Amazon Neptune 用の AWS Well-Architected フレームワークの適用

AWS 規範ガイダンス

AWS 規範ガイダンス: Amazon Neptune 用の AWS Well-Architected フレームワークの適用

Table of Contents

Amazon Neptune 用の AWS Well-Architected フレームワークの適用

Amazon Web Services (寄稿者)

2026 年 1 月 (ドキュメント履歴)

Amazon Neptune を使用して、Amazon Web Services (AWS) でグラフベースのソリューションを構築できます。このガイドは、Neptune のデプロイ計画にあたり AWS Well-Architected フレームワークの原則を適用するための規範ガイダンスを提供します。

AWS Well-Architected フレームワークは、さまざまなアプリケーションやワークロード向けに、安全で高性能、耐障害性、効率的なインフラストラクチャを構築するのに役立ちます。また、アーキテクチャを評価し、スケーラブルな設計を実装するための一貫したアプローチも提供します。

AWS Well-Architected フレームワークは、次の 6 つの柱を中心に構築されています。

- オペレーショナルエクセレンス
- セキュリティ
- 信頼性
- パフォーマンス効率
- コスト最適化
- 持続可能性

このガイドでは、 AWS Well-Architected フレームワークの設計の柱とベストプラクティス、および に Neptune をデプロイする際の考慮事項について説明します AWS。

対象者

このガイドは、 AWSでグラフを使用するソリューションを設計および実装するデータエンジニア、ソリューションアーキテクト、データアナリストを対象としています。

目的

このガイドは、お客様や組織が以下のことを行うのに役立ちます。

- ユースケースとクエリパターンに基づいて、サポートされているデプロイオプションとクエリ言語から選択します。
- 耐障害性とセキュリティの向上に役立つ AWS Well-Architected 設計パターンに従ってください。
- 最適なパフォーマンスとコスト削減のためのクエリを設計します。
- 本番環境で Neptune クラスターを管理する際に運用効率を高める方法について学びます。

運用上の優秀性の柱

AWS Well-Architected フレームワークの運用上の優秀性の柱は、システムの実行とモニタリング、プロセスと手順の継続的な改善に焦点を当てています。開発をサポートし、ワークロードを効率的に実行し、運用に関するインサイトを得て、ビジネス価値をもたらすサポートプロセスと手順を継続的に改善する能力が含まれます。人間の介入なしにほとんどの問題を検出して修復する自己修復ワークロードによって、運用上の複雑さを軽減できます。このセクションで説明されているベストプラクティスに従うことで、この目標を達成できます。Amazon Neptune メトリクス、API、各種メカニズムを使用して、ワークロードが予想される動作から逸脱した場合に適切に対応します。

運用上の優秀性の柱に関するこの説明では、以下の主要分野に焦点を当てています。

- Infrastructure as code (IaC)
- 変更管理
- レジリエンシー戦略
- インシデント管理
- コンプライアンスのための監査レポート
- ログ記録とモニタリング

IaC アプローチを使用してデプロイを自動化する

IaC を使用して Neptune でのデプロイを自動化するためのベストプラクティスは次のとおりです。

- 可能な限り、Infrastructure as Code (IaC) を適用して Neptune クラスターをデプロイします。一貫した環境設定を行うため、クラスターに必要なすべてのリソースを作成するには AWS CloudFormation テンプレート、AWS Cloud Development Kit (AWS CDK)、または HashiCorp Terraform を使用します。
- インスタンスのサイズ変更、リードレプリカの追加または削除、グローバルテーブルでの手動フェイルオーバーなど、Neptune の運用手順を可能な限り自動化します。
- 接続文字列をクライアントの外部に保存します。ブルー/グリーンデプロイ戦略、ディザスタリカバリ (DR)、新しいクラスターへのほぼゼロのダウンタイムでの移行を容易にするため、抽出、変換、ロード (ETL) プロセスを使用します。接続文字列は、AWS Secrets Manager、Amazon DynamoDB、または動的に変更できる任意の場所に保存できます。
- タグを使用してメタデータを Neptune リソースに追加し、タグに基づいて使用量を追跡します。詳細については、「Amazon Neptune リソースのタグ付け」を参照してください。

小規模かつ可逆的な変更を頻繁に行う

以下の推奨事項は、複雑さを最小限に抑え、ワークロードの中断の可能性を減らすための、小規模かつ可逆的な変更に焦点を当てています。

- IaC テンプレートとスクリプトを GitHub や GitLab といったソースコントロールサービスに保存します。

  > **Important**
  >
  > ソース管理に AWS 認証情報を保存しないでください。

- IaC デプロイでは、AWS CodePipeline や AWS CodeBuild などの継続的インテグレーションおよび継続的デリバリー (CI/CD) サービスを使用する必要があります。これらのサービスは、本番環境の Amazon Neptune クラスターに影響を与える前に、一時的な Neptune クラスターを含む非本番環境でコードをコンパイル、テスト、デプロイします。
- 本番環境にデプロイする前に、インフラストラクチャとアプリケーションのクエリをより低い環境でテストします。これにより、中断の可能性が最小限に抑えられ、ワークロードやスケールに合わせて適切に動作させることができます。

失敗を予測する

自己修復インフラストラクチャは、障害を予測し、介入なしで問題解決を試みることで、運用上の優秀性を実証します。以下の推奨事項は、Neptune でその成熟度を実現するのに役立ちます。

- Amazon CloudWatch メトリクスを使用して DB インスタンスの CPU とメモリの使用状況をモニタリングし、使用状況のパターンを理解するモニタリングプランを作成します。アプリケーションログにある主要なメトリクスと Neptune クライアントレスポンスを確認するための CloudWatch ダッシュボードとアラームを作成します。CPU 使用率の高低を示す指標の詳細については、Neptune ドキュメントの「Using CloudWatch to monitor DB instance performance in Neptune」を参照してください。

  クエリでout-of-memory例外が頻繁に発生する場合は、クエリが通過するノードの合計数を減らすか、RAM-to-CPU の比率が高い X2ファミリーのインスタンスの使用を検討してください。
- Neptune クラスターの状態をモニタリングするために通知を設定します。例えば、`BufferCacheHitRatio` は常に高い (99.9% を超える) 必要があります

が、MainRequestQueuePendingRequests は常に低い (理想的には 0 ですが、要件とレイテンシーの許容値によって異なります) 必要があります。

- Neptune 内で高可用性を実現するには、リードレプリカの使用を検討してください。フェイルオーバーイベント中にインスタンスが常に読み取りクエリを処理できるように、ライターