメトリクス定義オブジェクトを一覧表示します。各メトリクス定義オブジェクトには、メトリクスの名前 (「精度」、「f1」など) を指定できる Name キーがあります。Regex キー を使用すると、特定のメトリックがトレーニングログに出力される方法に一致する正規表現文字列を指定できます。メトリクスの定義方法の詳細については、SageMaker HyperParameter Tuning ページを参照してください。

eval_metric の tuning_objective オブジェクトでは、metric_definitions のどのメトリクスをハイパーパラメータ最適化の目標指標として機能する評価メトリクスとして使用するかを指定できます。MetricName の値は、metric_definitions の定義の 1 つである Name の値と一致する必要があります。メトリクスが greater-is-better (「精度」など) または less-is-better (「平均二乗誤差」など) のどちらで解釈されるべきかに応じて、Type の値は「最大化」または「最小化」のいずれかになります。

model-hpo-configuration.json ファイルのこのセクションのエラーは、SageMaker HyperParameter Tuning ジョブが最適なモデルを選択できないため、Neptune ML モデルトレーニング API ジョブが失敗する可能性があります。

Neptune ML でのカスタムモデル実装のローカルテスト

Neptune ML ツールキット Conda 環境を使用して、モデルをテストおよび検証するために、コードをローカルで実行できます。Neptune ノートブックインスタンスで開発している場合、この Conda 環境は Neptune Notebook インスタンスにプレインストールされます。別のインスタンスで開発する場合は、Neptune ML ツールキットのローカルセットアップ手順に入っています。

Conda 環境は、モデルトレーニング API を呼び出す際にモデルが実行する環境を正確に再現します。トレーニングスクリプトと変換スクリプトの例はすべて、簡単にデバッグできるように、ローカル環境でスクリプトを実行するコマンドライン --local フラグを渡すことができます。これは、モデルの実装を対話的かつ反復的にテストできるので、独自のモデルを開発する際には良い方法です。Neptune ML プロダクショントレーニング環境でのモデルトレーニング中は、このパラメータは省略されます。

照会のための推論エンドポイントの作成

推論エンドポイントを使用すると、モデルトレーニングプロセスによって構築された 1 つの特定のモデルを照会できます。エンドポイントは、トレーニングプロセスで生成できた特定のタイプの最もパフォーマンスの高いモデルにアタッチします。エンドポイントは Neptune からの Gremlin クエリを受け入れ、クエリ内の入力に対するモデルの予測を返すことができます。推論エンドポイントを作成した後も、削除するまでアクティブなままになります。

Neptune ML の推論エンドポイントの管理

Neptune からエクスポートしたデータに関するモデルトレーニングを完了したら、`curl` (または`awscurl`) コマンドを使って、以下のような推論コマンドが記述されます。

```
curl \
  -X POST https://(your Neptune endpoint)/ml/endpoints
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique ID for the new endpoint)",
        "mlModelTrainingJobId": "(the model-training job-id of a completed job)"
      }'
```

また、ほぼ同じ方法で、完成したモデル変換ジョブによって作成されたモデルから推論エンドポイントを作成することもできます。

```
curl \
  -X POST https://(your Neptune endpoint)/ml/endpoints
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique ID for the new endpoint)",
        "mlModelTransformJobId": "(the model-transform job-id of a completed job)"
      }'
```

これらのコマンドの使用方法の詳細については、エンドポイントコマンド を参照してください。また、エンドポイントのステータスの取得方法、エンドポイントの削除方法、およびすべての推論エンドポイントの一覧表示方法に関する情報も記載されています。

Neptune ML での推論クエリ

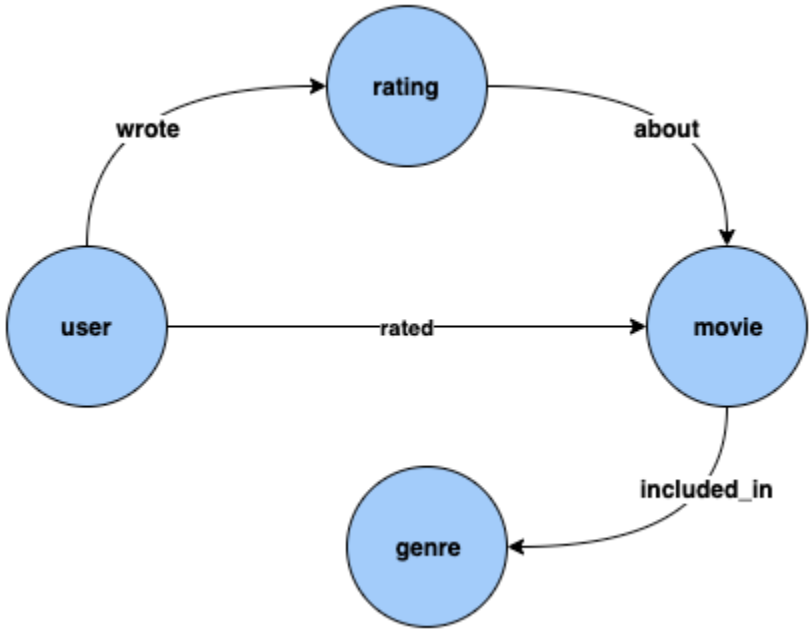
Gremlin または SPARQL のいずれかを使用して、Neptune ML 推論エンドポイントを照会できます。ただし、リアルタイムの帰納的推論は、現時点では、Gremlin クエリでのみサポートされています。

Neptune ML の Gremlin 推論クエリ

Neptune ML 機能 で説明しているように、Neptune ML は次の種類の推論タスクを実行できるトレーニングモデルをサポートしています。

- ノード分類 — 頂点プロパティのカテゴリカル特徴を予測します。
- ノード回帰 — 頂点の数値プロパティを予測します。
- エッジ分類 — エッジプロパティのカテゴリ別特徴を予測します。
- エッジ回帰 — エッジの数値プロパティを予測します。
- リンク予測 — 始点ノードと送信エッジが与えられた終点ノード、または終点ノードと着信エッジを指定した始点ノードを予測します。

これらのさまざまなタスクを、GroupLens Reserch 提供の MovieLens 100k データセット を使用する例で示すことができます。このデータセットは、映画、ユーザー、およびユーザーによる映画の評価から構成され、そこから次のようなプロパティグラフを作成しました。



ノード分類: 上記のデータセットでは、Genre は、エッジ included_in によって頂点タイプ Movie に接続されている頂点タイプです。ただし、データセットを微調整して Genre を頂点タイプ Movie のカテゴリ別特徴にすると、ナレッジグラフに追加された新しい映画の推論の問題 Genre は、ノード分類モデルを使用して解くことができます。

ノード回帰: 頂点タイプ Rating を、timestamp および score のようなプロパティを持つと考えると、Rating に対する数値 Score を推論する問題は、ノード回帰モデルを使用して解くことができます。

エッジ分類: 同様に、Rated エッジに関して、Love、Like、Dislike、Neutral、Hate のうち 1 つの値となる可能性があるプロパティ Scale がある場合、新しい映画/レーティング Rated エッジに対する推論の問題 Scale は、エッジ分類モデルを使用して解けます。

エッジ回帰: 同様に、同じように Rated エッジに関して、レーティングの数値を保持する Score プロパティがある場合、これはエッジ回帰モデルから推測できます。

リンク予測: 特定の映画を評価する可能性が最も高い上位 10 人のユーザーを見つけたり、特定のユーザーが評価する可能性が最も高い上位 10 本の映画を見つけたりするなどの問題は、リンク予測に該当します。

> **Note**
>
> Neptune ML ユースケースについては、各ユースケースについて実践的に理解できるように設計された非常に豊富なノートブックセットがあります。Neptune MLAWS CloudFormation テンプレート を使用して Neptune ML クラスタを作成すると、これらのノートブックを Neptune クラスターと一緒に作成することができます。これらのノートブックは、GitHub でも利用できます。

トピック

- Gremlin 推論クエリで使用される Neptune ML 述語
- Neptune ML の Gremlin ノード分類クエリ
- Neptune ML での Gremlin ノード回帰クエリ
- Neptune ML の Gremlin エッジ分類クエリ
- Neptune ML での Gremlin エッジ回帰クエリ
- Neptune ML のリンク予測モデルを使用した Gremlin リンク予測クエリ
- Neptune ML Gremlin 推論クエリの例外のリスト

Gremlin 推論クエリで使用される Neptune ML 述語

Neptune#ml.deterministic

この述語は、帰納的推論クエリ、つまり Neptune#ml.inductiveInference 述語を含むクエリではオプションです。

帰納的推論を使用する場合、Neptune エンジンはトレーニング済みの GNN モデルを評価するための適切なサブグラフを作成します。このサブグラフの要件は、最終モデルのパラメータによって異なります。具体的には、num-layer パラメータによってターゲットノードまたはエッジからのトラバーサルホップ数が決まり、fanouts パラメータによって各ホップでサンプリングするネイバーの数を指定します (HPO パラメータを参照)。

デフォルトでは、帰納的推論クエリは非決定論的モードで実行されます。このモードでは、Neptune は近傍をランダムに構築します。予測を行う場合、この通常のランダム近傍サンプリングによって予測が異なる場合があります。

帰納的推論クエリに Neptune#ml.deterministic を含めると、Neptune エンジンは、同じクエリを複数回呼び出しても毎回同じ結果が返されるように、決定論的な方法で近傍をサンプリングしようとします。ただし、基礎となるグラフの変更や分散システムのアーティファクトによって変動が生じる可能性があるため、結果が完全に決定的であることは保証できません。

次のように、クエリに Neptune#ml.deterministic 述語を含めます。

```
.with("Neptune#ml.deterministic")
```

Neptune#ml.deterministic 述語が Neptune#ml.inductiveInference も含まないクエリに含まれている場合は、単に無視されます。

Neptune#ml.disableInductiveInferenceMetadataCache

この述語は、帰納的推論クエリ、つまり Neptune#ml.inductiveInference 述語を含むクエリではオプションです。

帰納的推論クエリでは、Neptune は Amazon S3 に保存されているメタデータファイルを使用して、近傍を構築する際のホップ数とファンアウトを決定します。Neptune は通常、Amazon S3 からファイルを繰り返し取得しないように、このモデルメタデータをキャッシュします。クエリに Neptune#ml.disableInductiveInferenceMetadataCache 述語を含めることでキャッシュを無効にできます。Neptune が Amazon S3 から直接メタデータを取得するほうが時間がかかる場合が

ありますが、再トレーニングまたは変換後に SageMaker エンドポイントが更新され、キャッシュが古い場合に役立ちます。

次のように、クエリに Neptune#ml.disableInductiveInferenceMetadataCache 述語を含めます。

```
.with("Neptune#ml.disableInductiveInferenceMetadataCache")
```

Jupyter ノートブックでサンプルクエリがどのように表示されるかを次に示します。

```
%%gremlin
g.with("Neptune#ml.endpoint", "ep1")
 .with("Neptune#ml.iamRoleArn", "arn:aws:iam::123456789012:role/NeptuneMLRole")
 .with("Neptune#ml.disableInductiveInferenceMetadataCache")
 .V('101').properties("rating")
 .with("Neptune#ml.regression")
 .with("Neptune#ml.inductiveInference")
```

Neptune#ml.endpoint

Neptune#ml.endpoint 述語は、必要に応じて、with() 手順に従い推論エンドポイントを指定するために使用します。

```
 .with("Neptune#ml.endpoint", "the model's SageMaker inference endpoint")
```

エンドポイントは、id またはその URL のいずれかで識別できます。例:

```
 .with( "Neptune#ml.endpoint", "node-classification-movie-lens-endpoint" )
```

または:

```
 .with( "Neptune#ml.endpoint", "https://runtime.sagemaker.us-east-1.amazonaws.com/
endpoints/node-classification-movie-lens-endpoint/invocations" )
```

> **Note**
>
> Neptune DB クラスターのパラメータグループで neptune_ml_endpoint パラメータをエンドポイント id または URL に設定すれば、各クエリの Neptune#ml.endpoint 述語を含む必要はありません。

Neptune#ml.iamRoleArn

必要に応じて、with() ステップでSageMaker 実行 IAM ロールの ARN を指定するために Neptune#ml.iamRoleArn が使用されます。

```
.with("Neptune#ml.iamRoleArn", "the ARN for the SageMaker execution IAM role")
```

SageMaker 実行 IAM ロールを作成する方法については、カスタム NeptuneSageMakerIAMRole ロールを作成する を参照してください。

> **Note**
>
> Neptune DB クラスターのパラメータグループでneptune_ml_iam_role パラメータを SageMaker 実行 IAM ロールの ARN に設定すれば、各クエリの Neptune#ml.iamRoleArn 述語を含む必要はありません。

Neptune#ml.inductiveInference

Gremlin では、トランスダクティブ推論はデフォルトで有効です。リアルタイムの帰納的推論クエリを作成するには、次のように Neptune#ml.inductiveInference 述語を含めます。

```
.with("Neptune#ml.inductiveInference")
```

グラフが動的な場合、多くの場合、機能的推論は最善の選択ですが、グラフが静的な場合、トランスダクティブ推論の方が速くて効率的です。

Neptune#ml.limit

Neptune#ml.limit 述語は、必要に応じてエンティティごとに返される結果の数を制限します。

```
.with( "Neptune#ml.limit", 2 )
```

デフォルトでは、上限は 1 であり、設定できる最大数は 100 です。

Neptune#ml.threshold

Neptune#ml.threshold 述語はオプションで、結果スコアの切り捨てしきい値を設定します。

```
.with( "Neptune#ml.threshold", 0.5D )
```

これにより、指定したしきい値を下回るスコアを持つすべての結果を破棄できます。

Neptune#ml.classification

Neptune#ml.classification 述語は、ノード分類モデルの SageMaker エンドポイントから取得する必要があるプロパティを確立するために properties() ステップステップに追加されています。

```
.properties( "property key of the node classification
model" ).with( "Neptune#ml.classification" )
```

Neptune#ml.regression

Neptune#ml.regression 述語は、ノード回帰モデルの SageMaker エンドポイントから取得する必要があるプロパティを確立するために properties() ステップステップに添付されています。

```
.properties( "property key of the node regression
model" ).with( "Neptune#ml.regression" )
```

Neptune#ml.prediction

Neptune#ml.prediction 述語は、これがリンク予測クエリであることを確立するために in() および out() ステップに添付されています。

```
.in("edge label of the link prediction
model").with("Neptune#ml.prediction").hasLabel("target node label")
```

Neptune#ml.score

Neptune#ml.score 述語は Gremlin ノードまたはエッジ分類クエリで、機械学習の信頼スコアを取得するために使用されます。Neptune#ml.score 述語は、ノードまたはエッジ分類クエリの ML 信頼度スコアを取得するために、properties() ステップでクエリ述語とともに渡されます。

ノード分類の例については、その他のノード分類の例で、また、エッジ分類セクションのエッジ分類の例でご確認いただけます。

Neptune ML の Gremlin ノード分類クエリ

Neptune ML の Gremlin ノード分類について

- モデルは、頂点の 1 つのプロパティで学習されます。このプロパティの一意の値のセットは、ノードクラスのセット、または単にクラスと呼ばれます。

- 頂点のプロパティのノードクラスまたはカテゴリプロパティ値は、ノード分類モデルから推測できます。これは、このプロパティがまだ頂点にアタッチされていない場合に便利です。
- ノード分類モデルから 1 つ以上のクラスを取得するには、with() ステップで述語 Neptune#ml.classification を使い properties() ステップを設定する必要があります。出力形式は、それらが頂点プロパティである場合に予想されるものと似ています。

> **Note**
>
> ノード分類は、文字列プロパティ値でのみ機能します。つまり、0 や 1 などの数値プロパティ値はサポートされませんが、該当する文字列 "0" および "1" はサポートされます。同様に、ブールのプロパティ値 true および false は機能しませんが、"true" および "false" は機能します。

ノード分類クエリの例を次に示します。

```
g.with( "Neptune#ml.endpoint","node-classification-movie-lens-endpoint" )
 .with( "Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role" )
 .with( "Neptune#ml.limit", 2 )
 .with( "Neptune#ml.threshold", 0.5D )
 .V( "movie_1", "movie_2", "movie_3" )
 .properties("genre").with("Neptune#ml.classification")
```

このクエリの出力は、次のようになります。

```
==>vp[genre->Action]
==>vp[genre->Crime]
==>vp[genre->Comedy]
```

上記のクエリでは、V() および properties() ステップは次のように使用されます。

V() ステップには、ノード分類モデルからクラスを取得する頂点のセットが含まれます。

```
 .V( "movie_1", "movie_2", "movie_3" )
```

properties() ステップには、モデルがトレーニングされたキーが含まれており、これがノード分類 ML 推論クエリであることを示す .with("Neptune#ml.classification") があります。

現在、複数のプロパティキーは、properties().with("Neptune#ml.classification") ステップではサポートされていません。たとえば、次のクエリでは例外が発生します。

```
g.with("Neptune#ml.endpoint", "node-classification-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .V( "movie_1", "movie_2", "movie_3" )
 .properties("genre", "other_label").with("Neptune#ml.classification")
```

具体的なエラーメッセージについては、Neptune ML 例外の一覧を参照してください。

properties().with("Neptune#ml.classification") ステップは、以下のいずれかのステップと組み合わせて使用できます。

- value()
- value().is()
- hasValue()
- has(value,"")
- key()
- key().is()
- hasKey()
- has(key,"")
- path()

その他のノード分類クエリ

推論エンドポイントと対応する IAM ロールの両方が DB クラスターパラメータグループに保存されている場合、ノード分類クエリは次のようになります。

```
g.V("movie_1", "movie_2",
 "movie_3").properties("genre").with("Neptune#ml.classification")
```

union()ステップを使用して、クエリに頂点プロパティとクラスを混在させることができます。

```
g.with("Neptune#ml.endpoint","node-classification-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .V( "movie_1", "movie_2", "movie_3" )
 .union(
   properties("genre").with("Neptune#ml.classification"),
```

```
    properties("genre")
  )
```

次のような無制限クエリを作成することもできます。

```
g.with("Neptune#ml.endpoint","node-classification-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .V()
 .properties("genre").with("Neptune#ml.classification")
```

ノードクラスを頂点とともに取得するには、select() ステップと as() ステップを一緒に使用します。

```
g.with("Neptune#ml.endpoint","node-classification-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .V( "movie_1", "movie_2", "movie_3" ).as("vertex")
 .properties("genre").with("Neptune#ml.classification").as("properties")
 .select("vertex","properties")
```

次の例に示すように、ノードクラスでフィルタリングすることもできます。

```
g.with("Neptune#ml.endpoint", "node-classification-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .V( "movie_1", "movie_2", "movie_3" )
 .properties("genre").with("Neptune#ml.classification")
 .has(value, "Horror")

g.with("Neptune#ml.endpoint","node-classification-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .V( "movie_1", "movie_2", "movie_3" )
 .properties("genre").with("Neptune#ml.classification")
 .has(value, P.eq("Action"))

g.with("Neptune#ml.endpoint","node-classification-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .V( "movie_1", "movie_2", "movie_3" )
 .properties("genre").with("Neptune#ml.classification")
 .has(value, P.within("Action", "Horror"))
```

ノード分類の信頼スコアは、Neptune#ml.score 述語を使用して取得できます。

```
 g.with("Neptune#ml.endpoint","node-classification-movie-lens-endpoint")
```

```
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .V( "movie_1", "movie_2", "movie_3" )
 .properties("genre", "Neptune#ml.score").with("Neptune#ml.classification")
```

また、レスポンスは次のようになります。

```
==>vp[genre->Action]
==>vp[Neptune#ml.score->0.01234567]
==>vp[genre->Crime]
==>vp[Neptune#ml.score->0.543210]
==>vp[genre->Comedy]
==>vp[Neptune#ml.score->0.10101]
```

ノード分類クエリでの帰納的推論の使用

Jupyter ノートブックの既存のグラフに、次のように新しいノードを追加するとします。

```
%%gremlin
g.addV('label1').property(id,'101').as('newV')
 .V('1').as('oldV1')
 .V('2').as('oldV2')
 .addE('eLabel1').from('newV').to('oldV1')
 .addE('eLabel2').from('oldV2').to('newV')
```

その場合、帰納的推論クエリを使用して、新しいノードを反映したジャンルと信頼度スコアを取得できます。

```
%%gremlin
g.with("Neptune#ml.endpoint", "nc-ep")
 .with("Neptune#ml.iamRoleArn", "arn:aws:iam::123456789012:role/NeptuneMLRole")
 .V('101').properties("genre", "Neptune#ml.score")
 .with("Neptune#ml.classification")
 .with("Neptune#ml.inductiveInference")
```

ただし、クエリを複数回実行すると、結果が多少異なる可能性があります。

```
# First time
==>vp[genre->Action]
==>vp[Neptune#ml.score->0.12345678]

# Second time
==>vp[genre->Action]
```

```
==>vp[Neptune#ml.score->0.21365921]
```

同じクエリを決定論的にすることもできます。

```
%%gremlin
g.with("Neptune#ml.endpoint", "nc-ep")
 .with("Neptune#ml.iamRoleArn", "arn:aws:iam::123456789012:role/NeptuneMLRole")
 .V('101').properties("genre", "Neptune#ml.score")
 .with("Neptune#ml.classification")
 .with("Neptune#ml.inductiveInference")
 .with("Neptune#ml.deterministic")
```

その場合、結果は毎回ほぼ同じになります。

```
# First time
==>vp[genre->Action]
==>vp[Neptune#ml.score->0.12345678]
# Second time
==>vp[genre->Action]
==>vp[Neptune#ml.score->0.12345678]
```

Neptune ML での Gremlin ノード回帰クエリ

ノード回帰は、各ノードの回帰モデルから推測される値が数値であることを除いて、ノード分類に似ています。次の違いを除いて、ノード分類の場合と同じ Gremlin クエリをノード回帰に使用できます。

- 繰り返しますが、Neptune ML では、ノードは頂点を指します。
- `properties()` ステップは `properties().with("Neptune#ml.classification")` の代わりに `properties().with("Neptune#ml.regression")` 形式を取ります。
- `"Neptune#ml.limit"` と `"Neptune#ml.threshold"` 述語は適用されません。
- 値をフィルタリングするときは、数値を指定する必要があります。

頂点分類クエリの例を次に示します。

```
g.with("Neptune#ml.endpoint","node-regression-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn", "arn:aws:iam::0123456789:role/sagemaker-role")
 .V("movie_1","movie_2","movie_3")
 .properties("revenue").with("Neptune#ml.regression")
```

次の例に示すように、回帰モデルを使用して推論された値をフィルタリングできます。

```
g.with("Neptune#ml.endpoint","node-regression-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .V("movie_1","movie_2","movie_3")
 .properties("revenue").with("Neptune#ml.regression")
 .value().is(P.gte(1600000))

g.with("Neptune#ml.endpoint","node-regression-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .V("movie_1","movie_2","movie_3")
 .properties("revenue").with("Neptune#ml.regression")
 .hasValue(P.lte(1600000D))
```

ノード回帰クエリでの帰納的推論の使用

Jupyter ノートブックの既存のグラフに、次のように新しいノードを追加するとします。

```
%%gremlin
g.addV('label1').property(id,'101').as('newV')
 .V('1').as('oldV1')
 .V('2').as('oldV2')
 .addE('eLabel1').from('newV').to('oldV1')
 .addE('eLabel2').from('oldV2').to('newV')
```

その場合、帰納的推論クエリを使用して、新しいノードを考慮した評価を得ることができます。

```
%%gremlin
g.with("Neptune#ml.endpoint", "nr-ep")
 .with("Neptune#ml.iamRoleArn", "arn:aws:iam::123456789012:role/NeptuneMLRole")
 .V('101').properties("rating")
 .with("Neptune#ml.regression")
 .with("Neptune#ml.inductiveInference")
```

このクエリは決定論的ではないため、複数回実行すると、近傍によって多少異なる結果が返される可能性があります。

```
# First time
==>vp[rating->9.1]

# Second time
```

```
==>vp[rating->8.9]
```

より一貫性のある結果が必要な場合は、クエリを決定論的にすることもできます。

```
%%gremlin
g.with("Neptune#ml.endpoint", "nc-ep")
 .with("Neptune#ml.iamRoleArn", "arn:aws:iam::123456789012:role/NeptuneMLRole")
 .V('101').properties("rating")
 .with("Neptune#ml.regression")
 .with("Neptune#ml.inductiveInference")
 .with("Neptune#ml.deterministic")
```

これで、結果は毎回ほぼ同じになります。

```
# First time
==>vp[rating->9.1]

# Second time
==>vp[rating->9.1]
```

Neptune ML の Gremlin エッジ分類クエリ

Neptune MLにおける Gremlin エッジ分類について

- モデルは、エッジの 1 つのプロパティで学習されます。このプロパティの一意の値のセットは、クラスのセットと呼ばれます。
- エッジのクラスまたはカテゴリプロパティ値は、エッジ分類モデルから推測できます。これは、このプロパティがエッジにまだ添付されていない場合に便利です。
- エッジ分類モデルから 1 つ以上のクラスを取得するには、`with()` ステップで述語 `"Neptune#ml.classification"` を使い `properties()` ステップを設定する必要があります。出力形式は、それらがエッジプロパティである場合に予想されるものと似ています。

> **Note**
>
> エッジ分類は、文字列プロパティ値でのみ機能します。つまり、`0` や `1` などの数値プロパティ値はサポートされませんが、該当する文字列 `"0"` および `"1"` はサポートされます。同様に、ブールのプロパティ値 `true` および `false` は機能しませんが、`"true"` および `"false"` は機能します。

次に示すのは、Neptune#ml.score 述語を使用して信頼スコアを要求するエッジ分類クエリの例です。

```
g.with("Neptune#ml.endpoint","edge-classification-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .E("relationship_1","relationship_2","relationship_3")
 .properties("knows_by", "Neptune#ml.score").with("Neptune#ml.classification")
```

また、レスポンスは次のようになります。

```
==>p[knows_by->"Family"]
==>p[Neptune#ml.score->0.01234567]
==>p[knows_by->"Friends"]
==>p[Neptune#ml.score->0.543210]
==>p[knows_by->"Colleagues"]
==>p[Neptune#ml.score->0.10101]
```

Gremlin エッジ分類クエリの構文

User は先頭と末尾のノードであり、Relationship はそれらを接続するエッジである単純なグラフの場合、エッジ分類クエリの例は次のとおりです。

```
g.with("Neptune#ml.endpoint","edge-classification-social-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .E("relationship_1","relationship_2","relationship_3")
 .properties("knows_by").with("Neptune#ml.classification")
```

このクエリの出力は、次のようになります。

```
==>p[knows_by->"Family"]
==>p[knows_by->"Friends"]
==>p[knows_by->"Colleagues"]
```

上記のクエリでは、E() および properties() ステップは次のように使用されます。

- E() ステップには、エッジ分類モデルからクラスを取得するエッジのセットが含まれます。

  ```
  .E("relationship_1","relationship_2","relationship_3")
  ```

- properties() ステップには、モデルがトレーニングされたキーが含まれており、これがエッジ分類 ML 推論クエリであることを示す .with("Neptune#ml.classification") があります。

現在、複数のプロパティキーは、properties().with("Neptune#ml.classification") ステップではサポートされていません。たとえば、次のクエリでは、例外がスローされます。

```
g.with("Neptune#ml.endpoint","edge-classification-social-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .E("relationship_1","relationship_2","relationship_3")
 .properties("knows_by", "other_label").with("Neptune#ml.classification")
```

具体的なエラーメッセージについては、Neptune ML Gremlin 推論クエリの例外のリスト を参照してください。

properties().with("Neptune#ml.classification") ステップは、以下のいずれかのステップと組み合わせて使用できます。

- value()
- value().is()
- hasValue()
- has(value,"")
- key()
- key().is()
- hasKey()
- has(key,"")
- path()

エッジ分類クエリでの帰納的推論の使用

Jupyter ノートブックの既存のグラフに、次のように新しいエッジを追加するとします。

```
%%gremlin
g.V('1').as('fromV')
.V('2').as('toV')
.addE('eLabel1').from('fromV').to('toV').property(id, 'e101')
```

その場合、帰納的推論クエリを使用して、新しいエッジを考慮したスケールを得ることができます。

```
%%gremlin
g.with("Neptune#ml.endpoint", "ec-ep")
 .with("Neptune#ml.iamRoleArn", "arn:aws:iam::123456789012:role/NeptuneMLRole")
```

```
  .E('e101').properties("scale", "Neptune#ml.score")
  .with("Neptune#ml.classification")
  .with("Neptune#ml.inductiveInference")
```

このクエリは決定論的ではないため、複数回実行すると、ランダム近傍によって多少異なる結果が返される可能性があります。

```
# First time
==>vp[scale->Like]
==>vp[Neptune#ml.score->0.12345678]

# Second time
==>vp[scale->Like]
==>vp[Neptune#ml.score->0.21365921]
```

より一貫性のある結果が必要な場合は、クエリを決定論的にすることもできます。

```
%%gremlin
g.with("Neptune#ml.endpoint", "ec-ep")
  .with("Neptune#ml.iamRoleArn", "arn:aws:iam::123456789012:role/NeptuneMLRole")
  .E('e101').properties("scale", "Neptune#ml.score")
  .with("Neptune#ml.classification")
  .with("Neptune#ml.inductiveInference")
  .with("Neptune#ml.deterministic")
```

これで、クエリを実行するたびに結果がほぼ同じになります。

```
# First time
==>vp[scale->Like]
==>vp[Neptune#ml.score->0.12345678]

# Second time
==>vp[scale->Like]
==>vp[Neptune#ml.score->0.12345678]
```

Neptune ML での Gremlin エッジ回帰クエリ

エッジ回帰は、モデルから推測される値が数値であることを除いて、エッジ分類に似ています。エッジ回帰の場合、Neptune ML は分類と同じクエリをサポートします。

注意すべき重要な点は次のとおりです。

- このユースケースの properties() ステップを設定するために、ML 述語 "Neptune#ml.regression" を使用する必要があります。
- "Neptune#ml.limit" および "Neptune#ml.threshold" 述語はこのユースケースでは適用されません。
- 値をフィルタリングするには、値を数値として指定する必要があります。

Gremlin エッジ回帰クエリの構文

User は先頭ノード、Movie は末尾ノード、Rated はそれらを接続するエッジである単純なグラフの場合、エッジ Rated についてスコアと呼ばれる数値評価値を見つけるエッジ回帰クエリの例をここに示します。

```
g.with("Neptune#ml.endpoint","edge-regression-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .E("rating_1","rating_2","rating_3")
 .properties("score").with("Neptune#ml.regression")
```

ML 回帰モデルから推論された値でフィルタリングすることもできます。"rating_1"、"rating_2"、"rating_3" によって識別される既存の Rated エッジ (User から Movie へ) である、ここで、エッジプロパティ Score はこれらの評価には存在しません。9 以上のエッジに対して、次のようなクエリを使用して Score を推論できます。

```
g.with("Neptune#ml.endpoint","edge-regression-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .E("rating_1","rating_2","rating_3")
 .properties("score").with("Neptune#ml.regression")
 .value().is(P.gte(9))
```

エッジ回帰クエリでの帰納的推論の使用

Jupyter ノートブックの既存のグラフに、次のように新しいエッジを追加するとします。

```
%%gremlin
g.V('1').as('fromV')
.V('2').as('toV')
.addE('eLabel1').from('fromV').to('toV').property(id, 'e101')
```

この場合、帰納的推論クエリを使用して、新しいエッジを考慮したスコアを取得できます。

```
%%gremlin
g.with("Neptune#ml.endpoint", "er-ep")
 .with("Neptune#ml.iamRoleArn", "arn:aws:iam::123456789012:role/NeptuneMLRole")
 .E('e101').properties("score")
 .with("Neptune#ml.regression")
 .with("Neptune#ml.inductiveInference")
```

このクエリは決定論的ではないため、複数回実行すると、ランダム近傍によって多少異なる結果が返される可能性があります。

```
# First time
==>ep[score->96]

# Second time
==>ep[score->91]
```

より一貫性のある結果が必要な場合は、クエリを決定論的にすることもできます。

```
%%gremlin
g.with("Neptune#ml.endpoint", "er-ep")
 .with("Neptune#ml.iamRoleArn", "arn:aws:iam::123456789012:role/NeptuneMLRole")
 .E('e101').properties("score")
 .with("Neptune#ml.regression")
 .with("Neptune#ml.inductiveInference")
 .with("Neptune#ml.deterministic")
```

これで、クエリを実行するたびに結果がほぼ同じになります。

```
# First time
==>ep[score->96]

# Second time
==>ep[score->96]
```

Neptune ML のリンク予測モデルを使用した Gremlin リンク予測クエリ

リンク予測モデルでは、次のような問題が解けます。

- 先頭ノード予測: 頂点とエッジタイプを指定すると、その頂点からリンクする可能性が高い頂点はどれか?

- 末尾ノード予測: 頂点とエッジラベルを指定すると、その頂点へとリンクする可能性が高い頂点はどれか?

> **Note**
>
> エッジ予測は Neptune ML ではまだサポートされていません。

以下の例では、エッジ `Rated` でリンクされている `User` および `Movie` 頂点を持つ単純なグラフを考えてみましょう。

以下は、映画 `"movie_1"`、`"movie_2"` および `"movie_3"` を評価する可能性が最も高い上位 5 人のユーザーを予測するために使用されるサンプルヘッドノード予測クエリです。

```
g.with("Neptune#ml.endpoint","node-prediction-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .with("Neptune#ml.limit", 5)
 .V("movie_1", "movie_2", "movie_3")
 .in("rated").with("Neptune#ml.prediction").hasLabel("user")
```

これは、ユーザー `"user_1"` が評価する可能性が高い上位 5 本の映画を予測するために使用する、テールノード予測の類似のものです。

```
g.with("Neptune#ml.endpoint","node-prediction-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .V("user_1")
 .out("rated").with("Neptune#ml.prediction").hasLabel("movie")
```

エッジラベルと予測頂点ラベルの両方が必要です。いずれかが省略されると、例外がスローされます。たとえば、頂点ラベルが予測されていない次のクエリでは、例外がスローされます。

```
g.with("Neptune#ml.endpoint","node-prediction-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .V("user_1")
 .out("rated").with("Neptune#ml.prediction")
```

同様に、エッジラベルのない次のクエリでは、例外がスローされます。

```
g.with("Neptune#ml.endpoint","node-prediction-movie-lens-endpoint")
```

```
  .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
  .V("user_1")
  .out().with("Neptune#ml.prediction").hasLabel("movie")
```

これらの例外が返される具体的なエラーメッセージについては、Neptune ML 例外の一覧を参照してください。

その他のリンク予測クエリ

select() ステップと as(ステップを併用して、予測頂点を入力頂点とともに出力しすることができます。

```
g.with("Neptune#ml.endpoint","node-prediction-movie-lens-endpoint")
  .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
  .V("movie_1").as("source")
  .in("rated").with("Neptune#ml.prediction").hasLabel("user").as("target")
  .select("source","target")

g.with("Neptune#ml.endpoint","node-prediction-movie-lens-endpoint")
  .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
  .V("user_1").as("source")
  .out("rated").with("Neptune#ml.prediction").hasLabel("movie").as("target")
  .select("source","target")
```

次のような無制限クエリを作成できます。

```
g.with("Neptune#ml.endpoint","node-prediction-movie-lens-endpoint")
  .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
  .V("user_1")
  .out("rated").with("Neptune#ml.prediction").hasLabel("movie")

g.with("Neptune#ml.endpoint","node-prediction-movie-lens-endpoint")
  .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
  .V("movie_1")
  .in("rated").with("Neptune#ml.prediction").hasLabel("user")
```

リンク予測クエリでの帰納的推論の使用

Jupyter ノートブックの既存のグラフに、次のように新しいノードを追加するとします。

```
%%gremlin
g.addV('label1').property(id,'101').as('newV1')
```

```
  .addV('label2').property(id,'102').as('newV2')
  .V('1').as('oldV1')
  .V('2').as('oldV2')
  .addE('eLabel1').from('newV1').to('oldV1')
  .addE('eLabel2').from('oldV2').to('newV2')
```

その場合、帰納的推論クエリを使用して、新しいノードを考慮に入れたヘッドノードを予測できます。

```
%%gremlin
g.with("Neptune#ml.endpoint", "lp-ep")
 .with("Neptune#ml.iamRoleArn", "arn:aws:iam::123456789012:role/NeptuneMLRole")
 .V('101').out("eLabel1")
 .with("Neptune#ml.prediction")
 .with("Neptune#ml.inductiveInference")
 .hasLabel("label2")
```

結果:

```
==>V[2]
```

同様に、帰納的推論クエリを使用して、新しいノードを考慮に入れたテールノードを予測できます。

```
%%gremlin
g.with("Neptune#ml.endpoint", "lp-ep")
 .with("Neptune#ml.iamRoleArn", "arn:aws:iam::123456789012:role/NeptuneMLRole")
 .V('102').in("eLabel2")
 .with("Neptune#ml.prediction")
 .with("Neptune#ml.inductiveInference")
 .hasLabel("label1")
```

結果:

```
==>V[1]
```

Neptune ML Gremlin 推論クエリの例外のリスト

- **`BadRequestException`** — 指定されたロールの認証情報を読み込めません。

  メッセージ: `Unable to load credentials for role: *the specified IAM Role ARN*.`

- **BadRequestException** — 指定された IAM ロールは SageMaker エンドポイントを呼び出す権限がありません。

  メッセージ: `User: the specified IAM Role ARN is not authorized to perform: sagemaker:InvokeEndpoint on resource: the specified endpoint.`

- **BadRequestException** — 指定されたエンドポイントは存在しません。

  メッセージ: `Endpoint the specified endpoint not found.`

- **InternalFailureException** — Neptune ML のリアルタイム帰納的推論メタデータを Amazon S3 から取得することはできません。

  メッセージ: `Unable to fetch Neptune ML - Real-Time Inductive Inference metadata from S3. Check the permissions of the S3 bucket or if the Neptune instance can connect to S3.`

- **InternalFailureException** — Neptune ML は、Amazon S3 でリアルタイム帰納的推論のメタデータファイルを見つけることができません。

  メッセージ: `Neptune ML cannot find the metadata file for Real-Time Inductive Inference in S3.`

- **InvalidParameterException** — 指定されたエンドポイントは構文的に有効ではありません。

  メッセージ: `Invalid endpoint provided for external service query.`

- **InvalidParameterException** — 指定された SageMaker 実行 IAM ロール ARN は構文的に有効ではありません。

  メッセージ: `Invalid IAM role ARN provided for external service query.`

- **InvalidParameterException** — クエリの `properties()` ステップで複数のプロパティキーが指定されます。

  メッセージ: `ML inference queries are currently supported for one property key.`

- **InvalidParameterException** — クエリで複数のエッジラベルが指定されます。

  メッセージ: `ML inference are currently supported only with one edge label.`

- **InvalidParameterException** — クエリで複数の頂点ラベル制約が指定されます。

  メッセージ: `ML inference are currently supported only with one vertex label constraint.`

- **InvalidParameterException** — Neptune#ml.classification および Neptune#ml.regression 述語の両方が同じクエリに存在します。

  メッセージ: `Both regression and classification ML predicates cannot be specified in the query.`
- **InvalidParameterException** — リンク予測クエリの `in()` または `out()` ステップで複数のエッジラベルが指定されます。

  メッセージ: `ML inference are currently supported only with one edge label.`
- **InvalidParameterException** — Neptune#ml.score で複数のプロパティキーが指定されました。

  メッセージ: `Neptune ML inference queries are currently supported for one property key and one Neptune#ml.score property key.`
- **MissingParameterException** — エンドポイントがクエリで、または DB クラスターパラメータとして指定されていません。

  メッセージ: `No endpoint provided for external service query.`
- MissingParameterException — SageMaker 実行 IAM ロールがクエリで、または DB クラスターパラメーターとして指定されていません。

  メッセージ: `No IAM role ARN provided for external service query.`
- **MissingParameterException** — プロパティキーがクエリの `properties()` ステップにありません。

  メッセージ: `Property key needs to be specified using properties() step for ML inference queries.`
- **MissingParameterException** — リンク予測クエリの `in()` または `out()` ステップでエッジラベルが指定されていません。

  メッセージ: `Edge label needs to be specified while using in() or out() step for ML inference queries.`
- **MissingParameterException** — Neptune#ml.score でプロパティキーが指定されませんでした。

  メッセージ: `Property key needs to be specified along with Neptune#ml.score property key while using the properties() step for Neptune ML inference queries.`

- **UnsupportedOperationException** — both() ステップは、リンク予測クエリで使用されます。

  メッセージ: `ML inference queries are currently not supported with both() step.`
- **UnsupportedOperationException** — リンク予測クエリの in() または out() ステップとともにhas() ステップで予測された頂点ラベルが指定されませんでした。

  メッセージ: `Predicted vertex label needs to be specified using has() step for ML inference queries.`
- **UnsupportedOperationException** — Gremlin ML の帰納的推論クエリは、現在、最適化されていないステップではサポートされていません。

  メッセージ: `Neptune ML - Real-Time Inductive Inference queries are currently not supported with Gremlin steps which are not optimized for Neptune. Check the Neptune User Guide for a list of Neptune-optimized steps.`
- **UnsupportedOperationException** — Neptune ML 推論クエリは、現在、repeat ステップ内ではサポートされていません。

  メッセージ: `Neptune ML inference queries are currently not supported inside a repeat step.`
- **UnsupportedOperationException** — 現在、1 つの Gremlin クエリでサポートされている Neptune ML 推論クエリは 1 つだけです。

  メッセージ: `Neptune ML inference queries are currently supported only with one ML inference query per gremlin query.`

Neptune ML での SPARQL 推論クエリ

Neptune ML は、RDF グラフをプロパティグラフにマッピングし、ML タスクをモデル化します。現在、次のユースケースをサポートしています。

- オブジェクト分類 — オブジェクトのカテゴリ別特徴を予測します。
- オブジェクト回帰 — オブジェクトの数値プロパティを予測します。
- オブジェクト予測— 対象と関係が与えられたオブジェクトを予測します。
- 件名の予測 — オブジェクトと関係が与えられた対象を予測します。

> **Note**
>
> Neptune ML は SPARQL での主題分類と回帰のユースケースをサポートしていません。

SPARQL 推論クエリで使用されるNeptune ML 述語

SPARQL 推論では、次の述語が使用されます。

`neptune-ml:timeout` 述語

リモートサーバーとの接続のタイムアウトを指定します。サーバーが要求を満たすのにかかる最大時間であるクエリ要求のタイムアウトと混同しないでください。

クエリタイムアウトが、`neptune-ml:timeout` 述語発生で指定されたサービスタイムアウトの前に発生した場合は、サービス接続もキャンセルされます。

`neptune-ml:outputClass` 述語

`neptune-ml:outputClass` 述語は、オブジェクト予測の予測オブジェクトのクラスまたはサブジェクト予測の予測対象を定義するためにのみ使用されます。

`neptune-ml:outputScore` 述語

`neptune-ml:outputScore` 述語は、機械学習モデルの出力が正しい可能性を表す正の数です。

`neptune-ml:modelType` 述語

`neptune-ml:modelType` 述語は、学習する機械学習モデルのタイプを指定します。

- OBJECT_CLASSIFICATION
- OBJECT_REGRESSION
- OBJECT_PREDICTION
- SUBJECT_PREDICTION

neptune-ml:input 述語

neptune-ml:input 述語は、Neptune ML の入力として使用される URI のリストを指します。

neptune-ml:output 述語

neptune-ml:output 述語は、Neptune ML が結果を返す結合集合のリストを指します。

neptune-ml:predicate 述語

neptune-ml:predicate 述語は、実行されるタスクに応じて異なる方法で使用されます。

- オブジェクトまたはサブジェクト予測の場合: 述語のタイプ (エッジまたはリレーションシップタイプ) を定義します。
- オブジェクトの分類と回帰の場合: 予測するリテラル (プロパティ) を定義します。

neptune-ml:batchSize 述語

neptune-ml:batchSize はリモートサービスコールの入力サイズを指定します。

SPARQL オブジェクト分類の例

Neptune ML での SPARQL オブジェクト分類では、モデルは述語値の 1 つに基づいてトレーニングされます。これは、その述語が特定の主語にまだ存在していない場合に便利です。

オブジェクト分類モデルを使用すると、カテゴリカル述語値のみを推論できます。

次のクエリは、foaf:Person タイプのすべての入力の <http://www.example.org/team> 述語値を予測しようとします。

```
SELECT * WHERE { ?input a foaf:Person .
  SERVICE neptune-ml:inference {
    neptune-ml:config neptune-ml:modelType 'OBJECT_CLASSIFICATION' ;
                      neptune-ml:input ?input ;
                      neptune-ml:predicate <http://www.example.org/team> ;
```

```
                      neptune-ml:output ?output .
  }
}
```

このクエリは、次のようにカスタマイズできます。

```
SELECT * WHERE { ?input a foaf:Person .
  SERVICE neptune-ml:inference {
    neptune-ml:config neptune-ml:endpoint 'node-prediction-account-balance-endpoint' ;
                      neptune-ml:iamRoleArn 'arn:aws:iam::0123456789:role/sagemaker-
role' ;

                      neptune-ml:batchSize "40"^^xsd:integer ;
                      neptune-ml:timeout "1000"^^xsd:integer ;

                      neptune-ml:modelType 'OBJECT_CLASSIFICATION' ;
                      neptune-ml:input ?input ;
                      neptune-ml:predicate <http://www.example.org/team> ;
                      neptune-ml:output ?output .
  }
}
```

SPARQL オブジェクト回帰の例

オブジェクト回帰は、各ノードの回帰モデルから推定される数値の述語値を除いて、オブジェクト分類に似ています。オブジェクト回帰には、オブジェクト分類の場合と同じ SPARQL クエリを使用できます。ただし、the `Neptune#ml.limit` および `Neptune#ml.threshold` 述語は適用されません。

次のクエリは、`foaf:Person` タイプのすべての入力の <http://www.example.org/accountbalance> 述語値を予測しようとします。

```
SELECT * WHERE { ?input a foaf:Person .
  SERVICE neptune-ml:inference {
    neptune-ml:config neptune-ml:modelType 'OBJECT_REGRESSION' ;
                      neptune-ml:input ?input ;
                      neptune-ml:predicate <http://www.example.org/accountbalance> ;
                      neptune-ml:output ?output .
  }
}
```

このクエリは、次のようにカスタマイズできます。

```
SELECT * WHERE { ?input a foaf:Person .
  SERVICE neptune-ml:inference {
    neptune-ml:config neptune-ml:endpoint 'node-prediction-account-balance-endpoint' ;
                      neptune-ml:iamRoleArn 'arn:aws:iam::0123456789:role/sagemaker-
role' ;

                      neptune-ml:batchSize "40"^^xsd:integer ;
                      neptune-ml:timeout "1000"^^xsd:integer ;

                      neptune-ml:modelType 'OBJECT_REGRESSION' ;
                      neptune-ml:input ?input ;
                      neptune-ml:predicate <http://www.example.org/accountbalance> ;
                      neptune-ml:output ?output .
  }
}
```

SPARQL オブジェクト予測の例

オブジェクト予測は、指定されたサブジェクトと述語のオブジェクト値を予測します。

次のオブジェクト予測クエリは、タイプ `foaf:Person` の入力がどの映画を好むのかを予測します。

```
?x a foaf:Person .
?x   <http://www.example.org/likes> ?m .
?m a <http://www.example.org/movie> .

## Query
SELECT * WHERE { ?input a foaf:Person .
  SERVICE neptune-ml:inference {
    neptune-ml:config neptune-ml:modelType 'OBJECT_PREDICTION' ;
                      neptune-ml:input ?input ;
                      neptune-ml:predicate <http://www.example.org/likes> ;
                      neptune-ml:output ?output ;
                      neptune-ml:outputClass <http://www.example.org/movie> .
  }
}
```

クエリ自体は、次のようにカスタマイズできます。

```
SELECT * WHERE { ?input a foaf:Person .
  SERVICE neptune-ml:inference {
```

```
    neptune-ml:config neptune-ml:endpoint 'node-prediction-user-movie-prediction-
endpoint' ;
                      neptune-ml:iamRoleArn 'arn:aws:iam::0123456789:role/sagemaker-
role' ;

                      neptune-ml:limit "5"^^xsd:integer ;
                      neptune-ml:batchSize "40"^^xsd:integer ;
                      neptune-ml:threshold "0.1"^^xsd:double ;
                      neptune-ml:timeout "1000"^^xsd:integer ;
                      neptune-ml:outputScore ?score ;

                      neptune-ml:modelType 'OBJECT_PREDICTION' ;
                      neptune-ml:input ?input ;
                      neptune-ml:predicate <http://www.example.org/likes> ;
                      neptune-ml:output ?output ;
                      neptune-ml:outputClass <http://www.example.org/movie> .
  }
}
```

SPARQL サブジェクト予測の例

件名の予測は、述語とオブジェクトが与えられた主語を予測します。

たとえば、次のクエリは、特定の映画を視聴するのは誰か (タイプ `foaf:User`) を予測します。

```
SELECT * WHERE { ?input (a foaf:Movie) .
    SERVICE neptune-ml:inference {
        neptune-ml:config neptune-ml:modelType 'SUBJECT_PREDICTION' ;
                          neptune-ml:input ?input ;
                          neptune-ml:predicate <http://aws.amazon.com/neptune/csv2rdf/
object_Property/rated> ;
                          neptune-ml:output ?output ;
                          neptune-ml:outputClass <http://aws.amazon.com/neptune/
csv2rdf/class/User> ;        }
}
```

Neptune ML SPARQL 推論クエリの例外のリスト

- **`BadRequestException`** - メッセージ: `The SERVICE http://aws.amazon.com/neptune/vocab/v01/services/ml#inference expects at least 1 value for the parameter (parameter name), found zero.`。

- **BadRequestException** - メッセージ: The SERVICE http://aws.amazon.com/neptune/vocab/v01/services/ml#inference expects at most 1 value for the parameter *(parameter name)*, found *(a number)* values.。
- **BadRequestException** - メッセージ: Invalid predicate *(predicate name)* provided for external service http://aws.amazon.com/neptune/vocab/v01/services/ml#inference query.。
- **BadRequestException** -メッセージ: The SERVICE http://aws.amazon.com/neptune/vocab/v01/services/ml#inference expects the predicate *(predicate name)* to be defined。
- **BadRequestException** - メッセージ: The SERVICE http://aws.amazon.com/neptune/vocab/v01/services/ml#inference expects the value of (parameter) *(parameter name)* to be a variable, found: *(type)*"。
- **BadRequestException** -メッセージ: The SERVICE http://aws.amazon.com/neptune/vocab/v01/services/ml#inference expects the input *(parameter name)* to be a constant, found: *(type)*。
- **BadRequestException** -メッセージ: The SERVICE http://aws.amazon.com/neptune/vocab/v01/services/ml#inference is expected to return only 1 value。
- **BadRequestException** -メッセージ: "The SERVICE http://aws.amazon.com/neptune/vocab/v01/services/ml#inference only allows StatementPatternNodes。
- **BadRequestException** -メッセージ: The SERVICE http://aws.amazon.com/neptune/vocab/v01/services/ml#inference does not allow the predicate *(predicate name)*。
- **BadRequestException** -メッセージ: The SERVICE http://aws.amazon.com/neptune/vocab/v01/services/ml#inference predicates cannot be variables, found: *(type)*。
- **BadRequestException** -メッセージ: The SERVICE http://aws.amazon.com/neptune/vocab/v01/services/ml#inference predicates are expected to be part of the namespace *(namespace name)*, found: *(namespace name)*。

Neptune ML 管理 API リファレンス

目次

dataprocessing コマンドを使用したデータ処理

Neptune ML dataprocessing コマンドを実行して、データ処理ジョブの作成、ステータスの確認、停止、またはアクティブなデータ処理ジョブの一覧を表示します。

Neptune ML dataprocessing コマンドを使用したデータ処理ジョブの作成

新しいジョブを作成するための典型的なNeptune ML dataprocessing コマンドは次のようになります。

```
curl \
  -X POST https://(your Neptune endpoint)/ml/dataprocessing \
  -H 'Content-Type: application/json' \
  -d '{
        "inputDataS3Location" : "s3://(Amazon S3 bucket name)/(path to your input
 folder)",
        "id" : "(a job ID for the new job)",
        "processedDataS3Location" : "s3://(S3 bucket name)/(path to your output
 folder)"
      }'
```

インクリメンタル再処理を開始するコマンドは次のようになります。

```
curl \
  -X POST https://(your Neptune endpoint)/ml/dataprocessing \
  -H 'Content-Type: application/json' \
  -d '{
        "inputDataS3Location" : "s3://(Amazon S3 bucket name)/(path to your input
 folder)",
        "id" : "(a job ID for this job)",
        "processedDataS3Location" : "s3://(S3 bucket name)/(path to your output
 folder)"
        "previousDataProcessingJobId" : "(the job ID of a previously completed job to
 update)"
}'
```

dataprocessing ジョブ作成のパラメータ

- **id** — (オプション) 新しいジョブの一意の識別子。

  タイプ: 文字列。デフォルト値: 自動生成された UUID。

- **previousDataProcessingJobId** — (オプション) 以前のバージョンのデータで実行された完了したデータ処理ジョブのジョブ ID。

  タイプ: 文字列。デフォルト: なし。

  注意: グラフデータが変更されたときにモデルを更新するために、増分データ処理にこれを使用します (ただし、データが削除された場合を除く)。

- **inputDataS3Location** — (必須) データ処理ジョブの実行に必要なデータを SageMaker にダウンロードする Amazon S3 ロケーションの URI。

  タイプ: 文字列。

- **processedDataS3Location** — (必須) SageMaker にデータ処理ジョブの結果を保存する Amazon S3 ロケーションの URI。

  タイプ: 文字列。

- **sagemakerIamRoleArn** — (オプション) SageMaker 実行のための IAM ロールの ARN。

  タイプ: 文字列。注意: これは DB クラスターパラメータグループに一覧表示されている必要があります。そうしないと、エラーが発生します。

- **neptuneIamRoleArn** - (オプション) SageMaker がユーザーに代わってタスクを実行するために引き受けることができる IAM ロールの Amazon リソースネーム (ARN)。

  タイプ: 文字列。注意: これは DB クラスターパラメータグループに一覧表示されている必要があります。そうしないと、エラーが発生します。

- **processingInstanceType** — (オプション) データ処理中に使用される ML インスタンスのタイプ。処理されたデータセットを保持できる十分なメモリ容量が必要です。

  タイプ: 文字列。デフォルト値: ディスク上にエクスポートされたグラフデータのサイズの 10 倍のメモリを持つ最も小さい `ml.r5` タイプ。

  注意: Neptune ML はインスタンスタイプを自動的に選択できます。「データ処理用のインスタンスの選択」を参照してください。

- **processingInstanceVolumeSizeInGB** — (オプション) 処理インスタンスのディスクボリュームサイズ。入力データと処理されたデータの両方がディスクに保存されるため、ボリュームサイズは両方のデータセットを保持するのに十分な大きさでなければなりません。

  タイプ: 整数。デフォルト: `0`。

注意: 指定しない場合、または 0 の場合、Neptune ML はデータサイズに基づいてボリュームサイズを自動的に選択します。

- **processingTimeOutInSeconds** — (オプション) データ処理ジョブの秒単位で指定されたタイムアウト。

  タイプ: 整数。デフォルト: 86,400 (日単位)

- **modelType** — (オプション) Neptune ML が現在サポートしている 2 つのモデルタイプ、異種グラフモデル (heterogeneous)、ナレッジグラフ (kge) のうちの 1 つ。

  タイプ: 文字列。デフォルト: なし。

  注意: 指定しない場合、Neptune ML はデータに基づいてモデルタイプを自動的に選択します。

- **configFileName** — (オプション) トレーニング用にエクスポートされたグラフデータをロードする方法を説明するデータ仕様ファイル。ファイルは Neptune エクスポートツールキットによって自動的に生成されます。

  タイプ: 文字列。デフォルト: training-data-configuration.json。

- **subnets** — (オプション) Neptune VPC 内のサブネットの ID。

  タイプ: 文字列のリスト。デフォルト: なし。

- **securityGroupIds** — (オプション) VPC セキュリティグループ ID。

  タイプ: 文字列のリスト。デフォルト: なし。

- **volumeEncryptionKMSKey** — (オプション) 処理ジョブを実行する ML コンピューティングインスタンスにアタッチされたストレージボリュームのデータを暗号化するために SageMaker が使用する AWS Key Management Service (AWS KMS) キー。

  タイプ: 文字列。デフォルト: なし。

- **enableInterContainerTrafficEncryption** — (オプション) トレーニングジョブまたはハイパーパラメータチューニングジョブでのコンテナ間トラフィック暗号化を有効または無効にします。

  タイプ: ブール値。デフォルト: true。

> **Note**
>
> `enableInterContainerTrafficEncryption` パラメータは、エンジンリリース 1.2.0.2.R3 でのみ使用できます。

- **`s3OutputEncryptionKMSKey`** — (オプション) SageMaker がトレーニングジョブの出力を暗号化するために使用する AWS Key Management Service (AWS KMS) キー。

  タイプ: 文字列。デフォルト: なし。

Neptune ML **`dataprocessing`** コマンドを使用したデータ処理ジョブのステータスの取得

ジョブのステータスのサンプル Neptune ML `dataprocessing` コマンドは、次のようになります。

```
curl -s \
  "https://(your Neptune endpoint)/ml/dataprocessing/(the job ID)" \
  | python -m json.tool
```

`dataprocessing` ジョブステータスのパラメータ

- **`id`** — (必須) データ処理ジョブの一意の識別子。

  タイプ: 文字列。

- **`neptuneIamRoleArn`** — (オプション) SageMaker および Amazon S3 リソースへの Neptune アクセスを提供する IAM ロールの ARN。

  タイプ: 文字列。注意: これは DB クラスターパラメータグループに一覧表示されている必要があります。そうしないと、エラーが発生します。

Neptune ML **`dataprocessing`** コマンドを使用したデータ処理ジョブの停止

ジョブのを停止するサンプル Neptune ML `dataprocessing` コマンドは、次のようになります。

```
curl -s \
  -X DELETE "https://(your Neptune endpoint)/ml/dataprocessing/(the job ID)"
```

または、このようになります。

```
curl -s \
  -X DELETE "https://(your Neptune endpoint)/ml/dataprocessing/(the job ID)?clean=true"
```

dataprocessing 停止ジョブのパラメータ

- **id** — (必須) データ処理ジョブの一意の識別子。

  タイプ: 文字列。

- **neptuneIamRoleArn** — (オプション) SageMaker および Amazon S3 リソースへの Neptune アクセスを提供する IAM ロールの ARN。

  タイプ: 文字列。注意: これは DB クラスターパラメータグループに一覧表示されている必要があります。そうしないと、エラーが発生します。

- **clean** — (オプション) このフラグは、ジョブが停止したときにすべての Amazon S3 アーティファクトを削除する必要があることを指定します。

  タイプ: ブール値。デフォルト: FALSE。

Neptune ML **dataprocessing** コマンドを使用したアクティブなデータ処理ジョブの一覧表示

アクティブジョブの一覧表示のサンプル Neptune ML `dataprocessing` コマンドは、次のようになります。

```
curl -s "https://(your Neptune endpoint)/ml/dataprocessing"
```

または、このようになります。

```
curl -s "https://(your Neptune endpoint)/ml/dataprocessing?maxItems=3"
```

dataprocessing ジョブの一覧表示のパラメータ

- **maxItems** — (オプション) 返される項目の最大数。

  タイプ: 整数。デフォルト: `10`。最大許容値: `1024`。

- **neptuneIamRoleArn** — (オプション) SageMaker および Amazon S3 リソースへの Neptune アクセスを提供する IAM ロールの ARN。

タイプ: 文字列。注意: これは DB クラスターパラメータグループに一覧表示されている必要があります。そうしないと、エラーが発生します。

`modeltraining` コマンドを使用したモデルトレーニング

Neptune ML `modeltraining` コマンドを使用すると、モデルトレーニングジョブを作成したり、そのステータスを確認したり、停止したり、アクティブなモデルトレーニングジョブをすべて一覧表示したりできます。

Neptune ML **modeltraining**コマンドを使用したモデルトレーニングジョブの作成

まったく新しいジョブを作成するための Neptune ML `modeltraining` コマンドは次のようになります。

```
curl \
  -X POST https://(your Neptune endpoint)/ml/modeltraining
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique model-training job ID)",
        "dataProcessingJobId" : "(the data-processing job-id of a completed job)",
        "trainModelS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-graph-
autotrainer"
      }'
```

インクリメンタルモデルトレーニングの更新ジョブを作成するための Neptune ML `modeltraining` コマンドは、次のようになります。

```
curl \
  -X POST https://(your Neptune endpoint)/ml/modeltraining
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique model-training job ID)",
        "dataProcessingJobId" : "(the data-processing job-id of a completed job)",
        "trainModelS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-graph-
autotrainer"
        "previousModelTrainingJobId" : "(the job ID of a completed model-training job
 to update)",
      }'
```

ユーザー指定のカスタムモデル実装で新しいジョブを作成するための Neptune ML `modeltraining` コマンドは次のようになります。

```
curl \
  -X POST https://(your Neptune endpoint)/ml/modeltraining
```

```
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique model-training job ID)",
        "dataProcessingJobId" : "(the data-processing job-id of a completed job)",
        "trainModelS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-graph-
autotrainer"
        "modelName": "custom",
        "customModelTrainingParameters" : {
          "sourceS3DirectoryPath": "s3://(your Amazon S3 bucket)/(path to your Python
 module)",
          "trainingEntryPointScript": "(your training script entry-point name in the
 Python module)",
          "transformEntryPointScript": "(your transform script entry-point name in the
 Python module)"
        }
      }'
```

modeltraining ジョブ作成のパラメータ

- **id** — (オプション) 新しいジョブの一意の識別子。

  タイプ: 文字列。デフォルト値: 自動生成された UUID。
- **dataProcessingJobId** — (必須) トレーニングで使用するデータを作成した、完了したデータ処理ジョブのジョブ ID。

  タイプ: 文字列。
- **trainModelS3Location** — (必須) モデルアーティファクトが保存される Amazon S3 内の場所。

  タイプ: 文字列。
- **previousModelTrainingJobId** — (オプション) 更新されたデータに基づいて段階的に更新する、完了したモデルトレーニングジョブのジョブ ID。

  タイプ: 文字列。デフォルト: なし。
- **sagemakerIamRoleArn** — (オプション) SageMaker 実行のための IAM ロールの ARN。

  タイプ: 文字列。注意: これは DB クラスターパラメータグループに一覧表示されている必要があります。そうしないと、エラーが発生します。
- **neptuneIamRoleArn** — (オプション) SageMaker および Amazon S3 リソースへの Neptune アクセスを提供する IAM ロールの ARN。

タイプ: 文字列。注意: これは DB クラスターパラメータグループに一覧表示されている必要があります。そうしないと、エラーが発生します。

- **modelName** — (オプション) トレーニングのモデルタイプ。デフォルトでは、modelType に基づき ML モデルは自動的にデータ処理で使用されますが、ここで別のモデルタイプを指定できます。

  タイプ: 文字列。デフォルト値: rgcn の場合異種グラフと kge の場合ナレッジグラフ。有効な値: 異種グラフの場合: rgcn。kge グラフの場合: transe、distmult または rotate。カスタムモデル実装の場合: custom。

- **baseProcessingInstanceType** — (オプション) ML モデルのトレーニングの準備と管理に使用される ML インスタンスのタイプ。

  タイプ: 文字列。注意: これは、トレーニングデータとモデルの処理に必要なメモリに基づいて選択された CPU インスタンスです。「モデルトレーニングとモデル変換のインスタンスの選択」を参照してください。

- **trainingInstanceType** — (オプション) モデルトレーニングに使用される ML インスタンスのタイプ。すべてのNeptune ML モデルは、CPU、GPU、MultiGPU トレーニングをサポートしています。

  タイプ: 文字列。デフォルト: ml.p3.2xlarge。

  注意: トレーニングに適したインスタンスタイプは、タスクタイプ、グラフサイズ、および予算によって異なります。「モデルトレーニングとモデル変換のインスタンスの選択」を参照してください。

- **trainingInstanceVolumeSizeInGB** — (オプション) トレーニングインスタンスのディスクボリュームサイズ。入力データと出力モデルの両方がディスクに保存されるため、ボリュームサイズは両方のデータセットを保持するのに十分な大きさでなければなりません。

  タイプ: 整数。デフォルト: 0。

  注意: 指定しない場合、または 0 の場合、Neptune ML はデータ処理ステップで生成された推奨に基づいてディスクボリュームサイズを選択します。「モデルトレーニングとモデル変換のインスタンスの選択」を参照してください。

- **trainingTimeOutInSeconds** — (オプション) トレーニングジョブの秒単位で指定したタイムアウト。

  タイプ: 整数。デフォルト: 86,400 (日単位)

- **maxHPONumberOfTrainingJobs** — ハイパーパラメータ調整ジョブで開始するトレーニングジョブの最大総数。

  タイプ: 整数。デフォルト: 2。

  注意: Neptune ML は、機械学習モデルのハイパーパラメーターを自動的に調整します。うまく機能するモデルを得るには、少なくとも 10 個のジョブを使用します (つまり、maxHPONumberOfTrainingJobs を 10 と設定)。一般的に、チューニングの実行が多いほど、結果は良くなります。

- **maxHPOParallelTrainingJobs** — ハイパーパラメータ調整ジョブで開始する並列トレーニングジョブの最大総数。

  タイプ: 整数。デフォルト: 2。

  注意: 実行できる並列ジョブの数は、トレーニングインスタンスで使用可能なリソースによって制限されます。

- **subnets** — (オプション) Neptune VPC 内のサブネットの ID。

  タイプ: 文字列のリスト。デフォルト: なし。

- **securityGroupIds** — (オプション) VPC セキュリティグループ ID。

  タイプ: 文字列のリスト。デフォルト: なし。

- **volumeEncryptionKMSKey** — (オプション) トレーニングジョブを実行する ML コンピューティングインスタンスにアタッチされたストレージボリュームのデータを暗号化するために SageMaker が使用する AWS Key Management Service (AWS KMS) キー。

  タイプ: 文字列。デフォルト: なし。

- **s3OutputEncryptionKMSKey** — (オプション) SageMaker が処理ジョブの出力を暗号化するために使用する AWS Key Management Service (AWS KMS) キー。

  タイプ: 文字列。デフォルト: なし。

- **enableInterContainerTrafficEncryption** — (オプション) トレーニングジョブまたはハイパーパラメータチューニングジョブでのコンテナ間トラフィック暗号化を有効または無効にします。

  タイプ : ブール値 デフォルト: true。

> **Note**
>
> `enableInterContainerTrafficEncryption` パラメータは、エンジンリリース 1.2.0.2.R3 でのみ使用できます。

- **`enableManagedSpotTraining`** — (オプション) Amazon Elastic Compute Cloud スポットインスタンスを使用して、機械学習モデルのトレーニングのコストを最適化します。詳細については、Amazon SageMaker でのマネージドスポットトレーニングを参照してください。

  タイプ: ブール値。デフォルト: False。

- **`customModelTrainingParameters`** — (オプション) カスタムモデルトレーニングの設定。これは、次の構造を持つ JSON オブジェクトです。

  - **`sourceS3DirectoryPath`** — (必須) モデルを実装する Python モジュールがある Amazon S3 の場所へのパス。これは、少なくともトレーニングスクリプト、変換スクリプト、および `model-hpo-configuration.json` ファイルを含む有効な既存の Amazon S3 の場所を指定しなければなりません。

  - **`trainingEntryPointScript`** — (オプション) モデルトレーニングを実行し、固定ハイパーパラメーターを含むコマンドライン引数としてハイパーパラメーターを取るスクリプトのモジュール内のエントリポイントの名前。

    デフォルト: `training.py`。

  - **`transformEntryPointScript`** — (オプション) モデルのデプロイに必要なモデルアーティファクトを計算するために、ハイパーパラメーター検索の最適なモデルが特定された後に実行されるスクリプトのモジュール内のエントリポイントの名前。コマンドライン引数なしで実行できるはずです。

    デフォルト: `transform.py`。

- **`maxWaitTime`** — (オプション) スポットインスタンスを使用してモデルトレーニングを実行する場合の最大待機時間 (秒単位)。`trainingTimeOutInSeconds` より大きくなければなりません。

  タイプ: 整数。

Neptune ML **`modeltraining`** コマンドを使用したデータ処理ジョブのステータスの取得

ジョブのステータスのサンプル Neptune ML `modeltraining` コマンドは、次のようになります。

```
curl -s \
  "https://(your Neptune endpoint)/ml/modeltraining/(the job ID)" \
  | python -m json.tool
```

modeltraining ジョブステータスのパラメータ

- **id** — (必須) モデルトレーニングジョブの一意の識別子。

  タイプ: 文字列。
- **neptuneIamRoleArn** — (オプション) SageMaker および Amazon S3 リソースへの Neptune アクセスを提供する IAM ロールの ARN。

  タイプ: 文字列。注意: これは DB クラスターパラメータグループに一覧表示されている必要があります。そうしないと、エラーが発生します。

Neptune ML **modeltraining** コマンドを使用したモデルトレーニングジョブの停止

ジョブのを停止するサンプル Neptune ML `modeltraining` コマンドは、次のようになります。

```
curl -s \
  -X DELETE "https://(your Neptune endpoint)/ml/modeltraining/(the job ID)"
```

または、このようになります。

```
curl -s \
  -X DELETE "https://(your Neptune endpoint)/ml/modeltraining/(the job ID)?clean=true"
```

modeltraining 停止ジョブのパラメータ

- **id** — (必須) モデルトレーニングジョブの一意の識別子。

  タイプ: 文字列。
- **neptuneIamRoleArn** — (オプション) SageMaker および Amazon S3 リソースへの Neptune アクセスを提供する IAM ロールの ARN。

  タイプ: 文字列。注意: これは DB クラスターパラメータグループに一覧表示されている必要があります。そうしないと、エラーが発生します。
- **clean** — (オプション) このフラグは、ジョブが停止したときにすべての Amazon S3 アーティファクトを削除する必要があることを指定します。

タイプ: ブール値。デフォルト: FALSE。

Neptune ML **modeltraining** コマンドを使用したアクティブなモデルトレーニングジョブの一覧表示

アクティブジョブの一覧表示のサンプル Neptune ML `modeltraining` コマンドは、次のようになります。

```
curl -s "https://(your Neptune endpoint)/ml/modeltraining" | python -m json.tool
```

または、このようになります。

```
curl -s "https://(your Neptune endpoint)/ml/modeltraining?maxItems=3" | python -m
 json.tool
```

modeltraining ジョブの一覧表示のパラメータ

- **maxItems** — (オプション) 返される項目の最大数。

  タイプ: 整数。デフォルト: 10。最大許容値: 1024。
- **neptuneIamRoleArn** — (オプション) SageMaker および Amazon S3 リソースへの Neptune アクセスを提供する IAM ロールの ARN。

  タイプ: 文字列。注意: これは DB クラスターパラメータグループに一覧表示されている必要があります。そうしないと、エラーが発生します。

モデル変換を使用して **modeltransform** コマンド

Neptune ML `modeltransform` コマンドを使用すると、モデル変換ジョブを作成したり、そのステータスを確認したり、停止したり、アクティブなモデルトレーニングジョブをすべて一覧表示したりできます。

Neptune ML **modeltransform** コマンドを使用したモデル変換ジョブの作成

インクリメンタル変換ジョブを作成するための、モデルの再トレーニングなしでの Neptune ML `modeltransform` コマンドは、次のようになります。

```
curl \
  -X POST https://(your Neptune endpoint)/ml/modeltransform
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique model-transform job ID)",
        "dataProcessingJobId" : "(the job-id of a completed data-processing job)",
        "mlModelTrainingJobId" : "(the job-id of a completed model-training job)",
        "modelTransformOutputS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-
transform"
      }'
```

完了した SageMaker トレーニングジョブからジョブを作成するための Neptune ML `modeltransform` コマンドは次のようになります。

```
curl \
  -X POST https://(your Neptune endpoint)/ml/modeltransform
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique model-transform job ID)",
        "trainingJobName" : "(name of a completed SageMaker training job)",
        "modelTransformOutputS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-
transform",
        "baseProcessingInstanceType" : ""
      }'
```

カスタムモデル実装を使用してジョブを作成するための Neptune ML `modeltransform` コマンドは次のようになります。

```
curl \
  -X POST https://(your Neptune endpoint)/ml/modeltransform
```

```
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique model-training job ID)",
        "trainingJobName" : "(name of a completed SageMaker training job)",
        "modelTransformOutputS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-
transform/"
        "customModelTransformParameters" : {
          "sourceS3DirectoryPath": "s3://(your Amazon S3 bucket)/(path to your Python
 module)",
          "transformEntryPointScript": "(your transform script entry-point name in the
 Python module)"
        }
      }'
```

modeltransform ジョブ作成のパラメータ

- **id** — (オプション) 新しいジョブの一意の識別子。

  タイプ: 文字列。デフォルト値: 自動生成された UUID。
- **dataProcessingJobId** — 完了したデータ処理ジョブのジョブ ID。

  タイプ: 文字列。

  注意: dataProcessingJobId および mlModelTrainingJobId の両方を、または trainingJobName 含める必要があります。
- **mlModelTrainingJobId** — 完了したモデルトレーニングジョブのジョブ ID。

  タイプ: 文字列。

  注意: dataProcessingJobId および mlModelTrainingJobId の両方を、または trainingJobName 含める必要があります。
- **trainingJobName** — 完了した SageMaker トレーニングジョブの名前。

  タイプ: 文字列。

  注意: dataProcessingJobId および mlModelTrainingJobId パラメータの両方を、または trainingJobNameパラメータを含める必要があります。
- **sagemakerIamRoleArn** — (オプション) SageMaker 実行のための IAM ロールの ARN。

  タイプ: 文字列。注意: これは DB クラスターパラメータグループに一覧表示されている必要があります。そうしないと、エラーが発生します。

- **neptuneIamRoleArn** — (オプション) SageMaker および Amazon S3 リソースへの Neptune アクセスを提供する IAM ロールの ARN。

  タイプ: 文字列。注意: これは DB クラスターパラメータグループに一覧表示されている必要があります。そうしないと、エラーが発生します。

- **customModelTransformParameters** — (オプション) カスタムモデルを使用したモデル変換の構成情報。`customModelTransformParameters` オブジェクトには次のフィールドが含まれています。これらのフィールドには、トレーニングジョブの保存されたモデルパラメーターと互換性のある値が必要です。

  - **sourceS3DirectoryPath** — (必須) モデルを実装する Python モジュールがある Amazon S3 の場所へのパス。これは、少なくともトレーニングスクリプト、変換スクリプト、および `model-hpo-configuration.json` ファイルを含む有効な既存の Amazon S3 の場所を指定しなければなりません。

  - **transformEntryPointScript** — (オプション) モデルのデプロイに必要なモデルアーティファクトを計算するために、ハイパーパラメーター検索の最適なモデルが特定された後に実行されるスクリプトのモジュール内のエントリポイントの名前。コマンドライン引数なしで実行できるはずです。

    デフォルト: `transform.py`。

- **baseProcessingInstanceType** — (オプション) ML モデルのトレーニングの準備と管理に使用される ML インスタンスのタイプ。

  タイプ: 文字列。注意: これは、変換データとモデルの処理に必要なメモリに基づいて選択された CPU インスタンスです。「モデルトレーニングとモデル変換のインスタンスの選択」を参照してください。

- **baseProcessingInstanceVolumeSizeInGB** — (オプション) トレーニングインスタンスのディスクボリュームサイズ。入力データと出力モデルの両方がディスクに保存されるため、ボリュームサイズは両方のデータセットを保持するのに十分な大きさでなければなりません。

  タイプ: 整数。デフォルト: `0`。

  注意: 指定しない場合、または 0 の場合、Neptune ML はデータ処理ステップで生成された推奨に基づいてディスクボリュームサイズを選択します。「モデルトレーニングとモデル変換のインスタンスの選択」を参照してください。

- **subnets** — (オプション) Neptune VPC 内のサブネットの ID。

  タイプ: 文字列のリスト。デフォルト: なし。

- **securityGroupIds** — (オプション) VPC セキュリティグループ ID。

  タイプ: 文字列のリスト。デフォルト: なし。

- **volumeEncryptionKMSKey** — (オプション) 変換ジョブを実行する ML コンピューティングインスタンスにアタッチされたストレージボリュームのデータを暗号化するために SageMaker が使用する AWS Key Management Service (AWS KMS) キー。

  タイプ: 文字列。デフォルト: なし。

- **enableInterContainerTrafficEncryption** — (オプション) トレーニングジョブまたはハイパーパラメータチューニングジョブでのコンテナ間トラフィック暗号化を有効または無効にします。

  タイプ : ブール値 デフォルト: true。

  > **Note**
  >
  > enableInterContainerTrafficEncryption パラメータは、エンジンリリース 1.2.0.2.R3 でのみ使用できます。

- **s3OutputEncryptionKMSKey** — (オプション) SageMaker が処理ジョブの出力を暗号化するために使用する AWS Key Management Service (AWS KMS) キー。

  タイプ: 文字列。デフォルト: なし。

Neptune ML **modeltransform** コマンドを使用したモデル変換ジョブのステータスの取得

ジョブのステータスのサンプル Neptune ML modeltransform コマンドは、次のようになります。

```
curl -s \
  "https://(your Neptune endpoint)/ml/modeltransform/(the job ID)" \
  | python -m json.tool
```

modeltransform ジョブステータスのパラメータ

- **id** — (必須) モデル変換ジョブの一意の識別子。

  タイプ: 文字列。

- **neptuneIamRoleArn** — (オプション) SageMaker および Amazon S3 リソースへの Neptune アクセスを提供する IAM ロールの ARN。

  タイプ: 文字列。注意: これは DB クラスターパラメータグループに一覧表示されている必要があります。そうしないと、エラーが発生します。

Neptune ML **modeltransform** コマンドを使用したモデル変換ジョブの停止

ジョブのを停止するサンプル Neptune ML `modeltransform` コマンドは、次のようになります。

```
curl -s \
  -X DELETE "https://(your Neptune endpoint)/ml/modeltransform/(the job ID)"
```

または、このようになります。

```
curl -s \
  -X DELETE "https://(your Neptune endpoint)/ml/modeltransform/(the job ID)?clean=true"
```

modeltransform 停止ジョブのパラメータ

- **id** — (必須) モデル変換ジョブの一意の識別子。

  タイプ: 文字列。
- **neptuneIamRoleArn** — (オプション) SageMaker および Amazon S3 リソースへの Neptune アクセスを提供する IAM ロールの ARN。

  タイプ: 文字列。注意: これは DB クラスターパラメータグループに一覧表示されている必要があります。そうしないと、エラーが発生します。
- **clean** — (オプション) このフラグは、ジョブが停止したときにすべての Amazon S3 アーティファクトを削除する必要があることを指定します。

  タイプ: ブール値。デフォルト: `FALSE`。

Neptune ML **modeltransform** コマンドを使用したアクティブなモデル変換ジョブの一覧表示

アクティブジョブの一覧表示のサンプル Neptune ML `modeltransform` コマンドは、次のようになります。

```
curl -s "https://(your Neptune endpoint)/ml/modeltransform" | python -m json.tool
```

または、このようになります。

```
curl -s "https://(your Neptune endpoint)/ml/modeltransform?maxItems=3" | python -m
 json.tool
```

`modeltransform` ジョブの一覧表示のパラメータ

- **`maxItems`** — (オプション) 返される項目の最大数。

  タイプ: 整数。デフォルト: `10`。最大許容値: `1024`。
- **`neptuneIamRoleArn`** — (オプション) SageMaker および Amazon S3 リソースへの Neptune アクセスを提供する IAM ロールの ARN。

  タイプ: 文字列。注意: これは DB クラスターパラメータグループに一覧表示されている必要があります。そうしないと、エラーが発生します。

endpoints コマンドを使用して推論エンドポイントを管理する

Neptune ML endpoints コマンドを使用して、推論エンドポイントの作成、そのステータスの確認、削除、または既存の推論エンドポイントのリストを表示します。

Neptune ML endpoints コマンドを使用した推論エンドポイントの作成

トレーニングジョブによって作成されたモデルから推論エンドポイントを作成するための Neptune ML endpoints コマンドは、次のようになります。

```
curl \
  -X POST https://(your Neptune endpoint)/ml/endpoints
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique ID for the new endpoint)",
        "mlModelTrainingJobId": "(the model-training job-id of a completed job)"
      }'
```

トレーニングジョブによって作成されたモデルから既存の推論エンドポイントを更新するための Neptune ML endpoints コマンドは、次のようになります。

```
curl \
  -X POST https://(your Neptune endpoint)/ml/endpoints
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique ID for the new endpoint)",
        "update" : "true",
        "mlModelTrainingJobId": "(the model-training job-id of a completed job)"
      }'
```

モデル変換ジョブによって作成されたモデルから推論エンドポイントを作成するための Neptune ML endpoints コマンドは、次のようになります。

```
curl \
  -X POST https://(your Neptune endpoint)/ml/endpoints
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique ID for the new endpoint)",
        "mlModelTransformJobId": "(the model-training job-id of a completed job)"
      }'
```

モデル変換ジョブによって作成されたモデルから既存の推論エンドポイントを更新するための Neptune ML endpoints コマンドは、次のようになります。

```
curl \
  -X POST https://(your Neptune endpoint)/ml/endpoints
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique ID for the new endpoint)",
        "update" : "true",
        "mlModelTransformJobId": "(the model-training job-id of a completed job)"
      }'
```

endpoints 推論エンドポイントの作成のパラメータ

- **id** — (オプション) 新しい推論エンドポイントの一意の識別子。

  タイプ: 文字列。デフォルト値: 自動生成されたタイムスタンプ付きの名前。
- **mlModelTrainingJobId** — 推論エンドポイントが指すモデルを作成した、完了したモデルトレーニングジョブのジョブ ID。

  タイプ: 文字列。

  注意: 次のいずれかを支給する必要があります。mlModelTrainingJobId または mlModelTransformJobId。
- **mlModelTransformJobId** — 完了したモデル変換ジョブのジョブ ID。

  タイプ: 文字列。

  注意: 次のいずれかを支給する必要があります。mlModelTrainingJobId または mlModelTransformJobId。
- **update** — (オプション) 存在する場合、このパラメータはこれが更新リクエストであることを示します。

  タイプ: ブール値。デフォルト: false

  注意: 次のいずれかを支給する必要があります。mlModelTrainingJobId または mlModelTransformJobId。
- **neptuneIamRoleArn** — (オプション) SageMaker および Amazon S3 リソースへの Neptune アクセスを提供する IAM ロールの ARN。

タイプ: 文字列。注意: これは DB クラスターパラメータグループに一覧表示されている必要があります。そうしないと、エラーがスローされます。

- **modelName** — (オプション) トレーニングのモデルタイプ。デフォルトでは、modelType に基づき ML モデルは自動的にデータ処理で使用されますが、ここで別のモデルタイプを指定できます。

  タイプ: 文字列。デフォルト値: rgcn の場合異種グラフと kge の場合ナレッジグラフ。有効な値: 異種グラフの場合: rgcn。ナレッジグラフの場合: kge、transe、distmult または rotate。

- **instanceType** — (オプション) オンラインサービスに使用される ML インスタンスのタイプ。

  タイプ: 文字列。デフォルト: ml.m5.xlarge。

  注意: 推論エンドポイントの ML インスタンスの選択は、タスクタイプ、グラフサイズ、および予算によって異なります。「推論エンドポイントのインスタンスを選択する」を参照してください。

- **instanceCount** — (オプション) 予測のためにエンドポイントにデプロイする Amazon EC2 インスタンスの最小数。

  タイプ: 整数。デフォルト: 1。

- **volumeEncryptionKMSKey** — (オプション) エンドポイントを実行する ML コンピューティングインスタンスにアタッチされたストレージボリュームのデータを暗号化するために SageMaker が使用する AWS Key Management Service (AWS KMS) キー。

  タイプ: 文字列。デフォルト: なし。

Neptune ML **endpoints** コマンドを使用した推論エンドポイントのステータスの取得

インスタンスエンドポイントのステータスのサンプル Neptune ML endpoints コマンドは、次のようになります。

```
curl -s \
  "https://(your Neptune endpoint)/ml/endpoints/(the inference endpoint ID)" \
  | python -m json.tool
```

endpoints インスタンスエンドポイントステータスのパラメータ

- **id** — (必須) 推論エンドポイントの一意の識別子。

  タイプ: 文字列。

- **neptuneIamRoleArn** — (オプション) SageMaker および Amazon S3 リソースへの Neptune アクセスを提供する IAM ロールの ARN。

  タイプ: 文字列。注意: これは DB クラスターパラメータグループに一覧表示されている必要があります。そうしないと、エラーがスローされます。

Neptune ML **endpoints** コマンドを使用したインスタンスエンドポイントの削除

インスタンスエンドポイントを削除するサンプル Neptune ML endpoints コマンドは、次のようになります。

```
curl -s \
  -X DELETE "https://(your Neptune endpoint)/ml/endpoints/(the inference endpoint ID)"
```

または、このようになります。

```
curl -s \
  -X DELETE "https://(your Neptune endpoint)/ml/endpoints/(the inference endpoint ID)?
clean=true"
```

endpoints 推論エンドポイントの削除のパラメータ

- **id** — (必須) 推論エンドポイントの一意の識別子。

  タイプ: 文字列。
- **neptuneIamRoleArn** — (オプション) SageMaker および Amazon S3 リソースへの Neptune アクセスを提供する IAM ロールの ARN。

  タイプ: 文字列。注意: これは DB クラスターパラメータグループに一覧表示されている必要があります。そうしないと、エラーがスローされます。
- **clean** — (オプション) このエンドポイントに関連するすべてのアーティファクトも削除する必要があることを示します。

  タイプ: ブール値。デフォルト: FALSE。

Neptune ML **endpoints** コマンドを使用した推論エンドポイントの一覧表示

推論エンドポイントを一覧表示するための Neptune ML endpoints コマンドは次のようになります。

```
curl -s "https://(your Neptune endpoint)/ml/endpoints" \
  | python -m json.tool
```

または、このようになります。

```
curl -s "https://(your Neptune endpoint)/ml/endpoints?maxItems=3" \
  | python -m json.tool
```

dataprocessing 推論エンドポイントの一覧表示のパラメータ

- **maxItems** — (オプション) 返される項目の最大数。

  タイプ: 整数。デフォルト: 10。最大許容値: 1024。
- **neptuneIamRoleArn** — (オプション) SageMaker および Amazon S3 リソースへの Neptune アクセスを提供する IAM ロールの ARN。

  タイプ: 文字列。注意: これは DB クラスターパラメータグループに一覧表示されている必要があります。そうしないと、エラーがスローされます。

Neptune ML 管理 API のエラーと例外

すべての Neptune ML 管理 API 例外は 400 HTTP コードを返します。これらの例外を受け取った後、例外を生成したコマンドは再試行しないでください。

- **MissingParameterException** - エラーメッセージ:

  ```
  Required credentials are missing. Please add IAM role to the cluster or pass as a parameter to this request.
  ```

- **InvalidParameterException** - エラーメッセージ (複数) :
  - `Invalid ML instance type.`
  - `Invalid ID provided. ID can be 1-48 alphanumeric characters.`
  - `Invalid ID provided. Must contain only letters, digits, or hyphens.`
  - `Invalid ID provided. Please check whether a resource with the given ID exists.`
  - `Another resource with same ID already exists. Please use a new ID.`
  - `Failed to stop the job because it has already completed or failed.`
- **BadRequestException** - エラーメッセージ (複数) :
  - `Invalid S3 URL or incorrect S3 permissions. Please check your S3 configuration.`
  - `Provided ModelTraining job has not completed.`
  - `Provided SageMaker Training job has not completed.`
  - `Provided MLDataProcessing job is not completed.`
  - `Provided MLModelTraining job doesn't exist.`
  - `Provided ModelTransformJob doesn't exist.`
  - `Unable to find SageMaker resource. Please check your input.`

Neptune ML の制限

- 現在サポートされている推論のタイプは、ノード分類、ノード回帰、エッジ分類、エッジ回帰、リンク予測です (Neptune ML 機能 を参照)。
- Neptune ML がサポートできるグラフの最大サイズは、データ準備中、モデルトレーニング中、および推論中に必要なメモリとストレージの量によって異なります。
  - SageMaker データ処理インスタンスのメモリの最大サイズは 768 GB です。その結果、768 GB を超えるメモリが必要な場合、データ処理ステージは失敗します。
  - SageMaker トレーニングインスタンスのメモリの最大サイズは 732 GB です。その結果、732 GB を超えるメモリが必要な場合、トレーニングステージは失敗します。
- SageMaker エンドポイントの推論ペイロードの最大サイズは 6 MiB です。そのため、サブグラフのペイロードがこのサイズを超えた場合、帰納的推論は失敗します。
- Neptune ML は現在、Neptune と他の依存しているサービス (AWS Lambda、Amazon API Gateway、Amazon SageMaker など) がすべてサポートされているリージョンでのみ利用できます。

  IAM 認証のデフォルトの使用には、他の違いとともにここで説明しているように中国 (北京) と中国 (寧夏) の違いがあります。
- Neptune ML によって起動されたリンク予測推論エンドポイントは、現在、トレーニング時にグラフに存在したノードで可能なリンクのみを予測できます。

  例えば、`User` と `Movie` の頂点と `Rated` のエッジがあるグラフを考えてみましょう。対応する Neptune ML リンク予測推奨モデルを使用すると、グラフに新しいユーザーを追加し、モデルにそのユーザー向けの映画を予測させることができますが、モデルが推奨できるのは、モデルトレーニング時に存在した映画だけです。`User` ノード埋め込みは、ローカルサブグラフと GNN モデルを使用してリアルタイムで計算されるため、ユーザーが映画を評価するにつれて時間とともに変化する可能性がありますが、最終的なレコメンデーションでは、事前に計算された静的な映画の埋め込みと比較されます。
- Neptune ML でサポートされる KGE モデルは、リンク予測タスクに対してのみ機能し、表現はトレーニング中にグラフ内に存在する頂点とエッジタイプに固有です。つまり、推論クエリで参照されるすべての頂点とエッジタイプは、トレーニング中にグラフ内に存在している必要があります。新しいエッジタイプまたは頂点の予測は、モデルを再学習しなければ実行できません。

SageMaker リソース制限

アクティビティと時間経過に伴うリソースの使用状況に応じて、次のようなエラーメッセージが表示されることがあります。リソース上限を越えました (ResourceLimitExceeded)。SageMaker リソースをスケールアップする必要があるため、このページに記載の SageMaker リソースのサービスクォータの引き上げリクエスト の手順に従い AWS サポートから上限を増えすリクエストを送ります。

SageMaker リソース名は、次のように Neptune ML ステージに対応します。

- SageMaker `ProcessingJob` は、Neptune のデータ処理、モデルトレーニング、およびモデル変換ジョブで使用されます。
- SageMaker `HyperParameterTuningJob` は、Neptune のモデルトレーニングジョブによって使用されています。
- SageMaker `TrainingJob` は、Neptune のモデルトレーニングジョブによって使用されています。
- SageMaker `Endpoint` は Neptune 推論エンドポイントで使用されます。

Amazon Neptune リソースのモニタリング

Amazon Neptune では、パフォーマンスと使用状況をモニタリングするためのさまざまな方法をサポートしています。

- インスタンスステータス – Neptune クラスターのグラフデータベースエンジンの状態をチェックし 、インストールされているエンジンのバージョンの確認、インスタンスステータス API を使用したその他のインスタンス関連情報の取得を行います。
- グラフサマリー API — グラフサマリー API を使用すると、グラフのデータサイズと内容を大まかに把握できます。

> **Note**
>
> グラフサマリー API は DFE 統計に依存しているため、統計が有効になっている場合にのみ使用でき、T3 や T4g インスタンスタイプには当てはまりません。

- Amazon CloudWatch – Neptune は にメトリクスを自動的に送信 CloudWatch し、 CloudWatch アラームもサポートします。詳細については、「the section called “の使用 CloudWatch”」を参照してください。
- 監査ログファイル – Neptune コンソールを使用して、データベースログファイルを表示、ダウンロード、調査します。詳細については、「the section called “Neptune による監査ログ ”」を参照してください。
- Amazon CloudWatch Logs へのログの発行 – 監査ログデータを Amazon Logs のロググループに発行するように Neptune DB CloudWatch クラスターを設定できます。 CloudWatch Logs を使用すると、ログデータのリアルタイム分析を実行し、 CloudWatch を使用してアラームを作成し、メトリクスを表示し、 CloudWatch ログを使用してログレコードを耐久性の高いストレージに保存できます。Neptune CloudWatch ログ を参照してください。
- AWS CloudTrail — Neptune は、 を使用した API ログ記録をサポートしています CloudTrail。詳細については、「the section called “を使用した Neptune API コールのログ記録 AWS CloudTrail”」を参照してください。
- イベント通知サブスクリプション — Neptune イベントに登録して、何が起こっているのかを把握しましょう。詳細については、「the section called “イベント通知”」を参照してください。
- タグ付け – タグを使用してメタデータを Neptune リソースに追加し、タグに基づいて使用量を追跡します。詳細については、「the section called “Neptune リソースにタグを付ける”」を参照してください。

トピック

Neptune インスタンスのヘルスステータスをチェックする

Amazon Neptune では、ホストでのグラフデータベースのステータスを確認するメカニズムが用意されています。インスタンスに接続できるかどうかを確認する、優れた方法でもあります。

curl を使用してインスタンスの状態を確認し、DB クラスターのステータスを取得するには

```
curl -G https://your-neptune-endpoint:port/status
```

または、エンジンリリース 1.2.1.0.R6 以降では、代わりに次の CLI コマンドを使用できます。

```
aws neptunedata get-engine-status
```

インスタンスが正常であれば、status コマンドは以下のフィールドでJSON オブジェクトを返します。

- **status** — インスタンスに問題が発生していない場合、"healthy" に設定します。

  インスタンスがクラッシュまたは再起動から回復中で、最新のサーバーのシャットダウンからアクティブなトランザクションが実行されている場合、status は "recovery" に設定されます。
- **startTime** – 現在のサーバープロセスが開始する UTC 時間に設定します。
- **dbEngineVersion** – DB クラスターで実行されている Neptune エンジンのバージョンに設定します。

このバージョンのエンジンがリリースされた後に手動でパッチが適用された場合は、バージョン番号の先頭に "Patch-" が付加されます。

- **role** – インスタンスがリードレプリカの場合は "reader" に設定し、インスタンスがプライマリインスタンスの場合は "writer" に設定します。
- **dfeQueryEngine** - DFE エンジンが完全に有効になっている場合は "enabled" に設定し、useDFE クエリヒントが true (viaQueryHint がデフォルト) に設定されているクエリでのみ DFE エンジンが使用される場合は viaQueryHint に設定します。
- **gremlin** — クラスターで使用できる Gremlin クエリ言語に関する情報が含まれます。具体的には、エンジンが使用している現在の TinkerPop バージョンを指定するversionフィールドが含まれています。
- **sparql** — クラスターで使用できる SPARQL クエリ言語に関する情報が含まれます。具体的には、エンジンが使用している現在の SPARQL バージョンを指定する version フィールド。
- **opencypher** - クラスターで使用できる openCypher クエリ言語に関する情報を含みます。具体的には、エンジンが使用している現在の openCypher バージョンを指定する version フィールドを含みます。
- **labMode** — エンジンが使用している ラボモード 設定を含みます。
- **rollingBackTrxCount** — ロールバックされるトランザクションがある場合、このフィールドはそのようなトランザクションの数に設定されます。コメントがない場合、このフィールドは一切表示されません。
- **rollingBackTrxEarliestStartTime** – ロールバックされる最も早いトランザクションの開始時刻に設定します。ロールバックされるトランザクションがない場合、このフィールドは一切表示されません。
- **features** — DB クラスターで有効になっている機能に関するステータス情報が含まれます。
  - **lookupCache** - ルックアップキャッシュ の現在の状態。このフィールドは、ルックアップキャッシュが存在できる唯一のインスタンスであるため R5d インスタンスタイプに表示されます。このフィールドは、次の形式の JSON オブジェクトです。

    ```
    "lookupCache": {
      "status": "current lookup cache status"
    }
    ```

    R5dインスタンスで。

    - ルックアップキャッシュが有効な場合、ステータスは次のように表示されます。"Available"。

- ルックアップキャッシュが無効な場合、ステータスは次のように表示されます。"Disabled"。
- インスタンスのディスク制限に達した場合、ステータスは次のように表示されます。"Read Only Mode - Storage Limit Reached"。

- **ResultCache** - クエリ結果の使用 の現在の状態。このフィールドは、次の形式の JSON オブジェクトです。

```
"ResultCache": {
  "status": "current results cache status"
}
```

  - 結果キャッシュが有効になっている場合、ステータスは次のように表示されます。"Available"。
  - キャッシュが無効になっている場合、ステータスは次のように表示されます。"Disabled"。
- **IAMAuthentication** – DB クラスターで AWS Identity and Access Management (IAM) 認証が有効になっているかどうかを指定します。
  - IAM 認証が有効になっている場合、ステータスは次のように表示されます。"enabled"。
  - キャッシュが無効になっている場合、ステータスは次のように表示されます。"disabled"。
- **Streams** — DB クラスターで Neptune ストリームが有効になっているかどうかを指定します。
  - ストリームが有効な場合、ステータスは次のように表示されます。"enabled"。
  - ストリームが無効な場合、ステータスは次のように表示されます。"disabled"。
- **AuditLog**— 監査ログが有効な場合は enabled、そうでない場合は disabled です。
- **SlowQueryLogs** — スロークエリロギングが有効になっている場合は info または debug に等しく、そうでない場合は disabled に等しくなります。
- **QueryTimeout** — クエリタイムアウトの値 (ミリ秒単位) 。

- **settings** — インスタンスに適用される設定:
  - **clusterQueryTimeoutInMs** — クラスター全体に設定されるクエリタイムアウトの値 (ミリ秒単位)。
  - **SlowQueryLogsThreshold** — クラスター全体に設定されるクエリタイムアウトの値 (ミリ秒単位)。
- **serverlessConfiguration** — クラスターがサーバーレスで実行されている場合のサーバーレス設定:

- **minCapacity** — DB クラスター内のサーバーレスインスタンスを縮小できる最小サイズ (Neptune キャパシティユニット (NCU) 単位)。
- **maxCapacity** — DB クラスター内のサーバーレスインスタンスを拡大できる最大サイズ (Neptune キャパシティユニット (NCU) 単位)。

インスタンスステータスコマンドからの出力例

以下は、instance status コマンドからの出力例です (この場合は、R5dインスタンス):

```
{
  'status': 'healthy',
  'startTime': 'Thu Aug 24 21:47:12 UTC 2023',
  'dbEngineVersion': '1.2.1.0.R4',
  'role': 'writer',
  'dfeQueryEngine': 'viaQueryHint',
  'gremlin': {'version': 'tinkerpop-3.6.2'},
  'sparql': {'version': 'sparql-1.1'},
  'opencypher': {'version': 'Neptune-9.0.20190305-1.0'},
  'labMode': {
    'ObjectIndex': 'disabled',
    'ReadWriteConflictDetection': 'enabled'
  },
  'features': {
    'SlowQueryLogs': 'disabled',
    'ResultCache': {'status': 'disabled'},
    'IAMAuthentication': 'disabled',
    'Streams': 'disabled',
    'AuditLog': 'disabled'
  },
  'settings': {
    'clusterQueryTimeoutInMs': '120000',
    'SlowQueryLogsThreshold': '5000'
  },
  'serverlessConfiguration': {
    'minCapacity': '1.0',
    'maxCapacity': '128.0'
  }
}
```

インスタンスに問題がある場合、ステータスコマンドは HTTP 500 エラーコードを返します。ホストが到達不可能になった場合、リクエストはタイムアウトします。仮想プライベートクラウド (VPC)

内からインスタンスにアクセスしていること、およびセキュリティグループがそのクラスターあるいはインスタンスへのアクセスを許可していることを確認します。

Amazon を使用した Neptune のモニタリング CloudWatch

Amazon Neptune と Amazon CloudWatch は統合されているため、パフォーマンスメトリクスを収集して分析できます。これらのメトリクスは、 CloudWatch コンソール、 AWS Command Line Interface （ AWS CLI ）、または CloudWatch API を使用してモニタリングできます。

CloudWatch また、 では、メトリクス値が指定したしきい値を超えた場合に通知されるようにアラームを設定できます。違反が発生した場合に是正措置を講じるように CloudWatch イベントを設定することもできます。 CloudWatch および アラームの使用の詳細については、CloudWatch 「 ドキュメント」を参照してください。

トピック

- CloudWatch データの表示 (コンソール)
- CloudWatch データの表示 (AWS CLI)
- CloudWatch データの表示 (API)
- CloudWatch を使用して Neptune で DB インスタンスのパフォーマンスをモニタリングする
- Neptune CloudWatch メトリクス
- Neptune CloudWatch ディメンション

CloudWatch データの表示 (コンソール)

Neptune クラスター CloudWatch のデータを表示するには (コンソール)

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. ナビゲーションペインで [メトリクス]を選択します。
3. すべてのメトリクスペインで、Neptune を選択し、DB ClusterIdentifierを選択します。
4. 上部ペインで下にスクロールして、クラスターのメトリクスの詳細なリストを表示します。Neptune で使用できるメトリクスオプションが [Viewing] (表示中) リストに表示されます。

個々のメトリクスを選択または選択解除するには、結果ペインで、リソースネームとメトリクスの横にあるチェックボックスを選択します。選択した項目のメトリクスを示すグラフがコンソールの下部

に表示されます。 CloudWatch グラフの詳細については、「Amazon CloudWatch ユーザーガイド」の「グラフメトリクス」を参照してください。

CloudWatch データの表示 (AWS CLI)

Neptune クラスター CloudWatch のデータを表示するには (AWS CLI)

1. をインストールします AWS CLI。手順については、AWS Command Line Interface ユーザーガイドを参照してください。
2. AWS CLI を使用して情報を取得します。Neptune に関連する CloudWatch パラメータは、 に記載されていますNeptune CloudWatch メトリクス。

   次の例では、`gremlin-cluster`クラスターの 1 秒あたりの Gremlin リクエスト数の CloudWatch メトリクスを取得します。

```
aws cloudwatch get-metric-statistics \
    --namespace AWS/Neptune  --metric-name GremlinRequestsPerSec \
    --dimensions Name=DBClusterIdentifier,Value=gremlin-cluster \
    --start-time 2018-03-03T00:00:00Z --end-time 2018-03-04T00:00:00Z \
    --period 60 --statistics=Average
```

CloudWatch データの表示 (API)

CloudWatch は、プログラムで情報をリクエストできるように `Query`アクションもサポートしています。詳細については、CloudWatch 「Query API ドキュメント」および「Amazon CloudWatch API リファレンス」を参照してください。

CloudWatch アクションで などの Neptune モニタリングに固有のパラメータが必要な場合は`MetricName`、 にリストされている値を使用しますNeptune CloudWatch メトリクス。

次の例は、以下のパラメータを使用した低レベル CloudWatch リクエストを示しています。

- `Statistics.member.1 = Average`
- `Dimensions.member.1 = DBClusterIdentifier=gremlin-cluster`
- `Namespace  = AWS/Neptune`
- `StartTime  = 2013-11-14T00:00:00Z`
- `EndTime  = 2013-11-16T00:00:00Z`

- Period = 60
- MetricName = GremlinRequestsPerSec

CloudWatch リクエストは次のようになります。これは、リクエストの形式を示しているだけであるため、各自のメトリクスと時間枠に基づいてリクエストを作成する必要があります。

```
https://monitoring.amazonaws.com/
      ?SignatureVersion=2
      &Action=GremlinRequestsPerSec
      &Version=2010-08-01
      &StartTime=2018-03-03T00:00:00
      &EndTime=2018-03-04T00:00:00
      &Period=60
      &Statistics.member.1=Average
      &Dimensions.member.1=DBClusterIdentifier=gremlin-cluster
      &Namespace=AWS/Neptune
      &MetricName=GremlinRequests
      &Timestamp=2018-03-04T17%3A48%3A21.746Z
      &AWSAccessKeyId=AWS Access Key ID;
      &Signature=signature
```

CloudWatch を使用して Neptune で DB インスタンスのパフォーマンスをモニタリングする

Neptune の CloudWatch メトリクスを使用して、DB インスタンスで何が起こっているかをモニタリングし、データベースで観測されたクエリキューの長さを追跡できます。次のメトリックが特に便利です。

- **CPUUtilization** – CPU 使用率を表示します。
- **VolumeWriteIOPs** – クラスターボリュームに対する書き込みディスク I/O 操作の平均回数を表示し、これは 5 分間隔で報告されます。
- **MainRequestQueuePendingRequests** – 入力キューで実行を保留中のリクエストの数を示します。

サーバーで保留中のリクエストの数を確認するには、Gremlin クエリステータスエンドポイントと includeWaiting パラメータを使用します。これにより、待機中のすべてのクエリのステータスが表示されます。

次のインジケーターは、Neptune のプロビジョニングとクエリ戦略を調整して、効率とパフォーマンスを向上させるのに役立ちます。

- 一貫したレイテンシー、高 CPUUtilization、高 VolumeWriteIOPs、低 MainRequestQueuePendingRequests が合わせて見られれば、サーバが I/O の待ち時間をほとんど必要とせずに、持続可能なレートで同時書き込み要求の処理に積極的に関与していることを示しています。
- 一貫したレイテンシー、低 CPUUtilization、低 VolumeWriteIOPsそして MainRequestQueuePendingRequests がないことは、プライマリ DB インスタンスで書き込みリクエストを処理するための超過容量があることを共に示しています。
- 高 CPUUtilization と高 VolumeWriteIOPs、可変レイテンシー MainRequestQueuePendingRequests が合わせて見られれば、特定の間隔でサーバーが処理できる以上の作業を送信していることを示しています。トランザクションオーバーヘッドを少なくして同じ量の作業を行うように、バッチリクエストを作成またはサイズ変更し、プライマリインスタンスをスケーリングして、ライトリクエストを同時に処理できるクエリスレッドの数を増やすことを検討してください。
- 低 CPUUtilization かつ高 VolumeWriteIOPsは、クエリスレッドがストレージレイヤーへの I/O 操作の完了を待機していることを意味します。可変レイテンシーと多少の増加が MainRequestQueuePendingRequests で見られたら、トランザクションオーバーヘッドを少なくして同じ量の作業を行うように、バッチリクエストを作成またはサイズ変更することを検討してください。

Neptune CloudWatch メトリクス

> **Note**
>
> Amazon Neptune は、ゼロ以外の値がある CloudWatch 場合にのみメトリクスを に送信します。
> すべての Neptune メトリクスについて、集計の詳細度は 5 分です。

トピック

Neptune CloudWatch メトリクス

次の表に、Neptune がサポートする CloudWatch メトリクスを示します。

> **Note**
>
> 保守、再起動、クラッシュからの回復など、サーバーを再起動するたびに、すべての累積メトリクスがゼロにリセットされます。

Neptune CloudWatch メトリクス

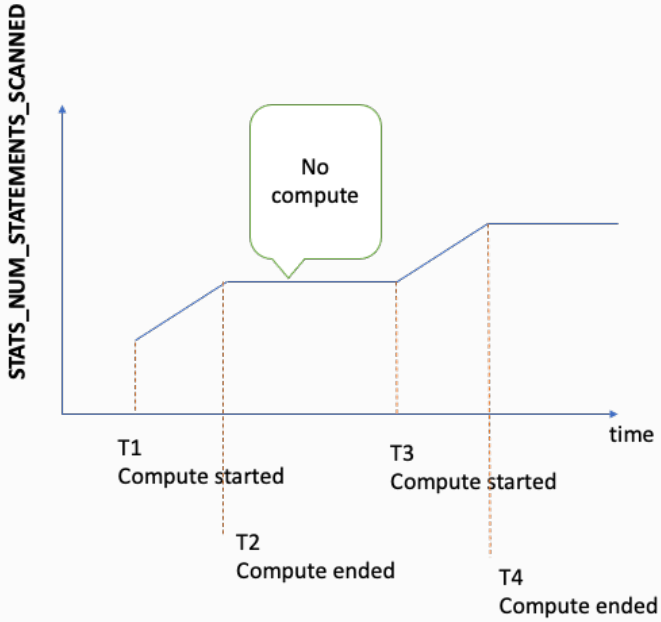
メトリクス	説明
`BackupRetentionPeriodStorageUsed`	Neptune DB クラスターのバックアップ保持ウィンドウからのサポートに使用されるバックアップストレージの総量 (バイト単位)。`TotalBackupStorageBilled` メトリクスによって報告される合計に含まれます。
`BufferCacheHitRatio`	バッファキャッシュから提供されたリクエストの割合 (パーセント)。このメトリクスは、キャッシュミスが大きなレイテンシーを引き起こすため、クエリレイテンシーの診断に役立ちます。キャッシュヒット率が 99.9% を下回っている場合は、より多くのデータをメモリにキャッシュするようにインスタンスタイプをアップグレードすることを検討してください。
`ClusterReplicaLag`	リードレプリカについて、プライマリインスタンスからアップデートをレプリケートする際の遅延時間 (ミリ秒単位)。
`ClusterReplicaLagMaximum`	DB クラスター内のプライマリインスタンスと各 Neptune DB インスタンス間の最大遅延時間 (ミリ秒単位)。

メトリクス	説明
ClusterReplicaLagMinimum	DB クラスター内のプライマリインスタンスと各 Neptune DB インスタンス間の最小遅延時間 (ミリ秒単位)。
CPUUtilization	CPU 使用率。
EngineUptime	インスタンス実行時間 (秒単位)。
FreeableMemory	使用可能な RAM の容量 (バイト単位)。
GlobalDbDataTransferBytes	AWS リージョン Neptune グローバルデータベースのプライマリから AWS リージョン セカンダリに転送された REDO ログデータのバイト数。
GlobalDbReplicatedWriteIO	グローバルデータベースのプライマリ AWS リージョン からセカンダリ AWS リージョンのクラスターボリュームに複製された書き込み I/O 操作の数。 Neptune グローバルデータベース内の各 DB クラスターの課金計算では、そのクラスター内で実行された書き込みを説明するために VolumeWriteIOPS メトリクスが使用されます。プライマリ DB クラスターの場合、課金計算では、GlobalDbReplicatedWriteIO が使用され、セカンダリ DB クラスターへのクロスリージョンレプリケーションが考慮されます。
GlobalDbProgressLag	ユーザートランザクションとシステムトランザクションの両方について、セカンダリクラスターがプライマリクラスターより遅れているミリ秒数。
GremlinRequestsPerSec	Gremlin エンジンに対する 1 秒あたりのリクエスト数。

メトリクス	説明
GremlinWebSocketOpenConnections	Neptune へのオープン WebSocket 接続の数。
LoaderRequestsPerSec	1 秒あたりのローダーリクエストの数。
MainRequestQueuePendingRequests	入力キューで実行を保留中のリクエストの数を示します。Neptune は、最大キュー容量を超えるとリクエストのスロットリングを開始します。
NCUUtilization	Neptune サーバーレス DB インスタンスまたは DB クラスターにのみ適用されます。インスタンスレベルでは、問題のインスタンスが現在使用している Neptune キャパシティユニット (NCU) の数を、クラスターの最大NCUキャパシティ設定で割って計算されたパーセンテージを報告します。スケーリング構成は Neptune 容量ユニット (NCU) で定義され、それぞれ 2 GiB (ギビバイト) のメモリ (RAM) と、関連する仮想プロセッサ容量 (vCPU) とネットワークで構成されています。 クラスターレベルでは、NCUUtilization はクラスター全体で使用されている最大容量の割合を報告します。
NetworkThroughput	Neptune DB クラスター内の各インスタンスが各クライアントで送受信したネットワークスループットの量 (バイト/秒単位)。このスループットには、DB クラスターとクラスターボリューム内のインスタンス間のネットワークトラフィックは、含まれません。

メトリクス	説明
NetworkTransmitThroughput	Neptune DB クラスター内の各インスタンスが各クライアントで送受信したネットワークスループットの量 (バイト/秒単位)。このスループットには、DB クラスターとクラスターボリューム内のインスタンス間のネットワークトラフィックは、含まれません。
NumTxCommitted	1 秒あたりの正常にコミットされたトランザクションの数。
NumTxOpened	1 秒あたりのサーバーで開いたトランザクションの数。
NumTxRolledBack	エラーのためにサーバーでロールバックされた 1 秒あたりのトランザクションの数。読み取り専用クエリの場合、この指標は 1 秒あたりに完了した読み取り専用トランザクションの数と同じです。
OpenCypherRequestsPerSec	openCypher エンジンへの 1 秒あたりのリクエスト数 (HTTPS と Bolt の両方)。
OpenCypherBoltOpenConnections	Neptune へのオープンな Bolt 接続の数。

メトリクス	説明
ServerlessDatabaseCapacity	インスタンスレベルのメトリクスとして、ServerlessDatabaseCapacity は特定の Neptune サーバーレスインスタンスの現在のインスタンス容量を NCU 単位で報告します。スケーリング構成は Neptune 容量ユニット (NCU) で定義され、それぞれ 2 GiB (ギビバイト) のメモリ (RAM) と、関連する仮想プロセッサ容量 (vCPU) とネットワークで構成されています。 クラスターレベルで、ServerlessDatabaseCapacity はクラスター内のすべての DB インスタンスの ServerlessDatabaseCapacity 値の平均を報告します。
SnapshotStorageUsed	バックアップ保持ウィンドウ外で、Neptune DB クラスターのすべてのスナップショットで消費されているバックアップストレージの合計容量 (バイト単位)。TotalBackupStorageBilled メトリクスによって報告される合計に含まれます。
SparqlRequestsPerSec	SPARQL エンジンへの 1 秒あたりのリクエストの数。

メトリクス	説明
StatsNumStatementsScanned	サーバーの起動後にDFE 統計計算用にスキャンされたステートメントの総数。 統計計算がトリガーされるたびに、この数は増加しますが、計算が行われていないときは静的なままです。その結果、時間の経過とともにグラフ化すると、計算がいつ起こったのか、それがなかったのかを知ることができます。  メトリックが増加している期間のグラフの傾きを見ることで、計算の速さを知ることもできます。 このようなメトリクスがない場合は、DB クラスターで統計機能が無効になっているか、実行中のエンジンバージョンに統計機能がないことを意味します。メトリクス値がゼロの場合は、統計計算が行われていないことを意味します。

メトリクス	説明
TotalBackupStorageBilled	特定の Neptune DB クラスターに関して請求対象のバックアップストレージの合計容量 (バイト単位)。BackupRetentionPeriodStorageUsed メトリクスおよび SnapshotStorageUsed メトリクスによって測定されるバックアップストレージが含まれます。
TotalRequestsPerSec	すべてのソースからサーバーへの 1 秒あたりのリクエストの総数。
TotalClientErrorsPerSec	クライアント側の問題が原因でエラーが発生したリクエストの 1 秒あたりの合計数。
TotalServerErrorsPerSec	内部障害が原因でサーバーでエラーが発生したリクエストの 1 秒あたりの合計数。

メトリクス	説明
UndoLogListSize	UNDO ログリスト内の UNDO ログの数。 UNDO ログには、すべてのアクティブなトランザクションがコミット時間より新しい時点で期限切れになるコミットされたトランザクションの記録が含まれます。期限切れのレコードは定期的に消去されます。削除操作のレコードは、他の種類のトランザクションのレコードよりも消去に時間がかかる場合があります。 削除は DB クラスターのライターインスタンスによってのみ行われるため、削除の速度はライターインスタンスタイプによって異なります。UndoLogListSize が高く、DB クラスター内で増加している場合は、ライターインスタンスをアップグレードして、パージ率を上げてください。 また、1.2.0.0 以前のバージョンから 1.2.0.0 以降のエンジンバージョンにアップグレードする場合は、まず、UndoLogListSize 値が 0 に近いことを確認してください。エンジンバージョン 1.2.0.0 以降では、UNDO ログの形式が異なるため、アップグレードは以前の UNDO ログが完全に消去された後にのみ開始できます。詳細については、「1.2.0.0 以降へのアップグレード」を参照してください。

メトリクス	説明
VolumeBytesUsed	Neptune DB クラスターに割り当てられたストレージの合計量 (バイト単位)。これは、請求されるストレージの量です。DB クラスターが存在している任意の時点で DB クラスターに割り当てられるストレージの最大量であり、現在使用している量ではありません (「Neptune ストレージ請求」を参照)。
VolumeReadIOPs	5 分間隔で報告された、クラスターボリュームからの課金読み取り I/O オペレーションの合計数。課金読み取りオペレーションはクラスターボリュームレベルで計算され、Neptune DB クラスター内のすべてのインスタンスから集計された後、5 分おきに報告されます。
VolumeWriteIOPs	5 分間隔で報告された、クラスターボリュームへの書き込みディスク I/O オペレーションの合計数。

CloudWatch Neptune で非推奨になったメトリクス

これらの Neptune メトリクスの使用は廃止されました。これらのメトリクスはまだサポートされていますが、より優れた新しいメトリクスが利用可能になることで、将来的に削除される可能性があります。

メトリクス	説明
GremlinHttp1xx	Gremlin エンドポイントの HTTP 1xx レスポンスの数 (1 秒あたり)。 代わりに新しい Http1xx を組み合わせたメトリクスを使用することをお勧めします。
GremlinHttp2xx	Gremlin エンドポイントの HTTP 2xx レスポンスの数 (1 秒あたり)。

メトリクス	説明
	代わりに新しい Http2xx を組み合わせたメトリクスを使用することをお勧めします。
GremlinHttp4xx	Gremlin エンドポイントの HTTP 4xx エラーの数 (1 秒あたり)。 代わりに新しい Http4xx を組み合わせたメトリクスを使用することをお勧めします。
GremlinHttp5xx	Gremlin エンドポイントの HTTP 5xx エラーの数 (1 秒あたり)。 代わりに新しい Http5xx を組み合わせたメトリクスを使用することをお勧めします。
GremlinErrors	Gremlin がトラバーサルしたエラー数。
GremlinRequests	Gremlin エンジンへのリクエスト数。
GremlinWebSocketSuccess	Gremlin エンドポイントへの 1 秒あたりの正常な WebSocket 接続の数。
GremlinWebSocketClientErrors	Gremlin エンドポイントでの 1 秒あたりの WebSocket クライアントエラーの数。
GremlinWebSocketServerErrors	Gremlin エンドポイントでの 1 秒あたりの WebSocket サーバーエラーの数。
GremlinWebSocketAvailableConnections	現在利用可能な潜在的な WebSocket 接続の数。
Http100	エンドポイントに対する HTTP 100 レスポンスの数 (1 秒あたり)。 代わりに新しい Http1xx を組み合わせたメトリクスを使用することをお勧めします。

メトリクス	説明
Http101	エンドポイントに対する HTTP 101 レスポンスの数 (1 秒あたり)。 代わりに新しい Http1xx を組み合わせたメトリクスを使用することをお勧めします。
Http1xx	エンドポイントの HTTP 1xx レスポンスの数 (1 秒あたり)。
Http200	エンドポイントに対する HTTP 200 レスポンスの数 (1 秒あたり)。 代わりに新しい Http2xx を組み合わせたメトリクスを使用することをお勧めします。
Http2xx	エンドポイントの HTTP 2xx レスポンスの数 (1 秒あたり)。
Http400	エンドポイントの HTTP 400 エラーの数 (1 秒あたり)。 代わりに新しい Http4xx を組み合わせたメトリクスを使用することをお勧めします。
Http403	エンドポイントの HTTP 403 エラーの数 (1 秒あたり)。 代わりに新しい Http4xx を組み合わせたメトリクスを使用することをお勧めします。
Http405	エンドポイントの HTTP 405 エラーの数 (1 秒あたり)。 代わりに新しい Http4xx を組み合わせたメトリクスを使用することをお勧めします。

メトリクス	説明
Http413	エンドポイントの HTTP 413 エラーの数 (1 秒あたり)。 代わりに新しい Http4xx を組み合わせたメトリクスを使用することをお勧めします。
Http429	エンドポイントの HTTP 429 エラーの数 (1 秒あたり)。 代わりに新しい Http4xx を組み合わせたメトリクスを使用することをお勧めします。
Http4xx	エンドポイントの HTTP 4xx エラーの数 (1 秒あたり)。
Http500	エンドポイントの HTTP 500 エラーの数 (1 秒あたり)。 代わりに新しい Http5xx を組み合わせたメトリクスを使用することをお勧めします。
Http501	エンドポイントの HTTP 501 エラーの数 (1 秒あたり)。 代わりに新しい Http5xx を組み合わせたメトリクスを使用することをお勧めします。
Http5xx	エンドポイントの HTTP 5xx エラーの数 (1 秒あたり)。
LoaderErrors	Loader リクエストからエラーの数。
LoaderRequests	Loader リクエストの数。

メトリクス	説明
SparqlHttp1xx	SPARQL エンドポイントの HTTP 1xx レスポンスの数 (1 秒あたり)。 代わりに新しい Http1xx を組み合わせたメトリクスを使用することをお勧めします。
SparqlHttp2xx	SPARQL エンドポイントの HTTP 2xx レスポンスの数 (1 秒あたり)。 代わりに新しい Http2xx を組み合わせたメトリクスを使用することをお勧めします。
SparqlHttp4xx	SPARQL エンドポイントの HTTP 4xx エラーの数 (1 秒あたり)。 代わりに新しい Http4xx を組み合わせたメトリクスを使用することをお勧めします。
SparqlHttp5xx	SPARQL エンドポイントの HTTP 5xx エラーの数 (1 秒あたり)。 代わりに新しい Http5xx を組み合わせたメトリクスを使用することをお勧めします。
SparqlErrors	SPARQL クエリのエラーの数。
SparqlRequests	SPARQL エンジンへのリクエストの数。
StatusErrors	ステータスエンドポイントからのエラーの数。
StatusRequests	ステータスエンドポイントへのリクエストの数。

Neptune CloudWatch ディメンション

Amazon Neptune のメトリクスは、アカウント、グラフ名、オペレーションなどの値で分類されます。Amazon CloudWatch コンソールを使用して、次の表のいずれかのディメンションとともに Neptune データを取得できます。

ディメンション	説明
`DBInstanceIdentifier`	クラスター内の特定のデータベースインスタンスに対してリクエストしたデータをフィルタ処理します。
`DBClusterIdentifier`	特定の Neptune DB クラスターに対してリクエストするデータをフィルタ処理します。
`DBClusterIdentifier` , `EngineName`	クラスターに基づいてデータをフィルタ処理します。すべての Neptune インスタンスに使用するエンジン名は `neptune` です。
`DBClusterIdentifier` , `Role`	特定の Neptune DB クラスターに対してリクエストしたデータをフィルタ処理して、インスタンスロール (WRITER/READER) 別にメトリクスを集計します。例えば、クラスターに属するすべての READER インスタンスのメトリクスを集計できます。
`DBClusterIdentifier` , `SourceRegion`	グローバルデータベースのプライマリリージョンのプライマリクラスターでデータをフィルタリングします。
`DatabaseClass`	特定のデータベースクラスに属するすべてのインスタンスに対してリクエストしたデータをフィルタ処理します。たとえば、メトリクスを組み合わせて、db.r4.large というデータベースクラスに属する全インスタンスを抽出することができます。

ディメンション	説明
EngineName	すべての Neptune インスタンスに使用するエンジン名は neptune です。
GlobalDbDBClusterIdentifier , SecondaryRegion	セカンダリリージョンの指定されたグローバルデータベースのセカンダリクラスタでデータをフィルタリングします。

Amazon Neptune クラスターで監査ログを使用する

Amazon Neptune DB クラスターのアクティビティを監査するには、DB クラスターパラメータを設定して監査ログの収集を有効にします。監査ログを有効にすると、この機能を使用して、サポートされているイベントの任意の組み合わせを記録できます。監査ログは、表示またはダウンロードして確認することができます。

Neptune 監査ログの有効化

監査ログを有効 (1) または無効 (0) にするには、neptune_enable_audit_log パラメータを使用します。

DB クラスターで使用されているパラメータグループにこのパラメータを設定します。「」に示されている手順を使用してパラメータDB クラスターパラメータグループまたは DB パラメータグループの編集を変更するか AWS Management Console、modify-db-cluster-parameter-group AWS CLI コマンドまたは ModifyDBClusterParameter Group API コマンドを使用してパラメータをプログラムで変更できます。

変更を適用するには、このパラメータを変更した後に DB クラスターを再起動する必要があります。

コンソールを使用して Neptune 監査ログを表示する

AWS Management Consoleを使用して、監査ログを表示およびダウンロードできます。[インスタンス] ページで、DB インスタンスをクリックして詳細を表示し、[ログ] セクションまでスクロールします。

ログファイルをダウンロードするには、[ログ] セクションでファイルを選択してから、[ダウンロード] を選択します。

Neptune 監査ログの詳細

ログファイルは UTF-8 形式です。ログは、複数のファイルに書き込まれます。ファイル数は、インスタンスのサイズによって異なります。最新のイベントを表示するには、すべての監査ログファイルの確認が必要な場合があります。

ログのエントリは、順番になっていません。並べ替えには、`timestamp` 値を使用できます。

ログファイルは、合計 100 MB に達するとローテーションされます。この制限は設定できません。

監査ログファイルの行には、次のカンマ区切りの情報が次の順序で含まれています。

フィールド	説明
タイムスタンプ	記録されたイベントの UNIX タイムスタンプ (マイクロ秒の精度)。
ClientHost	ユーザーの接続元のホスト名または IP。
ServerHost	イベントが記録されているインスタンスのホスト名または IP。
ConnectionType	接続タイプ。`Websocket` 、`HTTP_POST` 、`HTTP_GET`、または `Bolt` のいずれかです。

呼び出し元の IAM ARN	リクエストに署名するために使用される IAM ユーザーまたは IAM ロールの ARN。IAM 認証が無効な場合は空です。形式は次のとおりです。
	`arn:`*`partition`*` :`*`service`*`:`*`region`*`:`*`account`*`:`*`resource`*
	例:
	`arn:aws:iam::123456789012:user/Anna`
	`arn:aws:sts::123456789012:assumed-role/AWSNeptuneNotebookRole/SageMaker`

Auth Context	認証情報を持つシリアル化された JSON オブジェクトが含まれます。ユーザーが認証された場合、フィールド `authenticationSucceeded` は `True` です。 IAM 認証が無効な場合は空です。

フィールド	説明
HttpHeader	HTTP ヘッダー情報。クエリを含めることができます。および Bolt 接続の場合は WebSocket空です。
ペイロード	Gremlin、SPARQL、または openCypher クエリ。

Amazon Logs への Neptune CloudWatch ログの発行

Amazon Logs のロググループに監査ログデータやスロークエリログデータを発行するように Neptune DB CloudWatch クラスターを設定できます。 CloudWatch Logs を使用すると、ログデータのリアルタイム分析を実行し、 CloudWatch を使用してアラームを作成し、メトリクスを表示できます。 CloudWatch Logs を使用して、ログレコードを耐久性の高いストレージに保存できます。

監査ログを CloudWatch Logs に発行するには、監査ログを明示的に有効にする必要があります (「」を参照監査ログの有効化)。同様に、スロークエリログを CloudWatch ログに発行するには、スロークエリログを明示的に有効にする必要があります (「」を参照Amazon Neptune スロークエリロギングの使用)。

> **Note**
>
> 以下の点に注意してください。
>
> - にログを発行すると、追加料金が適用されます CloudWatch。詳細については、CloudWatch 料金表ページを参照してください。
> - 中国 (北京) または中国 (寧夏) リージョンの CloudWatch ログにログを発行することはできません。
> - Neptune は、無効化された監査ログデータをエクスポートする場合に、既存のロググループまたはログストリームを削除しません。ログデータのエクスポートが無効になっている場合、既存のログデータはログ保持期間に応じて CloudWatch Logs で引き続き使用できますが、保存された監査ログデータに対して料金が発生します。Logs コンソール、、 AWS CLIまたは CloudWatch Logs API を使用して、ログストリームと CloudWatch ロググループを削除できます。

コンソールを使用して Neptune ログを CloudWatch ログに発行する

コンソールから Neptune ログを CloudWatch ログに発行するには

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/neptune/home で Amazon Neptune コンソールを開きます。
2. ナビゲーションペインで [データベース] を選択します。
3. ログデータを公開する DB クラスターを選択します。
4. [アクション]、[変更] の順に選択します。
5. 「ログのエクスポート」セクションで、ログへの発行を開始する CloudWatch ログを選択します。
6. [続行] を選択し、概要ページで [Modify DB Cluster (DB クラスターの変更)] を選択します。

CLI を使用して Neptune 監査ログを CloudWatch Logs に発行する

次のパラメータを指定して `create-db-cluster` コマンドを使用して AWS CLI 、監査ログを CloudWatch Logs に発行する新しい DB クラスターを作成できます。

```
aws neptune create-db-cluster \
    --region us-east-1  \
    --db-cluster-identifier my_db_cluster_id \
    --engine neptune \
    --enable-cloudwatch-logs-exports '["audit"]'
```

以下のパラメータを指定して AWS CLI `modify-db-cluster` コマンドを使用して、監査ログを CloudWatch Logs に発行するように既存の DB クラスターを設定できます。

```
aws neptune modify-db-cluster \
    --region us-east-1  \
    --db-cluster-identifier my_db_cluster_id \
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":["audit"]}'
```

CLI を使用して Neptune スロークエリログを CloudWatch Logs に発行する

コマンド`create-db-cluster`と以下のパラメータを使用して AWS CLI 、スロークエリログを CloudWatch Logs に発行する新しい DB クラスターを作成することもできます。

```
aws neptune create-db-cluster \
```

```
    --region us-east-1  \
    --db-cluster-identifier my_db_cluster_id \
    --engine neptune \
    --enable-cloudwatch-logs-exports '["slowquery"]'
```

同様に、以下のパラメータを指定して AWS CLI `modify-db-cluster` コマンドを使用して、スロークエリログを CloudWatch Logs に発行するように既存の DB クラスターを設定できます。

```
aws neptune modify-db-cluster --region us-east-1  \
    --db-cluster-identifier my_db_cluster_id \
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":["slowquery"]}'
```

Amazon での Neptune ログイベントのモニタリング CloudWatch

Neptune ログを有効にすると、Amazon CloudWatch Logs でログイベントをモニタリングできます。新しいクラスターロググループは、*cluster-name* が DB クラスター名となり、*log_type* がログタイプとなる次のプレフィックスの Neptune DB クラスターに自動的に作成されます。

```
/aws/neptune/cluster-name/log_type
```

たとえば、エクスポート関数を設定して、`mydbcluster` という名前の DB クラスターの監査ログを作成すると、ログデータは、`/aws/neptune/mydbcluster/audit` ロググループに保存されます。

DB クラスターのすべての DB インスタンス におけるすべてのイベントは、別々のログストリーミングを使用してロググループにプッシュされます。

指定する名前のロググループがすでに存在する場合、Neptune はこのロググループを使用して Neptune DB クラスターにログデータをエクスポートします。などの自動設定を使用して、事前定義された ログ保持期間 AWS CloudFormation、メトリクスフィルター、カスタマーアクセスを持つロググループを作成できます。それ以外の場合、新しいロググループは、デフォルトのログ保持期間「有効期限なし」を使用して CloudWatch ログに自動的に作成されます。

CloudWatch Logs コンソール、 AWS CLI、または CloudWatch Logs API を使用して、ログの保持期間を変更できます。ログのログ保持期間の変更の詳細については、 CloudWatch 「 ログのログデータ保持期間の変更 CloudWatch 」を参照してください。

CloudWatch Logs コンソール、、または CloudWatch Logs API を使用して AWS CLI、DB クラスターのログイベント内の情報を検索できます。検索およびログデータのフィルタ処理の詳細については、「ログデータの検索およびフィルタ処理」を参照してください。

Neptune ノートブックの Amazon CloudWatch Logs の有効化

CloudWatch Neptune ノートブックのログはデフォルトで無効になっています。デバッグやその他の目的で有効にするには、次の手順に従います。

AWS Management Console を使用して Neptune ノートブックの CloudWatch ログを有効にする

1. https://console.aws.amazon.com/sagemaker/ で Amazon SageMaker コンソールを開きます。
2. 左側のナビゲーションペインで [ノートブック] を選択し、[ノートブックインスタンス] を選択します。ログを有効にする Neptune ノートブックの名前を探します。
3. 上記のステップで説明したノートブックインスタンスの名前を選択して、詳細ページに移動します。
4. ノートブックインスタンスが実行中の場合は、ノートブックの詳細ページの右上にある [停止] ボタンを選択します。
5. [アクセス許可と暗号化] に、[IAM ロール ARN] のフィールドがあります。このフィールドのリンクを選択して、このノートブックインスタンスが実行される IAM ロールに移動します。
6. 以下のポリシーを作成します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogDelivery",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:DeleteLogDelivery",
        "logs:Describe*",
        "logs:GetLogDelivery",
        "logs:GetLogEvents",
        "logs:ListLogDeliveries",
        "logs:PutLogEvents",
        "logs:PutResourcePolicy",
        "logs:UpdateLogDelivery"
      ],
      "Resource": "*"
    }
  ]
```

```
}
```

7. この新しいポリシーを保存し、ステップ 8 で IAM ロールにアタッチします。
8. ノートブックインスタンスの詳細ページの右上にある「開始」を選択します SageMaker。
9. ログが流れ始めると、詳細ページの [ノートブックインスタンス設定] セクションの左下にある [ライフサイクル設定] というラベルの付いたフィールドの下に [ログを表示] リンクが表示されます。

ノートブックの起動に失敗すると、 SageMaker コンソールのノートブックの詳細ページに、ノートブックインスタンスの起動に 5 分以上かかったことを示すメッセージが表示されます。この問題に関連する CloudWatch ログは、 という名前で確認できます*(your-notebook-name)*/`LifecycleConfigOnStart`。

必要に応じて、「Amazon SageMaker で Amazon イベントを CloudWatch ログに記録する」を参照してください。

Amazon Neptune スロークエリロギングの使用

実行速度が遅いクエリの特定、デバッグ、最適化は難しい場合があります。Neptune のスロークエリロギングを有効にすると、このプロセスを簡単にするために、長時間実行されるすべてのクエリの属性が自動的にログに記録されます。

> **Note**
>
> スロークエリロギングは Neptune エンジンリリース 1.2.1.0 で導入されました。

スロークエリロギングを有効にするには neptune_enable_slow_query_log DB クラスターパラメータを使用します。デフォルトでは、このパラメータは `disabled` に設定されます。これを `info` または `debug` に設定すると、スロークエリロギングが有効になります。`info` 設定は、実行速度が遅い各クエリの少数の有用な属性を記録するのに対して、`debug` 設定は、使用可能なすべての属性を記録します。

実行速度が遅いと見なされるクエリのしきい値を設定するには、neptune_slow_query_log_threshold DB クラスターパラメータを使用して、実行中のクエリが遅いと見なされ、低速クエリのロギングが有効になっている場合にログに記録されるまでの時間をミリ秒単位で指定します。デフォルト値は 5,000 ミリ秒 (5 秒) です。

これらの DB クラスターパラメータは、、 AWS Management Consoleまたは modify-db-cluster-parameter-group AWS CLI コマンド、または ModifyDBClusterParameter グループ管理関数を使用して設定できます。

> **Note**
>
> スロークエリのロギングパラメータは動的です。つまり、値を変更しても DB クラスターを再起動する必要はなく、再起動の原因にもなりません。

でスロークエリログを表示するには AWS Management Console

スロークエリログは AWS Management Console、次のように で表示およびダウンロードできます。

[インスタンス] ページで、DB インスタンスを選択し、[ログ] セクションまでスクロールします。その後、そこでログファイルを選択し、[ダウンロード] を選択してダウンロードできます。

Neptune スロークエリロギングによって生成されたファイル

Neptune のスロークエリロギングによって生成されるログファイルには、次のような特徴があります。

- ファイルは UTF-8 としてエンコードされます。
- クエリとその属性は JSON 形式で記録されます。
- NULL や空の属性は、`queryTime` データを除いてログに記録されません。
- ログは複数のファイルにまたがり、ファイル数はインスタンスのサイズによって異なります。
- ログのエントリは、順番になっていません。並べ替えには、`timestamp` 値を使用できます。
- 最新のイベントを表示するには、すべてのログファイルの確認が必要な場合があります。
- ログファイルは、合計 100 MB に達するとローテーションされます。この制限は設定できません。

`info` モードで記録されるクエリ属性

`neptune_enable_slow_query_log`DB クラスターパラメータがに設定されている場合、スロークエリでは以下の属性が記録されます`info`。

グループ	属性	説明
リクエストResponseMetadata	`requestId`	クエリのリクエスト ID。
	`requestType`	HTTP や などのリクエストタイプ WebSocket。
	`responseStatusCode`	200 などのクエリ応答ステータスコード。
	`exceptionClass`	クエリ実行後に返されるエラーの例外クラス。
QueryStats	`query`	クエリ文字列
	`queryFingerprint`	クエリのフィンガープリント。
	`queryLanguage`	グレムリン、SPA RQL、openCypher などのクエリ言語。
メモリ統計	`allocatedPermits`	クエリに割り当てられて許可されます。
	`approximateUsedMemoryBytes`	実行中にクエリが使用したおおよそのメモリ。
QueryTime	`startTime`	クエリ開始時刻 (UTC)。
	`overallRunTimeMs`	クエリの合計実行時間 (ミリ秒単位)。
	`parsingTimeMs`	クエリの解析時間 (ミリ秒単位)。
	`waitingTimeMs`	Gremlin/sparql/openCypher のクエリキュー待機時間 (ミリ秒単位)

グループ	属性	説明
	executionTimeMs	クエリの実行時間 (ミリ秒単位)。
	serializationTimeMs	クエリのシリアル化時間 (ミリ秒)。
ステートメント/カウンター	scanned	スキャンされたステートメントの数。
	written	書き込まれたステートメントの数。
	deleted	削除された要素の数。
トランザクションカウンター	committed	コミットされたトランザクションの数。
	rolledBack	ロールバックされるトランザクションの数。
頂点/カウンター	added	追加された頂点の数。
	removed	削除された頂点の数。
	propertiesAdded	追加された頂点プロパティの数。
	propertiesRemoved	削除された頂点プロパティの数。
エッジカウンター	added	追加されたエッジの数。
	removed	削除されたエッジの数。
	propertiesAdded	追加されたエッジプロパティの数。

グループ	属性	説明
	propertiesRemoved	削除されたエッジプロパティの数。
結果キャッシュ	hitCount	結果キャッシュのヒット数。
	missCount	結果キャッシュのミスカウント。
	putCount	結果キャッシュのプット数。
ConcurrentExecutions	acceptedQueryCountAtStart	並列クエリは、開始時に現在のクエリ実行で受け付けられました。
	runningQueryCountAtStart	開始時に現在のクエリ実行時に実行される並列クエリ。
	acceptedQueryCountAtEnd	現在のクエリ実行が終了した状態で並列クエリが受け入れられます。
	runningQueryCountAtEnd	現在のクエリ実行を終了して実行中の並列クエリ。
クエリー/バッチ	queryProcessingBatchSize	クエリ処理中のBatch サイズ。
	querySerialisationBatchSize	クエリのシリアル化中のBatch サイズ。

debug モードで記録されるクエリ属性

neptune_enable_slow_query_logDB クラスターパラメータをに設定するとdebug、in mode として記録される属性に加えて、以下のストレージカウンター属性が記録されます。info

属性	説明
statementsScannedInAllIndexes	すべてのインデックスでステートメントがスキャンされました。
statementsScannedSPOGIndex	SPOG インデックスでスキャンされたステートメント。
statementsScannedPOGSIndex	POGS インデックスでスキャンされたステートメント。
statementsScannedGPSOIndex	GPSO インデックスでスキャンされたステートメント。
statementsScannedOSGPIndex	OSGP インデックスでスキャンされたステートメント。
statementsScannedInChunk	ステートメントはチャンクにまとめてスキャンされました。
postFilteredStatementScans	スキャン後のポストフィルタリング後に残ったステートメント。
distinctStatementScans	個別のステートメントがスキャンされました。
statementsReadInAllIndexes	スキャン後に読み込まれたステートメントは、すべてのインデックスでフィルタリングされます。
statementsReadSPOGIndex	SPOG インデックスのスキャンポストフィルタリング後に読み込まれるステートメント。
statementsReadPOGSIndex	POGS インデックスのスキャンポストフィルタリング後に読み込まれるステートメント。
statementsReadGPSOIndex	GPSO インデックスのスキャンポストフィルタリング後に読み込まれるステートメント。

属性	説明
statementsReadOSGPIndex	OSGP インデックスのスキャンポストフィルタリング後に読み込まれたステートメント。
accessPathSearches	アクセスパス検索の数。
fullyBoundedAccessPathSearches	完全に制限されたキーアクセスパス検索の数。
accessPathSearchedByPrefix	プレフィックスで検索されたアクセスパスの数。
searchesWhereRecordsWereFound	1 つ以上のレコードが出力された検索の数。
searchesWhereRecordsWereNotFound	レコードが出力されなかった検索の数。
totalRecordsFoundInSearches	すべての検索で見つかったレコードの合計数。
statementsInsertedInAllIndexes	すべてのインデックスに挿入されたステートメントの数。
statementsUpdatedInAllIndexes	すべてのインデックスで更新されたステートメントの数。
statementsDeletedInAllIndexes	すべてのインデックスで削除されたステートメントの数。
predicateCount	述語の数。
dictionaryReadsFromValueToIdTable	値から ID テーブルにディクショナリを読み込む回数。
dictionaryReadsFromIdToValueTable	バリューテーブルの ID からのディクショナリ読み取り数。
dictionaryWritesToValueToIdTable	ID テーブルの値に書き込まれたディクショナリの数です。
dictionaryWritesToIdToValueTable	ID からバリューテーブルに書き込まれたディクショナリの数です。

属性	説明
rangeCountsInAllIndexes	すべてのインデックスの範囲カウント数。
deadlockCount	クエリ内のデッドロックの数。
singleCardinalityInserts	単一カーディナリティ・インサートの実行回数。
singleCardinalityInsertDeletions	1 回のカーディナリティ挿入中に削除されたステートメントの数。

スロークエリのデバッグログの例

次の Gremlin クエリは、スロークエリに設定されたしきい値よりも実行に時間がかかる場合があります。

```
gremlin=g.V().has('code','AUS').repeat(out().simplePath()).until(has('code','AGR')).path().by('
```

その場合、スロークエリロギングがデバッグモードで有効になっていると、クエリの次の属性が次のような形式でログに記録されます。

```
{
  "requestResponseMetadata": {
    "requestId": "5311e493-0e98-457e-9131-d250a2ce1e12",
    "requestType": "HTTP_GET",
    "responseStatusCode": 200
  },
  "queryStats": {
    "query":
 "gremlin=g.V().has('code','AUS').repeat(out().simplePath()).until(has('code','AGR')).path().by
    "queryFingerprint":
 "g.V().has(string0,string1).repeat(__.out().simplePath()).until(__.has(string0,string2)).path(
    "queryLanguage": "Gremlin"
  },
  "memoryStats": {
    "allocatedPermits": 20,
    "approximateUsedMemoryBytes": 14838
  },
  "queryTimeStats": {
```

```
    "startTime": "23/02/2023 11:42:52.657",
    "overallRunTimeMs": 2249,
    "executionTimeMs": 2229,
    "serializationTimeMs": 13
  },
  "statementCounters": {
    "read": 69979
  },
  "transactionCounters": {
    "committed": 1
  },
  "concurrentExecutionStats": {
    "acceptedQueryCountAtStart": 1
  },
  "queryBatchStats": {
    "queryProcessingBatchSize": 1000,
    "querySerialisationBatchSize": 1000
  },
  "storageCounters": {
    "statementsScannedInAllIndexes": 69979,
    "statementsScannedSPOGIndex": 44936,
    "statementsScannedPOGSIndex": 4,
    "statementsScannedGPSOIndex": 25039,
    "statementsReadInAllIndexes": 68566,
    "statementsReadSPOGIndex": 43544,
    "statementsReadPOGSIndex": 2,
    "statementsReadGPSOIndex": 25020,
    "accessPathSearches": 27,
    "fullyBoundedAccessPathSearches": 27,
    "dictionaryReadsFromValueToIdTable": 10,
    "dictionaryReadsFromIdToValueTable": 17,
    "rangeCountsInAllIndexes": 4
  }
}
```


を使用した Amazon Neptune API コールのログ記録 AWS CloudTrail

Amazon Neptune は AWS CloudTrail、Neptune コンソールからの呼び出しや Neptune API へのコード呼び出しを含む、Neptune の API コールをイベントとして Neptune. CloudTrail captures のユー

ザー、ロール、または AWS サービスによって実行されたアクションを記録するサービスである と統合APIs。

CloudTrail は、インスタンスやクラスターの作成など、Neptune Management API コールのイベントのみをログに記録します。グラフへの変更を監査する場合、監査ログを使用できます。詳細については、「Amazon Neptune クラスターで監査ログを使用する」を参照してください。

> **Important**
>
> Amazon Neptune コンソール、、および API コールは AWS CLI、Amazon Relational Database Service (Amazon RDS) API に対して行われたコールとして記録されます。

証跡を作成する場合は、Neptune の CloudTrail イベントなど、Amazon S3 バケットへのイベントの継続的な配信を有効にすることができます。証跡を設定しない場合でも、 CloudTrail コンソールのイベント履歴 で最新のイベントを表示できます。によって収集された情報を使用して CloudTrail、Neptune に対するリクエスト、リクエスト元の IP アドレス、リクエスト者、リクエスト日時などの詳細を確認できます。

の詳細については CloudTrail、「 AWS CloudTrail ユーザーガイド」を参照してください。

の Neptune 情報 CloudTrail

CloudTrail AWS アカウントを作成すると、 がアカウントで有効になります。Amazon Neptune でアクティビティが発生すると、そのアクティビティは CloudTrail イベント履歴 の他の AWS サービスイベントとともにイベントに記録されます。 AWS アカウントで最近のイベントを表示、検索、ダウンロードできます。詳細については、「イベント履歴を使用した CloudTrail イベントの表示」を参照してください。

Neptune のイベントなど、 AWS アカウント内のイベントの継続的な記録については、証跡を作成します。証跡により CloudTrail 、 はログファイルを Amazon S3 バケットに配信できます。デフォルトでは、コンソールで証跡を作成すると、すべての リージョンに証跡が適用されます。証跡は、AWS パーティション内のすべてのリージョンからのイベントをログに記録し、指定した Amazon S3 バケットにログファイルを配信します。さらに、 CloudTrail ログで収集されたイベントデータをより詳細に分析し、それに基づいて行動するように、他の AWS サービスを設定できます。詳細については、以下をご覧ください。

- 証跡を作成するための概要
- CloudTrail サポートされているサービスと統合

- の Amazon SNS 通知の設定 CloudTrail
- 複数のリージョンからの CloudTrail ログファイルの受信と複数のアカウントからの CloudTrail ログファイルの受信

Neptune コンソール、Neptune コマンドラインインターフェイス、または Neptune SDK API を使用して AWS アカウントに代わってアクションが実行された場合、 はアクションを Amazon RDS API への呼び出しとして AWS CloudTrail ログに記録します。 APIs 例えば、Neptune コンソールを使用して DB インスタンスを変更するか、 AWS CLI modify-db-instance コマンドを呼び出すと、 AWS CloudTrail ログには Amazon RDS API ModifyDBInstance アクションの呼び出しが表示されます。によってログに記録される Neptune API アクションのリストについては AWS CloudTrail、「Neptune API リファレンス」を参照してください。

> **Note**
>
> AWS CloudTrail は、インスタンスやクラスターの作成など、Neptune Management API コールのイベントのみをログに記録します。グラフへの変更を監査する場合、監査ログを使用できます。詳細については、「Amazon Neptune クラスターで監査ログを使用する」を参照してください。

各イベントまたはログエントリには、リクエストの生成者に関する情報が含まれます。同一性情報は次の判断に役立ちます。

- リクエストが、ルートと IAM ユーザー認証情報のどちらを使用して送信されたか。
- リクエストがロールまたはフェデレーションユーザーの一時的なセキュリティ認証情報を使用して行われたかどうか。
- リクエストが別の AWS サービスによって行われたかどうか。

詳細については、CloudTrail userIdentity Element」を参照してください。

Neptune ログファイルエントリの概要

証跡は、指定した Amazon S3 バケットにイベントをログファイルとして配信できるようにする設定です。 CloudTrail ログファイルには 1 つ以上のログエントリが含まれます。イベントは任意のソースからの単一のリクエストを表し、リクエストされたアクション、アクションの日時、リクエストパラメータなどに関する情報が含まれます。 CloudTrail ログファイルはパブリック API コールの順序付けられたスタックトレースではないため、特定の順序では表示されません。

次の例は、DB インスタンスのスナップショットを作成し、Neptune コンソールを使用してそのインスタンスを削除したユーザーの CloudTrail ログを示しています。コンソールは、userAgent 要素によって識別されます。コンソールによりリクエストされた API 呼び出し (CreateDBSnapshot と DeleteDBInstance) は各レコードの eventName 要素で確認できます。ユーザーに関する情報 (Alice) は userIdentity 要素で確認できます。


```
{
  Records:[
  {
    "awsRegion":"us-west-2",
    "eventName":"CreateDBSnapshot",
    "eventSource":"",
    "eventTime":"2014-01-14T16:23:49Z",
    "eventVersion":"1.0",
    "sourceIPAddress":"192.0.2.01",
    "userAgent":"AWS Console, aws-sdk-java\/unknown-version Linux\/2.6.18-
kaos_fleet-1108-prod.2 Java_HotSpot(TM)_64-Bit_Server_VM\/24.45-b08",
    "userIdentity":
    {
      "accessKeyId":"",
      "accountId":"123456789012",
      "arn":"arn:aws:iam::123456789012:user/Alice",
      "principalId":"AIDAI2JXM4FBZZEXAMPLE",
      "sessionContext":
      {
        "attributes":
        {
          "creationDate":"2014-01-14T15:55:59Z",
          "mfaAuthenticated":false
        }
      },
      "type":"IAMUser",
      "userName":"Alice"
    }
  },
  {
    "awsRegion":"us-west-2",
    "eventName":"DeleteDBInstance",
    "eventSource":"",
    "eventTime":"2014-01-14T16:28:27Z",
    "eventVersion":"1.0",
    "sourceIPAddress":"192.0.2.01",
```

```
    "userAgent":"AWS Console, aws-sdk-java\/unknown-version Linux\/2.6.18-
kaos_fleet-1108-prod.2 Java_HotSpot(TM)_64-Bit_Server_VM\/24.45-b08",
    "userIdentity":
    {
      "accessKeyId":"",
      "accountId":"123456789012",
      "arn":"arn:aws:iam::123456789012:user/Alice",
      "principalId":"AIDAI2JXM4FBZZEXAMPLE",
      "sessionContext":
      {
        "attributes":
        {
          "creationDate":"2014-01-14T15:55:59Z",
          "mfaAuthenticated":false
        }
      },
      "type":"IAMUser",
      "userName":"Alice"
    }
  }
  ]
}
```

Neptune イベント通知の使用

トピック

- Amazon Neptune イベントのカテゴリおよびイベントメッセージ
- Neptune イベント通知にサブスクライブする
- Neptune イベント通知サブスクリプションの管理

Amazon Neptune では、Neptune のイベントが発生したときに、Amazon Simple Notification Service (Amazon SNS) を使用して通知を送信します。これらの通知は、E メール、テキストメッセージ、HTTP エンドポイントへの呼び出しなど、 リージョンの Amazon SNS でサポートされている任意の形式で送信できます。 AWS

Neptune は、サブスクライブ可能なカテゴリにイベントをグループ分けします。これにより、そのカテゴリのイベントが発生すると、通知を受け取ることができます。DB インスタンス、DB クラスター、DB スナップショット、DB クラススナップショット、または DB パラメータグループのイベントのカテゴリをサブスクライブできます。例えば、特定の DB インスタンスのバックアップカ

テゴリにサブスクライブした場合、DB インスタンスに影響するバックアップ関連のイベントが発生するたびに通知が送信されます。また、イベント通知サブスクリプションが変更されても、通知を受け取ります。

イベントは DB クラスターレベルと DB インスタンスレベルの両方で発生するため、DB クラスターまたは DB インスタンスをサブスクライブするとイベントを受信できます。

イベント通知は、サブスクリプションを作成するときに指定したアドレスに送信されます。いくつかの異なるサブスクリプションを作成することもできます。例えば、すべてのイベント通知を受信するサブスクリプションと、本番稼働用の DB インスタンスに関する重要なイベントのみを含むサブスクリプションを作成できます。サブスクリプションを削除せずに通知を簡単にオフにできます。これを行うには、Neptune コンソールで[Enabled](有効) ラジオボタンを[No] (なし) に設定します。

> **Important**
>
> Amazon Neptune はイベントストリームのイベントの順番を保証しません。イベントの順番は変わる場合があります。

Neptune では、Amazon SNS トピックの Amazon リソースネーム (ARN) を使用して、各サブスクリプションを識別します。Neptune コンソールでは、サブスクリプションの作成時に ARN が作成されます。

Neptune イベント通知の請求は、Amazon SNS を通じて行われます。使用したイベント通知に対して、Amazon SNS 料金が適用されます。詳細については、Amazon Simple Notification Service 料金表を参照してください。

Amazon Neptune イベントのカテゴリおよびイベントメッセージ

Neptune では、Neptune コンソールを使用してサブスクライブできるカテゴリ内で多数のイベントが生成されます。各カテゴリは 1 つのソースタイプに適用されます。ソースタイプには、DB インスタンス、DB スナップショット、DB パラメータグループが含まれます。

> **Note**
>
> Neptune では既存の Amazon RDS イベントの定義と ID が使用されます。

DB インスタンスから発生する Neptune イベント

次の表は、DB インスタンスがソースタイプである場合のイベントカテゴリの一覧表を示しています。

カテゴリ	Amazon RDS イベント ID	説明
高可用性	RDS-EVENT-0006	DB インスタンスが再起動しました。
	RDS-EVENT-0004	DB インスタンスがシャットダウンしました。
	RDS-EVENT-0022	グラフエンジンの再起動中にエラーが発生しました。
バックアップ	RDS-EVENT-0001	DB インスタンスをバックアップする
	RDS-EVENT-0002	DB インスタンスのバックアップが完了しました。
設定変更	RDS-EVENT-0009	DB インスタンスがセキュリティグループに追加されました。
	RDS-EVENT-0024	DB インスタンスがマルチ AZ DB インスタンスに変換されています。
	RDS-EVENT-0030	DB インスタンスがシングル AZ DB インスタンスに変換されています。

カテゴリ	Amazon RDS イベント ID	説明
	RDS-EVENT-0012	変更をデータベースインスタンスクラスに適用しています。
	RDS-EVENT-0018	この DB インスタンスの現在のストレージ設定を変更しています。
	RDS-EVENT-0011	この DB インスタンスのパラメータグループが変更されました。
	RDS-EVENT-0092	この DB インスタンスのパラメータグループの更新が完了しました。
	RDS-EVENT-0028	この DB インスタンスの自動バックアップが無効になっています。
	RDS-EVENT-0032	この DB インスタンスの自動バックアップが有効になっています。
	RDS-EVENT-0025	DB インスタンスがマルチ AZ DB インスタンスに変換されました。

カテゴリ	Amazon RDS イベント ID	説明
	RDS-EVENT-0029	DB インスタンスがシングル AZ DB インスタンスに変換されました。
	RDS-EVENT-0014	この DB インスタンスの DB インスタンスクラスが変更されました。
	RDS-EVENT-0017	この DB インスタンスのストレージ設定が変更されました。
	RDS-EVENT-0010	DB インスタンスはセキュリティグループから削除されました。
作成	RDS-EVENT-0005	DB インスタンスが作成されました。
削除	RDS-EVENT-0003	DB インスタンスが削除されました。
フェイルオーバー	RDS-EVENT-0034	Neptune はリクエストされたフェイルオーバーを実行できません。これは、DB インスタンスでフェイルオーバーが最近発生したためです。

カテゴリ	Amazon RDS イベント ID	説明
	RDS-EVENT-0013	マルチ AZ フェイルオーバーがスタートされました。このフェイルオーバーにより、スタンバイインスタンスの昇格が行われました。
	RDS-EVENT-0015	マルチ AZ フェイルオーバーが完了しました。このフェイルオーバーにより、スタンバイインスタンスの昇格が行われました。DNS によって新しいプライマリ DB インスタンスへの転送が行われるまで数分かかることがあります。
	RDS-EVENT-0065	インスタンスが部分的なフェイルオーバーから復旧しました。
	RDS-EVENT-0049	マルチ AZ フェイルオーバーが完了しました。
	RDS-EVENT-0050	マルチ AZ アクティベーションが、正常なインスタンスの復旧後にスタートされました。

カテゴリ	Amazon RDS イベント ID	説明
	RDS-EVENT-0051	マルチ AZ アクティベーションが完了しました。データベースがアクセス可能になりました。
	RDS-EVENT-0031	DB インスタンスは、互換性のない設定または基本的なストレージの問題により失敗しました。DB インスタンス point-in-time-restore の を開始します。
	RDS-EVENT-0036	DB インスタンスが互換性のないネットワーク上にあります。指定したサブネット ID の一部は無効であるか、存在しません。

カテゴリ	Amazon RDS イベント ID	説明
	RDS-EVENT-0035	DB インスタンスに無効なパラメータがあります。例えば、メモリ関連のパラメータがこのインスタンスクラスには高すぎる値に設定されていて、DB インスタンスをスタートできなかった場合、ユーザーはメモリのパラメータを変更して、DB インスタンスを再起動してください。
	RDS-EVENT-0082	Neptune は、Amazon S3 バケットからバックアップデータをコピーできませんでした。Neptune から Amazon S3 バケットにアクセスするためのアクセス権限が正しく設定されていない可能性があります。

カテゴリ	Amazon RDS イベント ID	説明
ストレージの減少	RDS-EVENT-0089	DB インスタンスは割り当てられたストレージの 90% 以上を使用しています。[Free Storage Space] メトリクスを使用して、DB インスタンスのストレージ容量をモニタリングできます。
	RDS-EVENT-0007	DB インスタンスに割り当てられているストレージに空き領域がなくなりました。この問題を解決するには、DB インスタンスに追加のストレージを割り当てる必要があります。
メンテナンス	RDS-EVENT-0026	DB インスタンスのオフラインメンテナンスが実行中です。現在、DB インスタンスは利用できません。
	RDS-EVENT-0027	DB インスタンスのオフラインメンテナンスが完了しました。現在、DB インスタンスは利用できます。

カテゴリ	Amazon RDS イベント ID	説明
	RDS-EVENT-0047	DB インスタンスへのパッチ適用が完了しました。
通知	RDS-EVENT-0044	オペレータが発行する通知。詳細については、イベントメッセージを参照してください。
	RDS-EVENT-0048	DB インスタンスへのパッチ適用が遅れました。
	RDS-EVENT-0087	DB インスタンスが停止されました。
	RDS-EVENT-0088	DB インスタンスがスタートされました。
	RDS-EVENT-0154	停止中の最大許容時間を超えたため、DB インスタンスが起動されています。
	RDS-EVENT-0158	DB インスタンスはアップグレードできない状態です。
	RDS-EVENT-0173	DB インスタンスにパッチが適用されました。

カテゴリ	Amazon RDS イベント ID	説明
リードレプリカ	RDS-EVENT-0045	リードレプリケーションプロセスでエラーが発生しました。詳細については、イベントメッセージを参照してください。
	RDS-EVENT-0046	リードレプリカはレプリケーションを再開しました。このメッセージは、初期にリードレプリカを作成したとき、またはレプリケーションが適切に機能していることを確認するモニタリングメッセージとして表示されます。このメッセージがRDS-EVENT-0045通知の後に表示される場合は、エラーの後またはレプリケーションが停止した後で、レプリケーションが再開されました。
	RDS-EVENT-0057	リードレプリカのレプリケーションは終了しました。

カテゴリ	Amazon RDS イベント ID	説明
	RDS-EVENT-0062	リードレプリカのレプリケーションは手動で停止されました。
	RDS-EVENT-0063	リードレプリカのレプリケーションはリセットされました。
復旧	RDS-EVENT-0020	DB インスタンスの復旧がスタートされました。復旧時間は、復旧するデータの量に応じて変わります。
	RDS-EVENT-0021	DB インスタンスの復旧が完了しました。
	RDS-EVENT-0023	手動バックアップがリクエストされましたが、現在、Neptune は DB スナップショットの作成中です。Neptune で DB スナップショットの作成が完了した後で、リクエストをもう一度送信してください。
	RDS-EVENT-0052	マルチ AZ インスタンスの復旧がスタートされました。復旧時間は、復旧するデータの量に応じて変わります。

カテゴリ	Amazon RDS イベント ID	説明
	RDS-EVENT-0053	マルチ AZ インスタンスの復旧が完了しました。
復元	RDS-EVENT-0008	DB インスタンスが DB スナップショットから復元されました。
	RDS-EVENT-0019	DB インスタンスがバックアップから point-in-time復元されました。

DB クラスターから発生する Neptune イベント

次の表は、DB クラスターがソースタイプである場合のイベントカテゴリとイベントのリストを示します。

カテゴリ	RDS イベント ID	説明
フェイルオーバー	RDS-EVENT-0069	DB クラスターのフェイルオーバーが失敗しました。
	RDS-EVENT-0070	DB クラスターのフェイルオーバーが再びスタートされました。
	RDS-EVENT-0071	DB クラスターのフェイルオーバーが終了しました。
	RDS-EVENT-0072	同一アベイラビリティーゾーン内で DB ク

カテゴリ	RDS イベント ID	説明
		ラスターのフェイルオーバーがスタートされました。
	RDS-EVENT-0073	異なるアベイラビリティーゾーン間で DB クラスターのフェイルオーバーがスタートされました。
	RDS-EVENT-0083	Neptune は、Amazon S3 バケットからバックアップデータをコピーできませんでした。Neptune から Amazon S3 バケットにアクセスするためのアクセス権限が正しく設定されていない可能性があります。
メンテナンス	RDS-EVENT-0156	DB クラスターで DB エンジンのマイナーバージョンをアップグレードできます。
通知	RDS-EVENT-0076	Neptune DB クラスターへの移行に失敗しました。

カテゴリ	RDS イベント ID	説明
	RDS-EVENT-0077	ソースデータベースから InnoDB にテーブルを変換しようとしましたが、Neptune DB クラスターへの移行中に失敗しました。
	RDS-EVENT-0150	DB クラスターが停止されました。
	RDS-EVENT-0151	DB クラスターがスタートされました。
	RDS-EVENT-0152	DB クラスターの停止に失敗しました。
	RDS-EVENT-0153	停止中の最大許容時間を超えたため、DB クラスターがスタートされています。

DB クラススナップショットから発生する Neptune イベント

次の表は、 DB クラスターのスナップショットがソースタイプである場合のイベントカテゴリとイベントのリストを示しています。

カテゴリ	RDS イベント ID	説明
backup	RDS-EVENT-0074	手動 DB クラススナップショット作成がスタートされました。

カテゴリ	RDS イベント ID	説明
backup	RDS-EVENT-0075	手動 DB クラススナップショットが作成されました。
通知	RDS-EVENT-0162	DB クラスターのスナップショットのエクスポートタスクが失敗しました。
通知	RDS-EVENT-0163	DB クラスターのスナップショットのエクスポートタスクがキャンセルされました。
通知	RDS-EVENT-0164	DB クラスターのスナップショットのエクスポートタスクが完了しました。
backup	RDS-EVENT-0168	自動クラススナップショットを作成しています。
backup	RDS-EVENT-0169	自動クラススナップショットが作成されました。
作成	RDS-EVENT-0170	DB クラスターが作成されました。
削除	RDS-EVENT-0171	DB クラスターが削除されました。

カテゴリ	RDS イベント ID	説明
通知	RDS-EVENT-0172	DB クラスターの名前を [古い DB クラスター名] から [新しい DB クラスター名] に変更しました。

DB クラスターのパラメータグループから発生する Neptune イベント

次の表は、DB クラスターパラメータグループがソースタイプである場合のイベントカテゴリとイベントを示しています。

カテゴリ	RDS イベント ID	説明
設定変更	RDS-EVENT-0037	パラメータグループが変更されています。

セキュリティグループから発信される Neptune イベント

次の表は、セキュリティグループがソースタイプである場合のイベントカテゴリとイベントのリストを示します。

カテゴリ	RDS イベント ID	説明
設定変更	RDS-EVENT-0038	セキュリティグループが変更されています。
failure	RDS-EVENT-0039	[ユーザー] が所有するセキュリティグループが存在しません。セキュリティグループの認可が取り消されました。

Neptune イベント通知にサブスクライブする

次のように、Neptune コンソールを使用してイベント通知を購読できます。

Neptune イベント通知にサブスクライブするには

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/neptune/home で Amazon Neptune コンソールを開きます。
2. ナビゲーションペインで、[Event subscriptions] (イベントサブスクリプション) を選択します。
3. [イベントサブスクリプション] ページで、[イベントサブスクリプションの作成] を選択します。
4. [イベントサブスクリプションの作成] ダイアログボックスで、次の操作を行います。

    a. [名前] に、イベント通知サブスクリプションの名前を入力します。
    b. [通知の送信先] で既存の Amazon SNS トピックの Amazon SNS ARN を選択するか、[トピックを作成] を選択してトピックの名前と受取人のリストを入力します。
    c. [ソースタイプ] で、ソースタイプを選択します。
    d. [はい] を選択して、サブスクリプションを有効にします。サブスクリプションは作成するが、通知はまだ送信しない場合は、[いいえ] を選択します。
    e. 選択したソースタイプに応じて、イベント通知を受け取る対象のイベントカテゴリとソースを選択します。
    f. [Create] (作成) を選択します。

Neptune イベント通知サブスクリプションの管理

Neptune コンソールのナビゲーションペインでイベントサブスクリプションを選択すると、サブスクリプションカテゴリと現在のサブスクリプションのリストを表示できます。

特定のサブスクリプションを変更または削除することもできます。

Neptune イベント通知サブスクリプションを変更する

現在の Neptune イベント通知サブスクリプションをリストするには

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/neptune/home で Amazon Neptune コンソールを開きます。
2. ナビゲーションペインで、[イベントサブスクリプション] を選択します。[イベントサブスクリプション] ペインにイベント通知サブスクリプションが一覧表示されます。

3. [イベントサブスクリプション] ペインで、変更するサブスクリプションを選択し、[編集] をクリックします。
4. [ターゲット] セクションまたは [ソース] セクションのいずれかでサブスクリプションを変更します。ソース識別子をで選択または選択解除することで、ソース識別子をソースセクションに追加または削除できます。
5. [編集] を選択します。Neptune コンソールで、サブスクリプションが変更されることが示されます。

Neptune イベント通知サブスクリプションを削除する

不要になったサブスクリプションは削除できます。トピックへのすべてのサブスクライバは、サブスクリプションにより指定されたイベント通知を受け取らなくなります。

Neptuneイベント通知サブスクリプションを削除するには

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/neptune/home で Amazon Neptune コンソールを開きます。
2. ナビゲーションペインで、[Event subscriptions] (イベントサブスクリプション) を選択します。
3. [イベントサブスクリプション] ペインで、削除するサブスクリプションを選択します。
4. [削除] を選択します。
5. Neptune コンソールで、サブスクリプションが削除されることが示されます。

Amazon Neptune リソースのタグ付け

&neptune; タグを使用して &neptune; リソースにメタデータを追加できます。さらに、 AWS Identity and Access Management (IAM) ポリシーでタグを使用して Neptune リソースへのアクセスを管理し、それらのリソースに適用できるアクションを制御できます。最後に、タグを使用して、類似のリソースの費用をグループ化することで、コストを追跡できます。

すべての Neptune リソースには、次のようなタグを付けることができます。

- DB インスタンス
- DB クラスター
- リードレプリカ
- DB スナップショット

- DB クラスタースナップショット
- イベントサブスクリプション
- DB パラメータグループ
- DB クラスターのパラメータグループ
- DB サブネットグループ

Neptune リソースタグの概要

Amazon Neptune タグは、Neptune リソースを定義して Neptune リソースに関連付ける名前と値のペアです。その名前はキーと呼ばれます。キーの値の指定は省略可能です。タグを使用して、Neptune リソースに任意の情報を割り当てることができます。例えば、タグキーを使用してカテゴリを定義し、タグ値をそのカテゴリのアイテムにすることができます。具体的には、「project」というタグキーと「Salix」というタグ値を定義して、Neptune リソースが Salix プロジェクトに割り当てられていることを示すことができます。また、タグキーとして `environment=test` や `environment=production` などを使用して、Neptune リソースがテスト用なのか本稼働用なのかを示すこともできます。Neptune リソースに関連付けられているメタデータの追跡が簡単になるように、一貫した一連のタグキーを使用することをお勧めします。

タグを使用して請求書を整理し AWS 、独自のコスト構造を反映します。これを行うには、サインアップして、タグキー値を含む AWS アカウント 請求書を取得します。次に、結合したリソースのコストを見るには、同じタグキー値のリソースに従って請求書情報を整理します。例えば、複数のリソースに特定のアプリケーション名のタグを付け、請求情報を整理することで、複数のサービスを利用しているアプリケーションの合計コストを確認することができます。詳細については、AWS Billing ユーザーガイド の「コスト配分タグの使用」をご参照ください。

各 Neptune リソースにはタグセットがあり、それぞれの Neptune リソースに割り当てられているすべてのタグが含まれています。タグセットには最大 10 個のタグを含めることができ、空にすることもできます。Neptune リソースに追加したタグのキーがそのリソースの既存のタグのキーと同じ場合、既存の値は新しい値によって上書きされます。

AWS はタグに意味論的意味を適用しません。タグは厳密には文字列として解釈されます。リソース作成時に使用する設定によっては、Neptune によって DB インスタンスまたは他のNeptune リソースにタグが設定されることがあります。たとえば、Neptune によって DB インスタンスが本稼働用またはテスト用であることを示すタグが追加されることがあります。

- タグキーは、必須のタグ名です。文字列値は、1〜128 文字の Unicode 文字です。「`aws:`」または「`rds:`」をプレフィックスとして使用することはできません。文字列には、一連の Unicode

文字、数字、空白、「_」、「.」、「/」、「=」、「+」、「-」(Java 正規表現: ^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$) を含めることができます。

- タグ値は、タグの省略可能な文字列値です。文字列値は、1 ～ 256 文字の Unicode 文字です。「aws:」をプレフィックスとして使用することはできません。文字列には、一連の Unicode 文字、数字、空白、「_」、「.」、「/」、「=」、「+」、「-」(Java 正規表現: ^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$) を含めることができます。

  値はタグセット内で一意である必要はなく、null を指定できます。例えば、project/Trinity と cost-center/Trinity のタグセット内に 1 つのキーと値のペアを使用できます。

> **Note**
>
> スナップショットにタグを追加することができます。ただし、このグループは請求書に反映されません。

AWS Management Console、 AWS CLI、または Neptune API を使用して、Neptune リソースのタグを追加、一覧表示、削除できます。 AWS CLI または Neptune API を使用する場合は、使用する Neptune リソースの Amazon リソースネーム (ARN) を指定する必要があります。ARN の作成の詳細については、「Neptune 用 ARN の構築」を参照してください。

タグは承認用にキャッシュに格納されます。そのため、Neptune リソースに対するタグの追加や更新には数分かかることがあります。

Neptune でタグをコピーする

DB インスタンスを作成または復元するとき、DB インスタンスから DB インスタンスのスナップショットにタグがコピーされるように指定できます。タグをコピーすると、DB スナップショットとソース DB インスタンスのメタデータが確実に一致し、また、DB スナップショットとソース DB インスタンスのアクセスポリシーが確実に一致するようになります。タグは、デフォルトではコピーされません。

次のアクションでタグが DB スナップショットにコピーされるように指定できます。

- DB インスタンスの作成
- DB インスタンスの復元
- リードレプリカの作成。

- DB スナップショットのコピー

> **Note**
>
> create-db-cluster-snapshot AWS CLI コマンドの `--tag-key`パラメータの値を含める (または CreateDBClusterSnapshot API アクションに少なくとも 1 つのタグを指定する) 場合、Neptune はソース DB インスタンスから新しい DB スナップショットにタグをコピーしません。これは、ソース DB インスタンスの `--copy-tags-to-snapshot` (`CopyTagsToSnapshot`) オプションが有効になっている場合でも当てはまります。つまり、DB スナップショットから DB インスタンスのコピーを作成でき、新しい DB インスタンスに適用されないタグを追加することがありません。コマンド (または Neptune API アクション) を使用して AWS CLI `create-db-cluster-snapshot` DB `CreateDBClusterSnapshot` スナップショットを作成したら、このトピックで後述するようにタグを追加できます。

を使用した Neptune でのタグ付け AWS Management Console

Amazon Neptune リソースにタグを追加するプロセスはすべてのリソースで同様です。以下の手順では、Neptune DB インスタンスにタグを付加する方法を示します。

DB インスタンスにタグを追加するには

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/neptune/home で Amazon Neptune コンソールを開きます。
2. ナビゲーションペインで、[インスタンス] を選択します。

   > **Note**
   >
   > [インスタンス] ペインで DB インスタンスのリストをフィルタするには、[インスタンスのフィルター] ボックスの横にあるボックスにテキスト文字列を入力します。その文字列を含む DB インスタンスのみが表示されます。

3. タグを付加する DB インスタンスを選択します。
4. [Instance actions] を選択し、[See details] を選択します。
5. 詳細セクションで、下にスクロールし、[タグ] を選択します。
6. [追加] を選択します。[タグの追加] ウィンドウが表示されます。

7. [タグキー] と [値] の値を入力します。
8. 別のタグを追加するには、[別のタグを追加] を選択し、[タブキー] と [値] の値を入力します。

   このステップを必要な回数繰り返します。
9. [追加] を選択します。

DB インスタンスからタグを削除するには

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/neptune/home で Amazon Neptune コンソールを開きます。
2. ナビゲーションペインで、[インスタンス] を選択します。

   > **Note**
   >
   > [インスタンス] ペインで DB インスタンスのリストをフィルタするには、[インスタンスのフィルター] ボックスの横にあるボックスにテキスト文字列を入力します。その文字列を含む DB インスタンスのみが表示されます。
3. タグを付加する DB インスタンスを選択します。
4. [Instance actions] を選択し、[See details] を選択します。
5. 詳細セクションで、下にスクロールし、[タグ] を選択します。
6. 削除するタグを選択します。
7. [削除] を選択し、[タグの削除] ウィンドウで [削除] を選択します。

を使用した Neptune でのタグ付け AWS CLI

AWS CLIを使用して、Neptune で DB インスタンスのタグを追加、一覧表示、または削除できます。

- Neptune リソースに 1 つ以上のタグを追加するには、 AWS CLI コマンド を使用しますadd-tags-to-resource。
- Neptune リソースのタグを一覧表示するには、 AWS CLI コマンドを使用しますlist-tags-for-resource。
- Neptune リソースから 1 つ以上のタグを削除するには、 AWS CLI コマンドを使用しますremove-tags-from-resource。

必要な Amazon リソースネーム (ARN) を作成する方法の詳細については、「Neptune 用 ARN の構築」を参照してください。

Neptune で API を使用してタグを付ける

Neptune API を使用して DB インスタンスのタグを追加、一覧表示、または削除できます。

- Neptune リソースにタグを追加するには、`AddTagsToResource` オペレーションを使用します。
- Neptune リソースに割り当てられているタグを一覧表示するには、`ListTagsForResource` オペレーションを使用します。
- Neptune リソースからタグを削除するには、`RemoveTagsFromResource` オペレーションを使用します。

必要な ARN を作成する方法の詳細については、「Neptune 用 ARN の構築」をご参照ください。

Neptune API を使用して XML を操作する場合、タグでは以下のスキーマを使用します。

```
<Tagging>
    <TagSet>
        <Tag>
            <Key>Project</Key>
            <Value>Trinity</Value>
        </Tag>
        <Tag>
            <Key>User</Key>
            <Value>Jones</Value>
        </Tag>
    </TagSet>
</Tagging>
```

以下の表に示しているのは、使用可能な XML タグとその特性のリストです。`Key` と `Value` では大文字と小文字が区別されます。たとえば、`project=Trinity` と `PROJECT=Trinity` は 2 つの別個のタグです。

タグ付け要素	説明
`TagSet`	タグセットは、Neptune リソースに割り当てられるすべてのタグのコンテナです。リソースごとに割り当て可能なのは 1 つのタグセットのみです。Neptune API によってのみ `TagSet` を操作できます。

タグ付け要素	説明
Tag	タグはユーザー定義のキーと値のペアです。1 ~ 50 個のタグをタグセットに含めることができます。
Key	キーはタグの必須の名前です。文字列値は、1〜128 文字の Unicode 文字です。「rds:」または「aws:」をプレフィックスとして使用することはできません。文字列には、一連の Unicode 文字、数字、空白、「_」、「.」、「/」、「=」、「+」、「-」(Java 正規表現: ^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$) を含めることができます。 キーはタグセットに対して一意である必要があります。たとえば、タグセットでキーが同じで値が異なるキーと値のペアは使用できません。たとえば、project/Trinity や project/Xanadu は使用できません。
値	値はタグの省略可能な値です。文字列値は、1 〜 256 文字の Unicode 文字です。「rds:」または「aws:」をプレフィックスとして使用することはできません。文字列には、一連の Unicode 文字、数字、空白、「_」、「.」、「/」、「=」、「+」、「-」(Java 正規表現: ^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$) を含めることができます。 値はタグセット内で一意である必要はなく、null を指定できます。例えば、project/Trinity と cost-center/Trinity のタグセット内に 1 つのキーと値のペアを使用できます。

Amazon Neptune で ARN を使用する

Amazon Web Services で作成されたリソースは、Amazon リソースネーム (ARN) によってそれぞれ一意に識別されます。特定の Amazon Neptune オペレーションでは、ARN を指定して、Neptune リソースを一意に識別する必要があります。

> **Important**
>
> Amazon Neptune は、を使用する管理アクション用の Amazon RDS ARN の形式を共有します。管理 API リファレンスNeptune 管理 ARN rds にはが含まれる場合と含まれない場合が

あります。neptune-dbNeptune データリソースを識別するデータプレーン ARN については、「データリソースの指定」を参照してください。

トピック

- Neptune 用 ARN の構築
- Amazon Neptune での既存の ARN の取得

Neptune 用 ARN の構築

次の構文を使用して Amazon Neptune リソースの ARN を構築できます。Neptune は Amazon RDS ARN の形式を共有することにご注意ください。

`arn:aws:rds:<region>:<account number>:<resourcetype>:<name>`

次の表に、特定の Neptune リソースの ARN の構築時に使用する形式を示します。

リソースタイプ	ARN 形式
DB インスタンス	arn:aws:rds:*<region>*:*<account>* :db:*<name>* 例: `arn:aws:rds: us-east-2 :123456789012 :db:my-instance-1`
DB クラスター	arn:aws:rds:*<region>*:*<account>* :cluster: *<name>* 例: `arn:aws:rds: us-east-2 :123456789012 :cluster: my-cluster-1`
イベントサブスクリプション	arn:aws:rds:*<region>*:*<account>* :es:*<name>* 例: `arn:aws:rds: us-east-2 :123456789012 :es:my-subscription`

リソースタイプ	ARN 形式
DB パラメータグループ	arn:aws:rds:*<region>*:*<account>* :pg:*<name>* 例: `arn:aws:rds: us-east-2 :123456789012 :pg:my-param-enable-logs`
DB クラスターのパラメータグループ	arn:aws:rds:*<region>*:*<account>* :cluster-pg: *<name>* 例: `arn:aws:rds: us-east-2 :123456789012 :cluster-pg: my-cluster-param-timezone`
DB クラスタースナップショット	arn:aws:rds:*<region>*:*<account>* :cluster-snapshot: *<name>* 例: `arn:aws:rds: us-east-2 :123456789012 :cluster-snapshot: my-snap-20160809`
DB サブネットグループ	arn:aws:rds:*<region>*:*<account>* :subgrp:*<name>* 例: `arn:aws:rds: us-east-2 :123456789012 :subgrp:my-subnet-10`

Amazon Neptune での既存の ARN の取得

Neptune リソースの ARN は、、 AWS Command Line Interface （ AWS CLI ） AWS Management Console、または Neptune API を使用して取得できます。

を使用した既存の ARN の取得 AWS Management Console

コンソールから ARN を取得するには、ARN を取得したいリソースに移動し、リソースの詳細を表示します。たとえば、ナビゲーションパネルの [インスタンス] を選択し、リストから必要なインスタンスを選択して、DB インスタンスの ARN を取得できます。ARN は [インスタンスの詳細] セクションにあります。

を使用した既存の ARN の取得 AWS CLI

を使用して特定の Neptune リソースの ARN AWS CLI を取得するには、そのリソースの `describe` コマンドを使用します。次の表は、各 AWS CLI コマンドと、コマンドで ARN を取得するために使用する ARN プロパティを示しています。

AWS CLI コマンド	ARN プロパティ
describe-event-subscriptions	EventSubscriptionArn
describe-certificates	CertificateArn
describe-db-parameter-groups	DB ParameterGroupArn
describe-db-cluster-parameter-groups	DBClusterParameterGroupArn
describe-db-instances	DBInstanceArn
describe-events	SourceArn
describe-db-subnet-groups	DB SubnetGroupArn
describe-db-clusters	DBClusterArn
describe-db-cluster-snapshots	DB ClusterSnapshotArn

例えば、次の AWS CLI コマンドは DB インスタンスの ARN を取得します。

Example

Linux、OS X、Unix の場合:

```
aws neptune describe-db-instances \
```

```
--db-instance-identifier DBInstanceIdentifier \
--region us-west-2
```

Windows の場合:

```
aws neptune describe-db-instances ^
--db-instance-identifier DBInstanceIdentifier ^
--region us-west-2
```

API を使用した既存の ARN の取得

特定の Neptune リソースの ARN を取得するには、次の API アクションを呼び出し、次に示す ARN のプロパティを使用できます。

Neptune API アクション	ARN プロパティ
DescribeEventサブスクリプション	EventSubscriptionArn
DescribeCertificates	CertificateArn
DescribeDBParameterGroups	DB ParameterGroupArn
DescribeDBClusterParameterグループ	DBClusterParameterGroupArn
DescribeDBInstances	DBInstanceArn
DescribeEvents	SourceArn
DescribeDBSubnetGroups	DB SubnetGroupArn
DescribeDBClusters	DBClusterArn
DescribeDBClusterSnapshots	DB ClusterSnapshotArn

Amazon Neptune DB クラスターのバックアップと復元

このセクションでは、Amazon Neptune DB クラスターをバックアップおよび復元する方法を示します。

トピック

Neptune DB クラスターのバックアップと復元の概要

このセクションでは、Amazon Neptune でのデータのバックアップおよび復元に関する基本情報について説明します。

トピック

Neptune DB クラスターの耐障害性

Neptune DB クラスターは、耐障害性を持つように設計されています。クラスターボリュームは 1 つの AWS リージョン内の複数のアベイラビリティーゾーンにまたがり、各アベイラビリティーゾーンにはクラスターボリュームデータのコピーが含まれます。この機能は、DB クラスターがデータ喪失なしでアベイラビリティーゾーンの障害に耐えることができ、発生するのはサービスの短時間の中断のみであることを意味します。

DB クラスターのプライマリインスタンスが失敗した場合、Neptune は以下のいずれかの方法で、新しいプライマリインスタンスに自動的にフェイルオーバーします。

- 既存の Neptune レプリカを新しいプライマリインスタンスに昇格する
- 新しいプライマリインスタンスを作成する

DB クラスターに 1 つ以上の Neptune レプリカがある場合は、障害発生中に 1 つの Neptune レプリカがプライマリインスタンスに昇格されます。障害イベントによって短い中断が発生し、その間例外によって読み取りと書き込みオペレーションが失敗します。ただし、一般的なサービスの復元時間は 120 秒未満であり、多くの場合 60 秒未満で復元されます。DB クラスターの可用性を高めるために、複数のアベイラビリティーゾーン内で少なくとも 1 つ以上の Neptune レプリカを作成することをお勧めします。

各レプリカに優先度を割り当てることで、Neptune レプリカがプライマリインスタンスに昇格される順序をカスタマイズできます。優先度の範囲は、最も高い 0 から最も低い 15 までです。プライマ

リインスタンスが失敗した場合、Neptune は最も高い優先度の Neptune レプリカを新しいプライマリインスタンスに昇格します。Neptune レプリカの優先度はいつでも変更できます。優先度を変更しても、フェイルオーバーはトリガーされません。

AWS CLI を使用して、DB インスタンスのフェイルオーバー優先順位を次のように設定できます。

```
aws neptune modify-db-instance --db-instance-identifier (the instance ID) --promotion-tier (the failover priority value)
```

複数の Neptune レプリカで同じ優先度を共有できる場合は、昇格階層が発生します。複数の Neptune レプリカで同じ優先度を共有する場合、Neptune は最大サイズのレプリカを昇格します。複数の Neptune レプリカで同じ優先度とサイズを共有する場合、Neptune は同じ昇格階層の任意のレプリカを昇格します。

DB クラスターに Neptune レプリカが含まれていない場合、障害イベントの発生時にプライマリインスタンスが再作成されます。障害イベントによって中断が発生し、その間例外によって読み取りと書き込みオペレーションが失敗します。新しいプライマリインスタンスが再作成されると、サービスが回復します。これは、通常は 10 分未満で行われます。Neptune レプリカのプライマリインスタンスへの昇格は、新しいプライマリインスタンスの作成よりもはるかに短時間で実行されます。

Neptune バックアップ

Neptune は、クラスターボリュームを自動的にバックアップし、バックアップ保持期間中、復元データを保持します。Neptune のバックアップは連続的かつ増分的であるため、バックアップ保持期間内の任意の時点にすばやく復元できます。バックアップデータが書き込まれるときに、データベースサービスのパフォーマンスに影響が出たり、中断が発生したりすることはありません。DB クラスターを作成または変更するときに、バックアップ保持期間 (1 ~ 35 日) を指定できます。

バックアップストレージの使用状況を管理するには、バックアップの保存期間を短縮するか、不要になった古い手動スナップショットを削除するか、またはその両方を行います。コストを管理するには、保存期間を超えて保持されている継続的バックアップと手動スナップショットに使用されているストレージの量をモニタリングできます。バックアップの保存期間を短縮し、不要になった手動スナップショットを削除できます。

バックアップ保持期間を超えたバックアップを保持する場合は、クラスターボリュームの中にデータのスナップショットを作成できます。スナップショットの保存には、Neptune の標準ストレージ料金がかかります。Neptune のストレージ料金の詳細については、Amazon Neptune 料金表を参照してください。

Neptune は、バックアップ保存期間全体にわたって増分リストアデータを保持します。したがって、バックアップ保持期間を超えて保持したいデータのスナップショットを作成するだけで済みます。スナップショットから新しい DB クラスターを作成できます。

> **Important**
>
> DB クラスターを削除すると、自動バックアップはすべて同時に削除され、復旧できません。つまり、最終的な DB スナップショットを手動で作成しない限り、後で DB インスタンスを最終状態に復元することはできません。手動スナップショットは、クラスターを削除したときに削除されません。

> **Note**
>
> - Amazon Neptune DB クラスターの場合、DB クラスターの作成方法に関係なく、デフォルトのバックアップ保持期間は 1 日です。
> - Neptune の自動バックアップを無効にすることはできません。Neptune のバックアップ保持期間は、DB クラスターによって管理されます。

Neptune バックアップストレージの管理に役立つ CloudWatch メトリクス

Amazon CloudWatch `TotalBackupStorageBilled` メトリクス、`SnapshotStorageUsed` メトリクス、および `BackupRetentionPeriodStorageUsed` メトリクスを使用して、次に示すように、Neptune バックアップに使用されているストレージの量を確認およびモニタリングできます。

- `BackupRetentionPeriodStorageUsed` は、継続的バックアップを保存するために現時点までに使用されたバックアップストレージの量 (バイト単位) を示します。この値は、クラスターボリュームのサイズと、保存期間中に行った変更の量によって変わります。ただし、請求のため、この値が保存期間中に累積的なクラスターボリュームのサイズを超えることはありません。たとえば、クラスターの `VolumeBytesUsed` サイズが 107,374,182,400 バイト (100 GiB) で、保持期間が 2 日間である場合、`BackupRetentionPeriodStorageUsed` の最大値は 214,748,364,800 バイト (100 GiB + 100 GiB) になります。
- `SnapshotStorageUsed` は、バックアップ保持期間を超えて手動スナップショットを保存するために使用されたバックアップストレージの量 (バイト単位) を示します。手動スナップショットは、作成タイムスタンプが保持期間以内である間は、スナップショットのバックアップストレージにはカウントされません。すべての自動スナップショットも、スナップショットのバックアッ

プストレージにはカウントされません。各スナップショットのサイズは、そのスナップショットを作成した時点のクラスターボリュームのサイズです。SnapshotStorageUsed 値は、保持するスナップショットの数と、各スナップショットのサイズによって異なります。たとえば、保持期間外のスナップショットが 1 つあり、このスナップショットを作成した時点のクラスターの VolumeBytesUsed サイズが 100 GiB であったとします。この場合、SnapshotStorageUsed の量は 107,374,182,400 バイト (100 GiB) です。

- TotalBackupStorageBilled は、BackupRetentionPeriodStorageUsed と SnapshotStorageUsed の合計からバックアップストレージの空き容量 (1 日のクラスターボリュームのサイズと等しい) を引いた値を示します。バックアップストレージの空き容量は、最新のボリュームサイズと同じです。たとえば、クラスターの VolumeBytesUsed サイズが 100 GiB、保持期間が 2 日間、保持期間外の手動スナップショットが 1 つある場合、TotalBackupStorageBilled は 214,748,364,800 バイト (200 GiB + 100 GiB - 100 GiB) になります。

Neptune クラスターをモニタリングし、レポートを作成するには、CloudWatch コンソールから CloudWatch メトリクスを使用できます。CloudWatch メトリクスを使用する詳しい方法については、「Neptune のモニタリング」と、「Neptune CloudWatch メトリクス」のメトリクスの表を参照してください。

Neptune バックアップからのデータの復元

Neptune が保持するバックアップデータから、または保存した DB クラスターのスナップショットから、新しい Neptune DB クラスターを作成することで、データを回復できます。バックアップデータから作成された DB クラスターの新しいコピーは、バックアップ保持期間内の任意の時点にすばやく復元できます。バックアップ保持期間中の Neptune バックアップが継続的かつ増分的であることは、復元時間を短縮するためにデータのスナップショットを頻繁に作成する必要がないことを意味します。

DB インスタンスの最新の復元可能時刻または最も早い復元可能時刻を判断するには、Neptune コンソールで Latest Restorable Time 値または Earliest Restorable Time 値を探します。DB クラスターの最新の復元可能時間は、DB クラスターを復元できる最も直近の時点であり、通常は現在時間の 5 分以内です。最も早い復元時間は、クラスターボリュームをバックアップ保持期間内でどこまで遡って復元できるかを示します。

DB クラスターの復元が完了したことは、Latest Restorable Time および Earliest Restorable Time の値を確認することでわかります。復元オペレーションが完了するまで、Latest Restorable Time と Earliest Restorable Time の値としては NULL が返され

ます。`Latest Restorable Time` または `Earliest Restorable Time` が NULL を返す場合、バックアップまたは復元オペレーションをリクエストすることはできません。

AWS Management Console を使用して、特定の時点に DB インスタンスを復元するには

1. AWS マネジメントコンソールにサインインして Amazon Neptune コンソール (https://console.aws.amazon.com/neptune/home) を開きます。
2. ナビゲーションペインで、[インスタンス] を選択します。復元する DB クラスターのプライマリインスタンスを選択します。
3. [Instance actions]、[Restore to point in time] の順に選択します。

   [Launch DB Instance] (DB インスタンスの起動) ウィンドウで、[Restore time] (復元時間) の下にある[Custom] (カスタム) を選びます。
4. [Custom] で、復元する日時を指定します。
5. [Settings] (設定) の下の[DB instance identifier] (DB インスタンス識別子) に新しい復元された DB インスタンスの名前を入力します。
6. [DB インスタンスの起動] を選択して、復元された DB インスタンスを起動します。

   指定した名前で新しい DB インスタンスが作成された後、新しい DB クラスターが作成されます。DB クラスター名は、新しい DB インスタンスの名前の後に `-cluster` を付けたものです。たとえば、新しい DB インスタンスの名前が `myrestoreddb` の場合、新しい DB クラスターの名前は `myrestoreddb-cluster` になります。

Neptune のバックアップウィンドウ

自動バックアップは、優先されるバックアップウィンドウ中に毎日行われます。バックアップウィンドウに割り当てられた時間より長い時間がバックアップに必要な場合、ウィンドウが終了した後もバックアップが完了するまでバックアップが継続します。DB インスタンスの週 1 回のメンテナンス時間とバックアップウィンドウは重複できません。

自動バックアップウィンドウ中、バックアッププロセスのスタート時にストレージ I/O が一時中断することがあります (通常は数秒間)。マルチ AZ 配置のバックアップ中は、レイテンシーが数分間高くなることがあります。

バックアップウィンドウは通常、Neptune の基礎になる Amazon RDS コントロールプレーンによって、リージョンごとに 8 時間ブロックからランダムに選択されます。デフォルトのバックアップウィンドウが割り当てられる各リージョンの時間については、『Amazon RDS ユーザーガイド』のバックアップウィンドウセクションをご覧ください。

Neptune での DB クラスタースナップショットの作成

Neptune は DB クラスターのストレージボリュームのスナップショットを作成し、個々のデータベースだけではなく、その DB クラスター全体をバックアップします。DB クラスタースナップショットを作成するときは、どの DB クラスターをバックアップするのかを識別する必要があります。次に、DB クラスタースナップショットに名前を付けて、後でそこから復元できるようにします。DB クラスタースナップショットを作成するためにかかる時間は、データベースのサイズによって異なります。スナップショットにはストレージボリューム全体が含まれています。そのため、ファイル (一時ファイルなど) のサイズもスナップショットの作成にかかる時間に影響します。

DB クラスタースナップショットは、AWS Management Console、AWS CLI、または Neptune API を使用して作成できます。

コンソールを使用して DB クラスタースナップショットを作成します。

DB クラスタースナップショットを作成するには

1. AWS マネジメントコンソールにサインインして Amazon Neptune コンソール (https://console.aws.amazon.com/neptune/home) を開きます。
2. ナビゲーションペインで、[Databases] (データベース) を選択します。
3. DB インスタンスのリストで、DB クラスターのプライマリインスタンスを選択します。
4. [Instance actions] (インスタンスの操作) を選択してから、[Take snapshot] (スナップショットの取得) を選択します。

   [DB スナップショットの取得] ウィンドウが表示されます。
5. [スナップショット名] ボックスに DB クラスタースナップショットの名前を入力します。
6. [Take Snapshot] (スナップショットの取得) を選択します。

DB クラススナップショットの復元

DB クラスターの Amazon Neptune のスナップショットを作成すると、Neptune はクラスターのストレージボリュームのスナップショットを作成し、個々のインスタンスだけではなく、すべてのデータをバックアップします。新しい DB クラスターは、この DB クラススナップショットから復元することで後に作成できます。DB クラスターを復元する場合は、復元の元となる DB クラススナップショットの名前を指定し、復元によって作成される新しい DB クラスターの名前を指定します。

目次

スナップショットから Neptune DB クラスターを復元する際に留意すべき事項

既存の DB クラスターに復元することはできません。

復元プロセスでは常に新しい DB クラスターが作成されるため、すでに存在している DB クラスターには復元できません。

インスタンスは復元されません。

復元によって作成された新しい DB クラスターには、関連付けられたインスタンスがありません。

復元が完了し、新しい DB クラスターが利用できるようになると、必要なインスタンスが明示的に作成されます。これは、Neptune コンソールで、または CreateDBInstance API 使用のいずれかの方法で実行できます。

カスタムパラメータグループは復元されません。

復元によって作成される新しい DB クラスターには、デフォルトの DB パラメータグループが自動的に関連付けられます。

復元が完了し、新しい DB クラスターが利用できるようになった時点で、復元の元となるインスタンスによって使用されているカスタム DB パラメータグループを関連付ける必要があります。これを行うには、Neptune コンソールで Modify コマンドを実行するか、ModifyDBInstance API を使用します。

> **Important**
>
> スナップショットを作成している DB クラスターで使用されているカスタムパラメータグループを保存することをお勧めします。その後、そのスナップショットから復元するときに、復元された DB クラスターに正しいパラメータグループを容易に関連付けることができます。

カスタムセキュリティグループは復元されません。

復元によって作成される新しい DB クラスターには、デフォルトのセキュリティグループが自動的に関連付けられます。

復元が完了し、新しい DB クラスターが利用できるようになった時点で、復元の元となるインスタンスによって使用されているカスタムセキュリティグループを関連付ける必要があります。これを行うには、Neptune コンソールで Modify コマンドを実行するか、ModifyDBInstance API を使用します。

共有され暗号化されたスナップショットから復元することはできません。

共有および暗号化された DB クラスタースナップショットから、DB クラスターを復元することはできません。

代わりに、スナップショットの未共有コピーを作成し、そのコピーから復元できます。

復元された DB クラスターは、以前と同じ量のストレージを使用します。

DB クラスタースナップショットから DB クラスターを復元する場合、新しいクラスターに割り当てられるストレージの量は、その割り当て済みストレージが実際に使用されているかどうかにかかわらず、スナップショットの作成元の DB クラスターに割り当てられたストレージ量と同じです。

つまり、請求対象の「高いウォーターマーク」は変更されません。高いウォーターマークをリセットするには、グラフからデータをエクスポートし、新しい DB クラスターに再ロードする必要があります (「Neptune ストレージ請求」を参照)。

スナップショットからの復元方法

DB クラススナップショットから DB クラスターを復元するには、AWS Management Console、AWS CLI、または Neptune API を使用します。

コンソールを使用してスナップショットから復元する

1. AWS マネジメントコンソールにサインインして Amazon Neptune コンソール (https://console.aws.amazon.com/neptune/home) を開きます。
2. ナビゲーションペインで、[Snapshots] (スナップショット) を選択します。
3. 復元の元にする DB クラススナップショットを選択します。
4. [アクション]、[スナップショットの復元] の順に選択します。
5. [DB インスタンスの復元] ページの [DB インスタンス識別子] テキストボックスに、復元された DB クラスターの名前を入力します。
6. [DB インスタンスの復元] を選択します。
7. スナップショットが作成された DB クラスターの DB クラスター機能を復元する場合は、セキュリティグループを使用するように DB クラスターを変更する必要があります。次のステップは、DB インスタンスが仮想プライベートクラウド (VPC) 内にあることを前提としています。DB インスタンスが VPC 内にない場合は、Amazon EC2 コンソールを使用して、DB クラスターに必要なセキュリティグループを見つけます。

   a. Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。
   b. ナビゲーションペインで、[Security Groups] (セキュリティグループ) をクリックします。
   c. DB クラスターで使用するセキュリティグループを選択します。必要に応じて、EC2 インスタンスのセキュリティグループにセキュリティグループをリンクするためのルールを追加します。

DB クラスタースナップショットのコピー

Neptune を使用すると、自動または手動 DB クラスタースナップショットをコピーできます。スナップショットをコピーすると、そのコピーは手動スナップショットになります。

スナップショットは、同じ AWS リージョン内および複数の AWS リージョン間でコピーできます。

自動スナップショットを別の AWS アカウントにコピーするには、2 段階のプロセスを行います。まず、自動化スナップショットから手動スナップショットを作成してから、その手動スナップショットを他のアカウントにコピーします。

コピーの代わりに、他の AWS アカウントと手動スナップショットを共有することができます。詳細については、「DB クラスターのスナップショットの共有」を参照してください。

トピック

スナップショットのコピーに関する制限

スナップショットをコピーする際の制約は以下のとおりです。

- 中国 (北京) と中国 (寧夏) の間ではスナップショットをコピーできますが、これらの中国リージョンと他の AWS リージョンの間でスナップショットをコピーすることはできません。
- AWS GovCloud (米国東部) と AWSGovCloud (米国西部) の間ではスナップショットをコピーできますが、これらの AWS GovCloud (US) リージョンと他の AWS リージョンの間でスナップショットをコピーすることはできません。
- ターゲットスナップショットが使用可能になる前に出典スナップショットを削除すると、スナップショットはコピーされない場合があります。ターゲットスナップショットのステータスが `AVAILABLE` になったことを確認してから、出典スナップショットを削除してください。
- アカウントあたり 1 つのリージョンに対して、最大 5 つのスナップショットコピーリクエストを実行できます。

- コピー元とコピー先のリージョンおよびデータのコピー量に応じて、リージョン間のスナップショットのコピーは完了するまでに長時間かかることがあります。

  特定のコピー元の AWS リージョンから、クロスリージョンのスナップショットのコピーが大量にリクエストされた場合、 では、進行中のコピーが完了するまで、そのコピー元の AWS リージョンからの新しいクロスリージョンのコピーリクエストをキューに入れる場合があります。そのキューにコピーリクエストが入っている間は、そのリクエストに関する進行状況の情報は表示されません。進行状況の情報は、コピーの開始後にのみ表示されます。

DB クラスターのスナップショットのコピーの保持

Neptune では、自動スナップショットを次のように削除します。

- 保持期間の終了時。
- DB クラスターの自動スナップショットを無効にした場合。
- DB クラスターを削除した場合。

自動スナップショットをもっと長い期間保持したい場合は、そのコピーを手動スナップショットとして作成します。そうすると、削除するまで保持されます。デフォルトのストレージ領域を超える場合、手動スナップショットに Neptune ストレージコストが適用される場合があります。

バックアップストレージコストの詳細については、Neptune 料金表を参照してください。

スナップショットのコピー時の暗号化の処理

AWS KMS 暗号化キーを使用して暗号化されたスナップショットをコピーできます。暗号化されたスナップショットをコピーする場合は、スナップショットのコピーも暗号化する必要があります。元のスナップショットと同じ AWS KMS 暗号化キーを使用してコピーを暗号化できます。または、別の AWS KMS 暗号化キーを指定することもできます。

コピーするときに、暗号化されていない DB クラスタースナップショットを暗号化することはできません。

Amazon Neptune DB クラスタースナップショットの場合、DB クラスタースナップショットを暗号化しないままにし、復元時に AWS KMS 暗号化キーを指定するオプションもあります。復元された DB クラスターは、指定されたキーを使用して暗号化されます。

AWS リージョン間のスナップショットのコピー

> **Note**
>
> この機能は、Neptune エンジンリリース 1.0.2.1 からアクセスできます。

コピー元スナップショットの AWS リージョンとは異なる AWS リージョンにスナップショットをコピーする場合、最初のコピーでは、差分スナップショットをコピーした場合でもフルスナップショットコピーになります。フルスナップショットコピーには、DB インスタンスを復元するために必要なデータやメタデータすべてが含まれます。最初のスナップショットコピーの後、同じ DB インスタンスの増分スナップショットを同じ AWS アカウント内の同じコピー先リージョンにコピーできます。

差分スナップショットには、同じ DB インスタンスの最新のスナップショット以降に変更されたデータのみが含まれます。差分スナップショットのコピーは高速であり、フルスナップショットコピーよりストレージコストが低くなります。AWS リージョン間での増分スナップショットコピーは、暗号化されていないスナップショットと暗号化されたスナップショットの両方がサポートされています。

> **Important**
>
> 共有スナップショットの場合、増分スナップショットのコピーはサポートされていません。共有スナップショットの場合、同じリージョン内であっても、すべてのコピーは完全なスナップショットになります。

コピー元とコピー先の AWS リージョンおよびデータのコピー量に応じて、リージョン間のスナップショットのコピーは完了するまでに長時間かかることがあります。

コンソールを使用した DB クラスタースナップショットのコピー

コピー元のデータベースエンジンが Neptune である場合、スナップショットは DB クラスタースナップショットになります。AWS アカウントごとに、各 AWS リージョンで同時に最大 5 つの DB クラスタースナップショットをコピーできます。暗号化されている DB クラスタースナップショットと暗号化されていない DB クラスタースナップショットのどちらのコピーもサポートされています。

データ転送料金の詳細については、Neptune 料金表を参照してください。

コピーオペレーションが進行中にキャンセルするには、DB クラスタースナップショットが [copying (コピー)] ステータスの間にターゲット DB クラスタースナップショットを削除します。

次の手順は、暗号化されている DB クラスタースナップショットや暗号化されていない DB クラスタースナップショットのコピーに使用できます。

DB クラスタースナップショットをコピーするには

1. AWS マネジメントコンソールにサインインして Amazon Neptune コンソール (https://console.aws.amazon.com/neptune/home) を開きます。
2. ナビゲーションペインで、[Snapshots] (スナップショット) を選択します。
3. コピーする DB クラスタースナップショットのチェックボックスをオンにします。
4. [Actions] を選択してから、[Copy Snapshot] を選択します。[Make Copy of DB Snapshot] ページが表示されます。
5. [New DB Snapshot Identifier (新しい DB スナップショットの識別子)] に、DB クラスタースナップショットのコピーの名前を入力します。
6. スナップショットからスナップショットのコピーにタグと値をコピーするには、[Copy Tags] を選択します。
7. [Enable Encryption] で、次のいずれかのオプションを選択します。
   - DB クラスタースナップショットが暗号化されていなく、コピーを暗号化しない場合、[Disable encryption] を選択します。
   - DB クラスタースナップショットは暗号化されていないが、コピーを暗号化する場合、[Enable encryption] を選択します。この場合、[Master Key] (マスターキー) で DB クラスタースナップショットのコピーを暗号化するために使用する AWS KMS キー識別子を指定します。
   - DB クラスタースナップショットが暗号化されている場合は、[Enable encryption] を選択します。この場合、コピーを暗号化する必要があるため、[はい] がすでに選択されています。[Master Key] (マスターキー) で、DB クラスタースナップショットの暗号化に使用する AWS KMS キー識別子を指定します。
8. [Copy Snapshot] (スナップショットをコピー) を選択します。

AWS CLI による DB クラスタースナップショットのコピー

copy-db-cluster-snapshot AWS CLI のコマンドを使用して、DB スナップショットをコピーできます。

スナップショットのコピー先が新しい AWS リージョンである場合は、その新しいリージョンでコマンドを実行します。

次のパラメータの説明と例を使用して、AWS CLI でスナップショットをコピーするときに使用するパラメータを判断してください。

- `--source-db-cluster-snapshot-identifier` - コピー元の DB スナップショットの識別子。
  - スナップショットのコピー元とコピー先が同じ AWS リージョンである場合は、有効な DB スナップショットの識別子 (`neptune:instance1-snapshot-20130805` など) を指定します。
  - スナップショットのコピー元とコピー先が異なる AWS リージョンである場合は、有効な DB スナップショットの ARN (`arn:aws:neptune:us-west-2:123456789012:snapshot:instance1-snapshot-20130805` など) を指定します。
  - 共有された手動 DB スナップショットからコピーする場合、このパラメータは、共有された DB スナップショットの Amazon リソースネーム (ARN) であることが必要です。
  - 暗号化されたスナップショットをコピーする場合、このパラメータは、コピー元の AWS リージョンの ARN 形式であること、さらに `SourceDBSnapshotIdentifier` パラメータの `PreSignedUrl` と一致することが必要です。
- `--target-db-cluster-snapshot-identifier` – – 暗号化された DB スナップショットの新しいコピーの識別子。
- `--kms-key-id` – – 暗号化された DB スナップショットの AWS KMS キー ID。AWS KMS キー ID は、AWS KMS 暗号化キーの Amazon リソースネーム (ARN)、AWS KMS キー識別子、または AWS KMS 暗号化キーのキーエイリアスです。
  - AWS アカウントから暗号化された DB スナップショットをコピーする場合、このパラメータの値を指定して新しい AWS KMS 暗号化キーでコピーを暗号化できます。このパラメータの値を指定しないと、DB スナップショットのコピーはコピー元の DB スナップショットと同じ AWS KMS キー で暗号化されます。
  - このパラメータを使用して、暗号化されていないスナップショットの暗号化されたコピーを作成することはできません。これを試みると、エラーが発生します。
  - 暗号化されたスナップショットを別の AWS リージョンにコピーする場合、目的地の AWS リージョンに対して AWS KMS キーを特定しなければなりません。AWS KMS 暗号化キーは、AWS 作成元のリージョンに特定されるものであり、AWS リージョンから別の AWS リージョンへ暗号化キーを使用することはできません。
- `--source-region` – – コピー元の DB スナップショットの AWS リージョンの ID。暗号化されたスナップショットを別の AWS リージョンにコピーする場合は、このオプションを指定する必要があります。

- `--region` – – スナップショットをコピーする先の AWS リージョンの ID。暗号化されたスナップショットを別の AWS リージョンにコピーする場合は、このオプションを指定する必要があります。

Example 暗号化されていないソースを同じリージョン内でコピー

次のコードは、新しい名前 mydbsnapshotcopy で、us-east-1 AWS リージョンから us-west-2 リージョンへのスナップショットのコピーを作成します。

Linux、OS X、Unix の場合:

```
aws neptune copy-db-cluster-snapshot \
  --source-db-cluster-snapshot-identifier instance1-snapshot-20130805 \
  --target-db-cluster-snapshot-identifier mydbsnapshotcopy
```

Windows の場合:

```
aws neptune copy-db-cluster-snapshot ^
  --source-db-cluster-snapshot-identifier instance1-snapshot-20130805 ^
  --target-db-cluster-snapshot-identifier mydbsnapshotcopy
```

Example 暗号化されていないソースをリージョン間でコピー

次のコードは、新しい名前 mydbsnapshotcopy で、us-east-1 AWS リージョンから us-west-2 リージョンへのスナップショットのコピーを作成します。us-west-2 リージョンでコマンドを実行します。

Linux、OS X、Unix の場合:

```
aws neptune copy-db-cluster-snapshot \
  --source-db-cluster-snapshot-identifier arn:aws:neptune:us-
east-1:123456789012:snapshot:instance1-snapshot-20130805 \
  --target-db-cluster-snapshot-identifier mydbsnapshotcopy \
  --source-region us-east-1 \
  --region us-west-2
```

Windows の場合:

```
aws neptune copy-db-cluster-snapshot ^
```

```
  --source-db-cluster-snapshot-identifier arn:aws:neptune:us-
east-1:123456789012:snapshot:instance1-snapshot-20130805 ^
  --target-db-cluster-snapshot-identifier mydbsnapshotcopy ^
  --source-region us-east-1 ^
  --region us-west-2
```

Example 暗号化されたソースをリージョン間でコピー

次のコード例では、us-east-1 AWS リージョンから us-west-2 リージョンに暗号化された DB スナップショットをコピーします。us-west-2 リージョンでコマンドを実行します。

Linux、OS X、Unix の場合:

```
aws neptune copy-db-cluster-snapshot \
  --source-db-cluster-snapshot-identifier arn:aws:neptune:us-
west-2:123456789012:snapshot:instance1-snapshot-20161115 \
  --target-db-cluster-snapshot-identifier mydbsnapshotcopy \
  --source-region us-east-1 \
  --region us-west-2
  --kms-key-id my_us_west_2_key
```

Windows の場合:

```
aws neptune copy-db-cluster-snapshot ^
  --source-db-cluster-snapshot-identifier arn:aws:neptune:us-
west-2:123456789012:snapshot:instance1-snapshot-20161115 ^
  --target-db-cluster-snapshot-identifier mydbsnapshotcopy ^
  --source-region us-east-1 ^
  --region us-west-2
  --kms-key-id my-us-west-2-key
```

DB クラスターのスナップショットの共有

Neptune を使用すると、次の方法で手動 DB クラススナップショットを共有できます。

- 手動 DB クラススナップショットを共有すると、暗号化されているかいないかに関係なく、権限のある AWS アカウントがスナップショットをコピーできるようになります。
- 手動 DB クラススナップショットを共有すると、暗号化されているかいないかに関係なく、権限を持つ AWS アカウントが DB クラスターをコピーしてそこから復元するのではなく、スナップショットから DB クラスターを直接復元できるようになります。

> **Note**
>
> 自動 DB クラススナップショットを共有するには、自動化されたスナップショットをコピーしてそのコピーを共有することで、手動 DB クラススナップショットを作成します。

DB クラススナップショットから DB クラスターを復元する方法の詳細については、「スナップショットからの復元方法」を参照してください。

手動スナップショットを最大 20 のその他の AWS アカウントと共有することができます。暗号化されていない手動スナップショットをパブリックとして共有することもできます。これにより、このスナップショットをすべての AWS アカウントが使用できるようになります。スナップショットをパブリックとして共有する場合には、パブリック スナップショットにプライベート情報が含まれないように注意してください。

> **Note**
>
> AWS Command Line Interface (AWS CLI) または Neptune API を使用して共有スナップショットから DB クラスターを復元する際、スナップショット識別子として共有 DB スナップショットの Amazon リソースネーム (ARN) を指定する必要があります。

トピック

- 暗号化された DB クラススナップショットの共有
- DB クラスターのスナップショットの共有

暗号化された DB クラススナップショットの共有

AES-256 暗号化アルゴリズムを使用して暗号化された「保存中」である DB クラススナップショットを共有できます。詳細については、「保管時の Neptune リソースの暗号化」を参照してください。これを行うには、次のステップを実行する必要があります。

1. スナップショットの暗号化に使用された AWS Key Management Service (AWS KMS) 暗号化キーを、スナップショットにアクセスできるようにするすべてのアカウントと共有します。

   AWS KMS 暗号化キーは、KMS キーポリシーに他の AWS アカウントを追加することで、別のアカウントと共有できます。キーポリシーの更新の詳細については、AWS KMS デベロッパーガイドのキーポリシーを参照してください。キーポリシーの作成例については、このトピックで後述する暗号化されているスナップショットのコピーを可能にする IAM ポリシーの作成を参照してください。
2. AWS Management Console、AWS CLI 、または Neptune API を使用して、暗号化されているスナップショットを他のアカウントと共有します。

以下の制限は、暗号化されたスナップショットの共有に適用されます。

- 暗号化されたスナップショットをパブリックとして共有することはできません。
- スナップショットを共有する AWS アカウントのデフォルト AWS KMS 暗号化キーを使って暗号化されたスナップショットを共有することはできません。

AWS KMS 暗号化キーへのアクセス許可

お客様のアカウントから共有された暗号化された DB クラススナップショットを、別の AWS アカウントでコピーするには、お客様のスナップショットを共有したアカウントに、スナップショットを暗号化した KMS キーへのアクセス権限を付与する必要があります。別の AWS アカウントに AWS KMS キーへのアクセスを許可するには、KMS キーのキーポリシーを更新して、共有先の AWS アカウントの ARN を KMS キーポリシーの `Principal` に設定します。次に、`kms:CreateGrant` アクションを許可します。全般的な説明については、AWS Key Management Service デベロッパーガイドの「他のアカウントのユーザーに KMS キーの使用を許可する」を参照してください。

AWS アカウントアクセスを KMS 暗号化キーに付与した後、暗号化されたスナップショットをコピーするには、その AWS アカウントで IAM ユーザーを作成する必要があります (すでに存在していない場合)。KMS のセキュリティ制限により、この目的でのルート AWS アカウント ID の使用は許可されていません。さらに、その AWS アカウントは、IAM ユーザーが KMS キーを使用して暗号

化された DB クラススナップショットをコピーできるようにする IAM ポリシーをその IAM ユーザーにアタッチする必要があります。

次のキーポリシーの例では、ユーザー 111122223333 が KMS 暗号化キーの所有者であり、ユーザー 444455556666 がキーの共有先のアカウントです。この更新されたキーポリシーでは、ユーザー 444455556666 のルート AWS アカウント ID の ARN をポリシーの Principal として含み、kms:CreateGrant アクションを許可することで、AWS アカウントに KMS キーへのアクセス権限を付与しています。

```
{
  "Id=": "key-policy-1",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid=": "Allow use of the key",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::111122223333:user/KeyUser",
        "arn:aws:iam::444455556666:root"
      ]},
      "Action": [
        "kms:CreateGrant",
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
      ],
      "Resource": "*"
    },
    {
      "Sid=": "Allow attachment of persistent resources",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::111122223333:user/KeyUser",
        "arn:aws:iam::444455556666:root"
      ]},
      "Action": [
        "kms:CreateGrant",
        "kms:ListGrants",
        "kms:RevokeGrant"
      ],
      "Resource": "*",
```

```
      "Condition": {"Bool": {"kms:GrantIsForAWSResource": true}}
    }
  ]
}
```

暗号化されているスナップショットのコピーを可能にする IAM ポリシーの作成

外部の AWS アカウントに KMS キーへのアクセス権限がある場合、その AWS アカウントの所有者は、そのアカウントで作成された IAM ユーザーに対してその KMS キーで暗号化された スナップショットのコピーを許可するポリシーを作成できます。

次の例では、AWS アカウント 444455556666 の IAM ユーザーにアタッチできるポリシーを示します。これにより、IAM ユーザーは、us-west-2 リージョンの KMS キー c989c1dd-a3f2-4a5d-8d96-e793d082ab26 で暗号化されている AWS アカウント 111122223333 から共有スナップショットをコピーできます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid=": "AllowUseOfTheKey",
            "Effect": "Allow",
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKey*",
                "kms:DescribeKey",
                "kms:CreateGrant",
                "kms:RetireGrant"
            ],
            "Resource": ["arn:aws:kms:us-west-2:111122223333:key/c989c1dd-
a3f2-4a5d-8d96-e793d082ab26"]
        },
        {
            "Sid=": "AllowAttachmentOfPersistentResources",
            "Effect": "Allow",
            "Action": [
                "kms:CreateGrant",
                "kms:ListGrants",
                "kms:RevokeGrant"
            ],
```

```
            "Resource": ["arn:aws:kms:us-west-2:111122223333:key/c989c1dd-
a3f2-4a5d-8d96-e793d082ab26"],
            "Condition": {
                "Bool": {
                    "kms:GrantIsForAWSResource": true
                }
            }
        }
    ]
}
```

キーポリシーの更新の詳細については、AWS Key Management Service デベロッパーガイドのキーポリシーを参照してください。

DB クラスターのスナップショットの共有

DB クラススナップショットは、AWS Management Console、AWS CLI 、または Neptune API を使用して共有できます。

コンソールを使用した DB クラススナップショットの共有

Neptune コンソールを使用して、手動 DB クラススナップショットを最大 20 の AWS アカウントと共有することができます。また、1 つ以上のアカウントでの手動スナップショットの共有を停止できます。

手動の DB クラススナップショットを共有するには

1. AWS マネジメントコンソールにサインインして Amazon Neptune コンソール (https://console.aws.amazon.com/neptune/home) を開きます。
2. ナビゲーションペインで、[Snapshots] (スナップショット) を選択します。
3. 共有する手動スナップショットを選択します。
4. [アクション]、[スナップショットの共有] の順に選択します。
5. [DB snapshot visibility] で次のいずれかのオプションを選択します。

    - ソースが暗号化されていない場合は、手動の DB クラススナップショットからすべての AWS アカウントに DB クラスターの復元を許可するには、[Public] (公開) を選択します。手動 DB クラススナップショットからの DB クラスターの復元を、指定した AWS アカウントのみに許可するには、[Private] (プライベート) を選択します。

> **Warning**
>
> [DB snapshot visibility] (DB スナップショット可視化) を [Public] (公開) に設定すると、すべての AWS アカウントが手動 DB クラスタースナップショットから DB クラスターを復元し、データへアクセスすることができるようになります。プライベート情報を含む手動 DB クラスタースナップショットは、[Public] として共有しないでください。

- 出典 DB クラスターが暗号化されている場合、暗号化されているスナップショットはパブリックとして共有できないため、[DB snapshot visibility] が [Private] に設定されます。

6. [AWS Account ID] (AWS アカウント ID) で、手動 DB スナップショットからの DB クラスターの復元を許可するアカウントの AWS アカウント ID を入力します。その後、[Add] (追加) を選択します。この操作を繰り返して、AWS アカウント ID を最大 20 AWS アカウント追加できます。

   アクセス権限が付与されたアカウントのリストに AWS アカウント ID を誤って追加した場合には、その AWS アカウント ID の右側にある [Delete] を選択すれば、削除することができます。
7. 手動スナップショットの復元を許可する AWS アカウントにアカウント ID をすべて追加したら、[Save] (保存) を選択します。

AWS アカウントとの手動 DB クラスタースナップショットの共有を停止するには

1. Amazon Neptune コンソール (https://console.aws.amazon.com/neptune/home) を開きます。
2. ナビゲーションペインで、[Snapshots] (スナップショット) を選択します。
3. 共有を停止する手動スナップショットを選択します。
4. [Actions] を選択してから、[Share Snapshot] を選択します。
5. AWS アカウントのアクセス権限を削除するには、アクセス権限が付与されたアカウントのリストからそのアカウントの AWS アカウント ID を選択し、[Delete] を選択します。
6. [Save (保存)] を選択します。

Neptune スナップショットの削除

DB スナップショットは、AWS Management Console、AWS CLI、または 管理 API を使用して削除できます。

コンソールによる削除

1. AWS マネジメントコンソールにサインインして Amazon Neptune コンソール (https://console.aws.amazon.com/neptune/home) を開きます。
2. ナビゲーションペインで、「Snapshots」 を選択します。
3. 削除する DB スナップショットを選択します。
4. [アクション] で、[スナップショットの削除] を選択します。
5. 確認ページで、「削除」 を選択します。

AWS CLI による削除

DB スナップショットは、AWS CLI delete_db_cluster_snapshot コマンドを使用して削除することもできます。この場合、`--db-snapshot-identifier` パラメータを使用して削除対象のスナップショットを特定します。

Linux、OS X、Unix の場合:

```
aws neptune delete-db-cluster-snapshot \
    --db-snapshot-identifier <name-of-the-snapshot-to-delete>
```

Windows の場合:

```
aws neptune delete-db-cluster-snapshot ^
    --db-snapshot-identifier <name-of-the-snapshot-to-delete>
```

Neptune 管理 API による削除

いずれかの SDK を使用して DB スナップショットを削除するには、DeleteDBClusterSnapshot API を呼び出し、`DBSnapshotIdentifier` パラメータで削除対象の DB スナップショットを特定します。

ベストプラクティス: Neptune を最大限に活用する

以下に、Amazon Neptune を使用するための一般的な推奨事項をいくつか示します。Amazon Neptune を使用してパフォーマンスを最大にするための推奨事項をすばやく検索できるリファレンスとしてご使用ください。

目次

- 複数のリードレプリカエンドポイントを Gremlin Java 接続プールに追加する
- 接続制限を回避するためにクライアントを閉じる
- フェイルオーバー後の新しい接続の作成
- maxInProcessPerConnection と maxSimultaneousUsagePerConnection を使用し、値を同じに設定してください。
- 文字列ではなくバイトコードとしてサーバーにクエリを送信する
- クエリによって返される ResultSet またはイテレーターを常に完全に消費する
- 頂点とエッジをバッチで一括追加
- Java 仮想マシンで DNS キャッシュを無効にする
- クエリごとのレベルでタイムアウトを設定する (オプション)
- java.util.concurrent.TimeoutException のトラブルシューティング

- OpenCypher と Bolt を使用した Neptune のベストプラクティス
  - クエリでは双方向のエッジを優先する
  - Neptune は 1 つのトランザクションでの複数の同時クエリをサポートしていません。
  - フェイルオーバー後の新しい接続の作成
  - 存続期間の長いアプリケーションの接続処理
  - の接続処理 AWS Lambda
  - 完了したら、ドライバーオブジェクトを閉じます
  - 読み取りと書き込みには明示的なトランザクションモードを使用してください。
    - 読み取り専用トランザクション
    - 読み取り専用トランザクション
  - 例外の場合の再試行ロジック
  - 1 つの SET 句で複数のプロパティを一度に設定できます。
  - SET 句を使用すると、複数のプロパティを一度に削除できます。
  - パラメータ化されたクエリを使う
  - UNWIND 句では、ネストされたマップの代わりにフラット化されたマップを使用してください。
  - 可変長パス (VLP) 式では、より制限の厳しいノードを左側に配置します。
  - 詳細なリレーションシップ名を使用することにより、ノードラベルのチェックが重複しないようにします。
  - 可能な場合はエッジ・ラベルを指定してください。

- WITH 句はできるだけ使用しないでください。
- 制限付きフィルターはクエリのできるだけ早い段階で配置してください。
- プロパティが存在するかどうかを明示的にチェックしてください。
- 名前付きパスは (必要でない限り) 使用しないでください。
- コレクト (DISTINCT ()) は避けてください。
- すべてのプロパティ値を取得する場合は、個別のプロパティ検索よりもプロパティ関数を使用する方がよいでしょう。
- クエリーの外部で静的計算を実行する。
- 個々のステートメントの代わりに UNWIND を使用するBatch 入力
- ノード/リレーションシップにはカスタム ID を使用することを推奨します。
- ~idクエリでは計算は行わないでください。

- SPARQL を使用した Neptune のベストプラクティス
  - デフォルトですべての名前が付いたグラフのクエリの実行
  - ロード用に名前付きのグラフを指定する
  - クエリで FILTER、FILTER...IN、および VALUES を選択する

Amazon Neptune 基本操作ガイドライン

以下に示しているのは、基本的な運用についてのガイドラインであり、Neptune の使用時にユーザーが従う必要があります。

- Neptune DB インスタンスを理解して、パフォーマンスおよびユースケースの要件に合わせて適切なサイズを設定できるようにします。「Amazon Neptune DB クラスターとインスタンス」を参照してください。
- CPU、メモリの使用状況をモニタリングする。これは、必要なクエリのパフォーマンスを達成するために、より強力な CPU やメモリ容量を持つ DB インスタンスクラスにいつ移行するべきかを知るのに役立ちます。Amazon CloudWatch は、使用パターンが変更されたり、デプロイメントの最大容量に近づいたりすると、通知するように設定できます。これにより、システムのパフォーマンスと可用性を維持するのに役立ちます。詳細については、「インスタンスのモニタリング」と「Neptune のモニタリング」を参照してください。

  Neptune には独自のメモリマネージャーがあるため、CPU 使用率が高い場合でもメモリ使用率は比較的低いのが一般的です。クエリ実行時にメモリ不足の例外に遭遇するのは、空きメモリを増やす必要があることを示す最も良い目安です。

- 自動バックアップを有効にして、都合の良いときにバックアップウィンドウを設定します。
- DB インスタンスのフェイルオーバーをテストすることで、そのプロセスでユースケースにかかる時間を把握します。また、DB インスタンスにアクセスするアプリケーションがフェイルオーバー後に新しい DB インスタンスに自動的に接続できるようにします。
- 可能であれば、クライアントと Neptune クラスターを同じリージョンと VPC で実行します。VPC ピアリングを使用したクロスリージョン接続では、クエリの応答時間に遅延が生じる可能性があるためです。一桁のミリ秒のクエリ応答の場合、クライアントと Neptune クラスターを同じリージョンと VPC に保持する必要があります。
- リードレプリカインスタンスを作成するときは、少なくともプライマリライターインスタンスと同じ大きさにする必要があります。これにより、レプリケーションの遅延がチェックされ、レプリカの再起動が回避されます。「クラスターで異なるインスタンスクラスを避ける」を参照してください。
- 新しいメジャーエンジンバージョンにアップグレードする前に、必ずそのエンジンでアプリケーションをテストしてください。これを行うには、DB クラスターをクローンしてクローンクラスターで新しいエンジンバージョンを実行し、そのクローンでアプリケーションをテストします。
- フェイルオーバーを容易にするために、すべてのインスタンスを同じサイズにするのが理想的です。

トピック

- Amazon Neptune セキュリティベストプラクティス
- クラスターで異なるインスタンスクラスを避ける
- 一括ロード中の再起動の繰り返しの回避
- 多数の述語がある場合は、OSPG インデックスを有効にします。
- 可能な限り、実行時間が長いトランザクションを避ける
- Neptune メトリクスを使用するベストプラクティス
- Neptune クエリのチューニングのベストプラクティス
- リードレプリカ全体の負荷分散
- 一時的に大きなインスタンスを使用してロード時間を短縮
- リードレプリカにフェイルオーバーしてライターインスタンスのサイズを変更する
- データプリフェッチタスク中断エラー後のアップロードの再試行

Amazon Neptune セキュリティベストプラクティス

AWS Identity and Access Management (IAM) アカウントを使用して、Neptune API アクションに対するアクセスを制御します。特に、DB インスタンス、セキュリティグループ、オプショングループ、またはパラメータグループなどの Neptune リソースを作成、変更、削除するアクションが対象になります。DB インスタンスのバックアップや復元などの一般的な管理操作を実行するアクションも対象になります。

- 可能な限り、永続的な認証情報ではなく一時的な認証情報を使用してください。
- Amazon Relational Database Service (Amazon RDS) リソースを管理する各ユーザーにそれぞれの IAM アカウントを割り当てます。AWS アカウントのルートユーザーを使用して Neptune リソースを管理しないでください。お客様を含めて全員に IAM ユーザーを作成します。
- それぞれの職務の実行に最低限必要になる一連のアクセス許可を各ユーザーに付与します。
- IAM グループを使用して、複数のユーザーのアクセス許可を効果的に管理します。
- IAM 認証情報のローテーションを定期的に行います。

タグ付けを使用して Neptune リソースへのアクセスを制限する方法については、Amazon Neptune のセキュリティを参照してください。IAM の使用に関する一般的な情報については、AWS Identity and Access ManagementおよびIAM ユーザーガイドのIAM ベストプラクティスを参照してください。

クラスターで異なるインスタンスクラスを避ける

DB クラスターに異なるクラスのインスタンスが含まれている場合、時間の経過とともに問題が発生する可能性があります。最も一般的な問題は、レプリケーションのラグが原因で、小さなリーダーインスタンスが再起動を繰り返すサイクルに入る可能性があることです。リーダーノードの DB インスタンスクラスの設定が、ライター DB インスタンスの設定よりも弱い場合、変更のボリュームが大きすぎてリーダーが追いつくことができません。

> **Important**
>
> レプリケーションラグによる再起動が繰り返されないようにするには、すべてのインスタンスが同じインスタンスクラス (サイズ) を持つように DB クラスターを構成します。

ライタインスタンス (プライマリ) と DB クラスター内のリーダー間の遅延は、Amazon CloudWatch のメトリクスの `ClusterReplicaLag` メトリクスを使って確認できます。`VolumeWriteIOPs` メ

トリクスでは、レプリケーションラグが発生する可能性のあるクラスター内の書き込みアクティビティのバーストを検出することもできます。

一括ロード中の再起動の繰り返しの回避

一括ロード中のレプリケーション遅延が原因で、リードレプリカが繰り返し再起動されるサイクルが発生した場合、レプリカは DB クラスター内のライターに追いつけない可能性があります。

リーダーをライターよりも大きくするか、一括ロード中にリーダーを一時的に削除し、完了後に再作成してください。

多数の述語がある場合は、OSPG インデックスを有効にします。

データモデルに Distinct 述語が多数含まれていると (たいていは 1000 以上)、パフォーマンスが低下し、運用コストが高くなる可能性があります。

その場合は、OSPG インデックスを有効にしてパフォーマンスを改善できます。「OSGP インデックス」を参照してください。

可能な限り、実行時間が長いトランザクションを避ける

実行時間が長いトランザクション (読み取り専用または読み取り/書き込み) は、次の種類の予期しない問題を引き起こす可能性があります。

読み取りインスタンスまたは同時書き込みがあるライターインスタンスで実行時間が長いトランザクションは、異なるバージョンのデータが大量に蓄積される可能性があります。これにより、結果の大部分を除外する読み取りクエリのレイテンシーが高くなる可能性があります。

場合によっては、時間の経過とともに蓄積されたバージョンによって、新しい書き込みがスロットルされることがあります。

多くの書き込みを伴う実行時間が長い読み取り/書き込みトランザクションは、インスタンスが再起動した場合に問題を引き起こす可能性があります。インスタンスがメンテナンスイベントまたはクラッシュから再起動すると、コミットされていない書き込みはすべてロールバックされます。このような元に戻す操作は通常、バックグラウンドで実行され、インスタンスの復帰をブロックしませんが、ロールバックされる操作と競合する新しい書き込みは失敗します。

たとえば、前回の実行で接続が切断された後に同じクエリが再試行された場合、インスタンスの再起動時に失敗することがあります。

元に戻す操作に必要な時間は、関連する変更のサイズに比例します。

Neptune メトリクスを使用するベストプラクティス

リソース不足やその他の一般的なボトルネックによるパフォーマンスの問題を特定するには、Neptune DB クラスターに適用されるメトリクスをモニタリングできます。

パフォーマンスメトリクスを定期的に監視して、さまざまな時間範囲の平均値、最大値、最小値に関するデータを収集します。これは、いつパフォーマンスが低下しているかを特定するうえで有効です。このデータを使用して、特定のメトリクスしきい値に対して Amazon CloudWatch アラームを設定することにより、しきい値に達した場合に警告されるようにすることができます。

新しい DB クラスターをセットアップし、一般的なワークロードで実行するときは、さまざまな間隔 (1 時間、24 時間、1 週間、2 週間など) でのすべてのパフォーマンスメトリクスの平均値、最大値、最小値を収集します。これにより、正常な状態を把握することができます。それにより、オペレーションのピークおよびオフピークの時間帯を比較して、得られた情報から、いつパフォーマンスが標準レベルを下回っているかを特定し、それに応じてアラームを設定できます。

Neptune メトリクスの表示方法の詳細については、Amazon を使用した Neptune のモニタリング CloudWatchを参照してください。

開始時に最も重要なメトリクスは次のとおりです。

- BufferCacheHitRatio — バッファキャッシュから提供されたリクエストの割合 (パーセント)。キャッシュミスにより、クエリの実行に大きなレイテンシーが追加されます。キャッシュヒット率が 99.9% を下回っており、アプリケーションでレイテンシーが問題になる場合は、より多くのデータをメモリにキャッシュするようにインスタンスタイプをアップグレードすることを検討してください。
- CPU 使用率 - 使用されているコンピュータの処理能力の割合。クエリパフォーマンスの目標によっては、CPU 使用率に大きな値を設定することをお勧めします。
- Freeable memory — DB インスタンスで使用可能な RAM の量 (メガバイト単位)。Neptune には独自のメモリマネージャーがあるため、このメトリクスは予想よりも低くなる可能性があります。インスタンスクラスをより多くの RAM を持つクラスにアップグレードすることを検討する必要があるという良い兆候となるのは、クエリが頻繁にメモリ不足の例外をスローする場合です。

[モニタリング] タブのメトリクスの CPU、メモリ、メトリクスの 75% 地点に赤い線でマークされています。インスタンスのメモリ消費が頻繁にこの限界を超える場合は、ワークロードを確認し、クエリのパフォーマンスを向上させるためにインスタンスをアップグレードすることを検討してください。

Neptune クエリのチューニングのベストプラクティス

Neptune のパフォーマンスを向上させるには、大量のリソースを消費する使用頻度の最も高いクエリをチューニングして、実行コストを下げることをお勧めします。

Gremlin クエリを調整する方法の詳細については、Gremlin クエリヒント および Gremlin クエリのチューニングを参照してください。SPARQL クエリを調整する方法の詳細については、「SPARQL クエリヒント」を参照してください。

リードレプリカ全体の負荷分散

読み込みエンドポイントのラウンドロビンルーティングを実行するには、DNS エントリがポイントするホストを変更します。WebSocket 接続は長期間存続し続けることが多いので、クライアントは新しい接続を作成し、DNS レコードを解決して新しいリードレプリカへの接続を取得する必要があります。

連続するリクエストに対して異なるリードレプリカを取得するには、クライアントが接続するたびに DNS エントリを解決するようにします。このためには、接続を終了し、リーダーエンドポイントに再接続する必要があります。

インスタンスエンドポイントに明示的に接続することで、リードレプリカ全体で負荷を分散することもできます。

一時的に大きなインスタンスを使用してロード時間を短縮

大きなインスタンスサイズで、ロードパフォーマンスが向上します。大きなインスタンスタイプを使用していないが、ロード速度を向上させる場合は、大きいインスタンスを使用してロードし、削除することができます。

> **Note**
>
> 次の手順は新しいクラスターを対象としています。既存のクラスターがある場合は、新しい大きなインスタンスを追加して、プライマリ DB インスタンスに昇格させることができます。

大きいインスタンスサイズを使用してデータをロードするには

1. 単一の `r5.12xlarge` インスタンスでクラスターを作成します。このインスタンスはプライマリ DB インスタンスです。

2. 同じサイズのリードレプリカを 1 つ以上作成します (`r5.12xlarge`)。

   リードレプリカは小さいサイズで作成できますが、プライマリインスタンスによる書き込みに追いつくのに十分な大きさでない場合は、頻繁に再起動する必要があります。その結果、ダウンタイムによってパフォーマンスが大幅に低下します。

3. Bulk Loader コマンドで、`“parallelism” : “OVERSUBSCRIBE”` を含め、Neptune に利用可能なすべての CPU リソースをロードに使用するように指示します (Neptune ローダーのリクエストパラメータ 参照)。ロードオペレーションは I/O が許す限り高速に進み、通常は CPU リソースの 60 ~ 70% を必要とします。

4. Neptune ローダーを使用してデータをロードします。ロードジョブはプライマリ DB インスタンスで実行されます。

5. データのロードが完了したら、追加料金や再起動の問題を繰り返さないように、クラスター内のすべてのインスタンスを同じインスタンスタイプにスケールダウンするようにしてください (異なるインスタンスサイズを避ける参照)。

リードレプリカにフェイルオーバーしてライターインスタンスのサイズを変更する

ライターインスタンスを含む DB クラスター内のインスタンスのサイズを変更する最善の方法は、希望のサイズになるようにリードレプリカインスタンスを作成または変更し、そのリードレプリカに意図的にフェイルオーバーすることです。アプリケーションで見られるダウンタイムは、ライターの IP アドレスを変更するのに必要な時間だけであり、約 3 ~ 5 秒にする必要があります。

現在のライターインスタンスをリードレプリカインスタンスに意図的にフェイルオーバーするために使用する Neptune 管理 API は、FailoverDBCluster です。Gremlin Java クライアントを使用している場合は、ここに述べるように、フェイルオーバー後に新しいクライアントオブジェクトを作成して、新しい IP アドレスを取得しなければならない場合があります。

以下で説明するように、再起動を繰り返すサイクルを回避するために、すべてのインスタンスを同じサイズに変更してください。

データプリフェッチタスク中断エラー後のアップロードの再試行

バルクローダーを使用してデータを Neptune にロードするときに、`LOAD_FAILED` ステータスが発生し、詳細情報のリクエストのレスポンスで、次のような `PARSING_ERROR` および `Data prefetch task interrupted` メッセージが発生することがあります。

```
"errorLogs" : [
  {
    "errorCode" : "PARSING_ERROR",
    "errorMessage" : "Data prefetch task interrupted: Data prefetch task for 11467
 failed",
    "fileName" : "s3://some-source-bucket/some-source-file",
    "recordNum" : 0
  }
]
```

このエラーが発生した場合は、バルクアップロードリクエストを再試行します。

このエラーが発生するのは、通常はリクエストやデータが原因で発生しない一時的な中断があった場合であり、一般的にはバルクアップロードリクエストを再実行することで解決できます。

デフォルト設定 (`"mode":"AUTO"` および `"failOnError":"TRUE"`) を使用している場合、ローダーはすでに正常にロードされたファイルをスキップし、中断が発生したときにまだロードされていなかったファイルのロードを再開します。

Neptune で Gremlin を使用するための一般的なベストプラクティス

Neptune で Gremlin グラフトラバーサル言語を使用する際は次の推奨事項に従います。Neptune での Gremlin 利用の詳細については、the section called “Gremlin”を参照してください。

Important

TinkerPop バージョン 3.4.11 に変更が加えられ、クエリの処理方法の正確性が向上しましたが、現時点ではクエリのパフォーマンスに重大な影響を与える場合があります。
たとえば、この種類のクエリの実行速度が大幅に遅くなる可能性があります。

```
g.V().hasLabel('airport').
  order().
    by(out().count(),desc).
  limit(10).
  out()
```

TinkerPop 3.4.11 の変更により、制限ステップの後の頂点は、最適ではない方法でフェッチされるようになりました。これを回避するには、barrier() ステップを `order().by()` の次の任意のポイントに追加して、クエリを変更できます。例:

```
g.V().hasLabel('airport').
  order().
    by(out().count(),desc).
  limit(10).
  barrier().
  out()
```

TinkerPop 3.4.11 が Neptune エンジンバージョン 1.0.5.0 で有効になりました。

トピック

- Gremlin コードをデプロイするコンテキストでテストする
- DFE エンジンを活用するためのアップサートクエリの構築
- 効率的なマルチスレッドの Gremlin 書き込みの作成
- 作成時刻プロパティを使用したレコードの削除
- Groovy の時刻データに対する datetime( ) メソッドの使用
- GLV 時刻データのネイティブの日付と時刻の使用

Gremlin コードをデプロイするコンテキストでテストする

Gremlin では、クライアントがサーバーにクエリを送信する方法が複数あります。WebSocket または Bytecode GLV を使用するか、文字列ベースのスクリプトを使用して Gremlin コンソールを使用します。

Gremlin クエリの実行は、クエリの送信方法によって異なる可能性があることを認識することが重要です。空の結果を返すクエリは、バイトコードモードで送信された場合は成功したものとして扱われますが、スクリプトモードで送信された場合は失敗として扱われます。例えば、スクリプトモードのクエリに `next()` を含むと、`next()` はサーバに送信されますが、ByteCode を使用してクライアントは通常、`next()` 自体を処理します。最初のケースでは、結果が見つからなければクエリは失敗しますが、2 番目のケースでは、結果セットが空であるかどうかにかかわらず、クエリは成功します。

コードを 1 つのコンテキスト (たとえば、一般的にテキスト形式でクエリを送信する Gremlin コンソール) で開発およびテストし、別のコンテキスト (たとえば、Bytecode を使用して Java ドライバを介して) にコードをデプロイすると、コードの動作が開発環境では発生しなかった問題が本番環境で生じる可能性があります。

> **Important**
>
> 予期せぬ結果を避けるために、デプロイされる GLV コンテキストで Gremlin コードをテストしてください。

DFE エンジンを活用するためのアップサートクエリの構築

Neptune DFE エンジンを可能な限り最大限に活用することで、アップサートクエリのパフォーマンスを大幅に向上させることができます。

Gremlin `mergeV()` および `mergeE()` ステップによる効率的なアップサートの実行 DFE エンジンをできるだけ効果的に使用するようにアップサートクエリを構成する方法について説明します。

効率的なマルチスレッドの Gremlin 書き込みの作成

Gremlin を使用して Neptune にデータをマルチスレッドでロードするためのガイドラインがいくつかあります。

可能な場合は、各スレッドに衝突しないように挿入または変更するための頂点またはエッジのセットを渡します。たとえば、スレッド 1 は 1 ～ 50,000 の ID 範囲を、スレッド 2 は 50,001 ～ 100,000 の ID 範囲を、というように続きます。これにより、`ConcurrentModificationException` が発生する可能性が低くなります。安全を期すために、すべての書き込みに `try/catch` ブロックを付けてください。失敗した場合は、しばらくしてから再試行できます。

一般に、50～100 (頂点またはエッジ) の間のバッチサイズでの書き込みはうまく機能します。各頂点に追加されるプロパティがたくさんある場合は、100 よりも 50 に近い数が適しています。一部の実験は役立ちます。したがって、バッチオペレーションされた書き込みは次のようなものを使用できます。

```
g.addV('test').property(id,'1').as('a').
  addV('test').property(id,'2').
  addE('friend').to('a').
```

これは、バッチオペレーションごとに繰り返し表示されます。

バッチを使用すると、サーバーへの Gremlin ラウンドトリップごとに 1 つの頂点または工ッジを追加するよりもはるかに効率的です。

言語バリアント (GLV) クライアントを使用している場合は、最初にトラバーサルを作成することによってプログラム的にバッチを作成できます。次にそれに追加して、最後にそれを繰り返します。次に例を示します。

```
t.addV('test').property(id,'1').as('a')
t.addV('test').property(id,'2')
t.addE('friend').to('a')
t.iterate()
```

可能であれば Gremlin 言語バリアントクライアントを使用することをお勧めします。ただし、文字列を連結してバッチを構築することでクエリをテキスト文字列として送信するクライアントと同様のことを実行できます。

クエリに基本的な HTTP ではなく Gremlin クライアントライブラリのいずれかを使用している場合、スレッドはすべて同じクライアント、クラスター、または接続プールを共有する必要があります。最高のスループット (接続プールのサイズ、Gremlin クライアントが使用するワーカースレッドの数などの設定) を得るために設定を調整する必要がある場合もあります。

作成時刻プロパティを使用したレコードの削除

頂点のプロパティとして作成時刻を保存し、定期的に削除することで古いレコードを取り除くことができます。

特定の期間データを保存した後にグラフから削除する必要がある場合 (頂点の有効期限)、頂点の作成時にタイムスタンププロパティを保存できます。その後、特定の時間の前に作成されたすべての頂点に対する `drop()` クエリを定期的に発行できます。次に例を示します。

```
g.V().has("timestamp", lt(datetime('2018-10-11')))
```

Groovy の時刻データに対する **`datetime( )`** メソッドの使用

Neptune は、Gremlin Groovy バリアントで送信されるクエリの日付と時刻を指定する `datetime` メソッドを提供します。これには、Gremlin コンソール、HTTP REST API を使用するテキスト文字列、Groovy を使用する他のシリアル化が含まれます。

> **Important**
>
> これは Gremlin クエリをテキスト文字列として送信するメソッドのみに当てはまります。Gremlin 言語バリアントを使用している場合は、その言語のネイティブな日付のクラスと関数を使用する必要があります。詳細については、次のセクション「the section called “ネイティブの日時”」をご覧ください。
> TinkerPop `3.5.2` (Neptune エンジンリリース 1.1.1.0 で導入された) から、`datetime` は TinkerPop に欠かせない要素です。

`datetime` メソッドを使用して日付を保存および比較することができます。

```
g.V('3').property('date',datetime('2001-02-08'))
```

```
g.V().has('date',gt(datetime('2000-01-01')))
```

GLV 時刻データのネイティブの日付と時刻の使用

Gremlin 言語バリアント (GLV) を使用している場合は、Gremlin 時刻データのプログラミング言語によって提供されるネイティブな日時のクラスと関数を使用する必要があります。

公式な TinkerPop Java、Node.js (JavaScript)、Python、または .NET ライブラリはすべて Gremlin 言語バリアントライブラリです。

> **Important**
>
> これは Gremlin 言語バリアント ライブラリ (GLV) のみに当てはまります。テキスト文字列として Gremlin クエリを送信するメソッドを使用している場合、Neptune から提供される `datetime()` メソッドを使用する必要があります。これには、Gremlin コンソール、HTTP REST API を使用するテキスト文字列、Groovy を使用する他のシリアル化が含まれます。詳細については、前のセクション「the section called “`datetime( )`”」を参照してください。

Python

以下に、ID「`3`」を持つ頂点の「`date`」という 1 つのプロパティを作成する Python の例の一部を示します。Python `datetime.now()` メソッドを使用して作成された日付となる値を設定します。

```
import datetime

g.V('3').property('date',datetime.datetime.now()).next()
```

Python を使用して Neptune に接続するための完全な例については、Python を使用して Neptune DB インスタンスに接続するを参照してください。

Node.js (JavaScript)

以下に、ID「3」を持つ頂点の「date」という 1 つのプロパティを作成する JavaScript の例の一部を示します。Node.js Date() コンストラクタを使用して作成された日付となる値を設定します。

```
g.V('3').property('date', new Date()).next()
```

Node.js を使用して Neptune に接続するための完全な例については、Node.js を使用して Neptune DB インスタンスに接続する を参照してください。

Java

以下に、ID「3」を持つ頂点の「date」という 1 つのプロパティを作成する Java の例の一部を示します。Java Date() コンストラクタを使用して作成された日付となる値を設定します。

```
import java.util.date

g.V('3').property('date', new Date()).next();
```

Java を使用して Neptune に接続するための完全な例については、Java クライアントを使用して Neptune DB インスタンスに接続するを参照してください。

.NET (C#)

以下に、ID「3」を持つ頂点の「date」という 1 つのプロパティを作成する C# の例の一部を示します。.NET *DateTime.UtcNow* プロパティを使用して作成された日付となる値を設定します。

```
Using System;

g.V('3').property('date', DateTime.UtcNow).next()
```

C# を使用して Neptune に接続するための完全な例については、.NET を使用して Neptune DB インスタンスに接続するを参照してください。

Neptune で Gremlin Java クライアントを使用するためのベストプラクティス

Apache TinkerPop Java クライアントの互換性のある最新バージョンを使用する

可能な場合は、使用しているエンジンバージョンでサポートされている Apache TinkerPop Gremlin Java クライアントの最新バージョンを常に使用してください。新しいバージョンには、クライアントの安定性、パフォーマンス、使いやすさを向上させる多数のバグ修正が含まれています。

Neptune エンジンのさまざまなバージョンと互換性のあるクライアントバージョンのリスト Apache TinkerPop Java Gremlin クライアント を参照してください。

複数のスレッドにまたがってクライアントオブジェクトを再利用する

複数のスレッド間で同じクライアント (または GraphTraversalSource)オブジェクトを再利用します。つまり、スレッドごとではなく、アプリケーションで `org.apache.tinkerpop.gremlin.driver.Client` クラスの共有インスタンスを作成します。`Client` オブジェクトはスレッドセーフで、初期化にはかなりのオーバーヘッドがかかります。

これは、`GraphTraversalSource` にも当てはまります。この場合、`Client` オブジェクトが内部で作成されます。たとえば、次のコードでは、新しい `Client` オブジェクトがインスタンス化されます。

```
import static
 org.apache.tinkerpop.gremlin.process.traversal.AnonymousTraversalSource.traversal;

  /////

GraphTraversalSource traversal = traversal()
                                  .withRemote(DriverRemoteConnection.using(cluster));
```

読み取りと書き込みエンドポイントに別々の Gremlin Java クライアントオブジェクトを作成する

ライターエンドポイントへの書き込みと 1 つ以上の読み取り専用エンドポイントからの読み込みを実行するだけで、パフォーマンスが向上します。

```
Client readerClient = Cluster.build("https://reader-endpoint")
          ...
          .connect()

Client writerClient = Cluster.build("https://writer-endpoint")
          ...
          .connect()
```

複数のリードレプリカエンドポイントを Gremlin Java 接続プールに追加する

Gremlin Java `Cluster` オブジェクトを作成するとき、`.addContactPoint()` メソッドを使用して複数のリードレプリカインスタンスを接続プールの接点に追加することができます。

```
Cluster.Builder readerBuilder = Cluster.build()
          .port(8182)
          .minConnectionPoolSize(…)
          .maxConnectionPoolSize(…)
          ……
          .addContactPoint("reader-endpoint-1")
          .addContactPoint("reader-endpoint-2")
```

接続制限を回避するためにクライアントを閉じる

接続が終了したらクライアントを閉じて、サーバーによって WebSocket 接続が閉じられ、接続に関連付けられたすべてのリソースが解放されるようにすることが重要です。`client.close( )` は内部で呼び出されるため、`Cluster.close( )` を使用してクラスターを閉じるとリソースは自動的に開放されます。

クライアントが正しく閉じられていない場合、Neptune は 20～25 分後にすべてのアイドル WebSocket接続を終了します。ただし、接続が完了したときに明示的に接続を閉じ WebSocketず、ライブ接続の数が[WebSocket 同時接続制限](#) に達すると、HTTP 429 エラーコードで追加の接続が拒否されます。そうなった場合、Neptune インスタンスを再起動して接続を閉じる必要があります。

`cluster.close()` を呼び出すアドバイスは、Java AWS Lambda 関数には適用されません。詳細については、「[AWS Lambda 関数で Gremlin WebSocket 接続を管理する](#)」を参照してください。

フェイルオーバー後の新しい接続の作成

フェイルオーバーが発生した場合、クラスター DNS 名が IP アドレスに解決されるため、Gremlin Driver は古いライターへの接続を継続する可能性があります。このような場合、フェイルオーバー後に新しい `Client` オブジェクトを作成できます。

`maxInProcessPerConnection` と `maxSimultaneousUsagePerConnection` を使用し、値を同じに設定してください。

パラメータ`maxInProcessPerConnection`と `maxSimultaneousUsagePerConnection`パラメータはどちらも、1 つの WebSocket接続で送信できる同時クエリの最大数に関連しています。これらのパラメータは、内部的に相互に関連しており、もう一方のパラメータなしで一方を変更すると、クライアントがクライアント接続プールから接続を取得しようとしている間にタイムアウトになる可能性があります。

処理中のデフォルト最小値と同時使用値を維持し、`maxInProcessPerConnection` と `maxSimultaneousUsagePerConnection` を同じ値に設定することをお勧めします。

これらのパラメータを設定する値は、クエリの複雑さとデータモデルの関数です。クエリで大量のデータが返るユースケースでは、クエリあたりの接続帯域幅が大きくなるため、パラメータの値を小さくし、`maxConnectionPoolSize` の値を大きくする必要があります。

対照的に、クエリで少量のデータが返るケースでは、`maxInProcessPerConnection` および `maxSimultaneousUsagePerConnection` は、`maxConnectionPoolSize` よりも大きい値に設定する必要があります。

文字列ではなくバイトコードとしてサーバーにクエリを送信する

クエリを送信する際、文字列ではなくバイトコードを使用する利点があります。

- 無効なクエリ構文を早期に検出する: バイトコードのバリアントを使用すると、コンパイル段階で無効なクエリ構文を検出することができます。文字列ベースのバリエーションを使用した場合は、クエリがサーバーに送信され、エラーが返るまで無効な構文を検出することはできません。
- 文字列ベースのパフォーマンスペナルティを避ける: 文字列ベースのクエリを送信すると、WebSockets または HTTP のどちらを使用しているかにかかわらず、頂点がデタッチされます。これは、頂点オブジェクトが ID、ラベル、および頂点に関連付けられたすべてのプロパティで構成されていることを意味します (「要素のプロパティ」を参照)。

プロパティが不要な場合には、これにより、サーバー上で不要な計算が生じる可能性があります。たとえば、顧客がクエリ g.V("hakuna#1") を使用して ID ("hakuna#1) の頂点を取得する場合。クエリが文字列ベースの送信として送信された場合、サーバーではこの頂点の ID、ラベル、およびすべてのプロパティの取得が行われます。クエリがバイトコード送信として送信された場合、サーバーでは頂点の ID とラベルの取得のみ行われます。

つまり、このようなクエリは送信しません。

```
 final Cluster cluster = Cluster.build("localhost")
                                .port(8182)
                                .maxInProcessPerConnection(32)
                                .maxSimultaneousUsagePerConnection(32)
                                .serializer(Serializers.GRAPHBINARY_V1D0)
                                .create();

 try {
     final Client client = cluster.connect();
     List<Result> results =
client.submit("g.V().has('name','pumba').out('friendOf').id()").all().get();
     System.out.println(verticesWithNamePumba);
 } finally {
     cluster.close();
 }
```

代わりに、次のようにバイトコードを使用してクエリを送信します。

```
 final Cluster cluster = Cluster.build("localhost")
                                .port(8182)
                                .maxInProcessPerConnection(32)
                                .maxSimultaneousUsagePerConnection(32)
                                .serializer(Serializers.GRAPHBINARY_V1D0)
                                .create();

 try {
     final GraphTraversalSource g =
traversal().withRemote(DriverRemoteConnection.using(cluster));
     List<Object> verticesWithNamePumba = g.V().has("name",
"pumba").out("friendOf").id().toList();
     System.out.println(verticesWithNamePumba);
 } finally {
     cluster.close();
```

```
}
```

クエリによって返される ResultSet またはイテレーターを常に完全に消費する

クライアントオブジェクトでは常に `ResultSet` (文字列ベースの送信の場合)、または `GraphTraversal` より返るイテレータを完全に消費する必要があります。クエリ結果が完全に消費されない場合、サーバーはそれらを保持し、クライアントがそれらを消費し終わるまで待機します。

アプリケーションが部分的な結果セットしか必要としない場合は、クエリで `limit(X)` ステップを使用して、サーバーで生成される結果の数を制限することができます。

頂点とエッジをバッチで一括追加

Neptune DB へのクエリはすべて、単一のトランザクションの範囲内で実行されます。ただし、セッションを使用する場合は除きます。つまり、gremlin クエリを使用して大量のデータを挿入する必要がある場合は、それらを50〜100 のバッチサイズでまとめてバッチ処理すると、ロード用に作成されるトランザクションの数が減り、パフォーマンスが向上します。

例として、データベースに 5 つの頂点を追加すると、次のようになります。

```
// Create a GraphTraversalSource for the remote connection
final GraphTraversalSource g =
 traversal().withRemote(DriverRemoteConnection.using(cluster));
// Add 5 vertices in a single query
g.addV("Person").property(T.id, "P1")
 .addV("Person").property(T.id, "P2")
 .addV("Person").property(T.id, "P3")
 .addV("Person").property(T.id, "P4")
 .addV("Person").property(T.id, "P5").iterate();
```

Java 仮想マシンで DNS キャッシュを無効にする

複数のリードレプリカにまたがってリクエストをロードバランスする必要がある環境では、Java Virtual Machine (JVM) で DNS キャッシュを無効にし、クラスターを作成するときに Neptune のリーダーエンドポイントを提供する必要があります。JVM DNS キャッシュを無効にすると、すべてのリードレプリカにリクエストが分散されるように、新しい接続ごとに DNS が再度解決されます。これは、アプリケーションの初期化コードで次の行で実行できます。

```
java.security.Security.setProperty("networkaddress.cache.ttl", "0");
```

ただし、ロードバランシングのより完全で堅牢なソリューションは、 の Amazon Gremlin Java クライアントコードによって提供されます GitHub。Amazon Java Gremlin クライアントはクラスタートポロジを認識し、Neptune クラスター内のインスタンスのセット全体に接続とリクエストを公平に分散します。そのクライアントを使用する Java Lambda 関数のサンプルについてはこのブログ投稿を参照してください。

クエリごとのレベルでタイムアウトを設定する (オプション)

Neptune では、パラメータグループオプション neptune_query_timeout を使用して、クエリのタイムアウトを設定することができます (パラメータ 参照)。ただし、Java クライアントのバージョン 3.3.7 以降では、次のようなコードを使用して、グローバルタイムアウトを上書きすることもできます。

```
  final Cluster cluster = Cluster.build("localhost")
                                 .port(8182)
                                 .maxInProcessPerConnection(32)
                                 .maxSimultaneousUsagePerConnection(32)
                                 .serializer(Serializers.GRAPHBINARY_V1D0)
                                 .create();

  try {
      final GraphTraversalSource g =
traversal().withRemote(DriverRemoteConnection.using(cluster));
      List<Object> verticesWithNamePumba = g.with(ARGS_EVAL_TIMEOUT,
500L).V().has("name", "pumba").out("friendOf").id().toList();
      System.out.println(verticesWithNamePumba);
  } finally {
      cluster.close();
  }
```

また、文字列ベースのクエリ送信の場合、コードは次のようになります。

```
  RequestOptions options = RequestOptions.build().timeout(500).create();
  List<Result> result = client.submit("g.V()", options).all().get();
```

> **Note**
>
> 特にサーバーレスインスタンスでは、クエリのタイムアウト値を高く設定しすぎると、予期しないコストが発生する可能性があります。妥当なタイムアウト設定がないと、クエリは予想よりもずっと長く実行され続け、予想もしなかったコストが発生する可能性があります。これは、クエリの実行中に大規模で高価なインスタンスタイプにスケールアップする可能性があるサーバーレスインスタンスに特に当てはまります。
> 予想される実行時間に対応し、異常に長い実行でもタイムアウトが発生するだけのクエリタイムアウト値を使用することで、このような予期しない出費を回避できます。

`java.util.concurrent.TimeoutException` のトラブルシューティング

Gremlin Java クライアントは、Gremlin リクエストがクライアント自体でタイムアウトし、いずれかの WebSocket 接続のスロットが使用可能になるのを待っているjava.util.concurrent.TimeoutExceptionときに をスローします。このタイムアウト期間は、maxWaitForConnection クライアント側の設定可能なパラメータにより制御します。

> **Note**
>
> クライアントでタイムアウトしたリクエストはサーバーに送信されないため、GremlinRequestsPerSec のような、サーバーでキャプチャされたメトリクスには反映されません。

この種のタイムアウトは、通常、次の 2 つの方法のいずれかで発生します。

- サーバが実際に最大容量に達した。この場合、サーバー上のキューがいっぱいになり、MainRequestQueuePendingリクエスト CloudWatch メトリクスをモニタリングすることで検出できます。サーバーが処理できる並列クエリの数は、インスタンスのサイズによって異なります。

  MainRequestQueuePendingRequests メトリクスが、サーバー上の保留中のリクエストのビルドアップを表示しなければ、サーバーはリクエストをさらに処理でき、タイムアウトはクライアント側のスロットリングによって生じます。

- クライアントによるリクエストのスロットル。これは通常、クライアント構成設定を変更することで修正できます。

  クライアントが送信できる並列要求の最大数は、おおむね次のように推定できます。

  ```
  maxParallelQueries = maxConnectionPoolSize * Max( maxSimultaneousUsagePerConnection,
   maxInProcessPerConnection )
  ```

  クライアントに `maxParallelQueries` 以上を送信すると `java.util.concurrent.TimeoutException` 例外の原因になります。通常、いくつかの方法で修正できます。

  - 接続タイムアウト時間を増やす。アプリケーションでレイテンシーが重要でない場合は、クライアントの `maxWaitForConnection` 設定を増やします。その後、クライアントはタイムアウトするまで待機時間が長くなり、代わりにレイテンシーが増加する可能性があります。
  - 接続あたりの最大リクエスト数を増やす。これにより、同じ WebSocket 接続を使用してより多くのリクエストを送信できます。クライアントの `maxSimultaneousUsagePerConnection` および `maxInProcessPerConnection` 設定を増やしてこれを行います。これらの設定には通常、同じ値が設定されています。
  - 接続プール内の接続数を増やす。クライアントの `maxConnectionPoolSize` 設定を増やしてこれを行います。各接続はメモリとオペレーティングシステムのファイル記述子を使用し、初期化時に SSL と WebSocketハンドシェイクを必要とするため、コストはリソースの消費量が増加します。

OpenCypher と Bolt を使用した Neptune のベストプラクティス

Neptune で openCypher クエリ言語と Bolt プロトコルを使用する場合は、これらのベストプラクティスに従ってください。Neptune で openCypher を使用する方法については、openCypher で Neptune グラフにアクセスする を参照してください。

クエリでは双方向のエッジを優先する

Neptune がクエリの最適化を行う場合、双方向のエッジでは、最適なクエリプランを作成することが難しくなります。最適ではないプランでは、エンジンが不必要な作業を行う必要があり、その結果、パフォーマンスが低下します。

そのため、可能な限り、双方向のエッジではなく有向エッジを使用してください。例えば

```
MATCH p=(:airport {code: 'ANC'})-[:route]->(d) RETURN p)
```

ではなく、を使用します。

```
MATCH p=(:airport {code: 'ANC'})-[:route]-(d) RETURN p)
```

ほとんどのデータモデルは実際には両方向のエッジをトラバースする必要はないため、有向エッジを使用するように切り替えることでクエリのパフォーマンスを大幅に向上させることができます。

データモデルで双方向のエッジをトラバースする必要がある場合は、MATCH パターン内の最初のノード (左側) をフィルタリングの制限が最も厳しいノードにします。

「routes 空港と ANC 空港の間のすべてのルートを見つけて」という例を考えてみましょう。ANC 空港から出発する場合、このクエリは次のようになります。

```
MATCH p=(src:airport {code: 'ANC'})-[:route]-(d) RETURN p
```

最も制限の厳しいノードがパターン内の最初のノード (左側) に配置されるため、エンジンは最小限の作業でクエリを満たすことができます。その後、エンジンはクエリを最適化できます。

これは、次のように、パターンの最後で ANC 空港をフィルタリングするよりもはるかに望ましい方法です。

```
MATCH p=(d)-[:route]-(src:airport {code: 'ANC'}) RETURN p
```

最も制限の厳しいノードがパターン内の最初に配置されない場合、エンジンはクエリを最適化できず、結果を得るために追加の検索を実行する必要があるため、追加の作業を行う必要があります。

Neptune は 1 つのトランザクションでの複数の同時クエリをサポートしていません。

Bolt ドライバー自体ではトランザクション内での同時クエリが可能ですが、Neptune は同時に実行されているトランザクション内の複数のクエリをサポートしていません。その代わり、Neptune では、トランザクション内の複数のクエリを順次実行し、各クエリの結果を次のクエリが開始される前に完全に処理する必要があります。

以下の例は、Bolt を使用して 1 つのトランザクションで複数のクエリを連続して実行する方法を示しています。これにより、次のクエリが始まる前にそれぞれの結果が完全に消費されます。

```
final String query = "MATCH (n) RETURN n";

try (Driver driver = getDriver(HOST_BOLT, getDefaultConfig())) {
  try (Session session = driver.session(readSessionConfig)) {
    try (Transaction trx = session.beginTransaction()) {
      final Result res_1 = trx.run(query);
      Assert.assertEquals(10000, res_1.list().size());
      final Result res_2 = trx.run(query);
      Assert.assertEquals(10000, res_2.list().size());
    }
  }
}
```

フェイルオーバー後の新しい接続の作成

フェイルオーバーが発生した場合、DNS 名が特定の IP アドレスに解決されるため、Bolt ドライバーは新しいアクティブなライターインスタンスではなく古いライターインスタンスに接続し続けることがあります。

これを防ぐには、フェイルオーバー後に `Driver` オブジェクトを閉じて、再接続します。

存続期間の長いアプリケーションの接続処理

コンテナ内や Amazon EC2 インスタンスで実行されるアプリケーションなど、長期間有効なアプリケーションを構築する場合は、`Driver` オブジェクトを一度インスタンス化し、そのオブジェクトをアプリケーションの存続期間中再利用します。`Driver` オブジェクトはスレッドセーフで、初期化にはかなりのオーバーヘッドがかかります。

の接続処理 AWS Lambda

接続オーバーヘッドと管理要件のため、Bolt AWS Lambda ドライバーを機能内で使用することはお勧めしません。代わりに HTTPS エンドポイントを使用してください。

完了したら、ドライバーオブジェクトを閉じます

Bolt 接続がサーバーによって閉じられ、その接続に関連付けられているリソースがすべて解放されるように、終了時にはクライアントを閉じることが重要です。`driver.close()` を使用してドライバーを閉じると、この処理が自動的に行われます。

ドライバーが正しく閉じられていない場合、Neptune は 20 分後に、または IAM 認証を使用している場合は 10 日後に、アイドル状態の Bolt 接続をすべて終了します。

Neptune がサポートする Bolt の同時接続数は 1000 個までです。終了後に接続を明示的に閉じず、ライブ接続の数が 1000 という制限数に達すると、新しい接続試行は失敗します。

読み取りと書き込みには明示的なトランザクションモードを使用してください。

Neptune と Bolt ドライバーでトランザクションを使用するときは、読み取りトランザクションと書き込みトランザクションの両方のアクセスモードを適切な設定に明示的に設定するのが最善です。

読み取り専用トランザクション

読み取り専用トランザクションでは、セッションを構築するときに適切なアクセスモード構成を渡さないと、デフォルトの分離レベル、つまりミューテーションクエリ分離が使用されます。そのため、読み取り専用トランザクションでは、アクセスモードを `read` に明示的に設定することが重要です。

自動コミットによる読み取りトランザクションの例:

```
SessionConfig sessionConfig = SessionConfig
  .builder()
  .withFetchSize(1000)
  .withDefaultAccessMode(AccessMode.READ)
  .build();
Session session = driver.session(sessionConfig);
try {
  (Add your application code here)
} catch (final Exception e) {
  throw e;
} finally {
  driver.close()
}
```

読み取りトランザクションの例:

```
Driver driver = GraphDatabase.driver(url, auth, config);
SessionConfig sessionConfig = SessionConfig
  .builder()
  .withDefaultAccessMode(AccessMode.READ)
  .build();
driver.session(sessionConfig).readTransaction(
  new TransactionWork<List<String>>() {
    @Override
    public List<String> execute(org.neo4j.driver.Transaction tx) {
```

```
      (Add your application code here)
    }
  }
);
```

いずれの場合も、SNAPSHOT 分離は Neptune の読み取り専用トランザクションセマンティクスを使用して実現されます。

リードレプリカは読み取り専用クエリしか受け付けないため、リードレプリカに送信されるクエリはすべて SNAPSHOT 分離セマンティクスで実行されます。

読み取り専用トランザクションには、ダーティリードや繰り返し不可能なリードはありません。

読み取り専用トランザクション

ミューテーションクエリでは、書き込みトランザクションを作成するための 3 つの異なるメカニズムがあり、それぞれを以下に示します。

暗黙的な書き込みトランザクションの例:

```
Driver driver = GraphDatabase.driver(url, auth, config);
SessionConfig sessionConfig = SessionConfig
  .builder()
  .withDefaultAccessMode(AccessMode.WRITE)
  .build();
driver.session(sessionConfig).writeTransaction(
  new TransactionWork<List<String>>() {
    @Override
    public List<String> execute(org.neo4j.driver.Transaction tx) {
      (Add your application code here)
    }
  }
);
```

自動コミット書き込みトランザクションの例:

```
SessionConfig sessionConfig = SessionConfig
  .builder()
  .withFetchSize(1000)
  .withDefaultAccessMode(AccessMode.Write)
  .build();
Session session = driver.session(sessionConfig);
try {
```

```
  (Add your application code here)
} catch (final Exception e) {
    throw e;
} finally {
    driver.close()
}
```

明示的な書き込みトランザクションの例:

```
Driver driver = GraphDatabase.driver(url, auth, config);
SessionConfig sessionConfig = SessionConfig
  .builder()
  .withFetchSize(1000)
  .withDefaultAccessMode(AccessMode.WRITE)
  .build();
Transaction beginWriteTransaction = driver.session(sessionConfig).beginTransaction();
  (Add your application code here)
beginWriteTransaction.commit();
driver.close();
```

書き込みトランザクションの分離レベル

- ミューテーションクエリの一部として行われる読み取りは、READ COMMITTED トランザクション分離のもとで実行されます。
- ミューテーションクエリの一部として行われる読み取りにはダーティリードはありません。
- ミューテーションクエリを読み込むと、レコードとレコード範囲はロックされます。
- つまり、インデックスの範囲がミューテーショントランザクションによって読み取られた場合、この範囲は読み取りトランザクションが終了するまで同時トランザクションによって変更されないという強力な保証があります。

ミューテーションクエリはスレッドセーフではありません。

コンフリクトについては、ロック待機タイムアウトを使用した競合の解決 を参照してください。

ミューテーションクエリは、失敗しても自動的に再試行されません。

例外の場合の再試行ロジック

再試行を許可するすべての例外については、ConcurrentModificationException エラーなどの一時的な問題をより適切に処理するために、再試行までの待機時間を徐々に長くするエクスポネン

シャルバックオフおよび再試行戦略を使用するのが一般的には最適です。以下は、エクスポネンシャルバックオフおよび再試行のパターンの例を示しています。

```
public static void main() {
  try (Driver driver = getDriver(HOST_BOLT, getDefaultConfig())) {
    retriableOperation(driver, "CREATE (n {prop:'1'})")
        .withRetries(5)
        .withExponentialBackoff(true)
        .maxWaitTimeInMilliSec(500)
        .call();
  }
}

protected RetryableWrapper retriableOperation(final Driver driver, final String query){
  return new RetryableWrapper<Void>() {
    @Override
    public Void submit() {
      log.info("Performing graph Operation in a retry manner......");
      try (Session session = driver.session(writeSessionConfig)) {
        try (Transaction trx =  session.beginTransaction()) {
            trx.run(query).consume();
            trx.commit();
        }
      }
      return null;
    }

    @Override
    public boolean isRetryable(Exception e) {
      if (isCME(e)) {
        log.debug("Retrying on exception.... {}", e);
        return true;
      }
      return false;
    }

    private boolean isCME(Exception ex) {
      return ex.getMessage().contains("Operation failed due to conflicting concurrent
 operations");
    }
  };
}
```

```
/**
 * Wrapper which can retry on certain condition. Client can retry operation using this
 class.
 */
@Log4j2
@Getter
public abstract class RetryableWrapper<T> {

  private long retries = 5;
  private long maxWaitTimeInSec = 1;
  private boolean exponentialBackoff = true;

  /**
   * Override the method with custom implementation, which will be called in retryable
 block.
   */
  public abstract T submit() throws Exception;

  /**
   * Override with custom logic, on which exception to retry with.
   */
  public abstract boolean isRetryable(final Exception e);

  /**
   * Define the number of retries.
   *
   * @param retries -no of retries.
   */
  public RetryableWrapper<T> withRetries(final long retries) {
    this.retries = retries;
    return this;
  }

  /**
   * Max wait time before making the next call.
   *
   * @param time - max polling interval.
   */
  public RetryableWrapper<T> maxWaitTimeInMilliSec(final long time) {
    this.maxWaitTimeInSec = time;
    return this;
  }
```

```
 /**
  * ExponentialBackoff coefficient.
  */
 public RetryableWrapper<T> withExponentialBackoff(final boolean expo) {
   this.exponentialBackoff = expo;
   return this;
 }

 /**
  * Call client method which is wrapped in submit method.
  */
 public T call() throws Exception {
   int count = 0;
   Exception exceptionForMitigationPurpose = null;
   do {
     final long waitTime = exponentialBackoff ? Math.min(getWaitTimeExp(retries),
maxWaitTimeInSec) : 0;
     try {
         return submit();
     } catch (Exception e) {
       exceptionForMitigationPurpose = e;
       if (isRetryable(e) && count < retries) {
         Thread.sleep(waitTime);
         log.debug("Retrying on exception attempt - {} on exception cause - {}",
count, e.getMessage());
       } else if (!isRetryable(e)) {
         log.error(e.getMessage());
         throw new RuntimeException(e);
       }
     }
   } while (++count < retries);

   throw new IOException(String.format(
         "Retry was unsuccessful.... attempts %d. Hence throwing exception " + "back
to the caller...", count),
         exceptionForMitigationPurpose);
 }

 /*
  * Returns the next wait interval, in milliseconds, using an exponential backoff
  * algorithm.
  */
 private long getWaitTimeExp(final long retryCount) {
```

```
    if (0 == retryCount) {
      return 0;
    }
    return ((long) Math.pow(2, retryCount) * 100L);
  }
}
```

1 つの SET 句で複数のプロパティを一度に設定できます。

複数の SET 句を使用して個々のプロパティを設定する代わりに、マップを使用してエンティティの複数のプロパティを一度に設定します。

次を使用できます。

```
MATCH (n:SomeLabel {`~id`: 'id1'})
SET n += {property1 : 'value1',
property2 : 'value2',
property3 = 'value3'}
```

代わりに:

```
MATCH (n:SomeLabel {`~id`: 'id1'})
SET n.property1 = 'value1'
SET n.property2 = 'value2'
SET n.property3 = 'value3'
```

SET 句は 1 つのプロパティまたはマップのいずれかを受け入れます。1 つのエンティティの複数のプロパティを更新する場合、マップに 1 つの SET 句を使用すると、更新を複数の操作ではなく 1 回の操作で実行できるため、より効率的に実行できます。

SET 句を使用すると、複数のプロパティを一度に削除できます。

OpenCypher 言語を使用する場合、REMOVE を使用してエンティティからプロパティを削除します。Neptune では、削除するプロパティごとに個別の操作が必要になり、クエリの待ち時間が長くなります。代わりに SET をマップと一緒に使用してすべてのプロパティ値をに設定できます。Neptune では`null`、これはプロパティを削除するのと同じです。1 つのエンティティの複数のプロパティを削除する必要がある場合、Neptune のパフォーマンスは向上します。

使用アイテム:

```
WITH {prop1: null, prop2: null, prop3: null} as propertiesToRemove
```

```
MATCH (n)
SET n += propertiesToRemove
```

代わりに:

```
MATCH (n)
REMOVE n.prop1, n.prop2, n.prop3
```

パラメータ化されたクエリを使う

OpenCypher を使用してクエリを実行する場合は、常にパラメータ化されたクエリを使用することをおすすめします。クエリエンジンは、クエリプランキャッシュなどの機能に対して、パラメータ化されたクエリを繰り返し利用できます。同じパラメータ化された構造を異なるパラメータで繰り返し呼び出すと、キャッシュされたプランを利用できます。パラメータ化されたクエリ用に生成されたクエリプランは、100 ミリ秒以内に完了し、パラメータータイプが数値、BOOLEAN、または文字列のいずれかになった場合にのみキャッシュされ、再利用されます。

使用アイテム:

```
MATCH (n:foo) WHERE id(n) = $id RETURN n
```

パラメータ付き:

```
parameters={"id": "first"}
parameters={"id": "second"}
parameters={"id": "third"}
```

代わりに:

```
MATCH (n:foo) WHERE id(n) = "first" RETURN n
MATCH (n:foo) WHERE id(n) = "second" RETURN n
MATCH (n:foo) WHERE id(n) = "third" RETURN n
```

UNWIND 句では、ネストされたマップの代わりにフラット化されたマップを使用してください。

ネストされた構造が深いと、クエリエンジンが最適なクエリプランを生成する機能が制限されることがあります。この問題を部分的に軽減するため、以下の定義済みパターンによって以下のシナリオに最適なプランが作成されます。

- シナリオ 1: 数値、文字列、ブール値を含む暗号リテラルのリストを見て安心してください。
- シナリオ 2: 暗号リテラル (数値、文字列、BOOLEAN) のみを値として含むフラット化されたマップのリストを見て安心してください。

UNWIND 句を含むクエリを作成する場合は、上記の推奨事項を使用してパフォーマンスを向上させてください。

シナリオ 1 の例:

```
UNWIND $ids as x
MATCH(t:ticket {`~id`: x})
```

パラメータ付き:

```
parameters={
  "ids": [1, 2, 3]
}
```

シナリオ 2 の例は、CREATE または MERGE するノードのリストを生成することです。複数のステートメントを発行する代わりに、以下のパターンを使用してプロパティをフラット化されたマップのセットとして定義します。

```
UNWIND $props as p
CREATE(t:ticket {title: p.title, severity:p.severity})
```

パラメーター付き:

```
parameters={
  "props": [
    {"title": "food poisoning", "severity": "2"},
    {"title": "Simone is in office", "severity": "3"}
  ]
}
```

次のようなネストされたノードオブジェクトの代わりに:

```
UNWIND $nodes as n
CREATE(t:ticket n.properties)
```

パラメータ付き:

```
parameters={
  "nodes": [
    {"id": "ticket1", "properties": {"title": "food poisoning", "severity": "2"}},
    {"id": "ticket2", "properties": {"title": "Simone is in office", "severity": "3"}}
  ]
}
```

可変長パス (VLP) 式では、より制限の厳しいノードを左側に配置します。

可変長パス (VLP) クエリでは、クエリエンジンは式の左側または右側から探索を開始するように選択することで評価を最適化します。この決定は、左側と右側のパターンのカーディナリティに基づいて行われます。カーディナリティは、指定されたパターンに一致するノードの数です。

- 右側のパターンのカーディナリティが 1 の場合、右側が開始点になります。
- 左側と右側のカーディナリティが 1 の場合、拡張は両側で確認され、拡張の小さい側から開始されます。拡張は、VLP エクスプレッションの左側のノードと右側のノードの発信エッジまたは入力エッジの数です。最適化のこの部分は、VLP リレーションシップが単方向で、リレーションシップタイプが指定されている場合にのみ使用されます。
- それ以外の場合は、左側が開始点になります。

VLP エクスプレッションチェーンでは、この最適化は最初のエクスプレッションにのみ適用できます。他の VLP は左側から評価されます。例として、(a) と (b) のカーディナリティが 1 で、(c) のカーディナリティが 1 より大きいとします。

- `(a)-[*1..]->(c)`: 評価は (a) から始まります。
- `(c)-[*1..]->(a)`: 評価は (a) から始まります。
- `(a)-[*1..]-(c)`: 評価は (a) から始まります。
- `(c)-[*1..]-(a)`: 評価は (a) から始まります。

ここで、(a) の入ってくるエッジを 2 つ、(a) の外向きのエッジを 3、(b) の入ってくるエッジを 4、(b) の外向きのエッジを 5 とします。

- `(a)-[*1..]->(b)`: (a) の出射エッジが (b) の入射エッジよりも小さいため、評価は (a) から始まります。

- (a)<-[*1..]-(b): (a) の入力エッジが (b) の出射エッジよりも小さいため、評価は (a) から始まります。

原則として、より制限の厳しいパターンは VLP エクスプレッションの左側に配置します。

詳細なリレーションシップ名を使用することにより、ノードラベルのチェックが重複しないようにします。

パフォーマンスを最適化する場合、ノードパターン専用のリレーションシップラベルを使用すると、ノード上のラベルフィルタリングを解除できます。リレーションシップが 2 likes person つのノード間のリレーションシップを定義するためだけに使用されるグラフモデルを考えてみましょう。このパターンを見つけるには、次のクエリを書くことができます。

```
MATCH (n:person)-[:likes]->(m:person)
RETURN n, m
```

n と m person のラベルチェックは重複しています。なぜなら、両方が同じタイプの場合にのみ関係が表示されるように定義したからです。personパフォーマンスを最適化するために、クエリを次のように記述できます。

```
MATCH (n)-[:likes]->(m)
RETURN n, m
```

このパターンは、プロパティが 1 つのノードラベル専用である場合にも適用できます。personノードだけがプロパティを持っていると仮定するとemail、personノードラベルが一致することを確認するのは冗長です。このクエリを次のように記述します。

```
MATCH (n:person)
WHERE n.email = 'xxx@gmail.com'
RETURN n
```

このクエリを次のように記述するよりも効率が悪くなります。

```
MATCH (n)
WHERE n.email = 'xxx@gmail.com'
RETURN n
```

このパターンを採用するのは、パフォーマンスが重要で、モデリングプロセスでエッジラベルが他のノードラベルに関係するパターンに再利用されないようにチェックする必要がある場合だけにしてください。email後でなどの別のノードラベルにプロパティを導入した場合company、これら 2 つのクエリのバージョンでは結果が異なることになります。

可能な場合はエッジ・ラベルを指定してください。

パターン内のエッジを指定するときは、可能な限りエッジラベルを付けることを推奨します。次のクエリの例を考えてみましょう。このクエリは、ある都市に住むすべての人々と、その都市を訪れたすべての人々をリンクさせるために使用されます。

```
MATCH (person)-->(city {country: "US"})-->(anotherPerson)
RETURN person, anotherPerson
```

エンドラベルを指定しないことで、複数のエッジラベルを使用して人々を都市以外のノードにリンクするグラフモデルでは、Neptune eは後で破棄される追加のパスを評価する必要があります。上記のクエリでは、エッジラベルが指定されていないため、エンジンは最初に多くの作業を行い、次に値をフィルタリングして正しい結果を得ます。上記のクエリのより良いバージョンは、以下のようになります。

```
MATCH (person)-[:livesIn]->(city {country: "US"})-[:visitedBy]->(anotherPerson)
RETURN person, anotherPerson
```

これは評価に役立つだけでなく、クエリプランナーがより良いプランを作成できるようにします。このベストプラクティスを冗長ノードラベルチェックと組み合わせて、都市ラベルチェックを削除し、クエリを次のように記述することもできます。

```
MATCH (person)-[:livesIn]->({country: "US"})-[:visitedBy]->(anotherPerson)
RETURN person, anotherPerson
```

WITH 句はできるだけ使用しないでください。

OpenCypher の WITH 句は、実行前のすべてが実行される境界の役割を果たし、その結果得られた値がクエリの残りの部分に渡されます。WITH 句は、暫定的な集計が必要な場合や結果の数を制限したい場合に必要ですが、それ以外は WITH 句は使用しないようにしてください。一般的な指針は、このような単純な WITH 句 (集約、順序、制限なし) を削除して、クエリプランナーがクエリ全体を処理してグローバルに最適なプランを作成できるようにすることです。例として、以下の地域に住むすべてのユーザーを返すクエリを作成したとします。India

```
MATCH (person)-[:lives_in]->(city)
WITH person, city
MATCH (city)-[:part_of]->(country {name: 'India'})
RETURN collect(person) AS result
```

上記のバージョンでは、WITH 句は以前のパターンの配置を制限しています (city)-[:part_of]->(country {name: 'India'}) (より制限が厳しい)。(person)-[:lives_in]->(city)そのため、プランは最適とは言えません。このクエリの最適化は、WITH 句を削除して、プランナーが最適なプランを計算できるようにすることです。

```
MATCH (person)-[:lives_in]->(city)
MATCH (city)-[:part_of]->(country {name: 'India'})
RETURN collect(person) AS result
```

制限付きフィルターはクエリのできるだけ早い段階で配置してください。

どのシナリオでも、クエリの早い段階でフィルターを配置しておくと、クエリプランで検討しなければならない中間ソリューションを減らすのに役立ちます。つまり、クエリの実行に必要なメモリやコンピューティングリソースも少なくて済みます。

次の例は、これらの影響を理解するのに役立ちます。Indiaに住むすべてのユーザーを返すクエリを作成するとします。クエリの例としては、次のようなものがあります。

```
MATCH (n)-[:lives_in]->(city)-[:part_of]->(country)
WITH country, collect(n.firstName + " "  + n.lastName) AS result
WHERE country.name = 'India'
RETURN result
```

上記のバージョンのクエリは、このユースケースを実現するための最適な方法ではありません。country.name = 'India'フィルターはクエリパターンの後半に表示されます。まず、すべての人とその居住地を収集して国別にグループ化し、次にそのグループだけを絞り込みますcountry.name = India。India居住している人だけを検索し、収集集計を行う最適な方法です。

```
MATCH (n)-[:lives_in]->(city)-[:part_of]->(country)
WHERE country.name = 'India'
RETURN collect(n.firstName + " "  + n.lastName) AS result
```

一般的なルールは、変数が導入されたらできるだけ早くフィルターを設定することです。

プロパティが存在するかどうかを明示的にチェックしてください。

OpenCypher のセマンティクスに基づくと、プロパティにアクセスするとオプションの結合と同等になり、プロパティが存在しない場合でもすべての行を保持する必要があります。グラフスキーマに基づいて、そのエンティティには特定のプロパティが常に存在することがわかっている場合、そのプロパティが存在するかどうかを明示的に確認することで、クエリエンジンは最適なプランを作成し、パフォーマンスを向上させることができます。

personあるタイプのノードには必ずプロパティがあるグラフモデルを考えてみましょうname。これを行う代わりに:

```
MATCH (n:person)
RETURN n.name
```

IS NOT NULL チェックを使用して、クエリ内のプロパティの存在を明示的に確認します。

```
MATCH (n:person)
WHERE n.name IS NOT NULL
RETURN n.name
```

名前付きパスは (必要でない限り) 使用しないでください。

クエリ内の名前付きパスには常に追加コストがかかるため、レイテンシーやメモリ使用量の増加という点でペナルティが加わる可能性があります。次のクエリについて考えます。

```
MATCH p = (n)-[:commentedOn]->(m)
WITH p, m, n, n.score + m.score as total
WHERE total > 100
MATCH (m)-[:commentedON]->(o)
WITH p, m, n, distinct(o) as o1
RETURN p, m.name, n.name, o1.name
```

上記のクエリでは、ノードのプロパティだけを知りたいと仮定すると、パス「p」を使用する必要はありません。名前付きパスを変数として指定すると、DISTINCT を使った集計操作は、時間とメモリ使用量の両面でコストがかかります。上記のクエリのより最適化されたバージョンは、以下のようになります。

```
MATCH (n)-[:commentedOn]->(m)
```

```
WITH m, n, n.score + m.score as total
WHERE total > 100
MATCH (m)-[:commentedON]->(o)
WITH m, n, distinct(o) as o1
RETURN m.name, n.name, o1.name
```

コレクト (DISTINCT ()) は避けてください。

COLLECT (DISTINCT ()) は、異なる値を含むリストを作成する場合には必ず使用されます。COLLECT は集計関数で、同じステートメントに追加されるキーに基づいてグループ化されます。distinct を使用すると、入力は複数のチャンクに分割され、各チャンクは削減対象の 1 つのグループを表します。グループの数が増えると、パフォーマンスに影響が出ます。Neptune では、リストを実際に収集/形成する前に DISTINCT を実行するほうがはるかに効率的です。これにより、チャンク全体のグループ化キーで直接グループ化を行うことができます。

次のクエリについて考えます。

```
MATCH (n:Person)-[:commented_on]->(p:Post)
WITH n, collect(distinct(p.post_id)) as post_list
RETURN n, post_list
```

このクエリのより最適な記述方法は以下のとおりです。

```
MATCH (n:Person)-[:commented_on]->(p:Post)
WITH DISTINCT n, p.post_id as postId
WITH n, collect(postId) as post_list
RETURN n, post_list
```

すべてのプロパティ値を取得する場合は、個別のプロパティ検索よりもプロパティ関数を使用する方がよいでしょう。

properties()この関数はエンティティのすべてのプロパティを含むマップを返すために使用され、プロパティを個別に返すよりもはるかに効率的です。

Personノードに 5 つのプロパティ (、、firstNamelastNameage、dept) が含まれていると仮定するとcompany、次のクエリが推奨されます。

```
MATCH (n:Person)
WHERE n.dept = 'AWS'
```

```
RETURN properties(n) as personDetails
```

以下を使用する代わりに:

```
MATCH (n:Person)
WHERE n.dept = 'AWS'
RETURN n.firstName, n.lastName, n.age, n.dept, n.company

=== OR ===

MATCH (n:Person)
WHERE n.dept = 'AWS'
RETURN {firstName: n.firstName, lastName: n.lastName, age: n.age,
department: n.dept, company: n.company} as personDetails
```

クエリーの外部で静的計算を実行する。

静的計算 (単純な数学/文字列演算) はクライアント側で解決することが推奨されます。筆者より 1 歳年上かそれ以下の人をすべて検索する例を考えてみましょう。

```
MATCH (m:Message)-[:HAS_CREATOR]->(p:person)
WHERE p.age <= ($age + 1)
RETURN m
```

ここでは$age、パラメータを使ってクエリに注入し、固定値に加算します。次に、p.ageこの値がと比較されます。代わりに、クライアント側で加算を行い、計算された値をパラメーター $ageplusone として渡す方がよいでしょう。これにより、クエリエンジンが最適なプランを作成できるようになり、各入力行の静的な計算を回避できます。これらのガイドラインに従うと、クエリのより効率的なバージョンは次のようになります。

```
MATCH (m:Message)-[:HAS_CREATOR]->(p:person)
WHERE p.age <= $ageplusone
RETURN m
```

個々のステートメントの代わりに UNWIND を使用するBatch 入力

同じクエリを異なる入力に対して実行する必要があるときは、入力ごとに 1 つのクエリを実行するよりも、複数の入力に対してクエリを実行するほうがはるかに効率的です。

複数のノードをマージする場合、1 つの選択肢は、入力ごとにマージクエリを実行することです。

```
MERGE (n:Person {`~id`: $id})
SET n.name = $name, n.age = $age, n.employer = $employer
```

パラメータ付き:

```
params = {id: '1', name: 'john', age: 25, employer: 'Amazon'}
```

上記のクエリは、すべての入力に対して実行する必要があります。この方法は有効ですが、大量の入力に対して多数のクエリを実行する必要がある場合があります。このシナリオでは、バッチ処理によってサーバーで実行されるクエリの数が減り、全体的なスループットが向上する可能性があります。

次のパターンを使用します。

```
UNWIND $persons as person
MERGE (n:Person {`~id`: person.id})
SET n += person
```

パラメータの場合:

```
params = {persons: [{id: '1', name: 'john', age: 25, employer: 'Amazon'},
{id: '2', name: 'jack', age: 28, employer: 'Amazon'},
{id: '3', name: 'alice', age: 24, employer: 'Amazon'}...]}
```

ワークロードに最適なバッチサイズを決定するには、さまざまなバッチサイズを試してみることをお勧めします。

ノード/リレーションシップにはカスタム ID を使用することを推奨します。

Neptune では、ユーザーがノードとリレーションシップに ID を明示的に割り当てることができます。ID はデータセット内でグローバルに一意で、かつ確定的でなければ有効ではありません。決定的 ID は、プロパティと同様にルックアップやフィルタリングメカニズムとして使用できますが、ID を使用する方がプロパティを使用するよりもクエリ実行の観点からはるかに最適化されます。カスタム ID を使用する利点はいくつかあります。

- 既存のエンティティのプロパティは NULL でもかまいませんが、ID は存在している必要があります。これにより、クエリエンジンは実行時に最適化された結合を使用できます。

- ミューテーションクエリを並行実行すると、ID を使用してノードにアクセスするときに ID を使用する場合に同時変更例外 (CME) が発生する可能性が大幅に低下します。これは、ID が適用される一意性により、プロパティよりもロックの数が少なくなるためです。
- Neptune はプロパティとは異なり ID に一意性を強制するため、ID を使用すると重複データが作成される可能性を回避できます。

次のクエリ例ではカスタム ID を使用しています。

> **Note**
>
> `~id`プロパティは ID を指定するのに使用されますが`id`、他のプロパティと同様に格納されます。

```
CREATE (n:Person {`~id`: '1', name: 'alice'})
```

カスタム ID を使用しない場合:

```
CREATE (n:Person {id: '1', name: 'alice'})
```

後者のメカニズムを使用する場合、一意性の強制は行われず、後でクエリを実行できます。

```
CREATE (n:Person {id: '1', name: 'john'})
```

これにより、`id=1`という名前の付いた 2 つ目のノードが作成されます。`john`このシナリオでは、(alice と john) の名前がそれぞれ異なる 2 つのノードができあがります。`id=1`

`~id`クエリでは計算は行わないでください。

クエリでカスタム ID を使用するときは、常にクエリの外部で静的な計算を実行し、これらの値をパラメータに指定してください。静的な値を指定すると、エンジンはルックアップをより最適化でき、これらの値のスキャンやフィルタリングを回避できます。

データベースに存在するノード間にエッジを作成したい場合、次のような選択肢があります。

```
UNWIND $sections as section
MATCH (s:Section {`~id`: 'Sec-' + section.id})
```

```
MERGE (s)-[:IS_PART_OF]->(g:Group {`~id`: 'g1'})
```

パラメータ付き:

```
parameters={sections: [{id: '1'}, {id: '2'}]}
```

上のクエリでは、idセクションの部分がクエリ内で計算されています。計算は動的であるため、エンジンは ID を静的にインライン化できず、最終的にすべてのセクションノードをスキャンすることになります。その後、エンジンは必要なノードに対して事後フィルタリングを実行します。データベースにセクションノードが多数ある場合、これにはコストがかかる可能性があります。

これを実現するより良い方法は、データベースに渡される ID Sec- の先頭に付加しておくことです。

```
UNWIND $sections as section
MATCH (s:Section {`~id`: section.id})
MERGE (s)-[:IS_PART_OF]->(g:Group {`~id`: 'g1'})
```

パラメータ付き:

```
parameters={sections: [{id: 'Sec-1'}, {id: 'Sec-2'}]}
```

SPARQL を使用した Neptune のベストプラクティス

Neptune で SPARQL クエリ言語を使用する場合は、これらのベストプラクティスに従ってください。Neptune で SPARQL を使用する方法については、SPARQL を使用した Neptune グラフへのアクセスを参照してください。

デフォルトですべての名前が付いたグラフのクエリの実行

Amazon Neptune はすべてのトリプルを名前が付いたグラフに関連付けます。デフォルトグラフはすべての名前が付いたグラフの総合として定義されます。

GRAPH キーワードや構成 (FROM NAMED など) を使用してグラフを明示的に指定せずに SPARQL クエリを送信すると、Neptune は常に DB インスタンスのすべてのトリプルを考慮します。たとえば、次のクエリはすべてのトリプルを Neptune SPARQL エンドポイントから返します。

```
SELECT * WHERE { ?s ?p ?o }
```

1 つ以上のグラフに表されるトリプルは、1 度だけ返されます。

デフォルトグラフ仕様についての詳細は、SPARQL 1.1 クエリ言語仕様の「RDF データセット」セクションを参照してください。

ロード用に名前付きのグラフを指定する

Amazon Neptune はすべてのトリプルを名前が付いたグラフに関連付けます。トリプルのロード、挿入あるいは更新時に名前が付いたグラフを指定しない場合、Neptune は URI `http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph` が定義するフォールバックの名前付きグラフを使用します。

Neptune バルクローダーを使用する場合、`parserConfiguration: namedGraphUri` パラメータを使用して、すべてのトリプル (または 4 番目の位置に空白がある四角形) を使用するように名前付きのグラフを指定できます。Neptune ローダー `Load` コマンド構文についての詳細は、the section called “ローダーコマンド”を参照してください。

クエリで FILTER、FILTER...IN、および VALUES を選択する

SPARQL クエリに値を挿入するには、`FILTER`、`FILTER...IN`、`VALUES` の 3 つの基本的な方法があります。

たとえば、単一のクエリで複数の人から友人を検索するとします。`FILTER` を使用して、クエリを次のように構築します。

```
PREFIX ex: <https://www.example.com/>
PREFIX foaf : <http://xmlns.com/foaf/0.1/>

SELECT ?s ?o
WHERE {?s foaf:knows ?o. FILTER (?s = ex:person1 || ?s = ex:person2)}
```

これにより、`ex:person1` または `ex:person2` への `?s` バウンドがあり、`foaf:knows` というラベル付きの出ていく辺があるグラフにあるすべてのトリプルを返します。

また、同等の結果を返す `FILTER...IN` を使用してクエリを作成することができます。

```
PREFIX ex: <https://www.example.com/>
PREFIX foaf : <http://xmlns.com/foaf/0.1/>

SELECT ?s ?o
```

```
  WHERE {?s foaf:knows ?o. FILTER (?s IN (ex:person1, ex:person2))}
```

この場合は同等の結果を返す VALUES を使用してクエリを作成することもできます。

```
  PREFIX ex: <https://www.example.com/>
  PREFIX foaf : <http://xmlns.com/foaf/0.1/>

  SELECT ?s ?o
  WHERE {?s foaf:knows ?o. VALUES ?s {ex:person1 ex:person2}}
```

多くの場合、これらのクエリは意味的に同等ですが、2 つの FILTER バリアントが VALUES バリエーションと異なる場合があります。

- 最初のケースは、同じ個人を 2 回挿入するなど、重複した値を挿入する場合です。この場合、VALUES クエリには、結果の重複内容が含まれています。このような重複は、SELECT 句に DISTINCT を追加して、明示的に排除することができます。しかし、重複した値の挿入に対して、クエリ結果での重複が必要になるような状況が発生する可能性があります。

  ただし、FILTER バージョンと FILTER...IN バージョンでは、同じ値が繰り返し現れるとき、1 回だけ値を抽出します。
- 2 番目のケースは VALUES が常に完全一致を実行する一方で、FILTER ではタイププロモーションが適用され、あいまい一致を実行する場合があることに関係しています。

  たとえば、"2.0"^^xsd:float などのリテラルを値に含める場合、VALUES クエリは、リテラル値とデータ型を含め、このリテラルに正確に一致します。

  対照的に、FILTER では、これらの数値リテラルに対してあいまい一致が出されます。一致には、同じ値で、xsd:double など、数値データ型が異なるリテラルを含めることができます。

> **Note**
>
> FILTER と VALUES の動作に、文字列リテラルまたは URI を列挙する際の違いはありません。

FILTER と VALUES の違いは、最適化と生成されるクエリ評価戦略に影響を与える可能性があります。ユースケースであいまい一致を必要としない限り、VALUES を使用することをお勧めします。これにより、型キャストに関連する特殊なケースを確認するのを避けることができます。その結果、VALUES では、高速で動作し、低コストの効率的なクエリを作成できることがよくあります。

Amazon Neptune の制限

リージョン

Amazon Neptune は、次の AWS リージョンで利用できます。

- 米国東部 (バージニア北部): us-east-1
- 米国東部 (オハイオ): us-east-2
- 米国西部 (北カリフォルニア): us-west-1
- 米国西部 (オレゴン): us-west-2
- カナダ (中部): ca-central-1
- 南米 (サンパウロ): sa-east-1
- 欧州 (ストックホルム): eu-north-1
- 欧州 (アイルランド): eu-west-1
- 欧州 (ロンドン): eu-west-2
- 欧州 (パリ): eu-west-3
- 欧州 (フランクフルト): eu-central-1
- 中東 (バーレーン): me-south-1
- 中東 (アラブ首長国連邦): me-central-1
- イスラエル (テルアビブ): il-central-1
- アフリカ (ケープタウン): af-south-1
- アジアパシフィック (香港): ap-east-1
- アジアパシフィック (東京): ap-northeast-1
- アジアパシフィック (ソウル): ap-northeast-2
- アジアパシフィック (大阪): ap-northeast-3
- アジアパシフィック (シンガポール): ap-southeast-1
- アジアパシフィック (シドニー): ap-southeast-2
- アジアパシフィック (ムンバイ): ap-south-1
- 中国 (北京): cn-north-1

- 中国 (寧夏): `cn-northwest-1`
- AWS GovCloud （米国西部): `us-gov-west-1`
- AWS GovCloud （米国東部): `us-gov-east-1`

中国リージョンの違い

多くの AWS サービスに当てはまるように、Amazon Neptune は中国 (北京) と中国 (寧夏) では、他の AWS リージョンと若干異なります。

たとえば、Neptune ML が Amazon API Gateway を使用してエクスポートサービスを作成する場合、IAM 認証はデフォルトで有効になっています。中国のリージョンでは、そのオプションを変更するプロセスは、他の地域とは若干異なります。

これらと他の違いはここで説明しているとおりです。

ストレージクラスターボリュームの最大サイズ

Neptune クラスターボリュームは、中国と を除くサポートされているすべてのリージョンで最大サイズ 128 テビバイト (TiB) まで拡張できます。 GovCloud制限は 64 TiB です。これは、以降のすべてのエンジンリリースに当てはまります。リリース： 1.0.2.2 (2020-03-09)Amazon Neptune ストレージ、信頼性、可用性 を参照してください。

サポートされる DB インスタンスサイズ

Neptune は、異なる AWS リージョンで異なる DB インスタンスクラスをサポートします。特定のリージョンでどのクラスがサポートされているかを調べるには、「Amazon Neptune の料金」にアクセスし、関心のあるリージョンを選択します。

各 AWS アカウントの制限

特定の管理機能では、Amazon Neptune は Amazon Relational Database Service (Amazon RDS) と共有の運用テクノロジーを使用します。

各 AWS アカウントには、作成できる Amazon Neptune および Amazon RDS リソースの数に対するリージョンごとの制限があります。これらのリソースには、DB インスタンスと DB クラスターが含まれます。

リソースの制限に達すると、リソースを作成するための追加の呼び出しは、例外が発生して失敗します。

Amazon Neptune と Amazon RDS 間で共有される制限のリストについては、Amazon RDS ユーザーガイドのAmazon RDS での制限を参照してください。

Neptune への接続には VPC が必要

Amazon Neptune は仮想プライベートクラウド (VPC) 専用サービスです。

また、VPC の外部からインスタンスにアクセスすることはできません。

Neptune は SSL が必要

エンジンバージョン `1.0.4.0` から、Amazon Neptune では、インスタンスまたはクラスターエンドポイントへの HTTPS 経由での Secure Sockets Layer (SSL) 接続のみが可能です。

Neptune では、以下の強力な暗号スイートを使用する、TLS バージョン 1.2 が必要です。

- `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`
- `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`
- `TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384`
- `TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256`
- `TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA`
- `TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA`

アベイラビリティーゾーンおよび DB サブネットグループ

Amazon Neptune では、最低 2 つのサポートされたアベイラビリティーゾーンにサブネットを持つ各クラスター用の DB サブネットグループが必要です。

異なるアベイラビリティーゾーンにある 3 つ以上のサブネットを使用することをお勧めします。

HTTP リクエストペイロードの最大サイズ (150 MB)

Gremlin と SPARQL HTTP リクエストの合計サイズは、150 MB 未満である必要があります。リクエストがこのサイズを超える場合、Neptune は `HTTP 400: BadRequestException` を返します。

この制限は Gremlin WebSockets 接続には適用されません。

Gremlin 実装の相違点

Amazon Neptune Gremlin の実装には、他の Gremlin の実装とは異なる場合がある特定の実装の詳細があります。

詳細については、「Amazon Neptune の Gremlin 標準への準拠」を参照してください。

Neptune では、文字列データ内の NULL 文字はサポートされていません。

Neptune では、文字列内の NULL 文字はサポートされていません。これは Gremlin と openCypher のプロパティグラフデータや RDF/SPARQL データにも当てはまります。

URI からの SPARQL UPDATE LOAD

URI の `SPARQL UPDATE LOAD` は、同じ VPC 内にあるリソースでのみ機能します。

これには、Amazon S3 VPC エンドポイントが作成されたクラスターと同じリージョンの Amazon S3 の URL も含まれます。

Amazon S3 の URL は HTTPS である必要があり、認証はいずれもその URL に含める必要があります。詳細については、Amazon Simple Storage Service API リファレンスのAuthenticating Requests: Using Query Parametersを参照してください。

VPC エンドポイント作成の詳細については、「Amazon S3 VPC エンドポイントの作成」を参照してください。

ファイルからデータをロードする必要がある場合は、Amazon Neptune ローダー API の使用をお勧めします。詳細については、「Amazon Neptune 一括ローダーを使用したデータの取り込み」を参照してください。

> **Note**
>
> Amazon Neptune ローダー API は、非 ACID です。

IAM 認証とアクセスコントロール

リリース 1.2.0.0 より前の Neptune エンジンバージョンでは、IAM 認証とアクセス制御は DB クラスターレベルでのみサポートされています。ただし、1.2.0.0リリース以降は、IAM ポリシーの条件キーを使用して、クエリベースのアクセスをよりきめ細かく制御できます。詳細については、「Neptune データアクセスポリシーステートメントでのクエリアクションの使用」および「Amazon Neptune での AWS Identity and Access Management (IAM) の概要」を参照してください。

Amazon Neptune コンソールには アクセスNeptuneReadOnlyAccess許可が必要です。このアクセスを取り消すことで、IAM ユーザーへのアクセスを制限することができます。詳細については、AWS Amazon Neptune の マネージド (事前定義) ポリシー を参照してください。

Amazon Neptune では、ユーザー名/パスワードベースのアクセスコントロールはサポートされていません。

WebSocket 同時接続と最大接続時間

Neptune DB インスタンスあたりの同時 WebSocket 接続数には制限があります。この制限に達すると、Neptune は割り当てられたヒープメモリがすべて使用されないように、新しい WebSocket 接続を開くリクエストをスロットリングします。

Neptune およびすべてのサーバーレスインスタンスでサポートされているすべての大規模なインスタンスタイプでは、 WebSocket 接続の最大同時数は 32K (32,768) です。

小さいインスタンスタイプの最大同時 WebSocket 接続数を以下の表に示します。

インスタンスタイプ	最大同時 WebSocket 接続数
db.t3.medium	512
db.t4g.medium	512
db.r5.large	2,048
db.r5d.large	2,048
db.r5.xlarge	4,096
db.r5.2xlarge	8,192

インスタンスタイプ	最大同時 WebSocket 接続数
db.r5d.2xlarge	8,192
db.r5.4xlarge	16,384
db.r5d.4xlarge	16,384
db.r6g.large	2,048
db.r6gd.large	2,048
db.r6g.xlarge	4,096
db.r6gd.xlarge	4,096
db.r6g.2xlarge	8,192
db.r6gd.2xlarge	8,192
db.r6g.4xlarge	16,384
db.r6gd.4xlarge	16,384
db.x2g.large	2,048
db.x2gd.large	2,048
db.x2g.xlarge	4,096
db.x2gd.xlarge	4,096
db.x2iedn.xlarge	4,096
db.x2g.2xlarge	8,192
db.m6gd.2xlarge	8,192
db.x2g.4xlarge	16,384
db.m6gd.4xlarge	16,384

インスタンスタイプ	最大同時 WebSocket 接続数
db.x2iedn.2xlarge	16,384
db.x2iezn.2xlarge	16,384
サーバーレス	32,768
(その他のラージインスタンスタイプ)	32,768

> **Note**
>
> Neptune エンジンリリース 1.1.1.0.0 から、Neptune はインスタンスタイプをサポートしなくなりました。R4

クライアントが接続を適切に閉じると、それがオープン接続数にすぐに反映されます。

クライアントが接続を閉じない場合、20 ~ 25 分のアイドルタイムアウト後に接続は自動的に閉じられます (アイドルタイムアウトは、クライアントから最後のメッセージを受信してから経過した時間です)。ただし、アイドルタイムアウトに達していない限り、Neptune は接続を無期限に開いたままにします。

IAM 認証が有効になっている場合、 WebSocket 接続が確立されてから 10 日以上経っても、その時点でまだ接続が切断されていない場合は、接続は常に切断されます。

プロパティとラベルの制限

グラフに含めることができる頂点とエッジ、または RDF クワッドの数に制限はありません。

また、1 つの頂点またはエッジが持つことができるプロパティやラベルの数に制限はありません。

それぞれのプロパティまたはラベルのサイズには、55 MB のサイズ制限があります。RDF において、RDF クワッドのすべての列 (S、P、O、G) の値が 55 MB を超えてはならないことを意味します。

画像などの大きなオブジェクトをグラフの頂点またはノードに関連付ける必要がある場合は、そのオブジェクトをファイルとして Amazon S3 に格納し、その Amazon S3 パスをプロパティまたはラベルとして使用できます。

Neptune バルクローダーに影響を与える制限

一度に 64 個を超える Neptune バルクロードジョブをキューに入れることはできません。

Neptune は、直近 1,024 個のバルクロードジョブのみを追跡します。

Neptune は、ジョブごとに直近 10,000 個のエラーの詳細のみを保存します。

他の AWS サービスでの使用

Amazon Neptune は他の多くの AWS サービスと組み合わせて使用できます。

Neptune と他ののサービスの統合

- AWS Glue — AWS Glue は、データの抽出、変換、ロード (ETL) ジョブを実行するのに役立つサーバーレスデータ統合サービスです。

  Neptune には、Glue ジョブ内での Python と Gremlin の使用を簡単にするオープンソースのライブラリ、neptune-python-utilities が用意されています。Neo4j スパークコネクタは、Scala ジョブと openCypher Glue ジョブの実行にも対応しています。

- Amazon SageMaker — Amazon SageMaker は、高品質の機械学習モデルを構築、トレーニング、デプロイするためのフル機能の機械学習プラットフォームです。

  Neptune は、主に次の 2 つの方法で SageMaker と統合します。

  - Neptune は Jupyter ノートブック用のオープンソースの Python パッケージを提供しています。これは、GitHub の Neptune グラフノートブックプロジェクトにあります。このパッケージには、Jupyter のマジック、チュートリアルノートブック、コードサンプルが含まれています。これらのコードサンプルは、グラフテクノロジーと Neptune について学習できるインタラクティブなコーディング環境で提供されます。Neptune は、SageMaker がホストする Jupyter ノートブック用のフルマネージド環境を提供し、オープンソースの Neptune グラフノートブックプロジェクトのノートブックに自動的にリンクします。
  - Neptune ML 機能を使用すると、数週間ではなく数時間で、大きなグラフで便利な機械学習モデルを構築し、トレーニングすることができます。これを達成するために、Neptune ML は、Amazon SageMaker と Deep Graph Library (DGL) を活用したグラフニューラルネットワーク (GNN) 技術を使用しています。

- AWS Lambda - AWS Lambda 関数は Neptune アプリケーションで多くの用途があります。

  一般的な Gremlin ドライバーと言語バリアントで Lambda 関数を使用する方法と、Java、JavaScript、および Python で書かれた Lambda 関数の具体的な例については、「Amazon Neptune で AWS Lambda 関数を使用する」を参照してください。

- Amazon Athena - Amazon Athena は、インタラクティブなクエリサービスであり、標準 SQL を使用して Amazon Simple Storage Service やその他のフェデレーテッドデータソース内のデータの分析を容易にします。

Neptune には、Athena が Neptune に保存されているデータと通信できるようにする Athena へのコネクタが用意されています。

- AWS Database Migration Service(AWS DMS) — AWS Database Migration Service は、あるデータベースから別のデータベースにデータを移行するために使用できる AWS ウェブサービスです。

  AWS DMS はサポートされているソースデータベースから Neptune に迅速かつ安全にデータをロードできます。移行中でもソースデータベースは完全に利用可能な状態に保たれ、それを利用するアプリケーションのダウンタイムを最小限に抑えられます。

- AWS Backup - AWS Backup は、フルマネージド型のバックアップサービスであり、クラウド内およびオンプレミスの AWS サービス間でデータのバックアップを簡単に一元化および自動化できます。

  AWS Backup では、データベース、ストレージ、およびコンピューティングのサポート対象 AWS サービス全体にわたって、一元化されたデータ保護ポリシーを使用して、Neptune クラスターの定期的な自動スナップショットを作成できます。

- AWS パンダ用 SDK — AWS パンダ用 SDK (以前は AWS データラングラー、または `awswrangler` と呼ばれていました) は、AWS プロフェッショナルサービスのオープンソース Python イニシアチブであり、`pandas` Pythonデータ分析ライブラリの機能を AWS に拡張して、`DataFrames` と Neptune を含む 30 を超える AWS データ関連サービスを接続します。

  SDK に加えて、Neptuneでの使用方法に関するチュートリアルや、Neptune ノートブックのサンプル (不正リング検出、合成 ID 検出、ロジスティクス分析など) もあります。

- JDBC ドライバー — Neptune JDBC ドライバーは、openCypher、Gremlin、SQL-Gremlin、および SPARQL クエリをサポートしています。

  JDBC 接続により、Tableau などのビジネスインテリジェンス (BI) ツールを使用して Neptune に簡単に接続できます。

Neptune のツールとユーティリティ

Amazon Neptune には、グラフの操作を簡素化および自動化できるツールとユーティリティが多数用意されています。これには次のものが含まれます。

Amazon Neptune ツール

- GraphQL 用 Amazon Neptune ユーティリティ - GraphQL 用 Amazon Neptune ユーティリティは、オープンソースの Node.js コマンドラインツールであり、Neptune プロパティグラフデータベース用の GraphQL API の作成と管理に役立ちます。これは、可変数の入力パラメータを持ち、可変数のネストされたフィールドを返す GraphQL クエリ用の GraphQL リゾルバーをコードなしで作成する方法です。

GraphQL 用 Amazon Neptune ユーティリティ

GraphQL 用 Amazon Neptune ユーティリティは、オープンソースの Node.js コマンドラインツールであり、Neptune プロパティグラフデータベース用の GraphQL API の作成と管理に役立ちます (まだ RDF データでは動作しません)。これは、可変数の入力パラメータを持ち、可変数のネストされたフィールドを返す GraphQL クエリ用の GraphQL リゾルバーをコードなしで作成する方法です。

https://github.com/aws/amazon-neptune-for-graphql にあるオープンソースプロジェクトとしてリリースされました。

このユーティリティは NPM を使用して次のようにインストールできます (詳細については、「インストールとセットアップ」を参照してください)。

```
npm i @aws/neptune-for-graphql -g
```

このユーティリティは、ノード、エッジ、プロパティ、エッジカーディナリティなど、既存の Neptune プロパティグラフのグラフスキーマを検出できます。次に、GraphQL タイプをデータベースのノードとエッジにマップするために必要なディレクティブを含む GraphQL スキーマを生成し、リゾルバーコードを自動生成します。リゾルバーコードは、GraphQL クエリによって要求されたデータのみを返すことでレイテンシーを最小限に抑えるように設計されています。

既存の GraphQL スキーマと空の Neptune データベースから始めて、その GraphQL スキーマをデータベースにロードするデータのノードとエッジにマップするために必要なディレクティブをユーティリティに推測させることもできます。または、既に作成または変更した GraphQL スキーマとディレクティブから始めることもできます。

ユーティリティは、AWS AppSync API、IAM ロール、データソース、スキーマ、リゾルバー、Neptune をクエリする AWS Lambda 関数など、パイプラインに必要なすべてのAWSリソースを作成できます。

> **Note**
>
> ここでのコマンドラインの例は Linux コンソールを想定しています。Windows を使用している場合は、行末のバックスラッシュ (「\」) をキャレット (「^」) に置き換えます。

トピック

GraphQL 用の Amazon Neptune ユーティリティのインストールとセットアップ

このユーティリティを既存の Neptune データベースで使用する場合は、データベースエンドポイントに接続できる必要があります。デフォルトでは、Neptune データベースには、そのデータベースが配置されている VPC 内からのみアクセスできます。

このユーティリティは Node.js のコマンドラインツールなので、ユーティリティを実行するには Node.js (バージョン 18 以上) がインストールされている必要があります。Neptune データベースと同じ VPC にある EC2 インスタンスに Node.js をインストールするには、こちらの手順に従ってください。ユーティリティを実行するインスタンスの最小サイズは t2.micro です。インスタンスの作成時に、[共通セキュリティグループ] プルダウンメニューから Neptune データベース VPC を選択します。

ただし、エンジンバージョン 1.2.0.0 以降では、VPC の外部からアクセス可能な Neptune データベースのパブリックエンドポイントを作成できます。パブリックエンドポイントを作成した場合は、Node.js とユーティリティをローカルマシンにインストールできます。macOS または Windows に Node.js をインストールするには、Node.js の Web サイトにアクセスして、インストーラーをダウンロードします。

ユーティリティ自体を EC2 インスタンスまたはローカルマシンにインストールするには、NPM を使用します。

```
npm install aws-neptune-for-graphql -g
```

その後、ユーティリティの help コマンドを実行して、ユーティリティが正しくインストールされたかどうかを確認できます。

```
neptune-for-graphql --help
```

AWS リソースを管理するために、AWS CLI もインストールした方がいいかもしれません。

既存の Neptune データベースのデータのスキャン

GraphQL に精通しているかどうかにかかわらず、以下のコマンドは GraphQL API を作成する最も速い方法です。これは、インストールセクションで説明したように GraphQL 用 Neptune ユーティリティをインストールして設定し、Neptune データベースのエンドポイントに接続していることを前提としています。

```
neptune-for-graphql \
  --input-graphdb-schema-neptune-endpoint (your neptune database endpoint):(port
 number) \
  --create-update-aws-pipeline \
  --create-update-aws-pipeline-name (your new GraphQL API name) \
  --output-resolver-query-https
```

このユーティリティはデータベースを分析して、データベース内のノード、エッジ、プロパティのスキーマを検出します。そのスキーマに基づいて、関連するクエリとミューテーションを含む GraphQL スキーマを推測します。次に、 AppSync GraphQL API とその使用に必要なAWSリソースを作成します。これらのリソースには、一対の IAM ロールと、GraphQL リゾルバーコードを含む Lambda 関数が含まれます。

ユーティリティが完了すると、コマンドで AppSync割り当てた名前のコンソールに新しい GraphQL API が表示されます。テストするには、メニューの AppSync クエリオプションを使用します。

データベースにデータを追加した後に同じコマンドを再度実行すると、 はそれに応じて AppSync API と Lambda コードを更新します。

コマンドに関連付けられているすべてのリソースを解放するには、以下を実行します。

```
neptune-for-graphql \
  --remove-aws-pipeline-name (your new GraphQL API name from above)
```

ディレクティブのない GraphQL スキーマからの開始

空の Neptune データベースから開始し、ディレクティブのない GraphQL スキーマを使用してデータを作成し、クエリを実行できます。以下のコマンドは、そのための AWS リソースを自動的に作成します。

```
neptune-for-graphql \
  --input-schema-file (your GraphQL schema file)
  --create-update-aws-pipeline \
  --create-update-aws-pipeline-name (name for your new GraphQL API) \
  --create-update-aws-pipeline-neptune-endpoint (your Neptune database endpoint):(port
 number) \
  --output-resolver-query-https
```

GraphQL スキーマファイルには、以下の TODO 例に示すように GraphQL スキーマタイプが含まれている必要があります。このユーティリティはスキーマを分析し、そのタイプに基づいて拡張バージョンを作成します。グラフデータベースに格納されているノードのクエリとミューテーションを追加し、スキーマにネストされたタイプがある場合は、データベースにエッジとして格納されているタイプ間の関係を追加します。

ユーティリティは、 AppSync GraphQL API と必要なすべてのAWSリソースを作成します。これらのリソースには、一対の IAM ロールと、GraphQL リゾルバーコードを含む Lambda 関数が含まれます。コマンドが完了すると、 AppSync コンソールで指定した名前の新しい GraphQL API が表示されます。テストするには、 AppSync メニューでクエリを使用します。

以下の例は、このしくみを示しています。

Todo の例、ディレクティブのない GraphQL スキーマからの開始

この例では、ディレクティブのない Todo GraphQL スキーマから始めます。このスキーマは *??? samples???* ディレクトリにあります。次の 2 つのタイプが含まれます。

```
type Todo {
  name: String
  description: String
  priority: Int
  status: String
```

```
  comments: [Comment]
}

type Comment {
  content: String
}
```

このコマンドは、Todo スキーマと空の Neptune データベースのエンドポイントを処理して、 で GraphQL API を作成しますAWS AppSync。

```
neptune-for-graphql /
  --input-schema-file ./samples/todo.schema.graphql \
  --create-update-aws-pipeline \
  --create-update-aws-pipeline-name TodoExample \
  --create-update-aws-pipeline-neptune-endpoint (empty Neptune database endpoint):(port
 number) \
  --output-resolver-query-https
```

ユーティリティは、 という出力フォルダに新しいファイルを作成しTodoExample.source.graphql、 の GraphQL API を作成します AppSync。このユーティリティは以下を推測します。

- Todo タイプで、新しい CommentEdge タイプ@relationshipに追加されました。これにより、というグラフデータベースエッジを使用して Todo を コメントに接続するようにリゾルバーに指示します CommentEdge。
- クエリとミューテーションに役立つ TodoInput という新しい入力が追加されました。
- 各タイプ (Todo、Comment) に 2 つのクエリが追加されました。1 つは、id を使用して、または入力にリストされているタイプフィールドのいずれかを使用して 1 つのタイプを取得するためのもので、もう 1 つはそのタイプの入力を使用してフィルタリングされた複数の値を取得するためのものです。
- 各タイプに 3 つのミューテーション (作成、更新、削除) が追加されました。削除するタイプは、id またはそのタイプの入力を使用して指定します。これらのミューテーションは、Neptune データベースに保存されているデータに影響します。
- 接続に「接続」と「削除」の 2 つのミューテーションが追加されました。Neptune によって使用される開始頂点と終了頂点のノード ID を入力として取り、接続はデータベース内のエッジです。

リゾルバーはクエリとミューテーションを名前で認識しますが、以下のようにカスタマイズできます。

TodoExample.source.graphql ファイルの内容は次のとおりです。

```
type Todo {
  _id: ID! @id
  name: String
  description: String
  priority: Int
  status: String
  comments(filter: CommentInput, options: Options): [Comment] @relationship(type:
 "CommentEdge", direction: OUT)
  bestComment: Comment @relationship(type: "CommentEdge", direction: OUT)
  commentEdge: CommentEdge
}

type Comment {
  _id: ID! @id
  content: String
}

input Options {
  limit: Int
}

input TodoInput {
  _id: ID @id
  name: String
  description: String
  priority: Int
  status: String
}

type CommentEdge {
  _id: ID! @id
}

input CommentInput {
  _id: ID @id
  content: String
}

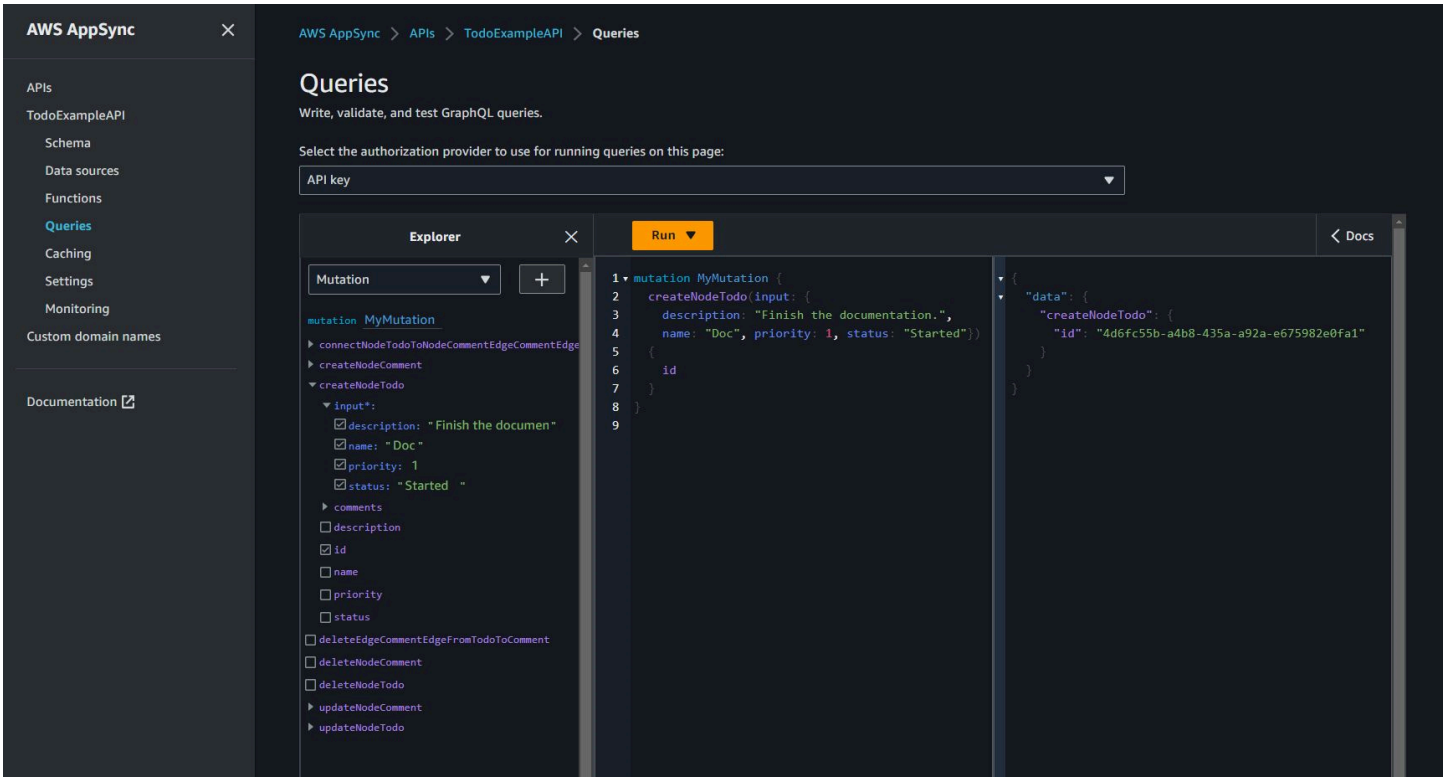
input Options {
  limit: Int
}
```

```
type Query {
  getNodeTodo(filter: TodoInput, options: Options): Todo
  getNodeTodos(filter: TodoInput): [Todo]
  getNodeComment(filter: CommentInput, options: Options): Comment
  getNodeComments(filter: CommentInput): [Comment]
}

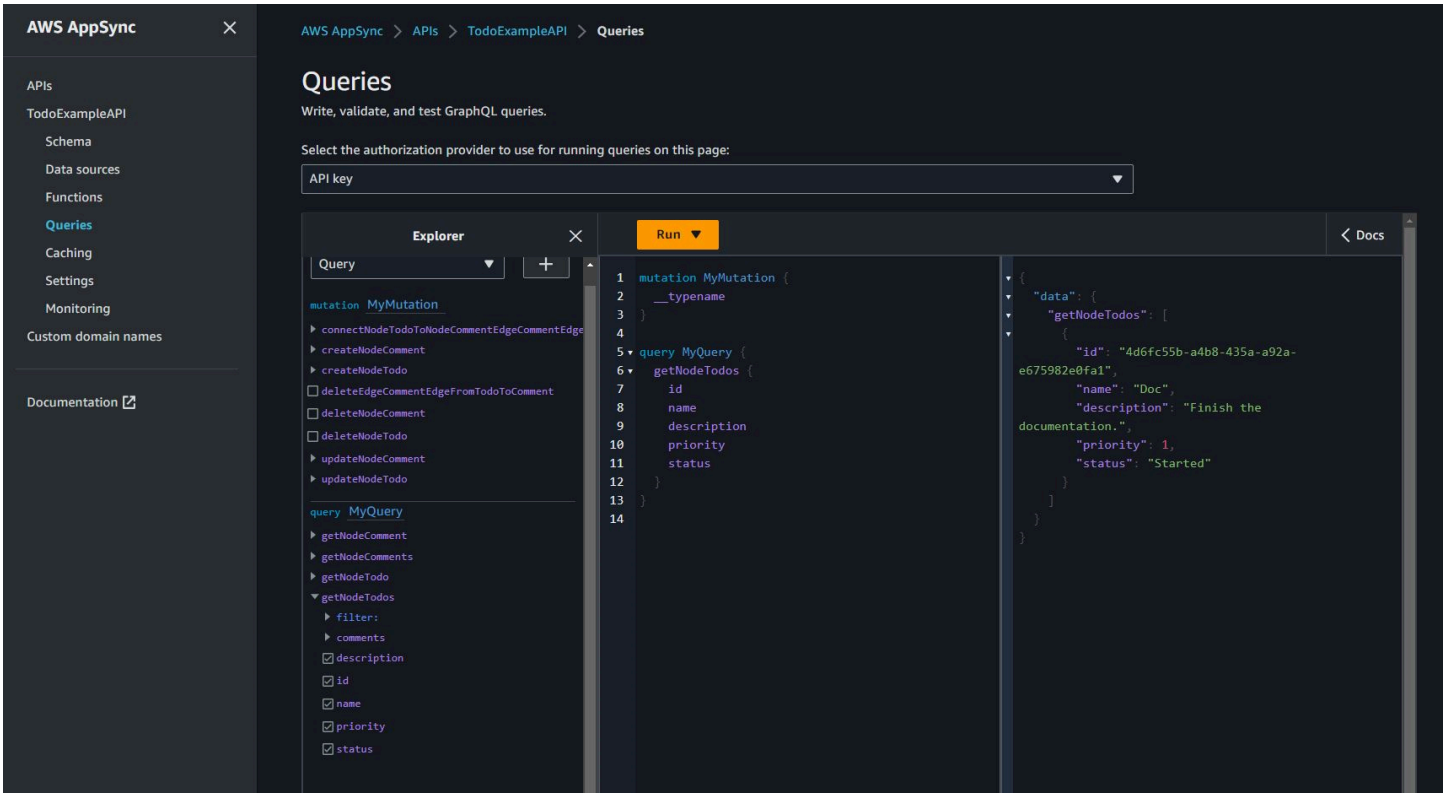
type Mutation {
  createNodeTodo(input: TodoInput!): Todo
  updateNodeTodo(input: TodoInput!): Todo
  deleteNodeTodo(_id: ID!): Boolean
  connectNodeTodoToNodeCommentEdgeCommentEdge(from_id: ID!, to_id: ID!): CommentEdge
  deleteEdgeCommentEdgeFromTodoToComment(from_id: ID!, to_id: ID!): Boolean
  createNodeComment(input: CommentInput!): Comment
  updateNodeComment(input: CommentInput!): Comment
  deleteNodeComment(_id: ID!): Boolean
}

schema {
  query: Query
  mutation: Mutation
}
```

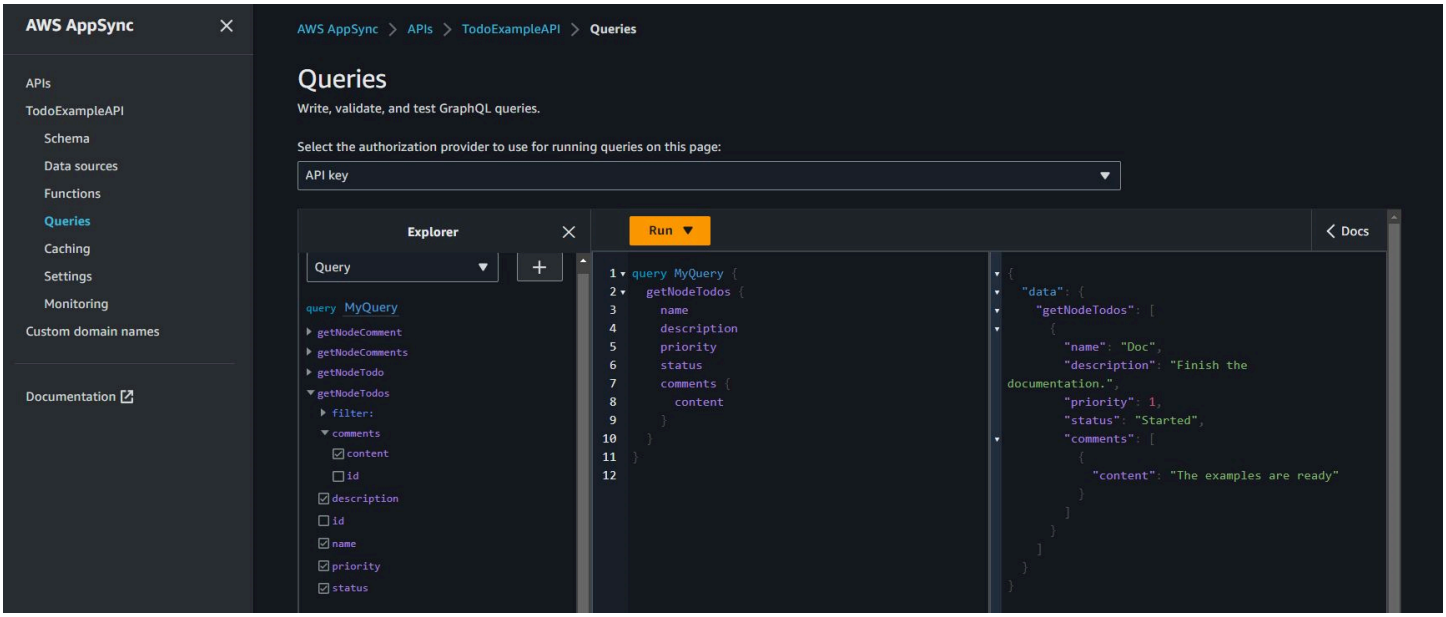
これで、データの作成とクエリが可能になりました。この場合は、 という名前の新しい GraphQL API のテストに使用されるクエリコンソールの AppSyncスナップショットTodoExampleAPIを次に示します。エクスプローラは、中央のウィンドウにクエリとミューテーションのリストを表示し、そこからクエリ、入力パラメータ、リターンフィールドを選択できます。このスクリーンショットには、createNodeTodo ミューテーションを使用して Todo ノードタイプを作成しているところが示されています。



このスクリーンショットは、getNodeTodos クエリを使用してすべての Todo ノードにクエリを実行しているところを示しています。



createNodeComment を使用してコメントを作成したら、connectNodeTodoToNodeCommentEdgeCommentEdge ミューテーションを使用してその ID を指定することで接続できます。Todo とそれに添付されたコメントを取得するネストされたクエリを以下に示します。



「ディレクティブの操作」で説明されているように、TodoExample.source.graphql ファイルに変更を加えたい場合は、編集したスキーマを入力として使用し、ユーティリティを再度実行できます。その後、ユーティリティはそれに応じて GraphQL API を変更します。

GraphQL スキーマのディレクティブの操作

次のようなコマンドを使用して、既にディレクティブが設定されている GraphQL スキーマから開始できます。

```
neptune-for-graphql \
  --input-schema-file (your GraphQL schema file with directives) \
  --create-update-aws-pipeline \
  --create-update-aws-pipeline-name (name for your new GraphQL API) \
  --create-update-aws-pipeline-neptune-endpoint  (empty Neptune database
 endpoint):(port number) \
  --output-resolver-query-https
```

ユーティリティが作成したディレクティブを変更したり、独自のディレクティブを GraphQL スキーマに追加したりできます。ディレクティブを操作するには次のような方法があります。

ミューテーションが発生しないようにユーティリティを実行する

ユーティリティが GraphQL API にミューテーションを生成しないようにするには、neptune-for-graphql コマンドの --output-schema-no-mutations オプションを使用します。

@alias ディレクティブ

@alias ディレクティブは、GraphQL スキーマタイプまたはフィールドに適用できます。グラフデータベースと GraphQL スキーマの間で異なる名前をマッピングします。構文は次のとおりです。

```
@alias(property: (property name))
```

以下の例では、airport は Airport GraphQL タイプにマッピングされたグラフデータベースノードラベルであり、desc は description フィールドにマッピングされたグラフノードプロパティです (「空路の例」を参照)。

```
type Airport @alias(property: "airport") {
  city: String
  description: String @alias(property: "desc")
}
```

標準の GraphQL フォーマットでは、パスカルケーシングの型名とキャメルケーシングのフィールド名が必要であることに注意してください。

@relationship ディレクティブ

@relationship ディレクティブは、ネストされた GraphQL タイプをグラフデータベースのエッジにマッピングします。構文は次のとおりです。

```
@relationship(edgeType: (edge name), direction: (IN or OUT))
```

コマンドの例を次に示します。

```
type Airport @alias(property: "airport") {
  ...
  continentContainsIn: Continent @relationship(edgeType: "contains", direction: IN)
  countryContainsIn: Country @relationship(edgeType: "contains", direction: IN)
  airportRoutesOut(filter: AirportInput, options: Options): [Airport]
 @relationship(edgeType: "route", direction: OUT)
```

```
  airportRoutesIn(filter: AirportInput, options: Options): [Airport]
 @relationship(edgeType: "route", direction: IN)
}
```

@relationship ディレクティブは「Todo の例」と「空路の例」の両方にあります。

@graphQuery および @cypher ディレクティブ

openCypher クエリを定義して、フィールド値を解決したり、クエリを追加したり、ミューテーションを追加したりできます。例えば、これは、アウトバウンドルートをカウントする新しい outboundRoutesCount フィールドを Airport タイプに追加します。

```
type Airport @alias(property: "airport") {
  ...
  outboundRoutesCount: Int @graphQuery(statement: "MATCH (this)-[r:route]->(a) RETURN
 count(r)")
}
```

以下は新しいクエリとミューテーションの例です。

```
type Query {
  getAirportConnection(fromCode: String!, toCode: String!): Airport \
    @cypher(statement: \
      "MATCH (:airport{code: '$fromCode'})-[:route]->(this:airport)-[:route]-
>(:airport{code:'$toCode'})")
}

type Mutation {
  createAirport(input: AirportInput!): Airport @graphQuery(statement: "CREATE
 (this:airport {$input}) RETURN this")
  addRoute(fromAirportCode:String, toAirportCode:String, dist:Int): Route \
    @graphQuery(statement: \
     "MATCH (from:airport{code:'$fromAirportCode'}),
 (to:airport{code:'$toAirportCode'}) \
      CREATE (from)-[this:route{dist:$dist}]->(to) \
      RETURN this")
}
```

RETURN を省略すると、リゾルバーはキーワード this を戻り値のスコープと見なすことに注意してください。

Gremlin クエリを使用してクエリやミューチュテーションを追加することもできます。

```
type Query {
  getAirportWithGremlin(code:String): Airport \
    @graphQuery(statement: "g.V().has('airport', 'code', '$code').elementMap()")  #
 single node
  getAirportsWithGremlin: [Airport] \
    @graphQuery(statement: "g.V().hasLabel('airport').elementMap().fold()")       #
 list of nodes
  getCountriesCount: Int \
    @graphQuery(statement: "g.V().hasLabel('country').count()")                   #
 scalar
}
```

現時点では、Gremlin クエリはスカラー値を返すもの、または単一ノードについて `elementMap()`、またはノードのリストについて `elementMap().fold()` を返すものに限られています。

@id ディレクティブ

`@id` ディレクティブは、`id` グラフデータベースエンティティにマッピングされたフィールドを識別します。Amazon Neptune のようなグラフデータベースには、一括インポート時に割り当てられたか、または自動生成された、ノードと エッジの一意の `id` が常にあります。例:

```
type Airport {
  _id: ID! @id
  city: String
  code: String
}
```

予約されているタイプ、クエリ、ミューテーション名

このユーティリティは、クエリとミューテーションを自動生成して、動作する GraphQL API を作成します。これらの名前のパターンはリゾルバーによって認識され、予約されています。タイプ `Airport` と接続タイプ `Route` の例を以下に示します。

`Options` タイプは予約されています。

```
input Options {
  limit: Int
}
```

filter および options 関数パラメータは予約されています。

```
type Query {
  getNodeAirports(filter: AirportInput, options: Options): [Airport]
}
```

クエリ名の getNode プレフィックスは予約されており、createNode、updateNode、deleteNode、connectNode、deleteNode、updateEdge、delete などのミューテーション名のプレフィックスは予約されています。

```
type Query {
  getNodeAirport(id: ID, filter: AirportInput): Airport
  getNodeAirports(filter: AirportInput): [Airport]
}

type Mutation {
  createNodeAirport(input: AirportInput!): Airport
  updateNodeAirport(id: ID!, input: AirportInput!): Airport
  deleteNodeAirport(id: ID!): Boolean
  connectNodeAirportToNodeAirportEdgeRout(from: ID!, to: ID!, edge: RouteInput!): Route
  updateEdgeRouteFromAirportToAirport(from: ID!, to: ID!, edge: RouteInput!): Route
  deleteEdgeRouteFromAirportToAirport(from: ID!, to: ID!): Boolean
}
```

GraphQL スキーマに変更を適用する

GraphQL ソーススキーマを変更して、ユーティリティを再実行すると、Neptune データベースから最新のスキーマを取得できます。ユーティリティは、データベース内の新しいスキーマを検出するたびに、新しい GraphQL スキーマを生成します。

GraphQL ソーススキーマを手動で編集し、Neptune データベースエンドポイントの代わりにソーススキーマを入力として使用してユーティリティを再実行することもできます。

最後に、次の JSON 形式を使用して、変更内容をファイルに入れることができます。

```
[
  {
    "type": "(GraphQL type name)",
    "field": "(GraphQL field name)",
    "action": "(remove or add)",
    "value": "(value)"
```

```
  }
]
```

例:

```
[
  {
    "type": "Airport",
    "field": "outboundRoutesCountAdd",
    "action": "add",
    "value":"outboundRoutesCountAdd: Int @graphQuery(statement: \"MATCH (this)-
[r:route]->(a) RETURN count(r)\")"
  },
  {
    "type": "Mutation",
    "field": "deleteNodeVersion",
    "action": "remove",
    "value": ""
  },
  {
    "type": "Mutation",
    "field": "createNodeVersion",
    "action": "remove",
    "value": ""
  }
]
```

次に、コマンドの `--input-schema-changes-file` パラメータを使用して、このファイルに対してユーティリティを実行すると、ユーティリティは変更を一度に適用します。

GraphQL ユーティリティのコマンドライン引数

- **`--help, -h`** — GraphQL ユーティリティのヘルプテキストをコンソールに返します。

- **`--input-schema`** ***`(schema text)`*** — 入力として使用する GraphQL スキーマ (ディレクティブあり、またはなし)。

- **`--input-schema-file`** ***`(file URL)`*** — 入力として使用する GraphQL スキーマを含んでいるファイルの URL。

- **`--input-schema-changes-file` *(file URL)*** — GraphQL スキーマに加えたい変更を含んでいるファイルの URL。Neptune データベースに対してユーティリティを複数回実行し、GraphQL ソーススキーマを手動で変更したり、カスタムクエリを追加したりすると、手動で行った変更は失われます。これを避けるには、変更内容を変更ファイルに入れ、この引数を使用して渡します。

  変更ファイルでは、以下の JSON 形式を使用します。

```
[
  {
    "type": "(GraphQL type name)",
    "field": "(GraphQL field name)",
    "action": "(remove or add)",
    "value": "(value)"
  }
]
```

  詳細については、「Todo の例」を参照してください。

- **`--input-graphdb-schema` *(schema text)*** — Neptune データベースに対してユーティリティを実行する代わりに、graphdb スキーマをテキスト形式で表現して入力として使用できます。graphdb スキーマは、次のような JSON 形式になっています。

```
{
  "nodeStructures": [
    { "label":"nodelabel1",
      "properties": [
        { "name":"name1", "type":"type1" }
      ]
    },
    { "label":"nodelabel2",
      "properties": [
          { "name":"name2", "type":"type1" }
      ]
    }
  ],
  "edgeStructures": [
    {
      "label":"label1",
```

```
      "directions": [
        { "from":"nodelabel1", "to":"nodelabel2", "relationship":"ONE-ONE|ONE-MANY|
MANY-MANY"  }
      ],
      "properties": [
        { "name":"name1", "type":"type1" }
      ]
    }
  ]
}
```

- **`--input-graphdb-schema-file`** ***`(file URL)`*** — Neptune データベースに対してユーティリティを実行する代わりに、graphdb スキーマをテキスト形式で表現して入力として使用できます。graphdb スキーマファイルの JSON 形式の例については、上記の「`--input-graphdb-schema`」を参照してください。

- **`--input-graphdb-schema-neptune-endpoint`** ***`(endpoint URL)`*** — ユーティリティが graphdb スキーマを抽出する Neptune データベースエンドポイント。

- **`--output-schema-file`** ***`(file name)`*** — GraphQL スキーマの出力ファイル名。指定しなかった場合、デフォルトは `output.schema.graphql` です。ただし、パイプライン名が `--create-update-aws-pipeline-name` を使用して設定された場合、デフォルトのファイル名は *`(pipline name)`*`.schema.graphql` です。

- **`--output-source-schema-file`** ***`(file name)`*** — ディレクティブを含む GraphQL スキーマの出力ファイル名。指定しなかった場合、デフォルトは `output.source.schema.graphql` です。ただし、パイプライン名が `--create-update-aws-pipeline-name` を使用して設定された場合、デフォルト名は *`(pipeline name)`*`.source.schema.graphql` です。

- **`--output-schema-no-mutations`** — この引数が存在する場合、ユーティリティは GraphQL API にミューテーションを生成せず、クエリのみを生成します。

- **`--output-neptune-schema-file` *(file name)*** — ユーティリティが検出した Neptune graphdb スキーマの出力ファイル名。指定しなかった場合、デフォルトは `output.graphdb.json` です。ただし、パイプライン名が `--create-update-aws-pipeline-name` を使用して設定された場合、デフォルトのファイル名は *(pipeline name)*`.graphdb.json` です。

- **`--output-js-resolver-file` *(file name)*** — リゾルバーコードのコピーの出力ファイル名。指定しなかった場合、デフォルトは `output.resolver.graphql.js` です。ただし、パイプライン名が `--create-update-aws-pipeline-name` を使用して設定された場合、ファイル名は *(pipeline name)*`.resolver.graphql.js` です。

  このファイルは、リゾルバーを実行する Lambda 関数にアップロードされたコードパッケージに圧縮されています。

- **`--output-resolver-query-sdk`** — この引数は、ユーティリティの Lambda 関数が Neptune データ SDK を使用して Neptune にクエリを実行するように指定します。このデータ SDK は Neptune エンジンバージョン 1.2.1.0.R5 (デフォルト) 以降で使用可能です。ただし、ユーティリティが古い Neptune エンジンバージョンを検出した場合は、代わりに HTTPS Lambda オプションを使用することを推奨します。このオプションは `--output-resolver-query-https` 引数を使用して呼び出すことができます。

- **`--output-resolver-query-https`** — この引数は、ユーティリティの Lambda 関数が Neptune HTTPS API を使用して Neptune にクエリを実行するように指定します。

- **`--create-update-aws-pipeline`** – この引数は、GraphQL API やリゾルバーを実行する Lambda など、使用する AppSync GraphQL API のAWSリソースの作成をトリガーします。

- **`--create-update-aws-pipeline-name` *(pipeline name)*** - この引数は、Lambda 関数の AppSync または `pipeline-name`関数の `pipeline-name` API のように、パイプラインの名前を設定します。名前が指定されなかった場合、`--create-update-aws-pipeline` は Neptune データベース名を使用します。

- **`--create-update-aws-pipeline-region` *`(AWS region)`*** — この引数は、GraphQL API のパイプラインが作成される AWS リージョンを設定します。指定されなかった場合、デフォルトのリージョンは、`us-east-1` か、データベースエンドポイントから抽出された Neptune データベースが置かれているリージョンのいずれかです。
- **`--create-update-aws-pipeline-neptune-endpoint` *`(endpoint URL)`*** — この引数は、Lambda 関数がデータベースにクエリするために使用する Neptune データベースエンドポイントを設定します。設定されていない場合は、`--input-graphdb-schema-neptune-endpoint` によって設定されたエンドポイントが使用されます。
- **`--remove-aws-pipeline-name` *`(pipeline name)`*** — この引数は、`--create-update-aws-pipeline` を使用して作成されたパイプラインを削除します。削除するリソースは、*`(pipeline name)`*`.resources.json` という名前のファイルにリストされます。
- **`--output-aws-pipeline-cdk`** – この引数は、GraphQL API やリゾルバーを実行する Lambda 関数など、 AppSync GraphQL API のAWSリソースの作成に使用できる CDK ファイルの作成をトリガーします。
- **`--output-aws-pipeline-cdk-neptume-endpoint` *`(endpoint URL)`*** — この引数は、Lambda 関数が Neptune データベースにクエリするために使用する Neptune データベースエンドポイントを設定します。設定されていない場合は、`--input-graphdb-schema-neptune-endpoint` によって設定されたエンドポイントが使用されます。
- **`--output-aws-pipeline-cdk-name` *`(pipeline name)`*** – この引数は、 AppSync API のパイプライン名と使用する Lambda pipeline-name 関数を設定します。指定されなかった場合、`--create-update-aws-pipeline` は Neptune データベース名を使用します。
- **`--output-aws-pipeline-cdk-region` *`(AWS region)`*** — この引数は、GraphQL API のパイプラインが作成される AWS リージョンを設定します。指定されなかった場合、デフォルトのリージョンは、`us-east-1` か、データベースエンドポイントから抽出された Neptune データベースが置かれているリージョンのいずれかです。
- **`--output-aws-pipeline-cdk-file` *`(file name)`*** — これは、CDK ファイル名を設定します。設定されなかった場合、デフォルトは *`(pipeline name)`*`-cdk.js` です。

Neptune サービスのエラー

Amazon Neptune には 2 セットの異なるエラーがあります。

- Neptune DB クラスターのエンドポイントのみのグラフエンジンのエラー。
- これらのエラーは、AWS SDK および AWS Command Line Interface (AWS CLI) を使用して Amazon Neptune リソースを作成および変更する API に関連付けられています。

トピック

グラフエンジンのエラーメッセージおよびコード

Amazon Neptune エンドポイントからは Gremlin と SPARQL の標準エラーが返されることがあります。

同じエンドポイントから Neptune に固有のエラーが返されることもあります。このセクションでは、Neptune のエラーメッセージ、コード、推奨アクションについて説明します。

> **Note**
>
> これらのエラーは Neptune DB クラスターのエンドポイント用です。AWS SDK および AWS CLI を使用して Neptune リソースを作成および変更する API には、異なる一連の一般的なエラーがあります。これらのエラーの詳細については、「the section called “API エラー”」を参照してください。

グラフエンジンのエラー形式

Neptune のエラーメッセージとしては、関連する HTTP エラーコードと JSON 形式のレスポンスが返されます。

```
HTTP/1.1 400 Bad Request
```

```
x-amzn-RequestId: LDM6CJP8RMQ1FHKSC1RBVJFPNVV4KQNSO5AEMF66Q9ASUAAJG
Content-Type: application/x-amz-json-1.0
Content-Length: 465
Date: Thu, 15 Mar 2017 23:56:23 GMT

{
    "requestId": "0dbcded3-a9a1-4a25-b419-828c46342e47",
    "code": "ReadOnlyViolationException",
    "detailedMessage": "The request is rejected because it violates some read-only
 restriction, such as a designation of a replica as read-only."
}
```

グラフエンジンのクエリエラー

以下の表には、エラーコード、メッセージ、HTTP ステータスを示しています。

また、リクエストを再試行できるかどうかも示しています。一般的に、リクエストは新しい試行で成功する可能性がある場合は再試行してもかまいません。

Neptune Service Error Code	HTTP status	Ok to Retry?	Message
`AccessDeniedException`	403	No	Authentication or authorization failure.
`BadRequestException`	400	No	The request could not be completed.
`BadRequestException`	400	No	Request size exceeds max allowed value of 157286400 bytes.
`CancelledByUserException`	500	Yes	The request processing was cancelled by an authorized client.
`ConcurrentModificationException`	500	Yes	The request processing did not succeed due to a modification conflict. The

Neptune Service Error Code	HTTP status	Ok to Retry?	Message
			client should retry the request.
`ConstraintViolationException`	400	Yes	The query engine discovered, during the execution of the request, that the completion of some operation is impossible without violating some data integrity constraints, such as persistence of in- and out-vertices while adding an edge. Such conditions are typically observed if there are concurrent modifications to the graph, and are transient. The client should retry the request.
`FailureByQueryException`	500	Yes	Calling fail() caused request processing to fail.
`InternalFailureException`	500	Yes	The request processing has failed.

Neptune Service Error Code	HTTP status	Ok to Retry?	Message
`InvalidNumericDataException`	400	No	Invalid use of numeric data which cannot be represented in 64-bit storage size.
`InvalidParameterException`	400	No	An invalid or out-of-range value was supplied for some input parameter or invalid syntax in a supplied RDF file.
`MalformedQueryException`	400	No	The request is rejected because it contains a query that is syntactically incorrect or does not pass additional validation.
`MemoryLimitExceededException`	500	Yes	The request processing did not succeed due to lack of memory, but can be retried when the server is less busy.
`MethodNotAllowedException`	405	No	The request is rejected because the chosen HTTP method is not supported by the used endpoint.

Neptune Service Error Code	HTTP status	Ok to Retry?	Message
`MissingParameterException`	400	No	A required parameter for the specified action is not supplied.
`QueryLimitExceededException`	500	Yes	The request processing did not succeed due to the lack of a limited resource, but can be retried when the server is less busy.
`QueryLimitException`	400	No	Size of query exceeds system limit.
`QueryTooLargeException`	400	No	The request was rejected because its body is too large.
`ReadOnlyViolationException`	400	No	The request is rejected because it violates some read-only restriction, such as a designation of a replica as read-only.
`ThrottlingException`	500	Yes	Rate of requests exceeds the maximum throughput. OK to retry.
`TimeLimitExceededException`	500	Yes	The request processing timed out.

Neptune Service Error Code	HTTP status	Ok to Retry?	Message
`TooManyRequestsException`	429	Yes	The rate of requests exceeds the maximum throughput. OK to retry.
`UnsupportedOperationException`	400	No	The request uses a currently unsupported feature or construct.

IAM 認証エラー

これらのエラーは IAM 認証が有効になっているクラスターに固有のものです。

以下の表には、エラーコード、メッセージ、HTTP ステータスを示しています。

Neptune Service Error Code	HTTP status	Message
Incorrect IAM User/Policy	403	You do not have sufficient access to perform this action.
Incorrect or Missing Region	403	Credential should be scoped to a valid Region, not '*region*'.
Incorrect or Missing Service Name	403	Credential should be scoped to correct service: '`neptune-db `'.
Incorrect or Missing Host Header / Invalid Signature	403	The request signature we calculated does not match the signature you provided. Check your AWS Secret Access Key and signing method. Consult the service documentation for

Neptune Service Error Code	HTTP status	Message
		details. Host header is missing or hostname is incorrect.
Missing `X-Amz-Security-Token`	403	'`x-amz-security-token`' is named as a `SignedHeader`, but it does not exist in the HTTP request
Missing Authorization Header	403	The request did not include the required authorization header, or it was malformed.
Missing Authentication Token	403	Missing Authentication Token.
Old Date	403	Signature expired: *`20181011T213907Z`* is now earlier than *`20181011T213915Z`* (*`20181011T214415Z`* - *`5 #`*.)
Future Date	403	Signature not yet current: *`20500224T213559Z`* is still later than *`20181108T225925Z`* (*`20181108T225425Z`* + *`5 #`*.)
Incorrect Date Format	403	Date must be in ISO-8601 'basic format'. Got '*`date`*'. See https://en.wikipedia.org/wiki/ISO_8601.
Unknown/Missing Access Key or Session Token	403	The security token included in the request is invalid.

Neptune Service Error Code	HTTP status	Message
Unknown/Missing Secret Key	403	The request signature we calculated does not match the signature you provided. Check your AWS Secret Access Key and signing method. Consult the service documentation for details. Host header is missing or hostname is incorrect.
`TooManyRequestsException`	429	The rate of requests exceeds the maximum throughput. OK to retry.

DB クラスターの管理 API のエラーメッセージおよびコード

これらの Amazon Neptune エラーは、AWS SDK および AWS CLI を使用して Neptune リソースを作成および変更する API に関連付けられています。

以下の表には、エラーコード、メッセージ、HTTP ステータスを示しています。

Neptune Service Error Code	HTTP status	Message
`AccessDeniedException`	403	このアクションを実行する十分なアクセス権限がありません。
`IncompleteSignature`	400	リクエストの署名がAWS 基準に適合しません。
`InternalFailure`	500	不明なエラー、例外、または障害により、リクエスト処理が失敗しました。
`InvalidAction`	400	リクエストされたアクション、またはオペレーションは無効です。アクションが正し

Neptune Service Error Code	HTTP status	Message
		く入力されていることを確認します。
InvalidClientTokenId	403	指定された x.509 証明書、または AWS アクセスキー ID が見つかりません。
InvalidParameterCombination	400	同時に使用できないパラメータが、同時使用されています。
InvalidParameterValue	400	無効な値または範囲外の値が入力パラメータとして指定されました。
InvalidQueryParameter	400	無効な値または範囲外の値が入力パラメータとして指定されました。
MalformedQueryString	400	クエリ文字列に構文エラーがあります。
MissingAction	400	リクエストに、アクションまたは必須パラメータが含まれていません。
MissingAuthenticationToken	403	リクエストには、有効な (登録された) AWS アクセスキー ID、または X.509 証明書のどちらか一方が含まれている必要があります。
MissingParameter	400	指定したアクションの必須パラメータが指定されていません。

Neptune Service Error Code	HTTP status	Message
`OptInRequired`	403	サービスを利用するためには、AWS アクセスキー ID を取得する必要があります。
`RequestExpired`	400	リクエストの日付スタンプの 15 分以上後またはリクエストの有効期限 (署名付き URL の場合など) の 15 分以上後に、リクエストが到着しました。または、リクエストの日付スタンプが現在より 15 分以上先です。
`ServiceUnavailable`	503	リクエストは、サーバーの一時的障害のために実行に失敗しました。
`ThrottlingException`	500	リクエストは、制限が必要なために実行が拒否されました。
`ValidationError`	400	入力が、AWS サービスで指定された制約を満たしていません。

Neptune ローダーのエラーおよびフィードメッセージ

次のメッセージは、Neptune ローダーの `status` エンドポイントから返されます。詳細については、「Get-Status API」を参照してください。

次の表は、ローダーフィードのコードと説明を示しています。

Error or Feed Code	Description
LOAD_NOT_STARTED	ロードは記録されましたが、開始されていません。
LOAD_IN_PROGRESS	ロードが進行中であることを示し、現在ロードされているファイルの数を指定します。 ローダーがファイルを解析すると、並列にロードする 1 つ以上のチャンクが作成されます。1 つのファイルは複数のチャンクを生成できるため、通常、このメッセージに含まれるファイル数は、バルクロードプロセスで使用されるスレッドの数よりも少なくなります。
LOAD_COMPLETED	エラーなしで、または許容されるしきい値内のエラーで、ロードは完了しました。
LOAD_CANCELLED_BY_USER	ユーザーによってロードがキャンセルされました。
LOAD_CANCELLED_DUE_TO_ERRORS	エラーのためシステムによってロードがキャンセルされました。
LOAD_UNEXPECTED_ERROR	予期しないエラーのためロードに失敗しました。
LOAD_FAILED	1 つ以上のエラーの結果としてロードが失敗しました。
LOAD_S3_READ_ERROR	断続的または一時的な Amazon S3 の接続の問題のため、フィードに失敗しました。いずれかのフィードがこのエラーを受信すると、全体的なロードステータスは LOAD_FAILED に設定されます。
LOAD_S3_ACCESS_DENIED_ERROR	S3 バケットへのアクセスが拒否されました。いずれかのフィードがこのエラーを受信する

Error or Feed Code	Description
	と、全体的なロードステータスは LOAD_FAILED に設定されます。
LOAD_COMMITTED_W_WRITE_CONFLICTS	未解決の書き込み競合で、ロードされたデータがコミットされました。 ローダーは、別のトランザクションで書き込み競合を解決しようとし、ロードの進行に合わせてフィードのステータスを更新します。最終的なフィードのステータスが LOAD_COMMITTED_W_WRITE_CONFLICTS の場合、ロードを再開しようとすると、書き込み競合なしで成功する可能性が高くなります。通常、書き込み競合は、不正な入力データには関係しませんが、データの重複は書き込み競合の可能性を高める可能性があります。
LOAD_DATA_DEADLOCK	Load was automatically rolled back due to deadlock.
LOAD_DATA_FAILED_DUE_TO_FEED_MODIFIED_OR_DELETED	ロードの開始後にファイルが削除または更新されたため、フィードに失敗しました。
LOAD_FAILED_BECAUSE_DEPENDENCY_NOT_SATISFIED	依存関係チェックが失敗したため、ロードリクエストは実行されませんでした。
LOAD_IN_QUEUE	ロードリクエストはキューに入れられ、実行の待機中です。
LOAD_FAILED_INVALID_REQUEST	リクエストが無効であったため、ロードに失敗しました (たとえば、指定されたソース/バケットが存在しないか、ファイル形式が無効である可能性があります)。

Amazon Neptune のエンジンリリース

Amazon Neptune は定期的にエンジンの更新をリリースします。

instance-status API または Neptune コンソールを使用して、現在インストールされているエンジンリリースバージョンを確認できます。バージョン番号は、元のメジャーリリース、マイナーリリース、またはパッチリリースのいずれを実行しているかを示します。リリースの番号付けの詳細については、「エンジンバージョン番号」を参照してください。

全般的な更新の詳細については、「クラスターのメンテナンス」を参照してください。

エンジンリリース 1.3.0.0 以降のエンジンバージョンは、次の表に示す構造になります。マイナーバージョン番号は、`AutoMinorVersionUpgrade` 処理で評価される番号です。

Version	製品バージョン	メジャーバージョン	マイナーバージョン	パッチバージョン	ステータス	解放済み	サポート終了	アップグレード後
1.3.2.1	1	3	2	1	ACTIVE	2024-06-20	2025-11-30	該当なし
1.3.2.0	1	3	2	0	ACTIVE	2024-06-10	2025-11-30	1.3.2.1
1.3.1.0	1	3	1	0	ACTIVE	2024-03-06	2025-11-30	1.3.2.1
1.3.0.0	1	3	0	0	ACTIVE	2023-11-15	2025-11-30	1.3.2.1

次の表は、1.0.1.0 以降のすべてのエンジンリリースと、バージョン に関する情報を示しています end-of-life。表の日付を参考にして、テストとアップグレードのサイクルを計画することができます。

Version	メジャーバージョン	マイナーバージョン	ステータス	解放済み	サポート終了	アップグレード後
1.2.1.1	1.2	1.1	ACTIVE	2024-03-11	2025-03-06	1.3.0.0
1.2.1.0	1.2	1.0	ACTIVE	2023-03-08	2025-03-06	1.3.0.0
1.2.0.2	1.2	0.2	ACTIVE	2022-11-16	2025-03-06	1.3.0.0
1.2.0.1	1.2	0.1	ACTIVE	2022-10-26	2025-03-06	1.3.0.0
1.2.0.0	1.2	0.0	ACTIVE	2022-07-21	2025-03-06	1.3.0.0
1.1.1.0	1.1	1.0	ACTIVE	2022-04-19	2024-10-31	1.2.1.0
1.1.0.0	1.1	0.0	ACTIVE	2021-11-19	2024-10-31	1.1.1.0
1.0.5.1	1.0	5.1	廃止済み	2021-10-01	2023-01-30	1.1.0.0
1.0.5.0	1.0	5.0	廃止済み	2021-07-27	2023-01-30	1.1.0.0
1.0.4.2	1.0	4.2	廃止済み	2021-06-01	2023-01-30	1.1.0.0
1.0.4.1	1.0	4.1	廃止済み	2020-12-08	2023-01-30	1.1.0.0
1.0.4.0	1.0	4.0	廃止済み	2020-10-12	2023-01-30	1.1.0.0
1.0.3.0	1.0	3.0	廃止済み	2020-08-03	2023-01-30	1.1.0.0
1.0.2.2	1.0	2.2	廃止済み	2020-03-09	2022-07-29	1.0.3.0
1.0.2.1	1.0	2.1	廃止済み	2019-11-22	2022-07-29	1.0.3.0
1.0.2.0	1.0	2.0	廃止済み	2019-11-08	2020-05-19	1.0.3.0
1.0.1.2	1.0	1.2	廃止済み	2019-10-15	—	—
1.0.1.1	1.0	1.1	廃止済み	2019-08-13	—	—

Version	メジャーバージョン	マイナーバージョン	ステータス	解放済み	サポート終了	アップグレード後
1.0.1.0.*	1.0	1.0.*	廃止済み	2019-07-02 およびそれ以前	—	—

メジャーエンジンのバージョン end-of-life 計画

Neptune のエンジンバージョンは、ほとんどの場合、暦四半期の終わりに有効期限を迎えます。例外が発生するのは、セキュリティや可用性に関する重要な問題が発生した場合のみです。

エンジンバージョンの有効期間が終了すると、Neptune データベースを新しいバージョンにアップグレードする必要があります。

一般的に、Neptune エンジンバージョンは以下のように引き続きご利用いただけます。

- マイナーエンジンバージョン: マイナーエンジンバージョンは、リリース後少なくとも 6 か月間はご利用いただけます。
- メジャーエンジンバージョン: メジャーエンジンバージョンは、リリース後少なくとも 12 か月間はご利用いただけます。

エンジンバージョンの有効期限の少なくとも 3 か月前に、 AWS はアカウント AWS に関連付けられた E メールアドレスに自動 E メール通知を送信し、同じメッセージを AWS Health Dashboard に投稿します。これにより、アップグレードを計画し、準備する時間を確保できます。

エンジンバージョンの有効期間が終了すると、そのバージョンを使用して新しいクラスターやインスタンスを作成できなくなり、オートスケーリングでもそのバージョンを使用してインスタンスを作成できなくなります。

実際に有効期間が終了したエンジンバージョンは、メンテナンス期間中に自動的にアップグレードされます。エンジンバージョンの有効期限の 3 か月前に送信されるメッセージには、自動的にアップグレードされるバージョン、DB クラスターへの影響、推奨アクションなど、この自動更新の内容に関する詳細が記載されています。

Important

データベースエンジンのバージョンを最新の状態に保つ責任はお客様にあります。 AWS は、セキュリティ、プライバシー、および可用性に関する最新の保護措置の恩恵を受けるために、すべてのお客様にデータベースを最新のエンジンバージョンにアップグレードするよう促します。廃止日を過ぎたサポートされていないエンジンまたはソフトウェア (「レガシーエンジン」) でデータベースを運用すると、セキュリティ、プライバシー、およびダウンタイムイベントを含む運用上のリスクにさらされる可能性が高くなります。
任意のエンジンでのデータベースの運用には、サービスの使用に適用される契約が適用されます AWS 。レガシーエンジンは一般公開されていません。 はレガシーエンジンのサポートを提供し AWS なくなり、 AWS レガシーエンジンが サービス、 AWSその子会社、または第三者にセキュリティまたは責任のリスク、または損害のリスクをもたらす AWS と判断した場合、いつでもレガシーエンジンへのアクセスまたは使用に制限を課す可能性があります。レガシーエンジンでコンテンツを引き続き実行すると、コンテンツが利用できなくなったり、破損したり、回復できなくなったりする可能性があります。レガシーエンジンで実行されているデータベースは、サービスレベルアグリーメント (SLA) の例外の対象となります。レガシーエンジンで実行されているデータベースおよび関連ソフトウェアには、バグ、エラー、欠陥、または有害なコンポーネントが含まれています。それに応じて、契約またはサービス条件に矛盾する内容にかかわらず、レガシーエンジンは AWS 「現状有姿」で提供されます。

Amazon Neptune エンジンバージョン 1.3.2.1 (2024-06-20)

2024-06-20 の時点で、エンジンバージョン 1.3.2.1 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

Note

エンジンリリース 1.3.0.0 では、カスタムパラメータグループとカスタムクラスターパラメータグループに新しい形式が導入されました。そのため、1.3.0.0 より前のエンジンバージョンからエンジンバージョン 1.3.0.0 以降にアップグレードする場合は、パラメータグループファミリー `neptune1.3` を使用している既存のカスタムパラメータグループとカスタムクラスターパラメータグループをすべて再作成する必要があります。以前のリリースではパラメータグループファミリー `neptune1` または `neptune1.2` を使用していましたが、

これらのパラメータグループはリリース 1.3.0.0 以降では動作しません。詳細については、「Amazon Neptune パラメータグループ」を参照してください。

このエンジンリリースで修正された不具合

openCypher の修正

- パラメータLIMITとして SKIPおよび を持つ内部WITH句を含むパラメータ化されたクエリのクエリプランキャッシュ機能でバグが検出されました。SKIP/LIMIT 値は適切にパラメータ化されなかったため、異なるパラメータ値を持つ同じキャッシュされたクエリプランの後続の実行でも、最初の実行と同じ結果が返されます。これは修正されています。

```
# insert some nodes
UNWIND range(1, 10) as i CREATE (s {name: i}) RETURN s

# sample query
MATCH (p)
WITH p ORDER BY p.name SKIP $s LIMIT $l
RETURN p.name as res

# first time executing with {"s": 2, "l": 1}
{
  "results" : [ {
    "res" : 3
  } ]
}

# second time executing with {"s": 2, "l": 10}
# due to bug, produces
{
  "results" : [ {
    "res" : 3
  } ]
}
# with fix, produces correct results:
{
  "results" : [ {
    "res" : 3
  }, {
    "res" : 4
  }, {
```

```
    "res" : 5
  }, {
    "res" : 6
  }, {
    "res" : 7
  }, {
    "res" : 8
  }, {
    "res" : 9
  }, {
    "res" : 10
  } ]
}%
```

- 渡されたパラメータがデータベースにまだ存在しないInternalFailureException場合に、パラメータ化されたミューテーションクエリが をスローするバグを修正しました。
- クエリリソースのクリーンアップ中に競合状態になった後、パラメータ化された Bolt クエリが停止するバグを修正しました。

1.3.2.1 の変更が 1.3.2.0 から引き継がれました

エンジンリリース 1.3.2.0 から引き継がれた改善点

全般的な機能強化

- 暗号スイート TLS_AES_128_GCM_SHA256 および TLS_AES_256_GCM_SHA384 を含む TLS バージョン 1.3 のサポート。TLS 1.3 はオプションです。TLS 1.2 はまだ最小です。
- openCypher の dateime 形式の拡張サポートは、このバージョンでは lab_mode です。テストすることをお勧めします。

Gremlin の改善点

- TinkerPop 3.7.x アップグレード
  - Gremlin 言語の大規模な拡張を提供します。
    - 文字列、リスト、日付を処理するための新しいステップ。
    - `mergeV()` ステップで基数を指定するための新しい構文。
    - `union()` を開始ステップとして使用できるようになりました。

    - 3.7.x の変更点の詳細については、TinkerPop アップグレードドキュメント を参照してください。
    - Java 用のクライアント Gremlin 言語ドライバーをアップグレードするときは、シリアライザクラスの名前変更が一部無効になっていることに注意してください。指定する場合は、設定ファイルとコードでパッケージとクラスの命名を更新する必要があります。
- `StrictTimeoutValidation` (を含める`StrictTimeoutValidation`ことでラボモードで有効にする場合のみ`StrictTimeoutValidation=enabled`): `StrictTimeoutValidation`パラメータの値が の場合`enabled`、リクエストオプションまたはクエリヒントとして指定されたクエリごとのタイムアウト値は、パラメータグループでグローバルに設定された値を超えることはできません。このような場合、Neptune は をスローします`InvalidParameterException`。この設定は、値が の場合`disabled`、`/status`エンドポイントのレスポンスで確認できます。Neptune バージョン 1.3.2.0 および 1.3.2.1 では、このパラメータのデフォルト値は ですDisabled。

openCypher の改善

- 低レイテンシークエリとスループットパフォーマンスの向上: 低レイテンシー openCypher クエリの全体的なパフォーマンスの向上。新しいバージョンでは、このようなクエリのスループットも向上します。パラメータ化されたクエリを使用すると、改善がより顕著になります。
- クエリプランキャッシュのサポート: Neptune にクエリが送信されると、クエリ文字列が解析、最適化され、クエリプランに変換され、エンジンによって実行されます。アプリケーションは、多くの場合、さまざまな値でインスタンス化された一般的なクエリパターンによってバックアップされます。クエリプランキャッシュは、クエリプランをキャッシュすることで全体的なレイテンシーを低減し、このような繰り返しパターンの解析と最適化を回避できます。
- DISTINCT 集約クエリのパフォーマンス向上。
- null 可能な変数を含む結合のパフォーマンスが向上しました。
- id(node/relationship) 述語と等しくないクエリのパフォーマンスが向上しました。
- 日時機能の拡張サポート (を含める`DatetimeMillisecond`ことでラボモードでのみ有効になります`DatetimeMillisecond=enabled`。詳細については、「Neptune openCypher 実装 (Neptune Analytics および Neptune データベース 1.3.2.0 以降) での一時的なサポート」を参照してください。

エンジンリリース 1.3.2.0 から引き継がれた不具合の修正

全般的な機能強化

- Graphlytics バケットへのアクセスを検証するときに NeptuneML エラーメッセージを更新しました。

Gremlin の修正

- パス以外の寄与ステップにラベルが含まれているシナリオで、DFE クエリ変換でラベル情報が欠落している問題を修正しました。例:

```
g.withSideEffect('Neptune#useDFE', true).
  V().
  has('name', 'marko').
  has("name", TextP.regex("mark.*")).as("p1").
  not(out().has("name", P.within("peter"))).
  out().as('p2').
  dedup('p1', 'p2')
```

- 2 つの DFE フラグメントでクエリが実行され、最初のフラグメントが満たされないノードに最適化されている場合に発生する DFE クエリ変換の`NullPointerException`バグを修正しました。例:

```
g.withSideEffect('Neptune#useDFE', true).
  V().
  has('name', 'doesNotExists').
  has("name", TextP.regex("mark.*")).
  inject(1).
  V().
  out().
  has('name', 'vadas')
```

- クエリに by() モジュレータ ValueTraversal が含まれ、その入力が Map `InternalFailureException`の場合に、Neptune が をスローすることがあるバグを修正しました。例:

```
g.V().
  hasLabel("person").
  project("age", "name").by("age").by("name").
```

```
  order().by("age")
```

openCypher の修正

- UNWIND オペレーションが改善され (値のリストを個々の値に拡張するなど)、メモリ不足 (OOM) 状態を防ぐことができます。例:

```
MATCH (n)-->(m)
WITH collect(m) AS list
UNWIND list AS m
RETURN m, list
```

- ID が UNWIND を介して挿入される複数の MERGE オペレーションの場合のカスタム ID 最適化を修正しました。例:

```
UNWIND [{nid: 'nid1', mid: 'mid1'}, {nid: 'nid2', mid: 'mid2'}] as ids
MERGE (n:N {`~id`: ids.nid})
MERGE (m:M {`~id`: ids.mid})
```

- プロパティアクセスと双方向の関係を持つ複数のホップを持つ複雑なクエリを計画する際のメモリの急増を修正しました。例:

```
MATCH (person1:person)-[:likes]->(res)-[:partOf]->(group)-[:knows]-(:entity {name:
 'foo'}),
      (person1)-[:knows]->(person2)-[:likes]-(res2), (comment)-[:presentIn]->(:Group
 {name: 'barGroup'}),
      (person1)-[:commented]->(comment2:comment)-[:partOf]->(post:Post), (comment2)-
[:presentIn]->(:Group {name: 'fooGroup'}),
      (comment)-[:contains]->(info:Details)-[:CommentType]->(:CommentType {name:
 'Positive'}),
      (comment2)-[:contains]->(info2:Details)-[:CommentType]->(:CommentType {name:
 'Positive'})
WHERE datetime('2020-01-01T00:00') <= person1.addedAfter <=
 datetime('2023-01-01T23:59') AND comment.approvedBy = comment2.approvedBy
MATCH (comment)-[:contains]->(info3:Details)-[:CommentType]->(:CommentType {name:
 'Neutral'})
RETURN person1, group.name, info1.value,  post.ranking, info3.value
```

- null を変数別のグループとして持つ集計クエリを修正しました。例:

```
MATCH (n)
```

```
RETURN null AS group, sum(n.num) AS result
```

SPARQL の修正

- 多くのトリプルと大きなトークンを含む INSERT DATA などの大規模なクエリの解析時間を短縮するように SPARQL パーサーを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.3.2.1 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: `3.6.2`
- サポートされている最も新しいバージョンの Gremlin: `3.7.1`
- openCypher バージョン: `Neptune-9.0.20190305-1.0`
- SPARQL バージョン: `1.1`

エンジンリリース 1.3.2.1 へのアップグレードパス

このリリースへは、エンジンリリース 1.2.0.0 以降からアップグレードできます。

このリリースへのアップグレード

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.3.2.1 \
    --allow-major-version-upgrade \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.3.2.1 ^
    --allow-major-version-upgrade ^
    --apply-immediately
```

--apply-immediately の代わりに --no-apply-immediately と指定することができます。メジャーバージョンアップグレードを実行するには、 allow-major-version-upgrade パラメータが必要です。また、エンジンバージョンを含めるようにしてください。そうしないと、エンジンが別のバージョンにアップグレードされる可能性があります。

クラスターでカスタムクラスターパラメータグループを使用する場合は、必ずこのパラメータを含めて、それを指定してください。

```
    --db-cluster-parameter-group-name (name of the custom DB cluster parameter group)
```

同様に、クラスター内のインスタンスがカスタム DB のパラメータグループを使用している場合は、必ずこのパラメータを指定して、次のようになります。

```
    --db-instance-parameter-group-name (name of the custom instance parameter group)
```

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
>   We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the
> instance before
>   proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムおよび AWS Premium Support を通じて AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.3.2.0 (2024-06-10)

2024-06-10 の時点で、エンジンバージョン 1.3.2.0 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

> **Note**
>
> エンジンリリース 1.3.0.0 では、カスタムパラメータグループとカスタムクラスターパラメータグループに新しい形式が導入されました。そのため、1.3.0.0 より前のエンジンバー

ジョンからエンジンバージョン 1.3.0.0 以降にアップグレードする場合は、パラメータグループファミリー neptune1.3 を使用している既存のカスタムパラメータグループとカスタムクラスターパラメータグループをすべて再作成する必要があります。以前のリリースではパラメータグループファミリー neptune1 または neptune1.2 を使用していましたが、これらのパラメータグループはリリース 1.3.0.0 以降では動作しません。詳細については、「Amazon Neptune パラメータグループ」を参照してください。

Warning

skip または が内部WITH句でlimit使用され、パラメータ化されている場合、クエリプランキャッシュで問題が検出されました。この問題を回避するには、パラメータ化されたスキップや制限のサブ句を含むクエリを送信するQUERY:PLANCACHE "disabled"ときに、クエリヒントを追加します。または、値をクエリにハードコーディングすることもできます。詳細については、クエリプランキャッシュの問題の軽減を参照してください。

このエンジンリリースの改善点

全般的な機能強化

- 暗号スイート TLS_AES_128_GCM_SHA256 および TLS_AES_256_GCM_SHA384 を含む TLS バージョン 1.3 のサポート。TLS 1.3 はオプションです。TLS 1.2 はまだ最小です。

Gremlin の改善

- TinkerPop 3.7.x アップグレード
  - Gremlin 言語の大規模な拡張を提供します。
    - 文字列、リスト、日付を処理するための新しいステップ。
    - mergeV() ステップで基数を指定するための新しい構文。
    - union() を開始ステップとして使用できるようになりました。
    - 3.7.x の変更点の詳細については、TinkerPop アップグレードドキュメント を参照してください。
  - Java 用のクライアント Gremlin 言語ドライバーをアップグレードするときは、シリアライザクラスの名前変更の一部が から解除されていることに注意してください。指定する場合は、設定ファイルとコードでパッケージとクラスの命名を更新する必要があります。

- StrictTimeoutValidation (を含めるStrictTimeoutValidationことでラボモードで有効にした場合にのみStrictTimeoutValidation=enabled): StrictTimeoutValidationパラメータの値が の場合enabled、リクエストオプションまたはクエリヒントとして指定されたクエリごとのタイムアウト値は、パラメータグループでグローバルに設定された値を超えることはできません。このような場合、Neptune は をスローしますInvalidParameterException。この設定は、値が の場合、/statusエンドポイントのレスポンスで確認できます。Neptune バージョン 1.3.2.0 ではdisabled、このパラメータのデフォルト値は ですDisabled。

openCypher の改善

- 低レイテンシークエリとスループットパフォーマンスの向上: 低レイテンシー openCypher クエリの全体的なパフォーマンスの向上。新しいバージョンでは、このようなクエリのスループットも向上します。パラメータ化されたクエリを使用すると、改善がより顕著になります。
- クエリプランキャッシュのサポート: Neptune にクエリが送信されると、クエリ文字列が解析、最適化され、クエリプランに変換され、エンジンによって実行されます。アプリケーションは、多くの場合、さまざまな値でインスタンス化された一般的なクエリパターンによってバックアップされます。クエリプランキャッシュは、クエリプランをキャッシュすることで全体的なレイテンシーを低減し、このような繰り返しパターンの解析と最適化を回避できます。
- DISTINCT 集約クエリのパフォーマンス向上。
- null 可能な変数を含む結合のパフォーマンスが向上しました。
- id(node/relationship) 述語と等しくないクエリのパフォーマンスが向上しました。
- 日時機能の拡張サポート (を含めるDatetimeMillisecondことでラボモードでのみ有効になりますDatetimeMillisecond=enabled。詳細については、「Neptune openCypher 実装 (Neptune Analytics および Neptune データベース 1.3.2.0 以降) での一時的なサポート」を参照してください。

このエンジンリリースで修正された不具合

全般的な機能強化

- Graphlytics バケットへのアクセスを検証するときに NeptuneML エラーメッセージを更新しました。

Gremlin の修正

- パス以外の寄与ステップにラベルが含まれているシナリオで、DFE クエリ翻訳でラベル情報が欠落している問題を修正しました。例:

```
g.withSideEffect('Neptune#useDFE', true).
  V().
  has('name', 'marko').
  has("name", TextP.regex("mark.*")).as("p1").
  not(out().has("name", P.within("peter"))).
  out().as('p2').
  dedup('p1', 'p2')
```

- 2 つの DFE フラグメントでクエリが実行され、最初のフラグメントが満たされないノードに最適化された場合に発生する DFE クエリ変換の`NullPointerException`バグを修正しました。例:

```
g.withSideEffect('Neptune#useDFE', true).
  V().
  has('name', 'doesNotExists').
  has("name", TextP.regex("mark.*")).
  inject(1).
  V().
  out().
  has('name', 'vadas')
```

- クエリに by() モジュレータ ValueTraversal が含まれ、その入力が Map `InternalFailureException`の場合に Neptune が をスローすることがあるバグを修正しました。例:

```
g.V().
  hasLabel("person").
  project("age", "name").by("age").by("name").
  order().by("age")
```

openCypher の修正

- UNWIND オペレーション (値のリストを個々の値に拡張するなど) が改善され、メモリ不足 (OOM) 状態を防止できるようになりました。例:

```
MATCH (n)-->(m)
```

```
WITH collect(m) AS list
UNWIND list AS m
RETURN m, list
```

- ID が UNWIND を介して挿入される複数の MERGE オペレーションの場合のカスタム ID 最適化を修正しました。例:

```
UNWIND [{nid: 'nid1', mid: 'mid1'}, {nid: 'nid2', mid: 'mid2'}] as ids
MERGE (n:N {`~id`: ids.nid})
MERGE (m:M {`~id`: ids.mid})
```

- プロパティアクセスと双方向の関係を持つ複数のホップを持つ複雑なクエリを計画する際のメモリの急増を修正しました。例:

```
MATCH (person1:person)-[:likes]->(res)-[:partOf]->(group)-[:knows]-(:entity {name:
 'foo'}),
      (person1)-[:knows]->(person2)-[:likes]-(res2), (comment)-[:presentIn]->(:Group
 {name: 'barGroup'}),
      (person1)-[:commented]->(comment2:comment)-[:partOf]->(post:Post), (comment2)-
[:presentIn]->(:Group {name: 'fooGroup'}),
      (comment)-[:contains]->(info:Details)-[:CommentType]->(:CommentType {name:
 'Positive'}),
      (comment2)-[:contains]->(info2:Details)-[:CommentType]->(:CommentType {name:
 'Positive'})
WHERE datetime('2020-01-01T00:00') <= person1.addedAfter <=
 datetime('2023-01-01T23:59') AND comment.approvedBy = comment2.approvedBy
MATCH (comment)-[:contains]->(info3:Details)-[:CommentType]->(:CommentType {name:
 'Neutral'})
RETURN person1, group.name, info1.value,  post.ranking, info3.value
```

- null を変数別のグループとして持つ集計クエリを修正しました。例:

```
MATCH (n)
RETURN null AS group, sum(n.num) AS result
```

SPARQL の修正

- 多くのトリプルと大きなトークンを含む INSERT DATA などの大規模なクエリの解析時間を短縮するために SPARQL パーサーを修正しました。

クエリプランキャッシュの問題の軽減

バージョン 1.3.2.0 では、 skipまたは limit が内部WITH句で使用され、パラメータ化されている場合、クエリプランキャッシュで問題が検出されました。例:

```
MATCH (n:Person)
WHERE n.age > $age
WITH n skip $skip LIMIT $limit
RETURN n.name, n.age

parameters={"age": 21, "skip": 2, "limit": 3}
```

この場合、最初の計画からのスキップと制限のパラメータ値が後続のクエリにも適用され、予期しない結果が発生します。

緩和策

この問題を回避するには、パラメータ化されたスキップや制限のサブ句を含むクエリを送信するQUERY:PLANCACHE "disabled"ときに、クエリヒントを追加します。または、値をクエリにハードコーディングすることもできます。

オプション 1: クエリヒントを使用してプランキャッシュを無効にする：

```
Using QUERY:PLANCACHE "disabled"
MATCH (n:Person) WHERE n.age > $age
WITH n skip $skip LIMIT $limit
RETURN n.name, n.age

parameters={"age": 21, "skip": 2, "limit": 3}
```

オプション 2: スキップと制限にハードコードされた値を使用する：

```
MATCH (n:Person)
WHERE n.age > $age
WITH n skip 2 LIMIT 3
RETURN n.name, n.age

parameters={"age": 21}
```

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.3.2.0 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: `3.6.2`
- サポートされている最も新しいバージョンの Gremlin: `3.7.1`
- openCypher バージョン: `Neptune-9.0.20190305-1.0`
- SPARQL バージョン: `1.1`

エンジンリリース 1.3.2.0 へのアップグレードパス

このリリースへは、エンジンリリース 1.2.0.0 以降からアップグレードできます。

このリリースへのアップグレード

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.3.2.0 \
    --allow-major-version-upgrade \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.3.2.0 ^
    --allow-major-version-upgrade ^
    --apply-immediately
```

`--apply-immediately` の代わりに `--no-apply-immediately` と指定することができます。メジャーバージョンアップグレードを実行するには、 allow-major-version-upgrade パラメータが必要

です。また、エンジンバージョンを含めるようにしてください。そうしないと、エンジンが別のバージョンにアップグレードされる可能性があります。

クラスターでカスタムクラスターパラメータグループを使用する場合は、必ずこのパラメータを含めて、それを指定してください。

```
--db-cluster-parameter-group-name (name of the custom DB cluster parameter group)
```

同様に、クラスター内のインスタンスがカスタム DB のパラメータグループを使用している場合は、必ずこのパラメータを指定して、次のようになります。

```
--db-instance-parameter-group-name (name of the custom instance parameter group)
```

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後

に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
>   We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the
> instance before
>   proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムおよび AWS Premium Support を通じて AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.3.1.0 (2024-03-06)

2024-03-06 の時点で、エンジンバージョン 1.3.1.0 が一般的に導入されています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

> **Note**
>
> エンジンリリース 1.3.0.0 では、カスタムパラメータグループとカスタムクラスターパラメータグループに新しい形式が導入されました。そのため、1.3.0.0 より前のエンジンバージョンからエンジンバージョン 1.3.0.0 以降にアップグレードする場合は、パラメータグループファミリー `neptune1.3` を使用している既存のカスタムパラメータグループとカスタムクラスターパラメータグループをすべて再作成する必要があります。以前のリリースではパラメータグループファミリー `neptune1` または `neptune1.2` を使用していましたが、これらのパラメータグループはリリース 1.3.0.0 以降では動作しません。詳細については、「Amazon Neptune パラメータグループ」を参照してください。

このエンジンリリースの改善点

全般的な機能強化

- Neptune は、プロファイル/説明に表示される警告を改善しました。
- TLS ネゴシエーション中に使用されるデフォルトの名前付きグループから、古い NIST EC 曲線を削除しました。削除された曲線は sect409k1、sect409r1、および sect571k1 です。

グレムリンの改良

- DFE 統計計算を改善して、サーバーレスインスタンスの NCU が非常に高くなるのを防ぎました。
- WITHIN の Gremlin のパフォーマンスが向上しました。

このエンジンリリースで修正された不具合

Gremlin の修正

- Gremlin DFE クエリプランにさまざまな改良が加えられました。
- オプションでトラバーサルを使用する Gremlin クエリのバグが修正されました。たとえば、`g.V () .hasLabel ('person') .group () .by (__.in ('friend') .id () .fold ()) `という形式のクエリでは、フレンドエッジのないユーザーはグループ化されませんでした。
- DFEエンジンを使用して実行すると、byモジュレーター内の合体ステップを含むGremlinクエリがエラーを返すバグを修正しました。
- Gremlin セッションで実行されている読み取り専用クエリがリードレプリカに接続されているときに機能しないバグを修正しました。
- Gremlin の最初のウェブソケット接続リクエストが成功した際に IAM ARN が監査ログに存在しなかったバグを修正しました。
- ステップを合体させ、DFE でステップカバレッジを特定します。
- DFE プラン全体の特性セット最適化。

openCypher の修正

- OpenCypher SET 句のバグが修正され、非変数式 (例:match (n: Test) セット (n.prop = 2 の後に n が終了する場合) に設定できるようになりました。.prop = 3 は.prop を返すようになります。
- 集計と順序付けに関する OpenCypher クエリの失敗に関するバグが修正されました。

- 静的マップを含む大きなリストの UNWIND を改善しました。
- 重複する値を持つカスタム ID を使用する OpenCypher MERGE クエリのバグが修正されました。

SPARQL の修正

- オプションパターンの変数スコープに関する SPARQL のバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.3.1.0 にアップグレードする前に、プロジェクトが以下のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: `3.6.2`
- サポートされている最も新しいバージョンの Gremlin: `3.6.5`
- openCypher バージョン: `Neptune-9.0.20190305-1.0`
- SPARQL バージョン: `1.1`

エンジンリリース 1.3.1.0 へのアップグレードパス

このリリースへは、エンジンリリース 1.2.0.0 以降からアップグレードできます。

このリリースへのアップグレード

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.3.1.0 \
    --allow-major-version-upgrade \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.3.1.0 ^
    --allow-major-version-upgrade ^
    --apply-immediately
```

--apply-immediately の代わりに --no-apply-immediately と指定することができます。メジャーバージョンアップグレードを実行するには、 allow-major-version-upgrade このパラメータが必要です。また、エンジンバージョンを含めるようにしてください。そうしないと、エンジンが別のバージョンにアップグレードされる可能性があります。

クラスターでカスタムクラスターパラメータグループを使用する場合は、必ずこのパラメータを含めて、それを指定してください。

```
    --db-cluster-parameter-group-name (name of the custom DB cluster parameter group)
```

同様に、クラスター内のインスタンスがカスタム DB のパラメータグループを使用している場合は、必ずこのパラメータを指定して、次のようになります。

```
    --db-instance-parameter-group-name (name of the custom instance parameter group)
```

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。質問や懸念がある場合は、 AWS AWS コミュニティフォーラムやプレミアムSupport を通じてSupport チームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.3.0.0 (2023-11-15)

2023 年 11 月 15 日現在、エンジンバージョン 1.3.0.0 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

> **Note**
>
> エンジンリリース 1.3.0.0 では、カスタムパラメータグループとカスタムクラスターパラメータグループに新しい形式が導入されました。そのため、1.3.0.0 より前のエンジンバー

ジョンからエンジンバージョン 1.3.0.0 以降にアップグレードする場合は、パラメータグループファミリー neptune1.3 を使用している既存のカスタムパラメータグループとカスタムクラスターパラメータグループをすべて再作成する必要があります。以前のリリースではパラメータグループファミリー neptune1 または neptune1.2 を使用していましたが、これらのパラメータグループはリリース 1.3.0.0 以降では動作しません。詳細については、「Amazon Neptune パラメータグループ」を参照してください。

このエンジンリリースの新機能

- Neptune データ API をリリースしました。

  Amazon Neptune データ API は、データの読み込み、クエリの実行、データ照会、機械学習など、Neptune の 40 を超えるデータオペレーションを SDK でサポートしています。Neptune の 3 つすべてのクエリ言語 (Gremlin、openCypher、SPARQL) をサポートしており、すべての SDK 言語で使用できます。API リクエストに自動的に署名し、Neptune のアプリケーションへの統合を大幅に簡素化します。
- OpenSearchサーバーレスと Neptune の統合のサポートが追加されました。

このエンジンリリースの改良点

Neptune エンジンアップデートの改善点

Neptune のエンジンアップデートのリリース方法が変わり、アップデートプロセスをより細かく制御できるようになりました。Neptune では、互換性を破る変更がない場合にパッチの代わりにマイナーバージョンをリリースし、これを AutoMinorVersionUpgrade インスタンスフィールドを使用して制御できるようになりました。RDS-EVENT-0156 イベントにサブスクライブすることで、マイナーバージョンに関する通知を受け取ることができます。

これらの変更の詳細については、「Amazon Neptune DB クラスターのメンテナンス」を参照してください。

転送中の暗号化の改善

Neptune では、以下の暗号スイートがサポートされなくなりました。

- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA

Neptune では、TLS 1.2 を使用した、以下の強力な暗号スイートのみがサポートされます。

- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256

Gremlin の改善

- 以下の Gremlin ステップに対するサポートを DFE エンジンに追加しました。
  - FoldStep
  - GroupStep
  - GroupCountStep
  - TraversalMapStep
  - UnfoldStep
  - LabelStep
  - PropertyKeyStep
  - PropertyValueStep
  - AndStep
  - OrStep
  - ConstantStep
  - CountGlobalStep
- Gremlin DFE クエリプランを最適化し、by( ) 変調の使用時に頂点全体がスキャンされないようにしました。
- 低カーディナリティおよび低レイテンシーのクエリのパフォーマンスが大幅に向上しました。
- フィルター述語の DFE サポートが追加されました。 TinkerPop Or
- 次のようなクエリに対して、同じキーでのフィルターのトラバーサルの DFE サポートを改善しました。

```
g.withSideEffect("Neptune#useDFE", true)
 .V()
 .has('name', 'marko')
 .and(
   or(
```

```
    has('name', eq("marko")),
    has('name', eq("vardas"))
  )
)
```

- fail() ステップのエラー処理を改善しました。

openCypher の改善

- 低カーディナリティおよび低レイテンシーのクエリのパフォーマンスが大幅に向上しました。
- クエリに多数のノードタイプが含まれる場合のクエリプランニングのパフォーマンスが向上しました。
- すべての VLP クエリのレイテンシーが短縮されました。
- 単一ノードパターンクエリの冗長なパイプライン結合を削除することでパフォーマンスが向上しました。
- 次のようなサイクルを伴うマルチホップパターンを含むクエリのパフォーマンスが向上しました。

```
MATCH (n)-->()-->()-->(m)
RETURN n m
```

SPARQL の改善

- 新しい SPARQL 演算子として PipelineHashIndexJoin を導入しました。
- SPARQL クエリの URI 検証のパフォーマンスが向上しました。
- 辞書用語のバッチ解決により、SPARQL 全文検索クエリのパフォーマンスが向上しました。

このエンジンリリースで修正された不具合

Gremlin の修正

- DFE エンジンでネイティブに処理されないステップの子トラバーサルに述語を含むクエリについて、Gremlin クエリステータスエンドポイントをチェックするときに、トランザクションリークが発生する Gremlin のバグを修正しました。
- by() トラバーサル時の DFE エンジンで valueMap() が最適化されないという Gremlin のバグを修正しました。

- UnionStep に付加したステップラベルが子トラバーサルの最後のパス要素にそれぞれ伝播されないという Gremlin のバグを修正しました。
- TinkerPop クエリに含まれるステップが多すぎてクエリが失敗し、クリーンアップされないという Gremlin のバグを修正しました。
- NullPointerException が mergeV ステップや mergeE ステップにスローされるという Gremlin のバグを修正しました。
- 文字列出力の一部にスペース文字が含まれていると order() によって正しくソートされない Gremlin のバグを修正しました。
- DFE エンジンで valueMap ステップの処理時に発生する Gremlin の正確性の問題を修正しました。
- GroupStep または GroupCountStep をキートラバーサルにネストする際に発生する Gremlin の正確性の問題を修正しました。

openCypher の修正

- NULL 文字に関するエラー処理に伴う OpenCypher のバグを修正しました。
- openCypher ボルトトランザクション処理に伴うバグを修正しました。

SPARQL の修正

- 再帰関数内の値が適切に解決されないという SPARQL のバグを修正しました。
- VALUES 句を使用して大量の値を挿入すると、パフォーマンスが低下するという SPARQL のバグを修正しました。
- 言語タグ付きリテラルでの REGEX 演算子の呼び出しが成功しないという SPARQL のバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.3.0.0 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: 3.6.2
- サポートされている最も新しいバージョンの Gremlin: 3.6.4
- openCypher バージョン: Neptune-9.0.20190305-1.0

- SPARQL バージョン: 1.1

エンジンリリース 1.3.0.0 へのアップグレードパス

このリリースへは、エンジンリリース 1.2.0.0 以降からアップグレードできます。

このリリースへのアップグレード

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.3.0.0 \
    --allow-major-version-upgrade \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.3.0.0 ^
    --allow-major-version-upgrade ^
    --apply-immediately
```

`--apply-immediately` の代わりに `--no-apply-immediately` と指定することができます。メジャーバージョンアップグレードを実行するには、 allow-major-version-upgrade このパラメータが必要です。また、エンジンバージョンを含めるようにしてください。そうしないと、エンジンが別のバージョンにアップグレードされる可能性があります。

クラスターでカスタムクラスターパラメータグループを使用する場合は、必ずこのパラメータを含めて、それを指定してください。

```
    --db-cluster-parameter-group-name (name of the custom DB cluster parameter group)
```

同様に、クラスター内のインスタンスがカスタム DB のパラメータグループを使用している場合は、必ずこのパラメータを指定して、次のようになります。

```
--db-instance-parameter-group-name (name of the custom instance parameter group)
```

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
> ```

```
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。質問や懸念がある場合は、 AWS AWS コミュニティフォーラムやプレミアムSupport を通じてSupport チームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.2.1.1 (2024-03-11)

2024-03-11 以降、エンジンバージョン 1.2.1.1 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

Note

1.2.0.0 より前のエンジンバージョンからアップグレードを行う場合:

- エンジンリリース 1.2.0.0 では、カスタムパラメータグループとカスタムクラスターパラメータグループに新しい形式が導入されました。そのため、1.2.0.0 より前のエンジンバージョンからエンジンバージョン 1.2.0.0 以降にアップグレードする場合は、パラメータグループファミリー neptune1.2 を使用している既存のカスタムパラメータグループとカスタムクラスターパラメータグループをすべて再作成する必要があります。以前のリリースではパラメータグループファミリー neptune1 が使用されていましたが、それらのパラメータグループはリリース 1.2.0.0 以降では動作しません。詳細については、「Amazon Neptune パラメータグループ」を参照してください。
- エンジンリリース 1.2.0.0 では、元に戻すログの新しい形式も導入されました。そのため、1.2.0.0 より前のバージョンからのアップグレードを開始する前に、以前のエンジンバージョンで作成された元に戻すログをすべて消去し、UndoLogsListSize CloudWatch メトリクスをゼロにする必要があります。更新を開始しようとしたときに UNDO ログレコードが多すぎる (200,000 以上) 場合、UNDO ログのパージが完了するのを待っている間にアップグレードがタイムアウトすることがあります。

パージが行われるクラスターのライターインスタンスをアップグレードすることで、パージの速度を上げることができます。アップグレードを試みる前にこれを行うと、開始前に UNDO ログの数を減らすことができます。ライターのサイズを 24XL インスタンスタイプに増やすと、パージ率が 1 時間あたり 100 万レコードを超えることがあります。

UndoLogsListSize CloudWatch メトリクスが非常に大きい場合は、サポートケースを開くと、メトリクスを減らすための追加の戦略を検討するのに役立ちます。

- 最後に、リリース 1.2.0.0 には、IAM 認証で Bolt プロトコルを使用していた以前のコードに影響する重大な変更がありました。リリース 1.2.0.0 以降、Bolt には IAM 署名用のリソースパスが必要です。Java では、リソースパスの設定は以下のようになります: `request.setResourcePath("/openCypher"));`。その他の言語では、`/openCypher` をエンドポイント URI に追加できます。例については、「Bolt プロトコルの使用」を参照してください。

このエンジンリリースの改良点

全般的な改善点

Neptune は、プロファイル/説明に表示される警告を改善しました。

Gremlin の改善

- DFE 統計の計算が改善され、サーバーレスインスタンスの NCUs。
- WITHIN の Gremlin パフォーマンスが向上しました。

このエンジンリリースで修正された不具合

Gremlin の修正

- Gremlin DFE エンジンクエリプランの順序付けによるバグ修正。
- 最初に として報告されたときの Gremlin out-of-memory エラーによるバグ修正 InternalFailureException。
- 最初の WebSocket 接続リクエストが成功した場合に、監査ログに IAM ARN が存在しないバグ修正。

- TinkerPop セッション内のクエリがすべて読み取り専用でリーダーインスタンスに接続されていても、セッション内のクエリが失敗した場合の、セッションが有効になっている Gremlin クエリのバグ修正。

openCypher の修正

- openCypher SET 句のバグ修正により、非可変式 (つまり、match(n:TEST) set(n.prop = 2 のときの大文字と小文字、n end).prop = 3 は n.prop を返します。
- 集計と順序を含む openCypher クエリが失敗した場合のバグ修正。
- 静的マップを含む大きなリストの UNWIND が改善されました。
- 重複する値を持つカスタム ID を使用して openCypher MERGE クエリをバグ修正します。

SPARQL の修正

- SPARQL DFE クエリプランナーのバグ修正。
- BIND キーワードと OPTIONAL キーワードとともに使用する場合の SPARQL のバグ修正。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.2.1.1 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: `3.6.2`
- サポートされている最も新しいバージョンの Gremlin: `3.6.2`
- openCypher バージョン: `Neptune-9.0.20190305-1.0`
- SPARQL バージョン: `1.1`

エンジンリリース 1.2.1.1 へのアップグレードパス

このリリースへは、エンジンリリース 1.2.0.0 以降からアップグレードできます。

このリリースへのアップグレード

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コン

ソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.1.1 \
    --allow-major-version-upgrade \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.1.1 ^
    --allow-major-version-upgrade ^
    --apply-immediately
```

`--apply-immediately` の代わりに `--no-apply-immediately` と指定することができます。メジャーバージョンアップグレードを実行するには、 allow-major-version-upgrade パラメータが必要です。また、エンジンバージョンを含めるようにしてください。そうしないと、エンジンが別のバージョンにアップグレードされる可能性があります。

クラスターでカスタムクラスターパラメータグループを使用する場合は、必ずこのパラメータを含めて、それを指定してください。

```
    --db-cluster-parameter-group-name (name of the custom DB cluster parameter group)
```

同様に、クラスター内のインスタンスがカスタム DB のパラメータグループを使用している場合は、必ずこのパラメータを指定して、次のようになります。

```
    --db-instance-parameter-group-name (name of the custom instance parameter group)
```

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the
> instance before
>   proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムおよび AWS Premium Support を通じて AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.2.1.0 (2023-03-08)

2023 年 3 月 8 日現在、エンジンバージョン 1.2.1.0 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

> **Note**
>
> 1.2.0.0 より前のエンジンバージョンからアップグレードを行う場合:
>
> - エンジンリリース 1.2.0.0 では、カスタムパラメータグループとカスタムクラスターパラメータグループに新しい形式が導入されました。そのため、1.2.0.0 より前のエンジンバージョンからエンジンバージョン 1.2.0.0 以降にアップグレードする場合は、パラメータグループファミリー `neptune1.2` を使用している既存のカスタムパラメータグループとカスタムクラスターパラメータグループをすべて再作成する必要があります。以前のリリースではパラメータグループファミリー `neptune1` が使用されていましたが、それらのパラメータグループはリリース 1.2.0.0 以降では動作しません。詳細については、「Amazon Neptune パラメータグループ」を参照してください。
> - エンジンリリース 1.2.0.0 では、UNDO ログの新しい形式も導入されました。そのため、1.2.0.0 より前のバージョンからのアップグレードを開始する前に、以前のエンジンバージョンで作成された UNDO ログをすべてパージし、`UndoLogsListSize` CloudWatch メトリクスをゼロにする必要があります。更新を開始しようとしたときに UNDO ログレコードが多すぎる (200,000 以上) 場合、UNDO ログのパージが完了するのを待っている間にアップグレードがタイムアウトすることがあります。
>
>   パージが行われるクラスターのライターインスタンスをアップグレードすることで、パージの速度を上げることができます。アップグレードを試みる前にこれを行うと、開始前に UNDO ログの数を減らすことができます。ライターのサイズを 24XL インスタンスタイプに増やすと、パージ率が 1 時間あたり 100 万レコードを超えることがあります。
>
>   `UndoLogsListSize` CloudWatch メトリクスが非常に大きい場合、サポートケースを開くと、それを停止するための追加の戦略を検討するのに役立つ場合があります。
> - 最後に、リリース 1.2.0.0 には、IAM 認証で Bolt プロトコルを使用していた以前のコードに影響する重大な変更がありました。リリース 1.2.0.0 以降、Bolt には IAM 署名用のリソースパスが必要です。Java では、リソースパスの設定は以下のようにな

ります: `request.setResourcePath("/openCypher"));`。その他の言語では、/openCypher をエンドポイント URI に追加できます。例については、「Bolt プロトコルの使用」を参照してください。

このリリースの後続のパッチリリース

- リリース: 1.2.1.0.R2 (2023-05-02)
- リリース: 1.2.1.0.R3 (2023-06-13)
- リリース: 1.2.1.0.R4 (2023-06-10)
- リリース: 1.2.1.0.R5 (2023-09-02)
- リリース: 1.2.1.0.R6 (2023-09-12)
- リリース: 1.2.1.0.R7 (2023-10-06)

このエンジンリリースの新機能

- TinkerPop 3.6.2 のサポートが追加されました。これにより、新しい `mergeV()`、`mergeE()`、`element()`、および `fail()` ステップなど、多くの新しい Gremlin 機能が追加されました。`mergeV()` および `mergeE()` ステップは、アップサートのような操作を実行するための待望の宣言型オプションを提供するため、特に注目すべき点です。これにより、既存のコードパターンが大幅に簡略化され、Gremlin が読みやすくなるはずです。3.6.x バージョンでは、正規表現述語、`Map` を取る `property()` ステップへの新しいオーバーロード、`by()` 変調動作の大幅な改訂も追加されました。変調動作の大幅な改訂により、これを使用するすべてのステップではるかに一貫性が保たれています。

  バージョン 3.6 の変更点とアップグレード時に考慮すべき点については、TinkerPop の変更ログとアップグレードのページを参照してください。

  `fold().coalesce(unfold(), <mutate>)` を条件付き挿入に使用している場合は、こことここで説明されている新しい `mergeV/E()` 構文に移行することをお勧めします。Neptune では、`Merge` については `Coalesce` よりも狭いロックパターンを使用するため、同時変更例外 (CME) を減らすことができます。

  今回の TinkerPop リリースで利用できる新機能の詳細については、Stephen Mallette のブログ「Amazon Neptune での Apache TinkerPop 3.6.x の新機能の探求」を参照してください。

- 第 3 世代の Intel Xeon スケーラブルプロセッサを搭載した R6i インスタンスタイプのサポートを追加しました。これらはメモリ集約的なワークロードに最適であり、同等の R5 インスタンスタイプよりも計算性能/価格パフォーマンスが最大 15% 向上し、vCPU あたりのメモリ帯域幅が最大 20% 高くなります。
- プロパティグラフと RDF グラフの両方にグラフサマリー API エンドポイントが追加され、グラフに関する概要レポートをすばやく取得できるようになりました。

  プロパティ (PG) グラフについて、グラフサマリー API は、ノードおよびエッジラベルとプロパティキーの読み取り専用リストを、ノード、エッジ、プロパティの数とともに返します。RDF グラフでは、クラスと述語キーのリストに加え、クワッド、主語、述語の数も表示されます。

  新しいグラフサマリー API に伴い、以下の変更が行われました。

  - 新しい GetGraphSummary データプレーンアクションが追加されました。
  - 廃止された `sparql/statistics` エンドポイントに代わる新しい `rdf/statistics` エンドポイントが追加されました。
  - グラフのサマリー情報と混同しないように、統計ステータスレスポンスの `summary` フィールドの名前が`signatureInfo` に変更されました。JSON レスポンスでは、以前のエンジンバージョンも引き続き `summary` を使用します。
  - 統計ステータスレスポンスの `date` フィールドの精度が分単位からミリ秒単位に変更されました。以前の形式は `2020-05-07T23:13Z` (分精度) でしたが、新しい形式は `2023-01-24T00:47:43.319Z` (ミリ秒精度) です。どちらも ISO 8601 に準拠していますが、日付の解析方法によっては、この変更により既存のコードが壊れる可能性があります。
  - DFE エンジンの統計情報を取得できる新しい `%statistics` ラインマジックがワークベンチに追加されました。
  - グラフの概要情報を取得できる新しい `%summary` ラインマジックがワークベンチに追加されました。
- 指定したしきい値よりも実行に時間がかかったクエリをログに記録するスロークエリロギングが追加されました。スロークエリロギングを有効化および制御するには、neptune_enable_slow_query_log と neptune_slow_query_log_threshold という 2 つの新しい動的パラメーターを使用します。
- 2 つの動的パラメータ、すなわち、新しいクラスターパラメータ neptune_enable_slow_query_log と neptune_slow_query_log_threshold のサポートが追加されました。動的パラメータを変更すると、インスタンスを再起動しなくてもすぐに有効になります。
- 指定されたキーをマップから削除し、生成された新しいマップを返す Neptune 固有の openCypher removeKeyFromMap() 関数が追加されました。

このエンジンリリースの改良点

- Gremlin DFE のサポートがローカルスコープの `limit` ステップに拡張されました。
- DFE エンジンでの Gremlin `DedupGlobalStep` の `by()` モジュレーションサポートが追加されました。
- Gremlin `SelectStep` および `SelectOneStep` の DFE サポートが追加されました。
- `repeat`、`coalesce`、`store`、`aggregate` など、さまざまな Gremlin 演算子のパフォーマンスの向上と正確性の修正を行いました。
- `MERGE` と `OPTIONAL MATCH` を含む openCypher クエリのパフォーマンスが向上しました。
- リテラル値のマップのリストの `UNWIND` を含む openCypher クエリのパフォーマンスが向上しました。
- `id` の `IN` フィルターの付いた openCypher クエリのパフォーマンスが向上しました。例:

```
MATCH (n) WHERE id(n) IN ['1', '2', '3'] RETURN n
```

- BASE ステートメントを使用して SPARQL クエリのベース IRI を指定する機能が追加されました (「クエリと更新のためのデフォルトベース IRI」を参照)。
- Gremlin と openCypher のエッジのみの一括ロードのロード処理待ち時間を短縮しました。
- 再開の試みが失敗するまでの Amazon S3 接続の問題による長時間の待ち時間を回避するため、Neptune の再起動時に一括読み込みが非同期で再開されるようになりました。
- describeMode クエリのヒントが `"CBD"` (簡潔で限定された説明) に設定されていて、多数の空白ノードを含む SPARQL DESCRIBE クエリの処理が改善されました。

このエンジンリリースで修正された不具合

- Bolt と SPARQL-JSON でクエリが NULL 値ではなく文字列 `"null"` を返す openCypher のバグを修正しました。
- リスト要素に指定された値ではなく NULL 値が生成される、リスト内包表記の openCypher のバグを修正しました。
- バイト値が正しくシリアル化されない openCypher のバグを修正しました。
- 子トラバーサル内で入力が頂点へのエッジトラバーサルであった場合に発生していた `UnionStep` の Gremlin のバグを修正しました。
- `UnionStep` に関連付けられたステップラベルが各子トラバーサルの最後のステップに正しく伝播されない Gremlin のバグを修正しました。

- `repeat` ステップの後にラベルがある dedup ステップで、dedup ステップに添付されたラベルがクエリで今後使用できなくなる Gremlin のバグを修正しました。
- union ステップ内の `repeat` ステップを変換すると内部エラーで失敗する Gremlin のバグを修正しました。
- Tinkerpop にフォールバックして `limit` を非結合ステップの子トラバーサルとして持つ DFE クエリの Gremlin の正確性の問題を修正しました。次のような形式のクエリが影響を受けます。

```
g.withSideEffect('Neptune#useDFE', true).V().as("a").select("a").by(out().limit(1))
```

- SPARQL GRAPH パターンが FROM NAMED 句によって指定されるデータセットを考慮しないという SPARQL のバグを修正しました。
- SPARQL DESCRIBE に一部の FROM や FROM NAMED 句があると、デフォルトグラフのデータが常に正しく使用されず、例外が発生することがあった SPARQL のバグを修正しました。「SPARQL DESCRIBE のデフォルトグラフに対する動作」を参照してください。
- SPARQL のバグを修正し、NULL 文字が拒否された場合に正しい例外メッセージが返されるようになりました。
- PipelinedHashIndexJoin 演算子を含むプランに影響を及ぼしていた SPARQL の説明バグを修正しました。
- 定数値を返すクエリが送信されると内部エラーがスローされる openCypher のバグを修正しました。
- デッドロック検出ロジックで、エンジンが応答しなくなることがあった問題を修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.2.1.0 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: `3.6.2`
- サポートされている最も新しいバージョンの Gremlin: `3.6.2`
- openCypher バージョン: `Neptune-9.0.20190305-1.1`
- SPARQL バージョン: `1.1`

エンジンリリース 1.2.1.0 へのアップグレードパス

1.1.0.0 以上の以前の Neptune エンジンリリースから、このリリースに手動でアップグレードできます。

> **Note**
>
> エンジンリリース 1.2.0.0 以降、1.2.0.0 より前のエンジンバージョンで使用していたすべてのカスタムパラメータグループとカスタムクラスターパラメータグループは、パラメータグループファミリー neptune1.2 を使用して再作成する必要があります。以前のリリースではパラメータグループファミリー neptune1 が使用されていましたが、これらのパラメータグループは 1.2.0.0 以降のリリースでは動作しなくなります。詳細については、「Amazon Neptune パラメータグループ」を参照してください。

このメジャーバージョンリリースに自動的にアップグレードされることはありません。

このリリースへのアップグレード

Amazon Neptune 1.2.1.0 が公開されました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.1.0 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.1.0 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
> ```

```
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.2.1.0.R7 (2023-10-06)

2023 年 10 月 6 日現在、エンジンバージョン 1.2.1.0.R7 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

Note

1.2.0.0 より前のエンジンバージョンからアップグレードを行う場合:

- エンジンリリース 1.2.0.0 では、カスタムパラメータグループとカスタムクラスターパラメータグループに新しい形式が導入されました。そのため、1.2.0.0 より前のエンジンバージョンからエンジンバージョン 1.2.0.0 以降にアップグレードする場合は、パラメータグループファミリー `neptune1.2` を使用している既存のカスタムパラメータグループとカスタムクラスターパラメータグループをすべて再作成する必要があります。以前のリリースではパラメータグループファミリー `neptune1` が使用されていましたが、それらのパラメータグループはリリース 1.2.0.0 以降では動作しません。詳細については、「Amazon Neptune パラメータグループ」を参照してください。
- エンジンリリース 1.2.0.0 では、UNDO ログの新しい形式も導入されました。そのため、1.2.0.0 より前のバージョンからのアップグレードを開始する前に、以前のエンジンバージョンで作成された UNDO ログをすべてパージし、`UndoLogsListSize` CloudWatch メトリクスをゼロにする必要があります。更新を開始しようとしたときに UNDO ログレコードが多すぎる (200,000 以上) 場合、UNDO ログのパージが完了するのを待っている間にアップグレードがタイムアウトすることがあります。

  パージが行われるクラスターのライターインスタンスをアップグレードすることで、パージの速度を上げることができます。アップグレードを試みる前にこれを行うと、開始前に

UNDO ログの数を減らすことができます。ライターのサイズを 24XL インスタンスタイプに増やすと、パージ率が 1 時間あたり 100 万レコードを超えることがあります。

UndoLogsListSize CloudWatch メトリクスが非常に大きい場合、サポートケースを開くと、それを停止するための追加の戦略を検討するのに役立つ場合があります。

- 最後に、リリース 1.2.0.0 には、IAM 認証で Bolt プロトコルを使用していた以前のコードに影響する重大な変更がありました。リリース 1.2.0.0 以降、Bolt には IAM 署名用のリソースパスが必要です。Java では、リソースパスの設定は以下のようになります: `request.setResourcePath("/openCypher"));`。その他の言語では、`/openCypher` をエンドポイント URI に追加できます。例については、「Bolt プロトコルの使用」を参照してください。

このエンジンリリースで修正された不具合

- 失敗したトランザクションが正しくクローズされないことがあるバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.2.1.0.R7 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: `3.6.2`
- サポートされている最も新しいバージョンの Gremlin: `3.6.2`
- openCypher バージョン: `Neptune-9.0.20190305-1.0`
- SPARQL バージョン: `1.1`

このリリースへのアップグレード

Amazon Neptune 1.2.1.0.R7 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.1.0 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.1.0 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できま

す。Neptune が手動スナップショットを作成する場合、その名前は preupgrade で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
>   We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the
> instance before
>   proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.2.1.0.R6 (2023-09-12)

2023 年 9 月 12 日現在、エンジンバージョン 1.2.1.0.R6 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

> **Note**
>
> 1.2.0.0 より前のエンジンバージョンからアップグレードを行う場合:
>
> - エンジンリリース 1.2.0.0 では、カスタムパラメータグループとカスタムクラスターパラメータグループに新しい形式が導入されました。そのため、1.2.0.0 より前のエンジンバージョンからエンジンバージョン 1.2.0.0 以降にアップグレードする場合は、パラメータグループファミリー neptune1.2 を使用している既存のカスタムパラメータグループとカスタムクラスターパラメータグループをすべて再作成する必要があります。以前のリリースではパラメータグループファミリー neptune1 が使用されていましたが、それらのパ

ラメータグループはリリース 1.2.0.0 以降では動作しません。詳細については、「Amazon Neptune パラメータグループ」を参照してください。

- エンジンリリース 1.2.0.0 では、UNDO ログの新しい形式も導入されました。そのため、1.2.0.0 より前のバージョンからのアップグレードを開始する前に、以前のエンジンバージョンで作成された UNDO ログをすべてパージし、`UndoLogsListSize` CloudWatch メトリクスをゼロにする必要があります。更新を開始しようとしたときに UNDO ログレコードが多すぎる (200,000 以上) 場合、UNDO ログのパージが完了するのを待っている間にアップグレードがタイムアウトすることがあります。

  パージが行われるクラスターのライターインスタンスをアップグレードすることで、パージの速度を上げることができます。アップグレードを試みる前にこれを行うと、開始前に UNDO ログの数を減らすことができます。ライターのサイズを 24XL インスタンスタイプに増やすと、パージ率が 1 時間あたり 100 万レコードを超えることがあります。

  `UndoLogsListSize` CloudWatch メトリクスが非常に大きい場合、サポートケースを開くと、それを停止するための追加の戦略を検討するのに役立つ場合があります。
- 最後に、リリース 1.2.0.0 には、IAM 認証で Bolt プロトコルを使用していた以前のコードに影響する重大な変更がありました。リリース 1.2.0.0 以降、Bolt には IAM 署名用のリソースパスが必要です。Java では、リソースパスの設定は以下のようになります: `request.setResourcePath("/openCypher"));`。その他の言語では、`/openCypher` をエンドポイント URI に追加できます。例については、「Bolt プロトコルの使用」を参照してください。

このエンジンリリースの新機能

- Neptune データ API をリリースしました。

  Amazon Neptune データ API は、データの読み込み、クエリの実行、データに関する情報の取得、機械学習操作の実行のための SDK サポートを提供します。Neptune における Gremlin および openCypher クエリ言語をサポートし、すべての SDK 言語で使用できます。API リクエストに自動的に署名し、Neptune のアプリケーションへの統合を大幅に簡素化します。

このエンジンリリースで修正された不具合

- スロークエリログが有効になっていると、高負荷時に CPU が急上昇するバグを修正しました。

- `inV()` または `outV()` が続くエッジとそのプロパティを追加すると `InternalFailureException` が発生する Gremlin のバグを修正しました。
- バルクローダーのパフォーマンスが低下する場合があった IAM ロールチェイニングのいくつかの問題を修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.2.1.0.R6 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: `3.6.2`
- サポートされている最も新しいバージョンの Gremlin: `3.6.2`
- openCypher バージョン: `Neptune-9.0.20190305-1.0`
- SPARQL バージョン: `1.1`

このリリースへのアップグレード

Amazon Neptune 1.2.1.0.R6 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.1.0 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.1.0 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
> ```

```
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.2.1.0.R5 (2023-09-02)

2023 年 9 月 2 日現在、エンジンバージョン 1.2.1.0.R5 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

Note

1.2.0.0 より前のエンジンバージョンからアップグレードを行う場合:

- エンジンリリース 1.2.0.0 では、カスタムパラメータグループとカスタムクラスターパラメータグループに新しい形式が導入されました。そのため、1.2.0.0 より前のエンジンバージョンからエンジンバージョン 1.2.0.0 以降にアップグレードする場合は、パラメータグループファミリー `neptune1.2` を使用している既存のカスタムパラメータグループとカスタムクラスターパラメータグループをすべて再作成する必要があります。以前のリリースではパラメータグループファミリー `neptune1` が使用されていましたが、それらのパラメータグループはリリース 1.2.0.0 以降では動作しません。詳細については、「Amazon Neptune パラメータグループ」を参照してください。
- エンジンリリース 1.2.0.0 では、UNDO ログの新しい形式も導入されました。そのため、1.2.0.0 より前のバージョンからのアップグレードを開始する前に、以前のエンジンバージョンで作成された UNDO ログをすべてパージし、`UndoLogsListSize` CloudWatch メトリクスをゼロにする必要があります。更新を開始しようとしたときに UNDO ログレコードが多すぎる (200,000 以上) 場合、UNDO ログのパージが完了するのを待っている間にアップグレードがタイムアウトすることがあります。

  パージが行われるクラスターのライターインスタンスをアップグレードすることで、パージの速度を上げることができます。アップグレードを試みる前にこれを行うと、開始前に

UNDO ログの数を減らすことができます。ライターのサイズを 24XL インスタンスタイプに増やすと、パージ率が 1 時間あたり 100 万レコードを超えることがあります。

`UndoLogsListSize` CloudWatch メトリクスが非常に大きい場合、サポートケースを開くと、それを停止するための追加の戦略を検討するのに役立つ場合があります。

- 最後に、リリース 1.2.0.0 には、IAM 認証で Bolt プロトコルを使用していた以前のコードに影響する重大な変更がありました。リリース 1.2.0.0 以降、Bolt には IAM 署名用のリソースパスが必要です。Java では、リソースパスの設定は以下のようになります: `request.setResourcePath("/openCypher"));`。その他の言語では、`/openCypher` をエンドポイント URI に追加できます。例については、「Bolt プロトコルの使用」を参照してください。

このエンジンリリースの新機能

- Neptune データ API をリリースしました。

  Amazon Neptune データ API は、データの読み込み、クエリの実行、データに関する情報の取得、機械学習操作の実行のための SDK サポートを提供します。Neptune における Gremlin および openCypher クエリ言語をサポートし、すべての SDK 言語で使用できます。API リクエストに自動的に署名し、Neptune のアプリケーションへの統合を大幅に簡素化します。

このエンジンリリースで修正された不具合

- `inV()` または `outV()` が続くエッジとそのプロパティを追加すると `InternalFailureException` が発生する Gremlin のバグを修正しました。
- バルクローダーのパフォーマンスが低下する場合があった IAM ロールチェイニングのいくつかの問題を修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.2.1.0.R5 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: `3.6.2`
- サポートされている最も新しいバージョンの Gremlin: `3.6.2`
- openCypher バージョン: `Neptune-9.0.20190305-1.0`

- SPARQL バージョン: 1.1

このリリースへのアップグレード

Amazon Neptune 1.2.1.0.R5 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.1.0 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.1.0 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the instance before
>  proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.2.1.0.R4 (2023-08-10)

2023 年 8 月 10 日現在、エンジンバージョン 1.2.1.0.R4 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

Important

このエンジンリリースで導入された変更により、一括ロードのパフォーマンスが低下する場合があります。そのため、このリリースへのアップグレードは、問題が解決されるまで一時的に中断されました。

Note

1.2.0.0 より前のエンジンバージョンからアップグレードを行う場合:

- エンジンリリース 1.2.0.0 では、カスタムパラメータグループとカスタムクラスターパラメータグループに新しい形式が導入されました。そのため、1.2.0.0 より前のエンジンバージョンからエンジンバージョン 1.2.0.0 以降にアップグレードする場合は、パラメータグループファミリー `neptune1.2` を使用している既存のカスタムパラメータグループとカスタムクラスターパラメータグループをすべて再作成する必要があります。以前のリリースではパラメータグループファミリー `neptune1` が使用されていましたが、それらのパラメータグループはリリース 1.2.0.0 以降では動作しません。詳細については、「Amazon Neptune パラメータグループ」を参照してください。
- エンジンリリース 1.2.0.0 では、UNDO ログの新しい形式も導入されました。そのため、1.2.0.0 より前のバージョンからのアップグレードを開始する前に、以前のエンジンバージョンで作成された UNDO ログをすべてパージし、`UndoLogsListSize` CloudWatch メトリクスをゼロにする必要があります。更新を開始しようとしたときに UNDO ログレコードが多すぎる (200,000 以上) 場合、UNDO ログのパージが完了するのを待っている間にアップグレードがタイムアウトすることがあります。

  パージが行われるクラスターのライターインスタンスをアップグレードすることで、パージの速度を上げることができます。アップグレードを試みる前にこれを行うと、開始前に UNDO ログの数を減らすことができます。ライターのサイズを 24XL インスタンスタイプに増やすと、パージ率が 1 時間あたり 100 万レコードを超えることがあります。

  `UndoLogsListSize` CloudWatch メトリクスが非常に大きい場合、サポートケースを開くと、それを停止するための追加の戦略を検討するのに役立つ場合があります。
- 最後に、リリース 1.2.0.0 には、IAM 認証で Bolt プロトコルを使用していた以前のコードに影響する重大な変更がありました。リリース 1.2.0.0 以降、Bolt には IAM 署名用のリソースパスが必要です。Java では、リソースパスの設定は以下のようになります: `request.setResourcePath("/openCypher"));`。その他の言語では、`/`

openCypher をエンドポイント URI に追加できます。例については、「Bolt プロトコルの使用」を参照してください。

このエンジンリリースの改良点

- Gremlin の GraphSON-1.0 サポートを追加しました。GraphSON-1.0 を使用するには、次の値を指定して `Accept header` を渡します。

```
application/vnd.gremlin-v1.0+json;types=false
```

このエンジンリリースで修正された不具合

- ネイティブに処理されないステップの子トラバーサルに述語を含むクエリについて、Gremlin クエリステータスエンドポイントを確認すると、トランザクションリークが発生する Gremlin のバグを修正しました。
- Bolt のトランザクション処理における openCypher のバグを修正しました。
- クラッシュを引き起こす可能性のある、ストレージレイヤーの同時実行の問題を修正しました。
- スロークエリログのバグを修正し、無効なときにはアクティブにならないようにしました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.2.1.0.R4 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: `3.6.2`
- サポートされている最も新しいバージョンの Gremlin: `3.6.5`
- openCypher バージョン: `Neptune-9.0.20190305-1.0`
- SPARQL バージョン: `1.1`

エンジンリリース 1.2.1.0.R4 へのアップグレードパス

このリリースへのアップグレード

Amazon Neptune 1.2.1.0.R4 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.1.0 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.1.0 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the instance before
>   proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.2.1.0.R3 (2023-06-13)

2023 年 6 月 13 日現在、エンジンバージョン 1.2.1.0.R3 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

Important

このエンジンリリースで導入された変更により、一括ロードのパフォーマンスが低下する場合があります。そのため、このリリースへのアップグレードは、問題が解決されるまで一時的に中断されました。

Note

1.2.0.0 より前のエンジンバージョンからアップグレードを行う場合:

- エンジンリリース 1.2.0.0 では、カスタムパラメータグループとカスタムクラスターパラメータグループに新しい形式が導入されました。そのため、1.2.0.0 より前のエンジンバージョンからエンジンバージョン 1.2.0.0 以降にアップグレードする場合は、パラメータグループファミリー `neptune1.2` を使用している既存のカスタムパラメータグループとカスタムクラスターパラメータグループをすべて再作成する必要があります。以前のリリースではパラメータグループファミリー `neptune1` が使用されていましたが、それらのパラメータグループはリリース 1.2.0.0 以降では動作しません。詳細については、「Amazon Neptune パラメータグループ」を参照してください。
- エンジンリリース 1.2.0.0 では、UNDO ログの新しい形式も導入されました。そのため、1.2.0.0 より前のバージョンからのアップグレードを開始する前に、以前のエンジンバージョンで作成された UNDO ログをすべてパージし、`UndoLogsListSize` CloudWatch メトリクスをゼロにする必要があります。更新を開始しようとしたときに UNDO ログレコードが多すぎる (200,000 以上) 場合、UNDO ログのパージが完了するのを待っている間にアップグレードがタイムアウトすることがあります。

  パージが行われるクラスターのライターインスタンスをアップグレードすることで、パージの速度を上げることができます。アップグレードを試みる前にこれを行うと、開始前に UNDO ログの数を減らすことができます。ライターのサイズを 24XL インスタンスタイプに増やすと、パージ率が 1 時間あたり 100 万レコードを超えることがあります。

  `UndoLogsListSize` CloudWatch メトリクスが非常に大きい場合、サポートケースを開くと、それを停止するための追加の戦略を検討するのに役立つ場合があります。
- 最後に、リリース 1.2.0.0 には、IAM 認証で Bolt プロトコルを使用していた以前のコードに影響する重大な変更がありました。リリース 1.2.0.0 以降、Bolt には IAM 署名用のリソースパスが必要です。Java では、リソースパスの設定は以下のようになります: `request.setResourcePath("/openCypher"));`。その他の言語では、`/`

openCypher をエンドポイント URI に追加できます。例については、「Bolt プロトコルの使用」を参照してください。

このエンジンリリースの新機能

- IAM ロールチェイニングを使用したクロスアカウント一括読み込みのサポートが追加されました。

このエンジンリリースの改良点

- Gremlin の `fail()` ステップを改善して、生成された例外を汎用 `InternalFailureException` と区別し、ユーザーによって指定されたメッセージが呼び出し元に確実に伝わるようにしました。
- `store`、`aggregate`、`cap`、`limit`、および `hasLabel` について、Gremlin クエリエンジンの最適化が改善されました。
- openCypher 三角関数のサポートが追加されました。
  - `acos()`
  - `asin()`
  - `atan()`
  - `atan2()`
  - `cos()`
  - `cot()`
  - `degrees()`
  - `pi()`
  - `radians()`
  - `sin()`
  - `tan()`
- いくつかの openCypher 集約関数のサポートが追加されました。
  - `percentileDisc()`
  - `stDev()`
- `datetime` を `epochmillis` に変換する openCypher `epochmillis()` 関数のサポートが追加されました。例:

```
MATCH (n) RETURN epochMillis(n.someDateTime)
1698972364782
```

- openCypher モジュロ (%) 演算子のサポートが追加されました。
- openCypher Static Debug Explain ツールのサポートが追加されました。
- openCypher `randomUUID()` 関数のサポートが追加されました。
- openCypher のパフォーマンスが向上しました。
  - パーサーとクエリプランナーを改善しました。
  - DFE エンジンの CPU 使用率が向上しました。
  - 同じ変数を再利用する複数の更新句を含むクエリのパフォーマンスが向上しました。例:

    ```
    MERGE (n {name: 'John'})
      or
    MERGE (m {name: 'Jim'})
      or
    MERGE (n)-[:knows {since: 2023}]#(m)
    ```

  - 次のようなマルチホップクエリパターンのクエリプランが最適化されました。

    ```
    MATCH (n)-->()-->()-->(m)
    RETURN n m
    ```

  - パラメータ化されたクエリにより、リストとマップの挿入のパフォーマンスが向上しました。例:

    ```
    UNWIND $idList as id MATCH (n {`~id`: id})
    RETURN n.name
    ```

  - `WITH` を含んでいるクエリ実行が、適切なバリアにすることで改善されました。
  - `Unfold` および集約関数内の値の重複した実体化を回避するように最適化されました。
- 次のような VALUES 句に多数の静的入力を含む SPARQL クエリのパフォーマンスが向上しました。

```
SELECT ?n WHERE { VALUES (?name) { ("John") ("Jim") ... many values ... } ?n a ?
n_type . ?n ?name . }
```

- SPARQL CBD クエリのパフォーマンスが向上しました。

このエンジンリリースで修正された不具合

- 深いネストを伴う長いクエリが、クエリのプランニング段階で CPU 使用率が高くなり、クエリがタイムアウトする Gremlin のバグを修正しました。
- `mergeV` または `mergeE` を使用すると、無効な `NullPointerException` がスローされる Gremlin のバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.2.1.0.R3 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: `3.6.2`
- サポートされている最も新しいバージョンの Gremlin: `3.6.2`
- openCypher バージョン: `Neptune-9.0.20190305-1.0`
- SPARQL バージョン: `1.1`

エンジンリリース 1.0.5.0.R3 へのアップグレードパス

このリリースへのアップグレード

Amazon Neptune 1.2.1.0.R3 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.1.0 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
```

```
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.1.0 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。

```
  We're sorry, your request to modify DB cluster (cluster identifier) has
failed.
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.2.1.0.R2 (2023-05-02)

2023 年 5 月 2 日現在、エンジンバージョン 1.2.1.0.R2 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

Note

1.2.0.0 より前のエンジンバージョンからアップグレードを行う場合:

- エンジンリリース 1.2.0.0 では、カスタムパラメータグループとカスタムクラスターパラメータグループに新しい形式が導入されました。そのため、1.2.0.0 より前のエンジンバージョンからエンジンバージョン 1.2.0.0 以降にアップグレードする場合は、パラメータグループファミリー neptune1.2 を使用している既存のカスタムパラメータグループとカスタムクラスターパラメータグループをすべて再作成する必要があります。以前のリリースではパラメータグループファミリー neptune1 が使用されていましたが、それらのパラメータグループはリリース 1.2.0.0 以降では動作しません。詳細については、「Amazon Neptune パラメータグループ」を参照してください。
- エンジンリリース 1.2.0.0 では、UNDO ログの新しい形式も導入されました。そのため、1.2.0.0 より前のバージョンからのアップグレードを開始する前に、以前のエンジンバージョンで作成された UNDO ログをすべてパージし、`UndoLogsListSize` CloudWatch メトリクスをゼロにする必要があります。更新を開始しようとしたときに UNDO ログレコードが多すぎる (200,000 以上) 場合、UNDO ログのパージが完了するのを待っている間にアップグレードがタイムアウトすることがあります。

パージが行われるクラスターのライターインスタンスをアップグレードすることで、パージの速度を上げることができます。アップグレードを試みる前にこれを行うと、開始前に UNDO ログの数を減らすことができます。ライターのサイズを 24XL インスタンスタイプに増やすと、パージ率が 1 時間あたり 100 万レコードを超えることがあります。

UndoLogsListSize CloudWatch メトリクスが非常に大きい場合、サポートケースを開くと、それを停止するための追加の戦略を検討するのに役立つ場合があります。

- 最後に、リリース 1.2.0.0 には、IAM 認証で Bolt プロトコルを使用していた以前のコードに影響する重大な変更がありました。リリース 1.2.0.0 以降、Bolt には IAM 署名用のリソースパスが必要です。Java では、リソースパスの設定は以下のようになります: `request.setResourcePath("/openCypher"));`。その他の言語では、`/openCypher` をエンドポイント URI に追加できます。例については、「Bolt プロトコルの使用」を参照してください。

このエンジンリリースの改良点

- すべての Neptune ML API に `enableInterContainerTrafficEncryption` パラメータを追加しました。これを使用して、トレーニングジョブやハイパーパラメータチューニングジョブでコンテナ間トラフィック暗号化を有効または無効にできます。
- Gremlin `mergeV()` および `mergeE()` のマルチラベルサポートが追加されました。

このエンジンリリースで修正された不具合

- 更新とリターンのクエリが `orderBy`、`limit`、または `skip` を正しく処理しない openCypher のバグを修正しました。
- あるリクエストに含まれるパラメータが、別の同時リクエストに含まれるパラメータによってオーバーライドされる可能性がある openCypher のバグを修正しました。
- スロークエリログに正しいクエリ時間が含まれていなかった openCypher のバグを修正しました。
- `GroupCountStep` を含んでいるクエリが文字列として送信されたときに、トランザクションリークが発生する可能性がある Gremlin のバグを修正しました。
- スロークエリログが有効になっていると WebSocket クエリが失敗する Gremlin のバグを修正しました。

- WebSocket リクエストのスロークエリログにストレージカウンターのデバッグログが欠落していた Gremlin のバグを修正しました。
- `mergeV()` と `mergeE()` に関係するいくつかの Gremlin のバグを修正しました。
- 名前付きグラフのクエリコストが誤って見積もられ、クエリプランが最適でなかったり、メモリ不足エラーが発生したりする SPARQL のバグを修正しました。
- IAM 対応クラスターでの Gremlin および openCypher クエリの承認に影響するバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.2.1.0.R2 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: `3.6.2`
- サポートされている最も新しいバージョンの Gremlin: `3.6.2`
- openCypher バージョン: `Neptune-9.0.20190305-1.0`
- SPARQL バージョン: `1.1`

エンジンリリース 1.2.1.0.R2 へのアップグレードパス

このリリースへのアップグレード

Amazon Neptune 1.2.1.0.R2 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.1.0 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.1.0 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
>   We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the
> instance before
>   proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.2.0.2 (2022-11-20)

2022 年 11 月 20 日現在、エンジンバージョン 1.2.0.2 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

> **Note**
>
> 1.2.0.0 より前のエンジンバージョンからアップグレードを行う場合:
>
> - エンジンリリース 1.2.0.0 では、カスタムパラメータグループとカスタムクラスターパラメータグループに新しい形式が導入されました。そのため、1.2.0.0 より前のエンジンバージョンからエンジンバージョン 1.2.0.0 以降にアップグレードする場合は、パラメータグループファミリー neptune1.2 を使用している既存のカスタムパラメータグループとカスタムクラスターパラメータグループをすべて再作成する必要があります。以前のリリースではパラメータグループファミリー neptune1 が使用されていましたが、それらのパラメータグループはリリース 1.2.0.0 以降では動作しません。詳細については、「Amazon Neptune パラメータグループ」を参照してください。

- エンジンリリース 1.2.0.0 では、UNDO ログの新しい形式も導入されました。そのため、1.2.0.0 より前のバージョンからのアップグレードを開始する前に、以前のエンジンバージョンで作成された UNDO ログをすべてパージし、`UndoLogsListSize` CloudWatch メトリクスをゼロにする必要があります。更新を開始しようとしたときに UNDO ログレコードが多すぎる (200,000 以上) 場合、UNDO ログのパージが完了するのを待っている間にアップグレードがタイムアウトすることがあります。

  パージが行われるクラスターのライターインスタンスをアップグレードすることで、パージの速度を上げることができます。アップグレードを試みる前にこれを行うと、開始前に UNDO ログの数を減らすことができます。ライターのサイズを 24XL インスタンスタイプに増やすと、パージ率が 1 時間あたり 100 万レコードを超えることがあります。

  `UndoLogsListSize` CloudWatch メトリクスが非常に大きい場合、サポートケースを開くと、それを停止するための追加の戦略を検討するのに役立つ場合があります。
- 最後に、リリース 1.2.0.0 には、IAM 認証で Bolt プロトコルを使用していた以前のコードに影響する重大な変更がありました。リリース 1.2.0.0 以降、Bolt には IAM 署名用のリソースパスが必要です。Java では、リソースパスの設定は以下のようになります: `request.setResourcePath("/openCypher"));`。その他の言語では、`/openCypher` をエンドポイント URI に追加できます。例については、「Bolt プロトコルの使用」を参照してください。

このリリースの後続のパッチリリース

- リリース: 1.2.0.2.R2 (2022-12-15)
- リリース: 1.2.0.2.R3 (2023-03-27)
- リリース: 1.2.0.2.R4 (2023-05-08)
- リリース: 1.2.0.2.R5 (2023-08-16)
- リリース: 1.2.0.2.R6 (2023-09-12)

このエンジンリリースの新機能

- Neptune ML に Gremlin のリアルタイム帰納的推論を導入しました。

- Neptune が生成する UUID の代わりに、エンティティのカスタム ID 値の指定をサポートする openCypher 拡張機能を導入しました。カスタム ID を割り当てることができるため、Neo4j から Neptune への移行が容易になります。

  > **Warning**
  >
  > ~id は、現在、予約済みのプロパティ名と見なされているため、openCypher 仕様に対するこの拡張は下位互換性がありません。~id をデータやクエリで既にプロパティとして使用している場合、このリリースにアップグレードする前に、~id プロパティを新しいプロパティキーに移行する必要があります。

- いくつかの新しい SPARQL DESCRIBE モードと、それらを設定するためのクエリーヒントが追加されました。

このエンジンリリースの改良点

- openCypher のパフォーマンス、特に VLP クエリのパフォーマンスが向上しました。
- 次のような端末以外の制限がある Gremlin クエリの DFE パフォーマンスが向上しました。

```
g.withSideEffect('Neptune#useDFE',true).V().hasLabel('Student').limit(5).out('takesCourse')
```

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.2.0.2 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: `3.5.2`
- サポートされている最も新しいバージョンの Gremlin: `3.5.4`
- openCypher バージョン: `Neptune-9.0.20190305-1.0`
- SPARQL バージョン: `1.1`

エンジンリリース 1.2.0.2 へのアップグレードパス

このリリースへのアップグレード

Amazon Neptune 1.2.0.2 が公開されました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.0.2 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.0.2 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.2.0.2.R6 (2023-09-12)

2023 年 9 月 12 日現在、エンジンバージョン 1.2.0.2.R6 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

Note

1.2.0.0 より前のエンジンバージョンからアップグレードを行う場合:

- エンジンリリース 1.2.0.0 では、カスタムパラメータグループとカスタムクラスターパラメータグループに新しい形式が導入されました。そのため、1.2.0.0 より前のエンジンバージョンからエンジンバージョン 1.2.0.0 以降にアップグレードする場合は、パラメータグループファミリー `neptune1.2` を使用している既存のカスタムパラメータグループとカスタムクラスターパラメータグループをすべて再作成する必要があります。以前のリリースではパラメータグループファミリー `neptune1` が使用されていましたが、それらのパラメータグループはリリース 1.2.0.0 以降では動作しません。詳細については、「Amazon Neptune パラメータグループ」を参照してください。
- エンジンリリース 1.2.0.0 では、UNDO ログの新しい形式も導入されました。そのため、1.2.0.0 より前のバージョンからのアップグレードを開始する前に、以前のエンジンバージョンで作成された UNDO ログをすべてパージし、`UndoLogsListSize` CloudWatch メトリクスをゼロにする必要があります。更新を開始しようとしたときに UNDO ログレコードが多すぎる (200,000 以上) 場合、UNDO ログのパージが完了するのを待っている間にアップグレードがタイムアウトすることがあります。

  パージが行われるクラスターのライターインスタンスをアップグレードすることで、パージの速度を上げることができます。アップグレードを試みる前にこれを行うと、開始前に UNDO ログの数を減らすことができます。ライターのサイズを 24XL インスタンスタイプに増やすと、パージ率が 1 時間あたり 100 万レコードを超えることがあります。

  `UndoLogsListSize` CloudWatch メトリクスが非常に大きい場合、サポートケースを開くと、それを停止するための追加の戦略を検討するのに役立つ場合があります。
- 最後に、リリース 1.2.0.0 には、IAM 認証で Bolt プロトコルを使用していた以前のコードに影響する重大な変更がありました。リリース 1.2.0.0 以降、Bolt には IAM 署名用のリソースパスが必要です。Java では、リソースパスの設定は以下のようになります: `request.setResourcePath("/openCypher"));`。その他の言語では、`/openCypher` をエンドポイント URI に追加できます。例については、「Bolt プロトコルの使用」を参照してください。

このエンジンリリースで修正された不具合

- 言語タグ付きリテラルで呼び出されると、REGEX 演算子が成功しないという SPARQL のバグを修正しました。
- 一括読み込みのパフォーマンスが低下する問題を修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.2.0.2.R6 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: `3.5.2`
- サポートされている最も新しいバージョンの Gremlin: `3.5.5`
- openCypher バージョン: `Neptune-9.0.20190305-1.0`
- SPARQL バージョン: `1.1`

エンジンリリース 1.2.0.2.R6 へのアップグレードパス

エンジンバージョン `1.2.0.2` を実行している場合、Neptune DB クラスターは次のメンテナンスウィンドウで自動的にこのメンテナンスパッチリリースにアップグレードされます。

このリリースへのアップグレード

Amazon Neptune 1.2.0.2.R6 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.0.2 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.0.2 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.2.0.2.R5 (2023-08-16)

2023 年 8 月 16 日現在、エンジンバージョン 1.2.0.2.R5 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

> **Important**
>
> このエンジンリリースで導入された変更により、一括ロードのパフォーマンスが低下する場合があります。そのため、このリリースへのアップグレードは、問題が解決されるまで一時的に中断されました。

> **Note**
>
> 1.2.0.0 より前のエンジンバージョンからアップグレードを行う場合:
>
> - エンジンリリース 1.2.0.0 では、カスタムパラメータグループとカスタムクラスターパラメータグループに新しい形式が導入されました。そのため、1.2.0.0 より前のエンジンバージョンからエンジンバージョン 1.2.0.0 以降にアップグレードする場合は、パラメータグ

ループファミリー neptune1.2 を使用している既存のカスタムパラメータグループとカスタムクラスターパラメータグループをすべて再作成する必要があります。以前のリリースではパラメータグループファミリー neptune1 が使用されていましたが、それらのパラメータグループはリリース 1.2.0.0 以降では動作しません。詳細については、「Amazon Neptune パラメータグループ」を参照してください。

- エンジンリリース 1.2.0.0 では、UNDO ログの新しい形式も導入されました。そのため、1.2.0.0 より前のバージョンからのアップグレードを開始する前に、以前のエンジンバージョンで作成された UNDO ログをすべてパージし、UndoLogsListSize CloudWatch メトリクスをゼロにする必要があります。更新を開始しようとしたときに UNDO ログレコードが多すぎる (200,000 以上) 場合、UNDO ログのパージが完了するのを待っている間にアップグレードがタイムアウトすることがあります。

  パージが行われるクラスターのライターインスタンスをアップグレードすることで、パージの速度を上げることができます。アップグレードを試みる前にこれを行うと、開始前に UNDO ログの数を減らすことができます。ライターのサイズを 24XL インスタンスタイプに増やすと、パージ率が 1 時間あたり 100 万レコードを超えることがあります。

  UndoLogsListSize CloudWatch メトリクスが非常に大きい場合、サポートケースを開くと、それを停止するための追加の戦略を検討するのに役立つ場合があります。

- 最後に、リリース 1.2.0.0 には、IAM 認証で Bolt プロトコルを使用していた以前のコードに影響する重大な変更がありました。リリース 1.2.0.0 以降、Bolt には IAM 署名用のリソースパスが必要です。Java では、リソースパスの設定は以下のようになります: `request.setResourcePath("/openCypher"));`。その他の言語では、/openCypher をエンドポイント URI に追加できます。例については、「Bolt プロトコルの使用」を参照してください。

このエンジンリリースで修正された不具合

- 文字列出力の一部にスペース文字が含まれていると `order()` によって正しくソートされない Gremlin のバグを修正しました。
- ネイティブに処理されないステップの子トラバーサルに述語を含むクエリについて、Gremlin クエリステータスエンドポイントを確認すると、トランザクションリークが発生する Gremlin のバグを修正しました。
- Bolt のトランザクション処理における openCypher のバグを修正しました。
- クラッシュを引き起こす可能性のある、ストレージレイヤーの同時実行の問題を修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.2.0.2.R5 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: `3.5.2`
- サポートされている最も新しいバージョンの Gremlin: `3.5.5`
- openCypher バージョン: `Neptune-9.0.20190305-1.0`
- SPARQL バージョン: `1.1`

エンジンリリース 1.2.0.2.R5 へのアップグレードパス

エンジンバージョン `1.2.0.2` を実行している場合、Neptune DB クラスターは次のメンテナンスウィンドウで自動的にこのメンテナンスパッチリリースにアップグレードされます。

このリリースへのアップグレード

Amazon Neptune 1.2.0.2.R5 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.0.2 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.0.2 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
> ```

```
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.2.0.2.R4 (2023-05-08)

2023 年 5 月 8 日現在、エンジンバージョン 1.2.0.2.R4 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

Note

1.2.0.0 より前のエンジンバージョンからアップグレードを行う場合:

- エンジンリリース 1.2.0.0 では、カスタムパラメータグループとカスタムクラスターパラメータグループに新しい形式が導入されました。そのため、1.2.0.0 より前のエンジンバージョンからエンジンバージョン 1.2.0.0 以降にアップグレードする場合は、パラメータグループファミリー `neptune1.2` を使用している既存のカスタムパラメータグループとカスタムクラスターパラメータグループをすべて再作成する必要があります。以前のリリースではパラメータグループファミリー `neptune1` が使用されていましたが、それらのパラメータグループはリリース 1.2.0.0 以降では動作しません。詳細については、「Amazon Neptune パラメータグループ」を参照してください。
- エンジンリリース 1.2.0.0 では、UNDO ログの新しい形式も導入されました。そのため、1.2.0.0 より前のバージョンからのアップグレードを開始する前に、以前のエンジンバージョンで作成された UNDO ログをすべてパージし、`UndoLogsListSize` CloudWatch メトリクスをゼロにする必要があります。更新を開始しようとしたときに UNDO ログレコードが多すぎる (200,000 以上) 場合、UNDO ログのパージが完了するのを待っている間にアップグレードがタイムアウトすることがあります。

  パージが行われるクラスターのライターインスタンスをアップグレードすることで、パージの速度を上げることができます。アップグレードを試みる前にこれを行うと、開始前に

UNDO ログの数を減らすことができます。ライターのサイズを 24XL インスタンスタイプに増やすと、パージ率が 1 時間あたり 100 万レコードを超えることがあります。

UndoLogsListSize CloudWatch メトリクスが非常に大きい場合、サポートケースを開くと、それを停止するための追加の戦略を検討するのに役立つ場合があります。

- 最後に、リリース 1.2.0.0 には、IAM 認証で Bolt プロトコルを使用していた以前のコードに影響する重大な変更がありました。リリース 1.2.0.0 以降、Bolt には IAM 署名用のリソースパスが必要です。Java では、リソースパスの設定は以下のようになります: `request.setResourcePath("/openCypher"));`。その他の言語では、`/openCypher` をエンドポイント URI に追加できます。例については、「Bolt プロトコルの使用」を参照してください。

このエンジンリリースで修正された不具合

- VALUES 句に大量の値が挿入されるとパフォーマンスが低下する可能性がある SPARQL のバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.2.0.2.R4 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: `3.5.2`
- サポートされている最も新しいバージョンの Gremlin: `3.5.6`
- openCypher バージョン: `Neptune-9.0.20190305-1.0`
- SPARQL バージョン: `1.1`

エンジンリリース 1.2.0.2.R4 へのアップグレードパス

エンジンバージョン `1.2.0.2` を実行している場合、Neptune DB クラスターは次のメンテナンスウィンドウで自動的にこのメンテナンスパッチリリースにアップグレードされます。

このリリースへのアップグレード

Amazon Neptune 1.2.0.2.R4 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.0.2 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.0.2 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.2.0.2.R3 (2023-03-27)

2023 年 3 月 27 日現在、エンジンバージョン 1.2.0.2.R3 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

> **Note**
>
> 1.2.0.0 より前のエンジンバージョンからアップグレードを行う場合:

- エンジンリリース 1.2.0.0 では、カスタムパラメータグループとカスタムクラスターパラメータグループに新しい形式が導入されました。そのため、1.2.0.0 より前のエンジンバージョンからエンジンバージョン 1.2.0.0 以降にアップグレードする場合は、パラメータグループファミリー neptune1.2 を使用している既存のカスタムパラメータグループとカスタムクラスターパラメータグループをすべて再作成する必要があります。以前のリリースではパラメータグループファミリー neptune1 が使用されていましたが、それらのパラメータグループはリリース 1.2.0.0 以降では動作しません。詳細については、「Amazon Neptune パラメータグループ」を参照してください。
- エンジンリリース 1.2.0.0 では、UNDO ログの新しい形式も導入されました。そのため、1.2.0.0 より前のバージョンからのアップグレードを開始する前に、以前のエンジンバージョンで作成された UNDO ログをすべてパージし、`UndoLogsListSize` CloudWatch メトリクスをゼロにする必要があります。更新を開始しようとしたときに UNDO ログレコードが多すぎる (200,000 以上) 場合、UNDO ログのパージが完了するのを待っている間にアップグレードがタイムアウトすることがあります。

  パージが行われるクラスターのライターインスタンスをアップグレードすることで、パージの速度を上げることができます。アップグレードを試みる前にこれを行うと、開始前に UNDO ログの数を減らすことができます。ライターのサイズを 24XL インスタンスタイプに増やすと、パージ率が 1 時間あたり 100 万レコードを超えることがあります。

  `UndoLogsListSize` CloudWatch メトリクスが非常に大きい場合、サポートケースを開くと、それを停止するための追加の戦略を検討するのに役立つ場合があります。
- 最後に、リリース 1.2.0.0 には、IAM 認証で Bolt プロトコルを使用していた以前のコードに影響する重大な変更がありました。リリース 1.2.0.0 以降、Bolt には IAM 署名用のリソースパスが必要です。Java では、リソースパスの設定は以下のようになります: `request.setResourcePath("/openCypher"));`。その他の言語では、`/openCypher` をエンドポイント URI に追加できます。例については、「Bolt プロトコルの使用」を参照してください。

このエンジンリリースの改良点

- サーバーレス DB クラスターでは、最小容量設定を 1.0 NCU に、最小有効最大設定を 2.5 NCU に変更しました。「Neptune サーバーレス DB クラスターの容量スケーリング」を参照してください。

- すべての Neptune ML API に enableInterContainerTrafficEncryption パラメータを追加しました。これを使用して、トレーニングジョブやハイパーパラメータチューニングジョブでコンテナ間トラフィック暗号化を有効または無効にできます。

このエンジンリリースで修正された不具合

- option(Predicate) が有効な Gremlin 構文として認識されない Gremlin のバグを修正しました。
- クエリに含まれるステップが多すぎるためにクエリが失敗した場合に正しくクリーンアップされない Gremlin のバグを修正しました。
- Tinkerpop にフォールバックすることにより、limit を非結合ステップの子トラバーサルとして持つ DFE クエリに影響していた Gremlin の正確性の問題を修正しました。このようなクエリの例は以下のとおりです。

```
g.withSideEffect('Neptune#useDFE', true).V().as("a").select("a").by(out().limit(1))
```

- 文字列として送信されたクエリに GroupCountStep が含まれるときの Gremlin トランザクションリークを修正しました。
- リストまたはマップのリストでパラメータ値のタイプが正しく解釈されない openCypher のバグを修正しました。
- 更新とリターンのクエリが orderBy、limit、または skip を正しく処理しない openCypher のバグを修正しました。
- あるリクエストに含まれるパラメータが、別の同時リクエストに含まれるパラメータによってオーバーライドされる可能性がある openCypher のバグを修正しました。
- VALUES 句に大量の値が挿入されるとパフォーマンスが低下する可能性がある SPARQL のバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.2.0.2.R3 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: 3.5.2
- サポートされている最も新しいバージョンの Gremlin: 3.5.6
- openCypher バージョン: Neptune-9.0.20190305-1.0

- SPARQL バージョン: 1.1

エンジンリリース 1.2.0.2.R3 へのアップグレードパス

エンジンバージョン 1.2.0.2 を実行している場合、Neptune DB クラスターは次のメンテナンスウィンドウで自動的にこのメンテナンスパッチリリースにアップグレードされます。

このリリースへのアップグレード

Amazon Neptune 1.2.0.2.R2 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.0.2 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.0.2 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.2.0.2.R2 (2022-12-15)

2022 年 12 月 15 日現在、エンジンバージョン 1.2.0.2.R2 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

> **Note**
>
> 1.2.0.0 より前のエンジンバージョンからアップグレードを行う場合:
>
> - エンジンリリース 1.2.0.0 では、カスタムパラメータグループとカスタムクラスターパラメータグループに新しい形式が導入されました。そのため、1.2.0.0 より前のエンジンバージョンからエンジンバージョン 1.2.0.0 以降にアップグレードする場合は、パラメータグループファミリー `neptune1.2` を使用している既存のカスタムパラメータグループとカスタムクラスターパラメータグループをすべて再作成する必要があります。以前のリリースではパラメータグループファミリー `neptune1` が使用されていましたが、それらのパラメータグループはリリース 1.2.0.0 以降では動作しません。詳細については、「Amazon Neptune パラメータグループ」を参照してください。
> - エンジンリリース 1.2.0.0 では、UNDO ログの新しい形式も導入されました。そのため、1.2.0.0 より前のバージョンからのアップグレードを開始する前に、以前のエンジンバージョンで作成された UNDO ログをすべてパージし、`UndoLogsListSize` CloudWatch メトリクスをゼロにする必要があります。更新を開始しようとしたときに UNDO ログレコードが多すぎる (200,000 以上) 場合、UNDO ログのパージが完了するのを待っている間にアップグレードがタイムアウトすることがあります。
>
>   パージが行われるクラスターのライターインスタンスをアップグレードすることで、パージの速度を上げることができます。アップグレードを試みる前にこれを行うと、開始前に UNDO ログの数を減らすことができます。ライターのサイズを 24XL インスタンスタイプに増やすと、パージ率が 1 時間あたり 100 万レコードを超えることがあります。
>
>   `UndoLogsListSize` CloudWatch メトリクスが非常に大きい場合、サポートケースを開くと、それを停止するための追加の戦略を検討するのに役立つ場合があります。
> - 最後に、リリース 1.2.0.0 には、IAM 認証で Bolt プロトコルを使用していた以前のコードに影響する重大な変更がありました。リリース 1.2.0.0 以降、Bolt には IAM 署名用のリソースパスが必要です。Java では、リソースパスの設定は以下のようにな

ります: `request.setResourcePath("/openCypher"));`。その他の言語では、/openCypher をエンドポイント URI に追加できます。例については、「Bolt プロトコルの使用」を参照してください。

このエンジンリリースの改良点

- `MERGE` と `OPTIONAL MATCH` を含む openCypher クエリのパフォーマンスが向上しました。
- リテラル値のマップのリストの `UNWIND` を含む openCypher クエリのパフォーマンスが向上しました。
- `id` の `IN` フィルターの付いた openCypher クエリのパフォーマンスが向上しました。例:

```
MATCH (n) WHERE id(n) IN ['1', '2', '3'] RETURN n
```

- `repeat`、`coalesce`、`store`、`aggregate` など、さまざまな Gremlin 演算子のパフォーマンスの向上と正確性の修正を行いました。

このエンジンリリースで修正された不具合

- Bolt と SPARQL-JSON でクエリが NULL 値ではなく文字列 `"null"` を返す openCypher のバグを修正しました。
- `UnionStep` に付加されたステップラベルが子トラバーサルの最後のパス要素に反映されない原因となっていた Gremlin のバグを修正しました。
- DFE エンジンの `by()` トラバーサル時に `valueMap()` が最適化されない原因となっていた Gremlin のバグを修正しました。
- 長い Gremlin トランザクションの一部として実行された読み取りクエリが行をロックしないという Gremlin のバグを修正しました。
- 不要な情報がログに記録されたり、特定のフィールドがログから欠落したりする監査ログのバグを修正しました。
- IAM 対応 DB クラスターへの HTTP リクエストの IAM ARN が記録されない監査ログのバグを修正しました。
- ルックアップキャッシュのバグを修正して、キャッシュへの書き込みに使用される増分メモリを制限しました。
- 書き込みが失敗したときにルックアップキャッシュに読み取り専用モードを設定するというルックアップキャッシュのバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.2.0.2.R2 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: `3.5.2`
- サポートされている最も新しいバージョンの Gremlin: `3.5.4`
- openCypher バージョン: `Neptune-9.0.20190305-1.0`
- SPARQL バージョン: `1.1`

エンジンリリース 1.2.0.2.R2 へのアップグレードパス

エンジンバージョン `1.2.0.2` を実行している場合、Neptune DB クラスターは次のメンテナンスウィンドウで自動的にこのメンテナンスパッチリリースにアップグレードされます。

このリリースへのアップグレード

Amazon Neptune 1.2.0.2.R2 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.0.2 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.0.2 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
> ```

```
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.2.0.1 (2022-10-26)

2022 年 10 月 26 日現在、エンジンバージョン1.2.0.1. は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

Note

1.2.0.0 より前のエンジンバージョンからアップグレードを行う場合:

- エンジンリリース 1.2.0.0 では、カスタムパラメータグループとカスタムクラスターパラメータグループに新しい形式が導入されました。そのため、1.2.0.0 より前のエンジンバージョンからエンジンバージョン 1.2.0.0 以降にアップグレードする場合は、パラメータグループファミリー neptune1.2 を使用している既存のカスタムパラメータグループとカスタムクラスターパラメータグループをすべて再作成する必要があります。以前のリリースではパラメータグループファミリー neptune1 が使用されていましたが、それらのパラメータグループはリリース 1.2.0.0 以降では動作しません。詳細については、「Amazon Neptune パラメータグループ」を参照してください。
- エンジンリリース 1.2.0.0 では、UNDO ログの新しい形式も導入されました。そのため、1.2.0.0 より前のバージョンからのアップグレードを開始する前に、以前のエンジンバージョンで作成された UNDO ログをすべてパージし、`UndoLogsListSize` CloudWatch メトリクスをゼロにする必要があります。更新を開始しようとしたときに UNDO ログレコードが多すぎる (200,000 以上) 場合、UNDO ログのパージが完了するのを待っている間にアップグレードがタイムアウトすることがあります。

  パージが行われるクラスターのライターインスタンスをアップグレードすることで、パージの速度を上げることができます。アップグレードを試みる前にこれを行うと、開始前に

UNDO ログの数を減らすことができます。ライターのサイズを 24XL インスタンスタイプに増やすと、パージ率が 1 時間あたり 100 万レコードを超えることがあります。

UndoLogsListSize CloudWatch メトリクスが非常に大きい場合、サポートケースを開くと、それを停止するための追加の戦略を検討するのに役立つ場合があります。

- 最後に、リリース 1.2.0.0 には、IAM 認証で Bolt プロトコルを使用していた以前のコードに影響する重大な変更がありました。リリース 1.2.0.0 以降、Bolt には IAM 署名用のリソースパスが必要です。Java では、リソースパスの設定は以下のようになります: `request.setResourcePath("/openCypher"));`。その他の言語では、`/openCypher` をエンドポイント URI に追加できます。例については、「Bolt プロトコルの使用」を参照してください。

このリリースの後続のパッチリリース

- メンテナンスリリース: 1.2.0.1.R2 (2022-12-13)
- メンテナンスリリース:1.2.0.1.R3 (2023-09-27)

このエンジンリリースの新機能

- Amazon Neptune サーバーレスを導入しました。これは、処理需要の増加に合わせて DB クラスターをスケールアップし、需要が減少すると再びスケールダウンするオンデマンド自動スケーリング設定です。

このエンジンリリースの改良点

- Gremlin `order-by` クエリのパフォーマンスが改善されました。`NeptuneGraphQueryStep` の末尾に `order-by` が付いている Gremlin クエリは、パフォーマンスを向上させるために、より大きなサイズのチャンクを使用するようになりました。これはクエリプランの内部 (ルート以外の) ノードの `order-by` には適用されません。
- Gremlin 更新クエリのパフォーマンスが改善されました。エッジやプロパティを追加している間は、削除されないように頂点とエッジをロックする必要があります。この変更により、トランザクション内の重複ロックがなくなり、パフォーマンスが向上します。
- dedup をネイティブ実行レイヤーにプッシュすることで、`repeat()` サブクエリの内部の `dedup()` を使用する Gremlin クエリのパフォーマンスが向上しました。

- IAM 認証エラーに関するユーザーフレンドリーなエラーメッセージを追加しました。これらのメッセージには、IAM ユーザーまたはロール ARN、リソース ARN、およびリクエストに対する不正アクションのリストが表示されるようになりました。不正アクションのリストは、使用している IAM ポリシーに何が欠けているか、または明示的に拒否されているかを確認するのに役立ちます。

このエンジンリリースで修正された不具合

- TinkerPop 3.5 にアップグレードした後に `PartitionStrategy` を使用すると、「PartitionStrategy は匿名トラバーサルでは機能しません」というメッセージを含むエラーが誤って発生し、トラバーサルが実行されなくなるという Gremlin のバグを修正しました。
- `WherePredicateStep` 変換に関する Gremlin の正確性に関するバグを修正しました。このバグでは、Neptune のクエリエンジンが、`where(P.neq('x'))` とそのバリエーションを使用するクエリについて誤った結果を生成していました。
- ノードとエッジが重複して作成されることがあった `MERGE` 句内の openCypher のバグが修正されました。
- `OPTIONAL` 句内に（`NOT`）`EXISTS` を含むクエリの処理において、クエリの結果が表示されない場合がある SPARQL のバグを修正しました。
- 挿入負荷が高い場合にパフォーマンスが低下するバルクローダーのバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.2.0.1 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: `3.5.2`
- サポートされている最も新しいバージョンの Gremlin: `3.5.4`
- openCypher バージョン: `Neptune-9.0.20190305-1.0`
- SPARQL バージョン: `1.1`

エンジンリリース 1.2.0.1 へのアップグレードパス

このリリースへのアップグレード

Amazon Neptune 1.2.0.1 が公開されました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.0.1 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.0.1 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the instance before
>   proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.2.0.1.R3 (2023-09-27)

2023 年 9 月 27 日現在、エンジンバージョン 1.2.0.1.R3 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

> **Note**
>
> 1.2.0.0 より前のエンジンバージョンからアップグレードを行う場合:

- エンジンリリース 1.2.0.0 では、カスタムパラメータグループとカスタムクラスターパラメータグループに新しい形式が導入されました。そのため、1.2.0.0 より前のエンジンバージョンからエンジンバージョン 1.2.0.0 以降にアップグレードする場合は、パラメータグループファミリー neptune1.2 を使用している既存のカスタムパラメータグループとカスタムクラスターパラメータグループをすべて再作成する必要があります。以前のリリースではパラメータグループファミリー neptune1 が使用されていましたが、それらのパラメータグループはリリース 1.2.0.0 以降では動作しません。詳細については、「Amazon Neptune パラメータグループ」を参照してください。
- エンジンリリース 1.2.0.0 では、UNDO ログの新しい形式も導入されました。そのため、1.2.0.0 より前のバージョンからのアップグレードを開始する前に、以前のエンジンバージョンで作成された UNDO ログをすべてパージし、`UndoLogsListSize` CloudWatch メトリクスをゼロにする必要があります。更新を開始しようとしたときに UNDO ログレコードが多すぎる (200,000 以上) 場合、UNDO ログのパージが完了するのを待っている間にアップグレードがタイムアウトすることがあります。

  パージが行われるクラスターのライターインスタンスをアップグレードすることで、パージの速度を上げることができます。アップグレードを試みる前にこれを行うと、開始前に UNDO ログの数を減らすことができます。ライターのサイズを 24XL インスタンスタイプに増やすと、パージ率が 1 時間あたり 100 万レコードを超えることがあります。

  UndoLogsListSize CloudWatch メトリクスが非常に大きい場合、サポートケースを開くと、それを停止するための追加の戦略を検討するのに役立つ場合があります。
- 最後に、リリース 1.2.0.0 には、IAM 認証で Bolt プロトコルを使用していた以前のコードに影響する重大な変更がありました。リリース 1.2.0.0 以降、Bolt には IAM 署名用のリソースパスが必要です。Java では、リソースパスの設定は以下のようになります: `request.setResourcePath("/openCypher"));`。その他の言語では、`/openCypher` をエンドポイント URI に追加できます。例については、「Bolt プロトコルの使用」を参照してください。

このエンジンリリースの改良点

- すべての Neptune ML API に `enableInterContainerTrafficEncryption` パラメータを追加しました。これを使用して、トレーニングジョブやハイパーパラメータチューニングジョブでコンテナ間トラフィック暗号化を有効または無効にできます。

- サーバーレス DB クラスターでは、最小容量設定を 1.0 NCU に、最小有効最大設定を 2.5 NCU に変更しました。「Neptune サーバーレス DB クラスターの容量スケーリング (((リリース前に、この変更をサーバーレスページにも反映する必要があります)))」を参照してください。

このエンジンリリースで修正された不具合

- 更新とリターンのクエリが `orderBy`、`limit`、または `skip` を正しく処理しない openCypher のバグを修正しました。
- あるリクエストに含まれるパラメータが、別の同時リクエストに含まれるパラメータによってオーバーライドされる可能性がある openCypher のバグを修正しました。
- Bolt のトランザクション処理における openCypher のバグを修正しました。
- Tinkerpop にフォールバックして `limit` を非結合ステップの子トラバーサルとして持つ DFE クエリの Gremlin の正確性の問題を修正しました。例えば、次のようなクエリの場合:

```
g.withSideEffect('Neptune#useDFE', true)
 .V()
 .as("a")
 .select("a")
 .by(out()
 .limit(1))
```

- TinkerPop のステップが多すぎるためにクエリが失敗し、クリーンアップされないという Gremlin のバグを修正しました。
- 文字列出力の一部にスペース文字が含まれていると `order()` によって正しくソートされない Gremlin のバグを修正しました。
- クエリが文字列として送信され、`GroupCountStep` を含んでいたときに、トランザクションリークが発生する可能性がある Gremlin のバグを修正しました。
- ネイティブに処理されないステップの子トラバーサルに述語を含むクエリについて、Gremlin クエリステータスエンドポイントを確認すると、トランザクションリークが発生する Gremlin のバグを修正しました。
- `inV()` または `outV()` が続くエッジとそのプロパティを追加すると `InternalFailureException` が発生する Gremlin のバグを修正しました。
- ストレージレイヤーの同時実行の問題を修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.2.1.R3 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: `3.5.2`
- サポートされている最も新しいバージョンの Gremlin: `3.5.6`
- openCypher バージョン: `Neptune-9.0.20190305-1.0`
- SPARQL バージョン: `1.1`

エンジンリリース 1.2.0.1.R3 へのアップグレードパス

エンジンバージョン 1.2.0.1 を実行している場合、Neptune DB クラスターは次のメンテナンスウィンドウで自動的にこのパッチリリースにアップグレードされます。

このリリースへのアップグレード

Amazon Neptune 1.0.2.1.R3 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.0.1 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.0.1 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
> ```

```
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.2.0.1.R2 (2022-12-13)

2022 年 12 月 13 日現在、エンジンバージョン 1.2.0.1.R2 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

Note

1.2.0.0 より前のエンジンバージョンからアップグレードを行う場合:

- エンジンリリース 1.2.0.0 では、カスタムパラメータグループとカスタムクラスターパラメータグループに新しい形式が導入されました。そのため、1.2.0.0 より前のエンジンバージョンからエンジンバージョン 1.2.0.0 以降にアップグレードする場合は、パラメータグループファミリー `neptune1.2` を使用している既存のカスタムパラメータグループとカスタムクラスターパラメータグループをすべて再作成する必要があります。以前のリリースではパラメータグループファミリー `neptune1` が使用されていましたが、それらのパラメータグループはリリース 1.2.0.0 以降では動作しません。詳細については、「Amazon Neptune パラメータグループ」を参照してください。
- エンジンリリース 1.2.0.0 では、UNDO ログの新しい形式も導入されました。そのため、1.2.0.0 より前のバージョンからのアップグレードを開始する前に、以前のエンジンバージョンで作成された UNDO ログをすべてパージし、`UndoLogsListSize` CloudWatch メトリクスをゼロにする必要があります。更新を開始しようとしたときに UNDO ログレコードが多すぎる (200,000 以上) 場合、UNDO ログのパージが完了するのを待っている間にアップグレードがタイムアウトすることがあります。

  パージが行われるクラスターのライターインスタンスをアップグレードすることで、パージの速度を上げることができます。アップグレードを試みる前にこれを行うと、開始前に

UNDO ログの数を減らすことができます。ライターのサイズを 24XL インスタンスタイプに増やすと、パージ率が 1 時間あたり 100 万レコードを超えることがあります。

UndoLogsListSize CloudWatch メトリクスが非常に大きい場合、サポートケースを開くと、それを停止するための追加の戦略を検討するのに役立つ場合があります。

- 最後に、リリース 1.2.0.0 には、IAM 認証で Bolt プロトコルを使用していた以前のコードに影響する重大な変更がありました。リリース 1.2.0.0 以降、Bolt には IAM 署名用のリソースパスが必要です。Java では、リソースパスの設定は以下のようになります: `request.setResourcePath("/openCypher"));`。その他の言語では、`/openCypher` をエンドポイント URI に追加できます。例については、「Bolt プロトコルの使用」を参照してください。

このエンジンリリースの改良点

- リテラル値のマップのリストの `UNWIND` に関係する openCypher クエリのパフォーマンスが向上しました。
- `repeat`、`coalesce`、`store`、`aggregate` など、さまざまな Gremlin 演算子のパフォーマンスの向上と正確性の修正を行いました。

このエンジンリリースで修正された不具合

- Bolt と SPARQL-JSON でクエリが NULL 値ではなく文字列 `"null"` を返す openCypher のバグを修正しました。
- 不要な情報がログに記録されたり、特定のフィールドがログから欠落したりする監査ログのバグを修正しました。
- ルックアップキャッシュのバグを修正して、キャッシュへの書き込みに使用される増分メモリを制限しました。
- 書き込みが失敗したときにルックアップキャッシュに読み取り専用モードを設定するというルックアップキャッシュのバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.2.0.1.R2 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: `3.5.2`
- サポートされている最も新しいバージョンの Gremlin: `3.5.4`
- openCypher バージョン: `Neptune-9.0.20190305-1.0`
- SPARQL バージョン: `1.1`

エンジンリリース 1.2.0.1.R2 へのアップグレードパス

エンジンバージョン 1.2.0.1 を実行している場合、Neptune DB クラスターは次のメンテナンスウィンドウで自動的にこのパッチリリースにアップグレードされます。

このリリースへのアップグレード

Amazon Neptune 1.2.0.1.R2 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.0.1 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.0.1 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のような エラーが発生する可能性があります。

```
  We're sorry, your request to modify DB cluster (cluster identifier) has
failed.
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.2.0.0 (2022-07-21)

2022 年 7 月 21 日現在、エンジンバージョン 1.2.0.0 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

Note

1.2.0.0 より前のエンジンバージョンからアップグレードを行う場合:

- エンジンリリース 1.2.0.0 では、カスタムパラメータグループとカスタムクラスターパラメータグループに新しい形式が導入されました。そのため、1.2.0.0 より前のエンジンバージョンからエンジンバージョン 1.2.0.0 以降にアップグレードする場合は、パラメータグループファミリー `neptune1.2` を使用している既存のカスタムパラメータグループとカスタムクラスターパラメータグループをすべて再作成する必要があります。以前のリリースではパラメータグループファミリー `neptune1` が使用されていましたが、それらのパラメータグループはリリース 1.2.0.0 以降では動作しません。詳細については、「Amazon Neptune パラメータグループ」を参照してください。
- エンジンリリース 1.2.0.0 では、UNDO ログの新しい形式も導入されました。そのため、1.2.0.0 より前のバージョンからのアップグレードを開始する前に、以前のエンジンバージョンで作成された UNDO ログをすべてパージし、`UndoLogsListSize` CloudWatch メトリクスをゼロにする必要があります。更新を開始しようとしたときに UNDO ログレコードが多すぎる (200,000 以上) 場合、UNDO ログのパージが完了するのを待っている間にアップグレードがタイムアウトすることがあります。

  パージが行われるクラスターのライターインスタンスをアップグレードすることで、パージの速度を上げることができます。アップグレードを試みる前にこれを行うと、開始前に UNDO ログの数を減らすことができます。ライターのサイズを 24XL インスタンスタイプに増やすと、パージ率が 1 時間あたり 100 万レコードを超えることがあります。

UndoLogsListSize CloudWatch メトリクスが非常に大きい場合、サポートケースを開くと、それを停止するための追加の戦略を検討するのに役立つ場合があります。

- 最後に、リリース 1.2.0.0 には、IAM 認証で Bolt プロトコルを使用していた以前のコードに影響する重大な変更がありました。リリース 1.2.0.0 以降、Bolt には IAM 署名用のリソースパスが必要です。Java では、リソースパスの設定は以下のようになります: `request.setResourcePath("/openCypher"));`。その他の言語では、`/openCypher` をエンドポイント URI に追加できます。例については、「Bolt プロトコルの使用」を参照してください。

このリリースの後続のパッチリリース

- リリース： 1.2.0.0.R2 (2022-10-14)
- リリース: 1.2.0.0.R3 (2022-12-15)
- リリース: 1.2.0.0.R4 (2023-09-29)

このエンジンリリースの新機能

- グローバルデータベースのサポートが追加されました。Neptune グローバルデータベースは複数の AWS リージョン にまたがり、1 つのリージョンにあるプライマリ DB クラスターと、他のリージョンにある最大 5 つのセカンダリ DB クラスターで構成されます。
- Neptune IAM ポリシーに、データプレーンアクションに基づく、以前よりもきめ細かいアクセスコントロールのサポートが追加されました。廃止された `connect` アクションに基づく既存の IAM ポリシーを、より詳細なデータプレーンアクションを使用するように調整する必要があるという点で、これは重大な変更です。「IAM ポリシーのタイプ」を参照してください。
- リーダーインスタンスの可用性が向上しました。これまでは、ライターインスタンスが再起動すると、Neptune クラスター内のすべてのリーダーインスタンスも再起動していました。エンジンリリース 1.2.0.0 以降、ライターの再起動後もリーダーインスタンスはアクティブなままになり、リーダーの可用性が向上しました。リーダーインスタンスはパラメータグループの変更を反映するために個別に再起動できます。「Amazon Neptune における DB インスタンスの再起動」を参照してください。
- 新しい neptune_streams_expiry_days DB クラスターパラメータが追加されました。これにより、ストリームレコードが削除されるまでにサーバー上に保持される日数を設定できます。範囲は 1 ~ 90 で、デフォルトは 7 です。

このエンジンリリースの改良点

- ByteCode クエリの Gremlin シリアル化のパフォーマンスが向上しました。
- Neptune は DFE エンジンを使用してテキスト述語を処理するようになり、パフォーマンスが向上しました。
- Neptune は、非終端制限や子トラバーサル制限を含め、DFE エンジンを使用して Gremlin `limit()` ステップを処理するようになりました。
- Gremlin `union()` ステップの DFE 処理を他の新機能と連携するように変更しました。つまり、リファレンスノードがクエリプロファイルに期待どおりに表示されるようになりました。
- DFE 内の一部の高価な結合操作を並列化することで、パフォーマンスが最大 5 倍向上しました。
- Gremlin DFE エンジンの `OrderGlobalStep order(global)` について `by()` モジュレーションサポートが追加されました。
- DFE の詳細説明に、注入された静的な値の表示を追加しました。
- 重複するパターンを削除するときのパフォーマンスが向上しました。
- Gremlin DFE エンジンに順序保持サポートが追加されました。
- 次のような空のフィルターを含む Gremlin クエリのパフォーマンスが向上しました。

```
g.V().hasId(P.within([]))
```

```
g.V().hasId([])
```

- SPARQL クエリが Neptune が内部的に表現するには大きすぎる数値を使用する場合のエラーメッセージを改善しました。
- ストリームが無効になっているときにインデックス検索を減らすことで、エッジが関連付けられた頂点を削除するときのパフォーマンスが向上しました。
- DFE サポートを `has()` ステップのより多くのバリアント、特に `hasKey()`、`hasLabel()` と、`has()` 内の文字列/URI の範囲述語に拡張しました。これは次のようなクエリに影響します。

```
// hasKey() on properties
g.V().properties().hasKey("name")
g.V().properties().has(T.key, TextP.startingWith("a"))
g.E().properties().hasKey("weight")
g.E().properties().hasKey(TextP.containing("t"))

// hasLabel() on vertex properties
g.V().properties().hasLabel("name")
```

```
// range predicates on ID and Label fields
g.V().has(T.label, gt("person"))
g.E().has(T.id, lte("(an ID value)"))
```

- リスト内の文字列を単一の文字列に連結する NepTune 固有の openCypher `join()` 関数を追加しました。
- データアクセス許可と新しいグローバルデータベース API のアクセス許可を含むように Neptune 管理ポリシーを更新しました。

このエンジンリリースで修正された不具合

- コンテンツタイプが指定されていない HTTP リクエストが自動的に失敗するバグを修正しました。
- クエリー内でサービスコールが使用できないというクエリーオプティマイザーの SPARQL バグを修正しました。
- Unicode データの特定の組み合わせが原因で障害が発生するという Turtle RDF パーサーの SPARQL バグを修正しました。
- GRAPH および SELECT 句の特定の組み合わせが誤ったクエリ結果を生成する SPARQL のバグを修正しました。
- 次のような結合ステップ内のフィルターステップを使用するクエリで正確性の問題が発生する Gremlin のバグを修正しました。

```
g.V("1").union(hasLabel("person"), out())
```

- `both().simplePath()` の `count()` によって、`count()` なしで返される結果の数が実際の 2 倍になるという Gremlin のバグを修正しました。
- IAM 認証が有効になっているクラスターへの Bolt リクエストに対して、サーバーで署名不一致の例外が生成される openCypher のバグを修正しました。
- HTTP キープアライブを使用するクエリが、リクエストが失敗した後に送信された場合に誤ってクローズされる可能性がある openCypher のバグを修正しました。
- 定数値を返すクエリが送信されると内部エラーがスローされる openCypher のバグを修正しました。
- 説明の詳細のバグが修正され、DFE サブクエリ `Time(ms)` が DFE サブクエリ内のオペレータの CPU 時間を正しく合計するようになりました。例として、次の説明出力の抜粋を考えてみましょう。

```
subQuery1
##########################################################################################
# ID # Out #1 # Out #2 # Name                   # Arguments                             #
 Mode # Units In # Units Out # Ratio # Time (ms) #
##########################################################################################
  ...
##########################################################################################
# 1  # 2      # -      # DFEChunkLocalSubQuery # subQuery=...graph#336e.../graph_1 #
 -    # 1        # 1         # 1.00  # 0.38      #
#    #        #        #                       # coordinationTime(ms)=0.026        #
     #          #           #       #           #
##########################################################################################
  ...
subQuery=...graph#336e.../graph_1
##########################################################################################
# ID # Out #1 # Out #2 # Name                   # Arguments                             #
 Mode # Units In # Units Out # Ratio # Time (ms) #
##########################################################################################
# 0  # 1      # -      # DFESolutionInjection  # solutions=[?100 -> [-10^^<LONG>]] #
 -    # 0        # 1         # 0.00  # 0.04      #
#    #        #        #                       # outSchema=[?100]                  #
     #          #           #       #           #
##########################################################################################
# 1  # 3      # -      # DFERelationalJoin     # joinVars=[]                         #
 -    # 2        # 1         # 0.50  # 0.29      #
##########################################################################################
# 2  # 1      # -      # DFESolutionInjection  # outSchema=[]                       #
 -    # 0        # 1         # 0.00  # 0.01      #
##########################################################################################
# 3  # -      # -      # DFEDrain              # -                                   #
 -    # 1        # 0         # 0.00  # 0.02      #
##########################################################################################
```

下の表の最後の列のサブクエリ時間は、合計で 0.36 ms (`.04 + .29 + .01 + .02 = .36`) になります。そのサブクエリの調整時間を追加すると (`.36 + .026 = .386`) 、上のテーブルの最後の列に記録されているサブクエリの時間、つまり `0.38` ms に近い結果が得られます。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.2.0.0 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: `3.5.2`
- サポートされている最も新しいバージョンの Gremlin: `3.5.4`
- openCypher バージョン: `Neptune-9.0.20190305-1.0`
- SPARQL バージョン: `1.1`

エンジンリリース 1.2.0.0 へのアップグレードパス

これはエンジンのメジャーリリースなので、自動アップグレードはありません。

エンジンリリース 1.1.1.0 の最新のパッチリリースからリリース `1.2.0.0` へのアップグレードは手動でのみ可能です。`1.2.0.0` にアップグレードするには、その前に、以前のエンジンリリースを `1.1.1.0` の最新リリースにアップグレードする必要があります。

そのため、このリリースにアップグレードする前に、リリース `1.1.1.0` の最新パッチリリースを現在実行していることを確認してください。そうでない場合は、`1.1.1.0` の最新パッチリリースにアップグレードすることから始めてください。

アップグレードする前に、パラメータグループファミリー `neptune1.2` を使用して、以前のバージョンで使用していたカスタム DB クラスターパラメータグループを再作成する必要もあります。詳細については、「Amazon Neptune パラメータグループ」を参照してください。

最初にリリース `1.1.1.0` にアップグレードし、その後すぐに `1.2.0.0` にアップグレードする場合、次のようなエラーが発生する可能性があります。

```
  We're sorry, your request to modify DB cluster (cluster identifier) has failed.
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます (「Amazon Neptune DB クラスターのメンテナンス」を参照)。

このリリースへのアップグレード

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.0.0 \
    --allow-major-version-upgrade \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.0.0 ^
    --allow-major-version-upgrade ^
    --apply-immediately
```

`--apply-immediately` の代わりに `--no-apply-immediately` と指定することができます。メジャーバージョンアップグレードを実行するには、allow-major-version-upgrade パラメータが必要です。また、エンジンバージョンを含めるようにしてください。そうしないと、エンジンが別のバージョンにアップグレードされる可能性があります。

クラスターでカスタムクラスターパラメータグループを使用する場合は、必ずこのパラメータを含めて、それを指定してください。

```
    --db-cluster-parameter-group-name (name of the custom DB cluster parameter group)
```

同様に、クラスター内のインスタンスがカスタム DB のパラメータグループを使用している場合は、必ずこのパラメータを指定して、次のようになります。

```
    --db-instance-parameter-group-name (name of the custom instance parameter group)
```

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.2.0.0.R4 (2023-09-29)

2023 年 9 月 29 日現在、エンジンバージョン1.2.0.0.R4 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

Note

1.2.0.0 より前のエンジンバージョンからアップグレードを行う場合:

- エンジンリリース 1.2.0.0 では、カスタムパラメータグループとカスタムクラスターパラメータグループに新しい形式が導入されました。そのため、1.2.0.0 より前のエンジンバージョンからエンジンバージョン 1.2.0.0 以降にアップグレードする場合は、パラメータグループファミリー `neptune1.2` を使用している既存のカスタムパラメータグループとカスタムクラスターパラメータグループをすべて再作成する必要があります。以前のリリースではパラメータグループファミリー `neptune1` が使用されていましたが、それらのパラメータグループはリリース 1.2.0.0 以降では動作しません。詳細については、「Amazon Neptune パラメータグループ」を参照してください。
- エンジンリリース 1.2.0.0 では、UNDO ログの新しい形式も導入されました。そのため、1.2.0.0 より前のバージョンからのアップグレードを開始する前に、以前のエンジンバージョンで作成された UNDO ログをすべてパージし、`UndoLogsListSize` CloudWatch メトリクスをゼロにする必要があります。更新を開始しようとしたときに UNDO ログレコードが多すぎる (200,000 以上) 場合、UNDO ログのパージが完了するのを待っている間にアップグレードがタイムアウトすることがあります。

  パージが行われるクラスターのライターインスタンスをアップグレードすることで、パージの速度を上げることができます。アップグレードを試みる前にこれを行うと、開始前に UNDO ログの数を減らすことができます。ライターのサイズを 24XL インスタンスタイプに増やすと、パージ率が 1 時間あたり 100 万レコードを超えることがあります。

UndoLogsListSize CloudWatch メトリクスが非常に大きい場合、サポートケースを開くと、それを停止するための追加の戦略を検討するのに役立つ場合があります。

- 最後に、リリース 1.2.0.0 には、IAM 認証で Bolt プロトコルを使用していた以前のコードに影響する重大な変更がありました。リリース 1.2.0.0 以降、Bolt には IAM 署名用のリソースパスが必要です。Java では、リソースパスの設定は以下のようになります: `request.setResourcePath("/openCypher"));`。その他の言語では、`/openCypher` をエンドポイント URI に追加できます。例については、「Bolt プロトコルの使用」を参照してください。

このエンジンリリースの改良点

- すべての Neptune ML API に `enableInterContainerTrafficEncryption` パラメータを追加しました。これを使用して、トレーニングジョブやハイパーパラメータチューニングジョブでコンテナ間トラフィック暗号化を有効または無効にできます。
- サーバーレス DB クラスターでは、最小容量設定を 1.0 NCU に、最小有効最大設定を 2.5 NCU に変更しました。「Neptune サーバーレス DB クラスターの容量スケーリング (((リリース前に、この変更をサーバーレスページにも反映する必要があります)))」を参照してください。

このエンジンリリースで修正された不具合

- 更新とリターンのクエリが `orderBy`、`limit`、または `skip` を正しく処理しない openCypher のバグを修正しました。
- あるリクエストに含まれるパラメータが、別の同時リクエストに含まれるパラメータによってオーバーライドされる可能性がある openCypher のバグを修正しました。
- Bolt のトランザクション処理における openCypher のバグを修正しました。
- Tinkerpop にフォールバックして `limit` を非結合ステップの子トラバーサルとして持つ DFE クエリの Gremlin の正確性の問題を修正しました。例えば、次のようなクエリの場合:

```
g.withSideEffect('Neptune#useDFE', true)
 .V()
 .as("a")
 .select("a")
 .by(out()
 .limit(1))
```

- TinkerPop のステップが多すぎるためにクエリが失敗し、クリーンアップされないという Gremlin のバグを修正しました。
- 文字列出力の一部にスペース文字が含まれていると `order()` によって正しくソートされない Gremlin のバグを修正しました。
- クエリが文字列として送信され、`GroupCountStep` を含んでいたときに、トランザクションリークが発生する可能性がある Gremlin のバグを修正しました。
- ネイティブに処理されないステップの子トラバーサルに述語を含むクエリについて、Gremlin クエリステータスエンドポイントを確認すると、トランザクションリークが発生する Gremlin のバグを修正しました。
- `inV()` または `outV()` が続く エッジとそのプロパティを追加すると `InternalFailureException` が発生する Gremlin のバグを修正しました。
- ストレージレイヤーの同時実行の問題を修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.2.0.0.R4 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: `3.5.2`
- サポートされている最も新しいバージョンの Gremlin: `3.5.6`
- openCypher バージョン: `Neptune-9.0.20190305-1.0`
- SPARQL バージョン: `1.1`

エンジンリリース 1.2.0.0.R4 へのアップグレードパス

エンジンバージョン `1.2.0.0` を実行している場合、クラスターは次のメンテナンス期間中に自動的にこのパッチリリースにアップグレードされます。

エンジンリリース 1.1.1.0 の最新のパッチリリースからリリース `1.2.0.0` へのアップグレードは手動でのみ可能です。`1.2.0.0` にアップグレードするには、その前に、以前のエンジンリリースを `1.1.1.0` の最新リリースにアップグレードする必要があります。

最初にリリース `1.1.1.0` にアップグレードし、その後すぐに `1.2.0.0` にアップグレードする場合、次のようなエラーが発生する可能性があります。

```
We're sorry, your request to modify DB cluster (cluster identifier) has failed.
```

```
    Cannot modify engine version because instance (instance identifier) is
    running on an old configuration. Apply any pending maintenance actions on the
  instance before
    proceeding with the upgrade.
```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

このリリースへのアップグレード

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.0.0 \
    --allow-major-version-upgrade \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.0.0 ^
    --allow-major-version-upgrade ^
    --apply-immediately
```

`--apply-immediately` の代わりに `--no-apply-immediately` と指定することができます。メジャーバージョンアップグレードを実行するには、allow-major-version-upgrade パラメータが必要です。また、エンジンバージョンを含めるようにしてください。そうしないと、エンジンが別のバージョンにアップグレードされる可能性があります。

クラスターでカスタムクラスターパラメータグループを使用する場合は、必ずこのパラメータを含めて、それを指定してください。

```
    --db-cluster-parameter-group-name (name of the custom DB cluster parameter group)
```

同様に、クラスター内のインスタンスがカスタム DB のパラメータグループを使用している場合は、必ずこのパラメータを指定して、次のようになります。

```
--db-instance-parameter-group-name (name of the custom instance parameter group)
```

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
> ```

```
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.2.0.0.R3 (2022-12-15)

2022 年 12 月 15 日現在、エンジンバージョン 1.2.0.0.R3 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

Note

1.2.0.0 より前のエンジンバージョンからアップグレードを行う場合:

- エンジンリリース 1.2.0.0 では、カスタムパラメータグループとカスタムクラスターパラメータグループに新しい形式が導入されました。そのため、1.2.0.0 より前のエンジンバージョンからエンジンバージョン 1.2.0.0 以降にアップグレードする場合は、パラメータグループファミリー `neptune1.2` を使用している既存のカスタムパラメータグループとカスタムクラスターパラメータグループをすべて再作成する必要があります。以前のリリースではパラメータグループファミリー `neptune1` が使用されていましたが、それらのパラメータグループはリリース 1.2.0.0 以降では動作しません。詳細については、「Amazon Neptune パラメータグループ」を参照してください。
- エンジンリリース 1.2.0.0 では、UNDO ログの新しい形式も導入されました。そのため、1.2.0.0 より前のバージョンからのアップグレードを開始する前に、以前のエンジンバージョンで作成された UNDO ログをすべてパージし、`UndoLogsListSize` CloudWatch メトリクスをゼロにする必要があります。更新を開始しようとしたときに UNDO ログレコードが多すぎる (200,000 以上) 場合、UNDO ログのパージが完了するのを待っている間にアップグレードがタイムアウトすることがあります。

  パージが行われるクラスターのライターインスタンスをアップグレードすることで、パージの速度を上げることができます。アップグレードを試みる前にこれを行うと、開始前に

UNDO ログの数を減らすことができます。ライターのサイズを 24XL インスタンスタイプに増やすと、パージ率が 1 時間あたり 100 万レコードを超えることがあります。

UndoLogsListSize CloudWatch メトリクスが非常に大きい場合、サポートケースを開くと、それを停止するための追加の戦略を検討するのに役立つ場合があります。

- 最後に、リリース 1.2.0.0 には、IAM 認証で Bolt プロトコルを使用していた以前のコードに影響する重大な変更がありました。リリース 1.2.0.0 以降、Bolt には IAM 署名用のリソースパスが必要です。Java では、リソースパスの設定は以下のようになります: `request.setResourcePath("/openCypher"));`。その他の言語では、/openCypher をエンドポイント URI に追加できます。例については、「Bolt プロトコルの使用」を参照してください。

このエンジンリリースの改良点

- `MERGE` と `OPTIONAL MATCH` を含む openCypher クエリのパフォーマンスが向上しました。
- リテラル値のマップのリストの `UNWIND` に関係する openCypher クエリのパフォーマンスが向上しました。
- `id` の `IN` フィルターの付いた openCypher クエリのパフォーマンスが向上しました。例:

```
MATCH (n) WHERE id(n) IN ['1', '2', '3'] RETURN n
```

- `repeat`、`coalesce`、`store`、`aggregate` など、さまざまな Gremlin 演算子のパフォーマンスの向上と正確性の修正を行いました。

このエンジンリリースで修正された不具合

- Bolt と SPARQL-JSON でクエリが NULL 値ではなく文字列 `"null"` を返す openCypher のバグを修正しました。
- openCypher のバグを修正し、値がリストまたはマップのリストの場合にパラメータタイプを正しく解釈できるようにしました。
- 不要な情報がログに記録されたり、特定のフィールドがログから欠落したりする監査ログのバグを修正しました。
- IAM 対応 DB クラスターへの HTTP リクエストの IAM ARN が記録されない監査ログのバグを修正しました。

- ルックアップキャッシュのバグを修正して、キャッシュへの書き込みに使用される増分メモリを制限しました。
- 書き込みが失敗したときにルックアップキャッシュに読み取り専用モードを設定するというルックアップキャッシュのバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.2.0.0.R3 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: `3.5.2`
- サポートされている最も新しいバージョンの Gremlin: `3.5.4`
- openCypher バージョン: `Neptune-9.0.20190305-1.0`
- SPARQL バージョン: `1.1`

エンジンリリース 1.2.0.0.R3 へのアップグレードパス

エンジンバージョン `1.2.0.0` を実行している場合、クラスターは次のメンテナンス期間中に自動的にこのパッチリリースにアップグレードされます。

エンジンリリース 1.1.1.0 の最新のパッチリリースからリリース `1.2.0.0` へのアップグレードは手動でのみ可能です。`1.2.0.0` にアップグレードするには、その前に、以前のエンジンリリースを `1.1.1.0` の最新リリースにアップグレードする必要があります。

最初にリリース `1.1.1.0` にアップグレードし、その後すぐに `1.2.0.0` にアップグレードする場合、次のようなエラーが発生する可能性があります。

```
  We're sorry, your request to modify DB cluster (cluster identifier) has failed.
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

このリリースへのアップグレード

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.0.0 \
    --allow-major-version-upgrade \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.0.0 ^
    --allow-major-version-upgrade ^
    --apply-immediately
```

`--apply-immediately` の代わりに `--no-apply-immediately` と指定することができます。メジャーバージョンアップグレードを実行するには、allow-major-version-upgrade パラメータが必要です。また、エンジンバージョンを含めるようにしてください。そうしないと、エンジンが別のバージョンにアップグレードされる可能性があります。

クラスターでカスタムクラスターパラメータグループを使用する場合は、必ずこのパラメータを含めて、それを指定してください。

```
    --db-cluster-parameter-group-name (name of the custom DB cluster parameter group)
```

同様に、クラスター内のインスタンスがカスタム DB のパラメータグループを使用している場合は、必ずこのパラメータを指定して、次のようになります。

```
    --db-instance-parameter-group-name (name of the custom instance parameter group)
```

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のような エラーが発生する可能性があります。

```
  We're sorry, your request to modify DB cluster (cluster identifier) has
failed.
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.2.0.0.R2 (2022-10-14)

2022 年 10 月 14 日現在、エンジンバージョン 1.2.0.0.R2 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

Note

1.2.0.0 より前のエンジンバージョンからアップグレードを行う場合:

- エンジンリリース 1.2.0.0 では、カスタムパラメータグループとカスタムクラスターパラメータグループに新しい形式が導入されました。そのため、1.2.0.0 より前のエンジンバージョンからエンジンバージョン 1.2.0.0 以降にアップグレードする場合は、パラメータグループファミリー `neptune1.2` を使用している既存のカスタムパラメータグループとカスタムクラスターパラメータグループをすべて再作成する必要があります。以前のリリースではパラメータグループファミリー `neptune1` が使用されていましたが、それらのパラメータグループはリリース 1.2.0.0 以降では動作しません。詳細については、「Amazon Neptune パラメータグループ」を参照してください。
- エンジンリリース 1.2.0.0 では、UNDO ログの新しい形式も導入されました。そのため、1.2.0.0 より前のバージョンからのアップグレードを開始する前に、以前のエンジンバージョンで作成された UNDO ログをすべてパージし、`UndoLogsListSize` CloudWatch メトリクスをゼロにする必要があります。更新を開始しようとしたときに UNDO ログレコードが多すぎる (200,000 以上) 場合、UNDO ログのパージが完了するのを待っている間にアップグレードがタイムアウトすることがあります。

  パージが行われるクラスターのライターインスタンスをアップグレードすることで、パージの速度を上げることができます。アップグレードを試みる前にこれを行うと、開始前に UNDO ログの数を減らすことができます。ライターのサイズを 24XL インスタンスタイプに増やすと、パージ率が 1 時間あたり 100 万レコードを超えることがあります。

UndoLogsListSize CloudWatch メトリクスが非常に大きい場合、サポートケースを開くと、それを停止するための追加の戦略を検討するのに役立つ場合があります。

- 最後に、リリース 1.2.0.0 には、IAM 認証で Bolt プロトコルを使用していた以前のコードに影響する重大な変更がありました。リリース 1.2.0.0 以降、Bolt には IAM 署名用のリソースパスが必要です。Java では、リソースパスの設定は以下のようになります: `request.setResourcePath("/openCypher"));`。その他の言語では、`/openCypher` をエンドポイント URI に追加できます。例については、「Bolt プロトコルの使用」を参照してください。

このエンジンリリースの改良点

- Gremlin `order-by` クエリのパフォーマンスが改善されました。`NeptuneGraphQueryStep` の末尾に `order-by` が付いている Gremlin クエリは、パフォーマンスを向上させるために、より大きなサイズのチャンクを使用するようになりました。これはクエリプランの内部 (ルート以外の) ノードの `order-by` には適用されません。
- Gremlin 更新クエリのパフォーマンスが改善されました。エッジやプロパティを追加している間は、削除されないように頂点とエッジをロックする必要があります。この変更により、トランザクション内の重複ロックがなくなり、パフォーマンスが向上します。
- dedup をネイティブ実行レイヤーにプッシュすることで、`repeat()` サブクエリの内部の `dedup()` を使用する Gremlin クエリのパフォーマンスが向上しました。
- Gremlin `Neptune#cardinalityEstimates` クエリヒントを追加しました。`false` に設定すると、カーディナリティ推定が無効になります。
- IAM 認証エラーに関するユーザーフレンドリーなエラーメッセージを追加しました。これらのメッセージには、IAM ユーザーまたはロール ARN、リソース ARN、およびリクエストに対する不正アクションのリストが表示されるようになりました。不正アクションのリストは、使用している IAM ポリシーに何が欠けているか、または明示的に拒否されているかを確認するのに役立ちます。

このエンジンリリースで修正された不具合

- `WherePredicateStep` 変換に関する Gremlin の正確性に関するバグを修正しました。このバグでは、Neptune のクエリエンジンが、`where(P.neq('x'))` とそのバリエーションを使用するクエリについて誤った結果を生成していました。

- TinkerPop 3.5 にアップグレードした後に PartitionStrategy を使用すると、「PartitionStrategy は匿名トラバーサルでは機能しません」というメッセージを含むエラーが誤って発生し、トラバーサルが実行されなくなるという Gremlin のバグを修正しました。
- 最終結合の joinTime と Project.ASK サブグループ内の統計に関するさまざまな Gremlin のバグが修正されました。
- ノードとエッジが重複して作成されることがあった MERGE 句内の openCypher のバグが修正されました。
- 対応する同時辞書挿入がロールバックされた場合でも、セッションがグラフデータを挿入してコミットする可能性があるというトランザクションのバグを修正しました。
- 挿入負荷が高い場合にパフォーマンスが低下するバルクローダーのバグを修正しました。
- OPTIONAL 句内に（NOT）EXISTS を含むクエリの処理において、クエリの結果が表示されない場合がある SPARQL のバグを修正しました。
- 評価開始前のタイムアウトによりリクエストがキャンセルされた場合に、ドライバーがハングアップしたように見えるバグを修正しました。リクエストキュー内のアイテムでタイムアウトが発生している間に、サーバー上のすべてのクエリ処理スレッドが消費されると、この状態になる可能性がありました。リクエストキューからのタイムアウトによってメッセージがすぐに送信されるわけではなかったため、クライアントには応答が保留中のままのように見えました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.2.0.0.R2 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: 3.5.2
- サポートされている最も新しいバージョンの Gremlin: 3.5.4
- openCypher バージョン: Neptune-9.0.20190305-1.0
- SPARQL バージョン: 1.1

エンジンリリース 1.2.0.0.R2 へのアップグレードパス

エンジンバージョン 1.2.0.0 を実行している場合、クラスターは次のメンテナンス期間中に自動的にこのパッチリリースにアップグレードされます。

エンジンリリース 1.1.1.0 の最新のパッチリリースからリリース 1.2.0.0 へのアップグレードは手動でのみ可能です。1.2.0.0 にアップグレードするには、その前に、以前のエンジンリリースを 1.1.1.0 の最新リリースにアップグレードする必要があります。

最初にリリース 1.1.1.0 にアップグレードし、その後すぐに 1.2.0.0 にアップグレードする場合、次のようなエラーが発生する可能性があります。

```
  We're sorry, your request to modify DB cluster (cluster identifier) has failed.
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
 instance before
  proceeding with the upgrade.
```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

このリリースへのアップグレード

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.0.0 \
    --allow-major-version-upgrade \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.0.0 ^
    --allow-major-version-upgrade ^
    --apply-immediately
```

--apply-immediately の代わりに --no-apply-immediately と指定することができます。メジャーバージョンアップグレードを実行するには、allow-major-version-upgrade パラメータが必要で

す。また、エンジンバージョンを含めるようにしてください。そうしないと、エンジンが別のバージョンにアップグレードされる可能性があります。

クラスターでカスタムクラスターパラメータグループを使用する場合は、必ずこのパラメータを含めて、それを指定してください。

```
--db-cluster-parameter-group-name (name of the custom DB cluster parameter group)
```

同様に、クラスター内のインスタンスがカスタム DB のパラメータグループを使用している場合は、必ずこのパラメータを指定して、次のようになります。

```
--db-instance-parameter-group-name (name of the custom instance parameter group)
```

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後

に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
>   We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the
> instance before
>   proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.1.1.0 (2022-04-19)

2022 年 4 月 19 日現在、エンジンバージョン 1.1.1.0 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

> **Important**
>
> **`1.1.0.0`** より前のバージョンからこのエンジンリリースにアップグレードすると、DB クラスター内のすべてのインスタンスでオペレーティングシステムのアップグレードもトリガーされます。オペレーティングシステムのアップグレード中に発生するアクティブな書き込み要求は処理されないため、アップグレードを開始する前に、一括データロードを含め、アップグレード中のクラスターへのすべての書き込みワークロードを一時停止する必要があります。
> アップグレードを正常に完了するには、すべてのアベイラビリティーゾーン (AZ) の各サブネットに、Neptune インスタンスごとに利用可能な IP アドレスが少なくとも 1 つ必要で

す。例えば、サブネット 1 に 1 つのライターインスタンスと 2 つのリーダーインスタンスがあり、サブネット 2 に 2 つのリーダーインスタンスがある場合、アップグレードを開始する前に、サブネット 1 には少なくとも 3 つの IP アドレスの空きがあり、サブネット 2 には少なくとも 2 つの IP アドレスが空いている必要があります。
アップグレードの開始時に、Neptune は preupgrade に DB クラスターの情報に基づいて自動生成された識別子が続く名前のスナップショットを生成します。このスナップショットには課金はされません。アップグレードプロセス中に問題が発生した場合は、DB クラスターを復元するために使用できます。
エンジンのアップグレード自体が完了すると、古いオペレーティングシステムで新しいエンジンバージョンが一時的に利用可能になりますが、5 分以内にクラスター内のすべてのインスタンスが同時にオペレーティングシステムのアップグレードを開始します。DB クラスターは、この時点で数分間使用できなくなります。アップグレードの完了後に、書き込みワークロードを再開できます。
このプロセスでは、次のイベントが生成されます。

- クラスターごとのイベントメッセージ。
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-(autogenerated snapshot ID)]`
  - `Database cluster major version has been upgraded`
- インスタンスごとのイベントメッセージ。
  - `Applying off-line patches to DB instance`
  - `DB instance shutdown`
  - `Finished applying off-line patches to DB instance`
  - `DB instance restarted`

このリリースの後続のパッチリリース

- メンテナンスリリース: 1.1.0.0.R2 (2022-05-16)
- リリース: 1.1.1.0.R3 (2022-06-07)
- リリース: 1.1.1.0.R4 (2022-06-23)
- リリース: 1.1.1.0.R5 (2022-07-21)
- リリース: 1.1.1.0.R6 (2022-09-23)
- リリース: 1.1.1.0.R7 (2023-01-23)

このエンジンリリースの新機能

- openCypher クエリ言語は、本稼働環境用に一般公開されるようになりました。

> **Warning**
>
> このリリースでは、IAM 認証で openCypher を使用するコードに重大な変更が加えられています。openCypher の Neptune プレビューでは、IAM シグネチャのホスト文字列に `bolt://` など、次のようなプロトコルが含まれていました。
>
> ```
> "Host":"bolt://(host URL):(port)"
> ```
>
> このエンジンリリース以降、このプロトコルは省略する必要があります。
>
> ```
> "Host":"(host URL):(port)"
> ```
>
> 例については、「Bolt プロトコルの使用」を参照してください。

- TinkerPop `3.5.2` のサポートが追加されました。このリリースの変更点には、リモートトランザクションのサポート、セッション (を使用 `g.tx`) のバイトコードサポート、Gremlin 言語への `datetime()` 関数の追加などがあります。

> **Warning**
>
> TinkerPop 3.5.0、3.5.1、3.5.2 では、Gremlin コードに影響を与える可能性のあるいくつかの重大な変更が導入されています。例えば、GraphTraversalSource によって生成されたトラバーサルを `g.V().union(identity(), g.V())` のように子として使用しても機能しなくなります。
> 今度は、代わりに `g.V().union(identity(), __.V())` のような匿名トラバーサルを使用してください。

- IAM データアクセスポリシーで使用して、Neptune DB クラスターに保存されているデータへのアクセスを制御できるAWS グローバル条件キーのサポートが追加されました。
- Neptune DFE クエリエンジンは OpenCypher クエリ言語による本番環境での使用が可能になりましたが、Gremlin および SPARQL クエリではまだ使用できません。lab-mode パラメータではなく、独自の neptune_dfe_query_engine インスタンスパラメータを使用して有効化できるようになりました。

このエンジンリリースの改良点

- パラメータ化されたクエリサポート、パラメータ化されたクエリのための抽象構文ツリー (AST) キャッシュ、可変長パス (VLP) の改良、新しい演算子と句などの新機能が openCypher に追加されました。現在の言語サポートレベルについては、「Amazon Neptune での openCypher 仕様コンプライアンス Amazon Neptune」を参照してください。
- 単純な読み取り/書き込みワークロードの openCypher のパフォーマンスを大幅に改善し、リリース 1.1.0.0 と比較してスループットが向上しました。
- 可変長パスを処理する openCypher の双方向と深度の制限がなくなりました。
- DFE エンジンでの Gremlin `within` および `without` 述部のサポートが完了しました。他の述語演算子と組み合わせる場合も含みます。例:

```
g.V().has('age', within(12, 15, 18).or(gt(30)))
```

- スコープがグローバルであり (つまり `order(local)` ではない)、`by()` モジュレーターが使用されていない場合の DFE エンジンでの Gremlin `order` ステップのサポートが拡張されました。例えば、このクエリは DFE によってサポートされるようになりました。

```
 g.V().values("age").order()
```

- レコードがトランザクションの最後の操作であることを示すために、Neptune ストリーム変更ログレスポンス形式に `isLastOp` フィールドが追加されました。
- 監査ログのパフォーマンスが大幅に向上し、監査ログが有効になっている場合の待ち時間が短縮されました。
- Gremlin WebSocket バイトコードと HTTP クエリが、監査ログでユーザーが読み取り可能な形式に変換されました。クエリを監査ログから直接コピーして、Neptune ノートブックなどで実行できるようになりました。現在の監査ログ形式のこの変更は、重大な変更であることに注意してください。

このエンジンリリースで修正された不具合

- 次のクエリのように、ネストされた `filter()` および `count()` ステップを組み合わせて使用すると結果が返されないというまれな Gremlin のバグを修正しました。

```
g.V("1").filter(out("knows")
        .filter(in("knows")
```

```
        .hasId("notExists"))
        .count())
```

- 集約ステップによって保存された頂点を to() または from() トラバーサルで addE ステップと組み合わせて使用するとエラーが返される Gremlin のバグを修正しました。このようなクエリの例は以下のとおりです。

```
g.V("id").aggregate("v").out().addE().to(select("v").unfold()))
```

- DFE エンジンを使用しているときに not ステップがエッジケースで失敗する Gremlin のバグを修正しました。例:

```
g.V().not(V())
```

- to() および from() トラバーサル内で sideEffect 値が使用できないという Gremlin のバグを修正しました。
- 高速リセットによってインスタンスフェイルオーバーがトリガーされることがあったバグを修正しました。
- 失敗したトランザクションが次のロードジョブを開始する前にクローズされないバルクローダーのバグを修正しました。
- メモリ不足によりシステムがクラッシュするバルクローダーのバグを修正しました。
- フェイルオーバー後に IAM 認証情報が利用可能になるまでローダーが長時間待たなかったというバルクローダーのバグを修正するためのリトライを追加しました。
- status エンドポイントなど、クエリ以外のエンドポイントの内部認証情報キャッシュが適切にクリアされないバグを修正しました。
- ストリームのコミットのシーケンス番号が正しい順序になるようにストリームのバグを修正しました。
- IAM 対応クラスターで長時間実行される接続が 10 日より早く終了するバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.1.1.0 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: 3.5.2
- サポートされている最も新しいバージョンの Gremlin: 3.5.4

- openCypher バージョン: `Neptune-9.0.20190305-1.0`
- SPARQL バージョン: `1.1`

エンジンリリース 1.1.1.0 へのアップグレードパス

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。メジャーバージョンエンジン (1.1.0.0) より前のバージョンでは、このリリースへのアップグレードに時間がかかることに注意してください。

このリリースに自動的にアップグレードされることはありません。

このリリースへのアップグレード

> **Important**
>
> **`1.1.0.0`** より前のバージョンからこのエンジンリリースにアップグレードすると、DB クラスター内のすべてのインスタンスでオペレーティングシステムのアップグレードもトリガーされます。オペレーティングシステムのアップグレード中に発生するアクティブな書き込み要求は処理されないため、アップグレードを開始する前に、一括データロードを含め、アップグレード中のクラスターへのすべての書き込みワークロードを一時停止する必要があります。
> アップグレードの開始時に、Neptune は `preupgrade` に DB クラスターの情報に基づいて自動生成された識別子が続く名前のスナップショットを生成します。このスナップショットには課金はされません。アップグレードプロセス中に問題が発生した場合は、DB クラスターを復元するために使用できます。
> エンジンのアップグレード自体が完了すると、古いオペレーティングシステムで新しいエンジンバージョンが一時的に利用可能になりますが、5 分以内にクラスター内のすべてのインスタンスが同時にオペレーティングシステムのアップグレードを開始します。DB クラスターはこの時点で約 6 分間使用できなくなります。アップグレードの完了後に、書き込みワークロードを再開できます。
> このプロセスでは、次のイベントが生成されます。
>
> - クラスターごとのイベントメッセージ。
>   - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-`*`(autogenerated snapshot ID)`*`]`
>   - `Database cluster major version has been upgraded`
> - インスタンスごとのイベントメッセージ。

```
    • Applying off-line patches to DB instance
    • DB instance shutdown
    • Finished applying off-line patches to DB instance
    • DB instance restarted
```

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine neptune \
    --engine-version 1.1.1.0 \
    --allow-major-version-upgrade \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine neptune ^
    --engine-version 1.1.1.0 ^
    --allow-major-version-upgrade ^
    --apply-immediately
```

`--apply-immediately` の代わりに `--no-apply-immediately` と指定することができます。メジャーバージョンアップグレードを実行するには、allow-major-version-upgrade パラメータが必要です。また、エンジンバージョンを含めるようにしてください。そうしないと、エンジンが別のバージョンにアップグレードされる可能性があります。

クラスターでカスタムクラスターパラメータグループを使用する場合は、必ずこのパラメータを含めて、それを指定してください。

```
    --db-cluster-parameter-group-name (name of the custom DB cluster parameter group)
```

同様に、クラスター内のインスタンスがカスタム DB のパラメータグループを使用している場合は、必ずこのパラメータを指定して、次のようになります。

```
--db-instance-parameter-group-name (name of the custom instance parameter group)
```

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
> ```

```
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.1.1.0.R7 (2023-01-23)

2023 年 1 月 23 日現在、エンジンバージョン1.1.1.0.R7 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

Important

`1.1.0.0` より前のバージョンからこのエンジンリリースにアップグレードすると、DB クラスター内のすべてのインスタンスでオペレーティングシステムのアップグレードもトリガーされます。オペレーティングシステムのアップグレード中に発生するアクティブな書き込み要求は処理されないため、アップグレードを開始する前に、一括データロードを含め、アップグレード中のクラスターへのすべての書き込みワークロードを一時停止する必要があります。

アップグレードを正常に完了するには、すべてのアベイラビリティーゾーン (AZ) の各サブネットに、Neptune インスタンスごとに利用可能な IP アドレスが少なくとも 1 つ必要です。例えば、サブネット 1 に 1 つのライターインスタンスと 2 つのリーダーインスタンスがあり、サブネット 2 に 2 つのリーダーインスタンスがある場合、アップグレードを開始する前に、サブネット 1 には少なくとも 3 つの IP アドレスの空きがあり、サブネット 2 には少なくとも 2 つの IP アドレスが空いている必要があります。

アップグレードの開始時に、Neptune は `preupgrade` に DB クラスターの情報に基づいて自動生成された識別子が続く名前のスナップショットを生成します。このスナップショットには課金はされません。アップグレードプロセス中に問題が発生した場合は、DB クラスターを復元するために使用できます。

エンジンのアップグレード自体が完了すると、古いオペレーティングシステムで新しいエンジンバージョンが一時的に利用可能になりますが、5 分以内にクラスター内のすべてのイ

ンスタンスが同時にオペレーティングシステムのアップグレードを開始します。DB クラスターは、この時点で数分間使用できなくなります。アップグレードの完了後に、書き込みワークロードを再開できます。
このプロセスでは、次のイベントが生成されます。

- クラスターごとのイベントメッセージ。
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-(autogenerated snapshot ID)]`
  - `Database cluster major version has been upgraded`
- インスタンスごとのイベントメッセージ。
  - `Applying off-line patches to DB instance`
  - `DB instance shutdown`
  - `Finished applying off-line patches to DB instance`
  - `DB instance restarted`

このエンジンリリースの改良点

- `MERGE` と `OPTIONAL MATCH` を含む openCypher クエリのパフォーマンスが向上しました。
- リテラル値のマップのリストの `UNWIND` を含む openCypher クエリのパフォーマンスが向上しました。
- `id` の `IN` フィルターの付いた openCypher クエリのパフォーマンスが向上しました。例:

```
MATCH (n) WHERE id(n) IN ['1', '2', '3'] RETURN n
```

- `repeat`、`coalesce`、`store`、`aggregate` など、さまざまな Gremlin 演算子のパフォーマンスの向上と正確性の修正を行いました。

このエンジンリリースで修正された不具合

- HTTP キープアライブを使用するリクエストが、リクエストが失敗した後に送信された場合に誤ってクローズされる可能性がある openCypher のバグを修正しました。
- リストまたはマップのリストでパラメータタイプが正しく解釈されないことがあるという openCypher のバグを修正しました。

- Bolt と SPARQL-JSON でクエリが NULL 値の代わりに文字列 "null" を返す openCypher のバグを修正しました。
- クエリタイムアウトエラーとメモリ不足エラーに関する openCypher エラーコードとエラーメッセージを修正しました。
- DFE エンジンの by() トラバーサル時に valueMap() が最適化されない原因となっていた Gremlin のバグを修正しました。
- デッドロック検出ロジックで、エンジンが応答しなくなることがあった問題を修正しました。
- 不要な情報がログに記録されたり、特定のフィールドがログから欠落したりする監査ログのバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.1.1.0.R7 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: `3.5.2`
- サポートされている最も新しいバージョンの Gremlin: `3.5.3`
- openCypher バージョン: `Neptune-9.0.20190305-1.0`
- SPARQL バージョン: `1.1`

エンジンリリース 1.1.1.0.R7 へのアップグレードパス

エンジンバージョン `1.1.1.0` を実行している場合、クラスターは次のメンテナンス期間中に自動的にこのパッチリリースにアップグレードされます。

このリリースへのアップグレード

Important

`1.1.0.0` より前のバージョンからこのエンジンリリースにアップグレードすると、DB クラスター内のすべてのインスタンスでオペレーティングシステムのアップグレードもトリガーされます。オペレーティングシステムのアップグレード中に発生するアクティブな書き込み要求は処理されないため、アップグレードを開始する前に、一括データロードを含め、アップグレード中のクラスターへのすべての書き込みワークロードを一時停止する必要があります。

アップグレードの開始時に、Neptune は preupgrade に DB クラスターの情報に基づいて自動生成された識別子が続く名前のスナップショットを生成します。このスナップショットには課金はされません。アップグレードプロセス中に問題が発生した場合は、DB クラスターを復元するために使用できます。
エンジンのアップグレード自体が完了すると、古いオペレーティングシステムで新しいエンジンバージョンが一時的に利用可能になりますが、5 分以内にクラスター内のすべてのインスタンスが同時にオペレーティングシステムのアップグレードを開始します。DB クラスターはこの時点で約 6 分間使用できなくなります。アップグレードの完了後に、書き込みワークロードを再開できます。
このプロセスでは、次のイベントが生成されます。

- クラスターごとのイベントメッセージ。
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-`*`(autogenerated snapshot ID)`*`]`
  - `Database cluster major version has been upgraded`
- インスタンスごとのイベントメッセージ。
  - `Applying off-line patches to DB instance`
  - `DB instance shutdown`
  - `Finished applying off-line patches to DB instance`
  - `DB instance restarted`

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.1.1.0 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
```

```
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.1.1.0 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のような エラーが発生する可能性があります。

```
  We're sorry, your request to modify DB cluster (cluster identifier) has
failed.
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.1.1.0.R6 (2022-09-23)

2022 年 9 月 23 日現在、エンジンバージョン1.1.1.0.R6 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

Important

`1.1.0.0` より前のバージョンからこのエンジンリリースにアップグレードすると、DB クラスター内のすべてのインスタンスでオペレーティングシステムのアップグレードもトリガーされます。オペレーティングシステムのアップグレード中に発生するアクティブな書き込み要求は処理されないため、アップグレードを開始する前に、一括データロードを含め、アップグレード中のクラスターへのすべての書き込みワークロードを一時停止する必要があります。

アップグレードを正常に完了するには、すべてのアベイラビリティーゾーン (AZ) の各サブネットに、Neptune インスタンスごとに利用可能な IP アドレスが少なくとも 1 つ必要です。例えば、サブネット 1 に 1 つのライターインスタンスと 2 つのリーダーインスタンスがあり、サブネット 2 に 2 つのリーダーインスタンスがある場合、アップグレードを開始する前に、サブネット 1 には少なくとも 3 つの IP アドレスの空きがあり、サブネット 2 には少なくとも 2 つの IP アドレスが空いている必要があります。

アップグレードの開始時に、Neptune は `preupgrade` に DB クラスターの情報に基づいて自動生成された識別子が続く名前のスナップショットを生成します。このスナップショットには課金はされません。アップグレードプロセス中に問題が発生した場合は、DB クラスターを復元するために使用できます。

エンジンのアップグレード自体が完了すると、古いオペレーティングシステムで新しいエンジンバージョンが一時的に利用可能になりますが、5 分以内にクラスター内のすべてのインスタンスが同時にオペレーティングシステムのアップグレードを開始します。DB クラスターは、この時点で数分間使用できなくなります。アップグレードの完了後に、書き込みワークロードを再開できます。
このプロセスでは、次のイベントが生成されます。

- クラスターごとのイベントメッセージ。
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-(autogenerated snapshot ID)]`
  - `Database cluster major version has been upgraded`
- インスタンスごとのイベントメッセージ。
  - `Applying off-line patches to DB instance`
  - `DB instance shutdown`
  - `Finished applying off-line patches to DB instance`
  - `DB instance restarted`

Note

このリリースでは、IAM 認証で openCypher を使用するコードに重大な変更が加えられています。これまでは、IAM シグネチャーのホスト文字列には、`bolt://` など、次のようなプロトコルが含まれていました。

```
"Host":"bolt://(host URL):(port)"
```

エンジンリリース `1.1.1.0` 以降、このプロトコルは省略する必要があります。

```
"Host":"(host URL):(port)"
```

例については、「Bolt プロトコルの使用」を参照してください。

このエンジンリリースの改良点

- Gremlin `order-by` クエリのパフォーマンスが改善されました。`NeptuneGraphQueryStep` の末尾に `order-by` が付いている Gremlin クエリは、パフォーマンスを向上させるために、より大きなサイズのチャンクを使用するようになりました。これはクエリプランの内部 (ルート以外の) ノードの `order-by` には適用されません。
- Gremlin 更新クエリのパフォーマンスが改善されました。エッジやプロパティを追加している間は、削除されないように頂点とエッジをロックする必要があります。この変更により、トランザクション内の重複ロックがなくなり、パフォーマンスが向上します。

このエンジンリリースで修正された不具合

- ノードとエッジが重複して作成されることがあった `MERGE` 句内の openCypher のバグが修正されました。
- `OPTIONAL` 句内に (`NOT`) `EXISTS` を含む SPARQL クエリの処理において、クエリの結果が表示されない場合があるバグを修正しました。
- 一括ロードの進行中にサーバーの再起動が遅れるバグを修正しました。
- リレーションシッププロパティにフィルターを設定した openCypher 可変長パターンの双方向トラバーサルでエラーが発生するバグを修正しました。このような可変長パターンの例としては、`(n)-[r*1..2]->(m)` があります。
- キャッシュされたデータをクライアントに送り返す方法に関するバグを修正しました。このバグが原因で、場合によっては予想外に長い待ち時間が発生することがありました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.1.1.0.R6 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: `3.5.2`
- サポートされている最も新しいバージョンの Gremlin: `3.5.4`
- openCypher バージョン: `Neptune-9.0.20190305-1.0`
- SPARQL バージョン: `1.1`

エンジンリリース 1.1.1.0.R6 へのアップグレードパス

エンジンバージョン `1.1.1.0` を実行している場合、クラスターは次のメンテナンス期間中に自動的にこのパッチリリースにアップグレードされます。

このリリースへのアップグレード

Important

`1.1.0.0` より前のバージョンからこのエンジンリリースにアップグレードすると、DB クラスター内のすべてのインスタンスでオペレーティングシステムのアップグレードもトリガーされます。オペレーティングシステムのアップグレード中に発生するアクティブな書き込み要求は処理されないため、アップグレードを開始する前に、一括データロードを含め、アップグレード中のクラスターへのすべての書き込みワークロードを一時停止する必要があります。

アップグレードの開始時に、Neptune は `preupgrade` に DB クラスターの情報に基づいて自動生成された識別子が続く名前のスナップショットを生成します。このスナップショットには課金はされません。アップグレードプロセス中に問題が発生した場合は、DB クラスターを復元するために使用できます。

エンジンのアップグレード自体が完了すると、古いオペレーティングシステムで新しいエンジンバージョンが一時的に利用可能になりますが、5 分以内にクラスター内のすべてのインスタンスが同時にオペレーティングシステムのアップグレードを開始します。DB クラスターはこの時点で約 6 分間使用できなくなります。アップグレードの完了後に、書き込みワークロードを再開できます。

このプロセスでは、次のイベントが生成されます。

- クラスターごとのイベントメッセージ。
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-`*`(autogenerated snapshot ID)`*`]`
  - `Database cluster major version has been upgraded`
- インスタンスごとのイベントメッセージ。
  - `Applying off-line patches to DB instance`
  - `DB instance shutdown`
  - `Finished applying off-line patches to DB instance`
  - `DB instance restarted`

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.1.1.0 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.1.1.0 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the instance before
>  proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.1.1.0.R5 (2022-07-21)

2022 年 7 月 21 日現在、エンジンバージョン1.1.1.0.R5 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

> **Important**
>
> **`1.1.0.0`** より前のバージョンからこのエンジンリリースにアップグレードすると、DB クラスター内のすべてのインスタンスでオペレーティングシステムのアップグレードもトリガーされます。オペレーティングシステムのアップグレード中に発生するアクティブな書き込み

要求は処理されないため、アップグレードを開始する前に、一括データロードを含め、アップグレード中のクラスターへのすべての書き込みワークロードを一時停止する必要があります。
アップグレードを正常に完了するには、すべてのアベイラビリティーゾーン (AZ) の各サブネットに、Neptune インスタンスごとに利用可能な IP アドレスが少なくとも 1 つ必要です。例えば、サブネット 1 に 1 つのライターインスタンスと 2 つのリーダーインスタンスがあり、サブネット 2 に 2 つのリーダーインスタンスがある場合、アップグレードを開始する前に、サブネット 1 には少なくとも 3 つの IP アドレスの空きがあり、サブネット 2 には少なくとも 2 つの IP アドレスが空いている必要があります。
アップグレードの開始時に、Neptune は `preupgrade` に DB クラスターの情報に基づいて自動生成された識別子が続く名前のスナップショットを生成します。このスナップショットには課金はされません。アップグレードプロセス中に問題が発生した場合は、DB クラスターを復元するために使用できます。
エンジンのアップグレード自体が完了すると、古いオペレーティングシステムで新しいエンジンバージョンが一時的に利用可能になりますが、5 分以内にクラスター内のすべてのインスタンスが同時にオペレーティングシステムのアップグレードを開始します。DB クラスターは、この時点で数分間使用できなくなります。アップグレードの完了後に、書き込みワークロードを再開できます。
このプロセスでは、次のイベントが生成されます。

- クラスターごとのイベントメッセージ。
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-`*`(autogenerated snapshot ID)`*`]`
  - `Database cluster major version has been upgraded`
- インスタンスごとのイベントメッセージ。
  - `Applying off-line patches to DB instance`
  - `DB instance shutdown`
  - `Finished applying off-line patches to DB instance`
  - `DB instance restarted`

> **Note**
>
> このリリースでは、IAM 認証で openCypher を使用するコードに重大な変更が加えられています。これまでは、IAM シグネチャーのホスト文字列には、`bolt://` など、次のようなプロトコルが含まれていました。
>
> ```
> "Host":"bolt://(host URL):(port)"
> ```
>
> エンジンリリース `1.1.1.0` 以降、このプロトコルは省略する必要があります。
>
> ```
> "Host":"(host URL):(port)"
> ```
>
> 例については、「Bolt プロトコルの使用」を参照してください。

このエンジンリリースの改良点

- デッドロック検出をサポートするように改良を加えました。

このエンジンリリースで修正された不具合

- 特定の条件下で DB クラスターをクリーンシャットダウンできないバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.1.1.0.R5 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: `3.5.2`
- サポートされている最も新しいバージョンの Gremlin: `3.5.4`
- openCypher バージョン: `Neptune-9.0.20190305-1.0`
- SPARQL バージョン: `1.1`

エンジンリリース 1.1.1.0.R5 へのアップグレードパス

エンジンバージョン `1.1.1.0` を実行している場合、クラスターは次のメンテナンス期間中に自動的にこのパッチリリースにアップグレードされます。

このリリースへのアップグレード

Important

`1.1.0.0` より前のバージョンからこのエンジンリリースにアップグレードすると、DB クラスター内のすべてのインスタンスでオペレーティングシステムのアップグレードもトリガーされます。オペレーティングシステムのアップグレード中に発生するアクティブな書き込み要求は処理されないため、アップグレードを開始する前に、一括データロードを含め、アップグレード中のクラスターへのすべての書き込みワークロードを一時停止する必要があります。

アップグレードの開始時に、Neptune は `preupgrade` に DB クラスターの情報に基づいて自動生成された識別子が続く名前のスナップショットを生成します。このスナップショットには課金はされません。アップグレードプロセス中に問題が発生した場合は、DB クラスターを復元するために使用できます。

エンジンのアップグレード自体が完了すると、古いオペレーティングシステムで新しいエンジンバージョンが一時的に利用可能になりますが、5 分以内にクラスター内のすべてのインスタンスが同時にオペレーティングシステムのアップグレードを開始します。DB クラスターはこの時点で約 6 分間使用できなくなります。アップグレードの完了後に、書き込みワークロードを再開できます。

このプロセスでは、次のイベントが生成されます。

- クラスターごとのイベントメッセージ。
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-`*`(autogenerated snapshot ID)`*`]`
  - `Database cluster major version has been upgraded`
- インスタンスごとのイベントメッセージ。
  - `Applying off-line patches to DB instance`
  - `DB instance shutdown`
  - `Finished applying off-line patches to DB instance`
  - `DB instance restarted`

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.1.1.0 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.1.1.0 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.1.1.0.R4 (2022-06-23)

2022 年 6 月 23 日現在、エンジンバージョン1.1.1.0.R4 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

> **Important**
>
> **`1.1.0.0`** より前のバージョンからこのエンジンリリースにアップグレードすると、DB クラスター内のすべてのインスタンスでオペレーティングシステムのアップグレードもトリガーされます。オペレーティングシステムのアップグレード中に発生するアクティブな書き込み

要求は処理されないため、アップグレードを開始する前に、一括データロードを含め、アップグレード中のクラスターへのすべての書き込みワークロードを一時停止する必要があります。
アップグレードを正常に完了するには、すべてのアベイラビリティーゾーン (AZ) の各サブネットに、Neptune インスタンスごとに利用可能な IP アドレスが少なくとも 1 つ必要です。例えば、サブネット 1 に 1 つのライターインスタンスと 2 つのリーダーインスタンスがあり、サブネット 2 に 2 つのリーダーインスタンスがある場合、アップグレードを開始する前に、サブネット 1 には少なくとも 3 つの IP アドレスの空きがあり、サブネット 2 には少なくとも 2 つの IP アドレスが空いている必要があります。
アップグレードの開始時に、Neptune は `preupgrade` に DB クラスターの情報に基づいて自動生成された識別子が続く名前のスナップショットを生成します。このスナップショットには課金はされません。アップグレードプロセス中に問題が発生した場合は、DB クラスターを復元するために使用できます。
エンジンのアップグレード自体が完了すると、古いオペレーティングシステムで新しいエンジンバージョンが一時的に利用可能になりますが、5 分以内にクラスター内のすべてのインスタンスが同時にオペレーティングシステムのアップグレードを開始します。DB クラスターは、この時点で数分間使用できなくなります。アップグレードの完了後に、書き込みワークロードを再開できます。
このプロセスでは、次のイベントが生成されます。

- クラスターごとのイベントメッセージ。
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-`*`(autogenerated snapshot ID)`*`]`
  - `Database cluster major version has been upgraded`
- インスタンスごとのイベントメッセージ。
  - `Applying off-line patches to DB instance`
  - `DB instance shutdown`
  - `Finished applying off-line patches to DB instance`
  - `DB instance restarted`

> **Note**
>
> このリリースでは、IAM 認証で openCypher を使用するコードに重大な変更が加えられています。これまでは、IAM シグネチャーのホスト文字列には、`bolt://` など、次のようなプロトコルが含まれていました。
>
> ```
> "Host":"bolt://(host URL):(port)"
> ```
>
> エンジンリリース `1.1.1.0` 以降、このプロトコルは省略する必要があります。
>
> ```
> "Host":"(host URL):(port)"
> ```
>
> 例については、「Bolt プロトコルの使用」を参照してください。

このエンジンリリースの改良点

- インスタンスタイプのインスタンス設定が更新されました。`x2g`
- 頂点ドロップのパフォーマンスが向上しました。

このエンジンリリースで修正された不具合

- 特定の種類の ASK 結合について、複数回呼び出されたり、複数のリーダーで呼び出されたクエリの順序が安定していなかったという Gremlin のバグを修正しました。
- また、Gremlin の特定の種類の ASK 結合でパフォーマンスの低下を引き起こしていた以前のリリースの変更の範囲を絞り込みました。
- 子トラバーサル内でエッジ入力と頂点へのトラバーサルがあった場合に発生していた `union()` ステップの Gremlin のバグを修正しました。
- 一部のステップが実際に最適化されていても最適化されていないと報告される Gremlin プロファイルのバグを修正しました。
- `UNION` 句にネストされた `FILTER` 式内で使用される変数に無効なスコープ情報が代入されるという SPARQL のバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.1.1.0.R4 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: `3.5.2`
- サポートされている最も新しいバージョンの Gremlin: `3.5.4`
- openCypher バージョン: `Neptune-9.0.20190305-1.0`
- SPARQL バージョン: `1.1`

エンジンリリース 1.1.1.0.R4 へのアップグレードパス

エンジンバージョン `1.1.1.0` を実行している場合、クラスターは次のメンテナンス期間中に自動的にこのパッチリリースにアップグレードされます。

このリリースへのアップグレード

> **Important**
>
> **`1.1.0.0`** より前のバージョンからこのエンジンリリースにアップグレードすると、DB クラスター内のすべてのインスタンスでオペレーティングシステムのアップグレードもトリガーされます。オペレーティングシステムのアップグレード中に発生するアクティブな書き込み要求は処理されないため、アップグレードを開始する前に、一括データロードを含め、アップグレード中のクラスターへのすべての書き込みワークロードを一時停止する必要があります。
>
> アップグレードの開始時に、Neptune は `preupgrade` に DB クラスターの情報に基づいて自動生成された識別子が続く名前のスナップショットを生成します。このスナップショットには課金はされません。アップグレードプロセス中に問題が発生した場合は、DB クラスターを復元するために使用できます。
>
> エンジンのアップグレード自体が完了すると、古いオペレーティングシステムで新しいエンジンバージョンが一時的に利用可能になりますが、5 分以内にクラスター内のすべてのインスタンスが同時にオペレーティングシステムのアップグレードを開始します。DB クラスターはこの時点で約 6 分間使用できなくなります。アップグレードの完了後に、書き込みワークロードを再開できます。
>
> このプロセスでは、次のイベントが生成されます。
>
> - クラスターごとのイベントメッセージ。

    - Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-*(autogenerated snapshot ID)*]
    - Database cluster major version has been upgraded
  - インスタンスごとのイベントメッセージ。
    - Applying off-line patches to DB instance
    - DB instance shutdown
    - Finished applying off-line patches to DB instance
    - DB instance restarted

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.1.1.0 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.1.1.0 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.1.1.0.R3 (2022-06-07)

2022 年 6 月 7 日現在、エンジンバージョン1.1.1.0.R3 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

Important

`1.1.0.0` より前のバージョンからこのエンジンリリースにアップグレードすると、DB クラスター内のすべてのインスタンスでオペレーティングシステムのアップグレードもトリガーされます。オペレーティングシステムのアップグレード中に発生するアクティブな書き込み要求は処理されないため、アップグレードを開始する前に、一括データロードを含め、アップグレード中のクラスターへのすべての書き込みワークロードを一時停止する必要があります。
アップグレードの開始時に、Neptune は `preupgrade` に DB クラスターの情報に基づいて自動生成された識別子が続く名前のスナップショットを生成します。このスナップショットには課金はされません。アップグレードプロセス中に問題が発生した場合は、DB クラスターを復元するために使用できます。
エンジンのアップグレード自体が完了すると、古いオペレーティングシステムで新しいエンジンバージョンが一時的に利用可能になりますが、5 分以内にクラスター内のすべてのインスタンスが同時にオペレーティングシステムのアップグレードを開始します。DB クラスターは、この時点で数分間使用できなくなります。アップグレードの完了後に、書き込みワークロードを再開できます。
このプロセスでは、次のイベントが生成されます。

- クラスターごとのイベントメッセージ。
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-(autogenerated snapshot ID)]`
  - `Database cluster major version has been upgraded`
- インスタンスごとのイベントメッセージ。
  - `Applying off-line patches to DB instance`
  - `DB instance shutdown`
  - `Finished applying off-line patches to DB instance`

- DB instance restarted

> **Note**
>
> このリリースでは、IAM 認証で openCypher を使用するコードに重大な変更が加えられています。これまでは、IAM シグネチャーのホスト文字列には、bolt:// など、次のようなプロトコルが含まれていました。
>
> ```
> "Host":"bolt://(host URL):(port)"
> ```
>
> エンジンリリース 1.1.1.0 以降、このプロトコルは省略する必要があります。
>
> ```
> "Host":"(host URL):(port)"
> ```
>
> 例については、「Bolt プロトコルの使用」を参照してください。

このエンジンリリースの改良点

- メモリを大量に消費するワークロード向けに最適化された Graviton2 x2g 搭載インスタンスタイプのサポートが追加されました。これらは当初、次の 4 つの AWS リージョン でのみ利用可能です。
  - 米国東部 (バージニア北部) (us-east-1)
  - 米国東部 (オハイオ) (us-east-2)
  - 米国西部 (オレゴン) (us-west-2)
  - 欧州 (アイルランド) (eu-west-1)

  詳細については、Neptune 料金表ページを参照してください。
- 複数のエッジまたは頂点のトラバーサル、プロパティ検索、またはラベル検索が必要な Gremlin ステップのパフォーマンスが向上しました。

このエンジンリリースで修正された不具合

- 子トラバーサル内の otherV() ステップ処理での Gremlin のバグを修正しました。

- 子としてフィルターステップしか存在しない union のクエリの Gremlin のバグを修正しました。例:

```
g.V().union(has("name"), out("knows")).out()
```

- UNION句にネストされた FILTER 式内で使用される変数に無効なスコープ情報が代入されるという SPARQL のバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.1.1.0.R3 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: `3.5.2`
- サポートされている最も新しいバージョンの Gremlin: `3.5.4`
- openCypher バージョン: `Neptune-9.0.20190305-1.0`
- SPARQL バージョン: `1.1`

エンジンリリース 1.1.1.0.R3 へのアップグレードパス

エンジンバージョン `1.1.1.0` を実行している場合、クラスターは次のメンテナンス期間中に自動的にこのパッチリリースにアップグレードされます。

このリリースへのアップグレード

Important

`1.1.0.0` より前のバージョンからこのエンジンリリースにアップグレードすると、DB クラスター内のすべてのインスタンスでオペレーティングシステムのアップグレードもトリガーされます。オペレーティングシステムのアップグレード中に発生するアクティブな書き込み要求は処理されないため、アップグレードを開始する前に、一括データロードを含め、アップグレード中のクラスターへのすべての書き込みワークロードを一時停止する必要があります。

アップグレードの開始時に、Neptune は `preupgrade` に DB クラスターの情報に基づいて自動生成された識別子が続く名前のスナップショットを生成します。このスナップショットには課金はされません。アップグレードプロセス中に問題が発生した場合は、DB クラスターを復元するために使用できます。

エンジンのアップグレード自体が完了すると、古いオペレーティングシステムで新しいエンジンバージョンが一時的に利用可能になりますが、5 分以内にクラスター内のすべてのインスタンスが同時にオペレーティングシステムのアップグレードを開始します。DB クラスターはこの時点で約 6 分間使用できなくなります。アップグレードの完了後に、書き込みワークロードを再開できます。
このプロセスでは、次のイベントが生成されます。

- クラスターごとのイベントメッセージ。
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-(autogenerated snapshot ID)]`
  - `Database cluster major version has been upgraded`
- インスタンスごとのイベントメッセージ。
  - `Applying off-line patches to DB instance`
  - `DB instance shutdown`
  - `Finished applying off-line patches to DB instance`
  - `DB instance restarted`

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.1.1.0 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.1.1.0 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
> ```

```
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune メンテナンスリリース、バージョン 1.1.1.0.R2 (2022-05-16)

2022 年 5 月 16 日現在、エンジンバージョン 1.1.1.0.R2 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

Important

`1.1.0.0` より前のバージョンからこのエンジンリリースにアップグレードすると、DB クラスター内のすべてのインスタンスでオペレーティングシステムのアップグレードもトリガーされます。オペレーティングシステムのアップグレード中に発生するアクティブな書き込み要求は処理されないため、アップグレードを開始する前に、一括データロードを含め、アップグレード中のクラスターへのすべての書き込みワークロードを一時停止する必要があります。

アップグレードを正常に完了するには、すべてのアベイラビリティーゾーン (AZ) の各サブネットに、Neptune インスタンスごとに利用可能な IP アドレスが少なくとも 1 つ必要です。例えば、サブネット 1 に 1 つのライターインスタンスと 2 つのリーダーインスタンスがあり、サブネット 2 に 2 つのリーダーインスタンスがある場合、アップグレードを開始する前に、サブネット 1 には少なくとも 3 つの IP アドレスの空きがあり、サブネット 2 には少なくとも 2 つの IP アドレスが空いている必要があります。

アップグレードの開始時に、Neptune は `preupgrade` に DB クラスターの情報に基づいて自動生成された識別子が続く名前のスナップショットを生成します。このスナップショットには課金はされません。アップグレードプロセス中に問題が発生した場合は、DB クラスターを復元するために使用できます。

エンジンのアップグレード自体が完了すると、古いオペレーティングシステムで新しいエンジンバージョンが一時的に利用可能になりますが、5 分以内にクラスター内のすべてのイ

ンスタンスが同時にオペレーティングシステムのアップグレードを開始します。DB クラスターは、この時点で数分間使用できなくなります。アップグレードの完了後に、書き込みワークロードを再開できます。
このプロセスでは、次のイベントが生成されます。

- クラスターごとのイベントメッセージ。
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-(autogenerated snapshot ID)]`
  - `Database cluster major version has been upgraded`
- インスタンスごとのイベントメッセージ。
  - `Applying off-line patches to DB instance`
  - `DB instance shutdown`
  - `Finished applying off-line patches to DB instance`
  - `DB instance restarted`

Note

このリリースでは、IAM 認証で openCypher を使用するコードに重大な変更が加えられています。これまでは、IAM シグネチャーのホスト文字列には、`bolt://` など、次のようなプロトコルが含まれていました。

```
"Host":"bolt://(host URL):(port)"
```

エンジンリリース `1.1.1.0` 以降、このプロトコルは省略する必要があります。

```
"Host":"(host URL):(port)"
```

例については、「Bolt プロトコルの使用」を参照してください。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.1.1.0.R2 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- サポートされている最も古いバージョンの Gremlin: `3.5.2`
- サポートされている最も新しいバージョンの Gremlin: `3.5.4`
- openCypher バージョン: `Neptune-9.0.20190305-1.0`
- SPARQL バージョン: `1.1`

エンジンリリース 1.1.1.0.R2 へのアップグレードパス

エンジンバージョン `1.1.1.0` を実行している場合、クラスターは次のメンテナンスウィンドウで自動的にこのメンテナンスパッチリリースにアップグレードされます。

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

このリリースへのアップグレード

Important

`1.1.0.0` より前のバージョンからこのエンジンリリースにアップグレードすると、DB クラスター内のすべてのインスタンスでオペレーティングシステムのアップグレードもトリガーされます。オペレーティングシステムのアップグレード中に発生するアクティブな書き込み要求は処理されないため、アップグレードを開始する前に、一括データロードを含め、アップグレード中のクラスターへのすべての書き込みワークロードを一時停止する必要があります。
アップグレードの開始時に、Neptune は `preupgrade` に DB クラスターの情報に基づいて自動生成された識別子が続く名前のスナップショットを生成します。このスナップショットには課金はされません。アップグレードプロセス中に問題が発生した場合は、DB クラスターを復元するために使用できます。
エンジンのアップグレード自体が完了すると、古いオペレーティングシステムで新しいエンジンバージョンが一時的に利用可能になりますが、5 分以内にクラスター内のすべてのインスタンスが同時にオペレーティングシステムのアップグレードを開始します。DB クラスターはこの時点で約 6 分間使用できなくなります。アップグレードの完了後に、書き込みワークロードを再開できます。
このプロセスでは、次のイベントが生成されます。

- クラスターごとのイベントメッセージ。
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-`*`(autogenerated snapshot ID)`*`]`
  - `Database cluster major version has been upgraded`

- インスタンスごとのイベントメッセージ。
  - Applying off-line patches to DB instance
  - DB instance shutdown
  - Finished applying off-line patches to DB instance
  - DB instance restarted

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.1.1.0 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.1.1.0 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 〜 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.1.0.0 (2021-11-19)

2021 年 11 月 19 日現在、エンジンバージョン 1.1.0.0 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

Important

1.1.0.0 より前のバージョンからこのエンジンリリースにアップグレードすると、DB クラスター内のすべてのインスタンスでオペレーティングシステムのアップグレードもトリガーされます。オペレーティングシステムのアップグレード中に発生するアクティブな書き込み要求は処理されないため、アップグレードを開始する前に、一括データロードを含め、アップグレード中のクラスターへのすべての書き込みワークロードを一時停止する必要があります。
アップグレードを正常に完了するには、すべてのアベイラビリティーゾーン (AZ) の各サブネットに、Neptune インスタンスごとに利用可能な IP アドレスが少なくとも 1 つ必要です。例えば、サブネット 1 に 1 つのライターインスタンスと 2 つのリーダーインスタンスがあり、サブネット 2 に 2 つのリーダーインスタンスがある場合、アップグレードを開始する前に、サブネット 1 には少なくとも 3 つの IP アドレスの空きがあり、サブネット 2 には少なくとも 2 つの IP アドレスが空いている必要があります。
アップグレードの開始時に、Neptune は `preupgrade` に DB クラスターの情報に基づいて自動生成された識別子が続く名前のスナップショットを生成します。このスナップショットには課金はされません。アップグレードプロセス中に問題が発生した場合は、DB クラスターを復元するために使用できます。
エンジンのアップグレード自体が完了すると、古いオペレーティングシステムで新しいエンジンバージョンが一時的に利用可能になりますが、5 分以内にクラスター内のすべてのインスタンスが同時にオペレーティングシステムのアップグレードを開始します。DB クラスターは、この時点で数分間使用できなくなります。アップグレードの完了後に、書き込みワークロードを再開できます。
このプロセスでは、次のイベントが生成されます。

- クラスターごとのイベントメッセージ。
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-`*`(autogenerated snapshot ID)`*`]`
  - `Database cluster major version has been upgraded`
- インスタンスごとのイベントメッセージ。
  - `Applying off-line patches to DB instance`

- `DB instance shutdown`
- `Finished applying off-line patches to DB instance`
- `DB instance restarted`

> **Note**
>
> このエンジンリリース以降、Neptune はR4 インスタンスタイプをサポートしなくなりました。DB クラスター内の R4 インスタンスを使用している場合は、このリリースにアップグレードする前に、手動で別のインスタンスタイプに置き換える必要があります。ライターインスタンスが R4 の場合は、次の手順に従って移動してください。

このリリースの後続のパッチリリース

- メンテナンスリリース: 1.1.0.0.R2 (2022-05-16)
- メンテナンスリリース: 1.1.0.0.R3 (2022-12-23)

このエンジンリリースの新機能

- AWS Graviton2 プロセッサにより稼働する汎用 T4g およびメモリ最適化 R6g データベースインスタンスを導入しました。Graviton2 ベースのインスタンスは、さまざまなワークロードに対して、同等の現行世代の x86 ベースのインスタンスよりも大幅に優れた価格/パフォーマンスを提供します。アプリケーションは、これらの新しいインスタンスタイプでは通常どおりに動作し、アップグレードするときにポートアプリケーションコードを移植する必要はありません。

  料金および使用可能なリージョンに関する詳細については、Amazon Neptune 料金表ページを参照してください。
- Neptune ML にカスタムモデルが導入されました。
- Neptune ML に SPARQL 推論クエリのサポートが追加されました。
- プロパティグラフデータ用に新しいストリームエンドポイントを追加しました。次のようになります。

```
https://Neptune-DNS:8182/propertygraph/stream
```

このエンドポイントの出力形式、つまり、PG_JSON は、古い gremlin/stream によって出力される GREMLIN_JSON フォーマットとまったく同じ動作をします。

新しい propertygraph/stream エンドポイントは、Neptune ストリームのサポートを openCypher に拡張し、gremlin/stream を、関連付けられたエンドポイント GREMLIN_JSON 出力形式に取り替えます。

このエンジンリリースの改良点

- Neptune ストリームを改善しました。
  - 変更ログストリーム内の各レコードのタイムスタンプを提供するために、commitTimestamp フィールドを records オブジェクト Neptune ストリーム変更ログ応答形式に追加しました。
  - ストリームから最後に有効な eventId を取得できるように、LATEST 値を iteratorType パラメータに追加しました。「Streams API の呼び出し」を参照してください。
- Gremlin ノード分類および回帰クエリでの推論信頼スコア取得のサポートが追加されました。
- openCypher の OPTIONAL MATCH 句用のサポートを追加しました。
- openCypher の MERGE 句用のサポートを追加しました。
- openCypher の WITH 句の ORDER BY 使用のサポートを追加しました。
- openCypher でのパターン理解のサポートが追加され、存在チェックを超えるパターン表現のサポートが拡張されました。
- openCypher で DELETE および DELETE DETACH のサポートを拡張しました。これにより他の更新句と併用できるようになりました。
- openCypherで RETURN とともに使用される CREATE および UPDATE のサポートを拡張しました。
- Gremlin limit、range および skip ステップの DFE エンジンでのサポートを追加しました。
- explain と profile のいずれもリクエストされていない場合の DFE エンジンでのクエリ実行が改善されました。
- value 表現に対するDFE エンジンでのクエリ実行を改善しました。
- 同時変更例外を回避し、次のようなクエリパターンの連鎖を可能にするために、多数の連鎖 Gremlin 条件付き挿入パターンを改善しました。
  - ID による条件付き頂点の挿入 (例)

```
g.V(ID).fold().coalesce(unfold(), g.addV("L1").property(id,ID))
```

- 次のような複数のラベルを使用した条件付き頂点の挿入

```
g.V(ID).fold().coalesce(unfold(), g.addV("L1::L2").property(id,ID))
```

- ID による条件付き エッジの挿入 (例)

```
g.E(ID).fold().coalesce(unfold(), V(from).addE(label).to(V(to)).property(id, ID))
```

- 次のような複数のラベルを使用した条件付き エッジの挿入

```
g.E(ID).fold().coalesce(unfold(),
 g.addE(label).from(V(from)).to(V(to)).property(id, ID))
```

- クエリが続く条件付き挿入 (例)

```
g.V(ID).fold().coalesce(unfold(),
 g.addV("L1").property(id,ID)).project("myvalues").by(valueMap())
```

- プロパティを追加した条件付き挿入 (例)

```
g.V(ID).fold().coalesce(unfold(),
 g.addV("L1").property(id,ID).property("name","pumba"))
```

このエンジンリリースで修正された不具合

- サポートできなかった `T3.medium` インスタンスタイプの統計機能を無効にしました。
- 非定数値を取った `IN` 関数を持つ `explain` にある SPARQL のバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.1.0.0 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.11`
- SPARQL バージョン: `1.1`

エンジンリリース 1.1.0.0 へのアップグレードパス

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

このリリースに自動的にアップグレードされることはありません。

このリリースへのアップグレード

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.1.0.0 \
    --allow-major-version-upgrade \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.1.0.0 ^
    --allow-major-version-upgrade ^
    --apply-immediately
```

`--apply-immediately` の代わりに `--no-apply-immediately` と指定することができます。メジャーバージョンアップグレードを実行するには、allow-major-version-upgrade パラメータが必要です。また、エンジンバージョンを含めるようにしてください。そうしないと、エンジンが別のバージョンにアップグレードされる可能性があります。

クラスターでカスタムクラスターパラメータグループを使用する場合は、必ずこのパラメータを含めて、それを指定してください。

```
    --db-cluster-parameter-group-name (name of the custom DB cluster parameter group)
```

同様に、クラスター内のインスタンスがカスタム DB のパラメータグループを使用している場合は、必ずこのパラメータを指定して、次のようになります。

```
--db-instance-parameter-group-name (name of the custom instance parameter group)
```

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
> ```

```
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune メンテナンスリリース、バージョン 1.1.0.0.R3 (2022-12-23)

2022 年 12 月 23 日現在、エンジンバージョン 1.1.0.0.R3 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

Important

`1.1.0.0` より前のバージョンからこのエンジンリリースにアップグレードすると、DB クラスター内のすべてのインスタンスでオペレーティングシステムのアップグレードもトリガーされます。オペレーティングシステムのアップグレード中に発生するアクティブな書き込み要求は処理されないため、アップグレードを開始する前に、一括データロードを含め、アップグレード中のクラスターへのすべての書き込みワークロードを一時停止する必要があります。

アップグレードを正常に完了するには、すべてのアベイラビリティーゾーン (AZ) の各サブネットに、Neptune インスタンスごとに利用可能な IP アドレスが少なくとも 1 つ必要です。例えば、サブネット 1 に 1 つのライターインスタンスと 2 つのリーダーインスタンスがあり、サブネット 2 に 2 つのリーダーインスタンスがある場合、アップグレードを開始する前に、サブネット 1 には少なくとも 3 つの IP アドレスの空きがあり、サブネット 2 には少なくとも 2 つの IP アドレスが空いている必要があります。

アップグレードの開始時に、Neptune は `preupgrade` に DB クラスターの情報に基づいて自動生成された識別子が続く名前のスナップショットを生成します。このスナップショットには課金はされません。アップグレードプロセス中に問題が発生した場合は、DB クラスターを復元するために使用できます。

エンジンのアップグレード自体が完了すると、古いオペレーティングシステムで新しいエンジンバージョンが一時的に利用可能になりますが、5 分以内にクラスター内のすべてのイ

ンスタンスが同時にオペレーティングシステムのアップグレードを開始します。DB クラスターは、この時点で数分間使用できなくなります。アップグレードの完了後に、書き込みワークロードを再開できます。
このプロセスでは、次のイベントが生成されます。

- クラスターごとのイベントメッセージ。
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-(autogenerated snapshot ID)]`
  - `Database cluster major version has been upgraded`
- インスタンスごとのイベントメッセージ。
  - `Applying off-line patches to DB instance`
  - `DB instance shutdown`
  - `Finished applying off-line patches to DB instance`
  - `DB instance restarted`

このエンジンリリースの改良点

- `repeat`、`coalesce`、`store`、`aggregate` など、さまざまな Gremlin 演算子のパフォーマンスの向上と正確性の修正を行いました。

このエンジンリリースで修正された不具合

- CPU スパイクの問題を修正しました。
- Bolt と SPARQL-JSON でクエリが NULL 値ではなく文字列 `"null"` を返す openCypher のバグを修正しました。
- 不要な情報がログに記録されたり、特定のフィールドがログから欠落したりする監査ログのバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.1.0.0.R3 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.11`

- openCypher バージョン: `Neptune-9.0.20190305-1.0`
- SPARQL バージョン: `1.1`

エンジンリリース 1.1.0.0.R3 へのアップグレードパス

エンジンバージョン `1.1.0.0` を実行している場合、クラスターは次のメンテナンス期間中に自動的にこのメンテナンスパッチリリースにアップグレードされます。

> **Important**
>
> **`1.1.0.0`** より前のバージョンからこのエンジンリリースにアップグレードすると、DB クラスター内のすべてのインスタンスでオペレーティングシステムのアップグレードもトリガーされます。オペレーティングシステムのアップグレード中に発生するアクティブな書き込み要求は処理されないため、アップグレードを開始する前に、一括データロードを含め、アップグレード中のクラスターへのすべての書き込みワークロードを一時停止する必要があります。
>
> アップグレードの開始時に、Neptune は `preupgrade` に DB クラスターの情報に基づいて自動生成された識別子が続く名前のスナップショットを生成します。このスナップショットには課金はされません。アップグレードプロセス中に問題が発生した場合は、DB クラスターを復元するために使用できます。
>
> エンジンのアップグレード自体が完了すると、古いオペレーティングシステムで新しいエンジンバージョンが一時的に利用可能になりますが、5 分以内にクラスター内のすべてのインスタンスが同時にオペレーティングシステムのアップグレードを開始します。DB クラスターはこの時点で約 6 分間使用できなくなります。アップグレードの完了後に、書き込みワークロードを再開できます。
>
> このプロセスでは、次のイベントが生成されます。
>
> - クラスターごとのイベントメッセージ。
>   - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-`*`(autogenerated snapshot ID)`*`]`
>   - `Database cluster major version has been upgraded`
> - インスタンスごとのイベントメッセージ。
>   - `Applying off-line patches to DB instance`
>   - `DB instance shutdown`
>   - `Finished applying off-line patches to DB instance`

- DB instance restarted

> **Note**
>
> このエンジンリリース以降、Neptune はR4 インスタンスタイプをサポートしなくなりました。DB クラスター内の R4 インスタンスを使用している場合は、このリリースにアップグレードする前に、手動で別のインスタンスタイプに置き換える必要があります。ライターインスタンスが R4 の場合は、次の手順に従って移動してください。

このリリースへのアップグレード

Amazon Neptune 1.1.0.0.R3 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.1.0.0 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.1.0.0 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のような エラーが発生する可能性があります。

```
  We're sorry, your request to modify DB cluster (cluster identifier) has
failed.
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune メンテナンスリリース、バージョン 1.1.1.0.R2 (2022-05-16)

2022 年 5 月 16 日現在、エンジンバージョン 1.1.0.0.R2 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

Important

`1.1.0.0` より前のバージョンからこのエンジンリリースにアップグレードすると、DB クラスター内のすべてのインスタンスでオペレーティングシステムのアップグレードもトリガーされます。オペレーティングシステムのアップグレード中に発生するアクティブな書き込み要求は処理されないため、アップグレードを開始する前に、一括データロードを含め、アップグレード中のクラスターへのすべての書き込みワークロードを一時停止する必要があります。
アップグレードの開始時に、Neptune は `preupgrade` に DB クラスターの情報に基づいて自動生成された識別子が続く名前のスナップショットを生成します。このスナップショットには課金はされません。アップグレードプロセス中に問題が発生した場合は、DB クラスターを復元するために使用できます。
エンジンのアップグレード自体が完了すると、古いオペレーティングシステムで新しいエンジンバージョンが一時的に利用可能になりますが、5 分以内にクラスター内のすべてのインスタンスが同時にオペレーティングシステムのアップグレードを開始します。DB クラスターは、この時点で数分間使用できなくなります。アップグレードの完了後に、書き込みワークロードを再開できます。
このプロセスでは、次のイベントが生成されます。

- クラスターごとのイベントメッセージ。
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-`*`(autogenerated snapshot ID)`*`]`
  - `Database cluster major version has been upgraded`

- インスタンスごとのイベントメッセージ。
  - `Applying off-line patches to DB instance`
  - `DB instance shutdown`
  - `Finished applying off-line patches to DB instance`
  - `DB instance restarted`

このエンジンリリースで修正された不具合

- ステータスエンドポイントなど、クエリ以外のエンドポイントの内部認証情報キャッシュが適切にクリアされないバグを修正しました。
- エンジンのアップグレード後にレプリケーションの遅延が長くなるバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.1.0.0.R2 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.11`
- openCypher バージョン: `Neptune-9.0.20190305-1.0`
- SPARQL バージョン: `1.1`

エンジンリリース 1.1.0.0.R2 へのアップグレードパス

エンジンバージョン `1.1.0.0` を実行している場合、クラスターは次のメンテナンス期間中に自動的にこのメンテナンスパッチリリースにアップグレードされます。

Important

`1.1.0.0` より前のバージョンからこのエンジンリリースにアップグレードすると、DB クラスター内のすべてのインスタンスでオペレーティングシステムのアップグレードもトリガーされます。オペレーティングシステムのアップグレード中に発生するアクティブな書き込み要求は処理されないため、アップグレードを開始する前に、一括データロードを含め、アップグレード中のクラスターへのすべての書き込みワークロードを一時停止する必要があります。

アップグレードの開始時に、Neptune は preupgrade に DB クラスターの情報に基づいて自動生成された識別子が続く名前のスナップショットを生成します。このスナップショットには課金はされません。アップグレードプロセス中に問題が発生した場合は、DB クラスターを復元するために使用できます。
エンジンのアップグレード自体が完了すると、古いオペレーティングシステムで新しいエンジンバージョンが一時的に利用可能になりますが、5 分以内にクラスター内のすべてのインスタンスが同時にオペレーティングシステムのアップグレードを開始します。DB クラスターはこの時点で約 6 分間使用できなくなります。アップグレードの完了後に、書き込みワークロードを再開できます。
このプロセスでは、次のイベントが生成されます。

- クラスターごとのイベントメッセージ。
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-(autogenerated snapshot ID)]`
  - `Database cluster major version has been upgraded`
- インスタンスごとのイベントメッセージ。
  - `Applying off-line patches to DB instance`
  - `DB instance shutdown`
  - `Finished applying off-line patches to DB instance`
  - `DB instance restarted`

> **Note**
>
> このエンジンリリース以降、Neptune はR4 インスタンスタイプをサポートしなくなりました。DB クラスター内の R4 インスタンスを使用している場合は、このリリースにアップグレードする前に、手動で別のインスタンスタイプに置き換える必要があります。ライターインスタンスが R4 の場合は、次の手順に従って移動してください。

このリリースへのアップグレード

Amazon Neptune 1.1.0.0.R2 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コン

ソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.1.0.0 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.1.0.0 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the instance before
>  proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.0.5.1 (2021-10-01)

2021 年 10 月 1 日現在、エンジンバージョン1.0.5.1. は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

このリリースの後続のパッチリリース

- リリース： 1.0.5.1.R2 (2021-10-26)
- リリース: 1.0.5.1.R3 (2022-01-13)

- メンテナンスリリース: 1.0.5.1.R4 (2022-05-16)

このエンジンリリースの新機能

- 指定したクエリの結果をキャッシュするための結果キャッシュが追加されました。
- Neptune openCypher で日付/時刻のサポートが追加されました。
- Neptune openCypher の要素への `List` および `Map` アクセスのサポートを追加しました。

このエンジンリリースの改良点

- Neptune openCypher エンドポイント名では大文字と小文字は区別されません。
- openCypher の説明が改善されました。
- Gremlin シングルアップサートクエリパターンが `iterate()` および `profile()` ステップで終了する問題が改善されました。
- Gremlin `keys()` および `property()` 関数のパフォーマンスが向上しました。
- Gremlin `dedup()` ステップは DFE でグローバルスコープとともに使用されるときに実行されます。
- 次の Gremlin HAS 述語は、DFE エンジンが有効な場合、DFE エンジンで実行されます。
  - `EQ`
  - `NEQ`
  - `LT`
  - `LTE`
  - `GT`
  - `GTE`
  - `BETWEEN`
  - `INSIDE`
  - `OUTSIDE`
  - `WITHIN`
  - `AND (connectives)`
  - `OR (connectives)`
- LIMIT クエリのパフォーマンスが向上しました。

- openCypher の一般的な集計クエリのパフォーマンスが向上しました。

このエンジンリリースで修正された不具合

- エッジを別のエッジに接続できる Gremlin のバグを修正しました。
- 最適でない結合戦略が選択される Gremlin のバグを修正しました。
- 100 を超えるプロパティが存在するとノードとリレーションシップのシリアル化が停止する Gremlin のバグを修正しました。
- グラフパターンが大きいクエリのクエリ実行計画が遅くなるバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.5.1 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.11`
- SPARQL バージョン: `1.1`

エンジンリリース 1.0.5.1 へのアップグレードパス

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

このリリースに自動的にアップグレードされることはありません。

このリリースへのアップグレード

Amazon Neptune 1.0.5.1 が利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.5.1 \
```

```
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.5.1 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
>   We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the
> instance before
>   proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune メンテナンスリリース、バージョン 1.0.5.1.R4 (2022-05-16)

2022 年 5 月 16 日現在、エンジンバージョン 1.0.5.1.R4 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.5.1.R4 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.11`
- SPARQL バージョン: `1.1`

エンジンリリース 1.0.5.1.R4 へのアップグレードパス

エンジンバージョン `1.0.5.1` を実行している場合、クラスターは次のメンテナンスウィンドウで自動的にこのメンテナンスパッチリリースにアップグレードされます。

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

このリリースへのアップグレード

Amazon Neptune 1.0.5.1.R4 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.5.1 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.5.1 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the instance before
>  proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.0.5.1.R3 (2022-01-13)

2022 年 1 月 13 日現在、エンジンバージョン 1.0.5.1.R3 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

このエンジンリリースで修正された不具合

- クエリが必要なリソースをすべて取得できなかった場合にリソースリークを引き起こすバグを修正しました。
- 要求されていないメモリ割り当てが原因で、クエリ実行中に発生していた小さなメモリリークを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.5.1.R3 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.11`
- SPARQL バージョン: `1.1`

エンジンリリース 1.0.5.1.R3 へのアップグレードパス

エンジンバージョン `1.0.5.1` を実行している場合、クラスターは次のメンテナンス期間中に自動的にこのパッチリリースにアップグレードされます。

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

このリリースへのアップグレード

Amazon Neptune 1.0.5.1.R3 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.5.1 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.5.1 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.0.5.1.R2 (2021-10-26)

2021 年 10 月 26 日現在、エンジンバージョン 1.0.5.1.R2 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

このエンジンリリースで修正された不具合

- 古いバージョンのグラフ要素の作成中に、繰り返し可能な読み取り分離において一時的なエラーが発生したときにサーバーを再起動するバグを修正しました。Neptune は代わりにエラーを返すので、クライアントは再試行できます。
- 単一のカーディナリティ更新中に一時的なエラーが発生したときにサーバーを再起動するバグを修正しました。Neptune は代わりにエラーを返すので、クライアントは再試行できます。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.5.1.R2 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.11`

- SPARQL バージョン: 1.1

エンジンリリース 1.0.5.1.R2 へのアップグレードパス

エンジンバージョン 1.0.5.1 を実行している場合、クラスターは次のメンテナンス期間中に自動的にこのパッチリリースにアップグレードされます。

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

このリリースへのアップグレード

Amazon Neptune 1.0.5.1.R2 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.5.1 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.5.1 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.0.5.0 (2021-07-27)

2021 年 7 月 27 日現在、エンジンバージョン1.0.5.0. は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

このリリースの後続のパッチリリース

- リリース： 1.0.5.0.R2 (2021-08-16)
- リリース： 1.0.5.0.R3 (2021-09-15)
- メンテナンスリリースバージョン 1.0.5.0.R5 (2022-05-16)

このエンジンリリースの新機能

- Neptune ML は、多くの新機能を備えた実稼働用にリリースされ、ラボモードではなくなっています。
- openCypher クエリ言語 (ラボモード)の初期サポートを追加しました。openCypher は、Cypher クエリ言語のオープンソース標準です。構文は、Cypher クエリ言語リファレンス (バージョン 9) に記載されており、openCypher プロジェクトによって維持されています。

  言語の Neptune の実装については、openCypher でNeptune グラフにアクセスする を参照してください。

  Neptune クライアントが openCypher クエリに使用する Bolt プロトコルもサポートされています。「Bolt プロトコルを使用して openCypher クエリを Neptune に作成する」を参照してください。

  opencyPher サポートは自動的に有効化されるようになりましたが、 Neptune DFEエンジン に依存し、現在ラボモードでののみ利用可能です。。`DFEQueryEngine` DB クラスターのパラメータのデフォルトである `neptune_lab_mode` 設定が `DFEQueryEngine=viaQueryHint` となりました。つまり、エンジンは有効ですが、`useDFE` クエリヒントが存在し、`true` に設定するためだけのクエリに使われます。`DFEQueryEngine=disabled` を設定して DFE エンジンを無効にすると、openCypher を使用できなくなります。

- SPARQL 1.1 Graph Store HTTP プロトコルのサポートを追加しました。「Amazon Neptune での SPARQL 1.1 グラフストア HTTP プロトコル (GSP) の使用」を参照してください。
- Neptune DFEエンジン に対するデフォルトのラボモード設定を `viaQueryHint` へ変更しました。すなわち、DFE エンジンがデフォルトで有効となりますが、`useDFE` クエリヒントが存在し、`true` に設定されているクエリのみに使用されます。
- Neptune DFE エンジンの統計量の計算を監視するために、新しい Amazon CloudWatch メトリクス、`StatsNumStatementsScanned`、を追加しました。「`StatsNumStatementsScanned` CloudWatch メトリクスを使用した統計計算のモニタリング」を参照してください。

このエンジンリリースの改良点

- Apache TinkerPop 3.4.11 のサポートを追加しました。

> **Important**
>
> TinkerPop バージョン 3.4.11 に変更が加えられ、クエリの処理方法の正確性が向上しましたが、現時点ではクエリのパフォーマンスに重大な影響を与える場合があります。
> たとえば、この種類のクエリの実行速度が大幅に遅くなる可能性があります。
>
> ```
> g.V().hasLabel('airport').
>   order().
>     by(out().count(),desc).
>   limit(10).
>   out()
> ```
>
> TinkerPop 3.4.11 の変更により、制限ステップの後の頂点は、最適ではない方法でフェッチされるようになりました。これを回避するには、barrier() ステップを `order().by()` の次の任意のポイントに追加して、クエリを変更できます。例:
>
> ```
> g.V().hasLabel('airport').
>   order().
>     by(out().count(),desc).
>   limit(10).
>   barrier().
>   out()
> ```

- SPARQL `joinOrder` クエリヒントが、Neptune DFE 代替クエリエンジンでサポートされるようになりました。
- DB クラスターの設定と機能についてより明確にするために、Neptune ステータス API の出力が拡張および再編成されました。

  新しい出力には、DB クラスターの機能に関するステータス情報を含む `features` 最上位オブジェクトと、設定情報を含む最上位 `settings` オブジェクトがあります。新しいフォーマットを確認するには、インスタンスステータスコマンドからの出力例 を参照してください。。
- `AFTER_SEQUENCE_NUMBER` ストリームがサーバー上の最後のイベント ID で要求され、そのイベント ID がすでに期限切れになっている場合のストリーミング変更ログの処理が改善されました。要求されたイベント ID がサーバー上で最後にパージされたイベント ID である場合、サーバーが期限切れのイベント ID エラーをスローしなくなりました。

このエンジンリリースで修正された不具合

- 数値の順序に関する Gremlin のバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.5.0 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.11`
- SPARQL バージョン: `1.1`

エンジンリリース 1.0.5.0 へのアップグレードパス

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

このリリースに自動的にアップグレードされることはありません。

このリリースへのアップグレード

Amazon Neptune 1.0.5.0 が利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コン

ソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.5.0 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.5.0 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the instance before
>  proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune メンテナンスリリース、バージョン 1.0.5.0.R5 (2022-05-16)

2022 年 5 月 16 日現在、エンジンバージョン 1.0.5.0.R5 は一般にデプロイされています。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.5.0.R5 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.11`
- SPARQL バージョン: `1.1`

エンジンリリース 1.0.5.0.R5 へのアップグレードパス

エンジンバージョン 1.0.5.0 を実行している場合、クラスターは次のメンテナンスウィンドウに自動的にこのメンテナンスパッチリリースにアップグレードされます。

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

このリリースへのアップグレード

Amazon Neptune 1.0.5.0.R5 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.5.0 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.5.0 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のような エラーが発生する可能性があります。

```
  We're sorry, your request to modify DB cluster (cluster identifier) has
failed.
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.0.5.0.R3 (2021-09-15)

2021 年 9 月 15 日現在、エンジンバージョン 1.0.5.0.R3 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

このエンジンリリースで修正された不具合

- 次のいずれかの状況でエンジンが応答しなくなるバグを修正しました。
  - バルクロードは、自動統計計算の実行と同時に発生します。
  - 統計情報の計算は、すでに発生していると同時に、手動で要求されました。
- デッドロック検出とロック取得でエンジンがクラッシュするバグを修正しました。
- Gremlin 推論クエリでリモート ML エンドポイントから不明なデータが検出されたときにエンジンがエラーをスローした Gremlin バグを修正しました。
- モデル変換ジョブとインスタンスの推奨事項に関連する ML モデル管理 API のいくつかのバグを修正しました。
- ノード ID とエッジ ID の生成時にエンジンがクラッシュする可能性があることの原因となるバグを修正しました。
- 大きなグラフパターンを持つクエリのクエリプランの生成が遅くなるバグを修正しました。
- 100 を超えるプロパティを持つノードを取得するときにクエリが停止する可能性がある openCypher のバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.5.0.R3 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.11`
- SPARQL バージョン: `1.1`

エンジンリリース 1.0.5.0.R3 へのアップグレードパス

エンジンバージョン 1.0.5.0 を実行している場合、クラスターは次のメンテナンス期間中に自動的にこのパッチリリースにアップグレードされます。

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

このリリースへのアップグレード

Amazon Neptune 1.0.5.0.R3 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.5.0 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.5.0 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.0.5.0.R2 (2021-08-16)

2021 年 8 月 16 日現在、エンジンバージョン 1.0.5.0.R2 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

このエンジンリリースで修正された不具合

- レプリカでエンジンの再起動が残る Neptune ルックアップキャッシュを作成したエンジンリリース `1.0.5.0`で行われた最適化を無効にしました。レプリカが再起動すると、ルックアップキャッシュがクリアされるようになりました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.5.0.R2 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.11`
- SPARQL バージョン: `1.1`

エンジンリリース 1.0.5.0.R2 へのアップグレードパス

エンジンバージョン `1.0.5.0` を実行している場合、クラスターは次のメンテナンス期間中に自動的にこのパッチリリースにアップグレードされます。

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

このリリースへのアップグレード

Amazon Neptune 1.0.5.0.R2 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
```

```
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.5.0 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.5.0 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できま

す。Neptune が手動スナップショットを作成する場合、その名前は preupgrade で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
>   We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the
> instance before
>   proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.0.4.2 (2021-06-01)

> **Note**
>
> エンジンリリースバージョン1.0.4.2.R2 は、1.0.4.2 の実際にリリースされる最初のバージョンでした。

トピック

- Amazon Neptune エンジンバージョン 1.0.4.2.R5 (2021-08-16)
- Amazon Neptune エンジンバージョン 1.0.4.2.R4 (2021-07-23)
- Amazon Neptune エンジンバージョン 1.0.4.2.R3 (2021-06-28)
- Amazon Neptune エンジンバージョン 1.0.4.2.R2 (2021-06-01)

- Amazon Neptune エンジンバージョン 1.0.4.2.R1 (2021-05-27)

Amazon Neptune エンジンバージョン 1.0.4.2.R5 (2021-08-16)

2021 年 8 月 16 日現在、エンジンバージョン 1.0.4.2.R5 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

このエンジンリリースで修正された不具合

- レプリカでエンジンの再起動が残る Neptune ルックアップキャッシュを作成したエンジンリリース `1.0.4.2.R4`で行われた最適化を無効にしました。レプリカが再起動すると、ルックアップキャッシュがクリアされるようになりました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.4.2.R5 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.10`
- SPARQL バージョン: `1.1`

エンジンリリース 1.0.4.2.R5 へのアップグレードパス

エンジンバージョン `1.0.4.2` を実行している場合、クラスターは次のメンテナンス期間中に自動的にこのパッチリリースにアップグレードされます。

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

Amazon Neptune エンジンバージョン 1.0.4.2.R4 (2021-07-23)

2021 年 7 月 23 日現在、エンジンバージョン1.0.4.2.R4 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

このエンジンリリースの改良点

- レプリカで高速リセットを実行した後の冗長キャッシュクリアを回避するために、ルックアップキャッシュの動作が改善されました。

- AFTER_SEQUENCE_NUMBER ストリームがサーバー上の最後のイベント ID で要求され、そのイベント ID がすでに期限切れになっている場合のストリーミング変更ログの処理が改善されました。要求されたイベント ID がサーバー上で最後にパージされたイベント ID である場合、サーバーが期限切れのイベント ID エラーをスローしなくなりました。

このエンジンリリースで修正された不具合

- クエリが 760 文字を超える文字列値の全体を返さない 1.0.4.0.R1 導入時のバグを修正しました。このバグの影響を受けた項は、RDF リテラルと URI、または Gremlin ID、キー、および文字列値でした。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.4.2.R4 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.10`
- SPARQL バージョン: `1.1`

エンジンリリース 1.0.4.2.R4 へのアップグレードパス

エンジンバージョン `1.0.4.2` を実行している場合、クラスターは次のメンテナンス期間中に自動的にこのパッチリリースにアップグレードされます。

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

Amazon Neptune エンジンバージョン 1.0.4.2.R3 (2021-06-28)

2021 年 6 月 28 日現在、エンジンバージョン1.0.4.2.R3 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

このエンジンリリースの既知の問題

問題:

スペースがあると `Accept` ヘッダーのメディアタイプが受け入れられない SPARQL バグ。

たとえば、 `-H "Accept: text/csv; q=1.0, */*; q=0.1"` があるクエリは CSV 出力ではなく JSON 出力を返します。

回避方法:

ヘッダー内の Accept 句中のスペースを削除すると、エンジンは正しい要求された形式で出力を返します。つまり、 `-H "Accept: text/csv; q=1.0, */*; q=0.1"` の代わりに、次のものを使用します。

```
-H "Accept: text/csv;q=1.0,*/*;q=0.1"
```

このエンジンリリースで修正された不具合

- 高速リセット後にレプリカのルックアップキャッシュをクリアする際のバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.4.2.R3 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.10`
- SPARQL バージョン: `1.1`

エンジンリリース 1.0.4.2.R3 へのアップグレードパス

DB クラスターが 1 つ以上の R5d インスタンスを使用していない限り、このパッチリリースはオプションです。クラスターに R5d インスタンスがある場合、次のメンテナンスウィンドウで自動的にアップグレードされます。そうしないと、このパッチリリースに自動的にアップグレードされません。

リリース`1.0.4.2.R2`を、AWS CLI apply-pending-maintenance-action (適用保留中のメンテナンスアクション) コマンド (ApplyPendingMaintenanceActionAPI) を使用して手動でこの `1.0.4.2.R3` リリースにアップグレードできます。

Amazon Neptune エンジンバージョン 1.0.4.2.R2 (2021-06-01)

2021 年 6 月 1 日現在、エンジンバージョン1.0.4.2.R2 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

このリリースの後続のパッチリリース

- リリース： 1.0.4.2.R3 (2021-06-28)

このエンジンリリースの既知の問題

問題:

スペースがあると Accept ヘッダーのメディアタイプが受け入れられない SPARQL バグ。

たとえば、 -H "Accept: text/csv; q=1.0, */*; q=0.1" があるクエリは CSV 出力ではなく JSON 出力を返します。

回避方法:

ヘッダー内の Accept 句中のスペースを削除すると、エンジンは正しい要求された形式で出力を返します。つまり、 -H "Accept: text/csv; q=1.0, */*; q=0.1" の代わりに、次のものを使用します。

```
-H "Accept: text/csv;q=1.0,*/*;q=0.1"
```

このエンジンリリースの新機能

- 新しい R5d インスタンスタイプを追加しました。これには、大量のプロパティ値または RDF リテラルルックアップを含むユースケースでの読み込みを高速化するためのルックアップキャッシュが含まれています。「Neptune ルックアップキャッシュは読み取りクエリを高速化できます。」を参照してください。
- 実験的な DFE エンジンを useDFE クエリヒントに基づきクエリ単位でのみ呼び出すことができる新しい lab-mode パラメータが追加されました。

このエンジンリリースの改良点

- TinkerPop 3.4.10 のサポートが追加されました。
- Gremlin スクリプト要求を送信するときの withStrategies( ) 設定ステップの使用のサポートが追加されました。具体的には、SubgraphStrategy、PartitionStrategy、ReadOnlyStrategy、EdgeLabelVerification および ReservedKeysVerificationStrategy すべてがサポートされています。
- クエリの途中での V( ) トラバーサルの最適化が追加されました。以前は、このようなトラバーサルは Neptune では最適化されていませんでした。
- バルクロードの baseUri および namedGraphUri パラメータとして使用する RFC 2141 URN のサポートを追加しました。

このエンジンリリースで修正された不具合

- 不正なクエリが有効として扱われるパーサーの Gremlin のバグを修正しました。
- `cap().unfold()` により `aggregate()` 副作用が展開され `valueMap()` 例外が発生する Gremlin のバグを修正しました。
- `addV()` ステップ後の `property()` ステップが「文字列にキャストできません」エラーで失敗する Gremlin のバグを修正しました。
- 一部の条件付き挿入パターンが同時変更例外を発生させないように Gremlin のバグを修正しました。
- クエリリクエストのタイムアウトがセッションタイムアウトを超えなくなるように、Gremlin のバグを修正しました。
- リモートサーバーが使用できないときに、LOAD または UNLOAD を使用した更新が HTTP コード 400 ではなく HTTP コード 500 で失敗することがあった SPARQL バグを修正しました。
- 32 ビット符号付き整数の制限 (2,147,483,647) より大きい `commitNum` または opNum 値が使用されるとストリーム API 呼び出しが失敗するバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.4.2.R2 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.10`
- SPARQL バージョン: `1.1`

エンジンリリース 1.0.4.2.R2 へのアップグレードパス

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

このリリースに自動的にアップグレードされることはありません。

このリリースへのアップグレード

Amazon Neptune 1.0.4.2.R2 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コン

ソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.4.2 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.4.2 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.0.4.2.R1 (2021-05-27)

エンジンリリース 1.0.4.2.R1 はデプロイされませんでした。

Amazon Neptune エンジンバージョン 1.0.4.1 (2020-12-08)

2020 年 12 月 8 日現在、エンジンバージョン1.0.4.1. は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

このリリースの後続のパッチリリース

- リリース： 1.0.4.1.R1.1 (2021-03-22)
- リリース： 1.0.4.1.R2 (2021-02-24)

> **Important**
>
> リリース： 1.0.4.0 (2020-10-12) は、Amazon Neptune へのすべての接続で TLS 1.2 と HTTPS を必須にしました。しかし、このリリースのバグにより、HTTPS 接続の強制を防ぐために以前に DB クラスターパラメーターを設定したお客様に対して、HTTP 接続や古くなった TLS 接続が機能し続けていました。
> このバグはパッチリリース 1.0.4.0.R2 および 1.0.4.1.R2 で修正されましたが、この修正によりパッチが自動的にインストールされるときに、予期しない接続エラーが生じるようになりました。このため、両方のパッチは元に戻され、TLS 1.2 のセットアップを更新できるよう、手動でのみインストールできます。
> Neptune へのすべての接続に SSL/TLS を使用することで、Gremlin コンソール、Gremlin ドライバー、Gremlin Python、.NET、nodeJs、REST API、およびロードバランサー接続との接続に影響します。これまで HTTP または古い TLS バージョンをこれらの一部またはすべてに使用していた場合は、最新のパッチにシステムアップデートする前に関連するクライアントとドライバを更新し、HTTPS のみを使用するようにコードを変更する必要があります。

このエンジンリリースの新機能

- Amazon Neptune に強力な機械学習機能を提供する Neptune ML 機能を導入しました。「グラフ上の機械学習のための Amazon Neptune ML」を参照してください。
- リモートソースから取得したデータを削除するためのカスタム SPARQL UNLOAD オペレーションを追加しました。「SPARQL UPDATE UNLOAD」を参照してください。

このエンジンリリースの改良点

- 同時変更例外を回避するために、Gremlin 条件付き挿入パターンをいくつか最適化しました。

このエンジンリリースで修正された不具合

- as() ステップを使用する特定のパターンのクエリに対して結果が欠損する場合がある Gremlin のバグを修正しました。
- project() のような別のステップ内にネストされている union() ステップを使用するとエラーとなる場合があったGremlin のバグを修正しました。
- project() ステップで Gremlin のバグを修正しました。
- none() ステップがうまくいかなかった文字列ベースのトラバーサルの Gremlin のバグを修正しました。
- inject() ステップに対して空のマップが引数としてサポートされていない文字列ベースのトラバーサルのグレムリンのバグを修正しました。。
- toList() が正常に動作しないなどのDFE エンジンでの文字列ベースのトラバーサル実行における Gremlin のバグを修正しました。
- 文字列クエリの iterate() ステップ使用時にトランザクションを閉じない Gremlin のバグを修正しました。
- is(P.gte(0)) パターン使用時にクエリが特定の状況において例外をスローする場合があった Gremlin のバグを修正しました。
- order().by(T.id) パターン使用時にクエリが特定の状況において例外をスローする場合があった Gremlin のバグを修正しました。
- addV().aggregate() パターン使用時にクエリが特定の状況において間違った結果となる場合があった Gremlin のバグを修正しました。
- path() ステップの後に project() パターンを使用する際にクエリが特定の状況において例外をスローする場合があった Gremlin のバグを修正しました。
- SUBSTR 関数が、空の文字列を返す代わりにエラーを通知する SPARQL のバグを修正しました。
- 非ブロッキングクエリプランでの結合操作が、バインドされていない変数の存在下で誤った結果を生成する可能性があった DFE エンジンのバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.4.1 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: 3.4.8
- SPARQL バージョン: 1.1

エンジンリリース 1.0.4.1 へのアップグレードパス

エンジンバージョン `1.0.4.1` を実行している場合、クラスターは次のメンテナンス期間中に自動的にこのパッチリリースにアップグレードされます。

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

このリリースへのアップグレード

Amazon Neptune 1.0.4.1 が公開されました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.4.1 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.4.1 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the
> instance before
>   proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.0.4.1.R1.1 (2021-03-22)

2021 年 3 月 22 日現在、エンジンバージョン 1.0.4.1.R1.1 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

このエンジンリリースで修正された不具合

- 既存のラベルとプロパティに追加または添付できる Gremlin 条件付き挿入パターンの最適化を無効にしました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.4.1.R1.1 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.8`
- SPARQL バージョン: `1.1`

エンジンリリース 1.0.4.1.R1.1 へのアップグレードパス

エンジンバージョン `1.0.4.1` を実行している場合、クラスターは次のメンテナンス期間中に自動的にこのパッチリリースにアップグレードされます。

このリリースへのアップグレード

Amazon Neptune 1.0.4.1.R1.1 が公開されました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
```

```
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.4.1 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.4.1 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できま

す。Neptune が手動スナップショットを作成する場合、その名前は preupgrade で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the instance before
>   proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.0.4.1.R2 (2021-02-24)

2021 年 2 月 24 日現在、エンジンバージョン 1.0.4.1.R2 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

このリリースの後続のパッチリリース

- リリース：1.0.4.1.R2.1 (2021-03-11)

このエンジンリリースの新機能

- Neptune は、bzip2 形式の単一ファイルの圧縮をサポートしています。「ロードデータ形式」を参照してください。

このエンジンリリースで修正された不具合

- リリース： 1.0.4.0 (2020-10-12) と TLS 1.2 ではなく HTTP または、TLSの以前のバージョンを使用して Neptune への接続ができた HTTPS のバグを修正しました。

> **Important**
>
> Neptune へのすべての接続に SSL/TLS を使用しなければならないことは、大きな変更になる可能性があります。これは、Gremlin コンソール、Gremlin ドライバー、Gremlin Python、.NET、nodeJs、REST API、およびロードバランサー接続との接続に影響します。これまで HTTP または古い TLS バージョンをこれらの一部またはすべてに使用していた場合は、このパッチをインストールする前に関連するクライアントとドライバを更新し、HTTPS のみを使用するようにコードを変更する必要があります。

- `InternalFailureException` が発生した際に特定の状況において、レスポンスコードとして `ConcurrentModificationException` が設定されるという Gremlin のバグを修正しました。
- 特定の条件下でエッジや頂点を更新すると過渡 `InternalFailureException` が発生する可能性がある Gremlin バグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.4.1.R2 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.8`
- SPARQL バージョン: `1.1`

エンジンリリース 1.0.4.1.R2 へのアップグレードパス

エンジンバージョン `1.0.4.1` を実行している場合、クラスターは次のメンテナンス期間中に自動的にこのパッチリリースにアップグレードされます。

このリリースへのアップグレード

Amazon Neptune 1.0.4.1.R2 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コン

ソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.4.1 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.4.1 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the instance before
>   proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.0.4.1.R2.1 (2021-03-11)

2021 年 3 月 11 日現在、エンジンバージョン 1.0.4.1.R2.1 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

このエンジンリリースで修正された不具合

- 既存のラベルとプロパティに追加または添付できる Gremlin 条件付き挿入パターンの最適化を無効にしました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.4.1.R2.1 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.8`
- SPARQL バージョン: `1.1`

エンジンリリース 1.0.4.1.R2.1 へのアップグレードパス

エンジンバージョン `1.0.4.1.R2` を実行している場合、クラスターは次のメンテナンス期間中に自動的にこのパッチリリースにアップグレードされます。

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

このリリースへのアップグレード

Amazon Neptune 1.0.4.1.R2.1 が公開されました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.4.1.R2 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.4.1.R2 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。

```
  We're sorry, your request to modify DB cluster (cluster identifier) has
failed.
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.0.4.0 (2020-10-12)

2020 年 10 月 12 日現在、エンジンバージョン1.0.4.0. は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

このリリースの後続のパッチリリース

- リリース： 1.0.4.0.R2 (2021-02-24)

このエンジンリリースの新機能

- Gremlin のフレームレベルの圧縮を追加しました。

このエンジンリリースの改良点

- Amazon Neptune では、次の強力な暗号スイートを使用して、すべてのリージョンで Neptune へのすべての接続に TLSv1.2 プロトコルで Secure Sockets Layer (SSL) を使用する必要があります。
  - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
  - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
  - TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
  - TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
  - TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
  - TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA

これは、Neptune への REST 接続と WebSocket 接続の両方に当てはまります。すべてのリージョンで Neptune に接続する場合は、HTTP ではなく HTTPS を使用する必要があることを意味します。

HTTP または TLS 1.1 を使用するクライアント接続はどこでもサポートされなくなるため、このエンジンリリースにアップグレードする前に、クライアントとコードが TLS 1.2 および HTTPS を使用するように更新されていることを確認してください。

> **Important**
>
> Neptune へのすべての接続に SSL/TLS を使用しなければならないことは、大きな変更になる可能性があります。これは、Gremlin コンソール、Gremlin ドライバー、Gremlin Python、.NET、nodeJs、REST API、およびロードバランサー接続との接続に影響します。これらのいずれかまたはすべてに HTTP を使用している場合は、関連するクライアントとドライバを更新し、HTTPS を使用するようにコードを変更する必要があります。そうしないと、接続が失敗します。
>
> このリリースのバグにより、HTTPS 接続の強制を防ぐために以前に DB クラスターパラメーターを設定したお客様に対して、HTTP 接続や古くなった TLS 接続が機能し続けることができました。このバグはパッチリリース 1.0.4.0.R2 および 1.0.4.1.R2 で修正されましたが、この修正によりパッチが自動的にインストールされるときに、予期しない接続エラーが生じるようになりました。
>
> このため、両方のパッチは元に戻され、TLS 1.2 のセットアップを更新できるよう、手動でのみインストールできます。

- TinkerPop 3.4.8 にアップグレードされました。これは下位互換のアップグレードです。詳細については、TinkerPop の変更ログを参照してください。
- `properties()` ステップの Gremlin のパフォーマンスが改善されました。
- `BindOp` および `MultiplexerOp` 説明レポートとプロファイルレポートに関する詳細の追加。
- キャッシュミスがある場合のパフォーマンスを向上させるために、データのプリフェッチを追加しました。
- 新しい追加されました。CSV ロードで空の文字列を有効なプロパティ値として処理できるようにするバルクローダーの `parserConfiguration` パラメータで `allowEmptyStrings` 設定を追加しました (Neptune ローダーのリクエストパラメータ を参照)。
- ローダーでは、複数値の CSV 列でエスケープされたセミコロンを使用できるようになりました。

このエンジンリリースで修正された不具合

- both( ) ステップに関連する Gremlin メモリリークの可能性を修正しました。
- 「/」で終わるエンドポイントが正しく処理されないため、リクエストメトリクスが欠落していたバグを修正しました。
- DFE エンジンがラボモードで有効になっているときに、レプリカが重い負荷で停止して再起動するバグを修正しました。
- メモリ不足が原因でバルクロードが失敗したときに、正しいエラーメッセージが報告されないバグを修正しました。
- SPARQL クエリレスポンスの Content-Encoding ヘッダーに文字エンコーディングが配置された SPARQL バグを修正しました。charset が代わりに Content-Type ヘッダーに配置され、HTTP クライアントが使用されている文字セットを自動的に認識できるようになります。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.4.0 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: 3.4.8
- SPARQL バージョン: 1.1

エンジンリリース 1.0.4.0 へのアップグレードパス

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

このリリースに自動的にアップグレードされることはありません。

このリリースへのアップグレード

Amazon Neptune 1.0.4.0 が利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.4.0 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.4.0 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できま

す。Neptune が手動スナップショットを作成する場合、その名前は preupgrade で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のような エラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has failed.
> Cannot modify engine version because instance (instance identifier) is
> running on an old configuration. Apply any pending maintenance actions on the instance before
> proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.0.4.0.R2 (2021-02-24)

2021 年 2 月 24 日現在、エンジンバージョン 1.0.4.0.R2 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

このエンジンリリースで修正された不具合

- リリース： 1.0.4.0 (2020-10-12) と TLS 1.2 ではなく HTTP または、TLSの以前のバージョンを使用して Neptune への接続ができた HTTPS のバグを修正しました。

> **Important**
>
> Neptune へのすべての接続に SSL/TLS を使用しなければならないことは、大きな変更になる可能性があります。これは、Gremlin コンソール、Gremlin ドライバー、Gremlin Python、.NET、nodeJs、REST API、およびロードバランサー接続との接続に影響しま

す。これまで HTTP または古い TLS バージョンをこれらの一部またはすべてに使用していた場合は、このパッチをインストールする前に関連するクライアントとドライバを更新し、HTTPS のみを使用するようにコードを変更する必要があります。

- # で終わるラベルを含む CSV 一括ロードのバグを修正しました。
- `InternalFailureException` が発生した際に特定の状況において、レスポンスコードとして `ConcurrentModificationException` が設定されるという Gremlin のバグを修正しました。
- 特定の条件下でエッジや頂点を更新すると過渡 `InternalFailureException` が発生する可能性がある Gremlin バグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.4.0.R2 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.8`
- SPARQL バージョン: `1.1`

エンジンリリース 1.0.4.0.R2 へのアップグレードパス

エンジンバージョン `1.0.4.0` を実行している場合、クラスターは次のメンテナンス期間中に自動的にこのパッチリリースにアップグレードされます。

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

このリリースへのアップグレード

Amazon Neptune 1.0.4.0.R2 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
```

```
    --engine-version 1.0.4.0 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.4.0 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後

に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
>   We're sorry, your request to modify DB cluster (cluster identifier) has failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the instance before
>   proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.0.3.0 (2020-08-03)

2020 年 8 月 3 日現在、エンジンバージョン1.0.3.0. は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

このリリースの後続のパッチリリース

- リリース： 1.0.3.0.R2 (2020-10-12)
- リリース： 1.0.3.0.R3 (2021-02-19)

このエンジンリリースの新機能

- Neptune は、クエリの実行を大幅に高速化できる、新しい代替クエリエンジン (DFE) を導入しました。「Amazon Neptune 代替クエリエンジン (DFE)」を参照してください。

- DFE は、新しい統計エンドポイントを介して管理される Neptune グラフデータに関する事前生成された統計情報に依存します。「DFE 統計」を参照してください。
- 新しい `includeQueuedLoads` パラメータを FALSE に設定することで Loader Get-Status API によって返されるロード ID のリストからキューに入れられたロードジョブを除外できるようになりました。「Neptune Loader Get-Status リクエストパラメータ」を参照してください。
- Neptune は、応答チャンクを返し始めた後にリクエストが失敗した場合にエラーコードとメッセージを含むことができる SPARQL クエリレスポンスの末尾ヘッダーをサポートするようになりました。「マルチパートの SPARQL レスポンスのオプションの HTTP 末尾ヘッダー」を参照してください。
- Neptune は、Gremlin クエリのチャンク応答エンコーディングを有効化できるようになりました。SPARQL の場合と同様に、応答チャンクには末尾のヘッダーがあり、クエリが応答チャンクを返し始めた後にエラーが発生した場合、エラーコードとメッセージを含めることができます。「オプションの HTTP 末尾ヘッダーを使用して、複数パートの Gremlin 応答を有効にする」を参照してください。

このエンジンリリースの改良点

- Gremlin でフルテキスト検索用のバッチリクエストのサイズを ElasticSearch に提供できるようになりました。
- SPARQL GROUP BY クエリのメモリ使用量が改善されました。
- 特定の非バインドフィルターをプルーニングする新しい Gremlin クエリオプティマイザを追加しました。
- IAM を使用して認証された WebSocket 接続をオープン状態に保つことができる最大時間を 36 時間から 10 日に延長しました。

このエンジンリリースで修正された不具合

- POST リクエストでエンコードされていない URL パラメータを送信すると、Neptune が HTTP ステータスコード 500 と `InternalServerErrorException` を返したバグを修正しました。現在、Neptune は HTTP ステータスコード 400 と `BadRequestException` を返し `Failure to process the POST request parameters` というメッセージが表示されます。
- WebSocket 接続の失敗が正しく報告されない Gremlin のバグを修正しました。
- sideEffects の消失に関する Gremlin のバグを修正しました。

- フルテキスト検索 batchsize パラメータが正しくサポートされていないという Gremlin のバグを修正しました。
- toV に関する各方向に対して fromV および bothE を個別に処理するGremlin のバグを修正しました。
- Edge pathType ステップの hasLabel に関する Gremlin のバグを修正しました。
- 静的バインディングによるジョインの並べ替えが正しく動作しない SPARQL バグを修正しました。
- 利用できない Amazon S3 バケットが正しく報告されない SPARQL UPDATE LOAD のバグを修正しました。
- サブクエリ内のサービスノードに関する問題が正しく報告されない SPARQL バグを修正しました。
- ネストされた FILTER EXISTS または FILTER NOT EXISTS 条件を含むクエリが適切に評価されない SPARQL のバグを修正しました。
- 生成クエリを使用して SPARQL サービスエンドポイントを呼び出すときに、生成された重複バインディングを正しく処理するように SPARQL バグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.3.0 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: 3.4.3
- SPARQL バージョン: 1.1

エンジンリリース 1.0.3.0 へのアップグレードパス

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

クラスターの AutoMinorVersionUpgrade パラメータが True に設定されている場合、クラスターは、このリリース日から 2 ~ 3 週間後、メンテナンス期間中に自動的にこのエンジンリリースにアップグレードされます。

このリリースへのアップグレード

Amazon Neptune 1.0.3.0 が公開されました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.3.0 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.3.0 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
>   We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the
> instance before
>   proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.0.3.0.R3 (2021-02-19)

2021 年 2 月 19 日現在、エンジンバージョン 1.0.3.0.R3 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

このエンジンリリースで修正された不具合

- # で終わるラベルを含む CSV 一括ロードのバグを修正しました。
- `as()` ステップを使用する特定のパターンのクエリに対して結果が欠損する場合がある Gremlin のバグを修正しました。

- union() のような別のステップ内にネストされている project() ステップを使用するとエラーとなる場合があった Gremlin のバグを修正しました。
- toList() のような終了メソッドが用いられる際に実験的な DFE エンジンにおいて文字列トラバーサル実行にあった Gremlin のバグを修正しました。
- 文字列クエリの iterate() ステップ使用時にトランザクションを閉じない Gremlin のバグを修正しました。
- 特定の条件下で例外をスローする is(P.gte(0)) パターン使用時にクエリが生じる場合があった Gremlin のバグを修正しました。
- ConcurrentModificationException が発生した際に特定の状況において、レスポンスコードとして InternalFailureException が設定されるという Gremlin のバグを修正しました。
- 特定の条件下でエッジや頂点を更新すると過渡 InternalFailureException が発生する可能性がある Gremlin バグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.3.0.R3 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: 3.4.8
- SPARQL バージョン: 1.1

エンジンリリース 1.0.3.0.R3 へのアップグレードパス

エンジンバージョン 1.0.3.0 を実行している場合、クラスターは次のメンテナンス期間中に自動的にこのパッチリリースにアップグレードされます。

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

このリリースへのアップグレード

Amazon Neptune 1.0.3.0.R3 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.3.0 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.3.0 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できま

す。Neptune が手動スナップショットを作成する場合、その名前は preupgrade で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
>   We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the
> instance before
>   proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.0.3.0.R2 (2020-10-12)

2020 年 10 月 12 日現在、エンジンバージョン 1.0.3.0.R2 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

このエンジンリリースの改良点

- `properties()` ステップの Gremlin のパフォーマンスが改善されました。
- `BindOp` および `MultiplexerOp` 説明レポートとプロファイルレポートに関する詳細の追加。
- SPARQL クエリレスポンスについて、`charset` を Content-Type ヘッダーに追加し、HTTP クライアントが使用されている文字セットを自動的に認識できるようにします。

このエンジンリリースで修正された不具合

- `CancellationException` が処理されないという SPARQL のバグを修正しました。

- ネストされたオプションを含むクエリが正しく動作しない SPARQL バグを修正しました。
- ConcurrentModificationException によりクエリがハングする可能性があった SPARQL のバグを修正しました。
- クエリ応答が gzip 圧縮されない SPARQL バグを修正しました。
- groupBy() ステップで Gremlin のバグを修正しました。
- local() ステップの内側にある aggregate() ステップの使用に関連する Gremlin のバグを修正しました。
- 集計値を使用する述語が続く bothE() の使用に関連する Gremlin のバグを修正しました。
- bothE() ステップによる repeat() ステップの使用に関連する Gremlin のバグを修正しました。
- both() ステップに関連する Gremlin メモリリークの可能性を修正しました。
- 「/」で終わるエンドポイントが正しく処理されないため、リクエストメトリクスが欠落していたバグを修正しました。
- リクエストキューがいっぱいでない場合でも ThrottlingException が発生する可能性があるバグを修正しました。
- LOAD_DATA_FAILED_DUE_TO_FEED_MODIFIED_OR_DELETE のような理由でロードが失敗したときにロードステータスを取得するバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.3.0.R2 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: 3.4.3
- SPARQL バージョン: 1.1

エンジンリリース 1.0.3.0.R2 へのアップグレードパス

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

クラスターの AutoMinorVersionUpgrade パラメータが True に設定されている場合、クラスターは、このリリース日から 2~3 週間後、メンテナンス期間中に自動的にこのエンジンリリースにアップグレードされます。

このリリースへのアップグレード

Amazon Neptune 1.0.3.0.R2 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.3.0 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.3.0 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.0.2.2 (2020-03-09)

2020 年 3 月 9 日現在、エンジンバージョン1.0.2.2. は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

このリリースの後続のパッチリリース

- リリース： 1.0.2.2.R2 (2020-04-02)
- リリース： 1.0.2.2.R3 (2020-07-22)

- リリース： 1.0.2.2.R4 (2020-07-23)
- リリース：1.0.2.2.R5 (2020-10-12)
- リリース： 1.0.2.2.R6 (2021-02-19)

このエンジンリリースの改良点

- ロールバック中のトランザクションに関するステータス API に情報を追加しました。「インスタンスのステータス」を参照してください。
- Apache TinkerPop のバージョンを 3.4.3 にアップグレードしました。

  バージョン 3.4.3 は、Neptune (3.4.1) でサポートされている以前のバージョンとの下位互換性があります。このバージョンには動作に小さな変更が 1 つあり、存在しないセッションを閉じようとすると、Gremlin はエラーを返さなくなりました (「Prevent error when closing sessions that don't exist」を参照)。
- Gremlin 全文検索ステップの実行におけるパフォーマンスのボトルネックを解消しました。

このエンジンリリースで修正された不具合

- クエリでの空のグラフパターン処理における SPARQL バグを修正しました。
- URL エンコードされたクエリのエンコードされていないセミコロンの処理における SPARQL バグを修正しました。
- Union ステップで繰り返される頂点の処理における Gremlin のバグを修正しました。
- `.repeat()` 内の `.simplePath()` または `.cyclicPath()` の一部のクエリが誤った結果を返す原因となっていた Gremlin のバグを修正しました。
- 子トラバーサルがソリューションを返さなかった場合に、`.project()` が誤った結果を返す原因となっていた Gremlin バグを修正しました。
- 読み取り/書き込みの競合によるエラーによって、`ConcurrentModificationException` ではなく `InternalFailureException` が発生していた Gremlin のバグを修正しました。
- `.group().by(...).by(values("property"))` 失敗の原因となっていた Gremlin のバグを修正しました。
- 全文検索ステップのプロファイル出力における Gremlin のバグを修正しました。
- Gremlin セッションでのリソースリークを修正しました。
- ステータス API が注文可能な正しいバージョンを報告できないバグを修正しました。

- バルクロードリクエストでソースとして使用する Amazon S3 以外の場所への URL を許可するバルクローダーのバグを修正しました。
- 詳細なロードステータスのバルクローダーのバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.2.2 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.3`
- SPARQL バージョン: `1.1`

エンジンリリース 1.0.2.2 へのアップグレードパス

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

クラスターの `AutoMinorVersionUpgrade` パラメータが `True` に設定されている場合、クラスターは、このリリース日から 2 ~ 3 週間後、メンテナンス期間中に自動的にこのエンジンリリースにアップグレードされます。

このリリースへのアップグレード

Amazon Neptune 1.0.2.2 が公開されました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.2.2 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
```

```
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.2.2 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。

```
  We're sorry, your request to modify DB cluster (cluster identifier) has
failed.
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.0.2.2.R6 (2021-02-19)

2021 年 2 月 19 日現在、エンジンバージョン1.0.2.2.R6 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

このエンジンリリースで修正された不具合

- `ConcurrentModificationException` が発生した際に特定の状況において、レスポンスコードとして `InternalFailureException` が設定されるという Gremlin のバグを修正しました。
- 特定の条件下でエッジや頂点を更新すると過渡 `InternalFailureException` が発生する可能性がある Gremlin バグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.2.2.R6 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.8`
- SPARQL バージョン: `1.1`

エンジンリリース 1.0.2.2.R6 へのアップグレードパス

エンジンバージョン `1.0.2.2` を実行している場合、クラスターは次のメンテナンス期間中に自動的にこのパッチリリースにアップグレードされます。

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

このリリースへのアップグレード

Amazon Neptune 1.0.2.2.R6 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.2.2 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.2.2 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.0.2.2.R5 (2020-10-12)

2020 年 10 月 12 日現在、エンジンバージョン1.0.2.2.R5 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

このエンジンリリースの改良点

- `properties()` ステップの Gremlin のパフォーマンスが改善されました。
- `BindOp` および `MultiplexerOp` 説明レポートとプロファイルレポートに関する詳細の追加。
- SPARQL クエリレスポンスについて、`charset` を Content-Type ヘッダーに追加し、HTTP クライアントが使用されている文字セットを自動的に認識できるようにします。

このエンジンリリースで修正された不具合

- `CancellationException` が処理されないという SPARQL のバグを修正しました。
- ネストされたオプションを含むクエリが正しく動作しない SPARQL バグを修正しました。
- `ConcurrentModificationException` によりクエリがハングする可能性があった SPARQL のバグを修正しました。
- クエリ応答が gzip 圧縮されない SPARQL バグを修正しました。
- `groupBy()` ステップで Gremlin のバグを修正しました。
- `local()` ステップの内側にある `aggregate()` ステップの使用に関連する Gremlin のバグを修正しました。
- 集計値を使用する述語が続く `bothE()` の使用に関連する Gremlin のバグを修正しました。
- `bothE()` ステップによる `repeat()` ステップの使用に関連する Gremlin のバグを修正しました。
- `both()` ステップに関連する Gremlin メモリリークの可能性を修正しました。
- 「/」で終わるエンドポイントが正しく処理されないため、リクエストメトリクスが欠落していたバグを修正しました。
- リクエストキューがいっぱいでない場合でも `ThrottlingException` が発生する可能性があるバグを修正しました。
- `LOAD_DATA_FAILED_DUE_TO_FEED_MODIFIED_OR_DELETE` のような理由でロードが失敗したときにロードステータスを取得するバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.2.2.R5 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.3`
- SPARQL バージョン: `1.1`

エンジンリリース 1.0.2.2.R5 へのアップグレードパス

エンジンバージョン `1.0.2.2` を実行している場合、クラスターは次のメンテナンス期間中に自動的にこのパッチリリースにアップグレードされます。

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

このリリースへのアップグレード

Amazon Neptune 1.0.2.2.R5 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.2.2 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.2.2 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のような エラーが発生する可能性があります。

```
  We're sorry, your request to modify DB cluster (cluster identifier) has
failed.
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.0.2.2.R4 (2020-07-23)

2020 年 7 月 23 日現在、エンジンバージョン1.0.2.2.R4 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

このエンジンリリースの改良点

- 未使用のメモリをより頻繁にオペレーティングシステムに解放することでメモリ使用量を改善しました。
- SPARQL GROUP BY クエリのメモリ使用率も改善されました。
- IAM を使用して認証された WebSocket 接続をオープン状態に保つことができる最大時間を 36 時間から 10 日に延長しました。
- クエリのレイテンシーの診断とインスタンスタイプのチューニングに役立つ `BufferCacheHitRatio` CloudWatch メトリクスが追加されました。「Neptune メトリクス」を参照してください。

このエンジンリリースで修正された不具合

- アイドル状態または期限切れの IAM WebSocket 接続を閉じるバグを修正しました。Neptune は、接続を閉じる前に閉じるフレームを送信するようになりました。
- ネストされた FILTER EXISTS および / または FILTER NOT EXISTS 条件を含むクエリの評価における SPARQL のバグを修正しました。
- 特定の極端な条件下でサーバーにブロックされたスレッドが発生した SPARQL クエリ終了バグを修正しました。
- `hasLabel` ステップの Edge pathType に関する Gremlin のバグを修正しました。
- `bothE` に関する各方向に対して`toV` および `fromV` を個別に処理するGremlin のバグを修正しました。

- sideEffects の消失に関する Gremlin のバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.2.2.R4 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.3`
- SPARQL バージョン: `1.1`

エンジンリリース 1.0.2.2.R4 へのアップグレードパス

エンジンバージョン `1.0.2.2` を実行している場合、クラスターは次のメンテナンス期間中に自動的にこのパッチリリースにアップグレードされます。

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

このリリースへのアップグレード

Amazon Neptune 1.0.2.2.R4 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.2.2 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.2.2 ^
```

```
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。

```
  We're sorry, your request to modify DB cluster (cluster identifier) has
failed.
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.0.2.2.R3 (2020-07-22)

エンジンリリース1.0.2.2.R3 はエンジンリリース 1.0.2.2.R4 に組み込まれています。

Amazon Neptune エンジンバージョン 1.0.2.2.R2 (2020-04-02)

2020 年 4 月 2 日現在、エンジンバージョン1.0.2.2.R2 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

このエンジンリリースの改良点

- 最大 64 個の一括ロードジョブをキューに入れることができるようになりました。ジョブが終了するのを待って次のジョブを開始する必要はありません。また、load コマンドの dependencies パラメータを使用して、以前にキューに入った 1 つ以上のロードジョブの正常な完了を条件に、キューに入っているロードリクエストを実行することもできます。「Neptune ローダーコマンド」を参照してください。
- 全文検索出力をソートできるようになりました (フルテキスト検索パラメータ を参照)。
- Neptune ストリームを呼び出すための DB クラスターパラメータが用意されました。この機能はラボモードから移行されています。「Neptune Streams の有効化」を参照してください。

このエンジンリリースで修正された不具合

- インスタンスの作成を遅らせる、サーバー起動時の確率的な障害を修正しました。

- クエリ内の BIND ステートメントが結合順序計画でオプティマイザを非選択パターンで開始するという、オプティマイザの問題を修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.2.2.R2 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.3`
- SPARQL バージョン: `1.1`

エンジンリリース 1.0.2.2.R2 へのアップグレードパス

エンジンバージョン `1.0.2.2` を実行している場合、クラスターは次のメンテナンス期間中に自動的にこのパッチリリースにアップグレードされます。

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

このリリースへのアップグレード

Amazon Neptune 1.0.2.2.R2 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.2.2 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
```

```
    --engine-version 1.0.2.2 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。

```
  We're sorry, your request to modify DB cluster (cluster identifier) has
failed.
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.0.2.1 (2019-11-22)

このリリースの後続のパッチリリース

- リリース：1.0.2.1.R6 (2020-04-22)
- リリース：1.0.2.1.R5 (2020-04-22) このパッチリリースはデプロイされませんでした。
- リリース：1.0.2.1.R4 (2019-12-20)
- リリース：1.0.2.1.R3 (2019-12-12)
- リリース：1.0.2.1.R2 (2019-11-25)

このエンジンリリースの新機能

- Amazon OpenSearch Service との統合により、フルテキスト検索機能を追加しました。Neptune フルテキスト検索 を参照してください。
- 多数の述語に対して 4 番目のインデックス (OSGP インデックス) を作成するためにラボモードを使用するオプションを追加しました。「OSGP インデックス」を参照してください。
- SPARQL Explain に詳細モードを追加しました。詳細については、「SPARQL explain を使う」と「詳細モード出力」を参照してください。
- エンジンステータスレポートにラボモード情報を追加しました。詳細については、「インスタンスのステータス」を参照してください。

- DB クラスターのスナップショットを AWS リージョン間でコピーできるようになりました。「スナップショットのコピー」を参照してください。

このエンジンリリースの改良点

- 多数の述語を処理する際のパフォーマンスが向上しました。
- クエリの最適化が強化されました。これはお客様には完全に透過的ですが、アプリケーションをアップグレードする前にテストし、正常に動作することを確認することをお勧めします。
- エラー報告のマイナーな機能強化。
- Gremlin の `.project()` ステップと `.identity()` ステップの最適化を追加しました。
- 非ターミナル Gremlin の `.union()` ケースの最適化を追加しました。
- Gremlin `.path().by()` トラバーサルのネイティブサポートを追加しました。
- Gremlin `.coalesce()` のネイティブサポートを追加しました。
- 一括書き込みのさらなる最適化。
- HTTPS 接続では、期限切れの安全でない暗号が使用されないように、TLS バージョン 1.2 以上の使用が要求されるようになりました。

このエンジンリリースで修正された不具合

- Gremlin `addE()` 内部トラバーサル処理のバグを修正しました。
- AST 注釈が子トラバーサルから親にリークすることで引き起こされる Gremlin のバグを修正しました。
- `.otherV()` が `select()` の後で呼び出されたときに Gremlin で発生するバグを修正しました。
- `bothE()` ステップの後に表示された場合、一部の `.hasLabel()` ステップが失敗する原因となった Gremlin のバグを修正しました。
- Gremlin の .sum() と .project() のマイナーな修正を行いました。
- 閉じ括弧がない SPARQL クエリの処理に伴うバグを修正しました。
- SPARQL Explain のいくつかのマイナーなバグを修正しました。
- 複数のロードステータス取得リクエストの同時処理に伴うバグを修正しました。
- 一部の Gremlin トラバーサルを `.project()` ステップで実行する際に消費されるメモリを削減しました。

- SPARQL での特殊値の数値比較を修正しました。「標準コンプライアンス」を参照してください。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.2.1 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.1`
- SPARQL バージョン: `1.1`

エンジンリリース 1.0.2.1 へのアップグレードパス

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

このリリースに自動的にアップグレードされることはありません。

このリリースへのアップグレード

Amazon Neptune 1.0.2.1 が公開されました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.2.1 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.2.1 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
> ```

```
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.0.2.1.R6 (2020-04-22)

2020 年 4 月 22 日現在、エンジンバージョン1.0.2.1.R6 は一般にご利用いただけます。新しいリリースがすべてのリージョンで利用可能になるまでに数日かかります。

このエンジンリリースで修正された不具合

- `ConcurrentModificationConflictException` と `TransactionException` が `NeptuneGremlinException` に変換されず、`InternalFailureException` がお客様に返送される不具合を修正しました。
- サーバーの準備が完了する前に、Neptune によりステータスが正常と報告されるバグを修正しました。
- 2 つの `value->id` マッピングを同時に挿入すると、ディクショナリとユーザートランザクションのコミットが機能しない不具合を修正しました。
- ロードステータスのシリアル化の不具合を修正しました。
- Gremlin セッションの不具合を修正しました。
- サーバーの起動に失敗したときに、Neptune により例外がスローされない不具合を修正しました。
- チャンネルを閉じる前に Neptune がウェブソケットクローズフレームを送信できない不具合を修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.2.1.R6 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.1`
- SPARQL バージョン: `1.1`

エンジンリリース 1.0.2.1.R6 へのアップグレードパス

エンジンバージョン `1.0.2.1` を実行している場合、クラスターは次のメンテナンス期間中に自動的にこのパッチリリースにアップグレードされます。

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

このリリースへのアップグレード

Amazon Neptune 1.0.2.1.R6 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.2.1 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.2.1 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のような エラーが発生する可能性があります。

```
  We're sorry, your request to modify DB cluster (cluster identifier) has
failed.
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.0.2.1.R5 (2020-04-22)

エンジンリリース 1.0.2.1.R5 はデプロイされませんでした。

Amazon Neptune エンジンバージョン 1.0.2.1.R4 (2019-12-20)

このエンジンリリースの改良点

- Neptune は、すべてのフルテキスト検索呼び出しを実行パイプラインで常に最初に試行するようになりました。これにより、OpenSearch への呼び出し量が減り、パフォーマンスが大幅に向上します。「フルテキスト検索クエリの実行」を参照してください。
- Neptune は、存在しないプロパティ、頂点、エッジにアクセスしようとした場合に `IllegalArgumentException` を発生させるようになりました。以前は、Neptune ではその状況で `UnsupportedOperationException` が発生しました。

  たとえば、存在しない頂点を参照するエッジを追加しようとすると、`IllegalArgumentException` が発生します。

このエンジンリリースで修正された不具合

- `project-by` 内部の `union` トラバーサルが結果を返さないか、誤った結果が返される Gremlin のバグを修正しました。
- ネストされた `.project().by()` ステップが誤った結果を返す原因となっていた Gremlin のバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.2.1.R4 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.1`
- SPARQL バージョン: `1.1`

エンジンリリース 1.0.2.1.R4 へのアップグレードパス

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

ただし、このリリースへの自動更新はサポートされていません。

このリリースへのアップグレード

Amazon Neptune 1.0.2.1.R4 が一般的に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.2.1 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.2.1 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.0.2.1.R3 (2019-12-12)

このエンジンリリースで修正された不具合

- neptune_lab_mode パラメータの ObjectIndex 値を使用して、ラボモード で機能を正しく有効化していても、OSGP インデックスが無効になっていたバグを修正しました。
- .project().by() ステップ内にある .fold() を使用して Gremlin クエリに影響したバグを修正しました。たとえば、次のクエリは不完全な結果を返します。

```
g.V().project("a").by(valueMap().fold())
```

- RDF データのバルクロードにおけるパフォーマンスのボトルネックを修正しました。
- ストリームが有効化され、レプリカがプライマリの前に再起動されたときに、レプリカでクラッシュするバグを修正しました。
- インスタンスのローテーションされた SSL 証明書が、インスタンスの再起動なしに取得されなかったバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.2.1.R3 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: 3.4.1
- SPARQL バージョン: 1.1

エンジンリリース 1.0.2.1.R3 へのアップグレードパス

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

ただし、このリリースへの自動更新はサポートされていません。

このリリースへのアップグレード

Amazon Neptune 1.0.2.1.R3 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.2.1 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.2.1 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.0.2.1.R2 (2019-11-25)

このエンジンリリースで修正された不具合

- ラウンドロビン以外の副次トラバーサルと `path()` 以外の副次トラバーサルを使用するすべての `project().by()` クエリに影響するバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.2.1.R2 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.1`
- SPARQL バージョン: `1.1`

エンジンリリース 1.0.2.1.R2 へのアップグレードパス

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

ただし、このリリースへの自動更新はサポートされていません。

このリリースへのアップグレード

Amazon Neptune 1.0.2.1.R2 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.2.1 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.2.1 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のような エラーが発生する可能性があります。

```
  We're sorry, your request to modify DB cluster (cluster identifier) has
failed.
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.0.2.0 (2019-11-08)

重要: このバージョンは非推奨になりました。

2020 年 5 月 19 日以降、このエンジンバージョンを使用する新しいインスタンスは作成されません。

このエンジンのバージョンはバージョン 1.0.2.1 に置き換えられました。このバージョンには、すべてのバグ修正に加え、フルテキスト検索の統合、OSGP インデックスのサポート、AWS リージョン間でのデータベーススナップショットクラスターコピーなどの追加機能が含まれています。

2020 年 6 月 1 日以降、 は、次のメンテナンスウィンドウに、このエンジンのバージョンを実行しているクラスターを バージョン 1.0.2.1 の最新のパッチに自動的にアップグレードします。ここで説明しているように、その前に手動でアップグレードできます。

アップグレードに問題がある場合は、AWS サポートまたは AWS 開発者フォーラムを通じてお問い合わせください。

このリリースの後続のパッチリリース

- リリース： 1.0.2.0.R3 (2020-05-05)
- リリース： 1.0.2.0.R2 (2019-11-21)

このエンジンリリースの新機能

このリリースでは、メンテナンスアップデートに加えて、一度に複数のエンジンバージョンをサポートする新機能が追加されています (「Amazon Neptune DB クラスターのメンテナンス」を参照)。

その結果、エンジンリリースの番号付けが変更されました (「エンジンリリース 1.3.0.0 より前のバージョン番号」を参照)。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.2.0 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.1`
- SPARQL バージョン: `1.1`

エンジンリリース 1.0.2.0 へのアップグレードパス

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

このリリースに自動的にアップグレードされることはありません。

このリリースへのアップグレード

Amazon Neptune 1.0.2.0 が公開されました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.2.0 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.2.0 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.0.2.0.R3 (2020-05-05)

重要: このバージョンは非推奨になりました。

2020 年 5 月 19 日以降、このエンジンバージョンを使用する新しいインスタンスは作成されません。

このエンジンのバージョンはバージョン 1.0.2.1 に置き換えられました。このバージョンには、すべてのバグ修正に加え、フルテキスト検索の統合、OSGP インデックスのサポート、AWS リージョン間でのデータベーススナップショットクラスターコピーなどの追加機能が含まれています。

2020 年 6 月 1 日以降、 は、次のメンテナンスウィンドウに、このエンジンのバージョンを実行しているクラスターを バージョン 1.0.2.1 の最新のパッチに自動的にアップグレードします。ここで説明しているように、その前に手動でアップグレードできます。

アップグレードに問題がある場合は、AWS サポートまたは AWS 開発者フォーラムを通じてお問い合わせください。

このエンジンリリースで修正された不具合

- `ConcurrentModificationConflictException` と `TransactionException` が汎用 `InternalFailureException` として報告されるバグを修正しました。
- 起動時にサーバーが頻繁に再起動するヘルスチェックの不具合を修正しました。
- 特定の条件下でコミットが機能しないため、レプリカにデータが表示されない不具合を修正しました。
- ロードステータスのシリアル化において、Amazon S3 のアクセス許可の不足によりロードが失敗する不具合を修正しました。
- Gremlin セッションでのリソースリークを修正しました。
- ヘルスチェックにおいて、IAM 認証を管理するコンポーネントの起動時に異常ステータスが非表示になる不具合を修正しました。

- チャンネルを閉じる前に Neptune が WebSocket クローズフレームを送信できない不具合を修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.2.0.R3 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.1`
- SPARQL バージョン: `1.1`

エンジンリリース 1.0.2.0.R3 へのアップグレードパス

エンジンバージョン `1.0.2.0` を実行している場合、クラスターは次のメンテナンス期間中に自動的にこのパッチリリースにアップグレードされます。

以前の Neptune エンジンリリースをこのリリースに手動でアップグレードできます。

このリリースへのアップグレード

Amazon Neptune 1.0.2.0.R3 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.2.0 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
```

```
    --engine-version 1.0.2.0 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のようなエラーが発生する可能性があります。

```
  We're sorry, your request to modify DB cluster (cluster identifier) has
failed.
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.0.2.0.R2 (2019-11-21)

重要: このバージョンは非推奨になりました。

2020 年 5 月 19 日以降、このエンジンバージョンを使用する新しいインスタンスは作成されません。

このエンジンのバージョンはバージョン 1.0.2.1 に置き換えられました。このバージョンには、すべてのバグ修正に加え、フルテキスト検索の統合、OSGP インデックスのサポート、AWS リージョン間でのデータベーススナップショットクラスターコピーなどの追加機能が含まれています。

2020 年 6 月 1 日以降、 は、次のメンテナンスウィンドウに、このエンジンのバージョンを実行しているクラスターを バージョン 1.0.2.1 の最新のパッチに自動的にアップグレードします。ここで説明しているように、その前に手動でアップグレードできます。

アップグレードに問題がある場合は、AWS サポートまたは AWS 開発者フォーラムを通じてお問い合わせください。

このエンジンリリースで修正された不具合

- サーバーでのダーティページのキャッシュ戦略を改善し、サーバーがメモリ不足状態になったときの `FreeableMemory` の回復を高速化しました。
- サーバーで多数のロードステータスやロード開始要求を同時に処理する場合に、競合状態やクラッシュを起こす可能性があるバグを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.2.0.R2 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin バージョン: `3.4.1`
- SPARQL バージョン: `1.1`

エンジンリリース 1.0.2.0.R2 へのアップグレードパス

以前の Neptune エンジンリリースを、このリリースに手動でアップグレードできます。

ただし、このリリースへの自動更新はサポートされていません。

このリリースへのアップグレード

Amazon Neptune 1.0.2.0.R2 が一般に利用可能になりました。

DB クラスターで、このリリースへのアップグレードパスがあるエンジンバージョンを実行している場合は、今すぐアップグレードできます。対象となるクラスターをアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドは、適格なクラスターをただちにアップグレードします。

Linux、OS X、Unix の場合:

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.2.0 \
    --apply-immediately
```

Windows の場合:

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.2.0 ^
    --apply-immediately
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。

アップグレードの前に必ずテストする

新しいメジャーまたはマイナーバージョンの Neptune エンジンがリリースされたら、アップグレードする前に、まず最初に Neptune アプリケーションをテストしてください。マイナーアップグレードでも、コードに影響する新しい機能や動作が導入される可能性があります。

まず、現在のバージョンのリリースノートページと対象バージョンのリリースノートページを比較して、クエリ言語のバージョンに変更があるか、その他の重大な変更がないかを確認します。

本番 DB クラスターをアップグレードする前に新しいバージョンをテストする最善の方法は、本番クラスターをクローンして、クローンで新しいエンジンバージョンを実行することです。その後、本番 DB クラスターに影響を与えずに、クローンに対してクエリを実行できます。

アップグレードの前に必ずスナップショットを手動で作成してください

アップグレードの前に必ず DB クラスターの手動スナップショットを作成することを強く推奨します。自動スナップショットを作成しても短期的な保護しか得られませんが、手動スナップショットは明示的に削除するまで使用できます。

場合によっては、Neptune がアップグレードプロセスの一環として手動スナップショットを作成することもありますが、これを頼りにすべきではなく、どのような場合でも独自の手動スナップショットを作成する必要があります。

DB クラスターをアップグレード前の状態に戻す必要がないことが確実な場合は、自分で作成した手動スナップショットと、Neptune が作成した手動スナップショットを明示的に削除できます。Neptune が手動スナップショットを作成する場合、その名前は `preupgrade` で始まり、その後に DB クラスターの名前、ソースエンジンのバージョン、ターゲットエンジンのバージョン、および日付が続きます。

> **Note**
>
> 保留中のアクションの処理中にアップグレードを試みた場合、次のような エラーが発生する可能性があります。

```
  We're sorry, your request to modify DB cluster (cluster identifier) has
failed.
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

このエラーが発生した場合は、保留中のアクションが終了するのを待つか、すぐにメンテナンスウィンドウをトリガーして、前回のアップグレードを完了させます。

お使いのエンジンバージョンのアップグレードの詳細については、Amazon Neptune DB クラスターのメンテナンス を参照してください。ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

Amazon Neptune エンジンバージョン 1.0.1.2 (2020-06-10)

重要: このバージョンは非推奨になりました。

2021 年 4 月 27 日以降、このエンジンバージョンを使用する新しいインスタンスは作成されません。

このエンジンリリースの改良点

- Neptune は、存在しないプロパティ、頂点、エッジにアクセスしようとした場合に `IllegalArgumentException` を発生させるようになりました。以前は、Neptune ではその状況で `UnsupportedOperationException` が発生しました。

  たとえば、存在しない頂点を参照するエッジを追加しようとすると、`IllegalArgumentException` が発生します。

このエンジンリリースで修正された不具合

- 2 つの `value->id` マッピングを同時に挿入すると、ディクショナリとユーザートランザクションのコミットが機能しない不具合を修正しました。
- ロードステータスのシリアル化の不具合を修正しました。
- インスタンスの作成を遅らせる、サーバー起動時の確率的な障害を修正しました。
- カーソルリークを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.1.2 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin のバージョン: 3.4.1
- SPARQL のバージョン: 1.1

Amazon Neptune エンジンバージョン 1.0.1.1 (2020-06-26)

重要: このバージョンは非推奨になりました。

2021 年 4 月 27 日以降、このエンジンバージョンを使用する新しいインスタンスは作成されません。

このエンジンリリースで修正された不具合

- 同時挿入時にコミットが順不同になる不具合を修正しました。
- ロードステータスのシリアル化の不具合を修正しました。
- インスタンスの作成を遅らせる、サーバー起動時の確率的な障害を修正しました。
- メモリリークを修正しました。

このリリースでサポートされるクエリ言語バージョン

DB クラスターをバージョン 1.0.1.1 にアップグレードする前に、プロジェクトが次のクエリ言語バージョンと互換性があることを確認してください。

- Gremlin のバージョン: 3.3.2
- SPARQL のバージョン: 1.1

Amazon Neptune エンジンバージョン 1.0.1.0 (2019-07-02)

重要: このエンジンバージョンは非推奨になりました

2021 年 4 月 27 日以降、このエンジンバージョンを使用する新しいインスタンスは作成されません。

Amazon Neptune エンジンの更新 (2019-10-31)

バージョン: 1.0.1.0.200502.0

重要: このエンジンバージョンは非推奨になりました

2021年 4 月 27 日以降、このエンジンバージョンを使用する新しいインスタンスは作成されません。

このエンジンリリースで修正された不具合

- クライアントが `traversal().withRemote(...)` を使用して (つまり GLV バイトコードを使用して) Neptune に接続するときの `tree()` ステップの応答のシリアル化における Gremlin バグが修正されました。

  このリリースでは、`traversal().withRemote(...)` を使用して Neptune に接続したクライアントが、`tree()` ステップを含む Gremlin クエリに対して無効なレスポンスを受信する問題に対応しました。
- 競合状態が原因でクエリ終了プロセスがハングし、クエリがタイムアウトする `DELETE WHERE LIMIT` クエリの SPARQL バグが修正されました。

Amazon Neptune エンジンの更新 (2019-10-15)

バージョン: 1.0.1.0.200463.0

重要: このエンジンバージョンは非推奨になりました

2021年 4 月 27 日以降、このエンジンバージョンを使用する新しいインスタンスは作成されません。

このエンジンリリースの新機能

- Gremlin の説明/プロファイル機能を追加しました (「Gremlin を使用して Neptune クエリ実行を分析する`explain`」を参照)。
- 1 つのトランザクションで複数の Gremlin トラバーサルを実行できるように Gremlin スクリプトベースのセッションのサポート が追加されました。
- Neptune に SPARQL フェデレーティッドクエリ拡張のサポートを追加しました (SPARQL 1.1 Federated Query および `SERVICE` 拡張を使用した Neptune での SPARQL フェデレーティッドクエリ を参照)。
- HTTP URL パラメータまたは SPARQL `queryId` クエリヒントを使用して、Gremlin または SPARQL クエリに独自の `queryId` を挿入する機能を追加しました (「Neptune Gremlin または SPARQL クエリにカスタム ID を挿入する」を参照)。
- Neptune に ラボモード 機能を追加しました。この機能を使用すると、本番環境ではまだ使用できない機能を試すことができます。

- データベースに加えられたすべての変更を 1 週間保持するストリームに確実に記録する次の Neptune Streams 機能を追加しました。この機能は、ラボモードでのみ使用できます。
- 同時トランザクションの正式なセマンティクスを更新しました (「Neptune でのトランザクションセマンティクス」を参照)。この機能では、同時実行に関する業界標準の保証を提供します。

  デフォルトでは、これらのトランザクションセマンティクスは有効になっています。シナリオによっては、この機能によって現在のロード動作が変更され、ロードパフォーマンスが低下することがあります。DB Cluster neptune_lab_mode パラメータを使用して、パラメータ値に ReadWriteConflictDetection=disabled を含めることで、前のセマンティクスに戻すことができます。

このエンジンリリースの改良点

- エンジンが使用している TinkerPop のバージョンと SPARQL のバージョンを報告することで、インスタンスのステータス API が改善されました。
- Gremlin サブグラフ演算子のパフォーマンスが改善されました。
- Gremlin レスポンスのシリアル化のパフォーマンスが改善されました。
- Gremlin Union ステップのパフォーマンスが改善されました。
- シンプルな SPARQL クエリのレイテンシーが改善されました。

このエンジンリリースで修正された不具合

- 内部エラーとしてタイムアウトが誤って返される Gremlin のバグを修正しました。
- 一部の変数セットに対する ORDER BY によって内部サーバーエラーが発生する SPARQL バグを修正しました。

Amazon Neptune エンジンの更新 (2019-09-19)

重要: このエンジンバージョンは非推奨になりました

2021年 4 月 27 日以降、このエンジンバージョンを使用する新しいインスタンスは作成されません。

バージョン: 1.0.1.0.200457.0

Amazon Neptune 1.0.1.0.200457.0が一般展開されています。そのリージョンでエンジンの更新が完了したら、新しい Neptune DB クラスター (スナップショットから復元されたものも含む) はすべて、Neptune1.0.1.0.200457.0 で作成されるようになります。

既存のクラスターをすぐにこのリリースにアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドを使用して、DB クラスターをアップグレードできます。

```
aws neptune apply-pending-maintenance-action \
    --apply-action system-update \
    --opt-in-type immediate \
    --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。メンテナンスウィンドウの設定は、Neptune コンソール で表示または変更できます。

ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

このエンジンリリースで修正された不具合

- 問題の原因となった接続述語処理のパフォーマンス改善を削除することにより、以前のエンジンリリース (1.0.1.0.200369.0) で発生した Gremlin の正確性の問題を修正しました。
- `DISTINCT` および `OPTIONAL` に含まれている単一のパターンを使用して `InternalServerError` を生成するクエリの原因となった SPARQL バグを修正しました。

Amazon Neptune エンジンの更新 (2019-08-13)

重要: このバージョンは非推奨になりました。

2021 年 4 月 27 日以降、このエンジンバージョンを使用する新しいインスタンスは作成されません。

このエンジンリリースの新機能

- Neptune ローダーコマンド の `parallelism` パラメータに `OVERSUBSCRIBE` オプションを追加します。これにより、Neptune バルクローダーは使用可能なすべてのスレッドとリソースを使用するようになります。

このエンジンリリースの改良点

- 単純な論理 OR 式を含む SPARQL フィルターのパフォーマンスが向上しました。
- 接続述語の処理における Gremlin のパフォーマンスが改善されました。

このエンジンリリースで修正された不具合

- `xsd:duration` から `xsd:date` を減算できない SPARQL のバグを修正しました。
- `UNION` が存在する場合に、静的インライン化によって不完全な結果が発生する SPARQL のバグを修正しました。
- クエリキャンセルの SPARQL のバグを修正しました。
- タイプの昇格中にオーバーフローが発生する Gremlin のバグを修正しました。
- `addE().from().to()` ステップでの頂点要素の処理における Gremlin のバグを修正しました。
- 単一濃度の挿入における NaN の倍精度と浮動小数点数の処理を含む Gremlin のバグ (エンジンバージョン 1.0.1.0.200366.0、2019-07-26 リリース) を修正しました。
- プロパティベースの検索を含むクエリプランの生成のバグを修正しました。

Amazon Neptune エンジンの更新 (2019-07-26)

バージョン: 1.0.1.0.200366.0

重要: このエンジンバージョンは非推奨になりました

2021年 4 月 27 日以降、このエンジンバージョンを使用する新しいインスタンスは作成されません。

このエンジンリリースの新機能

- TinkerPop 3.4.1 にアップグレードしました (TinkerPop アップグレード情報 および TinkerPop 3.4.1 変更ログを参照)。

Neptune のお客様の場合、これらの変更により、次のような新機能や改善がもたらされます。

- GraphBinary がシリアル化形式として使用可能になりました。
- TinkerPop Java ドライバーでメモリリークを引き起こすキープアライブバグが修正され、回避策は不要になりました。

ただし、場合によっては、Neptune の既存の Gremlin コードに影響する可能性があります。例:

- valueMap() は、Map<String,Object> の代わりに Map<Object,Object> を返すようになりました。
- within() ステップの不整合な動作が修正され、他のステップと一貫して動作するようにしました。以前は、比較が機能するために型が一致する必要がありました。現在では、さまざまなタイプの数を正確に比較できるようになりました。たとえば、33 は、以前とは異なり 33L と同じように比較するようになりました。
- ReducingBarrierStep のバグが修正され、出力に使用できる要素がない場合、値が返されないようになりました。
- select() スコープの順序が変更されました (順序は maps 、side-effects、paths になりました)。これにより、side-effects と select を、select と同じ side-effects のキー名と組み合わせた、まれなクエリの結果が変更されます。
- bulkSet() は GraphSON プロトコルの一部になりました。toBulkSet() で終わるクエリは、古いクライアントでは機能しません。
- Submit() ステップの 1 つのパラメーター化が 3.4 クライアントから削除されました。

TinkerPop 3.4 で導入された他の多くの変更は、Neptune の現在の動作には影響しません。たとえば、Gremlin io() は Traversal にステップとして追加され、Graph では非推奨となりましたが、Neptune では有効になっていません 。

- Gremlin 用の一括ローダーに、プロパティグラフデータをロードするための、単一の頂点濃度プロパティのサポートが追加されました。
- 一括ローダーの単一の頂点濃度プロパティの既存の値を上書きするオプションを追加しました。
- Gremlin クエリのステータスを取得する機能と、Gremlin クエリをキャンセルする機能が追加されました。
- SPARQL クエリタイムアウトのクエリヒントを追加しました。
- ステータス API でインスタンスのロールを確認する機能を追加しました (インスタンスのステータス 参照)。
- データベースのクローン作成のサポートを追加しました (「Neptune のデータベースのクローン化」を参照)。

このエンジンリリースの改良点

- FROM 句のグラフ変数が表示されるように SPARQL クエリの説明を修正しました。
- フィルタ、equal フィルタ、VALUES 句、および範囲カウントにおける SPARQL のパフォーマンスが向上しました。
- Gremlin ステップ順序のパフォーマンスが向上しました。
- Gremlin `.repeat.dedup` トラバーサルのパフォーマンスが向上しました。
- Gremlin の `valueMap()` と `path().by()` トラバーサルのパフォーマンスが向上しました。

このエンジンリリースで修正された不具合

- 名前付きグラフの操作を含む SPARQL プロパティパスに関する複数の問題を修正しました。
- SPARQL CONSTRUCT クエリでメモリの問題が生じる問題を修正しました。
- RDF Turtle パーサーとローカル名の問題を修正しました。
- ユーザーに表示されるエラーメッセージを修正するための問題を修正しました。
- Gremlin `repeat()...drop()` トラバーサルに関する問題を修正しました。
- Gremlin `drop()` ステップに関する問題を修正しました。
- Gremlin ラベルフィルタに関する問題を修正しました。
- Gremlin クエリタイムアウトに関する問題を修正しました。

Amazon Neptune エンジンの更新 (2019-07-02)

重要: このエンジンバージョンは非推奨になりました

2021年 4 月 27 日以降、このエンジンバージョンを使用する新しいインスタンスは作成されません。

このエンジンリリースで修正された不具合

- プロパティ名と値がバインドされた特定のパターンが最適化されないバグを修正しました。

以前の Neptune エンジンリリース

トピック

- Amazon Neptune エンジンの更新 (2019-06-12)

- Amazon Neptune エンジンの更新 (2019-05-01)
- Amazon Neptune エンジンの更新 (2019-01-21)
- Amazon Neptune エンジンの更新 (2018-11-19)
- Amazon Neptune エンジンの更新 (2018-11-08)
- Amazon Neptune エンジンの更新 (2018-10-29)
- Amazon Neptune エンジンの更新 (2018-09-06)
- Amazon Neptune エンジンの更新 (2018-07-24)
- Amazon Neptune エンジンの更新 (2018-06-22)

Amazon Neptune エンジンの更新 (2019-06-12)

バージョン: 1.0.1.0.200310.0

Amazon Neptune 1.0.1.0.200310.0 が一般に入手可能です。そのリージョンでエンジンの更新が完了したら、新しい Neptune DB クラスター (スナップショットから復元されたものも含む) はすべて、Neptune 1.0.1.0.200310.0 で作成されるようになります。

既存のクラスターをすぐにこのリリースにアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドを使用して、DB クラスターをこのリリースにすぐにアップグレードできます。

```
aws neptune apply-pending-maintenance-action \
    --apply-action system-update \
    --opt-in-type immediate \
    --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

Neptune DB クラスターは、システムのメンテナンス期間中に、エンジンリリース 1.0.1.0.200310.0 に自動的にアップグレードされます。更新が適用されるタイミングは、リージョンや DB クラスターのメンテナンスウィンドウの設定、および更新のタイプによって異なります。

> **Note**
>
> インスタンスメンテナンスウィンドウは、エンジンの更新には適用されません。

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分

のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。メンテナンスウィンドウの設定は、Neptune コンソール で表示または変更できます。

ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

改良点

- エッジの同時挿入や削除が原因によって、同じ ID を持つエッジが複数になる可能性があるというバグを修正しました。
- マイナー修正と機能向上。

Amazon Neptune エンジンの更新 (2019-05-01)

バージョン: 1.0.1.0.200296.0

Amazon Neptune 1.0.1.0.200296.0 が一般展開されています。そのリージョンでエンジンの更新が完了したら、新しい Neptune DB クラスター (スナップショットから復元されたものも含む) はすべて、Neptune 1.0.1.0.200296.0 で作成されるようになります。

既存のクラスターをすぐにこのリリースにアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドを使用して、DB クラスターをこのリリースへとすぐにアップグレードできます。

```
aws neptune apply-pending-maintenance-action \
    --apply-action system-update \
    --opt-in-type immediate \
    --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

Neptune DB クラスターは、システムのメンテナンス期間中に、エンジンリリース 1.0.1.0.200296.0 に自動的にアップグレードされます。更新が適用されるタイミングは、リージョンや DB クラスターのメンテナンスウィンドウの設定、および更新のタイプによって異なります。

> **Note**
>
> インスタンスメンテナンスウィンドウは、エンジンの更新には適用されません。

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分

のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。メンテナンスウィンドウの設定は、Neptune コンソール で表示または変更できます。

ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

改良点

- クエリプランの表示および必要に応じて最適化の手順を実行するために、Neptune SPARQL クエリに新しい `explain` 機能が追加されました。詳細については、SPARQL `explain`を参照してください。
- SPARQL のパフォーマンスおよびレポートがさまざまな点で改善されました。
- Gremlin のパフォーマンスと動作がさまざまな点で改善されました。
- 実行時間の長い `drop( )` クエリのタイムアウトが改善されました。
- `otherV( )` のクエリのパフォーマンスが改善されました。
- DB クラスターあるいはインスタンスの Neptune ヘルスステータスのクエリ時に、返される情報に 2 つのフィールドが追加されました。エンジンバージョン番号とクラスターあるいはインスタンスの開始時間です。「インスタンスのステータス」を参照してください。
- Neptune ローダー `Get-Status` API が、ロードジョブがいつ開始されたかを記録する `startTime` フィールドを返すようになりました。
- ローダーコマンドでは、ローダーが使用するスレッドの数を制限できる、オプションの `parallelism` パラメータが使用されるようになりました。

Amazon Neptune エンジンの更新 (2019-01-21)

バージョン: 1.0.1.0.200267.0

Amazon Neptune 1.0.1.0.200267.0 が一般展開されています。そのリージョンでエンジンの更新が完了したら、新しい Neptune DB クラスター (スナップショットから復元されたものも含む) はすべて、Neptune 1.0.1.0.200267.0 で作成されるようになります。

既存のクラスターをすぐにこのリリースにアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドを使用して、DB クラスターをこのリリースへすぐにアップグレードできます。

```
aws neptune apply-pending-maintenance-action \
    --apply-action system-update \
```

```
    --opt-in-type immediate \
    --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

Neptune DB クラスターは、システムのメンテナンス期間中に、エンジンリリース 1.0.1.0.200267.0 に自動的にアップグレードされます。更新が適用されるタイミングは、リージョンや DB クラスターのメンテナンスウィンドウの設定、および更新のタイプによって異なります。

> **Note**
>
> インスタンスメンテナンスウィンドウは、エンジンの更新には適用されません。

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。メンテナンスウィンドウの設定は、Neptune コンソール で表示または変更できます。

ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

改良点

- Neptune は競合が解決するまで (指定されたクエリタイムアウトの範囲内で) 長く待機します。これにより、クライアントが処理する必要がある同時変更の例外数は減少します (クエリのエラー を参照)。
- Gremlin 基数の使用が原因でエンジンが再起動される問題を修正しました。
- `emit.times` 繰り返しクエリの Gremlin パフォーマンスが向上しました。
- `repeat.until` が `.emit` ソリューションを許可していた Gremlin の問題を修正しました。
- Gremlin のエラー処理を強化しました。

Amazon Neptune エンジンの更新 (2018-11-19)

バージョン: 1.0.1.0.200264.0

Amazon Neptune 1.0.1.0.200264.0 が一般展開されています。そのリージョンでエンジンの更新が完了したら、新しい Neptune DB クラスター (スナップショットから復元されたものも含む) はすべて、Neptune 1.0.1.0.200264.0 で作成されるようになります。

既存のクラスターをすぐにこのリリースにアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドを使用して、DB クラスターをこのリリースにすぐにアップグレードできます。

```
aws neptune apply-pending-maintenance-action \
    --apply-action system-update \
    --opt-in-type immediate \
    --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

Neptune DB クラスターは、システムのメンテナンス期間中に、エンジンリリース 1.0.1.0.200264.0 に自動的にアップグレードされます。更新が適用されるタイミングは、リージョンや DB クラスターのメンテナンスウィンドウの設定、および更新のタイプによって異なります。

> **Note**
>
> インスタンスメンテナンスウィンドウは、エンジンの更新には適用されません。

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。メンテナンスウィンドウの設定は、Neptune コンソール で表示または変更できます。

ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

改良点

- the section called “クエリヒント” のサポートが追加されました。
- IAM 認証に関するエラーメッセージが改善されました。詳細については、「the section called “IAM エラー”」を参照してください。
- 多数の述語を使用して SPARQL クエリのパフォーマンスが改善されました。
- SPARQL プロパティパスのパフォーマンスが改善されました。
- `addV()`、`addE()`、および `property()` ステップと使用されたときに、`fold().coalesce(unfold(), …)` パターンなど、条件付きミューテーションの Gremlin のパフォーマンスが改善されました。
- `by()` および `sack()` モジュレーションの Gremlin のパフォーマンスが改善されました。
- `group()` および `groupCount()` ステップの Gremlin のパフォーマンスが改善されました。

- `store()`、`sideEffect()`、および `cap().unfold()` ステップの Gremlin のパフォーマンスが改善されました。
- Gremlin の単一濃度プロパティの制約に対するサポートが改善されました。
  - 単一濃度プロパティとしてマークされたエッジプロパティおよび頂点プロパティに対する単一濃度の適用が改善されました。
  - Neptune のロードジョブ中に追加のプロパティ値が既存のエッジプロパティに指定されている場合に、エラーが導入されました。

Amazon Neptune エンジンの更新 (2018-11-08)

バージョン: 1.0.1.0.200258.0

Amazon Neptune 1.0.1.0.200258.0 が一般展開されています。そのリージョンでエンジンの更新が完了したら、新しい Neptune DB クラスター (スナップショットから復元されたものも含む) はすべて、Neptune 1.0.1.0.200258.0 で作成されるようになります。

既存のクラスターをすぐにこのリリースにアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドを使用して、DB クラスターをこのリリースまですぐにアップグレードできます。

```
aws neptune apply-pending-maintenance-action \
    --apply-action system-update \
    --opt-in-type immediate \
    --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

Neptune DB クラスターは、システムのメンテナンス期間中に、エンジンリリース 1.0.1.0.200258.0 に自動的にアップグレードされます。更新が適用されるタイミングは、リージョンや DB クラスターのメンテナンスウィンドウの設定、および更新のタイプによって異なります。

> **Note**
>
> インスタンスメンテナンスウィンドウは、エンジンの更新には適用されません。

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。メンテナンスウィンドウの設定は、Neptune コンソール で表示または変更できます。

ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

改良点

- SPARQL クエリヒント のサポートが追加されました。
- SPARQL FILTER (NOT) Exists クエリのパフォーマンスが改善されました。
- SPARQL DESCRIBE クエリのパフォーマンスが改善されました。
- Gremlin のパターンまでの繰り返しのパフォーマンスが改善されました。
- Gremlin のエッジ追加のパフォーマンスが改善されました。
- 場合によって SPARQL Update DELETE クエリで障害が発生する問題を修正しました。
- Gremlin WebSocket サーバーを使用したタイムアウトの処理に関する問題を修正しました。

Amazon Neptune エンジンの更新 (2018-10-29)

バージョン: 1.0.1.0.200255.0

Amazon Neptune 1.0.1.0.200255.0 が一般展開されています。そのリージョンでエンジンの更新が完了したら、新しい Neptune DB クラスター (スナップショットから復元されたものも含む) はすべて、Neptune 1.0.1.0.200255.0 で作成されるようになります。

既存のクラスターをすぐにこのリリースにアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドを使用して、DB クラスターをこのリリースにすぐにアップグレードできます。

```
aws neptune apply-pending-maintenance-action \
    --apply-action system-update \
    --opt-in-type immediate \
    --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

Neptune DB クラスターは、システムのメンテナンス期間中に、エンジンリリース 1.0.1.0.200255.0 に自動的にアップグレードされます。更新が適用されるタイミングは、リージョンや DB クラスターのメンテナンスウィンドウの設定、および更新のタイプによって異なります。

> **Note**
>
> インスタンスメンテナンスウィンドウは、エンジンの更新には適用されません。

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。メンテナンスウィンドウの設定は、Neptune コンソール で表示または変更できます。

ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

改良点

- IAM 認証情報が監査ログに追加されました。
- IAM ロールおよびインスタンスプロファイルを使用する一時的な認証情報のサポートが追加されました。
- アクセス許可が取り消されたとき、または IAM ユーザーやロールが削除されたときに、IAM 認証の WebSocket 接続の終了が追加されました。
- WebSocket 接続の最大数がインスタンスあたり 60,000 に制限されました。
- 小さいインスタンスタイプの一括ロードパフォーマンスが改善されました。
- Gremlin の `and()`、`or()`、`not()`、`drop()` 演算子を含むクエリのパフォーマンスが改善されました。
- NTriples パーサーが、空白文字が含まれている URI などの無効な URI を拒否するようになりました。

Amazon Neptune エンジンの更新 (2018-09-06)

バージョン: 1.0.1.0.200237.0

Amazon Neptune 1.0.1.0.200237.0 が一般展開されています。そのリージョンでエンジンの更新が完了したら、新しい Neptune DB クラスター (スナップショットから復元されたものも含む) はすべて、 Neptune 1.0.1.0.200237.0 で作成されるようになります。

既存のクラスターをすぐにこのリリースにアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドを使用して、DB クラスターをこのリリースにすぐにアップグレードできます。

```
aws neptune apply-pending-maintenance-action \
    --apply-action system-update \
    --opt-in-type immediate \
```

```
    --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

Neptune DB クラスターは、システムのメンテナンスウィンドウズに、エンジンリリース 1.0.1.0.200237.0 に自動的にアップグレードされます。更新が適用されるタイミングは、リージョンや DB クラスターのメンテナンスウィンドウの設定、および更新のタイプによって異なります。

> **Note**
>
> インスタンスメンテナンスウィンドウは、エンジンの更新には適用されません。

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。メンテナンスウィンドウの設定は、Neptune コンソール で表示または変更できます。

ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

改良点

- 一部の SPARQL COUNT(DISTINCT) クエリが失敗する問題が修正されました。
- DISTINCT 句がある COUNT、SUM、MIN クエリがメモリ不足になることがあった問題が修正されました。
- BLOB タイプデータが Neptune ローダージョブの失敗の原因となる問題が修正されました。
- 重複した挿入がトランザクションの失敗の原因となる問題が修正されました。
- DROP ALL クエリがキャンセルできない問題が修正されました。
- Gremlin クライアントが断続的にフリーズする問題が修正されました。
- 150M より大きいペイロードのすべてのエラーコードを HTTP 400 に更新しました。
- 単一トリプルパターンの COUNT() クエリのパフォーマンスと正確度が改善されました。
- BIND 句の SPARQL UNION クエリのパフォーマンスが改善されました。

Amazon Neptune エンジンの更新 (2018-07-24)

バージョン: 1.0.1.0.200236.0

Amazon Neptune 1.0.1.0.200236.0 が一般に入手可能です。そのリージョンでエンジンの更新が完了したら、新しい Neptune DB クラスター (スナップショットから復元されたものも含む) はすべて、Neptune 1.0.1.0.200236.0 で作成されるようになります。

既存のクラスターをすぐにこのリリースにアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドを使用して、DB クラスターをこのリリースにすぐにアップグレードできます。

```
aws neptune apply-pending-maintenance-action \
    --apply-action system-update \
    --opt-in-type immediate \
    --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

Neptune DB クラスターは、システムのメンテナンス期間中に、エンジンリリース 1.0.1.0.200236.0 に自動的にアップグレードされます。更新が適用されるタイミングは、リージョンや DB クラスターのメンテナンスウィンドウの設定、および更新のタイプによって異なります。

> **Note**
>
> インスタンスメンテナンスウィンドウは、エンジンの更新には適用されません。

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。メンテナンスウィンドウの設定は、Neptune コンソール で表示または変更できます。

ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

改良点

- `xsd:string` datatype. xsd:string の更新された SPARQL シリアル化は、JSON シリアル化が他の出力形式と一致するようになったため、これに含まれなくなりました。
- SPARQL 1.1 UPDATE および SPARQL 1.1 Query などのすべての SPARQL データローダー形式で、`xsd:double`/`xsd:float` infinity. `-INF`、`NaN`、および `INF` 値の固定処理が適切に認識され処理されるようになりました。
- 空の文字列値のある Gremlin クエリが予期せず失敗する問題を修正しました。

- 空のグラフで、Gremlin `aggregate()` および `cap()` が予期せずに失敗する問題を修正しました。
- 基数の指定が無効な場合 (例: `.property(set,id,'10')` および `.property(single,id,'10')`)、Gremlin に対して誤ったエラーレスポンスが返される問題を修正しました。
- InternalFailureException として無効な Gremlin 構文が返される問題を修正しました。
- エラーメッセージのスペルを `TimeLimitExceeededException` から `TimeLimitExceededException` に修正しました。
- スクリプトが提供されない場合の SPARQL および GREMLIN エンドポイントのレスポンスを変更しました。
- 多すぎる同時リクエストのエラーメッセージを明確にしました。

Amazon Neptune エンジンの更新 (2018-06-22)

バージョン: 1.0.1.0.200233.0

Amazon Neptune 1.0.1.0.200233.0 が一般展開されています。そのリージョンでエンジンの更新が完了したら、新しい Neptune DB クラスター (スナップショットから復元されたものも含む) はすべて、Neptune 1.0.1.0.200233.0 で作成されるようになります。

既存のクラスターをすぐにこのリリースにアップグレードするには、コンソールの DB クラスターオペレーションまたは SDK を使用します。次の CLI コマンドを使用して、DB クラスターをこのリリースにすぐにアップグレードできます。

```
aws neptune apply-pending-maintenance-action \
    --apply-action system-update \
    --opt-in-type immediate \
    --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

Neptune DB クラスターは、システムのメンテナンス期間中に、エンジンリリース 1.0.1.0.200233.0 に自動的にアップグレードされます。更新が適用されるタイミングは、リージョンや DB クラスターのメンテナンスウィンドウの設定、および更新のタイプによって異なります。

> **Note**
>
> インスタンスメンテナンスウィンドウは、エンジンの更新には適用されません。

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ～ 30 秒から数分のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。メンテナンスウィンドウの設定は、Neptune コンソール で表示または変更できます。

ご質問やご不明点がございましたら、コミュニティフォーラムや AWS プレミアムサポートから AWS サポートチームにお問い合わせください。

改良点

- 大量の一括ロードリクエストが連続して発行されてエラーが発生する問題を修正しました。
- InternalServerError でクエリが失敗する可能性のあるデータ依存の問題を修正しました。以下の例では、該当するクエリのタイプを示しています。

```
g.V("my-id123").as("start").outE("knows").has("edgePropertyKey1",
 P.gt(0)).as("myedge").inV()
               .as("end").select("start", "end", "myedge").by("vertexPropertyKey1")
               .by("vertexPropertyKey1").by("edgePropertyKey1")
```

- Gremlin Java クライアントが長時間実行しているクエリのタイムアウト後に同じ WebSocket 接続を使用してサーバーに接続できない問題を修正しました。
- HTTP を介した Gremlin クエリまたは WebSocket 接続を介した文字列ベースのクエリに含まれるエスケープシーケンスが正しく処理されない問題を修正しました。

Amazon Neptune API の使用の概要

Amazon Neptune 管理 API は、Neptune DB クラスターとインスタンスを作成、管理、削除するための SDK サポートを提供します。一方、Neptune データ API は、グラフへのデータの読み込み、クエリの実行、グラフ内のデータに関する情報の取得、機械学習操作の実行のための SDK サポートを提供します。これらの API はすべての SDK 言語で使用できます。API リクエストに自動的に署名することによって、Neptune のアプリケーションへの統合を大幅に簡素化します。

このページでは、これらの API の使用方法に関する情報を提供します。

対応する Neptune データ API SDK とは異なる名前の IAM アクション

IAM 認証が有効になっているクラスターで Neptune API メソッドを呼び出す場合は、実行するアクションのアクセス許可を提供する呼び出しを行うユーザーまたはロールに IAM ポリシーをアタッチする必要があります。これらのアクセス許可は、対応する IAM アクションを使用してポリシーで設定します。IAM 条件キーを使用して、実行できるアクションを制限することもできます。

ほとんどの IAM アクションは、対応する API メソッドと同じ名前ですが、データ API の一部のメソッドは、複数のメソッドで共有されるため、名前が異なります。以下の表は、データメソッドとそれに対応する IAM アクションの一覧です。

データ API 操作名	対応する IAM	
CancelGremlinQuery (cancel_gremlin_query)	アクション: `neptune-db: CancelQuery`	
CancelLoaderJob (cancel_loader_job)	アクション: `neptune-db:CancelLoaderJob`	
CancelMLDataProcessingJob (cancel_ml_data_processing_job)	アクション: `neptune-db:CancelMLDataProcessingJob`	
CancelMLModelTrainingJob (cancel_ml_model_training_job)	アクション: `neptune-db:CancelMLModelTrainingJob`	

データ API 操作名	対応する IAM
CancelOpenCypherQuery (cancel_open_cypher_query)	アクション: neptune-db: **CancelQuery**
CreateMLEndpoint (create_ml_endpoint)	アクション: neptune-db:CreateMLEndpoint
DeleteMLEndpoint (delete_ml_endpoint)	アクション: neptune-db:DeleteMLEndpoint
DeletePropertygraphStatistics (delete_propertygraph_statistics)	アクション: neptune-db: **DeleteStatistics**
DeleteSparqlStatistics (delete_sparql_statistics)	アクション: neptune-db: **DeleteStatistics**
ExecuteFastReset execute_fast_reset()	アクション: neptune-db: **ResetDatabase**
ExecuteGremlinExplainQuery (execute_gremlin_explain_query)	アクション: • neptune-db: **ReadDataViaQuery** • neptune-db: **WriteDataViaQuery** • neptune-db: **DeleteDataViaQuery** 条件キー: neptune-db:QueryLanguage:Gremlin

データ API 操作名	対応する IAM	
ExecuteGremlinProfileQuery (execute_gremlin_profile_query)	アクション: `neptune-db:` **`ReadDataViaQuery`** 条件キー: `neptune-db:QueryLanguage:Gremlin`	
ExecuteGremlinQuery (execute_gremlin_query)	アクション: • `neptune-db:` **`ReadDataViaQuery`** • `neptune-db:` **`WriteDataViaQuery`** • `neptune-db:` **`DeleteDataViaQuery`** 条件キー: `neptune-db:QueryLanguage:Gremlin`	
ExecuteOpenCypherExplainQuery (execute_open_cypher_explain_query)	アクション: `neptune-db:` **`ReadDataViaQuery`** 条件キー: `neptune-db:QueryLanguage:OpenCypher`	

データ API 操作名	対応する IAM
ExecuteOpenCypherQuery (execute_open_cypher_query)	アクション: • neptune-db: **ReadDataViaQuery** • neptune-db: **WriteDataViaQuery** • neptune-db: **DeleteDataViaQuery** 条件キー: neptune-db:QueryLanguage:OpenCypher
GetEngineStatus (get_engine_status)	アクション: neptune-db:GetEngineStatus
GetGremlinQueryStatus (get_gremlin_query_status)	アクション: neptune-db: **:GetQueryStatus** 条件キー: neptune-db:QueryLanguage:Gremlin
GetLoaderJobStatus (get_loader_job_status)	アクション: neptune-db:GetLoaderJobStatus
GetMLDataProcessingJob (get_ml_data_processing_job)	アクション: neptune-db: **GetMLDataProcessingJobStatus**
GetMLEndpoint (get_ml_endpoint)	アクション: neptune-db: **GetMLEndpointStatus**

データ API 操作名	対応する IAM	
GetMLModelTrainingJob (get_ml_model_training_job)	アクション: neptune-db: **GetMLModelTrainingJobStatus**	
GetMLModelTransformJob (get_ml_model_transform_job)	アクション: neptune-db: **GetMLModelTransformJobStatus**	
GetOpenCypherQueryStatus (get_open_cypher_query_status)	アクション: neptune-db: **:GetQueryStatus** 条件キー: neptune-db:QueryLanguage:OpenCypher	
GetPropertygraphStatistics (get_propertygraph_statistics)	アクション: neptune-db: **GetStatisticsStatus**	
GetPropertygraphStream (get_propertygraph_stream)	アクション: neptune-db: **GetStreamRecords** 条件キー: • neptune-db:QueryLanguage:Gremlin • neptune-db:QueryLanguage:OpenCypher	
GetPropertygraphSummary (get_propertygraph_summary)	アクション: neptune-db: **GetGraphSummary**	
GetRDFGraphSummary (get_rdf_graph_summary)	アクション: neptune-db: **GetGraphSummary**	

データ API 操作名	対応する IAM	
GetSparqlStatistics (get_sparql_statistics)	アクション: neptune-db: **GetStatisticsStatus**	
GetSparqlStream (get_sparql_stream)	アクション: neptune-db: **:GetStreamRecords** 条件キー: neptune-db:QueryLanguage:Sparql	
ListGremlinQueries (list_gremlin_queries)	アクション: neptune-db: **:GetQueryStatus** 条件キー: neptune-db:QueryLanguage:Gremlin	
ListMLEndpoints (list_ml_endpoints)	アクション: neptune-db:ListMLEndpoints	
ListMLModelTrainingJobs (list_ml_model_training_jobs)	アクション: neptune-db:ListMLModelTrainingJobs	
ListMLModel_Transform_Jobs (list_ml_model_transform_jobs)	アクション: neptune-db:ListMLModelTransformJobs	
ListOpenCypherQueries (list_open_cypher_queries)	アクション: neptune-db: **:GetQueryStatus** 条件キー: neptune-db:QueryLanguage:OpenCypher	

データ API 操作名	対応する IAM
ManagePropertygraphStatistics (manage_propertygraph_statistics)	アクション: neptune-db: **ManageStatistics**
ManageSparqlStatistics (manage_sparql_statistics)	アクション: neptune-db: **ManageStatistics**
StartLoaderJob (start_loader_job)	アクション: neptune-db:StartLoaderJob
StartMLModelDataProcessingJob (start_ml_data_processing_job)	アクション: neptune-db:StartMLModelDataProcessingJob
StartMLModelTrainingJob (start_ml_model_training_job)	アクション: neptune-db:StartMLModelTrainingJob
StartMLModelTransformJob (start_ml_model_transform_job)	アクション: neptune-db:StartMLModelTransformJob

Amazon Neptune 管理 API リファレンス

この章では、Neptune DB クラスターを管理および維持するために使用できる Neptune API 操作について説明します。

Neptune は、レプリケーショントポロジに接続されているデータベースサーバーのクラスター上で動作します。そのため、Neptune を管理するには、複数のサーバーへの変更をデプロイし、すべての Neptune レプリカがプライマリサーバーで維持されていることを確認する必要があります。

Neptune では、データの増加に伴い、基本となるストレージを透過的にスケーリングしているため、Neptune の管理に必要なディスクストレージの管理は比較的わずかです。同様に、Neptune では、継続的バックアップが自動的に行われるため、Neptune クラスターでは、バックアップの実行に伴う過度な計画やダウンタイムは必要ありません。

目次

- Neptune グローバルデータベース API
  - CreateGlobalCluster (アクション)
  - DeleteGlobalCluster (アクション)
  - ModifyGlobalCluster (アクション)
  - DescribeGlobalClusters (アクション)
  - FailoverGlobalCluster (アクション)
  - RemoveFromGlobalCluster (アクション)
  - 構造:
  - GlobalCluster (構造)
  - GlobalClusterMember (構造)
- Neptune インスタンス API
  - CreateDBInstance (アクション)
  - DeleteDBInstance (アクション)
  - ModifyDBInstance (アクション)
  - RebootDBInstance (アクション)
  - DescribeDBInstances (アクション)
  - DescribeOrderableDBInstanceOptions (アクション)
  - DescribeValidDBInstanceModifications (アクション)
  - 構造:
  - DBInstance (構造)
  - DBInstanceStatusInfo (構造)
  - OrderableDBInstanceOption (構造)
  - PendingModifiedValues (構造)
  - ValidStorageOptions (構造)
  - ValidDBInstanceModificationsMessage (構造)
- Neptune パラメータ API
  - CopyDBParameterGroup (アクション)
  - CopyDBClusterParameterGroup (アクション)
  - CreateDBParameterGroup (アクション)
  - CreateDBClusterParameterGroup (アクション)

- DeleteDBParameterGroup (アクション)
- DeleteDBClusterParameterGroup (アクション)
- ModifyDBParameterGroup (アクション)
- ModifyDBClusterParameterGroup (アクション)
- ResetDBParameterGroup (アクション)
- ResetDBClusterParameterGroup (アクション)
- DescribeDBParameters (アクション)
- DescribeDBParameterGroups (アクション)
- DescribeDBClusterParameters (アクション)
- DescribeDBClusterParameterGroups (アクション)
- DescribeEngineDefaultParameters (アクション)
- DescribeEngineDefaultClusterParameters (アクション)
- 構造:
- パラメータ (構造)
- DBParameterGroup (構造)
- DBClusterParameterGroup (構造)
- DBParameterGroupStatus (構造)

- Neptune サブネット API
  - CreateDBSubnetGroup (アクション)
  - DeleteDBSubnetGroup (アクション)
  - ModifyDBSubnetGroup (アクション)
  - DescribeDBSubnetGroups (アクション)
  - 構造:
  - サブネット (構造)
  - DBSubnetGroup (構造)
- Neptune スナップショット API
  - CreateDBClusterSnapshot (アクション)
  - DeleteDBClusterSnapshot (アクション)
  - CopyDBClusterSnapshot (アクション)
  - ModifyDBClusterSnapshotAttribute (アクション)

- RestoreDBClusterFromSnapshot (アクション)
- RestoreDBClusterToPointInTime (アクション)
- DescribeDBClusterSnapshots (アクション)
- DescribeDBClusterSnapshotAttributes (アクション)
- 構造:
- DBClusterSnapshot (構造)
- DBClusterSnapshotAttribute (構造)
- DBClusterSnapshotAttributesResult (構造)

- Neptune イベント API
  - CreateEventSubscription (アクション)
  - DeleteEventSubscription (アクション)
  - ModifyEventSubscription (アクション)
  - DescribeEventSubscriptions (アクション)
  - AddSourceIdentifierToSubscription (アクション)
  - RemoveSourceIdentifierFromSubscription (アクション)
  - DescribeEvents (アクション)
  - DescribeEventCategories (アクション)
  - 構造:
  - イベント (構造)
  - EventCategoriesMap (構造)
  - EventSubscription (構造)
- その他の Neptune API
  - AddTagsToResource (アクション)
  - ListTagsForResource (アクション)
  - RemoveTagsFromResource (アクション)
  - ApplyPendingMaintenanceAction (アクション)
  - DescribePendingMaintenanceActions (アクション)
  - DescribeDBEngineVersions (アクション)
  - 構造:
  - DBEngineVersion (構造)

- EngineDefaults (構造)
- PendingMaintenanceAction (構造)
- ResourcePendingMaintenanceActions (構造)
- UpgradeTarget (構造)
- タグ (構造)

- 一般的な Neptune のデータ型
  - アベイラビリティーゾーン (構造)
  - DBSecurityGroupMembership (構造)
  - DomainMembership (構造)
  - DoubleRange (構造)
  - エンドポイント (構造)
  - フィルター (構造)
  - 範囲 (構造)
  - ServerlessV2ScalingConfiguration (構造)
  - ServerlessV2ScalingConfigurationInfo (構造)
  - タイムゾーン (構造)
  - VpcSecurityGroupMembership (構造)
- 個々の API に固有の Neptune 例外
  - AuthorizationAlreadyExistsFault (構造)
  - AuthorizationNotFoundFault (構造)
  - AuthorizationQuotaExceededFault (構造)
  - CertificateNotFoundFault (構造)
  - DBClusterAlreadyExistsFault (構造)
  - DBClusterNotFoundFault (構造)
  - DBClusterParameterGroupNotFoundFault (構造)
  - DBClusterQuotaExceededFault (構造)
  - DBClusterRoleAlreadyExistsFault (構造)
  - DBClusterRoleNotFoundFault (構造)
  - DBClusterRoleQuotaExceededFault (構造)
  - DBClusterSnapshotAlreadyExistsFault (構造)

- DBClusterSnapshotNotFoundFault (構造)
- DBInstanceAlreadyExistsFault (構造)
- DBInstanceNotFoundFault (構造)
- DBLogFileNotFoundFault (構造)
- DBParameterGroupAlreadyExistsFault (構造)
- DBParameterGroupNotFoundFault (構造)
- DBParameterGroupQuotaExceededFault (構造)
- DBSecurityGroupAlreadyExistsFault (構造)
- DBSecurityGroupNotFoundFault (構造)
- DBSecurityGroupNotSupportedFault (構造)
- DBSecurityGroupQuotaExceededFault (構造)
- DBSnapshotAlreadyExistsFault (構造)
- DBSnapshotNotFoundFault (構造)
- DBSubnetGroupAlreadyExistsFault (構造)
- DBSubnetGroupDoesNotCoverEnoughAZs (構造)
- DBSubnetGroupNotAllowedFault (構造)
- DBSubnetGroupNotFoundFault (構造)
- DBSubnetGroupQuotaExceededFault (構造)
- DBSubnetQuotaExceededFault (構造)
- DBUpgradeDependencyFailureFault (構造)
- DomainNotFoundFault (構造)
- EventSubscriptionQuotaExceededFault (構造)
- GlobalClusterAlreadyExistsFault (構造)
- GlobalClusterNotFoundFault (構造)
- GlobalClusterQuotaExceededFault (構造)
- InstanceQuotaExceededFault (構造)
- InsufficientDBClusterCapacityFault (構造)
- InsufficientDBInstanceCapacityFault (構造)
- InsufficientStorageClusterCapacityFault (構造)
- InvalidDBClusterEndpointStateFault (構造)

- InvalidDBClusterSnapshotStateFault (構造)
- InvalidDBClusterStateFault (構造)
- InvalidDBInstanceStateFault (構造)
- InvalidDBParameterGroupStateFault (構造)
- InvalidDBSecurityGroupStateFault (構造)
- InvalidDBSnapshotStateFault (構造)
- InvalidDBSubnetGroupFault (構造)
- InvalidDBSubnetGroupStateFault (構造)
- InvalidDBSubnetStateFault (構造)
- InvalidEventSubscriptionStateFault (構造)
- InvalidGlobalClusterStateFault (構造)
- InvalidOptionGroupStateFault (構造)
- InvalidRestoreFault (構造)
- InvalidSubnet (構造)
- InvalidVPCNetworkStateFault (構造)
- KMSKeyNotAccessibleFault (構造)
- OptionGroupNotFoundFault (構造)
- PointInTimeRestoreNotEnabledFault (構造)
- ProvisionedIopsNotAvailableInAZFault (構造)
- ResourceNotFoundFault (構造)
- SNSInvalidTopicFault (構造)
- SNSNoAuthorizationFault (構造)
- SNSTopicArnNotFoundFault (構造)
- SharedSnapshotQuotaExceededFault (構造)
- SnapshotQuotaExceededFault (構造)
- SourceNotFoundFault (構造)
- StorageQuotaExceededFault (構造)
- StorageTypeNotSupportedFault (構造)
- SubnetAlreadyInUse (構造)
- SubscriptionAlreadyExistFault (構造)

- SubscriptionCategoryNotFoundFault (構造)
- SubscriptionNotFoundFault (構造)

Neptune DB クラスター API

アクション:

- CreateDBCluster (アクション)
- DeleteDBCluster (アクション)
- ModifyDBCluster (アクション)
- StartDBCluster (アクション)
- StopDBCluster (アクション)
- AddRoleToDBCluster (アクション)
- RemoveRoleFromDBCluster (アクション)
- FailoverDBCluster (アクション)
- PromoteReadReplicaDBCluster (アクション)
- DescribeDBClusters (アクション)

構造:

- DBCluster (構造)
- DBClusterMember (構造)
- DBClusterRole (構造)
- CloudwatchLogsExportConfiguration (構造)
- PendingCloudwatchLogsExports (構造)
- ClusterPendingModifiedValues (構造)

CreateDBCluster (アクション)

> この API の AWS CLI 名は `create-db-cluster` です。

新しい Amazon Neptune DB クラスターを作成します。

ReplicationSourceIdentifier パラメータを使用して、別の DB クラスターまたは Amazon Neptune DB インスタンスのリードレプリカとして DB クラスターを作成できます。

CreateDBCluster を直接使用して新しいクラスターを作成する場合、削除保護はデフォルトで無効になっています (コンソールで新しい本番稼働用クラスターを作成する場合、削除保護はデフォルトで有効になっています)。DB クラスターは、DeletionProtection フィールドが false に設定されている場合にのみ削除できます。

リクエスト

- AvailabilityZones (CLI では: --availability-zones) — タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスター内のインスタンスを作成できる EC2 アベイラビリティーゾーンのリスト。

- BackupRetentionPeriod (CLI では: --backup-retention-period) — タイプ integer の IntegerOptional (符号付き 32 ビット整数)。

  自動バックアップを保管する日数。最小値 1 を指定しなければなりません。

  デフォルト: 1

  制約:

  - 1 ~ 35 の値にする必要があります。

- CopyTagsToSnapshot (CLI では: --copy-tags-to-snapshot) — タイプ boolean の BooleanOptional (ブール値 (真または偽) の値)。

  true に設定されている場合、タグは、作成された DB クラスターのいずれかのスナップショットにコピーされます。

- DatabaseName (CLI では: --database-name) — タイプ: string (UTF-8 でエンコードされた文字列) の文字列。

  データベースの名前を、英数字 64 文字以内で入力します。名前を入力しない場合は、Amazon Neptune はここで作成する DB クラスター上にデータベースを作成しません。

- DBClusterIdentifier (CLI では: --db-cluster-identifier) — 必須: string タイプの文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスター識別子。このパラメータは小文字で保存されます。

  制約:

- 1 ~ 63 の文字、数字またはハイフンを使用する必要があります。
- 1 字目は英字である必要があります。
- ハイフンを、文字列の最後に使用したり、2 つ続けて使用したりすることはできません。

例: `my-cluster1`

- DBClusterParameterGroupName (CLI では: `--db-cluster-parameter-group-name`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  このクラスターに関連付ける DB クラスターパラメータグループの名前。この引数を省略すると、デフォルトが使用されます。

  制約:

  - 指定した場合、既存の DBClusterParameterGroup の名前と一致する必要があります。
- DBSubnetGroupName (CLI では: `--db-subnet-group-name`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターに関連付ける DB サブネットグループ。

  制約: 既存の DBSubnetGroup の名前と一致する必要があります。デフォルト値を使用することはできません。

  例: `mySubnetgroup`
- DeletionProtection (CLI では: `--deletion-protection`) — BooleanOptional、タイプ: `boolean` (ブール値 (真または偽))。

  DB クラスターで削除保護が有効になっているかどうかを示す値。削除保護が有効になっている場合、データベースを削除することはできません。デフォルトでは、削除保護は有効です。
- EnableCloudwatchLogsExports (CLI では: `--enable-cloudwatch-logs-exports`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  この DB クラスターが CloudWatch Logs にエクスポートするログタイプのリスト。有効なログタイプは、`audit` (監査ログを公開する場合) と `slowquery` (スロークエリログを公開する場合) です。「Amazon CloudWatch Logs への Neptune ログの発行」を参照してください。
- EnableIAMDatabaseAuthentication (CLI では: `--enable-iam-database-authentication`) — タイプ `boolean` の BooleanOptional (ブール値 (真または偽) の値)。

  `true` に設定されている場合、DB クラスター全体の Amazon Identity and Access Management (IAM) 認証を有効にします (インスタンスレベルでは設定できません) 。

デフォルト: `false`。

- Engine (CLI では: `--engine`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  この DB クラスターに使用されるデータベースエンジンの名前。

  有効な値: `neptune`

- EngineVersion (CLI では: `--engine-version`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  新しい DB クラスターに使用するデータベースエンジンのバージョン。

  例: `1.2.1.0`

- GlobalClusterIdentifier (CLI では: `--global-cluster-identifier`) — GlobalClusterIdentifier、タイプ:`string`(UTF-8でエンコードされた文字列)、1 以上または 255 以下、正規表現 `[A-Za-z][0-9A-Za-z-:._]*` に一致。

  この新しい DB クラスターを追加する必要がある Neptune グローバルデータベースの ID。

- KmsKeyId (CLI では: `--kms-key-id`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  暗号化された DB クラスターの Amazon KMS キー識別子。

  KMS キー識別子は、KMS 暗号化キーの Amazon リソースネーム (ARN) です。新しい DB クラスターの暗号化に使用する KMS 暗号化キーを所有する Amazon アカウントと同じアカウントを使用して DB クラスターを作成する場合、KMS 暗号化キーの ARN の代わりに KMS キーのエイリアスを使用できます。

  `KmsKeyId` で暗号化キーが指定されていない場合:

  - `ReplicationSourceIdentifier` が暗号化されたソースを識別した場合、Amazon Neptune はソースの暗号化に使用された暗号化キーを使用します。それ以外の場合、Amazon Neptune はデフォルトの暗号化キーを使用します。
  - `StorageEncrypted` パラメータが true の場合で `ReplicationSourceIdentifier` が指定されていない場合、Amazon Neptune はデフォルトの暗号化キーを使用します。

  Amazon KMS は、Amazon アカウント用にデフォルトの暗号化キーを作成します。Amazon アカウントには、Amazon のリージョンごとにデフォルトの暗号化キーがあります。

別の Amazon リージョンで暗号化された DB クラスターのリードレプリカを作成する場合は、KmsKeyId を移行先の Amazon リージョンで有効な KMS キー ID に設定する必要があります。このキーはその Amazon リージョンでリードレプリカの暗号化に使用されます。

- Port (CLI では: --port) — タイプ integer の IntegerOptional (符号付き 32 ビット整数)。

  DB クラスターのインスタンスが接続を受け付けることができるポート番号。

  デフォルト: 8182

- PreferredBackupWindow (CLI では: --preferred-backup-window) — タイプ: string (UTF-8 でエンコードされた文字列) の文字列。

  BackupRetentionPeriod パラメータを使用して自動バックアップが有効になっている場合に、自動バックアップが作成される毎日の時間範囲。

  デフォルトのバックアップウィンドウは 30 分のウィンドウで、Amazon リージョンごとに 8 時間の時間ブロックからランダムに選択されます。使用可能な時間帯を表示するには、「Amazon Neptune ユーザーガイド」の「Neptune メンテナンスウィンドウ」を参照してください。

  制約:

  - hh24:mi-hh24:mi の形式である必要があります。
  - 時間は協定世界時 (UTC) である必要があります。
  - 必要なメンテナンス期間と競合してはいけません。
  - 少なくとも 30 分以上必要です。

- PreferredMaintenanceWindow (CLI では: --preferred-maintenance-window) — タイプ: string (UTF-8 でエンコードされた文字列) の文字列。

  週 1 回のシステムメンテナンスを実行できる時間帯 (世界標準時 (UTC))。

  形式: ddd:hh24:mi-ddd:hh24:mi

  デフォルトは、1 週間のうちのランダムな日に起こる、Amazon リージョンあたり 8 時間の範囲からランダムに選択された 30 分の時間窓です。使用可能な時間帯を表示するには、「Amazon Neptune ユーザーガイド」の「Neptune メンテナンスウィンドウ」を参照してください。

  有効な日数: Mon, Tue, Wed, Thu, Fri, Sat, Sun です。

  制約: 最小限の 30 分単位のウィンドウ。

- PreSignedUrl (CLI では: `--pre-signed-url`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  このパラメータは、現在サポートされていません。

- ReplicationSourceIdentifier (CLI では: `--replication-source-identifier`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターがリードレプリカとして作成されている場合の、ソース DB インスタンスまたは DB クラスターの Amazon リソースネーム (ARN)。

- ServerlessV2ScalingConfiguration (CLI では: `--serverless-v2-scaling-configuration`) — ServerlessV2ScalingConfiguration オブジェクト。

  Neptune サーバーレス DB クラスターのスケーリング構成を含みます。

  詳細については、「Amazon Neptune ユーザーガイド」の 「Amazon Neptune サーバーレスの使用」を参照してください。

- StorageEncrypted (CLI では: `--storage-encrypted`) — タイプ `boolean` の BooleanOptional (ブール値 (真または偽) の値)。

  DB クラスターが暗号化されているかどうかを指定します。

- StorageType (CLI では: `--storage-type`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  新しい DB クラスターのストレージタイプ。

  有効な値:

  - **`standard`** — (デフォルト) I/O 使用率が低いか中程度のアプリケーション向けに、費用対効果の高いデータベースストレージを設定します。`standard` に設定すると、ストレージタイプは応答で返されません。
  - **`iopt1`** — 料金が予測可能で、低い I/O レイテンシーと一貫した I/O スループットを必要とする I/O 集約型のグラフワークロードのニーズを満たすように設計された I/O 最適化ストレージを有効にします。

    Neptune I/O 最適化ストレージは、エンジンリリース 1.3.0.0 以降でのみ使用可能です。

- Tags (CLI では: `--tags`) - Tag オブジェクトの配列。

  新しい DB クラスターに割り当てるタグ。

- VpcSecurityGroupIds (CLI では: `--vpc-security-group-ids`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  この DB クラスターに関連付ける EC2 VPC セキュリティグループのリスト。

レスポンス

Amazon Neptune DB クラスターの詳細が含まれています。

このデータ型は、the section called “DescribeDBClusters” でレスポンス要素として使用されます。

- AllocatedStorage - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  `AllocatedStorage` は常に 1 を返します。これは、Neptune DB クラスターのストレージサイズが固定されていないため、必要に応じて自動的に調整されるためです。
- AssociatedRoles – DBClusterRole オブジェクトの配列。

  DB クラスターに関連付けられている Amazon Identity and Access Management (IAM) ロールのリストを入力します。DB クラスターに関連付けられている IAM ロールは、ユーザーに代わって他の Amazon のサービスにアクセスするための DB クラスターへのアクセス許可を付与します。
- AutomaticRestartTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  DB クラスターが自動的に再起動される時刻。
- AvailabilityZones — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターのインスタンスを作成できる EC2 アベイラビリティーゾーン (AZ) のリストを入力します。
- BacktrackConsumedChangeRecords — タイプ `long` の LongOptional (符号付き 64 ビット整数)。

  Neptune ではサポートされていません。
- BacktrackWindow — タイプ `long` の LongOptional (符号付き 64 ビット整数)。

  Neptune ではサポートされていません。
- BackupRetentionPeriod - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  自動 DB スナップショットが保持される日数を指定します。
- Capacity - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

Neptune ではサポートされていません。

- CloneGroupId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターが関連付けられているクローングループを識別します。

- ClusterCreateTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  DB クラスターが作成された時刻を協定世界時 (UTC) で指定します。

- CopyTagsToSnapshot — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽) の値)。

  `true` に設定されている場合、タグは、作成された DB クラスターのいずれかのスナップショットにコピーされます。

- CrossAccountClone — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽) の値)。

  `true`に設定されている場合、DB クラスターを複数のアカウント間でクローンできます。

- DatabaseName — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターの作成時に指定された場合は、作成時に提供されたこの DB クラスターの初期データベースの名前が入ります。DB クラスターの存続中はこれと同じ名前が返されます。

- DBClusterArn — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターの Amazon リソースネーム (ARN) を返します。

- DBClusterIdentifier — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  ユーザーが指定した DB クラスター識別子が含まれています。この識別子は、DB クラスターを識別する一意のキーです。

- DBClusterMembers – DBClusterMember オブジェクトの配列。

  DB クラスターを構成するインスタンスのリストを入力します。

- DBClusterParameterGroup — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターに使用する DB クラスターパラメータグループの名前を指定します。

- DbClusterResourceId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターの Amazon リージョン固有のイミュータブルな識別子。この識別子は、DB クラスターの Amazon KMS キーにアクセスするたびに Amazon CloudTrail ログエントリに記録されます。

- DBSubnetGroup — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  サブネットグループ内の名前、説明、サブネットなど、DB クラスターに関連付けられているサブネットグループに関する情報を指定します。
- DeletionProtection — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽) の値)。

  DB クラスターで削除保護が有効になっているかどうかを示します。削除保護が有効になっている場合、データベースを削除することはできません。
- EarliestBacktrackTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  Neptune ではサポートされていません。
- EarliestRestorableTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  特定時点への復元によりデータベースを復元できる最も早い時刻を指定します。
- EnabledCloudwatchLogsExports — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターが CloudWatch Logs にエクスポートするように設定されているログタイプのリスト。有効なログタイプは、`audit` (監査ログを CloudWatch に公開する場合) と slowquery (スロークエリログを CloudWatch に公開する場合) です。「Amazon CloudWatch Logs への Neptune ログの発行」を参照してください。
- Endpoint — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターのプライマリインスタンスの接続エンドポイントを指定します。
- Engine — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターに使用されるデータベースエンジンの名前を入力します。
- EngineVersion — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  データベースエンジンのバージョンを示します。
- GlobalClusterIdentifier — GlobalClusterIdentifier、Type: `string` (UTF-8 でエンコードされた文字列)、1 ~ 255、正規表現 `[A-Za-z][0-9A-Za-z-:._]*` に一致。

  ユーザーが指定したグローバルクラスター識別子を含みます。この識別子は、グローバルクラスターを識別する一意のキーです。
- HostedZoneId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

ホストゾーンを作成するときに Amazon Route 53 が割り当てる ID を指定します。

- IAMDatabaseAuthenticationEnabled — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  Amazon Identity and Access Management (IAM) アカウントのデータベースアカウントへのマッピングが有効な場合は true、それ以外の場合は false です。

- IOOptimizedNextAllowedModificationTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  次回は、`iopt1` ストレージタイプを使用するように DB クラスターを変更できます。

- KmsKeyId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  `StorageEncrypted` が true の場合、暗号化された DB クラスターの Amazon KMS キー識別子。

- LatestRestorableTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  特定時点への復元によりデータベースを復元できる最新の日時を指定します。

- MultiAZ — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  DB クラスターが複数のアベイラビリティーゾーンにインスタンスを持つかどうかを指定します。

- PendingModifiedValues – ClusterPendingModifiedValues オブジェクト。

  このデータ型は、`ModifyDBCluster` 操作でレスポンス要素として使用され、次のメンテナンスウィンドウに適用される変更を含みます。

- PercentProgress — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  オペレーションの進行状況をパーセンテージで指定します。

- Port - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  データベースエンジンがリッスンするポートを指定します。

- PreferredBackupWindow — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  `BackupRetentionPeriod` に応じた、自動バックアップが有効な場合に自動バックアップが作成される毎日の時間範囲を指定します。

- PreferredMaintenanceWindow — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  週 1 回のシステムメンテナンスを実行できる時間範囲を世界標準時 (UTC) で指定します。

- ReaderEndpoint — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

DB クラスターのリーダーエンドポイント。DB クラスターの読み込みエンドポイントは、DB クラスター内で使用できるリードレプリカ間で接続を負荷分散します。クライアントが読み取りエンドポイントへの新規接続をリクエストすると、Neptune によって接続リクエストが DB クラスターのリードレプリカ間で配分されます。この機能は、DB クラスターの複数のリードレプリカ間の読み取りワークロードを分散させる役に立ちます。

フェイルオーバーが発生し、接続しているリードレプリカがプライマリインスタンスに昇格すると、接続は削除されます。読み取りワークロードをクラスター内の他のリードレプリカに送信し続けるために、読み込みエンドポイントに再接続することができます。

- ReadReplicaIdentifiers — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターに関連付けられているリードレプリカの 1 つ以上の識別子を含みます。

- ReplicationSourceIdentifier — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  Neptune ではサポートされていません。

- ReplicationType — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  Neptune ではサポートされていません。

- ServerlessV2ScalingConfiguration – ServerlessV2ScalingConfigurationInfo オブジェクト。

  Neptune サーバーレス DB クラスターのスケーリング設定を表示します。

  詳細については、「Amazon Neptune ユーザーガイド」の 「Amazon Neptune サーバーレスの使用」を参照してください。

- Status — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターの現在の状態を指定します。

- StorageEncrypted - タイプ `boolean` のブール値 (ブール値 (真または偽))。

  DB クラスターが暗号化されているかどうかを指定します。

- StorageType — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターで使用するストレージタイプ。

  有効な値:

  - **`standard`** — (デフォルト) I/O 使用率が低いか中程度のアプリケーション向けに、費用対効果の高いデータベースストレージを提供します。

- **`iopt1`** — 料金が予測可能で、低い I/O レイテンシーと一貫した I/O スループットを必要とする I/O 集約型のグラフワークロードのニーズを満たすように設計された I/O 最適化ストレージを有効にします。

  Neptune I/O 最適化ストレージは、エンジンリリース 1.3.0.0 以降でのみ使用可能です。

- VpcSecurityGroups – VpcSecurityGroupMembership オブジェクトの配列。

  DB クラスターが属する VPC セキュリティグループのリストを入力します。

エラー

- DBClusterAlreadyExistsFault
- InsufficientStorageClusterCapacityFault
- DBClusterQuotaExceededFault
- StorageQuotaExceededFault
- DBSubnetGroupNotFoundFault
- InvalidVPCNetworkStateFault
- InvalidDBClusterStateFault
- InvalidDBSubnetGroupStateFault
- InvalidSubnet
- InvalidDBInstanceStateFault
- DBClusterParameterGroupNotFoundFault
- KMSKeyNotAccessibleFault
- DBClusterNotFoundFault
- DBInstanceNotFoundFault
- DBSubnetGroupDoesNotCoverEnoughAZs
- GlobalClusterNotFoundFault
- InvalidGlobalClusterStateFault

DeleteDBCluster (アクション)

この API の AWS CLI 名は `delete-db-cluster` です。

DeleteDBCluster アクションでは、以前にプロビジョニングされた DB クラスターを削除します。DB クラスターを削除すると、その DB クラスターの自動バックアップはすべて削除され、復旧できません。指定した DB クラスターの手動 DB クラスタースナップショットは削除されません。

削除保護が有効になっている場合、DB クラスターは削除できません。削除するには、まず `DeletionProtection` フィールドを `False` に設定する必要があります。

リクエスト

- DBClusterIdentifier (CLI では: `--db-cluster-identifier`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  削除する DB クラスターの DB クラスター識別子。このパラメータでは大文字と小文字は区別されません。

  制約:

  - 既存の DBClusterIdentifier と一致する必要があります。
- FinalDBSnapshotIdentifier (CLI では: `--final-db-snapshot-identifier`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  `SkipFinalSnapshot` が `false` に設定された場合の、作成された新規の DB クラスタースナップショットの DB クラスタースナップショット識別子。

  > **Note**
  >
  > このパラメータを指定して `SkipFinalShapshot` パラメータを true に設定すると、エラーが発生します。

  制約:

  - 1 ~ 255 の英字、数字、ハイフンである必要があります。
  - 1 字目は英字である必要があります。
  - ハイフンを、文字列の最後に使用したり、2 つ続けて使用したりすることはできません
- SkipFinalSnapshot (CLI では: `--skip-final-snapshot`) — タイプ: `boolean` のブール値 (ブール値 (真または偽) の値)。

DB クラスターの削除前に最終 DB クラススナップショットを作成するかどうかを指定します。`true` を指定した場合、DB クラススナップショットが作成されます。`false` が指定されている場合、DB クラスターの削除前に DB クラススナップショットが作成されます。

> **Note**
>
> `SkipFinalSnapshot` が `false` の場合は、`FinalDBSnapshotIdentifier` パラメータを指定する必要があります。

デフォルト: `false`

レスポンス

Amazon Neptune DB クラスターの詳細が含まれています。

このデータ型は、the section called “DescribeDBClusters” でレスポンス要素として使用されます。

- AllocatedStorage - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  `AllocatedStorage` は常に 1 を返します。これは、Neptune DB クラスターのストレージサイズが固定されていないため、必要に応じて自動的に調整されるためです。
- AssociatedRoles – DBClusterRole オブジェクトの配列。

  DB クラスターに関連付けられている Amazon Identity and Access Management (IAM) ロールのリストを入力します。DB クラスターに関連付けられている IAM ロールは、ユーザーに代わって他の Amazon のサービスにアクセスするための DB クラスターへのアクセス許可を付与します。
- AutomaticRestartTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  DB クラスターが自動的に再起動される時刻。
- AvailabilityZones — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターのインスタンスを作成できる EC2 アベイラビリティーゾーン (AZ) のリストを入力します。
- BacktrackConsumedChangeRecords — タイプ `long` の LongOptional (符号付き 64 ビット整数)。

  Neptune ではサポートされていません。

- BacktrackWindow — タイプ `long` の LongOptional (符号付き 64 ビット整数)。

  Neptune ではサポートされていません。
- BackupRetentionPeriod - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  自動 DB スナップショットが保持される日数を指定します。
- Capacity - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  Neptune ではサポートされていません。
- CloneGroupId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターが関連付けられているクローングループを識別します。
- ClusterCreateTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  DB クラスターが作成された時刻を協定世界時 (UTC) で指定します。
- CopyTagsToSnapshot — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽) の値)。

  `true` に設定されている場合、タグは、作成された DB クラスターのいずれかのスナップショットにコピーされます。
- CrossAccountClone — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽) の値)。

  `true`に設定されている場合、DB クラスターを複数のアカウント間でクローンできます。
- DatabaseName — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターの作成時に指定された場合は、作成時に提供されたこの DB クラスターの初期データベースの名前が入ります。DB クラスターの存続中はこれと同じ名前が返されます。
- DBClusterArn — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターの Amazon リソースネーム (ARN) を返します。
- DBClusterIdentifier — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  ユーザーが指定した DB クラスター識別子が含まれています。この識別子は、DB クラスターを識別する一意のキーです。
- DBClusterMembers – DBClusterMember オブジェクトの配列。

  DB クラスターを構成するインスタンスのリストを入力します。
- DBClusterParameterGroup — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

DB クラスターに使用する DB クラスターパラメータグループの名前を指定します。

- DbClusterResourceId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターの Amazon リージョン固有のイミュータブルな識別子。この識別子は、DB クラスターの Amazon KMS キーにアクセスするたびに Amazon CloudTrail ログエントリに記録されます。

- DBSubnetGroup — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  サブネットグループ内の名前、説明、サブネットなど、DB クラスターに関連付けられているサブネットグループに関する情報を指定します。

- DeletionProtection — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽) の値)。

  DB クラスターで削除保護が有効になっているかどうかを示します。削除保護が有効になっている場合、データベースを削除することはできません。

- EarliestBacktrackTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  Neptune ではサポートされていません。

- EarliestRestorableTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  特定時点への復元によりデータベースを復元できる最も早い時刻を指定します。

- EnabledCloudwatchLogsExports — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターが CloudWatch Logs にエクスポートするように設定されているログタイプのリスト。有効なログタイプは、`audit` (監査ログを CloudWatch に公開する場合) と slowquery (スロークエリログを CloudWatch に公開する場合) です。「Amazon CloudWatch Logs への Neptune ログの発行」を参照してください。

- Endpoint — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターのプライマリインスタンスの接続エンドポイントを指定します。

- Engine — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターに使用されるデータベースエンジンの名前を入力します。

- EngineVersion — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

データベースエンジンのバージョンを示します。

- GlobalClusterIdentifier — GlobalClusterIdentifier、Type: `string` (UTF-8 でエンコードされた文字列)、1～255、正規表現 `[A-Za-z][0-9A-Za-z-:._]*` に一致。

  ユーザーが指定したグローバルクラスター識別子を含みます。この識別子は、グローバルクラスターを識別する一意のキーです。

- HostedZoneId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  ホストゾーンを作成するときに Amazon Route 53 が割り当てる ID を指定します。

- IAMDatabaseAuthenticationEnabled — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  Amazon Identity and Access Management (IAM) アカウントのデータベースアカウントへのマッピングが有効な場合は true、それ以外の場合は false です。

- IOOptimizedNextAllowedModificationTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  次回は、`iopt1` ストレージタイプを使用するように DB クラスターを変更できます。

- KmsKeyId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  `StorageEncrypted` が true の場合、暗号化された DB クラスターの Amazon KMS キー識別子。

- LatestRestorableTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  特定時点への復元によりデータベースを復元できる最新の日時を指定します。

- MultiAZ — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  DB クラスターが複数のアベイラビリティーゾーンにインスタンスを持つかどうかを指定します。

- PendingModifiedValues – ClusterPendingModifiedValues オブジェクト。

  このデータ型は、`ModifyDBCluster` 操作でレスポンス要素として使用され、次のメンテナンスウィンドウに適用される変更を含みます。

- PercentProgress — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  オペレーションの進行状況をパーセンテージで指定します。

- Port - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  データベースエンジンがリッスンするポートを指定します。

- PreferredBackupWindow — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  `BackupRetentionPeriod` に応じた、自動バックアップが有効な場合に自動バックアップが作成される毎日の時間範囲を指定します。

- PreferredMaintenanceWindow — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  週 1 回のシステムメンテナンスを実行できる時間範囲を世界標準時 (UTC) で指定します。

- ReaderEndpoint — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターのリーダーエンドポイント。DB クラスターの読み込みエンドポイントは、DB クラスター内で使用できるリードレプリカ間で接続を負荷分散します。クライアントが読み取りエンドポイントへの新規接続をリクエストすると、Neptune によって接続リクエストが DB クラスターのリードレプリカ間で配分されます。この機能は、DB クラスターの複数のリードレプリカ間の読み取りワークロードを分散させる役に立ちます。

  フェイルオーバーが発生し、接続しているリードレプリカがプライマリインスタンスに昇格すると、接続は削除されます。読み取りワークロードをクラスター内の他のリードレプリカに送信し続けるために、読み込みエンドポイントに再接続することができます。

- ReadReplicaIdentifiers — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターに関連付けられているリードレプリカの 1 つ以上の識別子を含みます。

- ReplicationSourceIdentifier — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  Neptune ではサポートされていません。

- ReplicationType — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  Neptune ではサポートされていません。

- ServerlessV2ScalingConfiguration – ServerlessV2ScalingConfigurationInfo オブジェクト。

  Neptune サーバーレス DB クラスターのスケーリング設定を表示します。

  詳細については、「Amazon Neptune ユーザーガイド」の 「Amazon Neptune サーバーレスの使用」を参照してください。

- Status — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターの現在の状態を指定します。

- StorageEncrypted - タイプ `boolean` のブール値 (ブール値 (真または偽))。

DB クラスターが暗号化されているかどうかを指定します。

- StorageType — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターで使用するストレージタイプ。

  有効な値:

  - **`standard`** — (デフォルト) I/O 使用率が低いか中程度のアプリケーション向けに、費用対効果の高いデータベースストレージを提供します。
  - **`iopt1`** — 料金が予測可能で、低い I/O レイテンシーと一貫した I/O スループットを必要とする I/O 集約型のグラフワークロードのニーズを満たすように設計された I/O 最適化ストレージを有効にします。

    Neptune I/O 最適化ストレージは、エンジンリリース 1.3.0.0 以降でのみ使用可能です。
- VpcSecurityGroups – VpcSecurityGroupMembership オブジェクトの配列。

  DB クラスターが属する VPC セキュリティグループのリストを入力します。

エラー

- DBClusterNotFoundFault
- InvalidDBClusterStateFault
- DBClusterSnapshotAlreadyExistsFault
- SnapshotQuotaExceededFault
- InvalidDBClusterSnapshotStateFault

ModifyDBCluster (アクション)

この API の AWS CLI 名は `modify-db-cluster` です。

DB クラスターの設定を変更します。これらのパラメータとリクエストに新しい値を指定することで、1 つ以上のデータベース設定パラメータを変更できます。

リクエスト

- AllowMajorVersionUpgrade (CLI では: `--allow-major-version-upgrade`) — タイプ: `boolean` のブール値 (ブール値 (真または偽) の値)。

異なるメジャーバージョンへのアップグレードが許可されるかどうかを示す値。

制約: DB クラスターの現在のバージョンとは異なるメジャーバージョンを使用する EngineVersion パラメータを指定するときには、allow-major-version-upgrade フラグを設定する必要があります。

- ApplyImmediately (CLI では: `--apply-immediately`) — タイプ: `boolean` のブール値 (ブール値 (真または偽) の値)。

  DB クラスターの PreferredMaintenanceWindow の設定に関係なく、このリクエストの変更と保留中の変更をできるだけ早く非同期に適用するかどうかを指定する値。このパラメータを false に設定した場合、DB クラスターへの変更は次のメンテナンスウィンドウ中に適用されます。

  ApplyImmediately パラメータは、NewDBClusterIdentifier 値にのみ影響します。ApplyImmediately パラメータ値を false に設定した場合、NewDBClusterIdentifier の値への変更は、次のメンテナンス期間中に適用されます。他のすべての変更は、ApplyImmediately パラメータの値に関係なく、即時に適用されます。

  デフォルト: `false`

- BackupRetentionPeriod (CLI では: `--backup-retention-period`) — タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  自動バックアップを保管する日数。最小値 1 を指定しなければなりません。

  デフォルト: 1

  制約:

  - 1 ~ 35 の値にする必要があります。

- CloudwatchLogsExportConfiguration (CLI では: `--cloudwatch-logs-export-configuration`) — CloudwatchLogsExportConfiguration オブジェクト。

  特定の DB クラスターの CloudWatch Logs にエクスポートするために有効にする口グタイプの設定。「CLI を使用して Neptune 監査口グを CloudWatch Logs に発行する」を参照してください。

- CopyTagsToSnapshot (CLI では: `--copy-tags-to-snapshot`) — タイプ `boolean` の BooleanOptional (ブール値 (真または偽) の値)。

  true に設定されている場合、タグは、作成された DB クラスターのいずれかのスナップショットにコピーされます。

- DBClusterIdentifier (CLI では: `--db-cluster-identifier`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  変更するクラスターの DB クラスター識別子。このパラメータは大文字と小文字が区別されません。

  制約:

  - 既存の DBCluster の識別子と一致する必要があります。
- DBClusterParameterGroupName (CLI では: `--db-cluster-parameter-group-name`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  DB クラスターに使用する DB クラスターパラメータグループの名前。
- DBInstanceParameterGroupName (CLI では: `--db-instance-parameter-group-name`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  DB クラスターのすべてのインスタンスに適用する DB パラメータグループの名前です。

  > **Note**
  >
  > `DBInstanceParameterGroupName` を使用してパラメータグループを適用すると、パラメータの変更は次のメンテナンスウィンドウには適用されるのではなく、ただちに適用されます。

  デフォルト: 既存の名前設定

  制約:

  - DB パラメータグループは、ターゲットの DB クラスターのバージョンと同じ DB パラメータグループファミリーに属している必要があります。
  - `DBInstanceParameterGroupName` パラメータは、`AllowMajorVersionUpgrade` パラメータと組み合わせた場合のみ有効です。
- DeletionProtection (CLI では: `--deletion-protection`) — タイプ `boolean` の BooleanOptional (ブール値 (真または偽) の値)。

  DB クラスターで削除保護が有効になっているかどうかを示す値。削除保護が有効になっている場合、データベースを削除することはできません。デフォルトでは、削除保護は無効です。

- EnableIAMDatabaseAuthentication (CLI では: `--enable-iam-database-authentication`) — タイプ `boolean` の BooleanOptional (ブール値 (真または偽) の値)。

  Amazon Identity and Access Management (IAM) アカウントのデータベースアカウントへのマッピングが有効な場合は true、それ以外の場合は false です。

  デフォルト: `false`

- EngineVersion (CLI では: `--engine-version`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  アップグレードするデータベースエンジンのバージョン番号。このパラメータを変更すると、機能停止が発生します。`ApplyImmediately` パラメータが true に設定されされない限り、変更は次のメンテナンスウィンドウ中に適用されます。

  有効なエンジンバージョンのリストについては、「Amazon Neptune のエンジンリリース」を参照するか、the section called “DescribeDBEngineVersions” に連絡してください。

- NewDBClusterIdentifier (CLI では: `--new-db-cluster-identifier`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  DB クラスターの名前を変更する場合の DB クラスターの新規 DB クラスター識別子 この値は小文字で保存されます。

  制約:

  - 1 ~ 63 の文字、数字またはハイフンを使用する必要があります。
  - 1 字目は文字である必要があります
  - ハイフンを、文字列の最後に使用したり、2 つ続けて使用したりすることはできません

  例: `my-cluster2`

- Port (CLI では: `--port`) — タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  DB クラスターが接続を受け付けるポート番号。

  制約: 値は `1150-65535` である必要があります。

  デフォルト : 元の DB クラスターと同じポート。

- PreferredBackupWindow (CLI では: `--preferred-backup-window`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

BackupRetentionPeriod パラメータを使用して自動バックアップが有効になっている場合に、自動バックアップが作成される毎日の時間範囲。

デフォルトのバックアップウィンドウは 30 分のウィンドウで、Amazon リージョンごとに 8 時間の時間ブロックからランダムに選択されます。

制約:

- hh24:mi-hh24:mi の形式である必要があります。
- 時間は協定世界時 (UTC) である必要があります。
- 必要なメンテナンス期間と競合してはいけません。
- 少なくとも 30 分以上必要です。

- PreferredMaintenanceWindow (CLI では: --preferred-maintenance-window) — タイプ: string (UTF-8 でエンコードされた文字列) の文字列。

  週 1 回のシステムメンテナンスを実行できる時間帯 (世界標準時 (UTC))。

  形式: ddd:hh24:mi-ddd:hh24:mi

  デフォルトは、1 週間のうちのランダムな日に起こる、Amazon リージョンあたり 8 時間の範囲からランダムに選択された 30 分の時間窓です。

  有効な日数: Mon, Tue, Wed, Thu, Fri, Sat, Sun です。

  制約: 最小限の 30 分単位のウィンドウ。

- ServerlessV2ScalingConfiguration (CLI では: --serverless-v2-scaling-configuration) — ServerlessV2ScalingConfiguration オブジェクト。

  Neptune サーバーレス DB クラスターのスケーリング構成を含みます。

  詳細については、「Amazon Neptune ユーザーガイド」の 「Amazon Neptune サーバーレスの使用」を参照してください。

- StorageType (CLI では: --storage-type) — タイプ: string (UTF-8 でエンコードされた文字列) の文字列。

  DB クラスターに関連付けるストレージタイプ。

  有効な値:

- **`standard`** — (デフォルト) I/O 使用率が低いか中程度のアプリケーション向けに、費用対効果の高いデータベースストレージを設定します。
- **`iopt1`** — 料金が予測可能で、低い I/O レイテンシーと一貫した I/O スループットを必要とする I/O 集約型のグラフワークロードのニーズを満たすように設計された I/O 最適化ストレージを有効にします。

  Neptune I/O 最適化ストレージは、エンジンリリース 1.3.0.0 以降でのみ使用可能です。

- VpcSecurityGroupIds (CLI では: `--vpc-security-group-ids`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  DB クラスターが属する VPC セキュリティグループのリスト。

レスポンス

Amazon Neptune DB クラスターの詳細が含まれています。

このデータ型は、the section called “DescribeDBClusters” でレスポンス要素として使用されます。

- AllocatedStorage - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  `AllocatedStorage` は常に 1 を返します。これは、Neptune DB クラスターのストレージサイズが固定されていないため、必要に応じて自動的に調整されるためです。

- AssociatedRoles – DBClusterRole オブジェクトの配列。

  DB クラスターに関連付けられている Amazon Identity and Access Management (IAM) ロールのリストを入力します。DB クラスターに関連付けられている IAM ロールは、ユーザーに代わって他の Amazon のサービスにアクセスするための DB クラスターへのアクセス許可を付与します。

- AutomaticRestartTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  DB クラスターが自動的に再起動される時刻。

- AvailabilityZones — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターのインスタンスを作成できる EC2 アベイラビリティーゾーン (AZ) のリストを入力します。

- BacktrackConsumedChangeRecords — タイプ `long` の LongOptional (符号付き 64 ビット整数)。

  Neptune ではサポートされていません。

- BacktrackWindow — タイプ `long` の LongOptional (符号付き 64 ビット整数)。

  Neptune ではサポートされていません。
- BackupRetentionPeriod - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  自動 DB スナップショットが保持される日数を指定します。
- Capacity - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  Neptune ではサポートされていません。
- CloneGroupId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターが関連付けられているクローングループを識別します。
- ClusterCreateTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  DB クラスターが作成された時刻を協定世界時 (UTC) で指定します。
- CopyTagsToSnapshot — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽) の値)。

  `true` に設定されている場合、タグは、作成された DB クラスターのいずれかのスナップショットにコピーされます。
- CrossAccountClone — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽) の値)。

  `true`に設定されている場合、DB クラスターを複数のアカウント間でクローンできます。
- DatabaseName — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターの作成時に指定された場合は、作成時に提供されたこの DB クラスターの初期データベースの名前が入ります。DB クラスターの存続中はこれと同じ名前が返されます。
- DBClusterArn — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターの Amazon リソースネーム (ARN) を返します。
- DBClusterIdentifier — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  ユーザーが指定した DB クラスター識別子が含まれています。この識別子は、DB クラスターを識別する一意のキーです。
- DBClusterMembers – DBClusterMember オブジェクトの配列。

  DB クラスターを構成するインスタンスのリストを入力します。
- DBClusterParameterGroup — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

DB クラスターに使用する DB クラスターパラメータグループの名前を指定します。

- DbClusterResourceId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターの Amazon リージョン固有のイミュータブルな識別子。この識別子は、DB クラスターの Amazon KMS キーにアクセスするたびに Amazon CloudTrail ログエントリに記録されます。

- DBSubnetGroup — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  サブネットグループ内の名前、説明、サブネットなど、DB クラスターに関連付けられているサブネットグループに関する情報を指定します。

- DeletionProtection — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽) の値)。

  DB クラスターで削除保護が有効になっているかどうかを示します。削除保護が有効になっている場合、データベースを削除することはできません。

- EarliestBacktrackTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  Neptune ではサポートされていません。

- EarliestRestorableTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  特定時点への復元によりデータベースを復元できる最も早い時刻を指定します。

- EnabledCloudwatchLogsExports — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターが CloudWatch Logs にエクスポートするように設定されているログタイプのリスト。有効なログタイプは、`audit` (監査ログを CloudWatch に公開する場合) と slowquery (スロークエリログを CloudWatch に公開する場合) です。「Amazon CloudWatch Logs への Neptune ログの発行」を参照してください。

- Endpoint — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターのプライマリインスタンスの接続エンドポイントを指定します。

- Engine — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターに使用されるデータベースエンジンの名前を入力します。

- EngineVersion — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

データベースエンジンのバージョンを示します。

- GlobalClusterIdentifier — GlobalClusterIdentifier、Type: `string` (UTF-8 でエンコードされた文字列)、1～255、正規表現 `[A-Za-z][0-9A-Za-z-:._]*` に一致。

  ユーザーが指定したグローバルクラスター識別子を含みます。この識別子は、グローバルクラスターを識別する一意のキーです。

- HostedZoneId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  ホストゾーンを作成するときに Amazon Route 53 が割り当てる ID を指定します。

- IAMDatabaseAuthenticationEnabled — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  Amazon Identity and Access Management (IAM) アカウントのデータベースアカウントへのマッピングが有効な場合は true、それ以外の場合は false です。

- IOOptimizedNextAllowedModificationTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  次回は、`iopt1` ストレージタイプを使用するように DB クラスターを変更できます。

- KmsKeyId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  `StorageEncrypted` が true の場合、暗号化された DB クラスターの Amazon KMS キー識別子。

- LatestRestorableTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  特定時点への復元によりデータベースを復元できる最新の日時を指定します。

- MultiAZ — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  DB クラスターが複数のアベイラビリティーゾーンにインスタンスを持つかどうかを指定します。

- PendingModifiedValues – ClusterPendingModifiedValues オブジェクト。

  このデータ型は、`ModifyDBCluster` 操作でレスポンス要素として使用され、次のメンテナンスウィンドウに適用される変更を含みます。

- PercentProgress — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  オペレーションの進行状況をパーセンテージで指定します。

- Port - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  データベースエンジンがリッスンするポートを指定します。

- PreferredBackupWindow — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  `BackupRetentionPeriod` に応じた、自動バックアップが有効な場合に自動バックアップが作成される毎日の時間範囲を指定します。

- PreferredMaintenanceWindow — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  週 1 回のシステムメンテナンスを実行できる時間範囲を世界標準時 (UTC) で指定します。

- ReaderEndpoint — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターのリーダーエンドポイント。DB クラスターの読み込みエンドポイントは、DB クラスター内で使用できるリードレプリカ間で接続を負荷分散します。クライアントが読み取りエンドポイントへの新規接続をリクエストすると、Neptune によって接続リクエストが DB クラスターのリードレプリカ間で配分されます。この機能は、DB クラスターの複数のリードレプリカ間の読み取りワークロードを分散させる役に立ちます。

  フェイルオーバーが発生し、接続しているリードレプリカがプライマリインスタンスに昇格すると、接続は削除されます。読み取りワークロードをクラスター内の他のリードレプリカに送信し続けるために、読み込みエンドポイントに再接続することができます。

- ReadReplicaIdentifiers — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターに関連付けられているリードレプリカの 1 つ以上の識別子を含みます。

- ReplicationSourceIdentifier — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  Neptune ではサポートされていません。

- ReplicationType — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  Neptune ではサポートされていません。

- ServerlessV2ScalingConfiguration – ServerlessV2ScalingConfigurationInfo オブジェクト。

  Neptune サーバーレス DB クラスターのスケーリング設定を表示します。

  詳細については、「Amazon Neptune ユーザーガイド」の 「Amazon Neptune サーバーレスの使用」を参照してください。

- Status — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターの現在の状態を指定します。

- StorageEncrypted - タイプ `boolean` のブール値 (ブール値 (真または偽))。

DB クラスターが暗号化されているかどうかを指定します。

- StorageType — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターで使用するストレージタイプ。

  有効な値:

  - **`standard`** — (デフォルト) I/O 使用率が低いか中程度のアプリケーション向けに、費用対効果の高いデータベースストレージを提供します。
  - **`iopt1`** — 料金が予測可能で、低い I/O レイテンシーと一貫した I/O スループットを必要とする I/O 集約型のグラフワークロードのニーズを満たすように設計された I/O 最適化ストレージを有効にします。

    Neptune I/O 最適化ストレージは、エンジンリリース 1.3.0.0 以降でのみ使用可能です。
- VpcSecurityGroups – VpcSecurityGroupMembership オブジェクトの配列。

  DB クラスターが属する VPC セキュリティグループのリストを入力します。

エラー

- DBClusterNotFoundFault
- InvalidDBClusterStateFault
- StorageQuotaExceededFault
- DBSubnetGroupNotFoundFault
- InvalidVPCNetworkStateFault
- InvalidDBSubnetGroupStateFault
- InvalidSubnet
- DBClusterParameterGroupNotFoundFault
- InvalidDBSecurityGroupStateFault
- InvalidDBInstanceStateFault
- DBClusterAlreadyExistsFault
- StorageTypeNotSupportedFault

StartDBCluster (アクション)

この API の AWS CLI 名は `start-db-cluster` です。

Amazon コンソール、Amazon CLI の stop-db-cluster コマンド、または StopDBCluster API を使用して停止された Amazon Neptune DB クラスターを起動します。

リクエスト

- DBClusterIdentifier (CLI では: `--db-cluster-identifier`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  起動する Neptune DB クラスターの DB クラスター識別子。このパラメータは小文字で保存されます。

レスポンス

Amazon Neptune DB クラスターの詳細が含まれています。

このデータ型は、the section called “DescribeDBClusters” でレスポンス要素として使用されます。

- AllocatedStorage - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  `AllocatedStorage` は常に 1 を返します。これは、Neptune DB クラスターのストレージサイズが固定されていないため、必要に応じて自動的に調整されるためです。
- AssociatedRoles – DBClusterRole オブジェクトの配列。

  DB クラスターに関連付けられている Amazon Identity and Access Management (IAM) ロールのリストを入力します。DB クラスターに関連付けられている IAM ロールは、ユーザーに代わって他の Amazon のサービスにアクセスするための DB クラスターへのアクセス許可を付与します。
- AutomaticRestartTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  DB クラスターが自動的に再起動される時刻。
- AvailabilityZones — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターのインスタンスを作成できる EC2 アベイラビリティーゾーン (AZ) のリストを入力します。
- BacktrackConsumedChangeRecords — タイプ `long` の LongOptional (符号付き 64 ビット整数)。

Neptune ではサポートされていません。

- BacktrackWindow — タイプ `long` の LongOptional (符号付き 64 ビット整数)。

  Neptune ではサポートされていません。

- BackupRetentionPeriod - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  自動 DB スナップショットが保持される日数を指定します。

- Capacity - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  Neptune ではサポートされていません。

- CloneGroupId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターが関連付けられているクローングループを識別します。

- ClusterCreateTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  DB クラスターが作成された時刻を協定世界時 (UTC) で指定します。

- CopyTagsToSnapshot — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽) の値)。

  `true` に設定されている場合、タグは、作成された DB クラスターのいずれかのスナップショットにコピーされます。

- CrossAccountClone — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽) の値)。

  `true`に設定されている場合、DB クラスターを複数のアカウント間でクローンできます。

- DatabaseName — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターの作成時に指定された場合は、作成時に提供されたこの DB クラスターの初期データベースの名前が入ります。DB クラスターの存続中はこれと同じ名前が返されます。

- DBClusterArn — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターの Amazon リソースネーム (ARN) を返します。

- DBClusterIdentifier — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  ユーザーが指定した DB クラスター識別子が含まれています。この識別子は、DB クラスターを識別する一意のキーです。

- DBClusterMembers – DBClusterMember オブジェクトの配列。

DB クラスターを構成するインスタンスのリストを入力します。

- DBClusterParameterGroup — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターに使用する DB クラスターパラメータグループの名前を指定します。

- DbClusterResourceId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターの Amazon リージョン固有のイミュータブルな識別子。この識別子は、DB クラスターの Amazon KMS キーにアクセスするたびに Amazon CloudTrail ログエントリに記録されます。

- DBSubnetGroup — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  サブネットグループ内の名前、説明、サブネットなど、DB クラスターに関連付けられているサブネットグループに関する情報を指定します。

- DeletionProtection — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽) の値)。

  DB クラスターで削除保護が有効になっているかどうかを示します。削除保護が有効になっている場合、データベースを削除することはできません。

- EarliestBacktrackTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  Neptune ではサポートされていません。

- EarliestRestorableTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  特定時点への復元によりデータベースを復元できる最も早い時刻を指定します。

- EnabledCloudwatchLogsExports — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターが CloudWatch Logs にエクスポートするように設定されているログタイプのリスト。有効なログタイプは、`audit` (監査ログを CloudWatch に公開する場合) と slowquery (スロークエリログを CloudWatch に公開する場合) です。「Amazon CloudWatch Logs への Neptune ログの発行」を参照してください。

- Endpoint — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターのプライマリインスタンスの接続エンドポイントを指定します。

- Engine — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

この DB クラスターに使用されるデータベースエンジンの名前を入力します。

- EngineVersion — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  データベースエンジンのバージョンを示します。

- GlobalClusterIdentifier — GlobalClusterIdentifier、Type: `string` (UTF-8 でエンコードされた文字列)、1 ~ 255、正規表現 `[A-Za-z][0-9A-Za-z-:._]*` に一致。

  ユーザーが指定したグローバルクラスター識別子を含みます。この識別子は、グローバルクラスターを識別する一意のキーです。

- HostedZoneId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  ホストゾーンを作成するときに Amazon Route 53 が割り当てる ID を指定します。

- IAMDatabaseAuthenticationEnabled — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  Amazon Identity and Access Management (IAM) アカウントのデータベースアカウントへのマッピングが有効な場合は true、それ以外の場合は false です。

- IOOptimizedNextAllowedModificationTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  次回は、`iopt1` ストレージタイプを使用するように DB クラスターを変更できます。

- KmsKeyId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  `StorageEncrypted` が true の場合、暗号化された DB クラスターの Amazon KMS キー識別子。

- LatestRestorableTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  特定時点への復元によりデータベースを復元できる最新の日時を指定します。

- MultiAZ — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  DB クラスターが複数のアベイラビリティーゾーンにインスタンスを持つかどうかを指定します。

- PendingModifiedValues – ClusterPendingModifiedValues オブジェクト。

  このデータ型は、`ModifyDBCluster` 操作でレスポンス要素として使用され、次のメンテナンスウィンドウに適用される変更を含みます。

- PercentProgress — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  オペレーションの進行状況をパーセンテージで指定します。

- Port - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  データベースエンジンがリッスンするポートを指定します。

- PreferredBackupWindow — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  `BackupRetentionPeriod` に応じた、自動バックアップが有効な場合に自動バックアップが作成される毎日の時間範囲を指定します。

- PreferredMaintenanceWindow — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  週 1 回のシステムメンテナンスを実行できる時間範囲を世界標準時 (UTC) で指定します。

- ReaderEndpoint — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターのリーダーエンドポイント。DB クラスターの読み込みエンドポイントは、DB クラスター内で使用できるリードレプリカ間で接続を負荷分散します。クライアントが読み取りエンドポイントへの新規接続をリクエストすると、Neptune によって接続リクエストが DB クラスターのリードレプリカ間で配分されます。この機能は、DB クラスターの複数のリードレプリカ間の読み取りワークロードを分散させる役に立ちます。

  フェイルオーバーが発生し、接続しているリードレプリカがプライマリインスタンスに昇格すると、接続は削除されます。読み取りワークロードをクラスター内の他のリードレプリカに送信し続けるために、読み込みエンドポイントに再接続することができます。

- ReadReplicaIdentifiers — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターに関連付けられているリードレプリカの 1 つ以上の識別子を含みます。

- ReplicationSourceIdentifier — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  Neptune ではサポートされていません。

- ReplicationType — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  Neptune ではサポートされていません。

- ServerlessV2ScalingConfiguration – ServerlessV2ScalingConfigurationInfo オブジェクト。

  Neptune サーバーレス DB クラスターのスケーリング設定を表示します。

  詳細については、「Amazon Neptune ユーザーガイド」の 「Amazon Neptune サーバーレスの使用」を参照してください。

- Status — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

この DB クラスターの現在の状態を指定します。

- StorageEncrypted - タイプ `boolean` のブール値 (ブール値 (真または偽))。

  DB クラスターが暗号化されているかどうかを指定します。

- StorageType — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターで使用するストレージタイプ。

  有効な値:

  - **`standard`** — (デフォルト) I/O 使用率が低いか中程度のアプリケーション向けに、費用対効果の高いデータベースストレージを提供します。
  - **`iopt1`** — 料金が予測可能で、低い I/O レイテンシーと一貫した I/O スループットを必要とする I/O 集約型のグラフワークロードのニーズを満たすように設計された I/O 最適化ストレージを有効にします。

    Neptune I/O 最適化ストレージは、エンジンリリース 1.3.0.0 以降でのみ使用可能です。

- VpcSecurityGroups – VpcSecurityGroupMembership オブジェクトの配列。

  DB クラスターが属する VPC セキュリティグループのリストを入力します。

エラー

- DBClusterNotFoundFault
- InvalidDBClusterStateFault
- InvalidDBInstanceStateFault

StopDBCluster (アクション)

この API の AWS CLI 名は `stop-db-cluster` です。

Amazon Neptune DB クラスターを停止します。DB クラスターを停止すると、Neptune は DB クラスターのメタデータ (エンドポイントや DB パラメータグループを含む) を保持します。

Neptune はトランザクションログも保持するため、必要に応じてポイントインタイムの復元を実行できます。

リクエスト

- DBClusterIdentifier (CLI では: `--db-cluster-identifier`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  停止する Neptune DB クラスターの DB クラスター識別子。このパラメータは小文字で保存されます。

レスポンス

Amazon Neptune DB クラスターの詳細が含まれています。

このデータ型は、the section called “DescribeDBClusters” でレスポンス要素として使用されます。

- AllocatedStorage - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  `AllocatedStorage` は常に 1 を返します。これは、Neptune DB クラスターのストレージサイズが固定されていないため、必要に応じて自動的に調整されるためです。
- AssociatedRoles – DBClusterRole オブジェクトの配列。

  DB クラスターに関連付けられている Amazon Identity and Access Management (IAM) ロールのリストを入力します。DB クラスターに関連付けられている IAM ロールは、ユーザーに代わって他の Amazon のサービスにアクセスするための DB クラスターへのアクセス許可を付与します。
- AutomaticRestartTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  DB クラスターが自動的に再起動される時刻。
- AvailabilityZones — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターのインスタンスを作成できる EC2 アベイラビリティーゾーン (AZ) のリストを入力します。
- BacktrackConsumedChangeRecords — タイプ `long` の LongOptional (符号付き 64 ビット整数)。

  Neptune ではサポートされていません。
- BacktrackWindow — タイプ `long` の LongOptional (符号付き 64 ビット整数)。

  Neptune ではサポートされていません。
- BackupRetentionPeriod - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  自動 DB スナップショットが保持される日数を指定します。
- Capacity - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

Neptune ではサポートされていません。

- CloneGroupId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターが関連付けられているクローングループを識別します。

- ClusterCreateTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  DB クラスターが作成された時刻を協定世界時 (UTC) で指定します。

- CopyTagsToSnapshot — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽) の値)。

  `true` に設定されている場合、タグは、作成された DB クラスターのいずれかのスナップショットにコピーされます。

- CrossAccountClone — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽) の値)。

  `true`に設定されている場合、DB クラスターを複数のアカウント間でクローンできます。

- DatabaseName — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターの作成時に指定された場合は、作成時に提供されたこの DB クラスターの初期データベースの名前が入ります。DB クラスターの存続中はこれと同じ名前が返されます。

- DBClusterArn — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターの Amazon リソースネーム (ARN) を返します。

- DBClusterIdentifier — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  ユーザーが指定した DB クラスター識別子が含まれています。この識別子は、DB クラスターを識別する一意のキーです。

- DBClusterMembers – DBClusterMember オブジェクトの配列。

  DB クラスターを構成するインスタンスのリストを入力します。

- DBClusterParameterGroup — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターに使用する DB クラスターパラメータグループの名前を指定します。

- DbClusterResourceId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターの Amazon リージョン固有のイミュータブルな識別子。この識別子は、DB クラスターの Amazon KMS キーにアクセスするたびに Amazon CloudTrail ログエントリに記録されます。

- DBSubnetGroup — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  サブネットグループ内の名前、説明、サブネットなど、DB クラスターに関連付けられているサブネットグループに関する情報を指定します。
- DeletionProtection — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽) の値)。

  DB クラスターで削除保護が有効になっているかどうかを示します。削除保護が有効になっている場合、データベースを削除することはできません。
- EarliestBacktrackTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  Neptune ではサポートされていません。
- EarliestRestorableTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  特定時点への復元によりデータベースを復元できる最も早い時刻を指定します。
- EnabledCloudwatchLogsExports — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターが CloudWatch Logs にエクスポートするように設定されているログタイプのリスト。有効なログタイプは、`audit` (監査ログを CloudWatch に公開する場合) と slowquery (スロークエリログを CloudWatch に公開する場合) です。「Amazon CloudWatch Logs への Neptune ログの発行」を参照してください。
- Endpoint — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターのプライマリインスタンスの接続エンドポイントを指定します。
- Engine — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターに使用されるデータベースエンジンの名前を入力します。
- EngineVersion — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  データベースエンジンのバージョンを示します。
- GlobalClusterIdentifier — GlobalClusterIdentifier、Type: `string` (UTF-8 でエンコードされた文字列)、1 ~ 255、正規表現 `[A-Za-z][0-9A-Za-z-:._]*` に一致。

  ユーザーが指定したグローバルクラスター識別子を含みます。この識別子は、グローバルクラスターを識別する一意のキーです。
- HostedZoneId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

ホストゾーンを作成するときに Amazon Route 53 が割り当てる ID を指定します。

- IAMDatabaseAuthenticationEnabled — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  Amazon Identity and Access Management (IAM) アカウントのデータベースアカウントへのマッピングが有効な場合は true、それ以外の場合は false です。

- IOOptimizedNextAllowedModificationTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  次回は、`iopt1` ストレージタイプを使用するように DB クラスターを変更できます。

- KmsKeyId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  `StorageEncrypted` が true の場合、暗号化された DB クラスターの Amazon KMS キー識別子。

- LatestRestorableTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  特定時点への復元によりデータベースを復元できる最新の日時を指定します。

- MultiAZ — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  DB クラスターが複数のアベイラビリティーゾーンにインスタンスを持つかどうかを指定します。

- PendingModifiedValues – ClusterPendingModifiedValues オブジェクト。

  このデータ型は、`ModifyDBCluster` 操作でレスポンス要素として使用され、次のメンテナンスウィンドウに適用される変更を含みます。

- PercentProgress — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  オペレーションの進行状況をパーセンテージで指定します。

- Port - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  データベースエンジンがリッスンするポートを指定します。

- PreferredBackupWindow — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  `BackupRetentionPeriod` に応じた、自動バックアップが有効な場合に自動バックアップが作成される毎日の時間範囲を指定します。

- PreferredMaintenanceWindow — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  週 1 回のシステムメンテナンスを実行できる時間範囲を世界標準時 (UTC) で指定します。

- ReaderEndpoint — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

DB クラスターのリーダーエンドポイント。DB クラスターの読み込みエンドポイントは、DB クラスター内で使用できるリードレプリカ間で接続を負荷分散します。クライアントが読み取りエンドポイントへの新規接続をリクエストすると、Neptune によって接続リクエストが DB クラスターのリードレプリカ間で配分されます。この機能は、DB クラスターの複数のリードレプリカ間の読み取りワークロードを分散させる役に立ちます。

フェイルオーバーが発生し、接続しているリードレプリカがプライマリインスタンスに昇格すると、接続は削除されます。読み取りワークロードをクラスター内の他のリードレプリカに送信し続けるために、読み込みエンドポイントに再接続することができます。

- ReadReplicaIdentifiers — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターに関連付けられているリードレプリカの 1 つ以上の識別子を含みます。

- ReplicationSourceIdentifier — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  Neptune ではサポートされていません。

- ReplicationType — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  Neptune ではサポートされていません。

- ServerlessV2ScalingConfiguration – ServerlessV2ScalingConfigurationInfo オブジェクト。

  Neptune サーバーレス DB クラスターのスケーリング設定を表示します。

  詳細については、「Amazon Neptune ユーザーガイド」の 「Amazon Neptune サーバーレスの使用」を参照してください。

- Status — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターの現在の状態を指定します。

- StorageEncrypted - タイプ `boolean` のブール値 (ブール値 (真または偽))。

  DB クラスターが暗号化されているかどうかを指定します。

- StorageType — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターで使用するストレージタイプ。

  有効な値:

  - **`standard`** — (デフォルト) I/O 使用率が低いか中程度のアプリケーション向けに、費用対効果の高いデータベースストレージを提供します。

- **`iopt1`** — 料金が予測可能で、低い I/O レイテンシーと一貫した I/O スループットを必要とする I/O 集約型のグラフワークロードのニーズを満たすように設計された I/O 最適化ストレージを有効にします。

  Neptune I/O 最適化ストレージは、エンジンリリース 1.3.0.0 以降でのみ使用可能です。

- VpcSecurityGroups – VpcSecurityGroupMembership オブジェクトの配列。

  DB クラスターが属する VPC セキュリティグループのリストを入力します。

エラー

- DBClusterNotFoundFault
- InvalidDBClusterStateFault
- InvalidDBInstanceStateFault

AddRoleToDBCluster (アクション)

この API の AWS CLI 名は `add-role-to-db-cluster` です。

Neptune DB クラスターから Identity and Access Management (IAM) ロールを関連付けます。

リクエスト

- DBClusterIdentifier (CLI では: `--db-cluster-identifier`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  IAM ロールを関連付ける DB クラスターの名前。

- FeatureName (CLI では: `--feature-name`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  IAM ロールの関連付けを解除する Neptune DB クラスターの機能の名前。サポートされている機能名のリストについては、「the section called “DBEngineVersion”」を参照してください 。

- RoleArn (CLI では: `--role-arn`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  Neptune DB クラスターに関連付ける IAM ロールの Amazon リソースネーム (ARN) 。例: `arn:aws:iam::123456789012:role/NeptuneAccessRole`

レスポンス

- 応答パラメータはありません。

エラー

- DBClusterNotFoundFault
- DBClusterRoleAlreadyExistsFault
- InvalidDBClusterStateFault
- DBClusterRoleQuotaExceededFault

RemoveRoleFromDBCluster (アクション)

この API の AWS CLI 名は `remove-role-from-db-cluster` です。

DB クラスターから Identity and Access Management (IAM) ロールの関連付けを解除します。

リクエスト

- DBClusterIdentifier (CLI では: `--db-cluster-identifier`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  IAM ロールの関連付けを解除する DB クラスターの名前。
- FeatureName (CLI では: `--feature-name`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  IAM ロールの関連付けを解除する DB クラスターの機能の名前。サポートされている機能名のリストについては、「the section called “DescribeDBEngineVersions”」を参照してください 。
- RoleArn (CLI では: `--role-arn`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターから関連付けを解除する IAM ロールの Amazon リソースネーム (ARN) 。例: `arn:aws:iam::123456789012:role/NeptuneAccessRole`

レスポンス

- 応答パラメータはありません。

エラー

- DBClusterNotFoundFault
- DBClusterRoleNotFoundFault
- InvalidDBClusterStateFault

FailoverDBCluster (アクション)

この API の AWS CLI 名は `failover-db-cluster` です。

DB クラスターのフェイルオーバーを強制実行します。

DB クラスターのフェイルオーバーにより、DB クラスター内のリードレプリカの 1 つ (読み取り専用インスタンス) が、プライマリインスタンス (クラスターライター) に昇格されます。

Amazon Neptune は、プライマリインスタンスが失敗した場合、自動的にリードレプリカにフェイルオーバーします (存在する場合)。テストのため、プライマリインスタンスの失敗をシミュレートする場合は、フェイルオーバーを強制できます。DB クラスター内の各インスタンスには独自のエンドポイントアドレスがあるため、フェイルオーバーが完了したときに、それらのエンドポイントアドレスを使用する既存の接続をクリーンアップして再確立する必要があります。

リクエスト

- DBClusterIdentifier (CLI では: `--db-cluster-identifier`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  フェイルオーバーを強制する DB クラスター識別子。このパラメータは大文字と小文字が区別されません。

  制約:

  - 既存の DBCluster の識別子と一致する必要があります。
- TargetDBInstanceIdentifier (CLI では: `--target-db-instance-identifier`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  プライマリインスタンスに昇格するインスタンスの名前。

  DB クラスター内のリードレプリカのインスタンス識別子を指定する必要があります。例えば、`mydbcluster-replica1` です。

レスポンス

Amazon Neptune DB クラスターの詳細が含まれています。

このデータ型は、the section called “DescribeDBClusters” でレスポンス要素として使用されます。

- AllocatedStorage - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  `AllocatedStorage` は常に 1 を返します。これは、Neptune DB クラスターのストレージサイズが固定されていないため、必要に応じて自動的に調整されるためです。

- AssociatedRoles – DBClusterRole オブジェクトの配列。

  DB クラスターに関連付けられている Amazon Identity and Access Management (IAM) ロールのリストを入力します。DB クラスターに関連付けられている IAM ロールは、ユーザーに代わって他の Amazon のサービスにアクセスするための DB クラスターへのアクセス許可を付与します。

- AutomaticRestartTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  DB クラスターが自動的に再起動される時刻。

- AvailabilityZones — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターのインスタンスを作成できる EC2 アベイラビリティーゾーン (AZ) のリストを入力します。

- BacktrackConsumedChangeRecords — タイプ `long` の LongOptional (符号付き 64 ビット整数)。

  Neptune ではサポートされていません。

- BacktrackWindow — タイプ `long` の LongOptional (符号付き 64 ビット整数)。

  Neptune ではサポートされていません。

- BackupRetentionPeriod - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  自動 DB スナップショットが保持される日数を指定します。

- Capacity - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  Neptune ではサポートされていません。

- CloneGroupId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターが関連付けられているクローングループを識別します。

- ClusterCreateTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  DB クラスターが作成された時刻を協定世界時 (UTC) で指定します。
- CopyTagsToSnapshot — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽) の値)。

  `true` に設定されている場合、タグは、作成された DB クラスターのいずれかのスナップショットにコピーされます。
- CrossAccountClone — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽) の値)。

  `true`に設定されている場合、DB クラスターを複数のアカウント間でクローンできます。
- DatabaseName — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターの作成時に指定された場合は、作成時に提供されたこの DB クラスターの初期データベースの名前が入ります。DB クラスターの存続中はこれと同じ名前が返されます。
- DBClusterArn — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターの Amazon リソースネーム (ARN) を返します。
- DBClusterIdentifier — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  ユーザーが指定した DB クラスター識別子が含まれています。この識別子は、DB クラスターを識別する一意のキーです。
- DBClusterMembers – DBClusterMember オブジェクトの配列。

  DB クラスターを構成するインスタンスのリストを入力します。
- DBClusterParameterGroup — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターに使用する DB クラスターパラメータグループの名前を指定します。
- DbClusterResourceId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターの Amazon リージョン固有のイミュータブルな識別子。この識別子は、DB クラスターの Amazon KMS キーにアクセスするたびに Amazon CloudTrail ログエントリに記録されます。
- DBSubnetGroup — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  サブネットグループ内の名前、説明、サブネットなど、DB クラスターに関連付けられているサブネットグループに関する情報を指定します。
- DeletionProtection — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽) の値)。

DB クラスターで削除保護が有効になっているかどうかを示します。削除保護が有効になっている場合、データベースを削除することはできません。

- EarliestBacktrackTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  Neptune ではサポートされていません。

- EarliestRestorableTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  特定時点への復元によりデータベースを復元できる最も早い時刻を指定します。

- EnabledCloudwatchLogsExports — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターが CloudWatch Logs にエクスポートするように設定されているログタイプのリスト。有効なログタイプは、`audit` (監査ログを CloudWatch に公開する場合) と slowquery (スロークエリログを CloudWatch に公開する場合) です。「Amazon CloudWatch Logs への Neptune ログの発行」を参照してください。

- Endpoint — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターのプライマリインスタンスの接続エンドポイントを指定します。

- Engine — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターに使用されるデータベースエンジンの名前を入力します。

- EngineVersion — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  データベースエンジンのバージョンを示します。

- GlobalClusterIdentifier — GlobalClusterIdentifier、Type: `string` (UTF-8 でエンコードされた文字列)、1 ~ 255、正規表現 `[A-Za-z][0-9A-Za-z-:._]*` に一致。

  ユーザーが指定したグローバルクラスター識別子を含みます。この識別子は、グローバルクラスターを識別する一意のキーです。

- HostedZoneId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  ホストゾーンを作成するときに Amazon Route 53 が割り当てる ID を指定します。

- IAMDatabaseAuthenticationEnabled — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

Amazon Identity and Access Management (IAM) アカウントのデータベースアカウントへのマッピングが有効な場合は true、それ以外の場合は false です。

- IOOptimizedNextAllowedModificationTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  次回は、`iopt1` ストレージタイプを使用するように DB クラスターを変更できます。

- KmsKeyId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  `StorageEncrypted` が true の場合、暗号化された DB クラスターの Amazon KMS キー識別子。

- LatestRestorableTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  特定時点への復元によりデータベースを復元できる最新の日時を指定します。

- MultiAZ — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  DB クラスターが複数のアベイラビリティーゾーンにインスタンスを持つかどうかを指定します。

- PendingModifiedValues – ClusterPendingModifiedValues オブジェクト。

  このデータ型は、`ModifyDBCluster` 操作でレスポンス要素として使用され、次のメンテナンスウィンドウに適用される変更を含みます。

- PercentProgress — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  オペレーションの進行状況をパーセンテージで指定します。

- Port - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  データベースエンジンがリッスンするポートを指定します。

- PreferredBackupWindow — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  `BackupRetentionPeriod` に応じた、自動バックアップが有効な場合に自動バックアップが作成される毎日の時間範囲を指定します。

- PreferredMaintenanceWindow — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  週 1 回のシステムメンテナンスを実行できる時間範囲を世界標準時 (UTC) で指定します。

- ReaderEndpoint — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターのリーダーエンドポイント。DB クラスターの読み込みエンドポイントは、DB クラスター内で使用できるリードレプリカ間で接続を負荷分散します。クライアントが読み取りエンド

ポイントへの新規接続をリクエストすると、Neptune によって接続リクエストが DB クラスターのリードレプリカ間で配分されます。この機能は、DB クラスターの複数のリードレプリカ間の読み取りワークロードを分散させる役に立ちます。

フェイルオーバーが発生し、接続しているリードレプリカがプライマリインスタンスに昇格すると、接続は削除されます。読み取りワークロードをクラスター内の他のリードレプリカに送信し続けるために、読み込みエンドポイントに再接続することができます。

- ReadReplicaIdentifiers — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターに関連付けられているリードレプリカの 1 つ以上の識別子を含みます。
- ReplicationSourceIdentifier — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  Neptune ではサポートされていません。
- ReplicationType — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  Neptune ではサポートされていません。
- ServerlessV2ScalingConfiguration – ServerlessV2ScalingConfigurationInfo オブジェクト。

  Neptune サーバーレス DB クラスターのスケーリング設定を表示します。

  詳細については、「Amazon Neptune ユーザーガイド」の 「Amazon Neptune サーバーレスの使用」を参照してください。
- Status — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターの現在の状態を指定します。
- StorageEncrypted - タイプ `boolean` のブール値 (ブール値 (真または偽))。

  DB クラスターが暗号化されているかどうかを指定します。
- StorageType — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターで使用するストレージタイプ。

  有効な値:

  - **`standard`** — (デフォルト) I/O 使用率が低いか中程度のアプリケーション向けに、費用対効果の高いデータベースストレージを提供します。
  - **`iopt1`** — 料金が予測可能で、低い I/O レイテンシーと一貫した I/O スループットを必要とする I/O 集約型のグラフワークロードのニーズを満たすように設計された I/O 最適化ストレージを有効にします。

Neptune I/O 最適化ストレージは、エンジンリリース 1.3.0.0 以降でのみ使用可能です。

- VpcSecurityGroups – VpcSecurityGroupMembership オブジェクトの配列。

  DB クラスターが属する VPC セキュリティグループのリストを入力します。

エラー

- DBClusterNotFoundFault
- InvalidDBClusterStateFault
- InvalidDBInstanceStateFault

PromoteReadReplicaDBCluster (アクション)

この API の AWS CLI 名は `promote-read-replica-db-cluster` です。

サポート外。

リクエスト

- DBClusterIdentifier (CLI では: `--db-cluster-identifier`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  サポート外。

レスポンス

Amazon Neptune DB クラスターの詳細が含まれています。

このデータ型は、the section called “DescribeDBClusters” でレスポンス要素として使用されます。

- AllocatedStorage - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  `AllocatedStorage` は常に 1 を返します。これは、Neptune DB クラスターのストレージサイズが固定されていないため、必要に応じて自動的に調整されるためです。
- AssociatedRoles – DBClusterRole オブジェクトの配列。

DB クラスターに関連付けられている Amazon Identity and Access Management (IAM) ロールのリストを入力します。DB クラスターに関連付けられている IAM ロールは、ユーザーに代わって他の Amazon のサービスにアクセスするための DB クラスターへのアクセス許可を付与します。

- AutomaticRestartTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  DB クラスターが自動的に再起動される時刻。

- AvailabilityZones — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターのインスタンスを作成できる EC2 アベイラビリティーゾーン (AZ) のリストを入力します。

- BacktrackConsumedChangeRecords — タイプ `long` の LongOptional (符号付き 64 ビット整数)。

  Neptune ではサポートされていません。

- BacktrackWindow — タイプ `long` の LongOptional (符号付き 64 ビット整数)。

  Neptune ではサポートされていません。

- BackupRetentionPeriod - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  自動 DB スナップショットが保持される日数を指定します。

- Capacity - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  Neptune ではサポートされていません。

- CloneGroupId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターが関連付けられているクローングループを識別します。

- ClusterCreateTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  DB クラスターが作成された時刻を協定世界時 (UTC) で指定します。

- CopyTagsToSnapshot — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽) の値)。

  `true` に設定されている場合、タグは、作成された DB クラスターのいずれかのスナップショットにコピーされます。

- CrossAccountClone — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽) の値)。

  `true`に設定されている場合、DB クラスターを複数のアカウント間でクローンできます。

- DatabaseName — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターの作成時に指定された場合は、作成時に提供されたこの DB クラスターの初期データベースの名前が入ります。DB クラスターの存続中はこれと同じ名前が返されます。
- DBClusterArn — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターの Amazon リソースネーム (ARN) を返します。
- DBClusterIdentifier — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  ユーザーが指定した DB クラスター識別子が含まれています。この識別子は、DB クラスターを識別する一意のキーです。
- DBClusterMembers – DBClusterMember オブジェクトの配列。

  DB クラスターを構成するインスタンスのリストを入力します。
- DBClusterParameterGroup — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターに使用する DB クラスターパラメータグループの名前を指定します。
- DbClusterResourceId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターの Amazon リージョン固有のイミュータブルな識別子。この識別子は、DB クラスターの Amazon KMS キーにアクセスするたびに Amazon CloudTrail ログエントリに記録されます。
- DBSubnetGroup — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  サブネットグループ内の名前、説明、サブネットなど、DB クラスターに関連付けられているサブネットグループに関する情報を指定します。
- DeletionProtection — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽) の値)。

  DB クラスターで削除保護が有効になっているかどうかを示します。削除保護が有効になっている場合、データベースを削除することはできません。
- EarliestBacktrackTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  Neptune ではサポートされていません。
- EarliestRestorableTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  特定時点への復元によりデータベースを復元できる最も早い時刻を指定します。

- EnabledCloudwatchLogsExports — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターが CloudWatch Logs にエクスポートするように設定されているログタイプのリスト。有効なログタイプは、`audit` (監査ログを CloudWatch に公開する場合) と slowquery (スロークエリログを CloudWatch に公開する場合) です。「Amazon CloudWatch Logs への Neptune ログの発行」を参照してください。

- Endpoint — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターのプライマリインスタンスの接続エンドポイントを指定します。

- Engine — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターに使用されるデータベースエンジンの名前を入力します。

- EngineVersion — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  データベースエンジンのバージョンを示します。

- GlobalClusterIdentifier — GlobalClusterIdentifier、Type: `string` (UTF-8 でエンコードされた文字列)、1 ~ 255、正規表現 `[A-Za-z][0-9A-Za-z-:._]*` に一致。

  ユーザーが指定したグローバルクラスター識別子を含みます。この識別子は、グローバルクラスターを識別する一意のキーです。

- HostedZoneId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  ホストゾーンを作成するときに Amazon Route 53 が割り当てる ID を指定します。

- IAMDatabaseAuthenticationEnabled — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  Amazon Identity and Access Management (IAM) アカウントのデータベースアカウントへのマッピングが有効な場合は true、それ以外の場合は false です。

- IOOptimizedNextAllowedModificationTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  次回は、`iopt1` ストレージタイプを使用するように DB クラスターを変更できます。

- KmsKeyId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  `StorageEncrypted` が true の場合、暗号化された DB クラスターの Amazon KMS キー識別子。

- LatestRestorableTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

特定時点への復元によりデータベースを復元できる最新の日時を指定します。

- MultiAZ — ブール値、タイプ: boolean (ブール値 (真または偽))。

  DB クラスターが複数のアベイラビリティーゾーンにインスタンスを持つかどうかを指定します。

- PendingModifiedValues – ClusterPendingModifiedValues オブジェクト。

  このデータ型は、ModifyDBCluster 操作でレスポンス要素として使用され、次のメンテナンスウィンドウに適用される変更を含みます。

- PercentProgress — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  オペレーションの進行状況をパーセンテージで指定します。

- Port - タイプ integer の IntegerOptional (符号付き 32 ビット整数)。

  データベースエンジンがリッスンするポートを指定します。

- PreferredBackupWindow — タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  BackupRetentionPeriod に応じた、自動バックアップが有効な場合に自動バックアップが作成される毎日の時間範囲を指定します。

- PreferredMaintenanceWindow — タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  週 1 回のシステムメンテナンスを実行できる時間範囲を世界標準時 (UTC) で指定します。

- ReaderEndpoint — タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターのリーダーエンドポイント。DB クラスターの読み込みエンドポイントは、DB クラスター内で使用できるリードレプリカ間で接続を負荷分散します。クライアントが読み取りエンドポイントへの新規接続をリクエストすると、Neptune によって接続リクエストが DB クラスターのリードレプリカ間で配分されます。この機能は、DB クラスターの複数のリードレプリカ間の読み取りワークロードを分散させる役に立ちます。

  フェイルオーバーが発生し、接続しているリードレプリカがプライマリインスタンスに昇格すると、接続は削除されます。読み取りワークロードをクラスター内の他のリードレプリカに送信し続けるために、読み込みエンドポイントに再接続することができます。

- ReadReplicaIdentifiers — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  この DB クラスターに関連付けられているリードレプリカの 1 つ以上の識別子を含みます。

- ReplicationSourceIdentifier — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

Neptune ではサポートされていません。

- ReplicationType — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  Neptune ではサポートされていません。

- ServerlessV2ScalingConfiguration – ServerlessV2ScalingConfigurationInfo オブジェクト。

  Neptune サーバーレス DB クラスターのスケーリング設定を表示します。

  詳細については、「Amazon Neptune ユーザーガイド」の 「Amazon Neptune サーバーレスの使用」を参照してください。

- Status — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターの現在の状態を指定します。

- StorageEncrypted - タイプ `boolean` のブール値 (ブール値 (真または偽))。

  DB クラスターが暗号化されているかどうかを指定します。

- StorageType — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターで使用するストレージタイプ。

  有効な値:

  - **`standard`** — (デフォルト) I/O 使用率が低いか中程度のアプリケーション向けに、費用対効果の高いデータベースストレージを提供します。
  - **`iopt1`** — 料金が予測可能で、低い I/O レイテンシーと一貫した I/O スループットを必要とする I/O 集約型のグラフワークロードのニーズを満たすように設計された I/O 最適化ストレージを有効にします。

    Neptune I/O 最適化ストレージは、エンジンリリース 1.3.0.0 以降でのみ使用可能です。

- VpcSecurityGroups – VpcSecurityGroupMembership オブジェクトの配列。

  DB クラスターが属する VPC セキュリティグループのリストを入力します。

エラー

- DBClusterNotFoundFault
- InvalidDBClusterStateFault

DescribeDBClusters (アクション)

この API の AWS CLI 名は `describe-db-clusters` です。

プロビジョニングされた DB クラスターに関する情報を返し、ページ分割をサポートします。

> **Note**
>
> このオペレーションは、Amazon RDS クラスターと Amazon DocDB クラスターの情報を返すこともできます。

リクエスト

- DBClusterIdentifier (CLI では: `--db-cluster-identifier`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  ユーザーが指定した DB クラスター識別子。このパラメータを指定した場合は、特定の DB クラスターからの情報だけが返されます。このパラメータでは大文字と小文字は区別されません。

  制約:

  - 指定した場合、既存の DBClusterIdentifier と一致する必要があります。
- Filters (CLI では: `--filters`) - フィルター オブジェクトの配列。

  記述する 1 つ以上の DB クラスターを指定するフィルター。

  サポートされているフィルター:

  - `db-cluster-id` - DB クラスター識別子と DB クラスターの Amazon リソースネーム (ARN) を受け入れます。結果リストには、これらの ARN によって識別された DB クラスターに関する情報のみが含まれます。
  - `engine` - エンジン名 (neptuneなど) を受け入れ、そのエンジンによって作成された DB クラスターに結果リストを制限します。

  たとえば、Amazon CLI からこの API を呼び出し、Neptune DB クラスターのみが返されるようにフィルタリングするには、次のコマンドを使用します。

  Example

```
aws neptune describe-db-clusters \
```

```
            --filters  Name=engine,Values=neptune
```

- Marker (CLI では: `--marker`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  以前の the section called “DescribeDBClusters” リクエストによって提供されたオプションのページ割りトークン。このパラメータを指定した場合、レスポンスには `MaxRecords` で指定された値まで、マーカーを超えるレコードのみが含まれます。

- MaxRecords (CLI では: `--max-records`) — タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  レスポンスに含めるレコードの最大数。指定された `MaxRecords` の値よりも多くのレコードが存在する場合、マーカーと呼ばれるページ割りトークンがレスポンスに含まれるため、残りの結果を取得できます。

  デフォルト: 100

  制約: 最小 20、最大 100。

レスポンス

- DBClusters – DBCluster オブジェクトの配列。

  ユーザーの DB クラスターのリストが含まれます。

- Marker — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  後続の DescribeDBClusters リクエストで使用できるページ分割トークン。

エラー

- DBClusterNotFoundFault

構造:

DBCluster (構造)

Amazon Neptune DB クラスターの詳細が含まれています。

このデータ型は、the section called “DescribeDBClusters” でレスポンス要素として使用されます。

フィールド

- AllocatedStorage — これはタイプ `integer` の IntegerOptional です (符号付き 32 ビット整数)。

  `AllocatedStorage` は常に 1 を返します。これは、Neptune DB クラスターのストレージサイズが固定されていないため、必要に応じて自動的に調整されるためです。

- AssociatedRoles — これは DBClusterRole オブジェクトの配列です。

  DB クラスターに関連付けられている Amazon Identity and Access Management (IAM) ロールのリストを入力します。DB クラスターに関連付けられている IAM ロールは、ユーザーに代わって他の Amazon のサービスにアクセスするための DB クラスターへのアクセス許可を付与します。

- AutomaticRestartTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  DB クラスターが自動的に再起動される時刻。

- AvailabilityZones — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  DB クラスターのインスタンスを作成できる EC2 アベイラビリティーゾーン (AZ) のリストを入力します。

- BacktrackConsumedChangeRecords — タイプ `long` の LongOptional (符号付き 64 ビット整数)。

  Neptune ではサポートされていません。

- BacktrackWindow — タイプ `long` の LongOptional (符号付き 64 ビット整数)。

  Neptune ではサポートされていません。

- BackupRetentionPeriod — これはタイプ `integer` の IntegerOptional です (符号付き 32 ビット整数)。

  自動 DB スナップショットが保持される日数を指定します。

- Capacity — これはタイプ `integer` の IntegerOptional です (符号付き 32 ビット整数)。

  Neptune ではサポートされていません。

- CloneGroupId — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  DB クラスターが関連付けられているクローングループを識別します。

- ClusterCreateTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

DB クラスターが作成された時刻を協定世界時 (UTC) で指定します。

- CopyTagsToSnapshot — これは BooleanOptional、タイプ: `boolean` です (ブール値 (真または偽))。

  `true` に設定されている場合、タグは、作成された DB クラスターのいずれかのスナップショットにコピーされます。

- CrossAccountClone — これは BooleanOptional、タイプ: `boolean` です (ブール値 (真または偽))。

  `true`に設定されている場合、DB クラスターを複数のアカウント間でクローンできます。

- DatabaseName — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  DB クラスターの作成時に指定された場合は、作成時に提供されたこの DB クラスターの初期データベースの名前が入ります。DB クラスターの存続中はこれと同じ名前が返されます。

- DBClusterArn — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  DB クラスターの Amazon リソースネーム (ARN) を返します。

- DBClusterIdentifier — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  ユーザーが指定した DB クラスター識別子が含まれています。この識別子は、DB クラスターを識別する一意のキーです。

- DBClusterMembers — これは DBClusterMember オブジェクトの配列です。

  DB クラスターを構成するインスタンスのリストを入力します。

- DBClusterParameterGroup — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  DB クラスターに使用する DB クラスターパラメータグループの名前を指定します。

- DbClusterResourceId — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  DB クラスターの Amazon リージョン固有のイミュータブルな識別子。この識別子は、DB クラスターの Amazon KMS キーにアクセスするたびに Amazon CloudTrail ログエントリに記録されます。

- DBSubnetGroup — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  サブネットグループ内の名前、説明、サブネットなど、DB クラスターに関連付けられているサブネットグループに関する情報を指定します。

- DeletionProtection — これは BooleanOptional、タイプ: `boolean` です (ブール値 (真または偽))。

  DB クラスターで削除保護が有効になっているかどうかを示します。削除保護が有効になっている場合、データベースを削除することはできません。
- EarliestBacktrackTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  Neptune ではサポートされていません。
- EarliestRestorableTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  特定時点への復元によりデータベースを復元できる最も早い時刻を指定します。
- EnabledCloudwatchLogsExports — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  この DB クラスターが CloudWatch Logs にエクスポートするように設定されているログタイプのリスト。有効なログタイプは、`audit` (監査ログを CloudWatch に公開する場合) と slowquery (スロークエリログを CloudWatch に公開する場合) です。「Amazon CloudWatch Logs への Neptune ログの発行」を参照してください。
- Endpoint — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  DB クラスターのプライマリインスタンスの接続エンドポイントを指定します。
- Engine — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  この DB クラスターに使用されるデータベースエンジンの名前を入力します。
- EngineVersion — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  データベースエンジンのバージョンを示します。
- GlobalClusterIdentifier — これは GlobalClusterIdentifier、タイプ `string`、(UTF-8 でエンコードされた文字列)、1 ~ 255、正規表現 `[A-Za-z][0-9A-Za-z-:._]*` に一致。

  ユーザーが指定したグローバルクラスター識別子を含みます。この識別子は、グローバルクラスターを識別する一意のキーです。
- HostedZoneId — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  ホストゾーンを作成するときに Amazon Route 53 が割り当てる ID を指定します。
- IAMDatabaseAuthenticationEnabled — これはブール値、タイプ: `boolean` です (ブール値 (真または偽))。

Amazon Identity and Access Management (IAM) アカウントのデータベースアカウントへのマッピングが有効な場合は true、それ以外の場合は false です。

- IOOptimizedNextAllowedModificationTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  次回は、`iopt1` ストレージタイプを使用するように DB クラスターを変更できます。

- KmsKeyId — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  `StorageEncrypted` が true の場合、暗号化された DB クラスターの Amazon KMS キー識別子。

- LatestRestorableTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  特定時点への復元によりデータベースを復元できる最新の日時を指定します。

- MultiAZ — これはブール値、タイプ: `boolean` です (ブール値 (真または偽))。

  DB クラスターが複数のアベイラビリティーゾーンにインスタンスを持つかどうかを指定します。

- PendingModifiedValues - これは ClusterPendingModifiedValues オブジェクトです。

  このデータ型は、`ModifyDBCluster` 操作でレスポンス要素として使用され、次のメンテナンスウィンドウに適用される変更を含みます。

- PercentProgress — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  オペレーションの進行状況をパーセンテージで指定します。

- Port — これはタイプ `integer` の IntegerOptional です (符号付き 32 ビット整数)。

  データベースエンジンがリッスンするポートを指定します。

- PreferredBackupWindow — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  `BackupRetentionPeriod` に応じた、自動バックアップが有効な場合に自動バックアップが作成される毎日の時間範囲を指定します。

- PreferredMaintenanceWindow — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  週 1 回のシステムメンテナンスを実行できる時間範囲を世界標準時 (UTC) で指定します。

- ReaderEndpoint — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

DB クラスターのリーダーエンドポイント。DB クラスターの読み込みエンドポイントは、DB クラスター内で使用できるリードレプリカ間で接続を負荷分散します。クライアントが読み取りエンドポイントへの新規接続をリクエストすると、Neptune によって接続リクエストが DB クラスターのリードレプリカ間で配分されます。この機能は、DB クラスターの複数のリードレプリカ間の読み取りワークロードを分散させる役に立ちます。

フェイルオーバーが発生し、接続しているリードレプリカがプライマリインスタンスに昇格すると、接続は削除されます。読み取りワークロードをクラスター内の他のリードレプリカに送信し続けるために、読み込みエンドポイントに再接続することができます。

- ReadReplicaIdentifiers — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  この DB クラスターに関連付けられているリードレプリカの 1 つ以上の識別子を含みます。
- ReplicationSourceIdentifier — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  Neptune ではサポートされていません。
- ReplicationType — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  Neptune ではサポートされていません。
- ServerlessV2ScalingConfiguration - これは ServerlessV2ScalingConfigurationInfo オブジェクトです。

  Neptune サーバーレス DB クラスターのスケーリング設定を表示します。

  詳細については、「Amazon Neptune ユーザーガイド」の 「Amazon Neptune サーバーレスの使用」を参照してください。
- Status — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  この DB クラスターの現在の状態を指定します。
- StorageEncrypted — これはブール値、タイプ: `boolean` です (ブール値 (真または偽))。

  DB クラスターが暗号化されているかどうかを指定します。
- StorageType — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  DB クラスターで使用するストレージタイプ。

  有効な値:

- **`standard`** — (デフォルト) I/O 使用率が低いか中程度のアプリケーション向けに、費用対効果の高いデータベースストレージを提供します。
- **`iopt1`** — 料金が予測可能で、低い I/O レイテンシーと一貫した I/O スループットを必要とする I/O 集約型のグラフワークロードのニーズを満たすように設計された I/O 最適化ストレージを有効にします。

  Neptune I/O 最適化ストレージは、エンジンリリース 1.3.0.0 以降でのみ使用可能です。
- VpcSecurityGroups — これは VpcSecurityGroupMembership オブジェクトの配列です。

  DB クラスターが属する VPC セキュリティグループのリストを入力します。

`DBCluster` は、以下のレスポンス要素として使用されます。

- CreateDBCluster
- DeleteDBCluster
- FailoverDBCluster
- ModifyDBCluster
- PromoteReadReplicaDBCluster
- RestoreDBClusterFromSnapshot
- RestoreDBClusterToPointInTime
- StartDBCluster
- StopDBCluster

DBClusterMember (構造)

DB クラスターの一部であるインスタンスに関する情報が含まれています。

フィールド

- DBClusterParameterGroupStatus — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  DB クラスターのこのメンバーの DB クラスターパラメータグループのステータスを指定します。
- DBInstanceIdentifier — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  DB クラスターのこのメンバーのインスタンス ID を指定します。

- IsClusterWriter — これはブール値、タイプ: `boolean` です (ブール値 (真または偽))。

  クラスターメンバーが DB クラスターのプライマリインスタンスの場合は `true`、それ以外の場合は `false` です。

- PromotionTier — これはタイプ `integer` の IntegerOptional です (符号付き 32 ビット整数)。

  既存のプライマリインスタンスの障害後に、リードレプリカをプライマリインスタンスに昇格される順序を指定する値。

DBClusterRole (構造)

DB クラスターに関連付けられている Amazon Identity and Access Management (IAM) ロールを記述します。

フィールド

- FeatureName — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  Amazon Identity and Access Management(IAM) ロールに関連付けられている機能の名前。サポートされている機能名のリストについては、「the section called “DescribeDBEngineVersions”」を参照してください 。

- RoleArn — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  DB クラスターに関連付けられる IAM ロール Amazon リソースネーム (ARN)。

- Status — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  IAM ロールと DB クラスターの間の関連付けの状態を説明します。Status プロパティは以下のいずれかの値を返します。

  - `ACTIVE` - IAM ロール ARN は DB クラスターに関連付けられており、代わりに他の Amazon のサービスにアクセスするために使用できます。
  - `PENDING` - IAM ロール ARN は DB クラスターに関連付けられています。
  - `INVALID` - IAM ロール ARN は DB クラスターに関連付けられていますが、DB クラスターはユーザーに代わって他の Amazon のサービスにアクセスするために IAM ロールを引き受けることができません。

CloudwatchLogsExportConfiguration (構造)

特定の DB インスタンスまたは DB クラスターの CloudWatch Logs にエクスポートするために有効にするログタイプの設定。

`EnableLogTypes` および `DisableLogTypes` の配列では、CloudWatch Logs にエクスポートする (または、エクスポートしない) ログを決定します。

有効なログタイプは、`audit` (監査ログを公開する場合) と `slowquery` (スロークエリログを公開する場合) です。「Amazon CloudWatch Logs への Neptune ログの発行」を参照してください。

フィールド

- DisableLogTypes — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  無効にするログタイプのリスト。
- EnableLogTypes — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  有効にするログタイプのリスト。

PendingCloudwatchLogsExports (構造)

設定がまだ保留中のログタイプのリスト。つまり、これらのログタイプは有効化中または無効化中です。

有効なログタイプは、`audit` (監査ログを公開する場合) と `slowquery` (スロークエリログを公開する場合) です。「Amazon CloudWatch Logs への Neptune ログの発行」を参照してください。

フィールド

- LogTypesToDisable — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  有効化中のログタイプ。有効にすると、これらのログタイプは CloudWatch Logs にエクスポートされます。
- LogTypesToEnable — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  無効化中のログタイプ。無効化されたログタイプは CloudWatch Logs にエクスポートされません。

ClusterPendingModifiedValues (構造)

このデータ型は、ModifyDBCluster 操作でレスポンス要素として使用され、次のメンテナンスウィンドウに適用される変更を含みます。

フィールド

- AllocatedStorage — これはタイプ integer の IntegerOptional です (符号付き 32 ビット整数)。

  データベースエンジンに割り当てられたストレージのサイズ (ギビバイト (GiB) 単位)。Neptune の場合、AllocatedStorage は常に 1 を返します。これは、Neptune DB クラスターのストレージサイズが固定されていず、必要に応じて自動的に調整されるためです。
- BackupRetentionPeriod — これはタイプ integer の IntegerOptional です (符号付き 32 ビット整数)。

  自動 DB スナップショットが保持される日数。
- DBClusterIdentifier — これはタイプ string の文字列です (UTF-8 でエンコードされた文字列)。

  DB クラスターの DBClusterIdentifier 値。
- EngineVersion — これはタイプ string の文字列です (UTF-8 でエンコードされた文字列)。

  データベースエンジンのバージョン。
- IAMDatabaseAuthenticationEnabled — これは BooleanOptional、タイプ: boolean です (ブール値 (真または偽))。

  AWS Identity and Access Management (IAM) アカウントのデータベースアカウントへのマッピングを有効にするかどうかを示す値。
- Iops — これはタイプ integer の IntegerOptional です (符号付き 32 ビット整数)。

  プロビジョンド IOPS (I/O operations per second) 値。この設定は、マルチ AZ DB クラスター専用です。
- PendingCloudwatchLogsExports - これは PendingCloudwatchLogsExports オブジェクトです。

  この PendingCloudwatchLogsExports 構造は、CloudWatch ログを有効および無効にする保留中の変更を指定します。
- StorageType — これはタイプ string の文字列です (UTF-8 でエンコードされた文字列)。

  DB クラスターのストレージタイプにおける保留中の変更。 有効な値:

- **`standard`** — (デフォルト) I/O 使用率が低いか中程度のアプリケーション向けに、費用対効果の高いデータベースストレージを設定します。
- **`iopt1`** — 料金が予測可能で、低い I/O レイテンシーと一貫した I/O スループットを必要とする I/O 集約型のグラフワークロードのニーズを満たすように設計された I/O 最適化ストレージを有効にします。

  Neptune I/O 最適化ストレージは、エンジンリリース 1.3.0.0 以降でのみ使用可能です。

Neptune グローバルデータベース API

アクション:

- CreateGlobalCluster (アクション)
- DeleteGlobalCluster (アクション)
- ModifyGlobalCluster (アクション)
- DescribeGlobalClusters (アクション)
- FailoverGlobalCluster (アクション)
- RemoveFromGlobalCluster (アクション)

構造:

- GlobalCluster (構造)
- GlobalClusterMember (構造)

CreateGlobalCluster (アクション)

この API の AWS CLI 名は `create-global-cluster` です。

複数の Amazon リージョンにまたがる Neptune グローバルデータベースを作成します。グローバルデータベースには、読み取り/書き込み機能を備えた 1 つのプライマリクラスターと、Neptune ストレージサブシステムによって実行される高速レプリケーションによってプライマリクラスターからデータを受信する読み取り専用のセカンダリクラスターが含まれます。

最初は空のグローバルデータベースを作成し、そのデータベースに、プライマリクラスターとセカンダリクラスターを追加できます。または、作成操作時にグローバルデータベースのプライマリクラスターになるように既存の Neptune クラスターを指定できます。

リクエスト

- DatabaseName (CLI では: `--database-name`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  新しいグローバルデータベースの名前 (英数字 64 文字以内)。

- DeletionProtection (CLI では: `--deletion-protection`) — タイプ `boolean` の BooleanOptional (ブール値 (真または偽) の値)。

  新しいグローバルデータベースの削除保護設定。削除保護が有効な場合、グローバルデータベースは削除できません。

- Engine (CLI では: `--engine`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  グローバルデータベースで使用されるデータベースエンジンの名前。

  有効な値: `neptune`

- EngineVersion (CLI では: `--engine-version`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  グローバルデータベースによって使用される Neptune エンジンのバージョン。

  有効な値は、`1.2.0.0` 以上です。

- GlobalClusterIdentifier (CLI では: `--global-cluster-identifier`) — 必須: GlobalClusterIdentifier、タイプ: `string` (UTF-8 でエンコードされた文字列)、1 以上または 255 以下、正規表現 `[A-Za-z][0-9A-Za-z-:._]*` に一致。

  新しいグローバルデータベースクラスターのクラスター識別子。

- SourceDBClusterIdentifier (CLI では: `--source-db-cluster-identifier`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  (オプション) 新しいグローバルデータベースのプライマリクラスターとして使用する既存の Neptune DB クラスターの Amazon リソースネーム (ARN)。

- StorageEncrypted (CLI では: `--storage-encrypted`) — タイプ `boolean` の BooleanOptional (ブール値 (真または偽) の値)。

  新しいグローバルデータベースクラスターのストレージ暗号化設定。

レスポンス

Amazon Neptune グローバルデータベースの詳細を含みます。

このデータ型は、the section called “CreateGlobalCluster”、the section called “DescribeGlobalClusters”、the section called “ModifyGlobalCluster”、the section called “DeleteGlobalCluster”、the section called “FailoverGlobalCluster”、および the section called “RemoveFromGlobalCluster” アクションのレスポンス要素として使用されます。

- DeletionProtection — BooleanOptional、タイプ:boolean (ブール値 (真または偽))。

  グローバルデータベースの削除保護設定。

- Engine — タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  グローバルデータベースによって使用される Neptune データベースエンジン ("neptune")。

- EngineVersion — タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  グローバルデータベースによって使用される Neptune エンジンのバージョン。

- GlobalClusterArn — タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  グローバルデータベースの Amazon リソースネーム (ARN)。

- GlobalClusterIdentifier — GlobalClusterIdentifier、Type: string (UTF-8 でエンコードされた文字列)、1～255、正規表現 [A-Za-z][0-9A-Za-z-:._]* に一致。

  ユーザーが指定したグローバルエータベースクラスター識別子を含みます。この識別子は、グローバルデータベースを識別する一意のキーです。

- GlobalClusterMembers – GlobalClusterMember オブジェクトの配列。

  グローバルデータベースに含まれるすべての DB クラスターのクラスター ARN とインスタンス ARN のリスト。

- GlobalClusterResourceId — タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  すべてのリージョンで一意な、グローバルデータベースのイミュータブルな識別子。この識別子は、DB クラスターの KMS キーにアクセスするたびに CloudTrail ログエントリに記録されます。

- Status — タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  このグローバルデータベースの現在の状態を指定します。

- StorageEncrypted — BooleanOptional、タイプ:boolean (ブール値 (真または偽))。

  グローバルデータベースのストレージ暗号化設定。

エラー

- GlobalClusterAlreadyExistsFault
- GlobalClusterQuotaExceededFault
- InvalidDBClusterStateFault
- DBClusterNotFoundFault

DeleteGlobalCluster (アクション)

この API の AWS CLI 名は `delete-global-cluster` です。

グローバルデータベースを削除します。プライマリクラスターとすべてのセカンダリクラスターは、最初にデタッチまたは削除されているする必要があります。

リクエスト

- GlobalClusterIdentifier (CLI では: `--global-cluster-identifier`) — 必須: GlobalClusterIdentifier、タイプ: `string` (UTF-8 でエンコードされた文字列)、1 以上または 255 以下、正規表現 `[A-Za-z][0-9A-Za-z-:._]*` に一致。

  削除されるグローバルデータベースクラスターのクラスター識別子。

レスポンス

Amazon Neptune グローバルデータベースの詳細を含みます。

このデータ型は、the section called “CreateGlobalCluster”、the section called “DescribeGlobalClusters”、the section called “ModifyGlobalCluster”、the section called “DeleteGlobalCluster”、the section called “FailoverGlobalCluster”、および the section called “RemoveFromGlobalCluster” アクションのレスポンス要素として使用されます。

- DeletionProtection — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽))。

  グローバルデータベースの削除保護設定。
- Engine — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  グローバルデータベースによって使用される Neptune データベースエンジン (`"neptune"`)。
- EngineVersion — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

グローバルデータベースによって使用される Neptune エンジンのバージョン。

- GlobalClusterArn — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  グローバルデータベースの Amazon リソースネーム (ARN)。

- GlobalClusterIdentifier — GlobalClusterIdentifier、Type: `string` (UTF-8 でエンコードされた文字列)、1～255、正規表現 `[A-Za-z][0-9A-Za-z-:._]*` に一致。

  ユーザーが指定したグローバルエータベースクラスター識別子を含みます。この識別子は、グローバルデータベースを識別する一意のキーです。

- GlobalClusterMembers – GlobalClusterMember オブジェクトの配列。

  グローバルデータベースに含まれるすべての DB クラスターのクラスター ARN とインスタンス ARN のリスト。

- GlobalClusterResourceId — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  すべてのリージョンで一意な、グローバルデータベースのイミュータブルな識別子。この識別子は、DB クラスターの KMS キーにアクセスするたびに CloudTrail ログエントリに記録されます。

- Status — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  このグローバルデータベースの現在の状態を指定します。

- StorageEncrypted — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽))。

  グローバルデータベースのストレージ暗号化設定。

エラー

- GlobalClusterNotFoundFault
- InvalidGlobalClusterStateFault

ModifyGlobalCluster (アクション)

この API の AWS CLI 名は `modify-global-cluster` です。

Amazon Neptune グローバルクラスターの設定を変更します。これらのパラメータと新しい値をリクエストで指定することで、1 つ以上のデータベース設定パラメータを変更できます。

リクエスト

- AllowMajorVersionUpgrade (CLI では: `--allow-major-version-upgrade`) — タイプ `boolean` の BooleanOptional (ブール値 (真または偽) の値)。

  メジャーバージョンアップグレードを許可するかどうかを示す値。

  制約: DB クラスターの現在のバージョンとは異なるメジャーバージョンである `EngineVersion` パラメータの値を指定する場合は、メジャーバージョンアップグレードを許可する必要があります。

  グローバルデータベースのメジャーバージョンをアップグレードする場合、クラスターと DB インスタンスのパラメータグループは新しいバージョンのデフォルトパラメータグループに設定されるため、アップグレードの完了後にカスタムパラメータグループを適用する必要があります。

- DeletionProtection (CLI では: `--deletion-protection`) — タイプ `boolean` の BooleanOptional (ブール値 (真または偽) の値)。

  グローバルデータベースの削除保護が有効になっているかどうかを示します。削除保護が有効な場合、グローバルデータベースは削除できません。

- EngineVersion (CLI では: `--engine-version`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  アップグレードするデータベースエンジンのバージョン番号。このパラメータを変更すると、機能停止が発生します。`ApplyImmediately` が有効でない限り、変更は次のメンテナンスウィンドウ時に適用されます。

  使用可能なエンジンバージョンをすべて一覧表示するには、次のコマンドを使用します。

  Example

  ```
  aws neptune describe-db-engine-versions \
            --engine neptune \
            --query '*[]|[?SupportsGlobalDatabases == 'true'].[EngineVersion]'
  ```

- GlobalClusterIdentifier (CLI では: `--global-cluster-identifier`) — 必須: GlobalClusterIdentifier、タイプ: `string` (UTF-8 でエンコードされた文字列)、1 以上または 255 以下、正規表現 `[A-Za-z][0-9A-Za-z-:._]*` に一致。

  変更されるグローバルクラスターの DB クラスター識別子。このパラメータは大文字と小文字が区別されません。

  既存のグローバルデータベースクラスターの識別子と一致する必要があります。

- NewGlobalClusterIdentifier (CLI では: `--new-global-cluster-identifier`) — GlobalClusterIdentifier、タイプ: `string`(UTF-8 でエンコードされた文字列)、1 以上または 255 以下、正規表現 `[A-Za-z][0-9A-Za-z-:._]*` に一致。

  グローバルデータベースに割り当てる新しいクラスター識別子。この値は小文字で保存されます。

  制約:

  - 1 ~ 63 の文字、数字またはハイフンを使用する必要があります。
  - 1 字目は文字である必要があります。
  - 文字列の最後にハイフンを使用したり、ハイフンを 2 つ続けて使用したりすることはできません。

  例: `my-cluster2`

レスポンス

Amazon Neptune グローバルデータベースの詳細を含みます。

このデータ型は、the section called “CreateGlobalCluster”、the section called “DescribeGlobalClusters”、the section called “ModifyGlobalCluster”、the section called “DeleteGlobalCluster”、the section called “FailoverGlobalCluster”、および the section called “RemoveFromGlobalCluster” アクションのレスポンス要素として使用されます。

- DeletionProtection — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽))。

  グローバルデータベースの削除保護設定。

- Engine — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  グローバルデータベースによって使用される Neptune データベースエンジン (`"neptune"`)。

- EngineVersion — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  グローバルデータベースによって使用される Neptune エンジンのバージョン。

- GlobalClusterArn — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  グローバルデータベースの Amazon リソースネーム (ARN)。

- GlobalClusterIdentifier — GlobalClusterIdentifier、Type: `string` (UTF-8 でエンコードされた文字列)、1 ~ 255、正規表現 `[A-Za-z][0-9A-Za-z-:._]*` に一致。

ユーザーが指定したグローバルエータベースクラスター識別子を含みます。この識別子は、グローバルデータベースを識別する一意のキーです。

- GlobalClusterMembers – GlobalClusterMember オブジェクトの配列。

  グローバルデータベースに含まれるすべての DB クラスターのクラスター ARN とインスタンス ARN のリスト。

- GlobalClusterResourceId — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  すべてのリージョンで一意な、グローバルデータベースのイミュータブルな識別子。この識別子は、DB クラスターの KMS キーにアクセスするたびに CloudTrail ログエントリに記録されます。

- Status — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  このグローバルデータベースの現在の状態を指定します。

- StorageEncrypted — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽))。

  グローバルデータベースのストレージ暗号化設定。

エラー

- GlobalClusterNotFoundFault
- InvalidGlobalClusterStateFault

DescribeGlobalClusters (アクション)

> この API の AWS CLI 名は `describe-global-clusters` です。

Neptune グローバルデータベースクラスターに関する情報を返します。この API はページ分割をサポートします。

リクエスト

- GlobalClusterIdentifier (CLI では: `--global-cluster-identifier`) — GlobalClusterIdentifier、タイプ: `string`(UTF-8 でエンコードされた文字列)、1 以上または 255 以下、正規表現 `[A-Za-z][0-9A-Za-z-:._]*` に一致。

  ユーザーが指定した DB クラスター識別子。このパラメータが指定された場合は、指定された DB クラスターに関する情報だけが返されます。このパラメータは大文字と小文字が区別されません。

制約: 指定された場合、既存の DB クラスター識別子と一致している必要があります。

- Marker (CLI では: `--marker`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  (オプション) `DescribeGlobalClusters` に対する以前の呼び出しから返されたページ割りトークン。このパラメータが指定された場合、レスポンスには `MaxRecords` で指定された値まで、マーカーを超えるレコードのみが含まれます。

- MaxRecords (CLI では: `--max-records`) — タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  レスポンスに含めるレコードの最大数。指定された `MaxRecords` の値よりも多くのレコードが存在する場合、ページ分割マーカートークンがレスポンスに含まれるため、それを使用して残りの結果を取得できます。

  デフォルト: `100`

  制約: 最小 20、最大 100。

レスポンス

- GlobalClusters – GlobalCluster オブジェクトの配列。

  このリクエストによって返されたグローバルクラスタとインスタンスのリスト。

- Marker — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  ページ分割トークン。このパラメータがレスポンスで返された場合、さらに多くのレコードが使用可能になり、`DescribeGlobalClusters` をさらに 1 回以上呼び出すことで取得できます。

エラー

- GlobalClusterNotFoundFault

FailoverGlobalCluster (アクション)

この API の AWS CLI 名は `failover-global-cluster` です。

Neptune グローバルデータベースのフェイルオーバープロセスを開始します。

Neptune グローバルデータベースのフェイルオーバーは、読み取り専用のセカンダリ DB クラスターの 1 つをプライマリ DB クラスターに昇格させ、プライマリ DB クラスターをセカンダリ (読み取り専用) DB クラスターに降格させます。つまり、現在のプライマリ DB クラスターと選択されたターゲットのセカンダリ DB クラスターの役割が入れ替わります。選択されたセカンダリ DB クラスターは、Neptune グローバルデータベースの完全な読み取り/書き込み機能を前提としています。

> **Note**
>
> このアクションは、Neptune グローバルデータベースにのみ適用されます。このアクションは、Neptune DB クラスターが正常で、リージョン全体で停止していない正常な Neptune グローバルデータベースでの使用、災害復旧シナリオのテスト、またはグローバルデータベーストポロジの再構成のみを目的としています。

リクエスト

- GlobalClusterIdentifier (CLI では: `--global-cluster-identifier`) — 必須: GlobalClusterIdentifier、タイプ: `string` (UTF-8 でエンコードされた文字列)、1 以上または 255 以下、正規表現 `[A-Za-z][0-9A-Za-z-:._]*` に一致。

  フェイルオーバーする必要がある Neptune グローバルデータベースの識別子。識別子は、Neptune グローバルデータベースの作成時にユーザーによって割り当てられた一意のキーです。つまり、フェイルオーバーするグローバルデータベースの名前です。

  制約: 既存の Neptune グローバルデータベースの識別子と一致する必要があります。
- TargetDbClusterIdentifier (CLI では: `--target-db-cluster-identifier`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  グローバルデータベースのプライマリに昇格するセカンダリ Neptune DB クラスターの Amazon リソースネーム (ARN)。

レスポンス

Amazon Neptune グローバルデータベースの詳細を含みます。

このデータ型は、the section called “CreateGlobalCluster”、the section called “DescribeGlobalClusters”、the section called “ModifyGlobalCluster”、the section called “DeleteGlobalCluster”、the section called “FailoverGlobalCluster”、および the section called “RemoveFromGlobalCluster” アクションのレスポンス要素として使用されます。

- DeletionProtection — BooleanOptional、タイプ:boolean (ブール値 (真または偽))。

  グローバルデータベースの削除保護設定。

- Engine — タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  グローバルデータベースによって使用される Neptune データベースエンジン ("neptune")。

- EngineVersion — タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  グローバルデータベースによって使用される Neptune エンジンのバージョン。

- GlobalClusterArn — タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  グローバルデータベースの Amazon リソースネーム (ARN)。

- GlobalClusterIdentifier — GlobalClusterIdentifier、Type: string (UTF-8 でエンコードされた文字列)、1 ~ 255、正規表現 [A-Za-z][0-9A-Za-z-:._]* に一致。

  ユーザーが指定したグローバルエータベースクラスター識別子を含みます。この識別子は、グローバルデータベースを識別する一意のキーです。

- GlobalClusterMembers – GlobalClusterMember オブジェクトの配列。

  グローバルデータベースに含まれるすべての DB クラスターのクラスター ARN とインスタンス ARN のリスト。

- GlobalClusterResourceId — タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  すべてのリージョンで一意な、グローバルデータベースのイミュータブルな識別子。この識別子は、DB クラスターの KMS キーにアクセスするたびに CloudTrail ログエントリに記録されます。

- Status — タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  このグローバルデータベースの現在の状態を指定します。

- StorageEncrypted — BooleanOptional、タイプ:boolean (ブール値 (真または偽))。

  グローバルデータベースのストレージ暗号化設定。

エラー

- GlobalClusterNotFoundFault
- InvalidGlobalClusterStateFault
- InvalidDBClusterStateFault

- DBClusterNotFoundFault

RemoveFromGlobalCluster (アクション)

この API の AWS CLI 名は `remove-from-global-cluster` です。

Neptune DB クラスターを Neptune グローバルデータベースからデタッチします。セカンダリクラスターは、読み取り専用の代わりに読み書き機能を備えた通常のスタンドアロンクラスターになり、プライマリクラスターからデータを受信しなくなります。

リクエスト

- DbClusterIdentifier (CLI では: `--db-cluster-identifier`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  Neptune グローバルデータベースクラスターからデタッチされるクラスターを識別する Amazon リソースネーム (ARN)。
- GlobalClusterIdentifier (CLI では: `--global-cluster-identifier`) — 必須: GlobalClusterIdentifier、タイプ: `string` (UTF-8 でエンコードされた文字列)、1 以上または 255 以下、正規表現 `[A-Za-z][0-9A-Za-z-:._]*` に一致。

  指定された Neptune DB クラスターをデタッチする Neptune グローバルデータベースの識別子。

レスポンス

Amazon Neptune グローバルデータベースの詳細を含みます。

このデータ型は、the section called “CreateGlobalCluster”、the section called “DescribeGlobalClusters”、the section called “ModifyGlobalCluster”、the section called “DeleteGlobalCluster”、the section called “FailoverGlobalCluster”、および the section called “RemoveFromGlobalCluster” アクションのレスポンス要素として使用されます。

- DeletionProtection — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽))。

  グローバルデータベースの削除保護設定。
- Engine — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  グローバルデータベースによって使用される Neptune データベースエンジン (`"neptune"`)。
- EngineVersion — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

グローバルデータベースによって使用される Neptune エンジンのバージョン。

- GlobalClusterArn — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  グローバルデータベースの Amazon リソースネーム (ARN)。

- GlobalClusterIdentifier — GlobalClusterIdentifier、Type: `string` (UTF-8 でエンコードされた文字列)、1 ~ 255、正規表現 `[A-Za-z][0-9A-Za-z-:._]*` に一致。

  ユーザーが指定したグローバルエータベースクラスター識別子を含みます。この識別子は、グローバルデータベースを識別する一意のキーです。

- GlobalClusterMembers – GlobalClusterMember オブジェクトの配列。

  グローバルデータベースに含まれるすべての DB クラスターのクラスター ARN とインスタンス ARN のリスト。

- GlobalClusterResourceId — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  すべてのリージョンで一意な、グローバルデータベースのイミュータブルな識別子。この識別子は、DB クラスターの KMS キーにアクセスするたびに CloudTrail ログエントリに記録されます。

- Status — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  このグローバルデータベースの現在の状態を指定します。

- StorageEncrypted — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽))。

  グローバルデータベースのストレージ暗号化設定。

エラー

- GlobalClusterNotFoundFault
- InvalidGlobalClusterStateFault
- DBClusterNotFoundFault

構造:

GlobalCluster (構造)

Amazon Neptune グローバルデータベースの詳細を含みます。

このデータ型は、the section called “CreateGlobalCluster”、the section called “DescribeGlobalClusters”、the section called “ModifyGlobalCluster”、the section called “DeleteGlobalCluster”、the section called “FailoverGlobalCluster”、および the section called “RemoveFromGlobalCluster” アクションのレスポンス要素として使用されます。

フィールド

- DeletionProtection — BooleanOptional、タイプ: `boolean` (ブール値 (真または偽))。

  グローバルデータベースの削除保護設定。
- Engine — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  グローバルデータベースによって使用される Neptune データベースエンジン (`"neptune"`)。
- EngineVersion — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  グローバルデータベースによって使用される Neptune エンジンのバージョン。
- GlobalClusterArn — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  グローバルデータベースの Amazon リソースネーム (ARN)。
- GlobalClusterIdentifier — GlobalClusterIdentifier、タイプ `string` (UTF-8 でエンコードされた文字列)、1 ~ 255 バイト長、正規表現 `[A-Za-z][0-9A-Za-z-:._]*` に一致。

  ユーザーが指定したグローバルエータベースクラスター識別子を含みます。この識別子は、グローバルデータベースを識別する一意のキーです。
- GlobalClusterMembers — これは GlobalClusterMember オブジェクトの配列です。

  グローバルデータベースに含まれるすべての DB クラスターのクラスター ARN とインスタンス ARN のリスト。
- GlobalClusterResourceId — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  すべてのリージョンで一意な、グローバルデータベースのイミュータブルな識別子。この識別子は、DB クラスターの KMS キーにアクセスするたびに CloudTrail ログエントリに記録されます。
- Status — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  このグローバルデータベースの現在の状態を指定します。
- StorageEncrypted — BooleanOptional、タイプ: `boolean` (ブール値 (真または偽))。

  グローバルデータベースのストレージ暗号化設定。

GlobalCluster は、以下のレスポンス要素として使用されます。

- CreateGlobalCluster
- ModifyGlobalCluster
- DeleteGlobalCluster
- RemoveFromGlobalCluster
- FailoverGlobalCluster

GlobalClusterMember (構造)

Neptune グローバルデータベースに関連付けられているプライマリおよびセカンダリクラスターに関する情報を含むデータ構造。

フィールド

- DBClusterArn — これはタイプ string の文字列です (UTF-8 でエンコードされた文字列)。

  各 Neptune クラスターの Amazon リソースネーム (ARN)。
- IsWriter — ブール値、タイプ: boolean (ブール値 (真または偽))。

  Neptune クラスターが、関連付けられている Neptune グローバルデータベースのプライマリクラスター (すなわち、読み取り/書き込み機能を持つ) かどうかを指定します。
- Readers — これはタイプ string の文字列です (UTF-8 でエンコードされた文字列)。

  Neptune グローバルデータベースに関連付けられている各読み取り専用セカンダリクラスターの Amazon リソースネーム (ARN)。

Neptune インスタンス API

アクション:

- CreateDBInstance (アクション)
- DeleteDBInstance (アクション)
- ModifyDBInstance (アクション)
- RebootDBInstance (アクション)
- DescribeDBInstances (アクション)

- DescribeOrderableDBInstanceOptions (アクション)
- DescribeValidDBInstanceModifications (アクション)

構造:

- DBInstance (構造)
- DBInstanceStatusInfo (構造)
- OrderableDBInstanceOption (構造)
- PendingModifiedValues (構造)
- ValidStorageOptions (構造)
- ValidDBInstanceModificationsMessage (構造)

CreateDBInstance (アクション)

この API の AWS CLI 名は `create-db-instance` です。

新しい DB インスタンスを作成します。

リクエスト

- AutoMinorVersionUpgrade (CLI では: `--auto-minor-version-upgrade`) — タイプ `boolean` の BooleanOptional (ブール値 (真または偽))。

  メンテナンスの時間帯に、DB インスタンスに自動的にマイナーエンジンアップグレードが適用されることを示します。

  デフォルト: `true`
- AvailabilityZone (CLI では: `--availability-zone`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  DB インスタンスが作成される EC2 アベイラビリティーゾーン。

  デフォルト: エンドポイントの Amazon リージョン内でランダムにシステムが選択したアベイラビリティーゾーン。

  例: `us-east-1d`

制約: MultiAZ パラメータが `true` に設定されている場合、アベイラビリティーゾーンパラメータは指定できません。指定されたアベイラビリティーゾーンは、現在のエンドポイントと同じ Amazon リージョンにある必要があります。

- BackupRetentionPeriod (CLI では: `--backup-retention-period`) — タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  自動バックアップを保管する日数。

  該当しません。自動バックアップの保持期間は、DB クラスターによって管理されます。詳細については、「the section called “CreateDBCluster”」を参照してください。

  デフォルト: 1

  制約:

  - 0 ~ 35 の値にする必要があります。
  - DB インスタンスがリードレプリカのソースである場合、0 に設定することはできません。
- CopyTagsToSnapshot (CLI では: `--copy-tags-to-snapshot`) — タイプ `boolean` の BooleanOptional (ブール値 (真または偽))。

  DB インスタンスから DB インスタンスのスナップショットにすべてのタグをコピーする場合は true、それ以外の場合は false です。デフォルトは [False] (偽) です。
- DBClusterIdentifier (CLI では: `--db-cluster-identifier`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  インスタンスが属する DB クラスターの識別子。

  DB クラスターの作成については、「the section called “CreateDBCluster”」を参照してください。

  型: 文字列
- DBInstanceClass (CLI では: `--db-instance-class`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスのコンピューティング性能とメモリ容量 (例: `db.m4.large`)。すべての DB インスタンスクラスがすべての Amazon リージョンで使用できるわけではありません。
- DBInstanceIdentifier (CLI では: `--db-instance-identifier`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンス識別子。このパラメータは小文字で保存されます。

制約:

- 1 ~ 63 の文字、数字またはハイフンを使用する必要があります。
- 1 字目は英字である必要があります。
- ハイフンを、文字列の最後に使用したり、2 つ続けて使用したりすることはできません。

例: `mydbinstance`

- DBName (CLI では: `--db-name`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  サポート外。

- DBParameterGroupName (CLI では: `--db-parameter-group-name`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  この DB インスタンスに関連付ける DB パラメータグループの名前。この引数を省略すると、指定したエンジンのデフォルトの DBParameterGroup が使用されます。

  制約:

  - 1 ~ 255 の英字、数字、ハイフンである必要があります。
  - 1 字目は英字である必要があります。
  - ハイフンを、文字列の最後に使用したり、2 つ続けて使用したりすることはできません
- DBSecurityGroups (CLI では: `--db-security-groups`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB インスタンスに関連付ける DB セキュリティグループの一覧。

  デフォルト: データベースエンジンのデフォルトの DB セキュリティグループ。

- DBSubnetGroupName (CLI では: `--db-subnet-group-name`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  この DB インスタンスに関連付ける DB サブネットグループ。

  DB サブネットグループがない場合、それは VPC DB インスタンスではありません。

- DeletionProtection (CLI では: `--deletion-protection`) — タイプ `boolean` の BooleanOptional (ブール値 (真または偽))。

DB インスタンスで削除保護が有効になっているかどうかを示す値。削除保護が有効になっている場合、データベースを削除することはできません。デフォルトでは、削除保護は無効です。「DB インスタンスを削除する」を参照してください。

DB クラスターの DB インスタンスは、親 DB クラスターで削除保護が有効になっている場合でも削除できます。

- Domain (CLI では: `--domain`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  インスタンスを作成する Active Directory ドメインを指定します。

- DomainIAMRoleName (CLI では: `--domain-iam-role-name`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  ディレクトリサービスへの API 呼び出しを行うときに使用される IAM ロールの名前を指定します。

- EnableCloudwatchLogsExports (CLI では: `--enable-cloudwatch-logs-exports`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  CloudWatch Logs にエクスポートするために有効にする必要があるログタイプのリスト。

- EnableIAMDatabaseAuthentication (CLI では: `--enable-iam-database-authentication`) — タイプ `boolean` の BooleanOptional (ブール値 (真または偽))。

  Neptune ではサポートされていません (無視)。

- Engine (CLI では: `--engine`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  このインスタンスに使用するデータベースエンジンの名前。

  有効な値: `neptune`

- EngineVersion (CLI では: `--engine-version`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  使用するデータベースエンジンのバージョン番号。現在、このパラメータを設定しても効果はありません。

- Iops (CLI では: `--iops`) — タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  初めに DB インスタンスに割り当てられるプロビジョンド IOPS (1 秒あたりの入力/出力操作数) の合計です。

- KmsKeyId (CLI では: `--kms-key-id`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  暗号化された DB インスタンスの Amazon KMS キー識別子。

  KMS キー識別子は、KMS 暗号化キーの Amazon リソースネーム (ARN) です。新しい DB インスタンスの暗号化に使用する KMS 暗号化キーを所有する Amazon アカウントと同じアカウントを使用して DB インスタンスを作成する場合、KM 暗号化キーの ARN の代わりに KMS キーのエイリアスを使用できます。

  該当しません。KMS キー識別子は、DB クラスターによって管理されます。詳細については、「the section called “CreateDBCluster”」を参照してください。

  `StorageEncrypted` パラメータが true の場合で `KmsKeyId` パラメータの値を指定しない場合、Amazon Neptune はデフォルトの暗号化キーを使用します。Amazon KMS は、Amazon アカウント用にデフォルトの暗号化キーを作成します。Amazon アカウントには、Amazon リージョンごとにデフォルトの暗号化キーがあります。

- LicenseModel (CLI では: `--license-model`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  この DB インスタンスのライセンスモデル情報。

  有効な値: `license-included` | `bring-your-own-license` | `general-public-license`

- MonitoringInterval (CLI では: `--monitoring-interval`) — タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  DB インスタンスについて拡張モニターメトリクスが収集される時点間の間隔 (秒単位)。拡張モニタリングメトリクスの収集を無効にするには、0 を指定します。デフォルトは 0 です。

  `MonitoringRoleArn` を指定した場合は、`MonitoringInterval` を 0 以外の値に設定する必要があります。

  有効な値: `0, 1, 5, 10, 15, 30, 60`

- MonitoringRoleArn (CLI では: `--monitoring-role-arn`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  Neptune が拡張モニタリングメトリクスを Amazon CloudWatch Logs に送信することを許可する IAM ロールの ARN。例えば、`arn:aws:iam:123456789012:role/emaccess` です。

MonitoringInterval を 0 以外に設定した場合は、MonitoringRoleArn 値を設定する必要があります。

- MultiAZ (CLI では: --multi-az) — タイプ boolean の BooleanOptional (ブール値 (真または偽))。

  DB インスタンスが Multi-AZ デプロイかどうかを指定します。MultiAZ パラメータが true に設定されている場合は、アベイラビリティーゾーンパラメータを設定できません。

- Port (CLI では: --port) — タイプ integer の IntegerOptional (符号付き 32 ビット整数)。

  データベースが接続を受け入れるポート番号。

  該当しません。ポートは、DB クラスターによって管理されます。詳細については、「the section called “CreateDBCluster”」を参照してください。

  デフォルト: 8182

  型: 整数

- PreferredBackupWindow (CLI では: --preferred-backup-window) — タイプ: string (UTF-8 でエンコードされた文字列) の文字列。

  自動バックアップが作成される毎日の時間範囲。

  該当しません。自動バックアップを作成するための毎日の時間範囲は、DB クラスターによって管理されます。詳細については、「the section called “CreateDBCluster”」を参照してください。

- PreferredMaintenanceWindow (CLI では: --preferred-maintenance-window) — タイプ: string (UTF-8 でエンコードされた文字列) の文字列。

  毎週 1 回のシステムメンテナンスを実行できる時間帯、世界標準時 (UTC)。

  形式: ddd:hh24:mi-ddd:hh24:mi

  デフォルトは、1 週間のうちのランダムな日に起こる、Amazon リージョンあたり 8 時間の範囲からランダムに選択された 30 分の時間窓です。

  有効な日数: Mon, Tue, Wed, Thu, Fri, Sat, Sun です。

  制約: 最小限の 30 分単位のウィンドウ。

- PromotionTier (CLI では: --promotion-tier) — タイプ integer の IntegerOptional (符号付き 32 ビット整数)。

既存のプライマリインスタンスの障害後に、リードレプリカをプライマリインスタンスに昇格される順序を指定する値。

デフォルト: 1

有効な値: 0 ~ 15

- PubliclyAccessible (CLI では: `--publicly-accessible`) — タイプ `boolean` の BooleanOptional (ブール値 (真または偽))。

  このフラグは、今後は使用しないでください。

- StorageEncrypted (CLI では: `--storage-encrypted`) — タイプ `boolean` の BooleanOptional (ブール値 (真または偽))。

  DB インスタンスが暗号化されているかどうかを指定します。

  該当しません。DB インスタンスの暗号化は、DB クラスターによって管理されます。詳細については、「the section called “CreateDBCluster”」を参照してください。

  デフォルト: false

- StorageType (CLI では: `--storage-type`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  該当しません。Neptune の場合、ストレージタイプは DB クラスターレベルで管理されます。

- Tags (CLI では: `--tags`) - Tag オブジェクトの配列。

  新しいインスタンスに割り当てるタグ。

- TdeCredentialArn (CLI では: `--tde-credential-arn`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  TDE 暗号化のためにインスタンスに関連付けるキーストアからの ARN。

- TdeCredentialPassword (CLI の場合は `--tde-credential-password`) — `string` 型の SensitiveString (UTF-8 でエンコードされた文字列)。

  デバイスにアクセスするためのキーストアからの指定された ARN のパスワード。

- Timezone (CLI では: `--timezone`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスのタイムゾーン。

- VpcSecurityGroupIds (CLI では: `--vpc-security-group-ids`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  この DB インスタンスに関連付ける EC2 VPC セキュリティグループの一覧。

  該当しません。EC2 VPC セキュリティグループの関連リストは、DB クラスターによって管理されます。詳細については、「the section called “CreateDBCluster”」を参照してください。

  デフォルト: DB サブネットグループの VPC に対するデフォルトの EC2 VPC セキュリティグループ。

レスポンス

Amazon Neptune DB インスタンスの詳細が含まれています。

このデータ型は、the section called “DescribeDBInstances” アクションのレスポンス要素として使用されます。

- AutoMinorVersionUpgrade — タイプ `boolean` のブール値 (真または偽)。

  マイナーバージョンのパッチを自動的に適用することを指定します。
- AvailabilityZone — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスを配置するアベイラビリティーゾーンの名前を指定します。
- BackupRetentionPeriod - タイプ `integer` の整数 (符号付き 32 ビット整数)。

  自動 DB スナップショットが保持される日数を指定します。
- CACertificateIdentifier — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB インスタンスの CA 証明書の識別子。
- CopyTagsToSnapshot — タイプ `boolean` のブール値 (真または偽)。

  タグを DB インスタンスから DB インスタンスのスナップショットにコピーするかどうかを指定します。
- DBClusterIdentifier — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスが DB クラスターのメンバーである場合は、DB インスタンスがメンバーとなっている DB クラスターの名前が含まれます。
- DBInstanceArn — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

DB インスタンスの Amazon リソースネーム (ARN)。

- DBInstanceClass — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスのコンピューティング名とメモリ容量クラスの名前が含まれています。

- DBInstanceIdentifier — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  ユーザーが指定したデータベース識別子が含まれています。この識別子は、DB インスタンスを識別する一意のキーです。

- DbInstancePort - タイプ `integer` の整数 (符号付き 32 ビット整数)。

  DB インスタンスがリッスンするポートを指定します。DB インスタンスが DB クラスターの一部である場合は、DB クラスターのポートとは別のポートを指定できます。

- DBInstanceStatus — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  このデータベースの現在の状態を指定します。

- DbiResourceId — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスの Amazon リージョン固有のイミュータブルな識別子。この識別子は、DB インスタンスの Amazon KMS キーにアクセスするたびに Amazon CloudTrail ログエントリに記録されます。

- DBName — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  データベース名。

- DBParameterGroups – DBParameterGroupStatus オブジェクトの配列。

  この DB インスタンスに適用される DB パラメータグループのリストを入力します。

- DBSecurityGroups – DBSecurityGroupMembership オブジェクトの配列。

  `DBSecurityGroup.Name` と `DBSecurityGroup.Status` のサブ要素のみを含む DB セキュリティグループ要素の一覧を入力します。

- DBSubnetGroup – DBSubnetGroup オブジェクト。

  サブネットグループ内の名前、説明、サブネットなど、DB インスタンスに関連付けられているサブネットグループに関する情報を指定します。

- DeletionProtection — タイプ `boolean` の BooleanOptional (ブール値 (真または偽))。

DB インスタンスで削除保護が有効になっているかどうかを示します。削除保護が有効な場合、インスタンスは削除できません。「DB インスタンスを削除する」を参照してください。

- DomainMemberships – DomainMembership オブジェクトの配列。

  サポートされません

- EnabledCloudwatchLogsExports — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB インスタンスが CloudWatch Logs にエクスポートするように設定されているログタイプのリスト。

- Endpoint – エンドポイント オブジェクト。

  接続エンドポイントを指定します。

- Engine — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB インスタンスに使用されるデータベースエンジンの名前を提供します。

- EngineVersion — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  データベースエンジンのバージョンを示します。

- EnhancedMonitoringResourceArn — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスの拡張モニタリングメトリクスデータを受け取る Amazon CloudWatch Logs ログストリームの Amazon リソースネーム (ARN)。

- IAMDatabaseAuthenticationEnabled — タイプ `boolean` のブール値 (真または偽)。

  Amazon Identity and Access Management (IAM) 認証が有効な場合は true、それ以外の場合は false です。

- InstanceCreateTime — TStamp、タイプ `timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点) 。

  DB インスタンスが作成された日時を入力します。

- Iops - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  プロビジョンド IOPS(I/O operations per second) を指定します。

- KmsKeyId — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

サポートなし: DB インスタンスの暗号化は、DB クラスターによって管理されます。

- LatestRestorableTime — TStamp、タイプ `timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点) 。

  特定時点への復元によりデータベースを復元できる最新の日時を指定します。

- LicenseModel — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB インスタンスのライセンスモデル情報。

- MonitoringInterval - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  DB インスタンスについて拡張モニターメトリクスが収集される時点間の間隔 (秒単位)。

- MonitoringRoleArn — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  Neptune が拡張モニタリングメトリクスを Amazon CloudWatch Logs に送信することを許可する IAM ロールの ARN。

- MultiAZ — タイプ `boolean` のブール値 (真または偽)。

  DB インスタンスが Multi-AZ デプロイかどうかを指定します。

- PendingModifiedValues – PendingModifiedValues オブジェクト。

  DB インスタンスへの変更が保留中であることを指定します。この要素は、変更が保留中の場合にのみ含まれます。特定の変更は、サブエレメントによって識別されます。

- PreferredBackupWindow — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  `BackupRetentionPeriod` に応じた、自動バックアップが有効な場合に自動バックアップが作成される毎日の時間範囲を指定します。

- PreferredMaintenanceWindow — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  週 1 回のシステムメンテナンスを実行できる時間範囲を世界標準時 (UTC) で指定します。

- PromotionTier - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  既存のプライマリインスタンスの障害後に、リードレプリカをプライマリインスタンスに昇格される順序を指定する値。

- PubliclyAccessible — タイプ `boolean` のブール値 (真または偽)。

  このフラグは、今後は使用しないでください。

- ReadReplicaDBClusterIdentifiers — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB インスタンスに関連付けられているリードレプリカの 1 つ以上の DB クラスターの識別子を含みます。

- ReadReplicaDBInstanceIdentifiers — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB インスタンスに関連付けられているリードレプリカの 1 つ以上の識別子を含みます。

- ReadReplicaSourceDBInstanceIdentifier — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB インスタンスがリードレプリカの場合は、ソース DB インスタンスの識別子が含まれます。

- SecondaryAvailabilityZone — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  存在する場合は、マルチ AZ をサポートする DB インスタンスのセカンダリアベイラビリティーゾーンの名前を指定します。

- StatusInfos – DBInstanceStatusInfo オブジェクトの配列。

  リードレプリカのステータス。インスタンスがリードレプリカではない場合、これは空白です。

- StorageEncrypted — タイプ `boolean` のブール値 (真または偽)。

  サポートなし: DB インスタンスの暗号化は、DB クラスターによって管理されます。

- StorageType — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスに関連付けられたストレージタイプを指定します。

- TdeCredentialArn — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  インスタンスが TDE 暗号化のために関連付けられているキーストアからの ARN。

- Timezone — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  サポート外。

- VpcSecurityGroups – VpcSecurityGroupMembership オブジェクトの配列。

  DB インスタンスが属する VPC セキュリティグループ要素のリストを入力します。

エラー

- DBInstanceAlreadyExistsFault
- InsufficientDBInstanceCapacityFault
- DBParameterGroupNotFoundFault
- DBSecurityGroupNotFoundFault
- InstanceQuotaExceededFault
- StorageQuotaExceededFault
- DBSubnetGroupNotFoundFault
- DBSubnetGroupDoesNotCoverEnoughAZs
- InvalidDBClusterStateFault
- InvalidSubnet
- InvalidVPCNetworkStateFault
- ProvisionedIopsNotAvailableInAZFault
- OptionGroupNotFoundFault
- DBClusterNotFoundFault
- StorageTypeNotSupportedFault
- AuthorizationNotFoundFault
- KMSKeyNotAccessibleFault
- DomainNotFoundFault

DeleteDBInstance (アクション)

この API の AWS CLI 名は `delete-db-instance` です。

DeleteDBInstance アクションでは、以前にプロビジョニングされた DB インスタンスを削除します。DB インスタンスを削除すると、そのインスタンスの自動バックアップはすべて削除され、復旧することはできません。`DeleteDBInstance` によって削除される DB インスタンスの手動 DB スナップショットは削除されません。

最終的な DB スナップショットをリクエストした場合、Amazon Neptune DB インスタンスのステータスは、DB スナップショットが作成されるまで `deleting` になります。API アクション `DescribeDBInstance` は、このオペレーションのステータスを監視するために使用されます。一度送信したアクションをキャンセルまたは元に戻すことはできません。

DB インスタンスが障害状態にあり、ステータスが failed、incompatible-restore、または incompatible-network である場合、そのインスタンスを削除できるのは SkipFinalSnapshot パラメータが true に設定されている場合のみであることに注意してください。

DB インスタンスが DB クラスター内の唯一のインスタンスの場合、または削除保護が有効になっている場合、その DB インスタンスを削除することはできません。

リクエスト

- DBInstanceIdentifier (CLI では: --db-instance-identifier) — 必須: タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  削除する DB インスタンスの DB インスタンス識別子。このパラメータでは大文字と小文字は区別されません。

  制約:

  - 既存の DB インスタンスの名前と一致する必要があります。
- FinalDBSnapshotIdentifier (CLI では: --final-db-snapshot-identifier) — タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  SkipFinalSnapshot が false に設定されている場合に作成される新しい DBSnapshot の DBSnapshotIdentifier。

  > **Note**
  >
  > このパラメータを指定して SkipFinalShapshot パラメータを true に設定すると、エラーが発生します。

  制約:

  - 1 ~ 255 文字の英字または数字でなければなりません。
  - 1 字目は英字である必要があります。
  - ハイフンを、文字列の最後に使用したり、2 つ続けて使用したりすることはできません
  - リードレプリカの削除時に指定することはできません。
- SkipFinalSnapshot (CLI では: --skip-final-snapshot) — タイプ boolean のブール値 (ブール値 (真または偽))。

DB インスタンスの削除前に最終 DB スナップショットを作成するかどうかを決定します。`true` を指定した場合、DB スナップショットが作成されます。`false` が指定されている場合、DB インスタンスの削除前に DB スナップショットが作成されます。

DB インスタンスが障害状態にあり、ステータスが「失敗」、「互換性のない復元」、または「互換性のないネットワーク」の場合、SkipFinalSnapshot パラメータが「true」に設定されている場合にのみ削除できます。

リードレプリカを削除するときは、`true` を指定してください。

> **Note**
>
> SkipFinalSnapshot が `false` の場合は、FinalDBSnapshotIdentifier パラメータを指定する必要があります。

デフォルト: `false`

レスポンス

Amazon Neptune DB インスタンスの詳細が含まれています。

このデータ型は、the section called “DescribeDBInstances” アクションのレスポンス要素として使用されます。

- AutoMinorVersionUpgrade — タイプ `boolean` のブール値 (真または偽)。

  マイナーバージョンのパッチを自動的に適用することを指定します。
- AvailabilityZone — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスを配置するアベイラビリティーゾーンの名前を指定します。
- BackupRetentionPeriod - タイプ `integer` の整数 (符号付き 32 ビット整数)。

  自動 DB スナップショットが保持される日数を指定します。
- CACertificateIdentifier — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB インスタンスの CA 証明書の識別子。
- CopyTagsToSnapshot — タイプ `boolean` のブール値 (真または偽)。

タグを DB インスタンスから DB インスタンスのスナップショットにコピーするかどうかを指定します。

- DBClusterIdentifier — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスが DB クラスターのメンバーである場合は、DB インスタンスがメンバーとなっている DB クラスターの名前が含まれます。

- DBInstanceArn — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスの Amazon リソースネーム (ARN)。

- DBInstanceClass — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスのコンピューティング名とメモリ容量クラスの名前が含まれています。

- DBInstanceIdentifier — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  ユーザーが指定したデータベース識別子が含まれています。この識別子は、DB インスタンスを識別する一意のキーです。

- DbInstancePort - タイプ `integer` の整数 (符号付き 32 ビット整数)。

  DB インスタンスがリッスンするポートを指定します。DB インスタンスが DB クラスターの一部である場合は、DB クラスターのポートとは別のポートを指定できます。

- DBInstanceStatus — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  このデータベースの現在の状態を指定します。

- DbiResourceId — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスの Amazon リージョン固有のイミュータブルな識別子。この識別子は、DB インスタンスの Amazon KMS キーにアクセスするたびに Amazon CloudTrail ログエントリに記録されます。

- DBName — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  データベース名。

- DBParameterGroups – DBParameterGroupStatus オブジェクトの配列。

  この DB インスタンスに適用される DB パラメータグループのリストを入力します。

- DBSecurityGroups – DBSecurityGroupMembership オブジェクトの配列。

DBSecurityGroup.Name と DBSecurityGroup.Status のサブ要素のみを含む DB セキュリティグループ要素の一覧を入力します。

- DBSubnetGroup – DBSubnetGroup オブジェクト。

  サブネットグループ内の名前、説明、サブネットなど、DB インスタンスに関連付けられているサブネットグループに関する情報を指定します。

- DeletionProtection — タイプ boolean の BooleanOptional (ブール値 (真または偽))。

  DB インスタンスで削除保護が有効になっているかどうかを示します。削除保護が有効な場合、インスタンスは削除できません。「DB インスタンスを削除する」を参照してください。

- DomainMemberships – DomainMembership オブジェクトの配列。

  サポートされません

- EnabledCloudwatchLogsExports — タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  この DB インスタンスが CloudWatch Logs にエクスポートするように設定されているログタイプのリスト。

- Endpoint – エンドポイント オブジェクト。

  接続エンドポイントを指定します。

- Engine — タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  この DB インスタンスに使用されるデータベースエンジンの名前を提供します。

- EngineVersion — タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  データベースエンジンのバージョンを示します。

- EnhancedMonitoringResourceArn — タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスの拡張モニタリングメトリクスデータを受け取る Amazon CloudWatch Logs ログストリームの Amazon リソースネーム (ARN)。

- IAMDatabaseAuthenticationEnabled — タイプ boolean のブール値 (真または偽)。

  Amazon Identity and Access Management (IAM) 認証が有効な場合は true、それ以外の場合は false です。

- InstanceCreateTime — TStamp、タイプ `timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点) 。

  DB インスタンスが作成された日時を入力します。
- Iops - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  プロビジョンド IOPS(I/O operations per second) を指定します。
- KmsKeyId — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  サポートなし: DB インスタンスの暗号化は、DB クラスターによって管理されます。
- LatestRestorableTime — TStamp、タイプ `timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点) 。

  特定時点への復元によりデータベースを復元できる最新の日時を指定します。
- LicenseModel — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB インスタンスのライセンスモデル情報。
- MonitoringInterval - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  DB インスタンスについて拡張モニターメトリクスが収集される時点間の間隔 (秒単位)。
- MonitoringRoleArn — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  Neptune が拡張モニタリングメトリクスを Amazon CloudWatch Logs に送信することを許可する IAM ロールの ARN。
- MultiAZ — タイプ `boolean` のブール値 (真または偽)。

  DB インスタンスが Multi-AZ デプロイかどうかを指定します。
- PendingModifiedValues – PendingModifiedValues オブジェクト。

  DB インスタンスへの変更が保留中であることを指定します。この要素は、変更が保留中の場合にのみ含まれます。特定の変更は、サブエレメントによって識別されます。
- PreferredBackupWindow — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  `BackupRetentionPeriod` に応じた、自動バックアップが有効な場合に自動バックアップが作成される毎日の時間範囲を指定します。
- PreferredMaintenanceWindow — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  週 1 回のシステムメンテナンスを実行できる時間範囲を世界標準時 (UTC) で指定します。

- PromotionTier - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  既存のプライマリインスタンスの障害後に、リードレプリカをプライマリインスタンスに昇格される順序を指定する値。
- PubliclyAccessible — タイプ `boolean` のブール値 (真または偽)。

  このフラグは、今後は使用しないでください。
- ReadReplicaDBClusterIdentifiers — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB インスタンスに関連付けられているリードレプリカの 1 つ以上の DB クラスターの識別子を含みます。
- ReadReplicaDBInstanceIdentifiers — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB インスタンスに関連付けられているリードレプリカの 1 つ以上の識別子を含みます。
- ReadReplicaSourceDBInstanceIdentifier — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB インスタンスがリードレプリカの場合は、ソース DB インスタンスの識別子が含まれます。
- SecondaryAvailabilityZone — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  存在する場合は、マルチ AZ をサポートする DB インスタンスのセカンダリアベイラビリティーゾーンの名前を指定します。
- StatusInfos – DBInstanceStatusInfo オブジェクトの配列。

  リードレプリカのステータス。インスタンスがリードレプリカではない場合、これは空白です。
- StorageEncrypted — タイプ `boolean` のブール値 (真または偽)。

  サポートなし: DB インスタンスの暗号化は、DB クラスターによって管理されます。
- StorageType — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスに関連付けられたストレージタイプを指定します。
- TdeCredentialArn — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  インスタンスが TDE 暗号化のために関連付けられているキーストアからの ARN。
- Timezone — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

サポート外。

- VpcSecurityGroups – VpcSecurityGroupMembership オブジェクトの配列。

  DB インスタンスが属する VPC セキュリティグループ要素のリストを入力します。

エラー

- DBInstanceNotFoundFault
- InvalidDBInstanceStateFault
- DBSnapshotAlreadyExistsFault
- SnapshotQuotaExceededFault
- InvalidDBClusterStateFault

ModifyDBInstance (アクション)

この API の AWS CLI 名は `modify-db-instance` です。

DB インスタンスの設定を変更します。これらのパラメータとリクエストに新しい値を指定することで、1 つ以上のデータベース設定パラメータを変更できます。DB インスタンスにどのような変更を加えることができるかについて知るには、the section called “ModifyDBInstance” を呼び出す前に the section called “DescribeValidDBInstanceModifications” を呼び出します。

リクエスト

- AllowMajorVersionUpgrade (CLI では: `--allow-major-version-upgrade`) — タイプ `boolean` のブール値 (ブール値 (真または偽))。

  メジャーバージョンアップグレードが許可されることを示します。このパラメータの変更が停止につながらなければ、変更は非同期的に可能な限り迅速に適用されます。
- ApplyImmediately (CLI では: `--apply-immediately`) — タイプ `boolean` のブール値 (ブール値 (真または偽))。

  DB インスタンスの `PreferredMaintenanceWindow` の設定に関係なく、このリクエストの変更と保留中の変更をできるだけ早く非同期に適用するかどうかを指定します。

  このパラメータを `false` に設定した場合、DB インスタンスへの変更は次のメンテナンスウィンドウ中に適用されます。パラメータの変更によっては機能停止を引き起こす可能性があり、次回の

the section called “RebootDBInstance” への呼び出し、または次回の再起動の失敗時に適用されます。

デフォルト: `false`

- AutoMinorVersionUpgrade (CLI では: `--auto-minor-version-upgrade`) — タイプ `boolean` の BooleanOptional (ブール値 (真または偽))。

  メンテナンスの時間帯に、DB インスタンスに自動的にマイナーバージョンアップグレードが適用されることを示します。次の場合を除き、このパラメータを変更しても機能停止にはならず、変更はできるだけ早く非同期的に適用されます。機能停止は、メンテナンスウィンドウ中にこのパラメータが `true` に設定されており、新しいマイナーバージョンが使用可能で、Neptune でそのエンジンバージョンの自動パッチ適用が有効になっている場合に発生します。

- BackupRetentionPeriod (CLI では: `--backup-retention-period`) — タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  該当しません。自動バックアップの保持期間は、DB クラスターによって管理されます。詳細については、「the section called “ModifyDBCluster”」を参照してください。

  デフォルト: 既存の設定を仕様

- CACertificateIdentifier (CLI では: `--ca-certificate-identifier`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  インスタンスに関連付ける必要のある証明書を示します。

- CloudwatchLogsExportConfiguration (CLI では: `--cloudwatch-logs-export-configuration`) — CloudwatchLogsExportConfiguration オブジェクト。

  特定の DB インスタンスまたは DB クラスターの CloudWatch Logs にエクスポートするために有効にするログタイプの設定。

- CopyTagsToSnapshot (CLI では: `--copy-tags-to-snapshot`) — タイプ `boolean` の BooleanOptional (ブール値 (真または偽))。

  DB インスタンスから DB インスタンスのスナップショットにすべてのタグをコピーする場合は true、それ以外の場合は false です。デフォルトは [False] (偽) です。

- DBInstanceClass (CLI では: `--db-instance-class`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  DB インスタンスの新しいコンピューティング性能とメモリ容量 (例: `db.m4.large`)。すべての DB インスタンスクラスがすべての Amazon リージョンで使用できるわけではありません。

DB インスタンスクラスを変更すると、変更中に機能停止が発生します。ApplyImmediately がこのリクエストの true として指定されない限り、変更は次のメンテナンスウィンドウ中に適用されます。

デフォルト: 既存の設定を仕様

- DBInstanceIdentifier (CLI では: --db-instance-identifier) — 必須: タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンス識別子。この値は小文字で保存されます。

  制約:

  - 既存の DBInstance の識別子と一致する必要があります。
- DBParameterGroupName (CLI では: --db-parameter-group-name) — タイプ: string (UTF-8 でエンコードされた文字列) の文字列。

  この DB インスタンスに適用される DB パラメータグループの名前。この設定を変更しても機能は停止しません。パラメータグループ名自体は即時に変更されますが、フェイルオーバーなしでインスタンスを再起動するまで実際のパラメータの変更は適用されません。db インスタンスは自動的には再起動されず、次のメンテナンス時までパラメータの変更は適用されません。

  デフォルト: 既存の設定を仕様

  制約: DB パラメータグループは、この DB インスタンスと同じ DB パラメータグループファミリに属している必要があります。
- DBPortNumber (CLI では: --db-port-number) — タイプ integer の IntegerOptional (符号付き 32 ビット整数)。

  データベースが接続を受け入れるポート番号。

  DBPortNumber の値は、DB インスタンスのオプショングループのオプションに指定されているポート値と一致しないようにしてください。

  DBPortNumber 値を変更すると、ApplyImmediately パラメータの値に関係なくデータベースが再起動されます。

  デフォルト: 8182
- DBSecurityGroups (CLI では: --db-security-groups) — タイプ: string (UTF-8 でエンコードされた文字列) の文字列。

この DB インスタンスで承認する DB セキュリティグループの一覧。この設定を変更しても機能停止が発生しない場合、変更は可能な限り早く非同期的に適用されます。

制約:

- 指定した場合、既存の DBSecurityGroups と一致する必要があります。

- DBSubnetGroupName (CLI では: `--db-subnet-group-name`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  DB インスタンスの新しい DB サブネットグループ。このパラメータを使用して、DB インスタンスを別の VPC に移動できます。

  サブネットグループを変更すると、変更時に機能停止が発生します。`ApplyImmediately` パラメータに `true` を指定しない限り、変更は次のメンテナンス期間中に適用されます。

  制約: 指定した場合、既存の DBSubnetGroup の名前と一致する必要があります。

  例: `mySubnetGroup`

- DeletionProtection (CLI では: `--deletion-protection`) — タイプ `boolean` の BooleanOptional (ブール値 (真または偽))。

  DB インスタンスで削除保護が有効になっているかどうかを示す値。削除保護が有効になっている場合、データベースを削除することはできません。デフォルトでは、削除保護は無効です。「DB インスタンスを削除する」を参照してください。

- Domain (CLI では: `--domain`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  サポート外。

- DomainIAMRoleName (CLI では: `--domain-iam-role-name`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  サポートされません

- EnableIAMDatabaseAuthentication (CLI では: `--enable-iam-database-authentication`) — タイプ `boolean` の BooleanOptional (ブール値 (真または偽))。

  Amazon Identity and Access Management (IAM) アカウントのデータベースアカウントへのマッピングが有効な場合は true、それ以外の場合は false です。

  次のデータベースエンジンには、IAM データベース認証を有効にできます。

該当しません。Amazon IAM アカウントのデータベースアカウンへのマッピングは、DB クラスターによって管理されます。詳細については、「the section called “ModifyDBCluster”」を参照してください。

デフォルト: `false`

- EngineVersion (CLI では: `--engine-version`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  アップグレード先のデータベースエンジンのバージョン番号です。現在、このパラメータを設定しても効果はありません。データベースエンジンを最新のリリースにアップグレードするには、the section called “ApplyPendingMaintenanceAction” API を使用します。

- Iops (CLI では: `--iops`) — タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  インスタンスの新しいプロビジョンド IOPS (I/O operations per second) の値です。

  この設定を変更しても機能停止は発生しません。また、このリクエストで `ApplyImmediately` パラメータが `true` に設定されていない限り、変更は次のメンテナンスウィンドウ中に適用されます。

  デフォルト: 既存の設定を仕様

- MonitoringInterval (CLI では: `--monitoring-interval`) — タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  DB インスタンスについて拡張モニターメトリクスが収集される時点間の間隔 (秒単位)。拡張モニタリングメトリクスの収集を無効にするには、0 を指定します。デフォルトは 0 です。

  `MonitoringRoleArn` を指定した場合は、`MonitoringInterval` を 0 以外の値に設定する必要があります。

  有効な値: `0, 1, 5, 10, 15, 30, 60`

- MonitoringRoleArn (CLI では: `--monitoring-role-arn`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  Neptune が拡張モニタリングメトリクスを Amazon CloudWatch Logs に送信することを許可する IAM ロールの ARN。例えば、`arn:aws:iam:123456789012:role/emaccess` です。

  `MonitoringInterval` を 0 以外に設定した場合は、`MonitoringRoleArn` 値を設定する必要があります。

- MultiAZ (CLI では: `--multi-az`) — タイプ `boolean` の BooleanOptional (ブール値 (真または偽))。

  DB インスタンスが Multi-AZ デプロイかどうかを指定します。このパラメータを変更しても機能停止は発生しません。また、このリクエストで `ApplyImmediately` パラメータが `true` に設定されていない限り、変更は次のメンテナンスウィンドウ中に適用されます。

- NewDBInstanceIdentifier (CLI では: `--new-db-instance-identifier`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  DB インスタンスの名前を変更する場合の DB インスタンスの新規 DB インスタンス識別子 DB インスタンス識別子を変更すると、`Apply Immediately` を true に設定した場合はインスタンスが即時に再起動し、`Apply Immediately` を false にした場合は次のメンテナンス時に再起動します。この値は小文字で保存されます。

  制約:

  - 1 ~ 63 の文字、数字またはハイフンを使用する必要があります。
  - 1 字目は文字である必要があります。
  - ハイフンを、文字列の最後に使用したり、2 つ続けて使用したりすることはできません。

  例: `mydbinstance`

- PreferredBackupWindow (CLI では: `--preferred-backup-window`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  自動バックアップが有効になっている場合に自動バックアップが作成される毎日の時間帯。

  該当しません。自動バックアップを作成するための毎日の時間範囲は、DB クラスターによって管理されます。詳細については、「the section called “ModifyDBCluster”」を参照してください。

  制約:

  - hh24:mi-hh24:mi の形式である必要があります。
  - 時間は協定世界時 (UTC) である必要があります。
  - 必要なメンテナンス期間と競合してはいけません。
  - 少なくとも 30 分以上必要です。

- PreferredMaintenanceWindow (CLI では: `--preferred-maintenance-window`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

システムメンテナンスが発生する可能性がある週単位の時間範囲 (UTC) で、停止する可能性があります。次の場合を除き、このパラメータを変更しても機能停止にはならず、変更はできるだけ早く非同期的に適用されます。再起動を引き起こす保留中のアクションがあり、メンテナンスウィンドウが現在の時刻を含むように変更されている場合、このパラメータを変更すると DB インスタンスが再起動されます。このウィンドウを現在の時刻に変更した場合、保留中の変更が確実に適用されるように、現在の時刻からウィンドウの終わりまで 30 分以上必要です。

デフォルト: 既存の設定を仕様

形式: ddd:hh 24:mi-ddd:hh 24:mi

有効な日数: Mon | Tue | Wed | Thu | Fri | Sat | Sun

制約: 30 分以上にする必要があります

- PromotionTier (CLI では: `--promotion-tier`) — タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  既存のプライマリインスタンスの障害後に、リードレプリカをプライマリインスタンスに昇格される順序を指定する値。

  デフォルト: 1

  有効な値: 0 ~ 15

- PubliclyAccessible (CLI では: `--publicly-accessible`) — タイプ `boolean` の BooleanOptional (ブール値 (真または偽))。

  このフラグは、今後は使用しないでください。

- StorageType (CLI では: `--storage-type`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  該当しません。Neptune の場合、ストレージタイプは DB クラスターレベルで管理されます。

- TdeCredentialArn (CLI では: `--tde-credential-arn`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  TDE 暗号化のためにインスタンスに関連付けるキーストアからの ARN。

- TdeCredentialPassword (CLI の場合は `--tde-credential-password`) — `string` 型の SensitiveString (UTF-8 でエンコードされた文字列)。

デバイスにアクセスするためのキーストアからの指定された ARN のパスワード。

- VpcSecurityGroupIds (CLI では: `--vpc-security-group-ids`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB インスタンスで承認する EC2 VPC セキュリティグループの一覧。この変更は、可能な限り早く非同期的に適用されます。

  該当しません。EC2 VPC セキュリティグループの関連リストは、DB クラスターによって管理されます。詳細については、「the section called “ModifyDBCluster”」を参照してください。

  制約:

  - 指定した場合、既存の VpcSecurityGroupIds と一致する必要があります。

レスポンス

Amazon Neptune DB インスタンスの詳細が含まれています。

このデータ型は、the section called “DescribeDBInstances” アクションのレスポンス要素として使用されます。

- AutoMinorVersionUpgrade — タイプ `boolean` のブール値 (真または偽)。

  マイナーバージョンのパッチを自動的に適用することを指定します。
- AvailabilityZone — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスを配置するアベイラビリティーゾーンの名前を指定します。
- BackupRetentionPeriod - タイプ `integer` の整数 (符号付き 32 ビット整数)。

  自動 DB スナップショットが保持される日数を指定します。
- CACertificateIdentifier — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB インスタンスの CA 証明書の識別子。
- CopyTagsToSnapshot — タイプ `boolean` のブール値 (真または偽)。

  タグを DB インスタンスから DB インスタンスのスナップショットにコピーするかどうかを指定します。
- DBClusterIdentifier — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

DB インスタンスが DB クラスターのメンバーである場合は、DB インスタンスがメンバーとなっている DB クラスターの名前が含まれます。

- DBInstanceArn — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスの Amazon リソースネーム (ARN)。

- DBInstanceClass — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスのコンピューティング名とメモリ容量クラスの名前が含まれています。

- DBInstanceIdentifier — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  ユーザーが指定したデータベース識別子が含まれています。この識別子は、DB インスタンスを識別する一意のキーです。

- DbInstancePort - タイプ `integer` の整数 (符号付き 32 ビット整数)。

  DB インスタンスがリッスンするポートを指定します。DB インスタンスが DB クラスターの一部である場合は、DB クラスターのポートとは別のポートを指定できます。

- DBInstanceStatus — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  このデータベースの現在の状態を指定します。

- DbiResourceId — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスの Amazon リージョン固有のイミュータブルな識別子。この識別子は、DB インスタンスの Amazon KMS キーにアクセスするたびに Amazon CloudTrail ログエントリに記録されます。

- DBName — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  データベース名。

- DBParameterGroups – DBParameterGroupStatus オブジェクトの配列。

  この DB インスタンスに適用される DB パラメータグループのリストを入力します。

- DBSecurityGroups – DBSecurityGroupMembership オブジェクトの配列。

  `DBSecurityGroup.Name` と `DBSecurityGroup.Status` のサブ要素のみを含む DB セキュリティグループ要素の一覧を入力します。

- DBSubnetGroup – DBSubnetGroup オブジェクト。

サブネットグループ内の名前、説明、サブネットなど、DB インスタンスに関連付けられているサブネットグループに関する情報を指定します。

- DeletionProtection — タイプ `boolean` の BooleanOptional (ブール値 (真または偽))。

  DB インスタンスで削除保護が有効になっているかどうかを示します。削除保護が有効な場合、インスタンスは削除できません。「DB インスタンスを削除する」を参照してください。
- DomainMemberships – DomainMembership オブジェクトの配列。

  サポートされません
- EnabledCloudwatchLogsExports — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB インスタンスが CloudWatch Logs にエクスポートするように設定されているログタイプのリスト。
- Endpoint – エンドポイント オブジェクト。

  接続エンドポイントを指定します。
- Engine — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB インスタンスに使用されるデータベースエンジンの名前を提供します。
- EngineVersion — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  データベースエンジンのバージョンを示します。
- EnhancedMonitoringResourceArn — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスの拡張モニタリングメトリクスデータを受け取る Amazon CloudWatch Logs ログストリームの Amazon リソースネーム (ARN)。
- IAMDatabaseAuthenticationEnabled — タイプ `boolean` のブール値 (真または偽)。

  Amazon Identity and Access Management (IAM) 認証が有効な場合は true、それ以外の場合は false です。
- InstanceCreateTime — TStamp、タイプ `timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点) 。

  DB インスタンスが作成された日時を入力します。
- Iops - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

プロビジョンド IOPS(I/O operations per second) を指定します。

- KmsKeyId — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  サポートなし: DB インスタンスの暗号化は、DB クラスターによって管理されます。

- LatestRestorableTime — TStamp、タイプ `timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点) 。

  特定時点への復元によりデータベースを復元できる最新の日時を指定します。

- LicenseModel — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB インスタンスのライセンスモデル情報。

- MonitoringInterval - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  DB インスタンスについて拡張モニターメトリクスが収集される時点間の間隔 (秒単位)。

- MonitoringRoleArn — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  Neptune が拡張モニタリングメトリクスを Amazon CloudWatch Logs に送信することを許可する IAM ロールの ARN。

- MultiAZ — タイプ `boolean` のブール値 (真または偽)。

  DB インスタンスが Multi-AZ デプロイかどうかを指定します。

- PendingModifiedValues – PendingModifiedValues オブジェクト。

  DB インスタンスへの変更が保留中であることを指定します。この要素は、変更が保留中の場合にのみ含まれます。特定の変更は、サブエレメントによって識別されます。

- PreferredBackupWindow — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  `BackupRetentionPeriod` に応じた、自動バックアップが有効な場合に自動バックアップが作成される毎日の時間範囲を指定します。

- PreferredMaintenanceWindow — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  週 1 回のシステムメンテナンスを実行できる時間範囲を世界標準時 (UTC) で指定します。

- PromotionTier - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  既存のプライマリインスタンスの障害後に、リードレプリカをプライマリインスタンスに昇格される順序を指定する値。

- PubliclyAccessible — タイプ `boolean` のブール値 (真または偽)。

このフラグは、今後は使用しないでください。

- ReadReplicaDBClusterIdentifiers — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB インスタンスに関連付けられているリードレプリカの 1 つ以上の DB クラスターの識別子を含みます。

- ReadReplicaDBInstanceIdentifiers — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB インスタンスに関連付けられているリードレプリカの 1 つ以上の識別子を含みます。

- ReadReplicaSourceDBInstanceIdentifier — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB インスタンスがリードレプリカの場合は、ソース DB インスタンスの識別子が含まれます。

- SecondaryAvailabilityZone — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  存在する場合は、マルチ AZ をサポートする DB インスタンスのセカンダリアベイラビリティーゾーンの名前を指定します。

- StatusInfos – DBInstanceStatusInfo オブジェクトの配列。

  リードレプリカのステータス。インスタンスがリードレプリカではない場合、これは空白です。

- StorageEncrypted — タイプ `boolean` のブール値 (真または偽)。

  サポートなし: DB インスタンスの暗号化は、DB クラスターによって管理されます。

- StorageType — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスに関連付けられたストレージタイプを指定します。

- TdeCredentialArn — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  インスタンスが TDE 暗号化のために関連付けられているキーストアからの ARN。

- Timezone — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  サポート外。

- VpcSecurityGroups – VpcSecurityGroupMembership オブジェクトの配列。

  DB インスタンスが属する VPC セキュリティグループ要素のリストを入力します。

エラー

- InvalidDBInstanceStateFault
- InvalidDBSecurityGroupStateFault
- DBInstanceAlreadyExistsFault
- DBInstanceNotFoundFault
- DBSecurityGroupNotFoundFault
- DBParameterGroupNotFoundFault
- InsufficientDBInstanceCapacityFault
- StorageQuotaExceededFault
- InvalidVPCNetworkStateFault
- ProvisionedIopsNotAvailableInAZFault
- OptionGroupNotFoundFault
- DBUpgradeDependencyFailureFault
- StorageTypeNotSupportedFault
- AuthorizationNotFoundFault
- CertificateNotFoundFault
- DomainNotFoundFault

RebootDBInstance (アクション)

この API の AWS CLI 名は `reboot-db-instance` です。

通常はメンテナンスのために、DB インスタンスを再起動する必要がある場合があります。たとえば、特定の変更を行う場合や、DB インスタンスに関連付けられた DB パラメータグループを変更する場合は、変更を有効にするためにインスタンスを再起動する必要があります。

DB インスタンスを再起動すると、データベースエンジンサービスが再起動されます。DB インスタンスを再起動すると一時的に機能停止になります。その間、DB インスタンスのステータスは [rebooting] に設定されます。

リクエスト

- DBInstanceIdentifier (CLI では: `--db-instance-identifier`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

DB インスタンス識別子。このパラメータは小文字で保存されます。

制約:

- 既存の DBInstance の識別子と一致する必要があります。

- ForceFailover (CLI では: `--force-failover`) — タイプ `boolean` の BooleanOptional (ブール値 (真または偽))。

  `true` の場合、再起動はマルチ AZ フェイルオーバーによって行われます。

  制約: インスタンスがマルチ AZ 用に設定されていない場合、`true` を指定することはできません。

レスポンス

Amazon Neptune DB インスタンスの詳細が含まれています。

このデータ型は、the section called “DescribeDBInstances” アクションのレスポンス要素として使用されます。

- AutoMinorVersionUpgrade — タイプ `boolean` のブール値 (真または偽)。

  マイナーバージョンのパッチを自動的に適用することを指定します。

- AvailabilityZone — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスを配置するアベイラビリティーゾーンの名前を指定します。

- BackupRetentionPeriod - タイプ `integer` の整数 (符号付き 32 ビット整数)。

  自動 DB スナップショットが保持される日数を指定します。

- CACertificateIdentifier — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB インスタンスの CA 証明書の識別子。

- CopyTagsToSnapshot — タイプ `boolean` のブール値 (真または偽)。

  タグを DB インスタンスから DB インスタンスのスナップショットにコピーするかどうかを指定します。

- DBClusterIdentifier — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

DB インスタンスが DB クラスターのメンバーである場合は、DB インスタンスがメンバーとなっている DB クラスターの名前が含まれます。

- DBInstanceArn — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスの Amazon リソースネーム (ARN)。

- DBInstanceClass — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスのコンピューティング名とメモリ容量クラスの名前が含まれています。

- DBInstanceIdentifier — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  ユーザーが指定したデータベース識別子が含まれています。この識別子は、DB インスタンスを識別する一意のキーです。

- DbInstancePort - タイプ `integer` の整数 (符号付き 32 ビット整数)。

  DB インスタンスがリッスンするポートを指定します。DB インスタンスが DB クラスターの一部である場合は、DB クラスターのポートとは別のポートを指定できます。

- DBInstanceStatus — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  このデータベースの現在の状態を指定します。

- DbiResourceId — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスの Amazon リージョン固有のイミュータブルな識別子。この識別子は、DB インスタンスの Amazon KMS キーにアクセスするたびに Amazon CloudTrail ログエントリに記録されます。

- DBName — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  データベース名。

- DBParameterGroups – DBParameterGroupStatus オブジェクトの配列。

  この DB インスタンスに適用される DB パラメータグループのリストを入力します。

- DBSecurityGroups – DBSecurityGroupMembership オブジェクトの配列。

  `DBSecurityGroup.Name` と `DBSecurityGroup.Status` のサブ要素のみを含む DB セキュリティグループ要素の一覧を入力します。

- DBSubnetGroup – DBSubnetGroup オブジェクト。

サブネットグループ内の名前、説明、サブネットなど、DB インスタンスに関連付けられているサブネットグループに関する情報を指定します。

- DeletionProtection — タイプ `boolean` の BooleanOptional (ブール値 (真または偽))。

  DB インスタンスで削除保護が有効になっているかどうかを示します。削除保護が有効な場合、インスタンスは削除できません。「DB インスタンスを削除する」を参照してください。
- DomainMemberships – DomainMembership オブジェクトの配列。

  サポートされません
- EnabledCloudwatchLogsExports — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB インスタンスが CloudWatch Logs にエクスポートするように設定されているログタイプのリスト。
- Endpoint – エンドポイント オブジェクト。

  接続エンドポイントを指定します。
- Engine — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB インスタンスに使用されるデータベースエンジンの名前を提供します。
- EngineVersion — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  データベースエンジンのバージョンを示します。
- EnhancedMonitoringResourceArn — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスの拡張モニタリングメトリクスデータを受け取る Amazon CloudWatch Logs ログストリームの Amazon リソースネーム (ARN)。
- IAMDatabaseAuthenticationEnabled — タイプ `boolean` のブール値 (真または偽)。

  Amazon Identity and Access Management (IAM) 認証が有効な場合は true、それ以外の場合は false です。
- InstanceCreateTime — TStamp、タイプ `timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点) 。

  DB インスタンスが作成された日時を入力します。
- Iops - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

プロビジョンド IOPS(I/O operations per second) を指定します。

- KmsKeyId — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  サポートなし: DB インスタンスの暗号化は、DB クラスターによって管理されます。

- LatestRestorableTime — TStamp、タイプ `timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点) 。

  特定時点への復元によりデータベースを復元できる最新の日時を指定します。

- LicenseModel — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB インスタンスのライセンスモデル情報。

- MonitoringInterval - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  DB インスタンスについて拡張モニターメトリクスが収集される時点間の間隔 (秒単位)。

- MonitoringRoleArn — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  Neptune が拡張モニタリングメトリクスを Amazon CloudWatch Logs に送信することを許可する IAM ロールの ARN。

- MultiAZ — タイプ `boolean` のブール値 (真または偽)。

  DB インスタンスが Multi-AZ デプロイかどうかを指定します。

- PendingModifiedValues – PendingModifiedValues オブジェクト。

  DB インスタンスへの変更が保留中であることを指定します。この要素は、変更が保留中の場合にのみ含まれます。特定の変更は、サブエレメントによって識別されます。

- PreferredBackupWindow — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  `BackupRetentionPeriod` に応じた、自動バックアップが有効な場合に自動バックアップが作成される毎日の時間範囲を指定します。

- PreferredMaintenanceWindow — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  週 1 回のシステムメンテナンスを実行できる時間範囲を世界標準時 (UTC) で指定します。

- PromotionTier - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  既存のプライマリインスタンスの障害後に、リードレプリカをプライマリインスタンスに昇格される順序を指定する値。

- PubliclyAccessible — タイプ `boolean` のブール値 (真または偽)。

このフラグは、今後は使用しないでください。

- ReadReplicaDBClusterIdentifiers — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB インスタンスに関連付けられているリードレプリカの 1 つ以上の DB クラスターの識別子を含みます。

- ReadReplicaDBInstanceIdentifiers — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB インスタンスに関連付けられているリードレプリカの 1 つ以上の識別子を含みます。

- ReadReplicaSourceDBInstanceIdentifier — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB インスタンスがリードレプリカの場合は、ソース DB インスタンスの識別子が含まれます。

- SecondaryAvailabilityZone — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  存在する場合は、マルチ AZ をサポートする DB インスタンスのセカンダリアベイラビリティーゾーンの名前を指定します。

- StatusInfos – DBInstanceStatusInfo オブジェクトの配列。

  リードレプリカのステータス。インスタンスがリードレプリカではない場合、これは空白です。

- StorageEncrypted — タイプ `boolean` のブール値 (真または偽)。

  サポートなし: DB インスタンスの暗号化は、DB クラスターによって管理されます。

- StorageType — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスに関連付けられたストレージタイプを指定します。

- TdeCredentialArn — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  インスタンスが TDE 暗号化のために関連付けられているキーストアからの ARN。

- Timezone — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  サポート外。

- VpcSecurityGroups – VpcSecurityGroupMembership オブジェクトの配列。

  DB インスタンスが属する VPC セキュリティグループ要素のリストを入力します。

エラー

- InvalidDBInstanceStateFault
- DBInstanceNotFoundFault

DescribeDBInstances (アクション)

この API の AWS CLI 名は `describe-db-instances` です。

プロビジョニングされたインスタンスに関する情報を返し、ページ分割をサポートします。

> **Note**
>
> このオペレーションでは、Amazon RDS インスタンスと Amazon DocDB インスタンスの情報も返すことができます。

リクエスト

- DBInstanceIdentifier (CLI では: `--db-instance-identifier`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  ユーザーが指定したインスタンス識別子。このパラメータを指定した場合は、特定の DB インスタンスからの情報だけが返されます。このパラメータでは大文字と小文字は区別されません。

  制約:

  - 指定した場合、既存の DBInstance の識別子と一致している必要があります。
- Filters (CLI では: `--filters`) - フィルター オブジェクトの配列。

  記述する 1 つ以上の DB インスタンスを指定するフィルター。

  サポートされているフィルター:

  - `db-cluster-id` - DB クラスター識別子と DB クラスターの Amazon リソースネーム (ARN) を受け入れます。結果リストには、これらの ARN によって識別された DB クラスターに関連付けられた DB インスタンスに関する情報のみが含まれます。
  - `engine` - エンジン名 (`neptune` など) を受け入れ、そのエンジンによって作成された DB インスタンスに結果リストを制限します。

たとえば、Amazon CLI からこの API を呼び出し、Neptune DB インスタンスのみが返されるようにフィルタリングするには、次のコマンドを使用します。

Example

```
aws neptune describe-db-instances \
            --filters  Name=engine,Values=neptune
```

- Marker (CLI では: `--marker`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  以前の `DescribeDBInstances` リクエストによって提供されたオプションのページ割りトークン。このパラメータを指定した場合、レスポンスには `MaxRecords` で指定された値まで、マーカーを超えるレコードのみが含まれます。

- MaxRecords (CLI では: `--max-records`) — タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  レスポンスに含めるレコードの最大数。指定された `MaxRecords` の値よりも多くのレコードが存在する場合、マーカーと呼ばれるページ割りトークンがレスポンスに含まれるため、残りの結果を取得できます。

  デフォルト: 100

  制約: 最小 20、最大 100。

レスポンス

- DBInstances – DBInstance オブジェクトの配列。

  the section called “DBInstance” インスタンスのリスト。

- Marker — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  以前のリクエストによって提供されたオプションのページ割りトークン。このパラメータを指定した場合、レスポンスには `MaxRecords` で指定された値まで、マーカーを超えるレコードのみが含まれます。

エラー

- DBInstanceNotFoundFault

DescribeOrderableDBInstanceOptions (アクション)

この API の AWS CLI 名は `describe-orderable-db-instance-options` です。

指定されたエンジンの注文可能な DB インスタンスオプションのリストを返します。

リクエスト

- DBInstanceClass (CLI では: `--db-instance-class`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスクラスのフィルター値。指定した DB インスタンスのクラスと一致する使用可能なオファリングのみを表示するには、このパラメータを指定してください。
- Engine (CLI では: `--engine`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスオプションを取得するエンジンの名前。
- EngineVersion (CLI では: `--engine-version`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  エンジンバージョンフィルターの値。指定したエンジンのバージョンと一致する使用可能なオファリングのみを表示するには、このパラメータを指定してください。
- Filters (CLI では: `--filters`) - フィルター オブジェクトの配列。

  このパラメータは、現在サポートされていません。
- LicenseModel (CLI では: `--license-model`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  ライセンスモデルフィルター値。指定したライセンスモデルと一致する使用可能なオファリングのみを表示するには、このパラメータを指定してください。
- Marker (CLI では: `--marker`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  以前の DescribeOrderableDBInstanceOptions リクエストによって提供されたオプションのページ分割トークン。このパラメータを指定した場合、レスポンスには `MaxRecords` で指定された値まで、マーカーを超えるレコードのみが含まれます。
- MaxRecords (CLI では: `--max-records`) — タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

レスポンスに含めるレコードの最大数。指定された MaxRecords の値よりも多くのレコードが存在する場合、マーカーと呼ばれるページ割りトークンがレスポンスに含まれるため、残りの結果を取得できます。

デフォルト: 100

制約: 最小 20、最大 100。

- Vpc (CLI では: --vpc) — タイプ boolean の BooleanOptional (ブール値 (真または偽))。

  VPC フィルター値。このパラメータを指定すると、 VPC 環境または非 VPC 環境のみが表示されます。

レスポンス

- Marker — タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  以前の OrderableDBInstanceOptions リクエストによって提供されたオプションのページ分割トークン。このパラメータを指定した場合、レスポンスには MaxRecords で指定された値まで、マーカーを超えるレコードのみが含まれます。

- OrderableDBInstanceOptions – OrderableDBInstanceOption オブジェクトの配列。

  DB インスタンスの順序付可能なオプションに関する情報が含まれている the section called "OrderableDBInstanceOption" 構造。

DescribeValidDBInstanceModifications (アクション)

この API の AWS CLI 名は describe-valid-db-instance-modifications です。

DB インスタンスにどのような変更を加えることができるかについて知るには、the section called "DescribeValidDBInstanceModifications" を呼び出します。the section called "ModifyDBInstance" を呼び出すときにこの情報を使用できます。

リクエスト

- DBInstanceIdentifier (CLI では: --db-instance-identifier) — 必須: タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスのお客様の識別子または ARN。

レスポンス

DB インスタンスに加えることができる有効な変更に関する情報。the section called “DescribeValidDBInstanceModifications” アクションへの呼び出しが成功した結果が含まれています。the section called “ModifyDBInstance” を呼び出すときにこの情報を使用できます。

- Storage – ValidStorageOptions オブジェクトの配列。

  DB インスタンスの有効なストレージオプション。

エラー

- DBInstanceNotFoundFault
- InvalidDBInstanceStateFault

構造:

DBInstance (構造)

Amazon Neptune DB インスタンスの詳細が含まれています。

このデータ型は、the section called “DescribeDBInstances” アクションのレスポンス要素として使用されます。

フィールド

- AutoMinorVersionUpgrade — タイプ `boolean` のブール値 (ブール値 真または偽)。

  マイナーバージョンのパッチを自動的に適用することを指定します。
- AvailabilityZone — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  DB インスタンスを配置するアベイラビリティーゾーンの名前を指定します。
- BackupRetentionPeriod - タイプ `integer` の整数 (符号付き 32 ビット整数)。

  自動 DB スナップショットが保持される日数を指定します。
- CACertificateIdentifier — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  この DB インスタンスの CA 証明書の識別子。

- CopyTagsToSnapshot — タイプ `boolean` のブール値 (ブール値 真または偽))。

  タグを DB インスタンスから DB インスタンスのスナップショットにコピーするかどうかを指定します。

- DBClusterIdentifier — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  DB インスタンスが DB クラスターのメンバーである場合は、DB インスタンスがメンバーとなっている DB クラスターの名前が含まれます。

- DBInstanceArn — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  DB インスタンスの Amazon リソースネーム (ARN)。

- DBInstanceClass — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  DB インスタンスのコンピューティング名とメモリ容量クラスの名前が含まれています。

- DBInstanceIdentifier — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  ユーザーが指定したデータベース識別子が含まれています。この識別子は、DB インスタンスを識別する一意のキーです。

- DbInstancePort - タイプ `integer` の整数 (符号付き 32 ビット整数)。

  DB インスタンスがリッスンするポートを指定します。DB インスタンスが DB クラスターの一部である場合は、DB クラスターのポートとは別のポートを指定できます。

- DBInstanceStatus — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  このデータベースの現在の状態を指定します。

- DbiResourceId — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  DB インスタンスの Amazon リージョン固有のイミュータブルな識別子。この識別子は、DB インスタンスの Amazon KMS キーにアクセスするたびに Amazon CloudTrail ログエントリに記録されます。

- DBName — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  データベース名。

- DBParameterGroups — これは DBParameterGroupStatus オブジェクトの配列です。

  この DB インスタンスに適用される DB パラメータグループのリストを入力します。

- DBSecurityGroups — これは DBSecurityGroupMembership オブジェクトの配列です。

DBSecurityGroup.Name と DBSecurityGroup.Status のサブ要素のみを含む DB セキュリティグループ要素の一覧を入力します。

- DBSubnetGroup - これは DBSubnetGroup オブジェクトです。

  サブネットグループ内の名前、説明、サブネットなど、DB インスタンスに関連付けられているサブネットグループに関する情報を指定します。

- DeletionProtection — BooleanOptional、タイプ: boolean (ブール値 (真または偽))。

  DB インスタンスで削除保護が有効になっているかどうかを示します。削除保護が有効な場合、インスタンスは削除できません。「DB インスタンスを削除する」を参照してください。

- DomainMemberships — これは DomainMembership オブジェクトの配列です。

  サポートされません

- EnabledCloudwatchLogsExports — これはタイプ string の文字列です (UTF-8 でエンコードされた文字列)。

  この DB インスタンスが CloudWatch Logs にエクスポートするように設定されているログタイプのリスト。

- Endpoint - これは エンドポイント オブジェクトです。

  接続エンドポイントを指定します。

- Engine — これはタイプ string の文字列です (UTF-8 でエンコードされた文字列)。

  この DB インスタンスに使用されるデータベースエンジンの名前を提供します。

- EngineVersion — これはタイプ string の文字列です (UTF-8 でエンコードされた文字列)。

  データベースエンジンのバージョンを示します。

- EnhancedMonitoringResourceArn — これはタイプ string の文字列です (UTF-8 でエンコードされた文字列)。

  DB インスタンスの拡張モニタリングメトリクスデータを受け取る Amazon CloudWatch Logs ログストリームの Amazon リソースネーム (ARN)。

- IAMDatabaseAuthenticationEnabled — タイプ boolean のブール値 (ブール値 真または偽))。

  Amazon Identity and Access Management (IAM) 認証が有効な場合は true、それ以外の場合は false です。

- InstanceCreateTime — TStamp、タイプ: `timestamp` (一般的には 1970 年 1 月 1 日の深夜 0 時からのオフセットとして定義される特定の時点)。

  DB インスタンスが作成された日時を入力します。
- Iops — これはタイプ `integer` の IntegerOptional です (符号付き 32 ビット整数)。

  プロビジョンド IOPS(I/O operations per second) を指定します。
- KmsKeyId — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  サポートなし: DB インスタンスの暗号化は、DB クラスターによって管理されます。
- LatestRestorableTime — TStamp、タイプ: `timestamp` (一般的には 1970 年 1 月 1 日の深夜 0 時からのオフセットとして定義される特定の時点)。

  特定時点への復元によりデータベースを復元できる最新の日時を指定します。
- LicenseModel — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  この DB インスタンスのライセンスモデル情報。
- MonitoringInterval — これはタイプ `integer` の IntegerOptional です (符号付き 32 ビット整数)。

  DB インスタンスについて拡張モニターメトリクスが収集される時点間の間隔 (秒単位)。
- MonitoringRoleArn — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  Neptune が拡張モニタリングメトリクスを Amazon CloudWatch Logs に送信することを許可する IAM ロールの ARN。
- MultiAZ — タイプ `boolean` のブール値 (ブール値 真または偽))。

  DB インスタンスが Multi-AZ デプロイかどうかを指定します。
- PendingModifiedValues - これは PendingModifiedValues オブジェクトです。

  DB インスタンスへの変更が保留中であることを指定します。この要素は、変更が保留中の場合にのみ含まれます。特定の変更は、サブエレメントによって識別されます。
- PreferredBackupWindow — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  `BackupRetentionPeriod` に応じた、自動バックアップが有効な場合に自動バックアップが作成される毎日の時間範囲を指定します。
- PreferredMaintenanceWindow — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

週 1 回のシステムメンテナンスを実行できる時間範囲を世界標準時 (UTC) で指定します。

- PromotionTier — これはタイプ `integer` の IntegerOptional です (符号付き 32 ビット整数)。

  既存のプライマリインスタンスの障害後に、リードレプリカをプライマリインスタンスに昇格される順序を指定する値。

- PubliclyAccessible — タイプ `boolean` のブール値 (ブール値 真または偽))。

  このフラグは、今後は使用しないでください。

- ReadReplicaDBClusterIdentifiers — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  この DB インスタンスに関連付けられているリードレプリカの 1 つ以上の DB クラスターの識別子を含みます。

- ReadReplicaDBInstanceIdentifiers — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  この DB インスタンスに関連付けられているリードレプリカの 1 つ以上の識別子を含みます。

- ReadReplicaSourceDBInstanceIdentifier — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  この DB インスタンスがリードレプリカの場合は、ソース DB インスタンスの識別子が含まれます。

- SecondaryAvailabilityZone — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  存在する場合は、マルチ AZ をサポートする DB インスタンスのセカンダリアベイラビリティーゾーンの名前を指定します。

- StatusInfos — これは DBInstanceStatusInfo オブジェクトの配列です。

  リードレプリカのステータス。インスタンスがリードレプリカではない場合、これは空白です。

- StorageEncrypted — タイプ `boolean` のブール値 (ブール値 真または偽))。

  サポートなし: DB インスタンスの暗号化は、DB クラスターによって管理されます。

- StorageType — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  DB インスタンスに関連付けられたストレージタイプを指定します。

- TdeCredentialArn — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

インスタンスが TDE 暗号化のために関連付けられているキーストアからの ARN。

- Timezone — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  サポート外。

- VpcSecurityGroups — これは VpcSecurityGroupMembership オブジェクトの配列です。

  DB インスタンスが属する VPC セキュリティグループ要素のリストを入力します。

`DBInstance` は、以下のレスポンス要素として使用されます。

- CreateDBInstance
- DeleteDBInstance
- ModifyDBInstance
- RebootDBInstance

DBInstanceStatusInfo (構造)

DB インスタンスのステータス情報のリストを入力します。

フィールド

- Message — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  インスタンスにエラーがある場合のエラーの詳細。インスタンスがエラー状態にない場合、この値は空白です。

- Normal — タイプ `boolean` のブール値 (ブール値 真または偽))。

  インスタンスが正常に動作している場合はブール値が true、インスタンスがエラー状態の場合は false です。

- Status — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  DB インスタンスのステータス。リードレプリカの StatusType の場合、値はレプリケーション中、エラー、停止、または終了です。

- StatusType — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  この値は現在「リードレプリケーション」です。

OrderableDBInstanceOption (構造)

DB インスタンスで使用可能なオプションのリストが含まれています。

このデータ型は、the section called “DescribeOrderableDBInstanceOptions” アクションのレスポンス要素として使用されます。

フィールド

- AvailabilityZones — これは AvailabilityZone オブジェクトの配列です。

  DB インスタンスのアベイラビリティーゾーンのリスト。
- DBInstanceClass — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  DB インスタンスの DB インスタンスクラス。
- Engine — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  DB インスタンスのエンジンタイプ。
- EngineVersion — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  DB インスタンスのエンジンバージョン。
- LicenseModel — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  DB インスタンスのライセンスモデル。
- MaxIopsPerDbInstance — これはタイプ `integer` の IntegerOptional です (符号付き 32 ビット整数)。

  DB インスタンスのプロビジョンド IOPS の合計最大数。
- MaxIopsPerGib— DoubleOptional、タイプ: `double` (倍精度 IEEE 754 浮動小数点数)。

  DB インスタンスの 1 GiB あたりのプロビジョンド IOPS の最大数。
- MaxStorageSize — これはタイプ `integer` の IntegerOptional です (符号付き 32 ビット整数)。

  DB インスタンスの最大ストレージサイズ。
- MinIopsPerDbInstance — これはタイプ `integer` の IntegerOptional です (符号付き 32 ビット整数)。

  DB インスタンスのプロビジョンド IOPS の合計最小数。
- MinIopsPerGib— DoubleOptional、タイプ: `double` (倍精度 IEEE 754 浮動小数点数)。

DB インスタンスの 1 GiB あたりのプロビジョンド IOPS の最小数。

- MinStorageSize — これはタイプ `integer` の IntegerOptional です (符号付き 32 ビット整数)。

  DB インスタンスの最小ストレージサイズ。
- MultiAZCapable — タイプ `boolean` のブール値 (ブール値 真または偽))。

  DB インスタンスがマルチ AZ に対応しているかどうかを示します。
- ReadReplicaCapable — タイプ `boolean` のブール値 (ブール値 真または偽))。

  DB インスタンスがリードレプリカを持つことができるかどうかを示します。
- StorageType — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  該当しません。Neptune の場合、ストレージタイプは DB クラスターレベルで管理されます。
- SupportsEnhancedMonitoring — タイプ `boolean` のブール値 (ブール値 真または偽))。

  DB インスタンスが 1 ～ 60 秒の間隔で拡張モニタリングをサポートするかどうかを示します。
- SupportsGlobalDatabases — タイプ `boolean` のブール値 (ブール値 真または偽))。

  Neptune グローバルデータベースを他の DB エンジン属性の特定の組み合わせで使用できるかどうかを示す値。
- SupportsIAMDatabaseAuthentication — タイプ `boolean` のブール値 (ブール値 真または偽))。

  DB インスタンスが IAM データベース認証をサポートするかどうかを示します。
- SupportsIops — タイプ `boolean` のブール値 (ブール値 真または偽))。

  DB インスタンスがプロビジョンド IOPS をサポートするかどうかを示します。
- SupportsStorageEncryption — タイプ `boolean` のブール値 (ブール値 真または偽))。

  DB インスタンスが暗号化ストレージをサポートするかどうかを示します。
- Vpc — タイプ `boolean` のブール値 (ブール値 真または偽))。

  DB インスタンスが VPC 内にあるかどうかを示します。

PendingModifiedValues (構造)

このデータ型は、the section called “ModifyDBInstance” アクションのレスポンス要素として使用されます。

フィールド

- AllocatedStorage — これはタイプ `integer` の IntegerOptional です (符号付き 32 ビット整数)。

  適用される、または現在適用されている DB インスタンスの新しい `AllocatedStorage` サイズが含まれています。

- BackupRetentionPeriod — これはタイプ `integer` の IntegerOptional です (符号付き 32 ビット整数)。

  自動バックアップを保持する保留日数を指定します。

- CACertificateIdentifier — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  DB インスタンスの CA 証明書の識別子を指定します。

- DBInstanceClass — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  適用される、または現在適用されている DB インスタンスの新しい `DBInstanceClass` が含まれています。

- DBInstanceIdentifier — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  適用される、または現在適用されている DB インスタンスの新しい `DBInstanceIdentifier` が含まれています。

- DBSubnetGroupName — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  DB インスタンスの新しい DB サブネットグループ。

- EngineVersion — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  データベースエンジンのバージョンを示します。

- Iops — これはタイプ `integer` の IntegerOptional です (符号付き 32 ビット整数)。

  適用される、または現在適用されている DB インスタンスの新しいプロビジョンド IOPS 値を指定します。

- MultiAZ — これは BooleanOptional、タイプ: `boolean` です (ブール値 (真または偽))。

  Single-AZ DB インスタンスをマルチ AZ 配置に変更することを示します。

- PendingCloudwatchLogsExports - これは PendingCloudwatchLogsExports オブジェクトです。

この PendingCloudwatchLogsExports 構造は、CloudWatch ログを有効および無効にする保留中の変更を指定します。

- Port — これはタイプ integer の IntegerOptional です (符号付き 32 ビット整数)。

  DB インスタンスの保留ポートを指定します。

- StorageType — これはタイプ string の文字列です (UTF-8 でエンコードされた文字列)。

  該当しません。Neptune の場合、ストレージタイプは DB クラスターレベルで管理されます。

ValidStorageOptions (構造)

該当しません。Neptune の場合、ストレージタイプは DB クラスターレベルで管理されます。

フィールド

- IopsToStorageRatio — これは DoubleRange オブジェクトの配列です。

  該当しません。Neptune の場合、ストレージタイプは DB クラスターレベルで管理されます。

- ProvisionedIops — これは [Range] (範囲) オブジェクトの配列です。

  該当しません。Neptune の場合、ストレージタイプは DB クラスターレベルで管理されます。

- StorageSize — これは [Range] (範囲) オブジェクトの配列です。

  該当しません。Neptune の場合、ストレージタイプは DB クラスターレベルで管理されます。

- StorageType — これはタイプ string の文字列です (UTF-8 でエンコードされた文字列)。

  該当しません。Neptune の場合、ストレージタイプは DB クラスターレベルで管理されます。

ValidDBInstanceModificationsMessage (構造)

DB インスタンスに加えることができる有効な変更に関する情報。the section called "DescribeValidDBInstanceModifications" アクションへの呼び出しが成功した結果が含まれています。the section called "ModifyDBInstance" を呼び出すときにこの情報を使用できます。

フィールド

- Storage — これは ValidStorageOptions オブジェクトの配列です。

DB インスタンスの有効なストレージオプション。

`ValidDBInstanceModificationsMessage` は、以下のレスポンス要素として使用されます。

- DescribeValidDBInstanceModifications

Neptune パラメータ API

アクション:

- CopyDBParameterGroup (アクション)
- CopyDBClusterParameterGroup (アクション)
- CreateDBParameterGroup (アクション)
- CreateDBClusterParameterGroup (アクション)
- DeleteDBParameterGroup (アクション)
- DeleteDBClusterParameterGroup (アクション)
- ModifyDBParameterGroup (アクション)
- ModifyDBClusterParameterGroup (アクション)
- ResetDBParameterGroup (アクション)
- ResetDBClusterParameterGroup (アクション)
- DescribeDBParameters (アクション)
- DescribeDBParameterGroups (アクション)
- DescribeDBClusterParameters (アクション)
- DescribeDBClusterParameterGroups (アクション)
- DescribeEngineDefaultParameters (アクション)
- DescribeEngineDefaultClusterParameters (アクション)

構造:

- パラメータ (構造)
- DBParameterGroup (構造)

- DBClusterParameterGroup (構造)
- DBParameterGroupStatus (構造)

CopyDBParameterGroup (アクション)

この API の AWS CLI 名は `copy-db-parameter-group` です。

指定された DB パラメータグループをコピーします。

リクエスト

- SourceDBParameterGroupIdentifier (CLI では: `--source-db-parameter-group-identifier`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB パラメータグループの識別子または ARN。ARN 作成の詳細については、「Amazon リソースネーム (ARN) の構築」を参照してください。

  制約:

  - 有効な DB パラメータグループを指定する必要があります。
  - DB パラメータグループの識別子を指定する必要があります。例: `my-db-param-group`、または有効な ARN。
- Tags (CLI では: `--tags`) - Tag オブジェクトの配列。

  タグは、コピーされた DB パラメータグループに割り当てられます。
- TargetDBParameterGroupDescription (CLI では: `--target-db-parameter-group-description`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  コピーされた DB パラメータグループの説明。
- TargetDBParameterGroupIdentifier (CLI では: `--target-db-parameter-group-identifier`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  コピーされた DB パラメータグループの識別子。

  制約:

  - null、空、または空白にすることはできません。
  - 1 ~ 255 の文字、数字またはハイフンを使用する必要があります。
  - 1 字目は文字である必要があります。

- ハイフンを、文字列の最後に使用したり、2 つ続けて使用したりすることはできません。

例: `my-db-parameter-group`

レスポンス

Amazon Neptune DB パラメータグループの詳細が含まれています。

このデータ型は、the section called “DescribeDBParameterGroups” アクションのレスポンス要素として使用されます。

- DBParameterGroupArn — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB パラメータグループの Amazon リソースネーム (ARN)。
- DBParameterGroupFamily — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターパラメータグループと互換性がある DB パラメータグループファミリーの名前を指定します。
- DBParameterGroupName — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB パラメータグループの名前を入力します。
- Description — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB パラメータグループに対するユーザー定義の説明を指定します。

エラー

- DBParameterGroupNotFoundFault
- DBParameterGroupAlreadyExistsFault
- DBParameterGroupQuotaExceededFault

CopyDBClusterParameterGroup (アクション)

この API の AWS CLI 名は `copy-db-cluster-parameter-group` です。

指定された DB クラスターパラメータグループをコピーします。

リクエスト

- SourceDBClusterParameterGroupIdentifier (CLI では: `--source-db-cluster-parameter-group-identifier`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  識別子またはソース DB クラスターのパラメータグループの Amazon リソースネーム (ARN)。ARN 作成の詳細については、「Amazon リソースネーム (ARN) の構築」を参照してください。

  制約:

  - 有効な DB クラスターグループを指定する必要があります。
  - ソース DB クラスターパラメータグループがコピーと同じ Amazon リージョンにある場合は、有効な DB パラメータグループ識別子を指定します。例: `my-db-cluster-param-group` または有効な ARN。
  - ソース DB パラメータグループがコピーとは異なる Amazon リージョンにある場合は、有効な DB クラスターパラメータグループ ARN を指定します。例: `arn:aws:rds:us-east-1:123456789012:cluster-pg:custom-cluster-group1`。
- Tags (CLI では: `--tags`) - Tag オブジェクトの配列。

  コピーされた DB クラスターパラメータグループに割り当てられるタグ。
- TargetDBClusterParameterGroupDescription (CLI では: `--target-db-cluster-parameter-group-description`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  コピーされた DB クラスターパラメータグループの説明。
- TargetDBClusterParameterGroupIdentifier (CLI では: `--target-db-cluster-parameter-group-identifier`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  コピーされた DB クラスターパラメータグループの識別子。

  制約:

  - null、空、または空白にすることはできません。
  - 1 ~ 255 の文字、数字またはハイフンを使用する必要があります。
  - 1 字目は文字である必要があります。
  - ハイフンを、文字列の最後に使用したり、2 つ続けて使用したりすることはできません

  例: `my-cluster-param-group1`

レスポンス

Amazon Neptune DB クラスターパラメータグループの詳細が含まれています。

このデータ型は、the section called “DescribeDBClusterParameterGroups” アクションのレスポンス要素として使用されます。

- DBClusterParameterGroupArn — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターパラメータグループの Amazon リソースネーム (ARN)。
- DBClusterParameterGroupName — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターパラメータグループの名前を指定します。
- DBParameterGroupFamily — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターパラメータグループと互換性がある DB クラスターパラメータグループファミリーの名前を指定します。
- Description — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターパラメータグループに対するユーザー定義の説明を指定します。

エラー

- DBParameterGroupNotFoundFault
- DBParameterGroupQuotaExceededFault
- DBParameterGroupAlreadyExistsFault

CreateDBParameterGroup (アクション)

> この API の AWS CLI 名は `create-db-parameter-group` です。

DB パラメータグループを作成します。

DB パラメータグループは、最初は DB インスタンスで使用されるデータベースエンジンのデフォルトパラメータで作成されます。パラメータのいずれかにカスタム値を指定するには、ModifyDBParameterGroup を使用して作成した後でグループを変更する必要があります。DB パラメータグループを作成したら、ModifyDBInstance を使用して DB インスタンスに関連付ける必要があります。新しい DB パラメータグループを実行中の DB インスタンスに関連付ける場合、フェイ

ルオーバーせずに新しい DB パラメータグループと関連付け済みの設定を有効にするには、DB インスタンスを再起動する必要があります。

> **Important**
>
> DB パラメータグループの作成後、その DB パラメータグループをデフォルトのパラメータグループとして使用する最初の DB インスタンスが作成されるまで、5 分以上かかります。その間に、Amazon Neptune によってインスタンスの作成アクションが完了し、そのパラメータグループが新しい DB インスタンスのデフォルトとして使用されるようになります。この点は、DB インスタンスのデフォルトデータベースの作成時に必須になるパラメータ (デフォルトデータベースの文字セットを定義する `character_set_database` パラメータなど) に特に重要です。Amazon Neptune コンソールの [Parameter Groups] オプション、または DescribeDBParameters コマンドを使用して、DB パラメータグループが作成または変更されたことを確認できます。

リクエスト

- DBParameterGroupFamily (CLI では: `--db-parameter-group-family`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB パラメータグループファミリーの名前。DB パラメータグループは、1 つだけの DB パラメータグループファミリーに関連付けることが可能で、その DB パラメータグループファミリーと互換性のあるデータベースエンジンおよびエンジンバージョンを実行している DB インスタンスにのみ適用できます。

- DBParameterGroupName (CLI では: `--db-parameter-group-name`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB パラメータグループの名前。

  制約:

  - 1 ~ 255 の英字、数字、ハイフンである必要があります。
  - 1 字目は文字である必要があります。
  - ハイフンを、文字列の最後に使用したり、2 つ続けて使用したりすることはできません

> **Note**
>
> この値は小文字で保存されます。

- Description (CLI では: `--description`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB パラメータグループの説明。

- Tags (CLI では: `--tags`) - Tag オブジェクトの配列。

  タグは、新しい DB パラメータグループに割り当てられます。

レスポンス

Amazon Neptune DB パラメータグループの詳細が含まれています。

このデータ型は、the section called “DescribeDBParameterGroups” アクションのレスポンス要素として使用されます。

- DBParameterGroupArn — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB パラメータグループの Amazon リソースネーム (ARN)。

- DBParameterGroupFamily — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターパラメータグループと互換性がある DB パラメータグループファミリーの名前を指定します。

- DBParameterGroupName — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB パラメータグループの名前を入力します。

- Description — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB パラメータグループに対するユーザー定義の説明を指定します。

エラー

- DBParameterGroupQuotaExceededFault
- DBParameterGroupAlreadyExistsFault

CreateDBClusterParameterGroup (アクション)

この API の AWS CLI 名は `create-db-cluster-parameter-group` です。

新しい DB クラスターのパラメータグループを作成します。

DB クラスターパラメータグループのパラメータは、DB クラスター内のすべてのインスタンスに適用されます。

DB クラスターパラメータグループは、最初は DB クラスターのインスタンスで使用されるデータベースエンジンのデフォルトパラメータで作成されます。パラメータのいずれかにカスタム値を指定するには、the section called “ModifyDBClusterParameterGroup” を使用して作成後にグループを変更する必要があります。DB クラスターパラメータグループを作成したら、the section called “ModifyDBCluster” を使用して使用する DB クラスターに関連付ける必要があります。新しい DB クラスターパラメータグループを実行中の DB クラスターに関連付ける場合、フェイルオーバーせずに新しい DB パラメータグループと関連付け済みの設定を有効にするには、DB クラスターの DB インスタンスを再起動する必要があります。

> **Important**
>
> DB クラスターパラメータグループの作成後、デフォルトのパラメータグループとしてその DB クラスターパラメータグループを使用する最初の DB クラスターが作成されるまで、5 分以上かかります。その間に、Amazon Neptune によってインスタンスの作成アクションが完了し、その DB クラスターパラメータグループが新しい DB クラスターのデフォルトとして使用されるようになります。この点は、DB クラスターのデフォルトデータベースの作成時に必須になるパラメータ (デフォルトデータベースの文字セットを定義する `character_set_database` パラメータなど) に特に重要です。Amazon Neptune コンソールの [Parameter Groups] オプション、または the section called “DescribeDBClusterParameters” コマンドを使用して、DB クラスターパラメータグループが作成または変更されたことを確認できます。

リクエスト

- DBClusterParameterGroupName (CLI では: `--db-cluster-parameter-group-name`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターパラメータグループの名前。

  制約:

- 既存の DBClusterParameterGroup の名前と一致する必要があります。

> **Note**
>
> この値は小文字で保存されます。

- DBParameterGroupFamily (CLI では: `--db-parameter-group-family`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターパラメータグループファミリーの名前。DB クラスターパラメータグループは、1 つだけの DB クラスターパラメータグループファミリーに関連付けることが可能で、その DB クラスターパラメータグループファミリーと互換性のあるデータベースエンジンおよびエンジンバージョンを実行している DB クラスターにのみ適用できます。
- Description (CLI では: `--description`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターパラメータグループの説明。
- Tags (CLI では: `--tags`) - Tag オブジェクトの配列。

  新しい DB クラスターパラメータグループに割り当てられるタグ。

レスポンス

Amazon Neptune DB クラスターパラメータグループの詳細が含まれています。

このデータ型は、the section called “DescribeDBClusterParameterGroups” アクションのレスポンス要素として使用されます。

- DBClusterParameterGroupArn — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターパラメータグループの Amazon リソースネーム (ARN)。
- DBClusterParameterGroupName — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターパラメータグループの名前を指定します。
- DBParameterGroupFamily — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターパラメータグループと互換性がある DB クラスターパラメータグループファミリーの名前を指定します。

- Description — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターパラメータグループに対するユーザー定義の説明を指定します。

エラー

- DBParameterGroupQuotaExceededFault
- DBParameterGroupAlreadyExistsFault

DeleteDBParameterGroup (アクション)

> この API の AWS CLI 名は `delete-db-parameter-group` です。

指定された DBParameterGroup を削除します。削除する DBParameterGroup は、どの DB インスタンスにも関連付けることができません。

リクエスト

- DBParameterGroupName (CLI では: `--db-parameter-group-name`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB パラメータグループの名前。

  制約:

  - 既存の DB パラメータグループの名前にする必要があります。
  - デフォルトの DB パラメータグループを削除することはできません。
  - DB インスタンスに関連付けることはできません。

レスポンス

- 応答パラメータはありません。

エラー

- InvalidDBParameterGroupStateFault
- DBParameterGroupNotFoundFault

DeleteDBClusterParameterGroup (アクション)

この API の AWS CLI 名は `delete-db-cluster-parameter-group` です。

指定された DB クラスターパラメータグループを削除します。削除する DB クラスターパラメータグループは、どの DB クラスターにも関連付けることができません。

リクエスト

- DBClusterParameterGroupName (CLI では: `--db-cluster-parameter-group-name`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターパラメータグループの名前。

  制約:

  - 既存の DB クラスターパラメータグループの名前にする必要があります。
  - デフォルトの DB クラスターパラメータグループを削除することはできません。
  - DB クラスターに関連付けることはできません。

レスポンス

- 応答パラメータはありません。

エラー

- InvalidDBParameterGroupStateFault
- DBParameterGroupNotFoundFault

ModifyDBParameterGroup (アクション)

この API の AWS CLI 名は `modify-db-parameter-group` です。

DB パラメータグループのパラメータを変更します。複数のパラメータを変更するには、以下のリストを送信します: `ParameterName`、`ParameterValue`、`ApplyMethod`。最大 20 個のパラメータを単一のリクエストで修正できます。

> **Note**
>
> 動的パラメータの変更は直ちに適用されます。静的パラメータの変更の場合、変更を有効にするには、パラメータグループに関連付けられている DB インスタンスにフェイルオーバーせずに再起動する必要があります。

> **Important**
>
> DB パラメータグループの変更後、デフォルトのパラメータグループとしてその DB パラメータグループを使用する最初の DB インスタンスが作成されるまで、5 分以上かかります。その間に、Amazon Neptune によってインスタンスの変更アクションが完了し、そのパラメータグループが新しい DB インスタンスのデフォルトとして使用されるようになります。この点は、DB インスタンスのデフォルトデータベースの作成時に必須になるパラメータ (デフォルトデータベースの文字セットを定義する `character_set_database` パラメータなど) に特に重要です。Amazon Neptune コンソールの [Parameter Groups] オプション、または DescribeDBParameters コマンドを使用して、DB パラメータグループが作成または変更されたことを確認できます。

リクエスト

- DBParameterGroupName (CLI では: `--db-parameter-group-name`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB パラメータグループの名前。

  制約:

  - 指定した場合、既存の DBParameterGroup の名前と一致する必要があります。
- Parameters (CLI では: `--parameters`) - 必須: パラメータ オブジェクトの配列。

  パラメータ名、値、およびパラメータ更新用の apply メソッドの配列。少なくとも 1 つのパラメータ名、値、および適用方法を指定する必要があります。後続の引数はオプションです。最大 20 個のパラメータを単一のリクエストで修正できます。

  有効な値 (適用方法): `immediate | pending-reboot`

> **Note**
>
> 即時値は動的パラメータでのみ使用できます。pending-reboot 値は動的パラメータと静的パラメータの両方に使用でき、変更はフェイルオーバーなしで DB インスタンスを再起動すると適用されます。

レスポンス

- DBParameterGroupName — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB パラメータグループの名前を入力します。

エラー

- DBParameterGroupNotFoundFault
- InvalidDBParameterGroupStateFault

ModifyDBClusterParameterGroup (アクション)

この API の AWS CLI 名は `modify-db-cluster-parameter-group` です。

DB クラスターパラメータグループのパラメータを変更します。複数のパラメータを変更するには、以下のリストを送信します: `ParameterName`、`ParameterValue`、`ApplyMethod`。最大 20 個のパラメータを単一のリクエストで修正できます。

> **Note**
>
> 動的パラメータの変更は直ちに適用されます。静的パラメータの変更の場合、変更を有効にするには、パラメータグループに関連付けられている DB クラスターにフェイルオーバーせずに再起動する必要があります。

> **Important**
>
> DB クラスターパラメータグループの作成後、デフォルトのパラメータグループとしてその DB クラスターパラメータグループを使用する最初の DB クラスターが作成されるまで、5

分以上かかります。その間に、Amazon Neptune によってインスタンスの作成アクションが完了し、そのパラメータグループが新しい DB クラスターのデフォルトとして使用されるようになります。この点は、DB クラスターのデフォルトデータベースの作成時に必須になるパラメータ (デフォルトデータベースの文字セットを定義する `character_set_database` パラメータなど) に特に重要です。Amazon Neptune コンソールの [Parameter Groups] オプション、または the section called “DescribeDBClusterParameters” コマンドを使用して、DB クラスターパラメータグループが作成または変更されたことを確認できます。

リクエスト

- DBClusterParameterGroupName (CLI では: `--db-cluster-parameter-group-name`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  変更する DB クラスターパラメータグループの名前。
- Parameters (CLI では: `--parameters`) - 必須: パラメータ オブジェクトの配列。

  変更する DB クラスターパラメータグループ内のパラメータのリスト。

レスポンス

- DBClusterParameterGroupName — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターパラメータグループの名前。

  制約:
  - 1 ~ 255 文字の英字または数字でなければなりません。
  - 1 字目は文字である必要があります。
  - ハイフンを、文字列の最後に使用したり、2 つ続けて使用したりすることはできません

> **Note**
>
> この値は小文字で保存されます。

エラー

- DBParameterGroupNotFoundFault
- InvalidDBParameterGroupStateFault

ResetDBParameterGroup (アクション)

この API の AWS CLI 名は `reset-db-parameter-group` です。

DB パラメータグループのパラメータをエンジン/システムのデフォルト値に変更します。特定のパラメータをリセットするには、ParameterName と ApplyMethod のリストを指定します。DB パラメータグループ全体をリセットするには、DBParameterGroup 名と ResetAllParameters パラメータを指定します。グループ全体をリセットすると、動的パラメータが即座に更新され、静的パラメータが `pending-reboot` に設定されて、次回の DB インスタンスの再起動や `RebootDBInstance` リクエストで有効になります。

リクエスト

- DBParameterGroupName (CLI では: `--db-parameter-group-name`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB パラメータグループの名前。

  制約:

  - 既存の DBParameterGroup の名前と一致する必要があります。
- Parameters (CLI では: `--parameters`) - パラメータ オブジェクトの配列。

  DB パラメータグループ全体をリセットするには、DBParameterGroup 名と ResetAllParameters パラメータを指定します。特定のパラメータをリセットするには、ParameterName と ApplyMethod のリストを指定します。最大 20 個のパラメータを単一のリクエストで修正できます。

  有効な値 (適用方法): `pending-reboot`
- ResetAllParameters (CLI では: `--reset-all-parameters`) — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  DB パラメータグループ内のすべてのパラメータをデフォルト値にリセットする (`true`) かしないか (`false`) を指定します。

デフォルト: `true`

レスポンス

- DBParameterGroupName — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB パラメータグループの名前を入力します。

エラー

- InvalidDBParameterGroupStateFault
- DBParameterGroupNotFoundFault

ResetDBClusterParameterGroup (アクション)

この API の AWS CLI 名は `reset-db-cluster-parameter-group` です。

DB クラスターパラメータグループのパラメータをデフォルト値に変更します。特定のパラメータをリセットするには、`ParameterName` と `ApplyMethod` のリストを送信します。DB クラスターパラメータグループ全体をリセットするには、`DBClusterParameterGroupName` 名と `ResetAllParameters` パラメータを指定します。

グループ全体をリセットすると、動的パラメータが即座に更新され、静的パラメータが `pending-reboot` に設定されて、次回の DB インスタンスの再起動や the section called “RebootDBInstance” リクエストで有効になります。更新された静的パラメータを適用する DB クラスター内のすべての DB インスタンスに対して、the section called “RebootDBInstance” を呼び出す必要があります。

リクエスト

- DBClusterParameterGroupName (CLI では: `--db-cluster-parameter-group-name`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  リセットする DB クラスターパラメータグループの名前。
- Parameters (CLI では: `--parameters`) - パラメータ オブジェクトの配列。

  デフォルト値にリセットする DB クラスターパラメータグループ内のパラメータ名のリスト。`ResetAllParameters` パラメータが `true` に設定されている場合は、このパラメータを設定することはできません。

- ResetAllParameters (CLI では: `--reset-all-parameters`) — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  DB クラスターパラメータグループ内のすべてのパラメータをデフォルト値にリセットするには `true` に設定し、それ以外の場合は `false` に設定します。`Parameters` パラメータに指定されたパラメータ名のリストがある場合、このパラメータは使用できません。

レスポンス

- DBClusterParameterGroupName — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターパラメータグループの名前。

  制約:

  - 1 ~ 255 文字の英字または数字でなければなりません。
  - 1 字目は文字である必要があります。
  - ハイフンを、文字列の最後に使用したり、2 つ続けて使用したりすることはできません

  > **Note**
  > この値は小文字で保存されます。

エラー

- InvalidDBParameterGroupStateFault
- DBParameterGroupNotFoundFault

DescribeDBParameters (アクション)

この API の AWS CLI 名は `describe-db-parameters` です。

特定の DB パラメータグループの詳細なパラメータリストを返します。

リクエスト

- DBParameterGroupName (CLI では: `--db-parameter-group-name`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  詳細を返す特定の DB パラメータグループの名前。

  制約:

  - 指定した場合、既存の DBParameterGroup の名前と一致する必要があります。
- Filters (CLI では: `--filters`) - フィルター オブジェクトの配列。

  このパラメータは、現在サポートされていません。
- Marker (CLI では: `--marker`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  以前の `DescribeDBParameters` リクエストによって提供されたオプションのページ割りトークン。このパラメータを指定した場合、レスポンスには `MaxRecords` で指定された値まで、マーカーを超えるレコードのみが含まれます。
- MaxRecords (CLI では: `--max-records`) — IntegerOptional、タイプ: `integer` (符号付き 32 ビット整数)。

  レスポンスに含めるレコードの最大数。指定された `MaxRecords` の値よりも多くのレコードが存在する場合、マーカーと呼ばれるページ割りトークンがレスポンスに含まれるため、残りの結果を取得できます。

  デフォルト: 100

  制約: 最小 20、最大 100。
- Source (CLI では: `--source`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  返すパラメータのタイプ。

  デフォルト: 返されるすべてのパラメータタイプ

  有効な値: `user | system | engine-default`

レスポンス

- Marker — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

以前のリクエストによって提供されたオプションのページ割りトークン。このパラメータを指定した場合、レスポンスには MaxRecords で指定された値まで、マーカーを超えるレコードのみが含まれます。

- Parameters – パラメータ オブジェクトの配列。

  the section called “パラメータ” 値のリスト。

エラー

- DBParameterGroupNotFoundFault

DescribeDBParameterGroups (アクション)

この API の AWS CLI 名は describe-db-parameter-groups です。

DBParameterGroup の説明のリストを返します。DBParameterGroupName を指定した場合、リストには指定した DB パラメータグループの説明のみが含まれます。

リクエスト

- DBParameterGroupName (CLI では: --db-parameter-group-name) — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  詳細を返す特定の DB パラメータグループの名前。

  制約:

  - 指定した場合、既存の DBClusterParameterGroup の名前と一致する必要があります。
- Filters (CLI では: --filters) - フィルター オブジェクトの配列。

  このパラメータは、現在サポートされていません。
- Marker (CLI では: --marker) — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  以前の DescribeDBParameterGroups リクエストによって提供されたオプションのページ割りトークン。このパラメータを指定した場合、レスポンスには MaxRecords で指定された値まで、マーカーを超えるレコードのみが含まれます。
- MaxRecords (CLI では: --max-records) — IntegerOptional、タイプ: integer (符号付き 32 ビット整数)。

レスポンスに含めるレコードの最大数。指定された MaxRecords の値よりも多くのレコードが存在する場合、マーカーと呼ばれるページ割りトークンがレスポンスに含まれるため、残りの結果を取得できます。

デフォルト: 100

制約: 最小 20、最大 100。

レスポンス

- DBParameterGroups – DBParameterGroup オブジェクトの配列。

  the section called “DBParameterGroup” インスタンスのリスト。
- Marker — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  以前のリクエストによって提供されたオプションのページ割りトークン。このパラメータを指定した場合、レスポンスには MaxRecords で指定された値まで、マーカーを超えるレコードのみが含まれます。

エラー

- DBParameterGroupNotFoundFault

DescribeDBClusterParameters (アクション)

この API の AWS CLI 名は describe-db-cluster-parameters です。

特定の DB クラスターパラメータグループの詳細なパラメータリストを返します。

リクエスト

- DBClusterParameterGroupName (CLI では: --db-cluster-parameter-group-name) — 必須: タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  パラメータの詳細を返す特定の DB クラスターパラメータグループの名前。

  制約:
  - 指定した場合、既存の DBClusterParameterGroup の名前と一致する必要があります。

- Filters (CLI では: `--filters`) - フィルター オブジェクトの配列。

  このパラメータは、現在サポートされていません。

- Marker (CLI では: `--marker`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  以前の `DescribeDBClusterParameters` リクエストによって提供されたオプションのページ割りトークン。このパラメータを指定した場合、レスポンスには `MaxRecords` で指定された値まで、マーカーを超えるレコードのみが含まれます。

- MaxRecords (CLI では: `--max-records`) — IntegerOptional、タイプ: `integer` (符号付き 32 ビット整数)。

  レスポンスに含めるレコードの最大数。指定された `MaxRecords` の値よりも多くのレコードが存在する場合、マーカーと呼ばれるページ割りトークンがレスポンスに含まれるため、残りの結果を取得できます。

  デフォルト: 100

  制約: 最小 20、最大 100。

- Source (CLI では: `--source`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  特定のソースのパラメータのみを返すことを示す値。パラメータのソースは `engine`、`service`、または `customer` のいずれかとすることができます。

レスポンス

- Marker — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  以前の DescribeDBClusterParameters リクエストによって提供されたオプションのページ分割トークン。このパラメータを指定した場合、レスポンスには `MaxRecords` で指定された値まで、マーカーを超えるレコードのみが含まれます。

- Parameters – パラメータ オブジェクトの配列。

  DB クラスターパラメータグループのパラメータのリストを入力します。

エラー

- DBParameterGroupNotFoundFault

DescribeDBClusterParameterGroups (アクション)

この API の AWS CLI 名は `describe-db-cluster-parameter-groups` です。

`DBClusterParameterGroup` の説明のリストを返します。`DBClusterParameterGroupName` パラメータを指定した場合、リストには指定した DB クラスターパラメータグループの説明のみが含まれます。

リクエスト

- DBClusterParameterGroupName (CLI では: `--db-cluster-parameter-group-name`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  詳細を返す特定の DB クラスターパラメータグループの名前。

  制約:

  - 指定した場合、既存の DBClusterParameterGroup の名前と一致する必要があります。
- Filters (CLI では: `--filters`) - フィルター オブジェクトの配列。

  このパラメータは、現在サポートされていません。
- Marker (CLI では: `--marker`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  以前の `DescribeDBClusterParameterGroups` リクエストによって提供されたオプションのページ割りトークン。このパラメータを指定した場合、レスポンスには `MaxRecords` で指定された値まで、マーカーを超えるレコードのみが含まれます。
- MaxRecords (CLI では: `--max-records`) — IntegerOptional、タイプ: `integer` (符号付き 32 ビット整数)。

  レスポンスに含めるレコードの最大数。指定された `MaxRecords` の値よりも多くのレコードが存在する場合、マーカーと呼ばれるページ割りトークンがレスポンスに含まれるため、残りの結果を取得できます。

  デフォルト: 100

  制約: 最小 20、最大 100。

レスポンス

- DBClusterParameterGroups – DBClusterParameterGroup オブジェクトの配列。

DB クラスターパラメータグループのリスト。

- Marker — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  以前の `DescribeDBClusterParameterGroups` リクエストによって提供されたオプションのページ割りトークン。このパラメータを指定した場合、レスポンスには `MaxRecords` で指定された値まで、マーカーを超えるレコードのみが含まれます。

エラー

- DBParameterGroupNotFoundFault

DescribeEngineDefaultParameters (アクション)

この API の AWS CLI 名は `describe-engine-default-parameters` です。

指定されたデータベースエンジンのデフォルトのエンジンおよびシステムパラメータ情報を返します。

リクエスト

- DBParameterGroupFamily (CLI では: `--db-parameter-group-family`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB パラメータグループファミリーの名前。
- Filters (CLI では: `--filters`) - フィルター オブジェクトの配列。

  現在サポートされていません。
- Marker (CLI では: `--marker`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  以前の `DescribeEngineDefaultParameters` リクエストによって提供されたオプションのページ割りトークン。このパラメータを指定した場合、レスポンスには `MaxRecords` で指定された値まで、マーカーを超えるレコードのみが含まれます。
- MaxRecords (CLI では: `--max-records`) — IntegerOptional、タイプ: `integer` (符号付き 32 ビット整数)。

  レスポンスに含めるレコードの最大数。指定された `MaxRecords` の値よりも多くのレコードが存在する場合、マーカーと呼ばれるページ割りトークンがレスポンスに含まれるため、残りの結果を取得できます。

デフォルト: 100

制約: 最小 20、最大 100。

レスポンス

the section called “DescribeEngineDefaultParameters” アクションの呼び出しが成功した結果が含まれています。

- DBParameterGroupFamily — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  エンジンのデフォルトパラメータが適用される DB パラメータグループファミリーの名前を指定します。
- Marker — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  以前の EngineDefaults リクエストによって提供されたオプションのページ分割トークン。このパラメータを指定した場合、レスポンスには `MaxRecords` で指定された値まで、マーカーを超えるレコードのみが含まれます。
- Parameters – パラメータ オブジェクトの配列。

  エンジンのデフォルトパラメータのリストが含まれます。

DescribeEngineDefaultClusterParameters (アクション)

この API の AWS CLI 名は `describe-engine-default-cluster-parameters` です。

クラスターのデータベースエンジンのデフォルトのエンジンおよびシステムパラメータ情報を返します。

リクエスト

- DBParameterGroupFamily (CLI では: `--db-parameter-group-family`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  エンジンパラメータ情報を返す DB クラスターパラメータグループファミリーの名前。
- Filters (CLI では: `--filters`) - フィルター オブジェクトの配列。

  このパラメータは、現在サポートされていません。

- Marker (CLI では: `--marker`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  以前の `DescribeEngineDefaultClusterParameters` リクエストによって提供されたオプションのページ割りトークン。このパラメータを指定した場合、レスポンスには `MaxRecords` で指定された値まで、マーカーを超えるレコードのみが含まれます。
- MaxRecords (CLI では: `--max-records`) — IntegerOptional、タイプ: `integer` (符号付き 32 ビット整数)。

  レスポンスに含めるレコードの最大数。指定された `MaxRecords` の値よりも多くのレコードが存在する場合、マーカーと呼ばれるページ割りトークンがレスポンスに含まれるため、残りの結果を取得できます。

  デフォルト: 100

  制約: 最小 20、最大 100。

レスポンス

the section called “DescribeEngineDefaultParameters” アクションの呼び出しが成功した結果が含まれています。

- DBParameterGroupFamily — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  エンジンのデフォルトパラメータが適用される DB パラメータグループファミリーの名前を指定します。
- Marker — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  以前の EngineDefaults リクエストによって提供されたオプションのページ分割トークン。このパラメータを指定した場合、レスポンスには `MaxRecords` で指定された値まで、マーカーを超えるレコードのみが含まれます。
- Parameters – パラメータ オブジェクトの配列。

  エンジンのデフォルトパラメータのリストが含まれます。

構造:

パラメータ (構造)

パラメータを指定します。

フィールド

- AllowedValues — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  パラメータの有効な値の範囲を指定します。

- ApplyMethod — ApplyMethod、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  パラメータの更新を適用するタイミングを指定します。

- ApplyType — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  エンジン固有のパラメータタイプを指定します。

- DataType — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  パラメータの有効なデータ型を指定します。

- Description — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  パラメータの説明を入力します。

- IsModifiable — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  パラメータを変更できるか (`true`) 、できない (`false`) かを示します。いくつかのパラメータには、それらの変更を妨げるセキュリティ上または運用上の影響があります。

- MinimumEngineVersion — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  パラメータを適用できる最も古いエンジンバージョン。

- ParameterName — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  パラメータの名前を指定します。

- ParameterValue — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  パラメータの値を指定します。

- Source — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  パラメータ値のソースを示します。

DBParameterGroup (構造)

Amazon Neptune DB パラメータグループの詳細が含まれています。

このデータ型は、the section called “DescribeDBParameterGroups” アクションのレスポンス要素として使用されます。

フィールド

- DBParameterGroupArn — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB パラメータグループの Amazon リソースネーム (ARN)。
- DBParameterGroupFamily — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターパラメータグループと互換性がある DB パラメータグループファミリーの名前を指定します。
- DBParameterGroupName — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB パラメータグループの名前を入力します。
- Description — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB パラメータグループに対するユーザー定義の説明を指定します。

`DBParameterGroup` は、以下のレスポンス要素として使用されます。

- CopyDBParameterGroup
- CreateDBParameterGroup

DBClusterParameterGroup (構造)

Amazon Neptune DB クラスターパラメータグループの詳細が含まれています。

このデータ型は、the section called “DescribeDBClusterParameterGroups” アクションのレスポンス要素として使用されます。

フィールド

- DBClusterParameterGroupArn — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターパラメータグループの Amazon リソースネーム (ARN)。
- DBClusterParameterGroupName — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

DB クラスターパラメータグループの名前を指定します。

- DBParameterGroupFamily — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターパラメータグループと互換性がある DB クラスターパラメータグループファミリーの名前を指定します。

- Description — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターパラメータグループに対するユーザー定義の説明を指定します。

`DBClusterParameterGroup` は、以下のレスポンス要素として使用されます。

- CopyDBClusterParameterGroup
- CreateDBClusterParameterGroup

DBParameterGroupStatus (構造)

DB パラメータグループのステータス。

このデータ型は、以下のアクションのレスポンス要素として使用されます。

- the section called “CreateDBInstance”
- the section called “DeleteDBInstance”
- the section called “ModifyDBInstance”
- the section called “RebootDBInstance”

フィールド

- DBParameterGroupName — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DP パラメータグループの名前。

- ParameterApplyStatus — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  パラメータ更新のステータス。

Neptune サブネット API

アクション:

- CreateDBSubnetGroup (アクション)
- DeleteDBSubnetGroup (アクション)
- ModifyDBSubnetGroup (アクション)
- DescribeDBSubnetGroups (アクション)

構造:

- サブネット (構造)
- DBSubnetGroup (構造)

CreateDBSubnetGroup (アクション)

この API の AWS CLI 名は `create-db-subnet-group` です。

新しい DB サブネットグループを作成します。DB サブネットグループには、Amazon リージョン内の少なくとも 2 つの AZ に少なくとも 1 つのサブネットが含まれている必要があります。

リクエスト

- DBSubnetGroupDescription (CLI では: `--db-subnet-group-description`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB サブネットグループの説明。
- DBSubnetGroupName (CLI では: `--db-subnet-group-name`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB サブネットグループの名前。この値は小文字で保存されます。

  制約: 最大 255 文字の英字、数字、ピリオド、アンダースコア、スペース、またはハイフンのみを使用できます。デフォルト値を使用することはできません。

  例: `mySubnetgroup`
- SubnetIds (CLI では: `--subnet-ids`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

DB サブネットグループの EC2 サブネット ID。

- Tags (CLI では: `--tags`) - Tag オブジェクトの配列。

  新しい DB サブネットグループに割り当てられるタグ。

レスポンス

Amazon Neptune DB サブネットグループの詳細が含まれています。

このデータ型は、the section called “DescribeDBSubnetGroups” アクションのレスポンス要素として使用されます。

- DBSubnetGroupArn — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB サブネットグループの Amazon リソースネーム (ARN)。
- DBSubnetGroupDescription — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB サブネットグループの説明を入力します。
- DBSubnetGroupName — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB サブネットグループの名前。
- SubnetGroupStatus — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB サブネットグループのステータスを入力します。
- Subnets – サブネット オブジェクトの配列。

  the section called “サブネット” 要素のリストが含まれます。
- VpcId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB サブネットグループの VpcId を入力します。

エラー

- DBSubnetGroupAlreadyExistsFault
- DBSubnetGroupQuotaExceededFault
- DBSubnetQuotaExceededFault
- DBSubnetGroupDoesNotCoverEnoughAZs

- InvalidSubnet

DeleteDBSubnetGroup (アクション)

この API の AWS CLI 名は `delete-db-subnet-group` です。

DB サブネットグループを削除します。

> **Note**
>
> 指定されたデータベースサブネットグループは、どの DB インスタンスにも関連付けないでください。

リクエスト

- DBSubnetGroupName (CLI では: `--db-subnet-group-name`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  削除するデータベースサブネットグループの名前。

  > **Note**
  >
  > デフォルトのサブネットグループを削除することはできません。

  制約:

  制約: 既存の DBSubnetGroup の名前と一致する必要があります。デフォルト値を使用することはできません。

  例: `mySubnetgroup`

レスポンス

- 応答パラメータはありません。

エラー

- InvalidDBSubnetGroupStateFault
- InvalidDBSubnetStateFault
- DBSubnetGroupNotFoundFault

ModifyDBSubnetGroup (アクション)

この API の AWS CLI 名は `modify-db-subnet-group` です。

既存の DB サブネットグループを変更します。DB サブネットグループには、Amazon リージョン内の少なくとも 2 つの AZ に少なくとも 1 つのサブネットが含まれている必要があります。

リクエスト

- DBSubnetGroupDescription (CLI では: `--db-subnet-group-description`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB サブネットグループの説明。
- DBSubnetGroupName (CLI では: `--db-subnet-group-name`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB サブネットグループの名前。この値は小文字で保存されます。デフォルトのサブネットグループを変更することはできません。

  制約: 既存の DBSubnetGroup の名前と一致する必要があります。デフォルト値を使用することはできません。

  例: `mySubnetgroup`
- SubnetIds (CLI では: `--subnet-ids`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB サブネットグループの EC2 サブネット ID。

レスポンス

Amazon Neptune DB サブネットグループの詳細が含まれています。

このデータ型は、the section called “DescribeDBSubnetGroups” アクションのレスポンス要素として使用されます。

- DBSubnetGroupArn — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB サブネットグループの Amazon リソースネーム (ARN)。
- DBSubnetGroupDescription — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB サブネットグループの説明を入力します。
- DBSubnetGroupName — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB サブネットグループの名前。
- SubnetGroupStatus — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB サブネットグループのステータスを入力します。
- Subnets – サブネット オブジェクトの配列。

  the section called “サブネット” 要素のリストが含まれます。
- VpcId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB サブネットグループの VpcId を入力します。

エラー

- DBSubnetGroupNotFoundFault
- DBSubnetQuotaExceededFault
- SubnetAlreadyInUse
- DBSubnetGroupDoesNotCoverEnoughAZs
- InvalidSubnet

DescribeDBSubnetGroups (アクション)

この API の AWS CLI 名は `describe-db-subnet-groups` です。

DBSubnetGroup の説明のリストを返します。DBSubnetGroupName を指定した場合、リストには指定した DBSubnetGroup の説明のみが含まれます。

CIDR 範囲の概要については、Wikipedia のチュートリアルを参照してください。

リクエスト

- DBSubnetGroupName (CLI では: `--db-subnet-group-name`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  詳細を返す DB サブネットグループの名前。
- Filters (CLI では: `--filters`) - フィルター オブジェクトの配列。

  このパラメータは、現在サポートされていません。
- Marker (CLI では: `--marker`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  以前の DescribeDBSubnetGroups リクエストによって提供されたオプションのページ分割トークン。このパラメータを指定した場合、レスポンスには `MaxRecords` で指定された値まで、マーカーを超えるレコードのみが含まれます。
- MaxRecords (CLI では: `--max-records`) — IntegerOptional、タイプ: `integer` (符号付き 32 ビット整数)。

  レスポンスに含めるレコードの最大数。指定された `MaxRecords` の値よりも多くのレコードが存在する場合、マーカーと呼ばれるページ割りトークンがレスポンスに含まれるため、残りの結果を取得できます。

  デフォルト: 100

  制約: 最小 20、最大 100。

レスポンス

- DBSubnetGroups – DBSubnetGroup オブジェクトの配列。

  the section called “DBSubnetGroup” インスタンスのリスト。
- Marker — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  以前のリクエストによって提供されたオプションのページ割りトークン。このパラメータを指定した場合、レスポンスには `MaxRecords` で指定された値まで、マーカーを超えるレコードのみが含まれます。

エラー

- DBSubnetGroupNotFoundFault

構造:

サブネット (構造)

サブネットを指定します。

このデータ型は、the section called “DescribeDBSubnetGroups” アクションのレスポンス要素として使用されます。

フィールド

- SubnetAvailabilityZone - これは AvailabilityZone オブジェクトです。

  サブネットが属する EC2 アベイラビリティーゾーンを指定します。
- SubnetIdentifier — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  サブネットの識別子を指定します。
- SubnetStatus — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  サブネットのステータスを指定します。

DBSubnetGroup (構造)

Amazon Neptune DB サブネットグループの詳細が含まれています。

このデータ型は、the section called “DescribeDBSubnetGroups” アクションのレスポンス要素として使用されます。

フィールド

- DBSubnetGroupArn — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB サブネットグループの Amazon リソースネーム (ARN)。
- DBSubnetGroupDescription — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB サブネットグループの説明を入力します。
- DBSubnetGroupName — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB サブネットグループの名前。

- SubnetGroupStatus — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB サブネットグループのステータスを入力します。

- Subnets — これは サブネット オブジェクトの配列です。

  the section called “サブネット” 要素のリストが含まれます。

- VpcId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB サブネットグループの VpcId を入力します。

`DBSubnetGroup` は、以下のレスポンス要素として使用されます。

- CreateDBSubnetGroup
- ModifyDBSubnetGroup

Neptune スナップショット API

アクション:

- CreateDBClusterSnapshot (アクション)
- DeleteDBClusterSnapshot (アクション)
- CopyDBClusterSnapshot (アクション)
- ModifyDBClusterSnapshotAttribute (アクション)
- RestoreDBClusterFromSnapshot (アクション)
- RestoreDBClusterToPointInTime (アクション)
- DescribeDBClusterSnapshots (アクション)
- DescribeDBClusterSnapshotAttributes (アクション)

構造:

- DBClusterSnapshot (構造)
- DBClusterSnapshotAttribute (構造)
- DBClusterSnapshotAttributesResult (構造)

CreateDBClusterSnapshot (アクション)

この API の AWS CLI 名は `create-db-cluster-snapshot` です。

DB クラスターのスナップショットを作成します。

リクエスト

- DBClusterIdentifier (CLI では: `--db-cluster-identifier`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  スナップショットを作成する DB クラスターの識別子。このパラメータは大文字と小文字が区別されません。

  制約:

  - 既存の DBCluster の識別子と一致する必要があります。

  例: `my-cluster1`
- DBClusterSnapshotIdentifier (CLI では: `--db-cluster-snapshot-identifier`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスタースナップショットの識別子。このパラメータは小文字で保存されます。

  制約:

  - 1 ~ 63 の文字、数字またはハイフンを使用する必要があります。
  - 1 字目は英字である必要があります。
  - ハイフンを、文字列の最後に使用したり、2 つ続けて使用したりすることはできません。

  例: `my-cluster1-snapshot1`
- Tags (CLI では: `--tags`) - Tag オブジェクトの配列。

  DB クラスタースナップショットに割り当てられるタグ。

レスポンス

Amazon Neptune DB クラスタースナップショットの詳細が含まれています。

このデータ型は、the section called “DescribeDBClusterSnapshots” アクションのレスポンス要素として使用されます。

- AllocatedStorage - タイプ `integer` の整数 (符号付き 32 ビット整数)。

  割り当てられているストレージのサイズ (ギビバイト (GiB)) を指定します。

- AvailabilityZones — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラススナップショットのインスタンスを復元できる EC2 アベイラビリティーゾーン (AZ) のリストを入力します。

- ClusterCreateTime — TStamp、タイプ `timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点) 。

  DB クラスターが作成された時刻を協定世界時 (UTC) で指定します。

- DBClusterIdentifier — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB クラススナップショットの作成元の DB クラスターの DB クラスター識別子を指定します。

- DBClusterSnapshotArn — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラススナップショットの Amazon リソースネーム (ARN)。

- DBClusterSnapshotIdentifier — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラススナップショットの識別子を指定します。既存のスナップショットの識別子と一致する必要があります。

  DB クラスターの復元に `DBClusterSnapshotIdentifier` を使用した場合は、DB クラスターの以後の更新でも同じ `DBClusterSnapshotIdentifier` を指定する必要があります。更新でこのプロパティを指定すると、DB クラスターはスナップショットから再度復元されず、データベース内のデータは変更されません。

  ただし、`DBClusterSnapshotIdentifier` を指定しない場合は、空の DB クラスターが作成され、元の DB クラスターは削除されます。以前のスナップショットの復元プロパティとは異なるプロパティを指定すると、DB クラスターは `DBClusterSnapshotIdentifier` で指定したスナップショットから復元され、元の DB クラスターは削除されます。

- Engine — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  データベースエンジンの名前を指定します。

- EngineVersion — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

この DB クラススナップショットに使用されるデータベースエンジンのバージョンを入力します。

- IAMDatabaseAuthenticationEnabled — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  Amazon Identity and Access Management (IAM) アカウントのデータベースアカウントへのマッピングが有効な場合は true、それ以外の場合は false です。

- KmsKeyId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  `StorageEncrypted` が true の場合、暗号化された DB クラススナップショットの Amazon KMS キー識別子。

- LicenseModel — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB クラススナップショットのライセンスモデル情報を入力します。

- PercentProgress - タイプ `integer` の整数 (符号付き 32 ビット整数)。

  転送された推定データの割合を指定します。

- Port - タイプ `integer` の整数 (符号付き 32 ビット整数)。

  スナップショット時に DB クラスターが待機していたポートを指定します。

- SnapshotCreateTime — TStamp、タイプ `timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点) 。

  スナップショットが取られた時刻を協定世界時 (UTC) で入力します。

- SnapshotType — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラススナップショットのタイプを入力します。

- SourceDBClusterSnapshotArn — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラススナップショットがソース DB クラススナップショットからコピーされた場合は、ソース DB クラススナップショットの Amazon リソースネーム (ARN)、それ以外の場合は null 値。

- Status — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB クラススナップショットのステータスを指定します。

- StorageEncrypted — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  DB クラススナップショットが暗号化されているかどうかを指定します。

- StorageType — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラススナップショットに関連付けられているストレージタイプ。
- VpcId — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラススナップショットに関連付けられている VPC ID を入力します。

エラー

- DBClusterSnapshotAlreadyExistsFault
- InvalidDBClusterStateFault
- DBClusterNotFoundFault
- SnapshotQuotaExceededFault
- InvalidDBClusterSnapshotStateFault

DeleteDBClusterSnapshot (アクション)

この API の AWS CLI 名は `delete-db-cluster-snapshot` です。

DB クラススナップショットを削除します。スナップショットがコピーされている場合、コピー操作は終了します。

> **Note**
>
> DB クラススナップショットを削除するには、`available` 状態にする必要があります。

リクエスト

- DBClusterSnapshotIdentifier (CLI では: `--db-cluster-snapshot-identifier`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  削除する DB クラススナップショットの識別子。

  制約: `available` の状態にある既存の DB クラススナップショットの名前でなければなりません。

レスポンス

Amazon Neptune DB クラススナップショットの詳細が含まれています。

このデータ型は、the section called “DescribeDBClusterSnapshots” アクションのレスポンス要素として使用されます。

- AllocatedStorage - タイプ `integer` の整数 (符号付き 32 ビット整数)。

  割り当てられているストレージのサイズ (ギビバイト (GiB)) を指定します。
- AvailabilityZones — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラススナップショットのインスタンスを復元できる EC2 アベイラビリティーゾーン (AZ) のリストを入力します。
- ClusterCreateTime — TStamp、タイプ `timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点) 。

  DB クラスターが作成された時刻を協定世界時 (UTC) で指定します。
- DBClusterIdentifier — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB クラススナップショットの作成元の DB クラスターの DB クラスター識別子を指定します。
- DBClusterSnapshotArn — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラススナップショットの Amazon リソースネーム (ARN)。
- DBClusterSnapshotIdentifier — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラススナップショットの識別子を指定します。既存のスナップショットの識別子と一致する必要があります。

  DB クラスターの復元に `DBClusterSnapshotIdentifier` を使用した場合は、DB クラスターの以後の更新でも同じ `DBClusterSnapshotIdentifier` を指定する必要があります。更新でこのプロパティを指定すると、DB クラスターはスナップショットから再度復元されず、データベース内のデータは変更されません。

  ただし、`DBClusterSnapshotIdentifier` を指定しない場合は、空の DB クラスターが作成され、元の DB クラスターは削除されます。以前のスナップショットの復元プロパティとは異なるプロパティを指定すると、DB クラスターは `DBClusterSnapshotIdentifier` で指定したスナップショットから復元され、元の DB クラスターは削除されます。
- Engine — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

データベースエンジンの名前を指定します。

- EngineVersion — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB クラススナップショットに使用されるデータベースエンジンのバージョンを入力します。

- IAMDatabaseAuthenticationEnabled — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  Amazon Identity and Access Management (IAM) アカウントのデータベースアカウントへのマッピングが有効な場合は true、それ以外の場合は false です。

- KmsKeyId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  `StorageEncrypted` が true の場合、暗号化された DB クラススナップショットの Amazon KMS キー識別子。

- LicenseModel — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB クラススナップショットのライセンスモデル情報を入力します。

- PercentProgress - タイプ `integer` の整数 (符号付き 32 ビット整数)。

  転送された推定データの割合を指定します。

- Port - タイプ `integer` の整数 (符号付き 32 ビット整数)。

  スナップショット時に DB クラスターが待機していたポートを指定します。

- SnapshotCreateTime — TStamp、タイプ `timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点) 。

  スナップショットが取られた時刻を協定世界時 (UTC) で入力します。

- SnapshotType — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラススナップショットのタイプを入力します。

- SourceDBClusterSnapshotArn — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラススナップショットがソース DB クラススナップショットからコピーされた場合は、ソース DB クラススナップショットの Amazon リソースネーム (ARN)、それ以外の場合は null 値。

- Status — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB クラススナップショットのステータスを指定します。

- StorageEncrypted — ブール値、タイプ: boolean (ブール値 (真または偽))。

  DB クラススナップショットが暗号化されているかどうかを指定します。
- StorageType — タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラススナップショットに関連付けられているストレージタイプ。
- VpcId — タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラススナップショットに関連付けられている VPC ID を入力します。

エラー

- InvalidDBClusterSnapshotStateFault
- DBClusterSnapshotNotFoundFault

CopyDBClusterSnapshot (アクション)

この API の AWS CLI 名は copy-db-cluster-snapshot です。

DB クラスターのスナップショットをコピーします。

手動の共有 DB クラススナップショットから DB クラススナップショットをコピーするには、SourceDBClusterSnapshotIdentifier が、共有 DB クラススナップショットの Amazon リソースネーム (ARN) である必要があります。

リクエスト

- CopyTags (CLI では: --copy-tags) — BooleanOptional、タイプ: boolean (ブール値 (真または偽))。

  ソース DB クラススナップショットからターゲット DB クラススナップショットにすべてのタグをコピーする場合は true、それ以外の場合は false です。デフォルトは [False] (偽) です。
- KmsKeyId (CLI では: --kms-key-id) — タイプ: string (UTF-8 でエンコードされた文字列) の文字列。

  暗号化された DB クラススナップショットの Amazon Amazon KMS キー ID。KMS キー ID は、KMS 暗号化キーの Amazon リソースネーム (ARN)、KMS キー識別子、または KMS キーエイリアスです。

Amazon アカウントから暗号化された DB クラスタースナップショットをコピーする場合、KmsKeyId の値を指定して新しい KMS 暗号化キーでコピーを暗号化できます。KmsKeyId の値を指定しないと、DB クラスタースナップショットのコピーはコピー元の DB クラスタースナップショットと同じ KMS キーで暗号化されます。

別の Amazon アカウントと共有されている暗号化された DB クラスタースナップショットをコピーする場合は、KmsKeyId の値を指定する必要があります。

KMS キーは、それらが作成された Amazon リージョンに固有のものであるため、ある Amazon リージョンの暗号化キーを別の Amazon リージョンで使用することはできません。

コピーするときに、暗号化されていない DB クラスタースナップショットを暗号化することはできません。暗号化されていない DB クラスタースナップショットをコピーして KmsKeyId パラメータに値を指定しようとすると、エラーが返ります。

- PreSignedUrl (CLI では: `--pre-signed-url`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  現在サポートされていません。

- SourceDBClusterSnapshotIdentifier (CLI では: `--source-db-cluster-snapshot-identifier`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  コピーする DB クラスタースナップショットの識別子。このパラメータは大文字と小文字が区別されません。

  制約:

  - [available (使用可能)] 状態の有効なシステムスナップショットを指定する必要があります。
  - 有効な DB スナップショットの識別子を指定します。

  例: `my-cluster-snapshot1`

- Tags (CLI では: `--tags`) - Tag オブジェクトの配列。

  新しい DB クラスタースナップショットのコピーに割り当てるタグ。

- TargetDBClusterSnapshotIdentifier (CLI では: `--target-db-cluster-snapshot-identifier`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  ソース DB クラスタースナップショットから作成する新しい DB クラスタースナップショットの識別子。このパラメータは大文字と小文字が区別されません。

制約:

- 1 ~ 63 の文字、数字またはハイフンを使用する必要があります。
- 1 字目は英字である必要があります。
- ハイフンを、文字列の最後に使用したり、2 つ続けて使用したりすることはできません。

例: `my-cluster-snapshot2`

レスポンス

Amazon Neptune DB クラススナップショットの詳細が含まれています。

このデータ型は、the section called “DescribeDBClusterSnapshots” アクションのレスポンス要素として使用されます。

- AllocatedStorage - タイプ `integer` の整数 (符号付き 32 ビット整数)。

  割り当てられているストレージのサイズ (ギビバイト (GiB)) を指定します。
- AvailabilityZones — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラススナップショットのインスタンスを復元できる EC2 アベイラビリティーゾーン (AZ) のリストを入力します。
- ClusterCreateTime — TStamp、タイプ `timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点) 。

  DB クラスターが作成された時刻を協定世界時 (UTC) で指定します。
- DBClusterIdentifier — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB クラススナップショットの作成元の DB クラスターの DB クラスター識別子を指定します。
- DBClusterSnapshotArn — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラススナップショットの Amazon リソースネーム (ARN)。
- DBClusterSnapshotIdentifier — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラススナップショットの識別子を指定します。既存のスナップショットの識別子と一致する必要があります。

DB クラスターの復元に DBClusterSnapshotIdentifier を使用した場合は、DB クラスターの以後の更新でも同じ DBClusterSnapshotIdentifier を指定する必要があります。更新でこのプロパティを指定すると、DB クラスターはスナップショットから再度復元されず、データベース内のデータは変更されません。

ただし、DBClusterSnapshotIdentifier を指定しない場合は、空の DB クラスターが作成され、元の DB クラスターは削除されます。以前のスナップショットの復元プロパティとは異なるプロパティを指定すると、DB クラスターは DBClusterSnapshotIdentifier で指定したスナップショットから復元され、元の DB クラスターは削除されます。

- Engine — タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  データベースエンジンの名前を指定します。

- EngineVersion — タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  この DB クラスタースナップショットに使用されるデータベースエンジンのバージョンを入力します。

- IAMDatabaseAuthenticationEnabled — ブール値、タイプ: boolean (ブール値 (真または偽))。

  Amazon Identity and Access Management (IAM) アカウントのデータベースアカウントへのマッピングが有効な場合は true、それ以外の場合は false です。

- KmsKeyId — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  StorageEncrypted が true の場合、暗号化された DB クラスタースナップショットの Amazon KMS キー識別子。

- LicenseModel — タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  この DB クラスタースナップショットのライセンスモデル情報を入力します。

- PercentProgress - タイプ integer の整数 (符号付き 32 ビット整数)。

  転送された推定データの割合を指定します。

- Port - タイプ integer の整数 (符号付き 32 ビット整数)。

  スナップショット時に DB クラスターが待機していたポートを指定します。

- SnapshotCreateTime — TStamp、タイプ timestamp (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点) 。

  スナップショットが取られた時刻を協定世界時 (UTC) で入力します。

- SnapshotType — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラススナップショットのタイプを入力します。
- SourceDBClusterSnapshotArn — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラススナップショットがソース DB クラススナップショットからコピーされた場合は、ソース DB クラススナップショットの Amazon リソースネーム (ARN)、それ以外の場合は null 値。
- Status — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この DB クラススナップショットのステータスを指定します。
- StorageEncrypted — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  DB クラススナップショットが暗号化されているかどうかを指定します。
- StorageType — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラススナップショットに関連付けられているストレージタイプ。
- VpcId — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラススナップショットに関連付けられている VPC ID を入力します。

エラー

- DBClusterSnapshotAlreadyExistsFault
- DBClusterSnapshotNotFoundFault
- InvalidDBClusterStateFault
- InvalidDBClusterSnapshotStateFault
- SnapshotQuotaExceededFault
- KMSKeyNotAccessibleFault

ModifyDBClusterSnapshotAttribute (アクション)

この API の AWS CLI 名は `modify-db-cluster-snapshot-attribute` です。

属性および値を、手動 DB クラススナップショットに追加するか、ここから属性および値を削除します。

手動 DB クラスタースナップショットを他の Amazon アカウントと共有するには、`AttributeName` として `restore` を指定し、`ValuesToAdd` パラメータを使用して手動 DB クラスタースナップショットを復元することを承認された Amazon アカウントの ID のリストを追加します。手動の DB クラスタースナップショットをパブリックにするには、値 `all` を使用します。これは、すべての Amazon アカウントでコピーまたは復元できることを意味します。ただし、すべての Amazon アカウントには利用させたくないプライベート情報を含む手動 DB クラスタースナップショットについては、値 `all` を追加しないように注意してください。手動の DB クラスタースナップショットが暗号化されている場合は、`ValuesToAdd` パラメータに承認された Amazon アカウント ID のリストを指定することによってのみ共有できます。この場合、そのパラメータの値として `all` を使用することはできません。

どの Amazon アカウントが手動 DB クラスタースナップショットをコピーまたは復元するアクセス権を持っているか、または手動 DB クラスタースナップショットがパブリックかプライベートかを確認するには、the section called “DescribeDBClusterSnapshotAttributes” API アクションを使用します。

リクエスト

- AttributeName (CLI では: `--attribute-name`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  変更する DB クラスタースナップショット属性の名前。

  手動の DB クラスタースナップショットをコピーまたは復元するための他の Amazon アカウントの認可を管理するには、この値を `restore` に設定します。
- DBClusterSnapshotIdentifier (CLI では: `--db-cluster-snapshot-identifier`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  属性を変更する DB クラスタースナップショットの識別子。
- ValuesToAdd (CLI では: `--values-to-add`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  `AttributeName` で指定された属性に追加する DB クラスタースナップショット属性のリスト

  手動の DB クラスタースナップショットをコピーまたは復元することを他の Amazon アカウントに許可するには、このリストを 1 つ以上の Amazon アカウント ID を含めるか、手動の DB クラスタースナップショットを任意の Amazon アカウントで復元できるように `all` を設定します。ただし、すべての Amazon アカウントには利用させたくないプライベート情報を含む手動 DB クラスタースナップショットについては、値 `all` を追加しないように注意してください。

- ValuesToRemove (CLI では: `--values-to-remove`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  `AttributeName` で指定された属性から削除する DB クラスタースナップショット属性のリスト。

  手動の DB クラスタースナップショットをコピーまたは復元するための他の Amazon アカウントに対する権限を削除するには、このリストに 1 つ以上の Amazon アカウント ID を含めるか、`all` でコピーまたは復元する Amazon アカウントに対する権限を削除します DB クラスタースナップショット `all` を指定しても、アカウント ID が `restore` 属性に明示的に追加されている Amazon アカウントでも、手動の DB クラスタースナップショットをコピーまたは復元できます。

レスポンス

the section called “DescribeDBClusterSnapshotAttributes” API アクションへの呼び出しが成功した結果が含まれています。

手動 DB クラスタースナップショット属性は、他の Amazon アカウントが手動 DB クラスタースナップショットをコピーまたは復元することを承認するために使用されます。詳細については、the section called “ModifyDBClusterSnapshotAttribute” API アクションを参照してください。

- DBClusterSnapshotAttributes – DBClusterSnapshotAttribute オブジェクトの配列。

  手動 DB クラスタースナップショットの属性と値のリスト。

- DBClusterSnapshotIdentifier — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  属性が適用される手動 DB クラスタースナップショットの識別子。

エラー

- DBClusterSnapshotNotFoundFault
- InvalidDBClusterSnapshotStateFault
- SharedSnapshotQuotaExceededFault

RestoreDBClusterFromSnapshot (アクション)

この API の AWS CLI 名は `restore-db-cluster-from-snapshot` です。

DB スナップショットまたは DB クラスタースナップショットから新しい DB クラスターを作成します。

DB スナップショットが指定されている場合、ターゲット DB クラスターは、デフォルト構成とデフォルトセキュリティグループを使用してソース DB スナップショットから作成されます。

DB クラスタースナップショットを指定した場合、新しい DB クラスターが既定のセキュリティグループを使用して作成されることを除いて、ターゲット DB クラスターは元のソース DB クラスターと同じ構成でソース DB クラスターの復元ポイントから作成されます。

リクエスト

- AvailabilityZones (CLI では: `--availability-zones`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターのインスタンスを作成できる EC2 アベイラビリティーゾーンのリストを入力します。

- CopyTagsToSnapshot (CLI では: `--copy-tags-to-snapshot`) — BooleanOptional、タイプ: `boolean` (ブール値 (真または偽))。

  `true` に設定されている場合、タグは、作成された復元 DB クラスターのスナップショットにコピーされます。

- DatabaseName (CLI では: `--database-name`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  サポート外。

- DBClusterIdentifier (CLI では: `--db-cluster-identifier`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB スナップショットまたは DB クラスタースナップショットから作成する DB クラスターの名前。このパラメータでは大文字と小文字は区別されません。

  制約:

  - 1 ~ 63 の文字、数字またはハイフンを使用する必要があります。
  - 1 字目は英字である必要があります。
  - ハイフンを、文字列の最後に使用したり、2 つ続けて使用したりすることはできません

  例: `my-snapshot-id`

- DBClusterParameterGroupName (CLI では: `--db-cluster-parameter-group-name`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  新しい DB クラスターに関連付ける DB クラスターパラメータグループの名前。

  制約:

  - 指定した場合、既存の DBClusterParameterGroup の名前と一致する必要があります。
- DBSubnetGroupName (CLI では: `--db-subnet-group-name`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  新しい DB クラスターに使用する DB サブネットグループの名前。

  制約: 指定した場合、既存の DBSubnetGroup の名前と一致する必要があります。

  例: `mySubnetgroup`
- DeletionProtection (CLI では: `--deletion-protection`) — BooleanOptional、タイプ: `boolean` (ブール値 (真または偽))。

  DB クラスターで削除保護が有効になっているかどうかを示す値。削除保護が有効になっている場合、データベースを削除することはできません。デフォルトでは、削除保護は無効です。
- EnableCloudwatchLogsExports (CLI では: `--enable-cloudwatch-logs-exports`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  復元された DB クラスターが Amazon CloudWatch Logs にエクスポートするログのリスト。
- EnableIAMDatabaseAuthentication (CLI では: `--enable-iam-database-authentication`) — BooleanOptional、タイプ: `boolean` (ブール値 (真または偽))。

  Amazon Identity and Access Management (IAM) アカウントのデータベースアカウントへのマッピングが有効な場合は true、それ以外の場合は false です。

  デフォルト: `false`
- Engine (CLI では: `--engine`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  新しい DB クラスターに使用するデータベースエンジン。

  デフォルト: ソースオブジェクトと同じ

  制約: ソースのエンジンに対応する必要があります。

- EngineVersion (CLI では: `--engine-version`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  新しい DB クラスターに使用するデータベースエンジンのバージョン。
- KmsKeyId (CLI では: `--kms-key-id`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  DB スナップショットまたは DB クラススナップショットから暗号化された DB クラスターを復元するときに使用する Amazon KMS キー識別子。

  KMS キー識別子は、KMS 暗号化キーの Amazon リソースネーム (ARN) です。新しい DB クラスターの暗号化に使用する KMS 暗号化キーを所有する Amazon アカウントと同じアカウントを使用して DB クラスターを復元する場合、KMS 暗号化キーの ARN の代わりに KMS キーのエイリアスを使用できます。

  `KmsKeyId` パラメータの値を指定しない場合は、以下のようになります。
  - `SnapshotIdentifier` の DB スナップショットまたは DB クラススナップショットが暗号化されている場合、復元された DB クラスターは、DB スナップショットまたは DB クラススナップショットの暗号化に使用された KMS キーを使用して暗号化されます。
  - `SnapshotIdentifier` の DB スナップショットまたは DB クラススナップショットが暗号化されていない場合、復元された DB クラスターは暗号化されません。
- Port (CLI では: `--port`) — IntegerOptional、タイプ: `integer` (符号付き 32 ビット整数)。

  新しい DB クラスターが接続を受け付けるポート番号。

  制約: 値は `1150-65535` である必要があります。

  デフォルト : 元の DB クラスターと同じポート。
- ServerlessV2ScalingConfiguration (CLI では: `--serverless-v2-scaling-configuration`) — ServerlessV2ScalingConfiguration オブジェクト。

  Neptune サーバーレス DB クラスターのスケーリング構成を含みます。

  詳細については、「Amazon Neptune ユーザーガイド」の 「Amazon Neptune サーバーレスの使用」を参照してください。
- SnapshotIdentifier (CLI では: `--snapshot-identifier`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  復元元の DB スナップショットまたは DB クラススナップショットの識別子。

DB クラススナップショットを指定するには、名前または Amazon リソースネーム (ARN) のいずれかを使用できます。ただし、DB スナップショットを指定するには ARN のみを使用できます。

制約:

- 既存のスナップショットの識別子と一致する必要があります。

- StorageType (CLI では: `--storage-type`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  DB クラスターに関連付けられるストレージタイプを指定します。

  有効な値: `standard`、`iopt1`

  デフォルト: `standard`

- Tags (CLI では: `--tags`) - Tag オブジェクトの配列。

  復元された DB クラスターに割り当てられるタグ。

- VpcSecurityGroupIds (CLI では: `--vpc-security-group-ids`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  新しい DB クラスターが属する VPC セキュリティグループのリスト。

レスポンス

Amazon Neptune DB クラスターの詳細が含まれています。

このデータ型は、the section called “DescribeDBClusters” でレスポンス要素として使用されます。

- AllocatedStorage - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  `AllocatedStorage` は常に 1 を返します。これは、Neptune DB クラスターのストレージサイズが固定されていないため、必要に応じて自動的に調整されるためです。

- AssociatedRoles – DBClusterRole オブジェクトの配列。

  DB クラスターに関連付けられている Amazon Identity and Access Management (IAM) ロールのリストを入力します。DB クラスターに関連付けられている IAM ロールは、ユーザーに代わって他の Amazon のサービスにアクセスするための DB クラスターへのアクセス許可を付与します。

- AutomaticRestartTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  DB クラスターが自動的に再起動される時刻。
- AvailabilityZones — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターのインスタンスを作成できる EC2 アベイラビリティーゾーン (AZ) のリストを入力します。
- BacktrackConsumedChangeRecords — タイプ `long` の LongOptional (符号付き 64 ビット整数)。

  Neptune ではサポートされていません。
- BacktrackWindow — タイプ `long` の LongOptional (符号付き 64 ビット整数)。

  Neptune ではサポートされていません。
- BackupRetentionPeriod - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  自動 DB スナップショットが保持される日数を指定します。
- Capacity - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  Neptune ではサポートされていません。
- CloneGroupId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターが関連付けられているクローングループを識別します。
- ClusterCreateTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  DB クラスターが作成された時刻を協定世界時 (UTC) で指定します。
- CopyTagsToSnapshot — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽) の値)。

  `true` に設定されている場合、タグは、作成された DB クラスターのいずれかのスナップショットにコピーされます。
- CrossAccountClone — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽) の値)。

  `true`に設定されている場合、DB クラスターを複数のアカウント間でクローンできます。
- DatabaseName — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターの作成時に指定された場合は、作成時に提供されたこの DB クラスターの初期データベースの名前が入ります。DB クラスターの存続中はこれと同じ名前が返されます。

- DBClusterArn — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターの Amazon リソースネーム (ARN) を返します。

- DBClusterIdentifier — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  ユーザーが指定した DB クラスター識別子が含まれています。この識別子は、DB クラスターを識別する一意のキーです。

- DBClusterMembers – DBClusterMember オブジェクトの配列。

  DB クラスターを構成するインスタンスのリストを入力します。

- DBClusterParameterGroup — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターに使用する DB クラスターパラメータグループの名前を指定します。

- DbClusterResourceId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターの Amazon リージョン固有のイミュータブルな識別子。この識別子は、DB クラスターの Amazon KMS キーにアクセスするたびに Amazon CloudTrail ログエントリに記録されます。

- DBSubnetGroup — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  サブネットグループ内の名前、説明、サブネットなど、DB クラスターに関連付けられているサブネットグループに関する情報を指定します。

- DeletionProtection — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽) の値)。

  DB クラスターで削除保護が有効になっているかどうかを示します。削除保護が有効になっている場合、データベースを削除することはできません。

- EarliestBacktrackTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  Neptune ではサポートされていません。

- EarliestRestorableTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  特定時点への復元によりデータベースを復元できる最も早い時刻を指定します。

- EnabledCloudwatchLogsExports — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

この DB クラスターが CloudWatch Logs にエクスポートするように設定されているログタイプのリスト。有効なログタイプは、audit (監査ログを CloudWatch に公開する場合) と slowquery (スロークエリログを CloudWatch に公開する場合) です。「Amazon CloudWatch Logs への Neptune ログの発行」を参照してください。

- Endpoint — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  DB クラスターのプライマリインスタンスの接続エンドポイントを指定します。
- Engine — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  この DB クラスターに使用されるデータベースエンジンの名前を入力します。
- EngineVersion — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  データベースエンジンのバージョンを示します。
- GlobalClusterIdentifier — GlobalClusterIdentifier、Type: string (UTF-8 でエンコードされた文字列)、1 ~ 255、正規表現 [A-Za-z][0-9A-Za-z-:._]* に一致。

  ユーザーが指定したグローバルクラスター識別子を含みます。この識別子は、グローバルクラスターを識別する一意のキーです。
- HostedZoneId — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  ホストゾーンを作成するときに Amazon Route 53 が割り当てる ID を指定します。
- IAMDatabaseAuthenticationEnabled — ブール値、タイプ: boolean (ブール値 (真または偽))。

  Amazon Identity and Access Management (IAM) アカウントのデータベースアカウントへのマッピングが有効な場合は true、それ以外の場合は false です。
- IOOptimizedNextAllowedModificationTime — TStamp、タイプ:timestamp (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  次回は、iopt1 ストレージタイプを使用するように DB クラスターを変更できます。
- KmsKeyId — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  StorageEncrypted が true の場合、暗号化された DB クラスターの Amazon KMS キー識別子。
- LatestRestorableTime — TStamp、タイプ:timestamp (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  特定時点への復元によりデータベースを復元できる最新の日時を指定します。
- MultiAZ — ブール値、タイプ: boolean (ブール値 (真または偽))。

DB クラスターが複数のアベイラビリティーゾーンにインスタンスを持つかどうかを指定します。

- PendingModifiedValues – ClusterPendingModifiedValues オブジェクト。

  このデータ型は、ModifyDBCluster 操作でレスポンス要素として使用され、次のメンテナンスウィンドウに適用される変更を含みます。

- PercentProgress — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  オペレーションの進行状況をパーセンテージで指定します。

- Port - タイプ integer の IntegerOptional (符号付き 32 ビット整数)。

  データベースエンジンがリッスンするポートを指定します。

- PreferredBackupWindow — タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  BackupRetentionPeriod に応じた、自動バックアップが有効な場合に自動バックアップが作成される毎日の時間範囲を指定します。

- PreferredMaintenanceWindow — タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  週 1 回のシステムメンテナンスを実行できる時間範囲を世界標準時 (UTC) で指定します。

- ReaderEndpoint — タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターのリーダーエンドポイント。DB クラスターの読み込みエンドポイントは、DB クラスター内で使用できるリードレプリカ間で接続を負荷分散します。クライアントが読み取りエンドポイントへの新規接続をリクエストすると、Neptune によって接続リクエストが DB クラスターのリードレプリカ間で配分されます。この機能は、DB クラスターの複数のリードレプリカ間の読み取りワークロードを分散させる役に立ちます。

  フェイルオーバーが発生し、接続しているリードレプリカがプライマリインスタンスに昇格すると、接続は削除されます。読み取りワークロードをクラスター内の他のリードレプリカに送信し続けるために、読み込みエンドポイントに再接続することができます。

- ReadReplicaIdentifiers — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  この DB クラスターに関連付けられているリードレプリカの 1 つ以上の識別子を含みます。

- ReplicationSourceIdentifier — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  Neptune ではサポートされていません。

- ReplicationType — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

Neptune ではサポートされていません。

- ServerlessV2ScalingConfiguration – ServerlessV2ScalingConfigurationInfo オブジェクト。

  Neptune サーバーレス DB クラスターのスケーリング設定を表示します。

  詳細については、「Amazon Neptune ユーザーガイド」の 「Amazon Neptune サーバーレスの使用」を参照してください。
- Status — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターの現在の状態を指定します。
- StorageEncrypted - タイプ `boolean` のブール値 (ブール値 (真または偽))。

  DB クラスターが暗号化されているかどうかを指定します。
- StorageType — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターで使用するストレージタイプ。

  有効な値:

  - **`standard`** — (デフォルト) I/O 使用率が低いか中程度のアプリケーション向けに、費用対効果の高いデータベースストレージを提供します。
  - **`iopt1`** — 料金が予測可能で、低い I/O レイテンシーと一貫した I/O スループットを必要とする I/O 集約型のグラフワークロードのニーズを満たすように設計された I/O 最適化ストレージを有効にします。

    Neptune I/O 最適化ストレージは、エンジンリリース 1.3.0.0 以降でのみ使用可能です。
- VpcSecurityGroups – VpcSecurityGroupMembership オブジェクトの配列。

  DB クラスターが属する VPC セキュリティグループのリストを入力します。

エラー

- DBClusterAlreadyExistsFault
- DBClusterQuotaExceededFault
- StorageQuotaExceededFault
- DBSubnetGroupNotFoundFault
- DBSnapshotNotFoundFault

- DBClusterSnapshotNotFoundFault
- InsufficientDBClusterCapacityFault
- InsufficientStorageClusterCapacityFault
- InvalidDBSnapshotStateFault
- InvalidDBClusterSnapshotStateFault
- StorageQuotaExceededFault
- InvalidVPCNetworkStateFault
- InvalidRestoreFault
- DBSubnetGroupNotFoundFault
- InvalidSubnet
- OptionGroupNotFoundFault
- KMSKeyNotAccessibleFault
- DBClusterParameterGroupNotFoundFault

RestoreDBClusterToPointInTime (アクション)

この API の AWS CLI 名は `restore-db-cluster-to-point-in-time` です。

DB クラスターを任意の時点に復元します。ユーザーは `LatestRestorableTime` より前の任意の時点に最大 `BackupRetentionPeriod` 日間復元できます。DB クラスタースナップショットを指定した場合、新しい DB クラスターがデフォルトの DB セキュリティグループを使用して作成されることを除いて、ターゲット DB クラスターは元のソース DB クラスターと同じ構成でソース DB クラスターの復元ポイントから作成されます。

> **Note**
>
> このアクションでは DB クラスターのみを復元し、その DB クラスターの DB インスタンスは復元しません。`DBClusterIdentifier` に復元した DB クラスターの識別子を指定して、復元した DB クラスターの DB インスタンスを作成するには、the section called “CreateDBInstance” アクションを呼び出す必要があります。`RestoreDBClusterToPointInTime` アクションが完了し、DB クラスターのステータスが使用可能になった後でのみ、DB インスタンスを作成できます。

リクエスト

- DBClusterIdentifier (CLI では: `--db-cluster-identifier`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  作成される新しい DB クラスターの名前。

  制約:

  - 1 ~ 63 の文字、数字またはハイフンを使用する必要があります。
  - 1 字目は英字である必要があります。
  - ハイフンを、文字列の最後に使用したり、2 つ続けて使用したりすることはできません
- DBClusterParameterGroupName (CLI では: `--db-cluster-parameter-group-name`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  新しい DB クラスターに関連付ける DB クラスターパラメータグループの名前。

  制約:

  - 指定した場合、既存の DBClusterParameterGroup の名前と一致する必要があります。
- DBSubnetGroupName (CLI では: `--db-subnet-group-name`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  新しい DB クラスターに使用する DB サブネットグループ名。

  制約: 指定した場合、既存の DBSubnetGroup の名前と一致する必要があります。

  例: `mySubnetgroup`
- DeletionProtection (CLI では: `--deletion-protection`) — BooleanOptional、タイプ: `boolean` (ブール値 (真または偽))。

  DB クラスターで削除保護が有効になっているかどうかを示す値。削除保護が有効になっている場合、データベースを削除することはできません。デフォルトでは、削除保護は無効です。
- EnableCloudwatchLogsExports (CLI では: `--enable-cloudwatch-logs-exports`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  復元された DB クラスターが CloudWatch Logs にエクスポートするログのリスト。
- EnableIAMDatabaseAuthentication (CLI では: `--enable-iam-database-authentication`) — BooleanOptional、タイプ: `boolean` (ブール値 (真または偽))。

  Amazon Identity and Access Management (IAM) アカウントのデータベースアカウントへのマッピングが有効な場合は true、それ以外の場合は false です。

デフォルト: `false`

- KmsKeyId (CLI では: `--kms-key-id`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  DB スナップショットまたは暗号化された DB クラスターから暗号化された DB クラスターを復元するときに使用する Amazon KMS キー識別子。

  KMS キー識別子は、KMS 暗号化キーの Amazon リソースネーム (ARN) です。新しい DB クラスターの暗号化に使用する KMS 暗号化キーを所有する Amazon アカウントと同じアカウントを使用して DB クラスターを復元する場合、KMS 暗号化キーの ARN の代わりに KMS キーのエイリアスを使用できます。

  新しい DB クラスターに復元し、新しい DB クラスターをソース DB クラスターの暗号化に使用された KMS キーとは異なる KMS キーで暗号化することができます。新しい DB クラスターは、`KmsKeyId` パラメータで識別される KMS キーで暗号化されています。

  `KmsKeyId` パラメータの値を指定しない場合は、以下のようになります。

  - DB クラスターが暗号化されている場合、復元された DB クラスターは、ソース DB クラスターの暗号化に使用されたものと同じ KMS キーを使用して暗号化されます。
  - DB クラスターが暗号化されていない場合、復元された DB クラスターは暗号化されません。

  `DBClusterIdentifier` が暗号化されていない DB クラスターを参照している場合、復元リクエストは拒否されます。

- Port (CLI では: `--port`) — IntegerOptional、タイプ: `integer` (符号付き 32 ビット整数)。

  新しい DB クラスターが接続を受け付けるポート番号。

  制約: 値は `1150-65535` である必要があります。

  デフォルト : 元の DB クラスターと同じポート。

- RestoreToTime (CLI では: `--restore-to-time`) — TStamp、タイプ: `timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  DB クラスターを復元する日時。

  有効な値: 協定世界時 (UTC) 形式の時刻でなければなりません。

  制約:

- DB インスタンスの復元可能な最新時刻より前である必要があります。
- `UseLatestRestorableTime` パラメータを指定しない場合は、指定する必要があります。
- `UseLatestRestorableTime` パラメータが true である場合は、指定することはできません。
- `RestoreType` パラメータが `copy-on-write` である場合は、指定することはできません。

例: `2015-03-07T23:45:00Z`

- RestoreType (CLI では: `--restore-type`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  実行する復元のタイプ。次のいずれかの値を指定できます。

  - `full-copy` - 新しい DB クラスターは、ソース DB クラスターのフルコピーとして復元されます。
  - `copy-on-write` - 新しい DB クラスターは、ソース DB クラスターのクローンとして復元されます。

  `RestoreType` 値を指定しない場合、新しい DB クラスターは、ソース DB クラスターのフルコピーとして復元されます。

- ServerlessV2ScalingConfiguration (CLI では: `--serverless-v2-scaling-configuration`) — ServerlessV2ScalingConfiguration オブジェクト。

  Neptune サーバーレス DB クラスターのスケーリング構成を含みます。

  詳細については、「Amazon Neptune ユーザーガイド」の 「Amazon Neptune サーバーレスの使用」を参照してください。

- SourceDBClusterIdentifier (CLI では: `--source-db-cluster-identifier`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  復元元のソース DB クラスターの識別子。

  制約:

  - 既存の DBCluster の識別子と一致する必要があります。

- StorageType (CLI では: `--storage-type`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  DB クラスターに関連付けられるストレージタイプを指定します。

  有効な値: `standard`、`iopt1`

デフォルト: `standard`

- Tags (CLI では: `--tags`) - Tag オブジェクトの配列。

  復元された DB クラスターに適用されるタグ。

- UseLatestRestorableTime (CLI では: `--use-latest-restorable-time`) — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  DB クラスターを最新の復元可能なバックアップ時刻に復元する場合は `true` に、それ以外の場合は `false` に設定される値。

  デフォルト: `false`

  制約: `RestoreToTime` パラメータを指定した場合は、指定することはできません。

- VpcSecurityGroupIds (CLI では: `--vpc-security-group-ids`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  新しい DB クラスターが属する VPC セキュリティグループのリストを入力します。

レスポンス

Amazon Neptune DB クラスターの詳細が含まれています。

このデータ型は、the section called “DescribeDBClusters” でレスポンス要素として使用されます。

- AllocatedStorage - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  `AllocatedStorage` は常に 1 を返します。これは、Neptune DB クラスターのストレージサイズが固定されていないため、必要に応じて自動的に調整されるためです。

- AssociatedRoles – DBClusterRole オブジェクトの配列。

  DB クラスターに関連付けられている Amazon Identity and Access Management (IAM) ロールのリストを入力します。DB クラスターに関連付けられている IAM ロールは、ユーザーに代わって他の Amazon のサービスにアクセスするための DB クラスターへのアクセス許可を付与します。

- AutomaticRestartTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  DB クラスターが自動的に再起動される時刻。

- AvailabilityZones — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

DB クラスターのインスタンスを作成できる EC2 アベイラビリティーゾーン (AZ) のリストを入力します。

- BacktrackConsumedChangeRecords — タイプ `long` の LongOptional (符号付き 64 ビット整数)。

  Neptune ではサポートされていません。

- BacktrackWindow — タイプ `long` の LongOptional (符号付き 64 ビット整数)。

  Neptune ではサポートされていません。

- BackupRetentionPeriod - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  自動 DB スナップショットが保持される日数を指定します。

- Capacity - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  Neptune ではサポートされていません。

- CloneGroupId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターが関連付けられているクローングループを識別します。

- ClusterCreateTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  DB クラスターが作成された時刻を協定世界時 (UTC) で指定します。

- CopyTagsToSnapshot — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽) の値)。

  `true` に設定されている場合、タグは、作成された DB クラスターのいずれかのスナップショットにコピーされます。

- CrossAccountClone — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽) の値)。

  `true`に設定されている場合、DB クラスターを複数のアカウント間でクローンできます。

- DatabaseName — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターの作成時に指定された場合は、作成時に提供されたこの DB クラスターの初期データベースの名前が入ります。DB クラスターの存続中はこれと同じ名前が返されます。

- DBClusterArn — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターの Amazon リソースネーム (ARN) を返します。

- DBClusterIdentifier — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

ユーザーが指定した DB クラスター識別子が含まれています。この識別子は、DB クラスターを識別する一意のキーです。

- DBClusterMembers – DBClusterMember オブジェクトの配列。

  DB クラスターを構成するインスタンスのリストを入力します。

- DBClusterParameterGroup — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターに使用する DB クラスターパラメータグループの名前を指定します。

- DbClusterResourceId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB クラスターの Amazon リージョン固有のイミュータブルな識別子。この識別子は、DB クラスターの Amazon KMS キーにアクセスするたびに Amazon CloudTrail ログエントリに記録されます。

- DBSubnetGroup — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  サブネットグループ内の名前、説明、サブネットなど、DB クラスターに関連付けられているサブネットグループに関する情報を指定します。

- DeletionProtection — BooleanOptional、タイプ:`boolean` (ブール値 (真または偽) の値)。

  DB クラスターで削除保護が有効になっているかどうかを示します。削除保護が有効になっている場合、データベースを削除することはできません。

- EarliestBacktrackTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  Neptune ではサポートされていません。

- EarliestRestorableTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  特定時点への復元によりデータベースを復元できる最も早い時刻を指定します。

- EnabledCloudwatchLogsExports — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターが CloudWatch Logs にエクスポートするように設定されているログタイプのリスト。有効なログタイプは、`audit` (監査ログを CloudWatch に公開する場合) と slowquery (スロークエリログを CloudWatch に公開する場合) です。「Amazon CloudWatch Logs への Neptune ログの発行」を参照してください。

- Endpoint — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

DB クラスターのプライマリインスタンスの接続エンドポイントを指定します。

- Engine — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターに使用されるデータベースエンジンの名前を入力します。

- EngineVersion — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  データベースエンジンのバージョンを示します。

- GlobalClusterIdentifier — GlobalClusterIdentifier、Type: `string` (UTF-8 でエンコードされた文字列)、1 ~ 255、正規表現 `[A-Za-z][0-9A-Za-z-:._]*` に一致。

  ユーザーが指定したグローバルクラスター識別子を含みます。この識別子は、グローバルクラスターを識別する一意のキーです。

- HostedZoneId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  ホストゾーンを作成するときに Amazon Route 53 が割り当てる ID を指定します。

- IAMDatabaseAuthenticationEnabled — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  Amazon Identity and Access Management (IAM) アカウントのデータベースアカウントへのマッピングが有効な場合は true、それ以外の場合は false です。

- IOOptimizedNextAllowedModificationTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  次回は、`iopt1` ストレージタイプを使用するように DB クラスターを変更できます。

- KmsKeyId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  `StorageEncrypted` が true の場合、暗号化された DB クラスターの Amazon KMS キー識別子。

- LatestRestorableTime — TStamp、タイプ:`timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定の時点)。

  特定時点への復元によりデータベースを復元できる最新の日時を指定します。

- MultiAZ — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  DB クラスターが複数のアベイラビリティーゾーンにインスタンスを持つかどうかを指定します。

- PendingModifiedValues – ClusterPendingModifiedValues オブジェクト。

  このデータ型は、`ModifyDBCluster` 操作でレスポンス要素として使用され、次のメンテナンスウィンドウに適用される変更を含みます。

- PercentProgress — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  オペレーションの進行状況をパーセンテージで指定します。

- Port - タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  データベースエンジンがリッスンするポートを指定します。

- PreferredBackupWindow — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  `BackupRetentionPeriod` に応じた、自動バックアップが有効な場合に自動バックアップが作成される毎日の時間範囲を指定します。

- PreferredMaintenanceWindow — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  週 1 回のシステムメンテナンスを実行できる時間範囲を世界標準時 (UTC) で指定します。

- ReaderEndpoint — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターのリーダーエンドポイント。DB クラスターの読み込みエンドポイントは、DB クラスター内で使用できるリードレプリカ間で接続を負荷分散します。クライアントが読み取りエンドポイントへの新規接続をリクエストすると、Neptune によって接続リクエストが DB クラスターのリードレプリカ間で配分されます。この機能は、DB クラスターの複数のリードレプリカ間の読み取りワークロードを分散させる役に立ちます。

  フェイルオーバーが発生し、接続しているリードレプリカがプライマリインスタンスに昇格すると、接続は削除されます。読み取りワークロードをクラスター内の他のリードレプリカに送信し続けるために、読み込みエンドポイントに再接続することができます。

- ReadReplicaIdentifiers — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターに関連付けられているリードレプリカの 1 つ以上の識別子を含みます。

- ReplicationSourceIdentifier — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  Neptune ではサポートされていません。

- ReplicationType — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  Neptune ではサポートされていません。

- ServerlessV2ScalingConfiguration – ServerlessV2ScalingConfigurationInfo オブジェクト。

  Neptune サーバーレス DB クラスターのスケーリング設定を表示します。

詳細については、「Amazon Neptune ユーザーガイド」の 「Amazon Neptune サーバーレスの使用」を参照してください。

- Status — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  この DB クラスターの現在の状態を指定します。

- StorageEncrypted - タイプ `boolean` のブール値 (ブール値 (真または偽))。

  DB クラスターが暗号化されているかどうかを指定します。

- StorageType — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターで使用するストレージタイプ。

  有効な値:

  - **`standard`** — (デフォルト) I/O 使用率が低いか中程度のアプリケーション向けに、費用対効果の高いデータベースストレージを提供します。
  - **`iopt1`** — 料金が予測可能で、低い I/O レイテンシーと一貫した I/O スループットを必要とする I/O 集約型のグラフワークロードのニーズを満たすように設計された I/O 最適化ストレージを有効にします。

    Neptune I/O 最適化ストレージは、エンジンリリース 1.3.0.0 以降でのみ使用可能です。

- VpcSecurityGroups – VpcSecurityGroupMembership オブジェクトの配列。

  DB クラスターが属する VPC セキュリティグループのリストを入力します。

エラー

- DBClusterAlreadyExistsFault
- DBClusterNotFoundFault
- DBClusterQuotaExceededFault
- DBClusterSnapshotNotFoundFault
- DBSubnetGroupNotFoundFault
- InsufficientDBClusterCapacityFault
- InsufficientStorageClusterCapacityFault
- InvalidDBClusterSnapshotStateFault
- InvalidDBClusterStateFault

- InvalidDBSnapshotStateFault
- InvalidRestoreFault
- InvalidSubnet
- InvalidVPCNetworkStateFault
- KMSKeyNotAccessibleFault
- OptionGroupNotFoundFault
- StorageQuotaExceededFault
- DBClusterParameterGroupNotFoundFault

DescribeDBClusterSnapshots (アクション)

この API の AWS CLI 名は `describe-db-cluster-snapshots` です。

DB クラススナップショットに関する情報を返します。この API アクションはページ分割をサポートします。

リクエスト

- DBClusterIdentifier (CLI では: `--db-cluster-identifier`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラススナップショットのリストを取得する DB クラスターの ID。このパラメータは、`DBClusterSnapshotIdentifier` パラメータと組み合わせて使用することはできません。このパラメータは大文字と小文字が区別されません。

  制約:

  - 指定した場合、既存の DBCluster の識別子と一致している必要があります。
- DBClusterSnapshotIdentifier (CLI では: `--db-cluster-snapshot-identifier`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  記述する特定の DB クラススナップショット識別子。このパラメータは、`DBClusterIdentifier` パラメータと組み合わせて使用することはできません。この値は小文字で保存されます。

  制約:

  - 指定した場合、既存の DBClusterSnapshot の識別子と一致している必要があります。

  - この識別子が自動スナップショット用の場合、SnapshotType パラメータも指定する必要があります。
- Filters (CLI では: `--filters`) - フィルター オブジェクトの配列。

  このパラメータは、現在サポートされていません。
- IncludePublic (CLI では: `--include-public`) — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  パブリックであり、任意の Amazon アカウントでコピーまたは復元できる手動の DB クラスタースナップショットを含める場合は true、それ以外の場合は false です。デフォルトは `false` です。デフォルトは [False] (偽) です。

  手動 DB クラスタースナップショットは、the section called “ModifyDBClusterSnapshotAttribute” API アクションを使用してパブリックとして共有できます。
- IncludeShared (CLI では: `--include-shared`) — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  この Amazon アカウントにコピーまたは復元のアクセス許可が与えられている他の Amazon アカウントからの共有手動 DB クラスタースナップショットを含める場合は true、それ以外の場合は false。デフォルトは `false` です。

  the section called “ModifyDBClusterSnapshotAttribute” API アクションを使用すると、Amazon アカウントに手動の DB クラスタースナップショットを別の Amazon アカウントから復元するアクセス許可を付与できます。
- Marker (CLI では: `--marker`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  以前の `DescribeDBClusterSnapshots` リクエストによって提供されたオプションのページ割りトークン。このパラメータを指定した場合、レスポンスには `MaxRecords` で指定された値まで、マーカーを超えるレコードのみが含まれます。
- MaxRecords (CLI では: `--max-records`) — タイプ `integer` の IntegerOptional (符号付き 32 ビット整数)。

  レスポンスに含めるレコードの最大数。指定された `MaxRecords` の値よりも多くのレコードが存在する場合、マーカーと呼ばれるページ割りトークンがレスポンスに含まれるため、残りの結果を取得できます。

  デフォルト: 100

制約: 最小 20、最大 100。

- SnapshotType (CLI では: `--snapshot-type`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  返される DB クラスタースナップショットのタイプ。次のいずれかの値を指定できます。

  - `automated` - Amazon アカウント用に Amazon Neptune によって自動的に取得されたすべての DB クラスタースナップショットを返します。
  - `manual` - Amazon アカウント用に Amazon Neptune によって自動的に取得されたすべての DB クラスタースナップショットを返します。
  - `shared` - Amazon アカウントに共有されているすべての手動 DB クラスタースナップショットを返します。
  - `public` - パブリックとしてマークされているすべての DB クラスタースナップショットを返します。

  `SnapshotType` 値を指定しない場合、自動および手動の両方の DB クラスタースナップショットが返されます。`IncludeShared` パラメータを `true` に設定することで、これらの結果に共有 DB クラスタースナップショットを含めることができます。`IncludePublic` パラメータを `true` に設定することで、これらの結果にパブリック DB クラスタースナップショットを含めることができます。

  `IncludeShared` および `IncludePublic` パラメータは、`manual` または `automated` の `SnapshotType` 値には適用されません。`SnapshotType` が `shared` に設定されている場合、`IncludePublic` パラメータは適用されません。`SnapshotType` が `public` に設定されている場合、`IncludeShared` パラメータは適用されません。

レスポンス

- DBClusterSnapshots – DBClusterSnapshot オブジェクトの配列。

  ユーザーの DB クラスタースナップショットのリストを入力します。

- Marker — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  以前の the section called “DescribeDBClusterSnapshots” リクエストによって提供されたオプションのページ割りトークン。このパラメータを指定した場合、レスポンスには `MaxRecords` で指定された値まで、マーカーを超えるレコードのみが含まれます。

エラー

- DBClusterSnapshotNotFoundFault

DescribeDBClusterSnapshotAttributes (アクション)

この API の AWS CLI 名は `describe-db-cluster-snapshot-attributes` です。

手動の DB クラスタースナップショットの DB クラスタースナップショットの属性名と値のリストを返します。

スナップショットを他の Amazon アカウントと共有する場合、`DescribeDBClusterSnapshotAttributes` は `restore` 属性と手動 DB クラスタースナップショットのコピーまたは復元を許可された Amazon アカウントの ID のリストを返します。`all` が `restore` 属性の値の一覧に含まれている場合、手動 DB クラスタースナップショットは公開されており、すべての Amazon アカウントでコピーまたは復元できます。

手動の DB クラスタースナップショットをコピーまたは復元するため、または手動の DB クラスタースナップショットをパブリックまたはプライベートにするための Amazon アカウントへのアクセスを追加または削除するには、the section called “ModifyDBClusterSnapshotAttribute” API アクションを使用します。

リクエスト

- DBClusterSnapshotIdentifier (CLI では: `--db-cluster-snapshot-identifier`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  属性を記述する DB クラスタースナップショットの識別子。

レスポンス

the section called “DescribeDBClusterSnapshotAttributes” API アクションへの呼び出しが成功した結果が含まれています。

手動 DB クラスタースナップショット属性は、他の Amazon アカウントが手動 DB クラスタースナップショットをコピーまたは復元することを承認するために使用されます。詳細については、the section called “ModifyDBClusterSnapshotAttribute” API アクションを参照してください。

- DBClusterSnapshotAttributes – DBClusterSnapshotAttribute オブジェクトの配列。

手動 DB クラススナップショットの属性と値のリスト。

- DBClusterSnapshotIdentifier — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  属性が適用される手動 DB クラススナップショットの識別子。

エラー

- DBClusterSnapshotNotFoundFault

構造:

DBClusterSnapshot (構造)

Amazon Neptune DB クラススナップショットの詳細が含まれています。

このデータ型は、the section called “DescribeDBClusterSnapshots” アクションのレスポンス要素として使用されます。

フィールド

- AllocatedStorage — 整数、タイプ: `integer` (符号付き 32 ビット整数)。

  割り当てられているストレージのサイズ (ギビバイト (GiB)) を指定します。
- AvailabilityZones — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  DB クラススナップショットのインスタンスを復元できる EC2 アベイラビリティーゾーン (AZ) のリストを入力します。
- ClusterCreateTime — TStamp、タイプ: `timestamp` (一般的には 1970 年 1 月 1 日の深夜 0 時からのオフセットとして定義される特定の時点)。

  DB クラスターが作成された時刻を協定世界時 (UTC) で指定します。
- DBClusterIdentifier — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  この DB クラススナップショットの作成元の DB クラスターの DB クラスター識別子を指定します。
- DBClusterSnapshotArn — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

DB クラススナップショットの Amazon リソースネーム (ARN)。

- DBClusterSnapshotIdentifier — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  DB クラススナップショットの識別子を指定します。既存のスナップショットの識別子と一致する必要があります。

  DB クラスターの復元に `DBClusterSnapshotIdentifier` を使用した場合は、DB クラスターの以後の更新でも同じ `DBClusterSnapshotIdentifier` を指定する必要があります。更新でこのプロパティを指定すると、DB クラスターはスナップショットから再度復元されず、データベース内のデータは変更されません。

  ただし、`DBClusterSnapshotIdentifier` を指定しない場合は、空の DB クラスターが作成され、元の DB クラスターは削除されます。以前のスナップショットの復元プロパティとは異なるプロパティを指定すると、DB クラスターは `DBClusterSnapshotIdentifier` で指定したスナップショットから復元され、元の DB クラスターは削除されます。
- Engine — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  データベースエンジンの名前を指定します。
- EngineVersion — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  この DB クラススナップショットに使用されるデータベースエンジンのバージョンを入力します。
- IAMDatabaseAuthenticationEnabled — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  Amazon Identity and Access Management (IAM) アカウントのデータベースアカウントへのマッピングが有効な場合は true、それ以外の場合は false です。
- KmsKeyId — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  `StorageEncrypted` が true の場合、暗号化された DB クラススナップショットの Amazon KMS キー識別子。
- LicenseModel — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  この DB クラススナップショットのライセンスモデル情報を入力します。
- PercentProgress — 整数、タイプ: `integer` (符号付き 32 ビット整数)。

  転送された推定データの割合を指定します。

- Port — 整数、タイプ: `integer` (符号付き 32 ビット整数)。

  スナップショット時に DB クラスターが待機していたポートを指定します。

- SnapshotCreateTime — TStamp、タイプ: `timestamp` (一般的には 1970 年 1 月 1 日の深夜 0 時からのオフセットとして定義される特定の時点)。

  スナップショットが取られた時刻を協定世界時 (UTC) で入力します。

- SnapshotType — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  DB クラススナップショットのタイプを入力します。

- SourceDBClusterSnapshotArn — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  DB クラススナップショットがソース DB クラススナップショットからコピーされた場合は、ソース DB クラススナップショットの Amazon リソースネーム (ARN)、それ以外の場合は null 値。

- Status — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  この DB クラススナップショットのステータスを指定します。

- StorageEncrypted — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  DB クラススナップショットが暗号化されているかどうかを指定します。

- StorageType — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  DB クラススナップショットに関連付けられているストレージタイプ。

- VpcId — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  DB クラススナップショットに関連付けられている VPC ID を入力します。

`DBClusterSnapshot` は、以下のレスポンス要素として使用されます。

- CreateDBClusterSnapshot
- CopyDBClusterSnapshot
- DeleteDBClusterSnapshot

DBClusterSnapshotAttribute (構造)

手動 DB クラスタースナップショット属性の名前と値を含みます。

手動 DB クラスタースナップショット属性は、他の Amazon アカウントが手動 DB クラスタースナップショットを復元することを承認するために使用されます。詳細については、the section called “ModifyDBClusterSnapshotAttribute” API アクションを参照してください。

フィールド

- AttributeName — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  手動 DB クラスタースナップショット属性の名前。

  `restore` という名前の属性は、手動の DB クラスタースナップショットをコピーまたは復元するアクセス許可を持つ Amazon アカウントのリストを参照します。詳細については、the section called “ModifyDBClusterSnapshotAttribute” API アクションを参照してください。
- AttributeValues — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  手動 DB クラスタースナップショット属性の値。

  `AttributeName` フィールドが `restore` に設定されている場合、この要素は手動 DB クラスタースナップショットのコピーまたは復元を許可されている Amazon アカウントの ID のリストを返します。`all` の値がリストにある場合、手動 DB クラスタースナップショットは公開されており、どの Amazon アカウントでもコピーまたは復元できます。

DBClusterSnapshotAttributesResult (構造)

the section called “DescribeDBClusterSnapshotAttributes” API アクションへの呼び出しが成功した結果が含まれています。

手動 DB クラスタースナップショット属性は、他の Amazon アカウントが手動 DB クラスタースナップショットをコピーまたは復元することを承認するために使用されます。詳細については、the section called “ModifyDBClusterSnapshotAttribute” API アクションを参照してください。

フィールド

- DBClusterSnapshotAttributes — これは DBClusterSnapshotAttribute オブジェクトの配列です。

  手動 DB クラスタースナップショットの属性と値のリスト。

- DBClusterSnapshotIdentifier — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  属性が適用される手動 DB クラスタースナップショットの識別子。

`DBClusterSnapshotAttributesResult` は、以下のレスポンス要素として使用されます。

- DescribeDBClusterSnapshotAttributes
- ModifyDBClusterSnapshotAttribute

Neptune イベント API

アクション:

- CreateEventSubscription (アクション)
- DeleteEventSubscription (アクション)
- ModifyEventSubscription (アクション)
- DescribeEventSubscriptions (アクション)
- AddSourceIdentifierToSubscription (アクション)
- RemoveSourceIdentifierFromSubscription (アクション)
- DescribeEvents (アクション)
- DescribeEventCategories (アクション)

構造:

- イベント (構造)
- EventCategoriesMap (構造)
- EventSubscription (構造)

CreateEventSubscription (アクション)

> この API の AWS CLI 名は `create-event-subscription` です。

イベント通知サブスクリプションを作成します。このアクションには、Neptune コンソール、SNS コンソール、または SNS API のいずれかによって作成されたトピック ARN (Amazon リソースネー

ム) が必要です。SNS で ARN を取得するには、Amazon SNS でトピックを作成してそのトピックをサブスクライブする必要があります。ARN は SNS コンソールに表示されます。

通知するソースの種類 (SourceType) を指定し、イベントをトリガーする Neptune ソース (SourceIds) のリストを提供し、通知を受けたいイベントのイベントカテゴリのリスト (EventCategories) を指定することができます。たとえば、SourceType = db-instance, SourceIds = mydbinstance1, mydbinstance2 and EventCategories = Availability, Backup を指定できます。

SourceType = db-instance や SourceIdentifier = myDBInstance1 のように [SourceType] と [SourceIds] の両方を指定した場合は、指定したソースのすべての db-instance イベントが通知されます。[SourceType] を指定し、[SourceIdentifier] を指定しないと、すべての Neptune ソースについて、指定したソースタイプのイベントの通知を受け取ります。[SourceType] も [SourceIdentifier] も指定しない場合は、お客様のアカウントに属しているすべての Neptune ソースから生成されたイベントの通知を受け取ります。

リクエスト

- Enabled (CLI では: `--enabled`) — BooleanOptional、タイプ: `boolean` (ブール値 (真または偽))。

  ブール値。サブスクリプションを有効にするには true に設定し、サブスクリプションを作成するがアクティブにしない場合は false に設定します。
- EventCategories (CLI では: `--event-categories`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  サブスクライブする SourceType のイベントカテゴリのリスト。DescribeEventCategories アクションを使用して、特定の SourceType のカテゴリのリストを表示できます。
- SnsTopicArn (CLI では: `--sns-topic-arn`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  イベント通知用に作成された SNS トピックの Amazon リソースネーム (ARN)。ARN は、トピックを作成してそれをサブスクライブするときに Amazon SNS によって作成されます。
- SourceIds (CLI では: `--source-ids`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベントが返されるイベントソースの識別子のリスト。指定しない場合は、すべてのソースはレスポンスに含まれます。識別子は文字で開始し、ASCII 文字、数字、ハイフンのみを使用できます。最後の文字をハイフンにすることはできず、ハイフンを 2 つ続けて使用することもできません。

  制約:

- SourceIds が入力されている場合は、SourceType も指定する必要があります。
- ソースタイプが DB インスタンスである場合は、DBInstanceIdentifier を指定する必要があります。
- ソースタイプが DB セキュリティグループである場合は、DBSecurityGroupName を指定する必要があります。
- ソースタイプが DB パラメータグループである場合は、DBParameterGroupName を指定する必要があります。
- ソースタイプが DB スナップショットである場合は、DBSnapshotIdentifier を指定する必要があります。

- SourceType (CLI では: --source-type) — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  イベントを生成しているソースの種類。たとえば、DB インスタンスによって生成されたイベントの通知を受けたい場合は、このパラメータを db-instance に設定します。この値が指定されていない場合は、すべてのイベントが返されます。

  有効な値: db-instance | db-cluster | db-parameter-group | db-security-group | db-snapshot | db-cluster-snapshot
- SubscriptionName (CLI では: --subscription-name) — 必須: タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  サブスクリプションの名前。

  制約: この名前は 255 文字未満である必要があります。
- Tags (CLI では: --tags) - Tag オブジェクトの配列。

  新しいイベントサブスクリプションに適用されるタグ。

レスポンス

the section called “DescribeEventSubscriptions” アクションの呼び出しが成功した結果が含まれています。

- CustomerAwsId — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションに関連付けられている Amazon 顧客アカウント。
- CustSubscriptionId — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

イベント通知サブスクリプション ID。

- Enabled — ブール値、タイプ: boolean (ブール値 (真または偽))。

  サブスクリプションが有効になっているかどうかを示すブール値。True はサブスクリプションが有効になっていることを表します。
- EventCategoriesList — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションのイベントカテゴリのリスト。
- EventSubscriptionArn — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  イベントサブスクリプションの Amazon リソースネーム (ARN)。
- SnsTopicArn — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションのトピック ARN。
- SourceIdsList — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションのソース ID のリスト。
- SourceType — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションのソースタイプ。
- Status — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションのステータス。

  制約:

  次のいずれかを指定できます。作成 | 変更 | 削除 | アクティブ | 権限なし | トピックがありません

  ステータス「権限なし」は、Neptune が SNS トピックに投稿するアクセス許可を失ったことを示します。ステータス「トピックがありません」は、サブスクリプションの作成後にトピックが削除されたことを示します。
- SubscriptionCreationTime — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションが作成された時刻。

エラー

- EventSubscriptionQuotaExceededFault

- SubscriptionAlreadyExistFault
- SNSInvalidTopicFault
- SNSNoAuthorizationFault
- SNSTopicArnNotFoundFault
- SubscriptionCategoryNotFoundFault
- SourceNotFoundFault

DeleteEventSubscription (アクション)

この API の AWS CLI 名は `delete-event-subscription` です。

イベント通知サブスクリプションを削除します。

リクエスト

- SubscriptionName (CLI では: `--subscription-name`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  削除したいイベント通知サブスクリプションの名前。

レスポンス

the section called “DescribeEventSubscriptions” アクションの呼び出しが成功した結果が含まれています。

- CustomerAwsId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションに関連付けられている Amazon 顧客アカウント。
- CustSubscriptionId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプション ID。
- Enabled — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  サブスクリプションが有効になっているかどうかを示すブール値。True はサブスクリプションが有効になっていることを表します。
- EventCategoriesList — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションのイベントカテゴリのリスト。

- EventSubscriptionArn — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベントサブスクリプションの Amazon リソースネーム (ARN)。
- SnsTopicArn — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションのトピック ARN。
- SourceIdsList — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションのソース ID のリスト。
- SourceType — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションのソースタイプ。
- Status — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションのステータス。

  制約:

  次のいずれかを指定できます。作成 | 変更 | 削除 | アクティブ | 権限なし | トピックがありません

  ステータス「権限なし」は、Neptune が SNS トピックに投稿するアクセス許可を失ったことを示します。ステータス「トピックがありません」は、サブスクリプションの作成後にトピックが削除されたことを示します。
- SubscriptionCreationTime — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションが作成された時刻。

エラー

- SubscriptionNotFoundFault
- InvalidEventSubscriptionStateFault

ModifyEventSubscription (アクション)

この API の AWS CLI 名は `modify-event-subscription` です。

既存の RDS イベント通知サブスクリプションを変更します。この呼び出しを使ってソース識別子を変更することはできません。サブスクリプションのソース識別子を変更するには、the section called

"AddSourceIdentifierToSubscription" と the section called "RemoveSourceIdentifierFromSubscription" の呼び出しを使用します。

DescribeEventCategories アクションを使用して、特定の SourceType のイベントカテゴリのリストを表示できます。

リクエスト

- Enabled (CLI では: `--enabled`) — BooleanOptional、タイプ: `boolean` (ブール値 (真または偽))。

  ブール値。サブスクリプションを有効にするには、true に設定します。

- EventCategories (CLI では: `--event-categories`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  サブスクライブする SourceType のイベントカテゴリのリスト。DescribeEventCategories アクションを使用して、特定の SourceType のカテゴリのリストを表示できます。

- SnsTopicArn (CLI では: `--sns-topic-arn`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知用に作成された SNS トピックの Amazon リソースネーム (ARN)。ARN は、トピックを作成してそれをサブスクライブするときに Amazon SNS によって作成されます。

- SourceType (CLI では: `--source-type`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベントを生成しているソースの種類。たとえば、DB インスタンスによって生成されたイベントの通知を受けたい場合は、このパラメータを db-instance に設定します。この値が指定されていない場合は、すべてのイベントが返されます。

  有効な値: db-instance | db-parameter-group | db-security-group | db-snapshot

- SubscriptionName (CLI では: `--subscription-name`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションの名前。

レスポンス

the section called "DescribeEventSubscriptions" アクションの呼び出しが成功した結果が含まれています。

- CustomerAwsId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションに関連付けられている Amazon 顧客アカウント。
- CustSubscriptionId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプション ID。
- Enabled — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  サブスクリプションが有効になっているかどうかを示すブール値。True はサブスクリプションが有効になっていることを表します。
- EventCategoriesList — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションのイベントカテゴリのリスト。
- EventSubscriptionArn — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベントサブスクリプションの Amazon リソースネーム (ARN)。
- SnsTopicArn — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションのトピック ARN。
- SourceIdsList — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションのソース ID のリスト。
- SourceType — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションのソースタイプ。
- Status — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションのステータス。

  制約:

  次のいずれかを指定できます。作成 | 変更 | 削除 | アクティブ | 権限なし | トピックがありません

  ステータス「権限なし」は、Neptune が SNS トピックに投稿するアクセス許可を失ったことを示します。ステータス「トピックがありません」は、サブスクリプションの作成後にトピックが削除されたことを示します。
- SubscriptionCreationTime — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションが作成された時刻。

エラー

- EventSubscriptionQuotaExceededFault
- SubscriptionNotFoundFault
- SNSInvalidTopicFault
- SNSNoAuthorizationFault
- SNSTopicArnNotFoundFault
- SubscriptionCategoryNotFoundFault

DescribeEventSubscriptions (アクション)

この API の AWS CLI 名は `describe-event-subscriptions` です。

顧客アカウントのサブスクリプションの説明をすべて表示します。サブスクリプションの説明には、CustomerID、SourceType、SourceID SNSTopicARN SubscriptionName、CreationTime、およびステータスが含まれます。

SubscriptionName を指定した場合は、そのサブスクリプションの説明を一覧表示します。

リクエスト

- Filters (CLI では: `--filters`) - フィルター オブジェクトの配列。

  このパラメータは、現在サポートされていません。
- Marker (CLI では: `--marker`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  以前の DescribeOrderableDBInstanceOptions リクエストによって提供されたオプションのページ分割トークン。このパラメータを指定した場合、レスポンスには `MaxRecords` で指定された値まで、マーカーを超えるレコードのみが含まれます。
- MaxRecords (CLI では: `--max-records`) — IntegerOptional、タイプ: `integer` (符号付き 32 ビット整数)。

  レスポンスに含めるレコードの最大数。指定された `MaxRecords` の値よりも多くのレコードが存在する場合、マーカーと呼ばれるページ割りトークンがレスポンスに含まれるため、残りの結果を取得できます。

  デフォルト: 100

  制約: 最小 20、最大 100。

- SubscriptionName (CLI では: `--subscription-name`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  記述するイベント通知サブスクリプションの名前。

レスポンス

- EventSubscriptionsList – EventSubscription オブジェクトの配列。

  EventSubscriptions データ型のリスト。
- Marker — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  以前の DescribeOrderableDBInstanceOptions リクエストによって提供されたオプションのページ分割トークン。このパラメータを指定した場合、レスポンスには `MaxRecords` で指定された値まで、マーカーを超えるレコードのみが含まれます。

エラー

- SubscriptionNotFoundFault

AddSourceIdentifierToSubscription (アクション)

この API の AWS CLI 名は `add-source-identifier-to-subscription` です。

既存のイベント通知サブスクリプションにソース識別子を追加します。

リクエスト

- SourceIdentifier (CLI では: `--source-identifier`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  追加されるイベントソースの識別子。

  制約:

  - ソースタイプが DB インスタンスである場合は、`DBInstanceIdentifier` を指定する必要があります。
  - ソースタイプが DB セキュリティグループである場合は、`DBSecurityGroupName` を指定する必要があります。

  - ソースタイプが DB パラメータグループである場合は、DBParameterGroupName を指定する必要があります。
  - ソースタイプが DB スナップショットである場合は、DBSnapshotIdentifier を指定する必要があります。
- SubscriptionName (CLI では: --subscription-name) — 必須: タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  ソース識別子を追加するイベント通知サブスクリプションの名前。

レスポンス

the section called “DescribeEventSubscriptions” アクションの呼び出しが成功した結果が含まれています。

- CustomerAwsId — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションに関連付けられている Amazon 顧客アカウント。
- CustSubscriptionId — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプション ID。
- Enabled — ブール値、タイプ: boolean (ブール値 (真または偽))。

  サブスクリプションが有効になっているかどうかを示すブール値。True はサブスクリプションが有効になっていることを表します。
- EventCategoriesList — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションのイベントカテゴリのリスト。
- EventSubscriptionArn — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  イベントサブスクリプションの Amazon リソースネーム (ARN)。
- SnsTopicArn — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションのトピック ARN。
- SourceIdsList — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションのソース ID のリスト。
- SourceType — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

イベント通知サブスクリプションのソースタイプ。

- Status — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションのステータス。

  制約:

  次のいずれかを指定できます。作成 | 変更 | 削除 | アクティブ | 権限なし | トピックがありません

  ステータス「権限なし」は、Neptune が SNS トピックに投稿するアクセス許可を失ったことを示します。ステータス「トピックがありません」は、サブスクリプションの作成後にトピックが削除されたことを示します。
- SubscriptionCreationTime — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションが作成された時刻。

エラー

- SubscriptionNotFoundFault
- SourceNotFoundFault

RemoveSourceIdentifierFromSubscription (アクション)

この API の AWS CLI 名は `remove-source-identifier-from-subscription` です。

既存のイベント通知サブスクリプションからソース識別子を削除します。

リクエスト

- SourceIdentifier (CLI では: `--source-identifier`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB インスタンスの場合は DB インスタンス識別子など、サブスクリプションから削除されるソース識別子、またはセキュリティグループの名前。
- SubscriptionName (CLI では: `--subscription-name`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  ソース識別子を削除するイベント通知サブスクリプションの名前。

レスポンス

the section called “DescribeEventSubscriptions” アクションの呼び出しが成功した結果が含まれています。

- CustomerAwsId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションに関連付けられている Amazon 顧客アカウント。
- CustSubscriptionId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプション ID。
- Enabled — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  サブスクリプションが有効になっているかどうかを示すブール値。True はサブスクリプションが有効になっていることを表します。
- EventCategoriesList — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションのイベントカテゴリのリスト。
- EventSubscriptionArn — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベントサブスクリプションの Amazon リソースネーム (ARN)。
- SnsTopicArn — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションのトピック ARN。
- SourceIdsList — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションのソース ID のリスト。
- SourceType — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションのソースタイプ。
- Status — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションのステータス。

  制約:

  次のいずれかを指定できます。作成 | 変更 | 削除 | アクティブ | 権限なし | トピックがありません

ステータス「権限なし」は、Neptune が SNS トピックに投稿するアクセス許可を失ったことを示します。ステータス「トピックがありません」は、サブスクリプションの作成後にトピックが削除されたことを示します。

- SubscriptionCreationTime — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションが作成された時刻。

エラー

- SubscriptionNotFoundFault
- SourceNotFoundFault

DescribeEvents (アクション)

> この API の AWS CLI 名は `describe-events` です。

DB インスタンス、DB セキュリティグループ、DB スナップショット、DB パラメータグループに関連する過去 14 日間のイベントを返します。特定の DB インスタンス、DB セキュリティグループ、データベーススナップショット、または DB パラメータグループに固有のイベントは、名前をパラメータとして指定することによって取得できます。デフォルトでは、過去 1 時間のイベントが返されます。

リクエスト

- Duration (CLI では: `--duration`) — IntegerOptional、タイプ: `integer` (符号付き 32 ビット整数)。

  イベントを取得するための分数。

  デフォルト: 60

- EndTime (CLI では: `--end-time`) — TStamp、タイプ: `timestamp` (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定時点)。

  ISO 8601 形式で指定された、イベントを取得する時間間隔の終了時刻。ISO 8601 の詳細については、ISO8601 Wikipedia のページを参照してください。

  例: 2009-07-08T18:00Z

- EventCategories (CLI では: `--event-categories`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションの通知をトリガーするイベントカテゴリのリスト。

- Filters (CLI では: `--filters`) - フィルター オブジェクトの配列。

  このパラメータは、現在サポートされていません。

- Marker (CLI では: `--marker`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  以前の DescribeEvents リクエストによって提供されたオプションのページ分割トークン。このパラメータを指定した場合、レスポンスには `MaxRecords` で指定された値まで、マーカーを超えるレコードのみが含まれます。

- MaxRecords (CLI では: `--max-records`) — IntegerOptional、タイプ: `integer` (符号付き 32 ビット整数)。

  レスポンスに含めるレコードの最大数。指定された `MaxRecords` の値よりも多くのレコードが存在する場合、マーカーと呼ばれるページ割りトークンがレスポンスに含まれるため、残りの結果を取得できます。

  デフォルト: 100

  制約: 最小 20、最大 100。

- SourceIdentifier (CLI では: `--source-identifier`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベントが返されるイベントソースの識別子。指定しない場合は、すべてのソースはレスポンスに含まれます。

  制約:

  - SourceIdentifier が入力されている場合は、SourceType も指定する必要があります。
  - ソースタイプが `DBInstance` である場合は、`DBInstanceIdentifier` を指定する必要があります。
  - ソースタイプが `DBSecurityGroup` である場合は、`DBSecurityGroupName` を指定する必要があります。
  - ソースタイプが `DBParameterGroup` である場合は、`DBParameterGroupName` を指定する必要があります。

  - ソースタイプが DBSnapshot である場合は、DBSnapshotIdentifier を指定する必要があります。
  - ハイフンを、文字列の最後に使用したり、2 つ続けて使用したりすることはできません。
- SourceType(CLI では: --source-type) — SourceType、タイプ: string (UTF-8 でエンコードされた文字列)。

  イベントソースのイベントを取得します。値を指定しない場合、すべてのイベントが返されます。
- StartTime (CLI では: --start-time) — TStamp、タイプ: timestamp (通常は 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定時点)。

  ISO 8601 形式で指定された、イベントを取得する時間間隔の開始時刻。ISO 8601 の詳細については、ISO8601 Wikipedia のページを参照してください。

  例: 2009-07-08T18:00Z

レスポンス

- Events – イベント オブジェクトの配列。

  the section called “イベント” インスタンスのリスト。
- Marker — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  以前の Events リクエストによって提供されたオプションのページ分割トークン。このパラメータを指定した場合、レスポンスには MaxRecords で指定された値まで、マーカーを超えるレコードのみが含まれます。

DescribeEventCategories (アクション)

この API の AWS CLI 名は describe-event-categories です。

すべてのイベントソースタイプか、指定されている場合は、指定されたソースタイプのイベントカテゴリのリストを表示します。

リクエスト

- Filters (CLI では: --filters) - フィルター オブジェクトの配列。

  このパラメータは、現在サポートされていません。

- SourceType (CLI では: `--source-type`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベントを生成しているソースの種類。

  有効な値: db-instance | db-parameter-group | db-security-group | db-snapshot

レスポンス

- EventCategoriesMapList – EventCategoriesMap オブジェクトの配列。

  EventCategoriesMap データ型のリスト。

構造:

イベント (構造)

このデータ型は、the section called “DescribeEvents” アクションのレスポンス要素として使用されます。

フィールド

- Date — TStamp、タイプ: `timestamp` (一般的には 1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定時点)。

  イベントの日付と時刻を指定します。
- EventCategories — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベントのカテゴリを指定します。
- Message — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  このイベントのテキストを入力します。
- SourceArn — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベントの Amazon リソースネーム (ARN)
- SourceIdentifier — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベントの発生元の識別子を入力します。
- SourceType — SourceType、タイプ: `string` (UTF-8 でエンコードされた文字列)。

このイベントのソースタイプを指定します。

EventCategoriesMap (構造)

the section called “DescribeEventCategories” アクションの呼び出しが成功した結果が含まれています。

フィールド

- EventCategories — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  指定されたソースタイプのイベントカテゴリ
- SourceType — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  返されたカテゴリに属しているソースタイプ

EventSubscription (構造)

the section called “DescribeEventSubscriptions” アクションの呼び出しが成功した結果が含まれています。

フィールド

- CustomerAwsId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションに関連付けられている Amazon 顧客アカウント。
- CustSubscriptionId — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプション ID。
- Enabled — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  サブスクリプションが有効になっているかどうかを示すブール値。True はサブスクリプションが有効になっていることを表します。
- EventCategoriesList — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションのイベントカテゴリのリスト。
- EventSubscriptionArn — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

イベントサブスクリプションの Amazon リソースネーム (ARN)。

- SnsTopicArn — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションのトピック ARN。

- SourceIdsList — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションのソース ID のリスト。

- SourceType — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションのソースタイプ。

- Status — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションのステータス。

  制約:

  次のいずれかを指定できます。作成 | 変更 | 削除 | アクティブ | 権限なし | トピックがありません

  ステータス「権限なし」は、Neptune が SNS トピックに投稿するアクセス許可を失ったことを示します。ステータス「トピックがありません」は、サブスクリプションの作成後にトピックが削除されたことを示します。

- SubscriptionCreationTime — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  イベント通知サブスクリプションが作成された時刻。

`EventSubscription` は、以下のレスポンス要素として使用されます。

- CreateEventSubscription
- ModifyEventSubscription
- AddSourceIdentifierToSubscription
- RemoveSourceIdentifierFromSubscription
- DeleteEventSubscription

その他の Neptune API

アクション:

- AddTagsToResource (アクション)
- ListTagsForResource (アクション)
- RemoveTagsFromResource (アクション)
- ApplyPendingMaintenanceAction (アクション)
- DescribePendingMaintenanceActions (アクション)
- DescribeDBEngineVersions (アクション)

構造:

- DBEngineVersion (構造)
- EngineDefaults (構造)
- PendingMaintenanceAction (構造)
- ResourcePendingMaintenanceActions (構造)
- UpgradeTarget (構造)
- タグ (構造)

AddTagsToResource (アクション)

この API の AWS CLI 名は `add-tags-to-resource` です。

Amazon Neptune リソースにメタデータタグを追加します。これらのタグは、Amazon Neptune リソースに関連するコストを追跡するためのコスト割り当てレポートで使用することも、Amazon Neptune の IAM ポリシーの Condition ステートメントで使用することもできます。

リクエスト

- ResourceName (CLI では: `--resource-name`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  タグが追加された Amazon Neptune リソース。この値は Amazon リソースネーム (ARN) です。ARN 作成の詳細については、「Amazon リソースネーム (ARN) の構築」を参照してください。
- Tags (CLI では: `--tags`) - 必須: Tag オブジェクトの配列。

  Amazon Neptune リソースに割り当てられるタグ。

レスポンス

- 応答パラメータはありません。

エラー

- DBInstanceNotFoundFault
- DBSnapshotNotFoundFault
- DBClusterNotFoundFault

ListTagsForResource (アクション)

この API の AWS CLI 名は `list-tags-for-resource` です。

Amazon Neptune リソースのすべてのタグを一覧表示します。

リクエスト

- Filters (CLI では: `--filters`) - フィルター オブジェクトの配列。

  このパラメータは、現在サポートされていません。
- ResourceName (CLI では: `--resource-name`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  一覧されるタグを持つ Amazon Neptune リソース。この値は Amazon リソースネーム (ARN) です。ARN 作成の詳細については、「Amazon リソースネーム (ARN) の構築」を参照してください。

レスポンス

- TagList – Tag オブジェクトの配列。

  ListTagsForResource オペレーションによって返されるタグのリスト。

エラー

- DBInstanceNotFoundFault
- DBSnapshotNotFoundFault

- DBClusterNotFoundFault

RemoveTagsFromResource (アクション)

この API の AWS CLI 名は `remove-tags-from-resource` です。

Amazon Neptune リソースからメタデータタグを削除します。

リクエスト

- ResourceName (CLI では: `--resource-name`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  タグが削除された Amazon Neptune リソース。この値は Amazon リソースネーム (ARN) です。ARN 作成の詳細については、「Amazon リソースネーム (ARN) の構築」を参照してください。
- TagKeys (CLI では: `--tag-keys`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  削除するタグのタグキー (名前)。

レスポンス

- 応答パラメータはありません。

エラー

- DBInstanceNotFoundFault
- DBSnapshotNotFoundFault
- DBClusterNotFoundFault

ApplyPendingMaintenanceAction (アクション)

この API の AWS CLI 名は `apply-pending-maintenance-action` です。

保留中のメンテナンスアクションをリソース (たとえば DB インスタンス) に適用します。

リクエスト

- ApplyAction (CLI では: `--apply-action`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  このリソースに適用する保留中のメンテナンスアクション。

  有効な値: `system-update`、`db-upgrade`

- OptInType (CLI では: `--opt-in-type`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  オプトインリクエストのタイプを指定するか、オプトインクエストを元に戻す値。タイプが `immediate` のオプトインリクエストは元に戻すことができません。

  有効な値:

  - `immediate` - メンテナンスアクションをすぐに適用します。
  - `next-maintenance` - リソースの次のメンテナンスウィンドウ中にメンテナンスアクションを適用します。
  - `undo-opt-in` - 既存の `next-maintenance` オプトインリクエストをキャンセルします。

- ResourceIdentifier (CLI では: `--resource-identifier`) — 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  保留中のアクションが適用されるリソースの Amazon リソースネーム (ARN)。ARN 作成の詳細については、「Amazon リソースネーム (ARN) の構築」を参照してください。

レスポンス

リソースに対する保留中のメンテナンスアクションについて説明します。

- PendingMaintenanceActionDetails – PendingMaintenanceAction オブジェクトの配列。

  リソースの保留中のメンテナンスアクションに関する詳細を提供するリスト。

- ResourceIdentifier — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  保留中のメンテナンスアクションがあるリソースの ARN。

エラー

- ResourceNotFoundFault

DescribePendingMaintenanceActions (アクション)

この API の AWS CLI 名は `describe-pending-maintenance-actions` です。

少なくとも 1 つの保留中のメンテナンスアクションを含むリソース (例: DB インスタンス) のリストを返します。

リクエスト

- Filters (CLI では: `--filters`) - フィルター オブジェクトの配列。

  保留中のメンテナンスアクションを返す 1 つ以上のリソースを指定するフィルター。

  サポートされているフィルター:

  - `db-cluster-id` - DB クラスター識別子と DB クラスターの Amazon リソースネーム (ARN) を受け入れます。結果のリストには、これらの ARN で識別された DB クラスターの保留中のメンテナンスアクションのみが含まれます。
  - `db-instance-id` - DB インスタンス識別子と DB インスタンス ARN を受け入れます。結果のリストには、これらの ARN で識別された DB インスタンスの保留中のメンテナンスアクションのみが含まれます。
- Marker (CLI では: `--marker`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  以前の `DescribePendingMaintenanceActions` リクエストによって提供されたオプションのページ割りトークン。このパラメータを指定した場合、レスポンスには `MaxRecords` で指定されたレコード数まで、マーカーを超えるレコードのみが含まれます。
- MaxRecords (CLI では: `--max-records`) — IntegerOptional、タイプ: `integer` (符号付き 32 ビット整数)。

  レスポンスに含めるレコードの最大数。指定された `MaxRecords` の値よりも多くのレコードが存在する場合、マーカーと呼ばれるページ割りトークンがレスポンスに含まれるため、残りの結果を取得できます。

  デフォルト: 100

  制約: 最小 20、最大 100。
- ResourceIdentifier (CLI では: `--resource-identifier`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  保留中のメンテナンスアクションを返すリソースの ARN。

レスポンス

- Marker — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  以前の `DescribePendingMaintenanceActions` リクエストによって提供されたオプションのページ割りトークン。このパラメータを指定した場合、レスポンスには `MaxRecords` で指定されたレコード数まで、マーカーを超えるレコードのみが含まれます。

- PendingMaintenanceActions – ResourcePendingMaintenanceActions オブジェクトの配列。

  リソースに対する保留中のメンテナンスアクションのリスト。

エラー

- ResourceNotFoundFault

DescribeDBEngineVersions (アクション)

この API の AWS CLI 名は `describe-db-engine-versions` です。

利用可能な DB エンジンのリストを返します。

リクエスト

- DBParameterGroupFamily (CLI では: `--db-parameter-group-family`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  詳細を返す特定の DB パラメータグループファミリーの名前。

  制約:

  - 指定した場合、既存の DBParameterGroupFamily と一致する必要があります。

- DefaultOnly (CLI では: `--default-only`) — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  指定されたエンジンまたはエンジンとメジャーバージョンの組み合わせのデフォルトバージョンのみが返されることを示します。

- Engine (CLI では: `--engine`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  返されるデータベースエンジン。

- EngineVersion (CLI では: `--engine-version`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  返されるデータベースエンジンのバージョン。

  例: `5.1.49`

- Filters (CLI では: `--filters`) - フィルター オブジェクトの配列。

  現在サポートされていません。

- ListSupportedCharacterSets (CLI では: `--list-supported-character-sets`) — BooleanOptional、タイプ: `boolean` (ブール値 (真または偽))。

  このパラメータが指定され、要求されたエンジンが `CreateDBInstance` に対して `CharacterSetName` パラメータをサポートしている場合、レスポンスには各エンジンバージョンでサポートされている文字セットのリストが含まれます。

- ListSupportedTimezones (CLI では: `--list-supported-timezones`) — BooleanOptional、タイプ: `boolean` (ブール値 (真または偽))。

  このパラメータが指定され、要求されたエンジンが `CreateDBInstance` に対して `TimeZone` パラメータをサポートしている場合、レスポンスには各エンジンバージョンでサポートされているタイムゾーンのリストが含まれます。

- Marker (CLI では: `--marker`) — 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  以前のリクエストによって提供されたオプションのページ割りトークン。このパラメータを指定した場合、レスポンスには `MaxRecords` で指定された値まで、マーカーを超えるレコードのみが含まれます。

- MaxRecords (CLI では: `--max-records`) — IntegerOptional、タイプ: `integer` (符号付き 32 ビット整数)。

  レスポンスに含めるレコードの最大数。指定された `MaxRecords` の値よりも多くのレコードが利用可能な場合、マーカーと呼ばれるページ分割トークンがレスポンスに含まれるため、以下の結果を取得できます。

  デフォルト: 100

  制約: 最小 20、最大 100。

レスポンス

- DBEngineVersions – DBEngineVersion オブジェクトの配列。

  DBEngineVersion 要素のリスト。

- Marker — 文字列、タイプ: string (UTF-8 でエンコードされた文字列)。

  以前のリクエストによって提供されたオプションのページ割りトークン。このパラメータを指定した場合、レスポンスには MaxRecords で指定された値まで、マーカーを超えるレコードのみが含まれます。

構造:

DBEngineVersion (構造)

このデータ型は、the section called “DescribeDBEngineVersions” アクションのレスポンス要素として使用されます。

フィールド

- DBEngineDescription — 文字列、タイプ:string (UTF-8 でエンコードされた文字列)。

  データベースエンジンの説明。

- DBEngineVersionDescription — 文字列、タイプ:string (UTF-8 でエンコードされた文字列)。

  データベースエンジンのバージョンの説明。

- DBParameterGroupFamily — 文字列、タイプ:string (UTF-8 でエンコードされた文字列)。

  データベースエンジンの DB パラメータグループファミリーの名前。

- Engine — 文字列、タイプ:string (UTF-8 でエンコードされた文字列)。

  データベースエンジンの名前。

- EngineVersion — 文字列、タイプ:string (UTF-8 でエンコードされた文字列)。

  データベースエンジンのバージョン番号。

- ExportableLogTypes — 文字列、タイプ:string (UTF-8 でエンコードされた文字列)。

  データベースエンジンが CloudWatch Logs にエクスポートできるログのタイプ。

- SupportedTimezones — タイムゾーン オブジェクトの配列。

CreateDBInstance アクションの Timezone パラメータに対してこのエンジンでサポートされているタイムゾーンのリスト。

- SupportsGlobalDatabases — ブール値、タイプ: boolean (ブール値 (真または偽))。

  特定の DB エンジンバージョンを持つ Aurora グローバルデータベースを使用できるかどうかを示す値。

- SupportsLogExportsToCloudwatchLogs — ブール値、タイプ: boolean (ブール値 (真または偽))。

  エンジンのバージョンが ExportableLogTypes で指定されたログタイプの CloudWatch Logs へのエクスポートをサポートするかどうかを示す値。

- SupportsReadReplica — ブール値、タイプ: boolean (ブール値 (真または偽))。

  データベースエンジンのバージョンがリードレプリカをサポートしているかどうかを示します。

- ValidUpgradeTarget — UpgradeTarget オブジェクトの配列。

  このデータベースエンジンのバージョンをアップグレードできるエンジンのバージョンのリスト。

EngineDefaults (構造)

the section called “DescribeEngineDefaultParameters” アクションの呼び出しが成功した結果が含まれています。

フィールド

- DBParameterGroupFamily — 文字列、タイプ:string (UTF-8 でエンコードされた文字列)。

  エンジンのデフォルトパラメータが適用される DB パラメータグループファミリーの名前を指定します。

- Marker — 文字列、タイプ:string (UTF-8 でエンコードされた文字列)。

  以前の EngineDefaults リクエストによって提供されたオプションのページ分割トークン。このパラメータを指定した場合、レスポンスには MaxRecords で指定された値まで、マーカーを超えるレコードのみが含まれます。

- Parameters — パラメータ オブジェクトの配列。

  エンジンのデフォルトパラメータのリストが含まれます。

EngineDefaults は、以下のレスポンス要素として使用されます。

- DescribeEngineDefaultParameters
- DescribeEngineDefaultClusterParameters

PendingMaintenanceAction (構造)

リソースの保留中のメンテナンスアクションに関する情報を入力します。

フィールド

- Action — 文字列、タイプ:string (UTF-8 でエンコードされた文字列)。

  リソースに対して使用可能な保留中のメンテナンスアクションのタイプ。
- AutoAppliedAfterDate — TStamp、タイプ: timestamp (通常、1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定時点)。

  アクションが適用されたときのメンテナンスウィンドウの日付。メンテナンスアクションは、この日以降の最初のメンテナンスウィンドウ中にリソースに適用されます。この日付を指定した場合、next-maintenance オプトインリクエストは無視されます。
- CurrentApplyDate — TStamp、タイプ: timestamp (通常、1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定時点)。

  保留中のメンテナンスアクションがリソースに適用される有効日。この日付では、the section called “ApplyPendingMaintenanceAction” API、AutoAppliedAfterDate、および ForcedApplyDate から受信したオプトインリクエストが考慮されます。オプトインリクエストが受信されておらず、AutoAppliedAfterDate または ForcedApplyDate として何も指定されていない場合、この値は空白になります。
- Description — 文字列、タイプ:string (UTF-8 でエンコードされた文字列)。

  メンテナンスアクションについての詳細を提供する説明。
- ForcedApplyDate — TStamp、タイプ: timestamp (通常、1970 年 1 月 1 日深夜 0 時からのオフセットとして定義される特定時点)。

  アクションが自動的に適用されたときのメンテナンスウィンドウの日付。メンテナンスアクションは、リソースのメンテナンスウィンドウに関係なく、この日にリソースに適用されます。この日付を指定した場合、immediate オプトインリクエストは無視されます。
- OptInStatus — 文字列、タイプ:string (UTF-8 でエンコードされた文字列)。

リソースに対して受信されたオプトインリクエストのタイプを示します。

ResourcePendingMaintenanceActions (構造)

リソースに対する保留中のメンテナンスアクションについて説明します。

フィールド

- PendingMaintenanceActionDetails — PendingMaintenanceAction オブジェクトの配列。

  リソースの保留中のメンテナンスアクションに関する詳細を提供するリスト。
- ResourceIdentifier — 文字列、タイプ:`string` (UTF-8 でエンコードされた文字列)。

  保留中のメンテナンスアクションがあるリソースの ARN。

`ResourcePendingMaintenanceActions` は、以下のレスポンス要素として使用されます。

- ApplyPendingMaintenanceAction

UpgradeTarget (構造)

DB インスタンスをアップグレードできるデータベースエンジンのバージョン。

フィールド

- AutoUpgrade — ブール値、タイプ: `boolean` (ブール値 (真または偽))。

  AutoMinorVersionUpgrade が true に設定されているソース DB インスタンスにターゲットバージョンが適用されるかどうかを示す値。
- Description — 文字列、タイプ:`string` (UTF-8 でエンコードされた文字列)。

  DB インスタンスをアップグレードできるデータベースエンジンのバージョン。
- Engine — 文字列、タイプ:`string` (UTF-8 でエンコードされた文字列)。

  アップグレードターゲットデータベースエンジンの名前。
- EngineVersion — 文字列、タイプ:`string` (UTF-8 でエンコードされた文字列)。

  アップグレードターゲットのデータベースエンジンのバージョン番号。

- IsMajorVersionUpgrade — ブール値、タイプ: boolean (ブール値 (真または偽))。

  データベースエンジンがメジャーバージョンにアップグレードされているかどうかを示す値。
- SupportsGlobalDatabases — BooleanOptional、タイプ: boolean (ブール値 (真または偽))。

  ターゲットのエンジンバージョンを持つ Neptune グローバルデータベースを使用できるかどうかを示す値。

タグ (構造)

キーと値のペアで構成される Amazon Neptune リソースに割り当てられるメタデータ。

フィールド

- Key — 文字列、タイプ:string (UTF-8 でエンコードされた文字列)。

  キーはタグの必須の名前です。文字列値は、1〜128 文字の Unicode 文字です。aws: または rds: をプレフィックスとして使用することはできません。文字列には、一連の Unicode 文字、数字、空白、「_」、「.」、「/」、「=」、「+」、「-」 (Java 正規表現: "^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)") のみ使用できます。
- Value — 文字列、タイプ:string (UTF-8 でエンコードされた文字列)。

  値はタグの省略可能な値です。文字列値は、1〜256 文字の Unicode 文字です。aws: または rds: をプレフィックスとして使用することはできません。文字列には、一連の Unicode 文字、数字、空白、「_」、「.」、「/」、「=」、「+」、「-」 (Java 正規表現: "^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)") のみ使用できます。

一般的な Neptune のデータ型

構造:

- アベイラビリティーゾーン (構造)
- DBSecurityGroupMembership (構造)
- DomainMembership (構造)
- DoubleRange (構造)
- エンドポイント (構造)
- フィルター (構造)

- 範囲 (構造)
- ServerlessV2ScalingConfiguration (構造)
- ServerlessV2ScalingConfigurationInfo (構造)
- タイムゾーン (構造)
- VpcSecurityGroupMembership (構造)

アベイラビリティーゾーン (構造)

アベイラビリティーゾーンを指定します。

フィールド

- Name - 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  アベイラビリティーゾーンの名前。

DBSecurityGroupMembership (構造)

指定された DB セキュリティグループのメンバーシップを指定します。

フィールド

- DBSecurityGroupName - 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB セキュリティグループの名前。
- Status - 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB セキュリティグループのステータス。

DomainMembership (構造)

DB インスタンスに関連付けられている Active Directory ドメインメンバーシップレコード。

フィールド

- Domain - 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  Active Directory ドメインの識別子。

- FQDN - 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  Active Directory ドメインの完全修飾ドメイン名。
- IAMRoleName - 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  ディレクトリサービスへの API 呼び出しを行うときに使用される IAM ロールの名前。
- Status - 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  参加、保留中の結合、失敗など、DB インスタンスの Active Directory ドメインメンバーシップのステータス。

DoubleRange (構造)

倍精度値の範囲。

フィールド

- From — Double、タイプ: `double` (倍精度 IEEE 754 浮動小数点数)。

  範囲の最小値。
- To — Double、タイプ: `double` (倍精度 IEEE 754 浮動小数点数)。

  範囲の最大値。

エンドポイント (構造)

接続エンドポイントを指定します。

Amazon Neptune DB クラスターエンドポイントを表すデータ構造については、`DBClusterEndpoint` を参照してください。

フィールド

- Address - 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  DB インスタンスの DNS アドレスを指定します。
- HostedZoneId - 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  ホストゾーンを作成するときに Amazon Route 53 が割り当てる ID を指定します。

- Port — 整数、タイプ: `integer` (符号付き 32 ビット整数)。

  データベースエンジンがリッスンするポートを指定します。

フィルター (構造)

現時点でこのタイプはサポートされていません。

フィールド

- Name - 必須、文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  このパラメータは、現在サポートされていません。
- Values - 必須、文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  このパラメータは、現在サポートされていません。

範囲 (構造)

整数値の範囲。

フィールド

- From — 整数、タイプ: `integer` (符号付き 32 ビット整数)。

  範囲の最小値。
- Step — IntegerOptional、タイプ: `integer` (符号付き 32 ビット整数)。

  範囲の増分値。たとえば、5,000 ~ 10,000 の範囲で、ステップ値が 1,000 の場合、有効な値は 5,000 から始まり、1,000 ずつ増加します。7,500 が範囲内であっても、その範囲の有効な値ではありません。有効な値は、5,000、6,000、7,000、8,000... です。
- To — 整数、タイプ: `integer` (符号付き 32 ビット整数)。

  範囲の最大値。

ServerlessV2ScalingConfiguration (構造)

Neptune サーバーレス DB クラスターのスケーリング構成を含みます。

詳細については、「Amazon Neptune ユーザーガイド」の 「Amazon Neptune サーバーレスの使用」を参照してください。

フィールド

- MaxCapacity — DoubleOptional、タイプ: `double` (倍精度 IEEE 754 浮動小数点数)。

  Neptune サーバーレスクラスター内の DB インスタンスの Neptune 容量ユニット (NCU) の最大数。NCU 値は、40、40.5、41 など、半段階刻みで指定できます。
- MinCapacity — DoubleOptional、タイプ: `double` (倍精度 IEEE 754 浮動小数点数)。

  Neptune サーバーレスクラスター内の DB インスタンスの Neptune 容量ユニット (NCU) の最小数。NCU 値は、8、8.5、9 など、半段階刻みで指定できます。

ServerlessV2ScalingConfigurationInfo (構造)

Neptune サーバーレス DB クラスターのスケーリング設定を表示します。

詳細については、「Amazon Neptune ユーザーガイド」の 「Amazon Neptune サーバーレスの使用」を参照してください。

フィールド

- MaxCapacity — DoubleOptional、タイプ: `double` (倍精度 IEEE 754 浮動小数点数)。

  Neptune サーバーレスクラスター内の DB インスタンスの Neptune 容量ユニット (NCU) の最大数。NCU 値は、40、40.5、41 など、半段階刻みで指定できます。
- MinCapacity — DoubleOptional、タイプ: `double` (倍精度 IEEE 754 浮動小数点数)。

  Neptune サーバーレスクラスター内の DB インスタンスの Neptune 容量ユニット (NCU) の最小数。NCU 値は、8、8.5、9 など、半段階刻みで指定できます。

タイムゾーン (構造)

the section called “DBInstance” に関連付けられたタイムゾーン。

フィールド

- TimezoneName - 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

タイムゾーンの名前。

VpcSecurityGroupMembership (構造)

このデータ型は、VPC セキュリティグループメンバシップに関するクエリのレスポンス要素として使用されます。

フィールド

- Status - 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  VPC セキュリティグループのステータス。
- VpcSecurityGroupId - 文字列、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  VPC セキュリティグループの名前。

個々の API に固有の Neptune 例外

例外:

- AuthorizationAlreadyExistsFault (構造)
- AuthorizationNotFoundFault (構造)
- AuthorizationQuotaExceededFault (構造)
- CertificateNotFoundFault (構造)
- DBClusterAlreadyExistsFault (構造)
- DBClusterNotFoundFault (構造)
- DBClusterParameterGroupNotFoundFault (構造)
- DBClusterQuotaExceededFault (構造)
- DBClusterRoleAlreadyExistsFault (構造)
- DBClusterRoleNotFoundFault (構造)
- DBClusterRoleQuotaExceededFault (構造)
- DBClusterSnapshotAlreadyExistsFault (構造)
- DBClusterSnapshotNotFoundFault (構造)
- DBInstanceAlreadyExistsFault (構造)

- DBInstanceNotFoundFault (構造)
- DBLogFileNotFoundFault (構造)
- DBParameterGroupAlreadyExistsFault (構造)
- DBParameterGroupNotFoundFault (構造)
- DBParameterGroupQuotaExceededFault (構造)
- DBSecurityGroupAlreadyExistsFault (構造)
- DBSecurityGroupNotFoundFault (構造)
- DBSecurityGroupNotSupportedFault (構造)
- DBSecurityGroupQuotaExceededFault (構造)
- DBSnapshotAlreadyExistsFault (構造)
- DBSnapshotNotFoundFault (構造)
- DBSubnetGroupAlreadyExistsFault (構造)
- DBSubnetGroupDoesNotCoverEnoughAZs (構造)
- DBSubnetGroupNotAllowedFault (構造)
- DBSubnetGroupNotFoundFault (構造)
- DBSubnetGroupQuotaExceededFault (構造)
- DBSubnetQuotaExceededFault (構造)
- DBUpgradeDependencyFailureFault (構造)
- DomainNotFoundFault (構造)
- EventSubscriptionQuotaExceededFault (構造)
- GlobalClusterAlreadyExistsFault (構造)
- GlobalClusterNotFoundFault (構造)
- GlobalClusterQuotaExceededFault (構造)
- InstanceQuotaExceededFault (構造)
- InsufficientDBClusterCapacityFault (構造)
- InsufficientDBInstanceCapacityFault (構造)
- InsufficientStorageClusterCapacityFault (構造)
- InvalidDBClusterEndpointStateFault (構造)
- InvalidDBClusterSnapshotStateFault (構造)
- InvalidDBClusterStateFault (構造)

- InvalidDBInstanceStateFault (構造)
- InvalidDBParameterGroupStateFault (構造)
- InvalidDBSecurityGroupStateFault (構造)
- InvalidDBSnapshotStateFault (構造)
- InvalidDBSubnetGroupFault (構造)
- InvalidDBSubnetGroupStateFault (構造)
- InvalidDBSubnetStateFault (構造)
- InvalidEventSubscriptionStateFault (構造)
- InvalidGlobalClusterStateFault (構造)
- InvalidOptionGroupStateFault (構造)
- InvalidRestoreFault (構造)
- InvalidSubnet (構造)
- InvalidVPCNetworkStateFault (構造)
- KMSKeyNotAccessibleFault (構造)
- OptionGroupNotFoundFault (構造)
- PointInTimeRestoreNotEnabledFault (構造)
- ProvisionedIopsNotAvailableInAZFault (構造)
- ResourceNotFoundFault (構造)
- SNSInvalidTopicFault (構造)
- SNSNoAuthorizationFault (構造)
- SNSTopicArnNotFoundFault (構造)
- SharedSnapshotQuotaExceededFault (構造)
- SnapshotQuotaExceededFault (構造)
- SourceNotFoundFault (構造)
- StorageQuotaExceededFault (構造)
- StorageTypeNotSupportedFault (構造)
- SubnetAlreadyInUse (構造)
- SubscriptionAlreadyExistFault (構造)
- SubscriptionCategoryNotFoundFault (構造)
- SubscriptionNotFoundFault (構造)

AuthorizationAlreadyExistsFault (構造)

返された HTTP ステータスコード: 400。

指定された CIDRIP または EC2 セキュリティグループは、指定された DB セキュリティグループに対してすでに承認されています。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

AuthorizationNotFoundFault (構造)

返された HTTP ステータスコード: 404。

指定された CIDRIP または EC2 セキュリティグループは、指定された DB セキュリティグループに対して承認されていません。

Neptune は、ユーザーに代わって必要な行動を実行することを IAM 経由で承認することもできません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

AuthorizationQuotaExceededFault (構造)

返された HTTP ステータスコード: 400。

DB セキュリティグループの認可クォータに達しました。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

CertificateNotFoundFault (構造)

返された HTTP ステータスコード: 404。

CertificateIdentifier が既存の証明書を参照していません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

DBClusterAlreadyExistsFault (構造)

返された HTTP ステータスコード: 400。

ユーザーは指定された ID を持つ DB クラスターをすでに持っています。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

DBClusterNotFoundFault (構造)

返された HTTP ステータスコード: 404。

DBClusterIdentifier は、既存の DB クラスターを参照していません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

DBClusterParameterGroupNotFoundFault (構造)

返された HTTP ステータスコード: 404。

DBClusterParameterGroupName は、既存の DB クラスターパラメータグループを参照していません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

DBClusterQuotaExceededFault (構造)

返された HTTP ステータスコード: 403。

ユーザーが新しい DB クラスターを作成しようとしましたが、ユーザーはすでに最大許容 DB クラスタークォータに達しています。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

DBClusterRoleAlreadyExistsFault (構造)

返された HTTP ステータスコード: 400。

指定された IAM ロール Amazon リソースネーム (ARN) はすでに指定された DB クラスターに関連付けられています。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

DBClusterRoleNotFoundFault (構造)

返された HTTP ステータスコード: 404。

指定された IAM ロール Amazon リソースネーム (ARN) は、指定された DB クラスターに関連付けられていません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

DBClusterRoleQuotaExceededFault (構造)

返された HTTP ステータスコード: 400。

指定された DB クラスターに関連付けることができる IAM ロールの最大数を超えました。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

DBClusterSnapshotAlreadyExistsFault (構造)

返された HTTP ステータスコード: 400。

ユーザーは指定された ID を持つ DB クラススナップショットをすでに持っています。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

DBClusterSnapshotNotFoundFault (構造)

返された HTTP ステータスコード: 404。

DBClusterSnapshotIdentifier は、既存の DB クラススナップショットを参照していません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

DBInstanceAlreadyExistsFault (構造)

返された HTTP ステータスコード: 400。

ユーザーは指定された ID を持つ DB インスタンスをすでに持っています。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

DBInstanceNotFoundFault (構造)

返された HTTP ステータスコード: 404。

DBInstanceIdentifier は、既存の DB インスタンスを参照していません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

DBLogFileNotFoundFault (構造)

返された HTTP ステータスコード: 404。

LogFileName は、既存の DB ログファイルを参照していません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

DBParameterGroupAlreadyExistsFault (構造)

返された HTTP ステータスコード: 400。

同じ名前の DB パラメータグループが存在します。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

DBParameterGroupNotFoundFault (構造)

返された HTTP ステータスコード: 404。

DBParameterGroupName は、既存の DB パラメータグループを参照していません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

DBParameterGroupQuotaExceededFault (構造)

返された HTTP ステータスコード: 400。

リクエストにより、ユーザーは DB パラメータグループの許容数を超えます。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

DBSecurityGroupAlreadyExistsFault (構造)

返された HTTP ステータスコード: 400。

DBSecurityGroupName で指定された名前の DB セキュリティグループはすでに存在します。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

DBSecurityGroupNotFoundFault (構造)

返された HTTP ステータスコード: 404。

DBSecurityGroupName は、既存の DB セキュリティグループを参照していません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

DBSecurityGroupNotSupportedFault (構造)

返された HTTP ステータスコード: 400。

このアクションに DB セキュリティグループは許可されていません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

DBSecurityGroupQuotaExceededFault (構造)

返された HTTP ステータスコード: 400。

リクエストにより、ユーザーは DB セキュリティグループの許容数を超えます。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

DBSnapshotAlreadyExistsFault (構造)

返された HTTP ステータスコード: 400。

DBSnapshotIdentifier は、既存のスナップショットですでに使用されています。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

DBSnapshotNotFoundFault (構造)

返された HTTP ステータスコード: 404。

DBSnapshotIdentifier は、既存の DB スナップショットを参照していません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

DBSubnetGroupAlreadyExistsFault (構造)

返された HTTP ステータスコード: 400。

DBSubnetGroupName は、既存の DB サブネットグループによってすでに使用されています。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

DBSubnetGroupDoesNotCoverEnoughAZs (構造)

返された HTTP ステータスコード: 400。

アベイラビリティーゾーンが 1 つしかない場合を除き、DB サブネットグループ内のサブネットは少なくとも 2 つのアベイラビリティーゾーンをカバーする必要があります。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

問題の詳細を説明するメッセージ。

DBSubnetGroupNotAllowedFault (構造)

返された HTTP ステータスコード: 400。

ソースインスタンスと同じリージョンにあるリードレプリカを作成するときに DBSubnetGroup を指定しないでください。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

DBSubnetGroupNotFoundFault (構造)

返された HTTP ステータスコード: 404。

DBSubnetGroupName は、既存の DB サブネットグループを参照していません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

DBSubnetGroupQuotaExceededFault (構造)

返された HTTP ステータスコード: 400。

リクエストにより、ユーザーは DB サブネットグループの許容数を超えます。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

DBSubnetQuotaExceededFault (構造)

返された HTTP ステータスコード: 400。

リクエストにより、ユーザーは DB サブネットグループのサブネットの許容数を超えます。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

DBUpgradeDependencyFailureFault (構造)

返された HTTP ステータスコード: 400。

DB が依存しているリソースを変更できなかったため、DB のアップグレードに失敗しました。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

DomainNotFoundFault (構造)

返された HTTP ステータスコード: 404。

Domain は、既存の Active Directory ドメインを参照しません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

EventSubscriptionQuotaExceededFault (構造)

返された HTTP ステータスコード: 400。

サブスクライブできるイベントの数を超えました。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

GlobalClusterAlreadyExistsFault (構造)

返された HTTP ステータスコード: 400。

`GlobalClusterIdentifier` は既に存在します。新しいグローバルデータベース識別子 (一意の名前) を選択して、新しいグローバルデータベースクラスターを作成します。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

GlobalClusterNotFoundFault (構造)

返された HTTP ステータスコード: 404。

`GlobalClusterIdentifier` は既存のグローバルデータベースクラスターを参照していません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

GlobalClusterQuotaExceededFault (構造)

返された HTTP ステータスコード: 400。

このアカウントのグローバルデータベースクラスター数は、すでに許容最大数に達しています。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

問題の詳細を説明するメッセージ。

InstanceQuotaExceededFault (構造)

返された HTTP ステータスコード: 400。

リクエストにより、ユーザーは DB インスタンスの許容数を超えます。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

InsufficientDBClusterCapacityFault (構造)

返された HTTP ステータスコード: 403。

DB クラスターに現在のオペレーションに十分な容量がありません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

InsufficientDBInstanceCapacityFault (構造)

返された HTTP ステータスコード: 400。

指定された DB インスタンスクラスは、指定されたアベイラビリティーゾーンで利用できません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

InsufficientStorageClusterCapacityFault (構造)

返された HTTP ステータスコード: 400。

現在のアクションに使用可能なストレージが不十分です。使用可能なストレージがより多い別のアベイラビリティーゾーンを使用するようにサブネットグループを更新することで、このエラーを解決できる場合があります。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

InvalidDBClusterEndpointStateFault (構造)

返された HTTP ステータスコード: 400。

要求された操作は、エンドポイントがこの状態にある間は実行できません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

InvalidDBClusterSnapshotStateFault (構造)

返された HTTP ステータスコード: 400。

提供された値は有効な DB クラススナップショット状態ではありません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

InvalidDBClusterStateFault (構造)

返された HTTP ステータスコード: 400。

DB クラスターは有効な状態ではありません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

InvalidDBInstanceStateFault (構造)

返された HTTP ステータスコード: 400。

指定された DB インスタンスは利用可能な状態にありません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

InvalidDBParameterGroupStateFault (構造)

返された HTTP ステータスコード: 400。

DB パラメータグループが使用中または無効な状態です。パラメータグループを削除しようとしている場合、パラメータグループがこの状態のときは削除できません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

InvalidDBSecurityGroupStateFault (構造)

返された HTTP ステータスコード: 400。

DB セキュリティグループの状態では削除できません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

InvalidDBSnapshotStateFault (構造)

返された HTTP ステータスコード: 400。

DB スナップショットの状態では削除できません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

InvalidDBSubnetGroupFault (構造)

返された HTTP ステータスコード: 400。

DBSubnetGroup が、同じソースインスタンスの既存のクロスリージョンリードレプリカと同じ VPC に属していないことを示します。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

InvalidDBSubnetGroupStateFault (構造)

返された HTTP ステータスコード: 400。

DB サブネットグループは使用中のため削除できません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

問題の詳細を説明するメッセージ。

InvalidDBSubnetStateFault (構造)

返された HTTP ステータスコード: 400。

DB サブネットは利用可能な状態にありません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

InvalidEventSubscriptionStateFault (構造)

返された HTTP ステータスコード: 400。

イベントのサブスクリプションは無効な状態です。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

InvalidGlobalClusterStateFault (構造)

返された HTTP ステータスコード: 400。

グローバルクラスターは無効な状態であり、要求された操作を実行できません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

InvalidOptionGroupStateFault (構造)

返された HTTP ステータスコード: 400。

オプショングループは利用可能な状態にありません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

InvalidRestoreFault (構造)

返された HTTP ステータスコード: 400。

VPC バックアップから非 VPC DB インスタンスに復元できません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

InvalidSubnet (構造)

返された HTTP ステータスコード: 400。

リクエストされたサブネットが無効であるか、またはすべてが共通の VPC に含まれていない複数のサブネットがリクエストされました。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

InvalidVPCNetworkStateFault (構造)

返された HTTP ステータスコード: 400。

ユーザーが変更されたため、作成後に DB サブネットグループがすべてのアベイラビリティーゾーンをカバーするわけではありません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

KMSKeyNotAccessibleFault (構造)

返された HTTP ステータスコード: 400。

KMS キーへのアクセス中にエラーが発生しました。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

OptionGroupNotFoundFault (構造)

返された HTTP ステータスコード: 404。

指定したオプショングループが見つかりませんでした。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

PointInTimeRestoreNotEnabledFault (構造)

返された HTTP ステータスコード: 400。

SourceDBInstanceIdentifier は、BackupRetentionPeriod が 0 の DB インスタンスを指します。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

ProvisionedIopsNotAvailableInAZFault (構造)

返された HTTP ステータスコード: 400。

プロビジョンド IOPS は、指定されたアベイラビリティーゾーンでは使用できません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

ResourceNotFoundFault (構造)

返された HTTP ステータスコード: 404。

指定されたリソース ID は見つかりませんでした。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

SNSInvalidTopicFault (構造)

返された HTTP ステータスコード: 400。

SNS トピックが無効です。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

SNSNoAuthorizationFault (構造)

返された HTTP ステータスコード: 400。

SNS 認可はありません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

SNSTopicArnNotFoundFault (構造)

返された HTTP ステータスコード: 404。

SNS トピックの ARN が見つかりませんでした。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

SharedSnapshotQuotaExceededFault (構造)

返された HTTP ステータスコード: 400。

手動 DB スナップショットを共有できるアカウントの最大数を超えました。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

SnapshotQuotaExceededFault (構造)

返された HTTP ステータスコード: 400。

リクエストにより、ユーザーは DB スナップショットの許容数を超えます。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

SourceNotFoundFault (構造)

返された HTTP ステータスコード: 404。

ソースが見つかりませんでした。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

StorageQuotaExceededFault (構造)

返された HTTP ステータスコード: 400。

リクエストにより、ユーザーはすべての DB インスタンスで使用可能な許容ストレージ容量を超えます。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

StorageTypeNotSupportedFault (構造)

返された HTTP ステータスコード: 400。

指定された StorageType を DB インスタンスに関連付けることができません。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

問題の詳細を説明するメッセージ。

SubnetAlreadyInUse (構造)

返された HTTP ステータスコード: 400。

DB サブネットはアベイラビリティーゾーンにすでに使われています。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

SubscriptionAlreadyExistFault (構造)

返された HTTP ステータスコード: 400。

このサブスクリプションはすでに存在します。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

SubscriptionCategoryNotFoundFault (構造)

返された HTTP ステータスコード: 404。

指定されたサブスクリプションカテゴリが見つかりませんでした。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

SubscriptionNotFoundFault (構造)

返された HTTP ステータスコード: 404。

指定されたサブスクリプションが見つかりませんでした。

フィールド

- message — ExceptionMessage、タイプ: `string` (UTF-8 でエンコードされた文字列)。

  問題の詳細を説明するメッセージ。

Amazon Neptune データ API リファレンス

この章では、Neptune グラフでのデータの読み込み、アクセス、管理に使用できる Neptune データ API 操作について説明します。

Neptune データ API は、データの読み込み、クエリの実行、データに関する情報の取得、機械学習操作の実行のための SDK サポートを提供します。Neptune の Gremlin クエリ言語と openCypher クエリ言語をサポートしており、すべての SDK 言語で使用できます。API リクエストに自動的に署名し、Neptune のアプリケーションへの統合を大幅に簡素化します。

目次

- CancelledByUserException (構造)
- ClientTimeoutException (構造)
- ConcurrentModificationException (構造)
- ConstraintViolationException (構造)
- ExpiredStreamException (構造)
- FailureByQueryException (構造)
- IllegalArgumentException (構造)
- InternalFailureException (構造)
- InvalidArgumentException (構造)
- InvalidNumericDataException (構造)
- InvalidParameterException (構造)
- LoadUrlAccessDeniedException (構造)
- MalformedQueryException (構造)
- MemoryLimitExceededException (構造)
- MethodNotAllowedException (構造)
- MissingParameterException (構造)
- MLResourceNotFoundException (構造)
- ParsingException (構造)
- PreconditionsFailedException (構造)
- QueryLimitExceededException (構造)
- QueryLimitException (構造)
- QueryTooLargeException (構造)
- ReadOnlyViolationException (構造)
- S3Exception (構造)
- ServerShutdownException (構造)
- StatisticsNotAvailableException (構造)
- StreamRecordsNotFoundException (構造)
- ThrottlingException (構造)
- TimeLimitExceededException (構造)
- TooManyRequestsException (構造)

- UnsupportedOperationException (構造)
- UnloadUrlAccessDeniedException (構造)

Neptune データプレーンエンジン、高速リセット、および一般構造 API

エンジン操作:

- GetEngineStatus (アクション)
- ExecuteFastReset (アクション)

エンジン操作構造:

- QueryLanguageVersion (構造)
- FastResetToken (構造)

GetEngineStatus (アクション)

この API の AWS CLI 名は `get-engine-status` です。

ホストでグラフデータベースのステータスを取得します。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:GetEngineStatus IAM アクションを許可するポリシーがアタッチされている必要があります。

リクエスト

- リクエストパラメータなし

レスポンス

- dbEngineVersion — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DB クラスターで実行されている Neptune エンジンのバージョンに設定します。このバージョンのエンジンがリリースされた後に手動でパッチが適用された場合は、バージョン番号の先頭に `Patch-` が付加されます。

- dfeQueryEngine — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  DFE エンジンが完全に有効になっている場合は `enabled` に設定し、`useDFE` クエリヒントが `true` に設定されているクエリでのみ DFE エンジンを使用する場合は `viaQueryHint` (デフォルト) に設定します。

- features – キと値のペアのマップ配列。

  各キーは、タイプ `string` の文字列 (UTF-8 でエンコードされた文字列) です。

  各値は、タイプ `document` のドキュメントです (JSON のようなデータモデルで表される、プロトコルに依存しないオープンコンテンツ)。

  DB クラスターで有効になっている機能に関するステータス情報を含みます。

- gremlin – QueryLanguageVersion オブジェクト。

  クラスターで使用できる Gremlin クエリ言語に関する情報を含みます。具体的には、エンジンが使用している現在の TinkerPop バージョンを指定するバージョンフィールドを含みます。

- labMode – キと値のペアのマップ配列。

  各キーは、タイプ `string` の文字列 (UTF-8 でエンコードされた文字列) です。

  各値は、タイプ `string` の文字列 (UTF-8 でエンコードされた文字列) です。

  エンジンが使用しているラボモード設定を含みます。

- opencypher – QueryLanguageVersion オブジェクト。

  クラスターで使用できる openCypher クエリ言語に関する情報を含みます。具体的には、エンジンが使用している現在の opencypher バージョンを指定するバージョンフィールドを含みます。

- role — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  インスタンスがリードレプリカの場合は `reader` に設定し、インスタンスがプライマリインスタンスの場合は `writer` に設定します。

- rollingBackTrxCount - タイプ `integer` の整数 (符号付き 32 ビット整数)。

  ロールバックされるトランザクションがある場合、このフィールドは、そのようなトランザクションの数に設定されます。コメントがない場合、このフィールドは一切表示されません。

- rollingBackTrxEarliestStartTime — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

ロールバックされる最も早いトランザクションの開始時刻に設定します。ロールバックされるトランザクションがない場合、このフィールドは一切表示されません。

- settings – キと値のペアのマップ配列。

  各キーは、タイプ `string` の文字列 (UTF-8 でエンコードされた文字列) です。

  各値は、タイプ `string` の文字列 (UTF-8 でエンコードされた文字列) です。

  DB クラスターの現在の設定に関する情報を含みます。例えば、現在のクラスタークエリのタイムアウト設定 (`clusterQueryTimeoutInMs`) を含みます。

- sparql – QueryLanguageVersion オブジェクト。

  クラスターで使用できる SPARQL クエリ言語に関する情報を含みます。具体的には、エンジンが使用している現在の SPARQL バージョンを指定するバージョンフィールドを含みます。

- startTime — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  現在のサーバープロセスが開始した UTC 時間に設定します。

- status — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  インスタンスに問題が発生していない場合、`healthy` に設定します。インスタンスがクラッシュまたは再起動から回復中で、最新のサーバーのシャットダウンからアクティブなトランザクションが実行されている場合、ステータスは `recovery` に設定されます。

エラー

- UnsupportedOperationException
- InternalFailureException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- IllegalArgumentException
- TooManyRequestsException

ExecuteFastReset (アクション)

この API の AWS CLI 名は `execute-fast-reset` です。

高速リセット REST API を使用すると、Neptune グラフをすばやく簡単にリセットし、すべてのデータを削除できます。

Neptune 高速リセットは、2 ステップのプロセスです。まず、`action` を `initiateDatabaseReset` に設定して `ExecuteFastReset` を呼び出します。これにより UUID トークンが返され、`action` を `performDatabaseReset` に設定して再び `ExecuteFastReset` を呼び出すときに、このトークンを含めます。「高速リセット API を使用して Amazon Neptune DB クラスターを空にする」を参照してください。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:ResetDatabase IAM アクションを許可するポリシーがアタッチされている必要があります。

リクエスト

- action (CLI では: `--action`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  高速リセットアクション。次のいずれかの値になります。

  - **`initiateDatabaseReset`** — このアクションは、高速リセットを実際に実行するのに必要な一意のトークンを生成します。
  - **`performDatabaseReset`** — このアクションは、`initiateDatabaseReset` アクションによって生成されたトークンを使用して、実際に高速リセットを実行します。

- token (CLI では: `--token`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  リセットを開始するための高速リセットトークン。

レスポンス

- payload – FastResetToken オブジェクト。

  `payload` は `initiateDatabaseReset` アクションによってのみ返され、リセットを行うために `performDatabaseReset` アクションで使用する一意のトークンを含みます。
- status - 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

`status` は `performDatabaseReset` アクションに対してのみ返され、高速リセットリクエストが受け入れられたかどうかを示します。

エラー

- InvalidParameterException
- ClientTimeoutException
- AccessDeniedException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- ServerShutdownException
- PreconditionsFailedException
- MethodNotAllowedException
- ReadOnlyViolationException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException

エンジン操作構造:

QueryLanguageVersion (構造)

クエリ言語バージョンを表す構造。

フィールド

- version - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  クエリ言語のバージョン。

FastResetToken (構造)

高速リセットを開始するために使用される高速リセットトークンを含む構造体。

フィールド

- token — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  `initiateDatabaseReset` アクションでデータベースによって生成され、`performDatabaseReset` がデータベースをリセットするために使用する UUID。

Neptune クエリ API

Gremlin クエリアクション:

- ExecuteGremlinQuery (アクション)
- ExecuteGremlinExplainQuery (アクション)
- ExecuteGremlinProfileQuery (アクション)
- ListGremlinQueries (アクション)
- GetGremlinQueryStatus (アクション)
- CancelGremlinQuery (アクション)

openCypher クエリアクション:

- ExecuteOpenCypherQuery (アクション)
- ExecuteOpenCypherExplainQuery (アクション)
- ListOpenCypherQueries (アクション)
- GetOpenCypherQueryStatus (アクション)
- CancelOpenCypherQuery (アクション)

クエリ構造:

- QueryEvalStats (構造)
- GremlinQueryStatus (構造)
- GremlinQueryStatusAttributes (構造)

ExecuteGremlinQuery (アクション)

この API の AWS CLI 名は `execute-gremlin-query` です。

このコマンドは Gremlin クエリを実行します。Amazon Neptune は Apache TinkerPop3 および Gremlin と互換性があるため、Apache TinkerPop3 ドキュメントの「グラフ」で述べられているように、Gremlin トラバーサル言語を使用してグラフにクエリを実行できます。詳細は、「Gremlin による Neptune グラフへのアクセス」にも記載されています。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、クエリに応じて、そのクラスターで以下の IAM アクションを許可するポリシーがアタッチされている必要があります。

- neptune-db:ReadDataViaQuery
- neptune-db:WriteDataViaQuery
- neptune-db:DeleteDataViaQuery

neptune-db:QueryLanguage:Gremlin IAM 条件キーをポリシードキュメントで使用して、Gremlin クエリの使用を制限できることに注意してください (「Neptune IAM データアクセスポリシーステートメントで利用可能な条件キー」を参照)。

リクエスト

- gremlinQuery (CLI では: `--gremlin-query`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  この API を使用すると、HTTP エンドポイントを使用する場合と同様に、Gremlin クエリを文字列形式で実行できます。このインターフェイスは、DB クラスターが使用しているすべての Gremlin バージョンと互換性があります (エンジンバージョンがサポートしている Gremlin リリースを特定するには、「Tinkerpop クライアントセクション」を参照してください)。
- serializer (CLI では: `--serializer`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  null 以外の場合、クエリ結果は、このパラメータで指定された形式でシリアル化されたレスポンスメッセージで返されます。現在サポートされているフォーマットのリストについては、TinkerPop ドキュメントの「GraphSON」セクションを参照してください。

レスポンス

- meta - タイプ `document` のドキュメント (JSON のようなデータモデルで表される、プロトコルに依存しないオープンコンテンツ)。

Gremlin クエリに関するメタデータ。

- requestId — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  Gremlin クエリの一意識別子。

- result - タイプ `document` のドキュメント (JSON のようなデータモデルで表される、プロトコルに依存しないオープンコンテンツ)。

  サーバーからの Gremlin クエリ出力。

- status – GremlinQueryStatusAttributes オブジェクト。

  Gremlin クエリのステータス。

エラー

- QueryTooLargeException
- BadRequestException
- QueryLimitExceededException
- InvalidParameterException
- QueryLimitException
- ClientTimeoutException
- CancelledByUserException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- FailureByQueryException
- MemoryLimitExceededException
- PreconditionsFailedException
- MalformedQueryException
- ParsingException
- ConstraintViolationException
- TimeLimitExceededException

- InvalidArgumentException
- MissingParameterException
- ConcurrentModificationException

ExecuteGremlinExplainQuery (アクション)

この API の AWS CLI 名は `execute-gremlin-explain-query` です。

Gremlin 説明クエリを実行します。

Amazon Neptune は、`explain` という名前の Gremlin 機能を追加しました。これは、Neptune エンジンがクエリのために使用する実行アプローチを理解するために役立つセルフサービスツールです。この機能を呼び出すには、Gremlin クエリを送信する HTTP コールに `explain` パラメータを追加します。

説明機能は、クエリ実行プランの論理構造に関する情報を提供します。この情報を使用して潜在的な評価と実行障害を明らかにし、「Gremlin クエリの調整」で説明されているように、クエリを調整できます。また、クエリに関するヒントを使用して、クエリ実行プランを改善できます。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、クエリに応じて、そのクラスターで以下の IAM アクションを許可するポリシーがアタッチされている必要があります。

- neptune-db:ReadDataViaQuery
- neptune-db:WriteDataViaQuery
- neptune-db:DeleteDataViaQuery

neptune-db:QueryLanguage:Gremlin IAM 条件キーをポリシードキュメントで使用して、Gremlin クエリの使用を制限できることに注意してください (「Neptune IAM データアクセスポリシーステートメントで利用可能な条件キー」を参照)。

リクエスト

- gremlinQuery (CLI では: `--gremlin-query`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  Gremlin の説明クエリ文字列。

レスポンス

- output — タイプ `blob` の ReportAsText (未解釈のバイナリデータのブロック)。

  「Gremlin クエリの調整」で説明されているように、Gremlin の説明結果を含むテキストブロブ。

エラー

- QueryTooLargeException
- BadRequestException
- QueryLimitExceededException
- InvalidParameterException
- QueryLimitException
- ClientTimeoutException
- CancelledByUserException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- FailureByQueryException
- MemoryLimitExceededException
- PreconditionsFailedException
- MalformedQueryException
- ParsingException
- ConstraintViolationException
- TimeLimitExceededException
- InvalidArgumentException
- MissingParameterException
- ConcurrentModificationException

ExecuteGremlinProfileQuery (アクション)

この API の AWS CLI 名は `execute-gremlin-profile-query` です。

Gremlin プロファイルクエリを実行します。これは、指定されたトラバーサルを実行し、実行に関するさまざまなメトリクスを収集して、出力としてプロファイルレポートを生成します。詳細については、「Neptune での Gremlin プロファイル API」を参照してください。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:ReadDataViaQuery IAM アクションを許可するポリシーがアタッチされている必要があります。

neptune-db:QueryLanguage:Gremlin IAM 条件キーをポリシードキュメントで使用して、Gremlin クエリの使用を制限できることに注意してください (「Neptune IAM データアクセスポリシーステートメントで利用可能な条件キー」を参照)。

リクエスト

- chop (CLI では: `--chop`) — タイプ `integer` の整数 (符号付き 32 ビット整数)。

  ゼロ以外の場合、結果の文字列はその文字数で切り捨てられます。ゼロに設定すると、文字列にはすべての結果が含まれます。
- gremlinQuery (CLI では: `--gremlin-query`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  プロファイリングする Gremlin クエリ文字列。
- indexOps (CLI では: `--index-ops`) — タイプ `boolean` のブール値 (ブール値 (真または偽) の値)。

  このフラグが `TRUE` に設定された場合、結果には、クエリの実行およびシリアル化中に行われたすべてのインデックス操作の詳細レポートが含まれます。
- results (CLI では: `--results`) — タイプ `boolean` のブール値 (ブール値 (真または偽) の値)。

  このフラグが `TRUE` に設定された場合、クエリ結果が収集され、プロファイルレポートの一部として表示されます。`FALSE` の場合、結果カウントのみが表示されます。
- serializer (CLI では: `--serializer`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  null 以外の場合、収集された結果は、このパラメータで指定された形式でシリアル化されたレスポンスメッセージで返されます。詳細については、「Neptune での Gremlin プロファイル API」を参照してください。

レスポンス

- output — タイプ `blob` の ReportAsText (未解釈のバイナリデータのブロック)。

  Gremlin プロファイルの結果を含むテキストブロブ。詳細については、「Neptune での Gremlin プロファイル API」を参照してください。

エラー

- QueryTooLargeException
- BadRequestException
- QueryLimitExceededException
- InvalidParameterException
- QueryLimitException
- ClientTimeoutException
- CancelledByUserException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- FailureByQueryException
- MemoryLimitExceededException
- PreconditionsFailedException
- MalformedQueryException
- ParsingException
- ConstraintViolationException
- TimeLimitExceededException
- InvalidArgumentException
- MissingParameterException
- ConcurrentModificationException

ListGremlinQueries (アクション)

この API の AWS CLI 名は `list-gremlin-queries` です。

アクティブな Gremlin クエリを一覧表示します。出力の詳細については、「Gremlin クエリステータス API」を参照してください。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:GetQueryStatus IAM アクションを許可するポリシーがアタッチされている必要があります。

neptune-db:QueryLanguage:Gremlin IAM 条件キーをポリシードキュメントで使用して、Gremlin クエリの使用を制限できることに注意してください (「Neptune IAM データアクセスポリシーステートメントで利用可能な条件キー」を参照)。

リクエスト

- includeWaiting (CLI では: `--include-waiting`) — タイプ `boolean` のブール値 (ブール値 (真または偽) の値)。

  `TRUE` に設定された場合、返されるリストには待機中のクエリが含まれます。デフォルトは `FALSE` です。

レスポンス

- acceptedQueryCount - タイプ `integer` の整数 (符号付き 32 ビット整数)。

  受け入れられながらもまだ完了していないクエリの数 (キュー内のクエリを含む)。
- queries – GremlinQueryStatus オブジェクトの配列。

  現在の SPARQL クエリのリスト。
- runningQueryCount - タイプ `integer` の整数 (符号付き 32 ビット整数)。

  現在実行されている Gremlin クエリの数。

エラー

- BadRequestException
- InvalidParameterException
- ClientTimeoutException
- AccessDeniedException
- IllegalArgumentException

- TooManyRequestsException
- UnsupportedOperationException
- FailureByQueryException
- PreconditionsFailedException
- ParsingException
- ReadOnlyViolationException
- ConstraintViolationException
- TimeLimitExceededException
- InvalidArgumentException
- MissingParameterException
- ConcurrentModificationException

GetGremlinQueryStatus (アクション)

この API の AWS CLI 名は `get-gremlin-query-status` です。

指定された Gremlin クエリのステータスを取得します。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:GetQueryStatus IAM アクションを許可するポリシーがアタッチされている必要があります。

neptune-db:QueryLanguage:Gremlin IAM 条件キーをポリシードキュメントで使用して、Gremlin クエリの使用を制限できることに注意してください (「Neptune IAM データアクセスポリシーステートメントで利用可能な条件キー」を参照)。

リクエスト

- queryId (CLI では: `--query-id`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  Gremlin クエリを識別する一意識別子。

レスポンス

- queryEvalStats – QueryEvalStats オブジェクト。

Gremlin クエリの評価ステータス。

- queryId — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  ステータスが返されるクエリの ID。

- queryString — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  Gremlin クエリ文字列。

エラー

- BadRequestException
- InvalidParameterException
- ClientTimeoutException
- AccessDeniedException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- FailureByQueryException
- PreconditionsFailedException
- ParsingException
- ReadOnlyViolationException
- ConstraintViolationException
- TimeLimitExceededException
- InvalidArgumentException
- MissingParameterException
- ConcurrentModificationException

CancelGremlinQuery (アクション)

この API の AWS CLI 名は `cancel-gremlin-query` です。

Gremlin クエリをキャンセルします。詳細については、「Gremlin クエリのキャンセル」を参照してください。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:CancelQuery IAM アクションを許可するポリシーがアタッチされている必要があります。

リクエスト

- queryId (CLI では: `--query-id`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  キャンセルされるクエリを識別する一意識別子。

レスポンス

- status — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  キャンセルのステータス。

エラー

- BadRequestException
- InvalidParameterException
- ClientTimeoutException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- FailureByQueryException
- PreconditionsFailedException
- ParsingException
- ConstraintViolationException
- TimeLimitExceededException
- InvalidArgumentException
- MissingParameterException
- ConcurrentModificationException

openCypher クエリアクション:

ExecuteOpenCypherQuery (アクション)

この API の AWS CLI 名は `execute-open-cypher-query` です。

openCypher クエリを実行します。詳細については、「openCypher による Neptune グラフへのアクセス」を参照してください。

Neptune は、openCypher を使用したグラフアプリケーションの構築をサポートしています。これは、現在、グラフデータベースを操作する開発者にとって最も人気のあるクエリ言語の 1 つです。開発者、ビジネスアナリスト、データサイエンティストは、openCypher の SQL の影響が大きい宣言型構文を好みます。これは、プロパティグラフのクエリによく使用される構造を備えているためです。

openCypher 言語は、当初は Neo4J によって開発され、その後、2015 年にオープンソース化され、Apache 2 オープンソースライセンスの下で openCypher プロジェクトに貢献しました。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、クエリに応じて、そのクラスターで以下の IAM アクションを許可するポリシーがアタッチされている必要があります。

- neptune-db:ReadDataViaQuery
- neptune-db:WriteDataViaQuery
- neptune-db:DeleteDataViaQuery

また、neptune-db:QueryLanguage:OpenCypher IAM 条件キーをポリシードキュメントで使用して、openCypher クエリの使用を制限できることにも注意してください (「Neptune IAM データアクセスポリシーステートメントで利用可能な条件キー」を参照)。

リクエスト

- openCypherQuery (CLI では: `--open-cypher-query`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  実行される openCypher クエリ文字列。
- parameters (CLI では: `--parameters`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

クエリを実行するための openCypher クエリパラメータ。詳細については、「openCypher パラメーター化クエリの例」を参照してください。

レスポンス

- results - 必須: タイプ document のドキュメント (JSON のようなデータモデルで表される、プロトコルに依存しないオープンコンテンツ)。

  openCypher クエリ結果。

エラー

- QueryTooLargeException
- InvalidNumericDataException
- BadRequestException
- QueryLimitExceededException
- InvalidParameterException
- QueryLimitException
- ClientTimeoutException
- CancelledByUserException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- FailureByQueryException
- MemoryLimitExceededException
- PreconditionsFailedException
- MalformedQueryException
- ParsingException
- ConstraintViolationException
- TimeLimitExceededException
- InvalidArgumentException

- MissingParameterException
- ConcurrentModificationException

ExecuteOpenCypherExplainQuery (アクション)

この API の AWS CLI 名は `execute-open-cypher-explain-query` です。

openCypher `explain` リクエストを実行します。詳細については、[openCypher の説明機能」を参照してください。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:ReadDataViaQuery IAM アクションを許可するポリシーがアタッチされている必要があります。

また、neptune-db:QueryLanguage:OpenCypher IAM 条件キーをポリシードキュメントで使用して、openCypher クエリの使用を制限できることにも注意してください (「Neptune IAM データアクセスポリシーステートメントで利用可能な条件キー」を参照)。

リクエスト

- explainMode (CLI では: `--explain-mode`) — 必須: `string` タイプの OpenCypherExplainMode (UTF-8 でエンコードされた文字列)。

  openCypher `explain` モード。`static`、`dynamic`、または `details` のいずれか。
- openCypherQuery (CLI では: `--open-cypher-query`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  openCypher クエリ文字列。
- parameters (CLI では: `--parameters`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  openCypher クエリパラメータ。

レスポンス

- results — 必須: タイプ `blob` のブロブ (未解釈のバイナリデータのブロック)。

  openCypher `explain` の結果を含むテキストブロブ。

エラー

- QueryTooLargeException
- InvalidNumericDataException
- BadRequestException
- QueryLimitExceededException
- InvalidParameterException
- QueryLimitException
- ClientTimeoutException
- CancelledByUserException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- FailureByQueryException
- MemoryLimitExceededException
- PreconditionsFailedException
- MalformedQueryException
- ParsingException
- ConstraintViolationException
- TimeLimitExceededException
- InvalidArgumentException
- MissingParameterException
- ConcurrentModificationException

ListOpenCypherQueries (アクション)

この API の AWS CLI 名は `list-open-cypher-queries` です。

アクティブな openCypher クエリを一覧表示します。詳細については、「Neptune openCypher ステータスエンドポイント」を参照してください。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:GetQueryStatus IAM アクションを許可するポリシーがアタッチされている必要があります。

また、neptune-db:QueryLanguage:OpenCypher IAM 条件キーをポリシードキュメントで使用して、openCypher クエリの使用を制限できることにも注意してください (「Neptune IAM データアクセスポリシーステートメントで利用可能な条件キー」を参照)。

リクエスト

- includeWaiting (CLI では: `--include-waiting`) — タイプ `boolean` のブール値 (ブール値 (真または偽) の値)。

  TRUE に設定され、他のパラメータが存在しないときには、待機中のクエリと実行中のクエリのステータス情報が返されます。

レスポンス

- acceptedQueryCount - タイプ `integer` の整数 (符号付き 32 ビット整数)。

  受け入れられながらもまだ完了していないクエリの数 (キュー内のクエリを含む)。
- queries – GremlinQueryStatus オブジェクトの配列。

  現在の openCypher クエリのリスト。
- runningQueryCount - タイプ `integer` の整数 (符号付き 32 ビット整数)。

  現在実行中の openCypher クエリの数。

エラー

- InvalidNumericDataException
- BadRequestException
- InvalidParameterException
- ClientTimeoutException
- AccessDeniedException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- FailureByQueryException
- PreconditionsFailedException

- ParsingException
- ReadOnlyViolationException
- ConstraintViolationException
- TimeLimitExceededException
- InvalidArgumentException
- MissingParameterException
- ConcurrentModificationException

GetOpenCypherQueryStatus (アクション)

この API の AWS CLI 名は `get-open-cypher-query-status` です。

指定された openCypher クエリのステータスを取得します。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:GetQueryStatus IAM アクションを許可するポリシーがアタッチされている必要があります。

また、neptune-db:QueryLanguage:OpenCypher IAM 条件キーをポリシードキュメントで使用して、openCypher クエリの使用を制限できることにも注意してください (「Neptune IAM データアクセスポリシーステートメントで利用可能な条件キー」を参照)。

リクエスト

- queryId (CLI では: `--query-id`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  クエリのステータスを取得する openCypher クエリの一意 ID。

レスポンス

- queryEvalStats – QueryEvalStats オブジェクト。

  openCypher クエリの評価ステータス。
- queryId — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  ステータスが返されるクエリの一意 ID。
- queryString — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

openCypher クエリ文字列。

エラー

- InvalidNumericDataException
- BadRequestException
- InvalidParameterException
- ClientTimeoutException
- AccessDeniedException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- FailureByQueryException
- PreconditionsFailedException
- ParsingException
- ReadOnlyViolationException
- ConstraintViolationException
- TimeLimitExceededException
- InvalidArgumentException
- MissingParameterException
- ConcurrentModificationException

CancelOpenCypherQuery (アクション)

この API の AWS CLI 名は `cancel-open-cypher-query` です。

指定された openCypher クエリをキャンセルします。詳細については、「Neptune openCypher ステータスエンドポイント」を参照してください。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:CancelQuery IAM アクションを許可するポリシーがアタッチされている必要があります。

リクエスト

- queryId (CLI では: `--query-id`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  キャンセルする openCypher クエリの一意 ID。
- silent (CLI では: `--silent`) — タイプ `boolean` のブール値 (ブール値 (真または偽) の値)。

  `TRUE` に設定された場合、openCypher クエリのキャンセルは予告なしに行われます。

レスポンス

- payload - タイプ `boolean` のブール値 (ブール値 (真または偽))。

  openCypher クエリのキャンセルペイロード。
- status — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  openCypher クエリのキャンセルステータス。

エラー

- InvalidNumericDataException
- BadRequestException
- InvalidParameterException
- ClientTimeoutException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- FailureByQueryException
- PreconditionsFailedException
- ParsingException
- ConstraintViolationException
- TimeLimitExceededException
- InvalidArgumentException
- MissingParameterException
- ConcurrentModificationException

クエリ構造:

QueryEvalStats (構造)

実行中、承認済み、または待機中のクエリの数とその詳細など、クエリ統計をキャプチャする構造。

フィールド

- cancelled - これはタイプ `boolean` のブール値です (ブール値 (真または偽))。

  クエリがキャンセルされた場合は TRUE に設定され、それ以外の場合は FALSE に設定されます。
- elapsed - タイプ `integer` の整数 (符号付き 32 ビット整数)。

  これまでクエリが実行されていた時間 (マイクロ秒)。
- subqueries - タイプ `document` のドキュメント (JSON のようなデータモデルで表される、プロトコルに依存しないオープンコンテンツ)。

  このクエリのサブクエリの数。
- waited - タイプ `integer` の整数 (符号付き 32 ビット整数)。

  クエリが待機していた時間をミリ秒単位で示します。

GremlinQueryStatus (構造)

Gremlin クエリのステータスをキャプチャします (「Gremlin クエリステータス API」ページを参照)。

フィールド

- queryEvalStats - これは QueryEvalStats オブジェクトです。

  Gremlin クエリのクエリ統計。
- queryId — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  Gremlin クエリの ID。
- queryString — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  Gremlin クエリのクエリ文字列。

GremlinQueryStatusAttributes (構造)

Gremlin クエリのステータスコンポーネントを含みます。

フィールド

- attributes - タイプ `document` のドキュメント (JSON のようなデータモデルで表される、プロトコルに依存しないオープンコンテンツ)。

  Gremlin クエリステータスの属性。
- code - タイプ `integer` の整数 (符号付き 32 ビット整数)。

  Gremlin クエリリクエストから返された HTTP レスポンスコード。
- message — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  ステータスメッセージ。

Neptune データプレーンバルクローダー API

一括ロードアクション:

- StartLoaderJob (アクション)
- GetLoaderJobStatus (アクション)
- ListLoaderJobs (アクション)
- CancelLoaderJob (アクション)

一括ロード構造:

- LoaderIdResult (構造)

StartLoaderJob (アクション)

> この API の AWS CLI 名は `start-loader-job` です。

Neptune バルクローダージョブを開始して、Amazon S3 バケットから Neptune DB インスタンスにデータをロードします。「Amazon Neptune バルクローダーを使用したデータの取り込み」を参照してください。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:StartLoaderJob IAM アクションを許可するポリシーがアタッチされている必要があります。

リクエスト

- dependencies (CLI では: `--dependencies`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  これは、キュー内の 1 つ以上の前のジョブが正常に完了することを条件に、キューに入れられたロードリクエストを作成することができるオプションのパラメータです。

  Neptune は、`queueRequest` パラメータが `"TRUE"` に設定されている場合、一度に最大 64 個のロードリクエストをキューに入れることができます。`dependencies` パラメータを使用すると、キュー内で指定された 1 つ以上前のリクエストが正常に完了したかどうかに応じて、キューに入れられたそのようなリクエストを実行できます。

  たとえば、ロード `Job-A` と `Job-B` が互いに独立しているものの、ロード `Job-C` を開始する前に `Job-A` および `Job-B` を終了する必要がある場合は、次の手順を実行します。

  1. 任意の順序で `load-job-A` と `load-job-B` を 1 つずつ送信し、load-id を保存します。
  2. `dependencies` フィールドで 2 つのジョブの load-id を付けて `load-job-C` を送信します。

  Example

  ```
  "dependencies" : ["(job_A_load_id)", "(job_B_load_id)"]
  ```

  `dependencies` パラメータのため、`Job-A` と `Job-B` が正常に完了するまで、バルクローダーは `Job-C` を起動しません。いずれかが失敗すると、Job-C は実行されず、そのステータスは `LOAD_FAILED_BECAUSE_DEPENDENCY_NOT_SATISFIED` に設定されます。

  この方法で複数のレベルの依存関係を設定できます。これにより、1 つのジョブが失敗すると、直接的または間接的に依存するすべてのリクエストがキャンセルされます。

- failOnError (CLI では: `--fail-on-error`) — タイプ `boolean` のブール値 (ブール値 (真または偽) の値)。

  `failOnError` - エラー時に完全な停止を切り替えるフラグ。

  使用できる値: `"TRUE"`、`"FALSE"`。

デフォルト値: "TRUE"。

このパラメータを "FALSE" に設定すると、ローダーは指定された場所のすべてのデータをロードし、エラーのあるエントリはスキップします。

このパラメータを "TRUE" に設定すると、ローダーはエラー発生時にすぐに停止します。その時点までロードされたデータは保持されます。

- format (CLI では: `--format`) — 必須: `string` タイプの形式 (UTF-8 でエンコードされた文字列)。

  データの形式です。Neptune Loader コマンドのデータ形式の詳細については、「データ読み込み形式」を参照してください。

  許可される値

  - **`csv`** Gremlin CSV データ形式用。
  - **`opencypher`** openCypher CSV データ形式用。
  - **`ntriples`** N-Triples RDF データ形式用。
  - **`nquads`** N-Quads RDF データ形式用。
  - **`rdfxml`** RDF\XML RDF データ形式用。
  - **`turtle`** Turtle RDF データ形式用。
- iamRoleArn (CLI では: `--iam-role-arn`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  S3 バケットにアクセスするために、Neptune DB インスタンスによって想定される IAM ロールの Amazon リソースネーム (ARN)。ここで指定した IAM ロール ARN は DB クラスターにアタッチする必要があります (「Amazon Neptune クラスターへの IAM ロールの追加」を参照)。
- mode (CLI では: `--mode`) — タイプ `string` のモード (UTF-8 でエンコードされた文字列)。

  ロードジョブモード。

  使用できる値: `RESUME`、`NEW`、`AUTO`。

  デフォルト値: `AUTO`。

  - `RESUME` - RESUME モードでは、ローダーはこのソースからの以前のロードを検索し、見つかった場合は、そのロードジョブを再開します。以前のロードジョブが見つからない場合、ローダーは停止します。

ローダーは、以前のジョブで正常にロードされたファイルの再ロードを回避します。失敗したファイルの処理のみを試行します。以前にロードしたデータを Neptune クラスターから削除している場合、そのデータはこのモードでは再ロードされません。以前のロードジョブが同じソースからすべてのファイルを正常にロードした場合、何も再ロードされず、ローダーは成功を返します。

- NEW - NEW モードでは、以前のロードに関係なく、新しいロードリクエストが作成されます。このモードを使用して、以前にロードされたデータを Neptune クラスターから削除した後にソースからすべてのデータを再ロードしたり、同じソースから利用可能な新しいデータをロードしたりするために使用できます。
- AUTO - AUTO モードでは、ローダーは同じソースから以前のロードジョブを検索し、見つかった場合は、RESUME モードと同様にそのジョブを再開します。

  ローダーが同じソースから以前のロードジョブを見つけられない場合、NEW モードの場合と同様に、ソースからすべてのデータがロードされます。

- parallelism (CLI では: --parallelism) — タイプ string のパラレリズム (UTF-8 でエンコードされた文字列)。

  一括ロードプロセスで使用されるスレッド数を減らすように設定できるオプションの parallelism パラメータ。

  許可される値:

  - LOW - 使用されるスレッドの数は、コア数を 8 で割った値です。
  - MEDIUM - 使用されるスレッドの数は、コア数を 2 で割った値です。
  - HIGH - 使用されるスレッドの数は、コア数と同じです。
  - OVERSUBSCRIBE - 使用されるスレッドの数は、コア数に 2 をかけた値です。この値を使用する場合、バルクローダーは利用可能なすべてのリソースを消費します。

    ただし、これは、OVERSUBSCRIBE 設定すると、CPU 使用率が 100% になります。ロードオペレーションは I/O バウンドであるため、予想される CPU 最高使用率は 60% ~ 70% の範囲にあります。

  デフォルト値: HIGH

  parallelism openCypher データをロードするときに、設定によってスレッド間でデッドロックが発生することがあります。こうなると、Neptune は LOAD_DATA_DEADLOCK というエラーを返

します。通常、この問題は次のようにして修正できます。parallelism より低い設定にし、ロードコマンドを再試行します。

- parserConfiguration (CLI では: --parser-configuration) - キと値のペアのマップ配列。

  各キーは、タイプ string の文字列 (UTF-8 でエンコードされた文字列) です。

  各値は、タイプ string の文字列 (UTF-8 でエンコードされた文字列) です。

  parserConfiguration - 追加のパーサー設定値のあるオプションのオブジェクト。それぞれの子パラメータもオプションです。

  - **namedGraphUri** - グラフが指定されていない場合の、すべての RDF 形式のデフォルトのグラフ (四角形でない形式、およびグラフのない NQUAD エントリ)。

    デフォルトは https://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph です。

  - **baseUri** - RDF/XML および Turtle 形式の基本 URI。

    デフォルトは https://aws.amazon.com/neptune/default です。

  - **allowEmptyStrings** - Gremlin ユーザーは、CSV データを読み込むときに、ノードおよびエッジのプロパティとして空の文字列値 ("") を渡す必要があります。allowEmptyStrings が false (デフォルト) に設定されている場合、そのような空の文字列は NULL として扱われ、ロードされません。

    allowEmptyStrings が true に設定されている場合、ローダーは空の文字列を有効なプロパティ値として扱い、それに応じてロードします。

- queueRequest (CLI では: --queue-request) — タイプ boolean のブール値 (ブール値 (真または偽) の値)。

  これは、ロードリクエストをキューに入れることができるかどうかを示すオプションのフラグパラメータです。

  queueRequest パラメータをすべて "TRUE"に設定している場合、Neptune は一度に最大 64 個のジョブをキューに入れることができるので、次のジョブを発行する前に 1 つのロードジョブが完了するのを待つ必要はありません。ジョブのキューの順序は、FIFO (先入れ先出し) です。

  queueRequest パラメータを省略するるか、"FALSE" に設定した場合、別のロードジョブがすでに実行されていると、ロードリクエストは失敗します。

使用できる値: "TRUE"、"FALSE"。

デフォルト値: "FALSE"。

- s3BucketRegion (CLI では: `--s-3-bucket-region`) — 必須: `string` タイプの S3BucketRegion (UTF-8 でエンコードされた文字列)。

  S3 バケットの Amazon リージョン。これは DB クラスターの Amazon リージョンと一致する必要があります。

- source (CLI では: `--source`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  `source` パラメータは、単一のファイル、複数のファイル、1 つのフォルダ、または複数のフォルダを識別する S3 URI を受け入れます。Neptune は、指定されたフォルダ内のすべてのデータファイルをロードします。

  URI は、以下の形式のいずれかになります。

  - `s3://(bucket_name)/(object-key-name)`
  - `https://s3.amazonaws.com/(bucket_name)/(object-key-name)`
  - `https://s3.us-east-1.amazonaws.com/(bucket_name)/(object-key-name)`

  URI の `object-key-name` 要素は、S3 の ListObjects API コール内の prefix パラメータと同等です。これは、名前がそのプレフィックスで始まる指定された S3 バケット内のすべてのオブジェクトを識別します。これは、単一のファイルまたはフォルダ、または複数のファイルやフォルダにすることができます。

  特定のフォルダには複数の頂点ファイルおよび複数のエッジファイルが含まれている場合があります。

- updateSingleCardinalityProperties (CLI では: `--update-single-cardinality-properties`) — タイプ `boolean` のブール値 (ブール値 (真または偽) の値)。

  `updateSingleCardinalityProperties` は、バルクローダーが単一濃度の頂点または エッジプロパティの新しい値を処理する方法を制御するオプションのパラメータです。これは openCypher データのロードではサポートされていません。

  使用できる値: "TRUE"、"FALSE"。

  デフォルト値: "FALSE"。

デフォルトでは、または updateSingleCardinalityProperties が明示的に "FALSE" に設定されている場合、ローダーは単一の濃度に違反するため、新しい値をエラーとして扱います。

一方、updateSingleCardinalityProperties が "TRUE" に設定されている場合、バルクローダーは既存の値を新しい値に置き換えます。ロードされるソースファイルで複数のエッジまたは単一濃度の頂点プロパティ値が指定されている場合、バルクロードの終了時の最終値は、これらの新しい値のいずれかになります。ローダーは、既存の値が新しい値の 1 つに置き換えられたことを保証します。

- userProvidedEdgeIds (CLI では: --user-provided-edge-ids) — タイプ boolean のブール値 (ブール値 (真または偽) の値)。

  このパラメータは、リレーションシップ ID を含む openCypher データをロードする場合にのみ必要です。openCypher 関係 ID がロードデータに明示的に指定されている場合 (推奨)、必ず含められ True に設定されています。

  userProvidedEdgeIds が存在しないか、True に設定されている場合、:ID 列は、ロード内のすべてのリレーションシップファイルに存在する必要があります。

  userProvidedEdgeIds が存在しないか、False に設定されている場合、ロード内のリレーションシップファイルに :ID 列があってはなりません。代わりに、Neptune ローダーは各リレーションシップの ID を自動的に生成します。

  CSV データのエラーが修正された後にローダーがロードを再開できるように、リレーションシップ ID を明示的に指定すると便利です。すでにロードされているリレーションシップをリロードする必要はありません。リレーションシップ ID が明示的に割り当てられていない場合、リレーションシップファイルを修正する必要があれば、ローダーは失敗したロードを再開できず、代わりにすべてのリレーションシップを再ロードする必要があります。

レスポンス

- payload – 必須: キーと値のペアのマップ配列。

  各キーは、タイプ string の文字列 (UTF-8 でエンコードされた文字列) です。

  各値は、タイプ string の文字列 (UTF-8 でエンコードされた文字列) です。

  ロード操作の識別子となる loadId 名前と値のペアを含みます。
- status - 必須: タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

ロードジョブのステータスを示す HTTP リターンコード。

エラー

- BadRequestException
- InvalidParameterException
- BulkLoadIdNotFoundException
- ClientTimeoutException
- LoadUrlAccessDeniedException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- InternalFailureException
- S3Exception
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException

GetLoaderJobStatus (アクション)

この API の AWS CLI 名は `get-loader-job-status` です。

指定されたロードジョブに関する情報を取得します。Neptune は、直近 1,024 個のバルクロードジョブのみを追跡し、ジョブごとに直近 10,000 個のエラーの詳細のみを保存します。

詳細については、「Neptune Loader Get-Status API」を参照してください。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:GetLoaderJobStatus IAM アクションを許可するポリシーがアタッチされている必要があります。

リクエスト

- details (CLI では: `--details`) — タイプ `boolean` のブール値 (ブール値 (真または偽) の値)。

  全体のステータス以外の詳細を含めるかどうかを示すフラグ (`TRUE` または `FALSE`、デフォルトは `FALSE`)。
- errors (CLI では: `--errors`) — タイプ `boolean` のブール値 (ブール値 (真または偽) の値)。

  発生したエラーのリストを含めるかどうかを示すフラグ (`TRUE` または `FALSE`、デフォルトは `FALSE`)。

  エラーのリストはページ分割されます。`page` および `errorsPerPage` パラメータで、すべてのエラーをページ分割できます。
- errorsPerPage (CLI では: `--errors-per-page`) — タイプ `integer` の PositiveInteger (符号付き32ビット整数)、少なくとも 1? st?。

  各ページで返されるエラーの数 (正の整数、デフォルトは `10`)。`errors` パラメータが `TRUE` に設定されている場合にのみ有効です。
- loadId (CLI では: `--load-id`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  ステータスを取得するロードジョブのロード ID。
- page (CLI では: `--page`) — タイプ `integer` の PositiveInteger (符号付き32ビット整数)、少なくとも 1? st?。

  エラーページ番号 (正の整数、デフォルトは `1`)。`errors` パラメータが `TRUE` に設定されている場合にのみ有効です。

レスポンス

- payload - 必須: タイプ `document` のドキュメント (JSON のようなデータモデルで表される、プロトコルに依存しないオープンコンテンツ)。

  ロードジョブに関するステータス情報。レイアウトは次のようになります。

  Example

```
{
          "status" : "200 OK",
          "payload" : {
            "feedCount" : [
```

```
            {
              "LOAD_FAILED" : (number)
            }
          ],
          "overallStatus" : {
            "fullUri" : "s3://(bucket)/(key)",
            "runNumber" : (number),
            "retryNumber" : (number),
            "status" : "(string)",
            "totalTimeSpent" : (number),
            "startTime" : (number),
            "totalRecords" : (number),
            "totalDuplicates" : (number),
            "parsingErrors" : (number),
            "datatypeMismatchErrors" : (number),
            "insertErrors" : (number),
          },
          "failedFeeds" : [
            {
              "fullUri" : "s3://(bucket)/(key)",
              "runNumber" : (number),
              "retryNumber" : (number),
              "status" : "(string)",
              "totalTimeSpent" : (number),
              "startTime" : (number),
              "totalRecords" : (number),
              "totalDuplicates" : (number),
              "parsingErrors" : (number),
              "datatypeMismatchErrors" : (number),
              "insertErrors" : (number),
            }
          ],
          "errors" : {
            "startIndex" : (number),
            "endIndex" : (number),
            "loadId" : "(string),
            "errorLogs" : [ ]
          }
        }
      }
```

- status - 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  リクエストの HTTP レスポンスコード。

エラー

- BadRequestException
- InvalidParameterException
- BulkLoadIdNotFoundException
- ClientTimeoutException
- LoadUrlAccessDeniedException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- InternalFailureException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException

ListLoaderJobs (アクション)

この API の AWS CLI 名は `list-loader-jobs` です。

すべてのアクティブローダージョブの `loadIds` のリストを取得します。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:ListLoaderJobs IAM アクションを許可するポリシーがアタッチされている必要があります。

リクエスト

- includeQueuedLoads (CLI では: `--include-queued-loads`) — タイプ `boolean` のブール値 (ブール値 (真または偽) の値)。

  パラメータを `FALSE` に設定することによってロード ID のリストをリクエストしたときに、キューに入れられたロードリクエストのロード ID を除外するために使用できるオプションのパラメータ。デフォルト値は、「`TRUE`」です。
- limit (CLI では: `--limit`) — ListLoaderJobsInputLimitInteger、タイプ `integer` (符号付き 32 ビット整数)、1 以上か 1024 以上？ st? s。

一覧表示されるロード ID の数。0 より大きく、100 (デフォルト) 以下の正の整数でなければなりません。

レスポンス

- payload – 必須: LoaderIdResult オブジェクト。

  要求されたジョブ ID のリスト。
- status - 必須: タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  ジョブリストリクエストのステータスを返します。

エラー

- UnsupportedOperationException
- BadRequestException
- InvalidParameterException
- BulkLoadIdNotFoundException
- InternalFailureException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- LoadUrlAccessDeniedException
- IllegalArgumentException
- TooManyRequestsException

CancelLoaderJob (アクション)

この API の AWS CLI 名は cancel-loader-job です。

指定されたロードジョブをキャンセルします。これは HTTP DELETE リクエストです。詳細については、「Neptune Loader Get-Status API」を参照してください。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:CancelLoaderJob IAM アクションを許可するポリシーがアタッチされている必要があります。

リクエスト

- loadId (CLI では: `--load-id`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  削除されるロードジョブの ID。

レスポンス

- status — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  キャンセルステータス。

エラー

- BadRequestException
- InvalidParameterException
- BulkLoadIdNotFoundException
- ClientTimeoutException
- LoadUrlAccessDeniedException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- InternalFailureException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException

一括ロード構造:

LoaderIdResult (構造)

ロード ID のリストを含みます。

フィールド

- loadIds — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  ロード ID のリスト。

Neptune ストリームデータプレーン API

ストリームアクセスアクション:

- GetPropertygraphStream (アクション)

ストリームデータ構造:

- PropertygraphRecord (構造)
- PropertygraphData (構造)

GetPropertygraphStream (アクション)

> この API の AWS CLI 名は `get-propertygraph-stream` です。

プロパティグラフのストリームを取得します。

Neptune ストリーム機能を使用すると、グラフデータに加えられたすべての変更を記録する、変更ログエントリの完全なシーケンスを生成できます。`GetPropertygraphStream` では、プロパティグラフについて、これらの変更ログエントリを収集できます。

Neptune ストリーム機能を Neptune DB クラスターで有効にする必要があります。ストリームを有効にするには、neptune_stream DB クラスターパラメータを 1 に設定します。

「Neptune ストリームを使用してグラフの変更をリアルタイムでキャプチャする」を参照してください。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:GetStreamRecords IAM アクションを許可するポリシーがアタッチされている必要があります。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、クエリに応じて、そのクラスターで以下の IAM アクションを許可するポリシーがアタッチされている必要があります。

以下の IAM コンテキストキーを使用してプロパティグラフクエリを制限できることに注意してください。

- neptune-db:QueryLanguage:Gremlin
- neptune-db:QueryLanguage:OpenCypher

「Neptune (IAM データアクセスポリシーステートメントで使用可能な条件キー」を参照してください。

リクエスト

- commitNum (CLI では: `--commit-num`) — タイプ `long` の長整数 (符号付き 32 ビット整数)。

  変更ログストリームから読み取る開始レコードのコミット番号。このパラメータは、`iteratorType` が `AT_SEQUENCE_NUMBER` または `AFTER_SEQUENCE_NUMBER` の場合は必須であり、`iteratorType` が `TRIM_HORIZON` または `LATEST` の場合は無視されます。
- encoding (CLI では: `--encoding`) — タイプ `string` のエンコーディング (UTF-8 でエンコードされた文字列)。

  TRUE に設定された場合、Neptune は gzip エンコーディングを使用して応答を圧縮します。
- iteratorType (CLI では: `--iterator-type`) — タイプ `string` の IteratorType (UTF-8 でエンコードされた文字列)。

  次のいずれかの値を指定できます。

  - `AT_SEQUENCE_NUMBER` - `commitNum` および opNum パラメータで一緒に指定されたイベントシーケンス番号から読み取りを開始することを示します。
  - `AFTER_SEQUENCE_NUMBER` - `commitNum` および opNum パラメータで一緒に指定されたイベントシーケンス番号の直後に読み取りが開始されることを示します。

  - TRIM_HORIZON - 読み取りは、システム内の最後のトリミングされていないレコードから開始することを示します。これは、変更ログストリームで最も古い (まだ削除されていない) レコードであることを示しています。
  - LATEST - 読み取りは、システム内の最新のレコードから開始することを示します。これは、変更ログストリームで最近の (まだ削除されていない) レコードであることを示しています。
- limit (CLI では: --limit) — GetPropertygraphStreamInputLimitLong、タイプ: long (符号付き 64 ビット整数)、1 以上か 100000 未満 ?st?s。

  返すレコードの最大数を指定します。また、レスポンスのサイズ制限は 10 MB であり、これは変更できず、limit パラメータで指定されたレコード数よりも優先されます。10 MB の制限に達した場合、レスポンスにはしきい値超過レコードが含まれます。

  limit の範囲は 1 から 100,000 であり、デフォルトは 10 です。
- opNum (CLI では: --op-num) — タイプ long の長整数 (符号付き 32 ビット整数)。

  変更ログストリームデータからの読み取りを開始するための、指定されたコミット内のオペレーションシーケンス番号。デフォルトは 1 です。

レスポンス

- format - 必須: タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  返される変更レコードのシリアル化形式。現在、サポートされている値は PG_JSON のみです。
- lastEventId – 必須: キーと値のペアのマップ配列。

    各キーは、タイプ string の文字列 (UTF-8 でエンコードされた文字列) です。

    各値は、タイプ string の文字列 (UTF-8 でエンコードされた文字列) です。

  ストリームレスポンスの最後の変更のシーケンス識別子。

  イベント ID は 2 つのフィールドで構成されます。commitNum はグラフを変更したトランザクションを識別し、opNum はそのトランザクション内の特定の操作を識別します。

  Example

```
"eventId": {
          "commitNum": 12,
          "opNum": 1
```

```
        }
```

- lastTrxTimestampInMillis — 必須: タイプ `long` の長整数(符号付き 64 ビット整数)。

  トランザクションのコミットがリクエストされた時間 (Unix エポックからのミリ秒単位)。
- records – 必須: PropertygraphRecord オブジェクトの配列。

  レスポンスに含まれるシリアル化された変更ログストリームレコードの配列。
- totalRecords - 必須: タイプ `integer` の整数 (符号付き 32 ビット整数)。

  レスポンス内のレコードの総数。

エラー

- UnsupportedOperationException
- ExpiredStreamException
- InvalidParameterException
- MemoryLimitExceededException
- StreamRecordsNotFoundException
- ClientTimeoutException
- PreconditionsFailedException
- ThrottlingException
- ConstraintViolationException
- InvalidArgumentException
- IllegalArgumentException
- TooManyRequestsException

ストリームデータ構造:

PropertygraphRecord (構造)

プロパティグラフレコードの構造。

フィールド

- commitTimestampInMillis — 必須: タイプ `long` の長整数(符号付き 64 ビット整数)。

トランザクションのコミットがリクエストされた時間 (Unix エポックからのミリ秒単位)。

- data — 必須: PropertygraphData オブジェクト。

  シリアル化された Gremlin または openCypher 変更レコード。

- eventId – 必須: キーと値のペアのマップ配列。

  各キーは、タイプ `string` の文字列 (UTF-8 でエンコードされた文字列) です。

  各値は、タイプ `string` の文字列 (UTF-8 でエンコードされた文字列) です。

  ストリーム変更レコードのシーケンス識別子。

- isLastOp - これはタイプ `boolean` のブール値です (ブール値 (真または偽))。

  この操作がトランザクションの最後の操作である場合にのみ表示されます。存在する場合、true に設定されます。トランザクション全体が確実に消費されるようにする場合に便利です。

- op - これは必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  変更を作成した操作。

PropertygraphData (構造)

Gremlin または openCypher 変更レコード。

フィールド

- from — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  これがエッジ (type = e) の場合、対応する `from` 頂点またはソースノードの ID。

- id - これは必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  Gremlin または openCypher 要素の ID。

- key - これは必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  プロパティ名。要素ラベルの場合、これは `label` です。

- to — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  これがエッジ (type = e) の場合、対応する `to` 頂点またはターゲットノードの ID。

- type - これは必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

この Gremlin または openCypher 要素のタイプ。次のいずれかにする必要があります。

- **v1** — Gremlin の頂点ラベル、または openCypher のノードラベル。
- **vp** — Gremlin の頂点プロパティ、または openCypher のノードプロパティ。
- **e** — Gremlin のエッジとエッジラベル、または openCypher のリレーションシップとリレーションシップタイプ。
- **ep** — Gremlin のエッジプロパティ、または openCypher のリレーションシッププロパティ。

- value - 必須: タイプ `document` のドキュメント (JSON のようなデータモデルで表される、プロトコルに依存しないオープンコンテンツ)。

  これは、値自体の値フィールドと、その値の JSON データ型のデータ型フィールドを含む JSON オブジェクトです。

  Example

```
"value": {
          "value": "(the new value"),
          "dataType": "(the JSON datatypenew value")
        }
```

Neptune データプレーン統計およびグラフサマリー API

プロパティグラフ統計アクション:

- GetPropertygraphStatistics (アクション)
- ManagePropertygraphStatistics (アクション)
- DeletePropertygraphStatistics (アクション)
- GetPropertygraphSummary (アクション)

統計構造:

- 統計 (構造)
- StatisticsSummary (構造)
- DeleteStatisticsValueMap (構造)
- RefreshStatisticsIdMap (構造)

- NodeStructure (構造)
- EdgeStructure (構造)
- SubjectStructure (構造)
- PropertygraphSummaryValueMap (構造)
- PropertygraphSummary (構造)

GetPropertygraphStatistics (アクション)

この API の AWS CLI 名は `get-propertygraph-statistics` です。

プロパティグラフ統計 (Gremlin と openCypher) を取得します。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:GetStatisticsStatus IAM アクションを許可するポリシーがアタッチされている必要があります。

リクエスト

- リクエストパラメータなし

レスポンス

- payload – 必須: 統計 オブジェクト。

  プロパティグラフデータの統計。
- status - 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  リクエストの HTTP リターンコード。リクエストが成功した場合、コードは 200 です。よくあるエラーのリストについては、「DFE 統計リクエストの一般的なエラーコード」を参照してください。

エラー

- BadRequestException
- InvalidParameterException
- StatisticsNotAvailableException
- ClientTimeoutException

- AccessDeniedException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- PreconditionsFailedException
- ReadOnlyViolationException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException

ManagePropertygraphStatistics (アクション)

この API の AWS CLI 名は `manage-propertygraph-statistics` です。

プロパティグラフ統計の生成と使用を管理します。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:ManageStatistics IAM アクションを許可するポリシーがアタッチされている必要があります。

リクエスト

- mode (CLI では: `--mode`) — タイプ `string` の StatisticsAutoGenerationMode (UTF-8 でエンコードされた文字列)。

  統計生成モード。`DISABLE_AUTOCOMPUTE`、`ENABLE_AUTOCOMPUTE`、または `REFRESH` のいずれか。 を指定すると、DFE 統計の生成が手動でトリガーされます。

レスポンス

- payload – RefreshStatisticsIdMap オブジェクト。

  これは更新モードの場合にのみ返されます。
- status - 必須: タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  リクエストの HTTP リターンコード。リクエストが成功した場合、コードは 200 です。

エラー

- BadRequestException
- InvalidParameterException
- StatisticsNotAvailableException
- ClientTimeoutException
- AccessDeniedException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- PreconditionsFailedException
- ReadOnlyViolationException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException

DeletePropertygraphStatistics (アクション)

この API の AWS CLI 名は `delete-propertygraph-statistics` です。

Gremlin と openCypher (プロパティグラフ) データの統計を削除します。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:DeleteStatistics IAM アクションを許可するポリシーがアタッチされている必要があります。

リクエスト

- リクエストパラメータなし

レスポンス

- payload – DeleteStatisticsValueMap オブジェクト。

  削除ペイロード。
- status — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

キャンセルステータス。

- statusCode - タイプ `integer` の整数 (符号付き 32 ビット整数)。

  HTTP レスポンスコード: 削除が成功した場合は 200、または削除する統計がない場合は 204。

エラー

- BadRequestException
- InvalidParameterException
- StatisticsNotAvailableException
- ClientTimeoutException
- AccessDeniedException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- PreconditionsFailedException
- ReadOnlyViolationException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException

GetPropertygraphSummary (アクション)

この API の AWS CLI 名は `get-propertygraph-summary` です。

プロパティグラフのグラフ概要を取得します。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:GetGraphSummary IAM アクションを許可するポリシーがアタッチされている必要があります。

リクエスト

- mode (CLI では: `--mode`) — タイプ `string` の GraphSummaryType (UTF-8 でエンコードされた文字列)。

Mode には、BASIC (デフォルト) と DETAILED の 2 つの値のいずれかを指定できます。

レスポンス

- payload – PropertygraphSummaryValueMap オブジェクト。

  プロパティグラフサマリーレスポンスを含むペイロード。
- statusCode - タイプ integer の整数 (符号付き 32 ビット整数)。

  リクエストの HTTP リターンコード。リクエストが成功した場合、コードは 200 です。

エラー

- BadRequestException
- InvalidParameterException
- StatisticsNotAvailableException
- ClientTimeoutException
- AccessDeniedException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- PreconditionsFailedException
- ReadOnlyViolationException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException

統計構造:

統計 (構造)

統計情報を含みます。DFE エンジンは、Neptune グラフのデータに関する情報を使用して、クエリの実行を計画する際に効果的なトレードオフを行います。この情報は、クエリ計画を導くことができ

る、いわゆる特性セットと述語統計を含む統計の形式をとります。「Neptune DFE が使用する統計の管理」を参照してください。

フィールド

- active - これはタイプ boolean のブール値です (ブール値 (真または偽))。

  DFE 統計情報の生成が有効になっているかどうかを示します。
- autoCompute - これはタイプ boolean のブール値です (ブール値 (真または偽))。

  統計情報の自動生成が有効になっているかどうかを示します。
- date — これは、タイプ string の SyntheticTimestamp_date_time です (UTF-8 でエンコードされた文字列)。

  DFE 統計情報が最近生成された UTC 時刻。
- note — これはタイプ string の文字列です (UTF-8 でエンコードされた文字列)。

  統計情報が無効な場合の問題に関するメモ。
- signatureInfo - これは StatisticsSummary オブジェクトです。

  以下を含む統計サマリー構造体。
  - signatureCount — すべての特性セットにおけるシグニチャの総数。
  - instanceCount — 特性セットインスタンスの合計数。
  - predicateCount — 一意述語の合計数。
- statisticsId — これはタイプ string の文字列です (UTF-8 でエンコードされた文字列)。

  現在の統計生成の実行の ID を報告します。-1 の値は統計が生成されていないことを示します。

StatisticsSummary (構造)

統計で生成された特性セットに関する情報。

フィールド

- instanceCount - タイプ integer の整数 (符号付き 32 ビット整数)。

  特性セットインスタンスの合計数。
- predicateCount - タイプ integer の整数 (符号付き 32 ビット整数)。

一意述語の合計数。

- signatureCount - タイプ `integer` の整数 (符号付き 32 ビット整数)。

  すべての特性セットにおけるシグニチャの総数。

DeleteStatisticsValueMap (構造)

統計削除のペイロード。

フィールド

- active - これはタイプ `boolean` のブール値です (ブール値 (真または偽))。

  統計の現在のステータス。
- statisticsId — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  現在発生している統計生成の実行の ID。

RefreshStatisticsIdMap (構造)

`REFRESH` モードの統計。

フィールド

- statisticsId — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  現在発生している統計生成の実行の ID。

NodeStructure (構造)

ノード構造

フィールド

- count — タイプ `long` の長整数 (符号付き 64 ビット整数)。

  この特定の構造を持つノードの数。
- distinctOutgoingEdgeLabels — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

この特定の構造に存在する個別の出力エッジラベルのリスト。

- nodeProperties — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  この特定の構造に存在するノードプロパティのリスト。

EdgeStructure (構造)

エッジ構造。

フィールド

- count — タイプ `long` の長整数 (符号付き 64 ビット整数)。

  この特定の構造を持つエッジの数。
- edgeProperties — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  この特定の構造に存在するエッジプロパティのリスト。

SubjectStructure (構造)

サブジェクト構造。

フィールド

- count — タイプ `long` の長整数 (符号付き 64 ビット整数)。

  この特定の構造の出現回数。
- predicates — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  この特定の構造に存在する述語のリスト。

PropertygraphSummaryValueMap (構造)

プロパティグラフサマリーレスポンスのペイロード。

フィールド

- graphSummary - これは PropertygraphSummary オブジェクトです。

グラフの要約。

- lastStatisticsComputationTime — これは、タイプ `string` の SyntheticTimestamp_date_time です (UTF-8 でエンコードされた文字列)。

  Neptune が最後に統計を計算した時刻のタイムスタンプ (ISO 8601 形式) 。

- version — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  このグラフサマリーレスポンスのバージョン。

PropertygraphSummary (構造)

グラフサマリー API は、ノードおよびエッジラベルとプロパティキーの読み取り専用リストを、ノード、エッジ、プロパティの数とともに返します。「プロパティグラフ (PG) のグラフサマリーレスポンス」を参照してください。

フィールド

- edgeLabels — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  グラフ内の異なるエッジラベルのリスト。

- edgeProperties — LongValuedMap オブジェクト。キーと値のペアのマップ配列。

  各キーは、タイプ `string` の文字列 (UTF-8 でエンコードされた文字列) です。

  各値はタイプ `long` の長整数です (符号付き 64 ビット整数)。

  グラフ内の個別のエッジプロパティのリストと、各プロパティが使用されるエッジの数です。

- edgeStructures — これは EdgeStructure オブジェクトの配列です。

  このフィールドは、要求されたモードが `DETAILED` の場合にのみ表示されます。エッジ構造のリストを含みます。

- nodeLabels — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  グラフ内の異なるノードラベルのリスト。

- nodeProperties — LongValuedMap オブジェクト。キーと値のペアのマップ配列。

  各キーは、タイプ `string` の文字列 (UTF-8 でエンコードされた文字列) です。

各値はタイプ `long` の長整数です (符号付き 64 ビット整数)。

グラフ内の異なるノードプロパティの数。

- nodeStructures — これは NodeStructure オブジェクトの配列です。

  このフィールドは、要求されたモードが `DETAILED` の場合にのみ表示されます。ノード構造のリストを含みます。
- numEdgeLabels — タイプ `long` の長整数 (符号付き 64 ビット整数)。

  グラフ内の異なるエッジラベルの数。
- numEdgeProperties — タイプ `long` の長整数 (符号付き 64 ビット整数)。

  グラフ内の異なるエッジプロパティの数。
- numEdges — タイプ `long` の長整数 (符号付き 64 ビット整数)。

  グラフ内のエッジの数。
- numNodeLabels — タイプ `long` の長整数 (符号付き 64 ビット整数)。

  グラフ内の異なるノードラベルの数。
- numNodeProperties — タイプ `long` の長整数 (符号付き 64 ビット整数)。

  グラフ内の個別のノードプロパティのリストと、各プロパティが使用されるノードの数。
- numNodes — タイプ `long` の長整数 (符号付き 64 ビット整数)。

  グラフ内のノードの数。
- totalEdgePropertyValues — タイプ `long` の長整数 (符号付き 64 ビット整数)。

  すべてのエッジプロパティの使用回数の合計。
- totalNodePropertyValues — タイプ `long` の長整数 (符号付き 64 ビット整数)。

  すべてのノードプロパティの使用回数の合計。

Neptune ML データ処理 API

データ処理アクション:

- StartMLDataProcessingJob (アクション)

- ListMLDataProcessingJobs (アクション)
- GetMLDataProcessingJob (アクション)
- CancelMLDataProcessingJob (アクション)

ML 汎用構造体:

- MlResourceDefinition (構造)
- MLConfigDefinition (構造)

StartMLDataProcessingJob (アクション)

この API の AWS CLI 名は `start-ml-data-processing-job` です。

Neptune からエクスポートされたグラフデータをトレーニング用に処理するための新しい Neptune ML データ処理ジョブを作成します。「`dataprocessing` コマンド」を参照してください。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:StartMLModelDataProcessingJob IAM アクションを許可するポリシーがアタッチされている必要があります。

リクエスト

- configFileName (CLI では: `--config-file-name`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  トレーニング用にエクスポートされたグラフデータをロードする方法を説明するデータ仕様ファイル。ファイルは Neptune エクスポートツールキットによって自動的に生成されます。デフォルトは `training-data-configuration.json` です。
- id (CLI では: `--id`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  新しいジョブの一意識別子。デフォルト値は、自動生成された UUID です。
- inputDataS3Location (CLI では: `--input-data-s3-location`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  データ処理ジョブの実行に必要なデータを SageMaker がダウンロードする Amazon S3 ロケーションの URI。
- modelType (CLI では: `--model-type`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

Neptune ML が現在サポートしている 2 つのモデルタイプ、異種グラフモデル (heterogeneous) とナレッジグラフ (kge) のうちの 1 つ。デフォルトは none です。指定されなかった場合、Neptune ML はデータに基づいてモデルタイプを自動的に選択します。

- neptuneIamRoleArn (CLI では: `--neptune-iam-role-arn`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  SageMaker がユーザーに代わってタスクを実行するために引き受けることができる IAM ロールの Amazon リソースネーム (ARN)。これは DB クラスターパラメータグループに一覧表示されている必要があり、そうでない場合はエラーが発生します。

- previousDataProcessingJobId (CLI では: `--previous-data-processing-job-id`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  以前のバージョンのデータに対して実行された完了したデータ処理ジョブのジョブ ID。

- processedDataS3Location (CLI では: `--processed-data-s3-location`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  SageMaker がデータ処理ジョブの結果を保存する Amazon S3 ロケーションの URI。

- processingInstanceType (CLI では: `--processing-instance-type`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  データ処理中に使用される ML インスタンスのタイプ。処理されたデータセットを保持できる十分なメモリ容量が必要です。デフォルト値は、ディスク上にエクスポートされたグラフデータのサイズの 10 倍のメモリを持つ最も小さい ml.r5 タイプです。

- processingInstanceVolumeSizeInGB (CLI では: `--processing-instance-volume-size-in-gb`) — タイプ `integer` の整数 (符号付き 32 ビット整数)。

  処理インスタンスのディスクボリュームサイズ。入力データと処理されたデータの両方がディスクに保存されるため、ボリュームサイズは両方のデータセットを保持するのに十分な大きさでなければなりません。デフォルトは 0 です。指定されなかったか、0 の場合、Neptune ML はデータサイズに基づいてボリュームサイズを自動的に選択します。

- processingTimeOutInSeconds (CLI では: `--processing-time-out-in-seconds`) — タイプ `integer` の整数 (符号付き 32 ビット整数)。

  データ処理ジョブのタイムアウト (秒単位)。デフォルトは 86,400 (1 日) です。

- s3OutputEncryptionKMSKey (CLI では: `--s-3-output-encryption-kms-key`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

SageMaker が処理ジョブの出力を暗号化するために使用する Amazon キー管理サービス (Amazon KMS) キー。デフォルトは none です。

- sagemakerIamRoleArn (CLI では: `--sagemaker-iam-role-arn`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  SageMaker 実行のための IAM ロールの ARN。これは DB クラスターパラメータグループに一覧表示されている必要があり、そうでない場合はエラーが発生します。
- securityGroupIds (CLI では: `--security-group-ids`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  VPC セキュリティグループ ID。デフォルトは [なし] です。
- subnets (CLI では: `--subnets`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  Neptune VPC 内のサブネットの ID。デフォルトは [なし] です。
- volumeEncryptionKMSKey (CLI では: `--volume-encryption-kms-key`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  トレーニングジョブを実行する ML コンピューティングインスタンスにアタッチされたストレージボリュームのデータを暗号化するために SageMaker が使用する Amazon キー管理サービス (Amazon KMS) キー。デフォルトは [なし] です。

レスポンス

- arn — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  データ処理ジョブの ARN。
- creationTimeInMillis — タイプ `long` の長整数 (符号付き 64 ビット整数)。

  新しい処理ジョブの作成にかかった時間 (ミリ秒単位)。
- id — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  新しいデータ処理ジョブの一意 ID。

エラー

- UnsupportedOperationException
- BadRequestException

- MLResourceNotFoundException
- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

ListMLDataProcessingJobs (アクション)

この API の AWS CLI 名は `list-ml-data-processing-jobs` です。

Neptune ML データ処理ジョブのリストを返します。「Neptune ML データ処理コマンドを使用したアクティブなデータ処理ジョブの一覧表示」を参照してください。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:ListMLDataProcessingJobs IAM アクションを許可するポリシーがアタッチされている必要があります。

リクエスト

- maxItems (CLI では: `--max-items`) — ListMLDataProcessingJobsInputMaxItemsInteger、タイプ `integer` (符号付き 32 ビット整数)、1 以上か 1024 以上？ st? s。

  返される項目の最大数 (1 ~ 1024、デフォルトは 10)。
- neptuneIamRoleArn (CLI では: `--neptune-iam-role-arn`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  SageMaker および Amazon S3 リソースへの Neptune アクセスを提供する IAM ロールの ARN。これは DB クラスターパラメータグループに一覧表示されている必要があり、そうでない場合はエラーが発生します。

レスポンス

- ids — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

データ処理ジョブ ID を一覧表示するページ。

エラー

- UnsupportedOperationException
- BadRequestException
- MLResourceNotFoundException
- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

GetMLDataProcessingJob (アクション)

この API の AWS CLI 名は `get-ml-data-processing-job` です。

指定されたデータ処理ジョブに関する情報を取得します。「`dataprocessing` コマンド」を参照してください。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:neptune-db:GetMLDataProcessingJobStatus IAM アクションを許可するポリシーがアタッチされている必要があります。

リクエスト

- id (CLI では: `--id`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  取得されるデータ処理ジョブの一意識別子。
- neptuneIamRoleArn (CLI では: `--neptune-iam-role-arn`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

SageMaker および Amazon S3 リソースへの Neptune アクセスを提供する IAM ロールの ARN。これは DB クラスターパラメータグループに一覧表示されている必要があり、そうでない場合はエラーが発生します。

レスポンス

- id — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  このデータ処理ジョブの一意識別子。
- processingJob – MlResourceDefinition オブジェクト。

  データ処理ジョブの定義。
- status — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  データ処理ジョブのステータス。

エラー

- UnsupportedOperationException
- BadRequestException
- MLResourceNotFoundException
- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

CancelMLDataProcessingJob (アクション)

この API の AWS CLI 名は `cancel-ml-data-processing-job` です。

Neptune ML データ処理ジョブをキャンセルします。「dataprocessing コマンド」を参照してください。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:CancelMLDataProcessingJob IAM アクションを許可するポリシーがアタッチされている必要があります。

リクエスト

- clean (CLI では: `--clean`) — タイプ `boolean` のブール値 (ブール値 (真または偽) の値)。

  TRUE に設定された場合、このフラグは、ジョブが停止したときにすべての Neptune ML S3 アーティファクトが削除される必要があることを指定します。デフォルトは FALSE です。
- id (CLI では: `--id`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  データ処理ジョブの一意識別子。
- neptuneIamRoleArn (CLI では: `--neptune-iam-role-arn`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  SageMaker および Amazon S3 リソースへの Neptune アクセスを提供する IAM ロールの ARN。これは DB クラスターパラメータグループに一覧表示されている必要があり、そうでない場合はエラーが発生します。

レスポンス

- status — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  キャンセルリクエストのステータス。

エラー

- UnsupportedOperationException
- BadRequestException
- MLResourceNotFoundException
- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException

- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

ML 汎用構造体:

MlResourceDefinition (構造)

Neptune ML リソースを定義します。

フィールド

- arn — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  リソース ARN。
- cloudwatchLogUrl — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  リソースの CloudWatch ログ URL。
- failureReason — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  障害が発生した場合の障害理由。
- name — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  リソース名。
- outputLocation — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  出力場所。
- status — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  リソースステータス。

MLConfigDefinition (構造)

Neptune ML 構成を含みます。

フィールド

- arn — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  構成の ARN。
- name — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  設定名。

Neptune ML モデルトレーニング API

モデルトレーニングアクション:

- StartMLModelTrainingJob (アクション)
- ListMLModelTrainingJobs (アクション)
- GetMLModelTrainingJob (アクション)
- CancelMLModelTrainingJob (アクション)

モデルトレーニング構造:

- CustomModelTrainingParameters (構造)

StartMLModelTrainingJob (アクション)

> この API の AWS CLI 名は `start-ml-model-training-job` です。

新しい Neptune ML モデルトレーニングジョブを作成します。「`modeltraining` コマンドを使用したモデルトレーニング」を参照してください。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:StartMLModelTrainingJob IAM アクションを許可するポリシーがアタッチされている必要があります。

リクエスト

- baseProcessingInstanceType (CLI では: `--base-processing-instance-type`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

ML モデルのトレーニングの準備と管理に使用される ML インスタンスのタイプ。これは、トレーニングデータとモデルの処理に必要なメモリに基づいて選択された CPU インスタンスです。

- customModelTrainingParameters (CLI では: `--custom-model-training-parameters`) — CustomModelTrainingParameters オブジェクト。

  カスタムモデルトレーニングの設定。これは JSON オブジェクトです。

- dataProcessingJobId (CLI では: `--data-processing-job-id`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  トレーニングで使用するデータを作成した、完了したデータ処理ジョブのジョブ ID。

- enableManagedSpotTraining (CLI では: `--enable-managed-spot-training`) — タイプ `boolean` のブール値 (ブール値 (真または偽) の値)。

  Amazon Elastic Compute Cloud スポットインスタンスを使用して、機械学習モデルのトレーニングコストを最適化します。デフォルトは `False` です。

- id (CLI では: `--id`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  新しいジョブの一意識別子。デフォルト値は、自動生成された UUID です。

- maxHPONumberOfTrainingJobs (CLI では: `--max-hpo-number-of-training-jobs`) — タイプ `integer` の整数 (符号付き 32 ビット整数)。

  ハイパーパラメータ調整ジョブで開始するトレーニングジョブの最大総数。デフォルトは 2 です。Neptune ML は、機械学習モデルのハイパーパラメータを自動的に調整します。うまく機能するモデルを得るには、少なくとも 10 個のジョブを使用します (つまり、`maxHPONumberOfTrainingJobs` を 10 と設定)。一般的に、チューニングの実行が多いほど、結果は良くなります。

- maxHPOParallelTrainingJobs (CLI では: `--max-hpo-parallel-training-jobs`) — タイプ `integer` の整数 (符号付き 32 ビット整数)。

  ハイパーパラメータ調整ジョブで開始する並列トレーニングジョブの最大総数。デフォルトは 2 です。実行できる並列ジョブ数は、トレーニングインスタンスで使用可能なリソースによって制限されます。

- neptuneIamRoleArn (CLI では: `--neptune-iam-role-arn`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

SageMaker および Amazon S3 リソースへの Neptune アクセスを提供する IAM ロールの ARN。これは DB クラスターパラメータグループに一覧表示されている必要があります。そうでない場合はエラーが発生します。

- previousModelTrainingJobId (CLI では: `--previous-model-training-job-id`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  更新されたデータに基づいて増分的に更新する、完了したモデルトレーニングジョブのジョブ ID。

- s3OutputEncryptionKMSKey (CLI では: `--s-3-output-encryption-kms-key`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  SageMaker が処理ジョブの出力を暗号化するために使用する Amazon キー管理サービス (KMS) キー。デフォルトは none です。

- sagemakerIamRoleArn (CLI では: `--sagemaker-iam-role-arn`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  SageMaker を実行する IAM ロールの ARN。これは DB クラスターパラメータグループに一覧表示されている必要があり、そうでない場合はエラーが発生します。

- securityGroupIds (CLI では: `--security-group-ids`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  VPC セキュリティグループ ID。デフォルトは [なし] です。

- subnets (CLI では: `--subnets`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  Neptune VPC 内のサブネットの ID。デフォルトは [なし] です。

- trainingInstanceType (CLI では: `--training-instance-type`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  モデルトレーニングに使用される ML インスタンスのタイプ。すべてのNeptune ML モデルは、CPU、GPU、MultiGPU トレーニングをサポートしています。デフォルトは `ml.p3.2xlarge` です。トレーニングに適したインスタンスタイプの選択は、タスクタイプ、グラフサイズ、および予算によって異なります。

- trainingInstanceVolumeSizeInGB (CLI では: `--training-instance-volume-size-in-gb`) — タイプ `integer` の整数 (符号付き 32 ビット整数)。

トレーニングインスタンスのディスクボリュームサイズ。入力データと出力モデルの両方がディスクに保存されるため、ボリュームサイズは両方のデータセットを保持するのに十分な大きさでなければなりません。デフォルトは 0 です。指定しなかった場合、または 0 の場合、Neptune ML はデータ処理ステップで生成された推奨に基づいてディスクボリュームサイズを選択します。

- trainingTimeOutInSeconds (CLI では: `--training-time-out-in-seconds`) — タイプ `integer` の整数 (符号付き 32 ビット整数)。

  トレーニングジョブのタイムアウト (秒単位)。デフォルトは 86,400 (1 日) です。

- trainModelS3Location (CLI では: `--train-model-s3-location`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  モデルアーティファクトが保存される Amazon S3 内の場所。

- volumeEncryptionKMSKey (CLI では: `--volume-encryption-kms-key`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  トレーニングジョブを実行する ML コンピューティングインスタンスにアタッチされたストレージボリュームのデータを暗号化するために SageMaker が使用する Amazon キー管理サービス (KMS) キー。デフォルトは [なし] です。

レスポンス

- arn — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  新しいモデルトレーニングジョブの ARN。

- creationTimeInMillis — `long` 型の長整数(符号付き 64 ビット整数)。

  モデルトレーニングジョブの作成時間 (ミリ秒単位)。

- id — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  新しいモデルトレーニングジョブの一意 ID。

エラー

- UnsupportedOperationException
- BadRequestException
- MLResourceNotFoundException

- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

ListMLModelTrainingJobs (アクション)

この API の AWS CLI 名は `list-ml-model-training-jobs` です。

Neptune ML モデルトレーニングジョブを一覧表示します。「modeltraining コマンドを使用したモデルトレーニング」を参照してください。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:neptune-db:ListMLModelTrainingJobs IAM アクションを許可するポリシーがアタッチされている必要があります。

リクエスト

- maxItems (CLI では: `--max-items`) — ListMLModelTrainingJobsInputMaxItemsInteger、タイプ `integer` (符号付き 32 ビット整数)、1 以上か 1024 以上？ st? s。

  返される項目の最大数 (1 ~ 1024、デフォルトは 10)。
- neptuneIamRoleArn (CLI では: `--neptune-iam-role-arn`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  SageMaker および Amazon S3 リソースへの Neptune アクセスを提供する IAM ロールの ARN。これは DB クラスターパラメータグループに一覧表示されている必要があります。そうでない場合はエラーが発生します。

レスポンス

- ids — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

モデルトレーニングジョブ ID のリストのページ。

エラー

- UnsupportedOperationException
- BadRequestException
- MLResourceNotFoundException
- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

GetMLModelTrainingJob (アクション)

この API の AWS CLI 名は `get-ml-model-training-job` です。

Neptune ML モデルトレーニングジョブに関する情報を取得します。「`modeltraining` コマンドを使用したモデルトレーニング」を参照してください。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:GetMLModelTrainingJobStatus IAM アクションを許可するポリシーがアタッチされている必要があります。

リクエスト

- id (CLI では: `--id`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  取得するモデルトレーニングジョブの一意識別子。
- neptuneIamRoleArn (CLI では: `--neptune-iam-role-arn`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

SageMaker および Amazon S3 リソースへの Neptune アクセスを提供する IAM ロールの ARN。これは DB クラスターパラメータグループに一覧表示されている必要があります。そうでない場合はエラーが発生します。

レスポンス

- hpoJob – MlResourceDefinition オブジェクト。

  HPO ジョブ。
- id — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  このモデルトレーニングジョブの一意識別子。
- mlModels – MlConfigDefinition オブジェクトの配列。

  使用されている ML モデルの設定のリスト。
- modelTransformJob – MlResourceDefinition オブジェクト。

  モデル変換ジョブ。
- processingJob – MlResourceDefinition オブジェクト。

  データ処理ジョブ。
- status — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  モデルトレーニングジョブのステータス。

エラー

- UnsupportedOperationException
- BadRequestException
- MLResourceNotFoundException
- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException

- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

CancelMLModelTrainingJob (アクション)

この API の AWS CLI 名は `cancel-ml-model-training-job` です。

Neptune ML モデルトレーニングジョブをキャンセルします。「`modeltraining` コマンドを使用したモデルトレーニング」を参照してください。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:CancelMLModelTrainingJob IAM アクションを許可するポリシーがアタッチされている必要があります。

リクエスト

- clean (CLI では: `--clean`) — タイプ `boolean` のブール値 (ブール値 (真または偽) の値)。

  `TRUE` に設定された場合、このフラグは、ジョブが停止したときにすべての Amazon S3 アーティファクトが削除される必要があることを指定します。デフォルトは `FALSE` です。
- id (CLI では: `--id`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  キャンセルされるモデルトレーニングジョブの一意識別子。
- neptuneIamRoleArn (CLI では: `--neptune-iam-role-arn`) — タイプ: `string` (UTF-8 でエンコードされた文字列) の文字列。

  SageMaker および Amazon S3 リソースへの Neptune アクセスを提供する IAM ロールの ARN。これは DB クラスターパラメータグループに一覧表示されている必要があります。そうでない場合はエラーが発生します。

レスポンス

- status — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  キャンセルのステータス。

エラー

- UnsupportedOperationException
- BadRequestException
- MLResourceNotFoundException
- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

モデルトレーニング構造:

CustomModelTrainingParameters (構造)

カスタムモデルトレーニングパラメータを含みます。「Neptune ML のカスタムモデル」を参照してください。

フィールド

- sourceS3DirectoryPath - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  モデルを実装する Python モジュールがある Amazon S3 の場所へのパス。これは、少なくともトレーニングスクリプト、変換スクリプト、および `model-hpo-configuration.json` ファイルを含む有効な既存の Amazon S3 の場所を指定しなければなりません。
- trainingEntryPointScript — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  モデルトレーニングを実行し、固定ハイパーパラメータを含め、コマンドライン引数としてハイパーパラメータを取るスクリプトのモジュール内のエントリポイントの名前。デフォルトは `training.py` です。

- transformEntryPointScript — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  モデルのデプロイに必要なモデルアーティファクトを計算するために、ハイパーパラメータ検索で最適なモデルが特定された後に実行されるスクリプトのモジュール内のエントリポイントの名前。コマンドライン引数なしで実行できるはずです。デフォルトは `transform.py` です。

Neptune ML モデル変換 API

モデル変換アクション:

- StartMLModelTransformJob (アクション)
- ListMLModelTransformJobs (アクション)
- GetMLModelTransformJob (アクション)
- CancelMLModelTransformJob (アクション)

モデル変換構造:

- CustomModelTransformParameters (構造)

StartMLModelTransformJob (アクション)

> この API の AWS CLI 名は `start-ml-model-transform-job` です。

新しいモデル変換ジョブを作成します。「トレーニング済みモデルを使用して新しいモデルアーティファクトを生成する」を参照してください。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:StartMLModelTransformJob IAM アクションを許可するポリシーがアタッチされている必要があります。

リクエスト

- baseProcessingInstanceType (CLI では: `--base-processing-instance-type`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

ML モデルのトレーニングの準備と管理に使用される ML インスタンスのタイプ。これは、トレーニングデータとモデルの処理に必要なメモリに基づいて選択された ML コンピューティングインスタンスです。

- baseProcessingInstanceVolumeSizeInGB (CLI では: `--base-processing-instance-volume-size-in-gb`) — タイプ `integer` の整数 (符号付き 32 ビット整数)。

  トレーニングインスタンスのディスクボリュームサイズ (ギガバイト単位)。デフォルトは 0 です。入力データと出力モデルの両方がディスクに保存されるため、ボリュームサイズは両方のデータセットを保持するのに十分な大きさでなければなりません。指定しなかった場合、または 0 の場合、Neptune ML はデータ処理ステップで生成された推奨に基づいてディスクボリュームサイズを選択します。

- customModelTransformParameters (CLI では: `--custom-model-transform-parameters`) — CustomModelTransformParameters オブジェクト。

  カスタムモデルを使用したモデル変換の構成情報。`customModelTransformParameters` オブジェクトには次のフィールドが含まれています。これらのフィールドには、トレーニングジョブの保存されたモデルパラメーターと互換性のある値が必要です。

- dataProcessingJobId (CLI では: `--data-processing-job-id`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  完了したデータ処理ジョブのジョブ ID。`dataProcessingJobId` と `mlModelTrainingJobId` のいずれか、または `trainingJobName` を含める必要があります。

- id (CLI では: `--id`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  新しいジョブの一意識別子。デフォルト値は、自動生成された UUID です。

- mlModelTrainingJobId (CLI では: `--ml-model-training-job-id`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  完了したモデルトレーニングジョブのジョブ ID。`dataProcessingJobId` と `mlModelTrainingJobId` のいずれか、または `trainingJobName` を含める必要があります。

- modelTransformOutputS3Location (CLI では: `--model-transform-output-s3-location`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  モデルアーティファクトが保存される Amazon S3 内の場所。

- neptuneIamRoleArn (CLI では: `--neptune-iam-role-arn`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

SageMaker および Amazon S3 リソースへの Neptune アクセスを提供する IAM ロールの ARN。これは DB クラスターパラメータグループに一覧表示されている必要があります。そうでない場合はエラーが発生します。

- s3OutputEncryptionKMSKey (CLI では: `--s-3-output-encryption-kms-key`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  SageMaker が処理ジョブの出力を暗号化するために使用する Amazon キー管理サービス (KMS) キー。デフォルトは none です。

- sagemakerIamRoleArn (CLI では: `--sagemaker-iam-role-arn`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  SageMaker 実行のための IAM ロールの ARN。これは DB クラスターパラメータグループに一覧表示されている必要があります。そうでない場合はエラーが発生します。

- securityGroupIds (CLI では: `--security-group-ids`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  VPC セキュリティグループ ID。デフォルトは [なし] です。

- subnets (CLI では: `--subnets`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  Neptune VPC 内のサブネットの ID。デフォルトは [なし] です。

- trainingJobName (CLI では: `--training-job-name`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  完了した SageMaker トレーニングジョブの名前。`dataProcessingJobId` と `mlModelTrainingJobId` のいずれか、または `trainingJobName` を含める必要があります。

- volumeEncryptionKMSKey (CLI では: `--volume-encryption-kms-key`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  トレーニングジョブを実行する ML コンピューティングインスタンスにアタッチされたストレージボリュームのデータを暗号化するために SageMaker が使用する Amazon キー管理サービス (KMS) キー。デフォルトは [なし] です。

レスポンス

- arn — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  モデル変換ジョブの ARN。

- creationTimeInMillis — `long` 型の長整数(符号付き 64 ビット整数)。

  モデル変換ジョブの作成時間 (ミリ秒単位)。
- id — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  新しいモデル変換ジョブの一意 ID。

エラー

- UnsupportedOperationException
- BadRequestException
- MLResourceNotFoundException
- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

ListMLModelTransformJobs (アクション)

この API の AWS CLI 名は `list-ml-model-transform-jobs` です。

モデル変換ジョブ ID のリストを返します。「トレーニング済みモデルを使用して新しいモデルアーティファクトを生成する」を参照してください。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:ListMLModelTransformJobs IAM アクションを許可するポリシーがアタッチされている必要があります。

リクエスト

- maxItems (CLI では: `--max-items`) — ListMLModelTransformJobsInputMaxItemsInteger、タイプ `integer` (符号付き 32 ビット整数)、1 以上か 1024 以上？ st? s。

返される項目の最大数 (1 ~ 1024、デフォルトは 10)。

- neptuneIamRoleArn (CLI では: `--neptune-iam-role-arn`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  SageMaker および Amazon S3 リソースへの Neptune アクセスを提供する IAM ロールの ARN。これは DB クラスターパラメータグループに一覧表示されている必要があります。そうでない場合はエラーが発生します。

レスポンス

- ids — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  モデル変換 ID のリストにあるページ。

エラー

- UnsupportedOperationException
- BadRequestException
- MLResourceNotFoundException
- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

GetMLModelTransformJob (アクション)

この API の AWS CLI 名は `get-ml-model-transform-job` です。

指定されたモデル変換ジョブに関する情報を取得します。「トレーニング済みモデルを使用して新しいモデルアーティファクトを生成する」を参照してください。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:GetMLModelTransformJobStatus IAM アクションを許可するポリシーがアタッチされている必要があります。

リクエスト

- id (CLI では: `--id`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  取得されるモデル変換ジョブの一意識別子。

- neptuneIamRoleArn (CLI では: `--neptune-iam-role-arn`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  SageMaker および Amazon S3 リソースへの Neptune アクセスを提供する IAM ロールの ARN。これは DB クラスターパラメータグループに一覧表示されている必要があります。そうでない場合はエラーが発生します。

レスポンス

- baseProcessingJob – MlResourceDefinition オブジェクト。

  基本データ処理ジョブ。

- id — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  取得されるモデル変換ジョブの一意識別子。

- models – MlConfigDefinition オブジェクトの配列。

  使用されているモデルの設定情報のリスト。

- remoteModelTransformJob – MlResourceDefinition オブジェクト。

  リモートモデル変換ジョブ。

- status — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  モデル変換ジョブのステータス。

エラー

- UnsupportedOperationException
- BadRequestException

- MLResourceNotFoundException
- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

CancelMLModelTransformJob (アクション)

この API の AWS CLI 名は `cancel-ml-model-transform-job` です。

指定されたモデル変換ジョブをキャンセルします。「トレーニング済みモデルを使用して新しいモデルアーティファクトを生成する」を参照してください。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:CancelMLModelTransformJob IAM アクションを許可するポリシーがアタッチされている必要があります。

リクエスト

- clean (CLI では: `--clean`) — タイプ `boolean` のブール値 (ブール値 (真または偽) の値)。

  このフラグが `TRUE` に設定された場合、すべての Neptune ML S3 アーティファクトは、ジョブが停止したときに削除される必要があります。デフォルトは `FALSE` です。
- id (CLI では: `--id`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  キャンセルされるモデル変換ジョブの一意 ID。
- neptuneIamRoleArn (CLI では: `--neptune-iam-role-arn`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  SageMaker および Amazon S3 リソースへの Neptune アクセスを提供する IAM ロールの ARN。これは DB クラスターパラメータグループに一覧表示されている必要があります。そうでない場合はエラーが発生します。

レスポンス

- status — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  キャンセルのステータス。

エラー

- UnsupportedOperationException
- BadRequestException
- MLResourceNotFoundException
- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

モデル変換構造:

CustomModelTransformParameters (構造)

カスタムモデル変換パラメータを含みます。「トレーニング済みモデルを使用して新しいモデルアーティファクトを生成する」を参照してください。

フィールド

- sourceS3DirectoryPath - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  モデルを実装する Python モジュールがある Amazon S3 の場所へのパス。これは、少なくともトレーニングスクリプト、変換スクリプト、および `model-hpo-configuration.json` ファイルを含む有効な既存の Amazon S3 の場所を指定しなければなりません。

- transformEntryPointScript — これはタイプ `string` の文字列です (UTF-8 でエンコードされた文字列)。

  モデルのデプロイに必要なモデルアーティファクトを計算するために、ハイパーパラメータ検索で最適なモデルが特定された後に実行されるスクリプトのモジュール内のエントリポイントの名前。コマンドライン引数なしで実行できるはずです。デフォルトは `transform.py` です。

Neptune ML 推論エンドポイント API

推論エンドポイントアクション:

- CreateMLEndpoint (アクション)
- ListMLEndpoints (アクション)
- GetMLEndpoint (アクション)
- DeleteMLEndpoint (アクション)

CreateMLEndpoint (アクション)

> この API の AWS CLI 名は `create-ml-endpoint` です。

モデルトレーニングプロセスによって構築された 1 つの特定のモデルをクエリできる新しい Neptune ML 推論エンドポイントを作成します。「エンドポイントコマンドを使用して推論エンドポイントを管理する」を参照してください。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:CreateMLEndpoint IAM アクションを許可するポリシーがアタッチされている必要があります。

リクエスト

- id (CLI では: `--id`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  新しい推論エンドポイントの一意識別子。デフォルト値は、自動生成されたタイムスタンプ付きの名前です。

- instanceCount (CLI では: `--instance-count`) — タイプ `integer` の整数 (符号付き 32 ビット整数)。

予測のためにエンドポイントにデプロイする Amazon EC2 インスタンスの最小数。デフォルトは 1 です。

- instanceType (CLI では: `--instance-type`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  オンラインサービスに使用される Neptune ML インスタンスのタイプ。デフォルトは `ml.m5.xlarge` です。推論エンドポイントの ML インスタンスの選択は、タスクタイプ、グラフサイズ、および予算によって異なります。

- mlModelTrainingJobId (CLI では: `--ml-model-training-job-id`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  推論エンドポイントが指すモデルを作成した、完了したモデルトレーニングジョブのジョブ ID。`mlModelTrainingJobId` または `mlModelTransformJobId` のいずれかを指定する必要があります。

- mlModelTransformJobId (CLI では: `--ml-model-transform-job-id`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  完了したモデル変換ジョブのジョブ ID。`mlModelTrainingJobId` または `mlModelTransformJobId` のいずれかを指定する必要があります。

- modelName (CLI では: `--model-name`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  トレーニング用のモデルタイプ。デフォルトでは、Neptune ML モデルは、データ処理に使用される `modelType` に自動的に基づきますが、ここで別のモデルタイプを指定することができます。デフォルトは、異種グラフについては `rgcn` であり、ナレッジグラフについては `kge` です。異種グラフの唯一有効な値は `rgcn` です。ナレッジグラフの有効値は、`kge`、`transe`、`distmult`、および `rotate` です。

- neptuneIamRoleArn (CLI では: `--neptune-iam-role-arn`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  SageMaker および Amazon S3 リソースへの Neptune アクセスを提供する IAM ロールの ARN。これは DB クラスターパラメータグループに一覧表示されている必要があり、そうでない場合、エラーがスローされます。

- update (CLI では: `--update`) — タイプ `boolean` のブール値 (ブール値 (真または偽) の値)。

`true` に設定された場合、`update` は、これが更新リクエストであることを示します。デフォルトは `false` です。`mlModelTrainingJobId` または `mlModelTransformJobId` のいずれかを指定する必要があります。

- volumeEncryptionKMSKey (CLI では: `--volume-encryption-kms-key`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  トレーニングジョブを実行する ML コンピューティングインスタンスにアタッチされたストレージボリュームのデータを暗号化するために SageMaker が使用する Amazon キー管理サービス (Amazon KMS) キー。デフォルトは [なし] です。

レスポンス

- arn — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  新しい推論エンドポイントの ARN。

- creationTimeInMillis — `long` 型の長整数(符号付き 64 ビット整数)。

  エンドポイントの作成時間 (ミリ秒単位)。

- id — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  新しい推論エンドポイントの一意 ID。

エラー

- UnsupportedOperationException
- BadRequestException
- MLResourceNotFoundException
- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException

- TooManyRequestsException

ListMLEndpoints (アクション)

この API の AWS CLI 名は `list-ml-endpoints` です。

既存の推論エンドポイントを一覧表示します。「エンドポイントコマンドを使用して推論エンドポイントを管理する」を参照してください。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:ListMLEndpoints IAM アクションを許可するポリシーがアタッチされている必要があります。

リクエスト

- maxItems (CLI では: `--max-items`) — ListMLEndpointsInputMaxItemsInteger、タイプ `integer` (符号付き 32 ビット整数)、1 以上か 1024 以上？ st? s。

  返される項目の最大数 (1 ~ 1024、デフォルトは 10)。
- neptuneIamRoleArn (CLI では: `--neptune-iam-role-arn`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  SageMaker および Amazon S3 リソースへの Neptune アクセスを提供する IAM ロールの ARN。これは DB クラスターパラメータグループに一覧表示されている必要があります。そうでない場合はエラーが発生します。

レスポンス

- ids — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  推論エンドポイント ID のリストにあるページ。

エラー

- UnsupportedOperationException
- BadRequestException
- MLResourceNotFoundException
- InvalidParameterException

- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

GetMLEndpoint (アクション)

この API の AWS CLI 名は `get-ml-endpoint` です。

推論エンドポイントに関する詳細を取得します。「エンドポイントコマンドを使用して推論エンドポイントを管理する」を参照してください。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:GetMLEndpointStatus IAM アクションを許可するポリシーがアタッチされている必要があります。

リクエスト

- id (CLI では: `--id`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  推論エンドポイントの一意識別子。
- neptuneIamRoleArn (CLI では: `--neptune-iam-role-arn`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  SageMaker および Amazon S3 リソースへの Neptune アクセスを提供する IAM ロールの ARN。これは DB クラスターパラメータグループに一覧表示されている必要があります。そうでない場合はエラーが発生します。

レスポンス

- endpoint – MlResourceDefinition オブジェクト。

  エンドポイント定義。
- endpointConfig – MlConfigDefinition オブジェクト。

エンドポイントの設定

- id — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  推論エンドポイントの一意識別子。

- status — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  推論エンドポイントのステータス。

エラー

- UnsupportedOperationException
- BadRequestException
- MLResourceNotFoundException
- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

DeleteMLEndpoint (アクション)

この API の AWS CLI 名は `delete-ml-endpoint` です。

Neptune ML 推論エンドポイントの作成をキャンセルします。「エンドポイントコマンドを使用して推論エンドポイントを管理する」を参照してください。

IAM 認証が有効になっている Neptune クラスターでこの操作を呼び出すときには、リクエストを行う IAM ユーザーまたはロールに、そのクラスターで neptune-db:DeleteMLEndpoint IAM アクションを許可するポリシーがアタッチされている必要があります。

リクエスト

- clean (CLI では: `--clean`) — タイプ `boolean` のブール値 (ブール値 (真または偽) の値)。

  このフラグが `TRUE` に設定された場合、すべての Neptune ML S3 アーティファクトは、ジョブが停止したときに削除される必要があります。デフォルトは `FALSE` です。

- id (CLI では: `--id`) — 必須: `string` タイプの文字列 (UTF-8 でエンコードされた文字列)。

  推論エンドポイントの一意識別子。

- neptuneIamRoleArn (CLI では: `--neptune-iam-role-arn`) — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  SageMaker および Amazon S3 リソースへの Neptune アクセスを提供する IAM ロールの ARN。これは DB クラスターパラメータグループに一覧表示されている必要があり、そうでない場合、エラーがスローされます。

レスポンス

- status — タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  キャンセルのステータス。

エラー

- UnsupportedOperationException
- BadRequestException
- MLResourceNotFoundException
- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

Neptune データプレーン API の例外

例外:

- AccessDeniedException (構造)
- BadRequestException (構造)
- BulkLoadIdNotFoundException (構造)
- CancelledByUserException (構造)
- ClientTimeoutException (構造)
- ConcurrentModificationException (構造)
- ConstraintViolationException (構造)
- ExpiredStreamException (構造)
- FailureByQueryException (構造)
- IllegalArgumentException (構造)
- InternalFailureException (構造)
- InvalidArgumentException (構造)
- InvalidNumericDataException (構造)
- InvalidParameterException (構造)
- LoadUrlAccessDeniedException (構造)
- MalformedQueryException (構造)
- MemoryLimitExceededException (構造)
- MethodNotAllowedException (構造)
- MissingParameterException (構造)
- MLResourceNotFoundException (構造)
- ParsingException (構造)
- PreconditionsFailedException (構造)
- QueryLimitExceededException (構造)
- QueryLimitException (構造)
- QueryTooLargeException (構造)
- ReadOnlyViolationException (構造)
- S3Exception (構造)

- ServerShutdownException (構造)
- StatisticsNotAvailableException (構造)
- StreamRecordsNotFoundException (構造)
- ThrottlingException (構造)
- TimeLimitExceededException (構造)
- TooManyRequestsException (構造)
- UnsupportedOperationException (構造)
- UnloadUrlAccessDeniedException (構造)

AccessDeniedException (構造)

認証または承認に失敗した場合にレイズされます。

フィールド

- code - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。
- detailedMessage - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題のリクエストの ID。

BadRequestException (構造)

処理できないリクエストが送信されたときにレイズされます。

フィールド

- code - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。
- detailedMessage - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

問題を説明する詳細メッセージ。

- requestId - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  正しくないリクエストの ID。

BulkLoadIdNotFoundException (構造)

指定された一括読み込みジョブ ID が見つからないときにレイズされます。

フィールド

- code - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。
- detailedMessage - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  見つからなかった一括読み込みジョブ ID。

CancelledByUserException (構造)

ユーザーがリクエストをキャンセルしたときにレイズされます。

フィールド

- code - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。
- detailedMessage - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題のリクエストの ID。

ClientTimeoutException (構造)

クライアントでリクエストがタイムアウトになったときにレイズされます。

フィールド

- code - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。
- detailedMessage - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題のリクエストの ID。

ConcurrentModificationException (構造)

リクエストが変更しようとしたデータが、別のプロセスによって同時に変更されているときにレイズされます。

フィールド

- code - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。
- detailedMessage - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題のリクエストの ID。

ConstraintViolationException (構造)

リクエストフィールドの値が必要な制約を満たさないときにレイズされます。

フィールド

- code - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。
- detailedMessage - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題のリクエストの ID。

ExpiredStreamException (構造)

リクエストが期限切れのストリームにアクセスしようとしたときにレイズされます。

フィールド

- code - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。
- detailedMessage - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題のリクエストの ID。

FailureByQueryException (構造)

リクエストが失敗したときにレイズされます。

フィールド

- code - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。

- detailedMessage - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題のリクエストの ID。

IllegalArgumentException (構造)

リクエスト内の引数がサポートされていないときにレイズされます。

フィールド

- code - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。
- detailedMessage - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題のリクエストの ID。

InternalFailureException (構造)

リクエストの処理が予期せず失敗したときにレイズされます。

フィールド

- code - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。
- detailedMessage - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

問題のリクエストの ID。

InvalidArgumentException (構造)

リクエスト内の引数が無効な値を持つときにレイズされます。

フィールド

- code - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。
- detailedMessage - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題のリクエストの ID。

InvalidNumericDataException (構造)

リクエストを処理しているときに無効な数値データが見つかったときにレイズされます。

フィールド

- code - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。
- detailedMessage - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題のリクエストの ID。

InvalidParameterException (構造)

パラメータ値が無効なときにレイズされます。

フィールド

- code - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。
- detailedMessage - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  無効なパラメータを含むリクエストの ID。

LoadUrlAccessDeniedException (構造)

指定されたロード URL へのアクセスが拒否されたときにレイズされます。

フィールド

- code - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。
- detailedMessage - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題のリクエストの ID。

MalformedQueryException (構造)

構文的に正しくないか、追加の検証に合格しなかったクエリが送信されたときにレイズされます。

フィールド

- code - これは 必須です。タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。
- detailedMessage - これは 必須です。タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  不正な形式のクエリリクエストの ID。

MemoryLimitExceededException (構造)

メモリリソースが不足しているためにリクエストが失敗したときにレイズされます。リクエストは再試行できます。

フィールド

- code - これは 必須です。タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。
- detailedMessage - これは 必須です。タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  失敗したリクエストの ID。

MethodNotAllowedException (構造)

リクエストによって使用された HTTP メソッドが、使用中のエンドポイントでサポートされていないときにレイズされます。

フィールド

- code - これは 必須です。タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

例外とともに返される HTTP ステータスコード。

- detailedMessage - これは 必須です。タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。

- requestId - これは 必須です。タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  問題のリクエストの ID。

MissingParameterException (構造)

必須パラメータがないときにレイズされます。

フィールド

- code - これは 必須です。タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。

- detailedMessage - これは 必須です。タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。

- requestId - これは 必須です。タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  パラメータが欠落しているリクエストの ID。

MLResourceNotFoundException (構造)

指定された機械学習リソースが見つからなかったときにレイズされます。

フィールド

- code - これは 必須です。タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。

- detailedMessage - これは 必須です。タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

問題を説明する詳細メッセージ。

- requestId - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題のリクエストの ID。

ParsingException (構造)

解析の問題が検出されたときにレイズされます。

フィールド

- code - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。
- detailedMessage - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題のリクエストの ID。

PreconditionsFailedException (構造)

リクエストを処理するための前提条件が満たされていないときにレイズされます。

フィールド

- code - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。
- detailedMessage - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題のリクエストの ID。

QueryLimitExceededException (構造)

アクティブなクエリの数がサーバーが処理できる数を超えているときにレイズされます。問題のクエリは、システムの負荷が少なくなったときに再試行できます。

フィールド

- code - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。
- detailedMessage - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  制限を超えたリクエストの ID。

QueryLimitException (構造)

クエリのサイズがシステム制限を超えているときにレイズされます。

フィールド

- code - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。
- detailedMessage - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  制限を超えたリクエストの ID。

QueryToolLargeException (構造)

クエリの本文が大きすぎるときにレイズされます。

フィールド

- code - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。
- detailedMessage - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  大きすぎるリクエストの ID。

ReadOnlyViolationException (構造)

リクエストが読み取り専用リソースに書き込もうとしたときにレイズされます。

フィールド

- code - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。
- detailedMessage - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  パラメータが欠落しているリクエストの ID。

S3Exception (構造)

Amazon S3 へのアクセスに問題があるときにレイズされます。

フィールド

- code - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。

- detailedMessage - これは 必須です。タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  問題のリクエストの ID。

ServerShutdownException (構造)

リクエストの処理中にサーバーがシャットダウンしたときにレイズされます。

フィールド

- code - これは 必須です。タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。
- detailedMessage - これは 必須です。タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  問題のリクエストの ID。

StatisticsNotAvailableException (構造)

リクエストを満たすのに必要な統計情報がないときにレイズされます。

フィールド

- code - これは 必須です。タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。
- detailedMessage - これは 必須です。タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ string の文字列 (UTF-8 でエンコードされた文字列)。

問題のリクエストの ID。

StreamRecordsNotFoundException (構造)

クエリによって要求されたストリームレコードが見つからないときにレイズされます。

フィールド

- code - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。
- detailedMessage - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題のリクエストの ID。

ThrottlingException (構造)

リクエストの速度が最大スループットを超えているときにレイズされます。この例外が発生した後、リクエストを再試行できます。

フィールド

- code - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。
- detailedMessage - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この理由で処理できなかったリクエストの ID。

TimeLimitExceededException (構造)

操作が許可されている時間制限を超えるとレイズされます。

フィールド

- code - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。
- detailedMessage - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この理由で処理できなかったリクエストの ID。

TooManyRequestsException (構造)

処理中のリクエストの数が制限を超えるとレイズされます。

フィールド

- code - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。
- detailedMessage - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  この理由で処理できなかったリクエストの ID。

UnsupportedOperationException (構造)

サポートされていない操作をリクエストが開始しようとしたときにレイズされます。

フィールド

- code - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。
- detailedMessage - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題のリクエストの ID。

UnloadUrlAccessDeniedException (構造)

アンロード対象の URL へのアクセスが拒否されたときにレイズされます。

フィールド

- code - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  例外とともに返される HTTP ステータスコード。
- detailedMessage - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題を説明する詳細メッセージ。
- requestId - これは 必須です。タイプ `string` の文字列 (UTF-8 でエンコードされた文字列)。

  問題のリクエストの ID。

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。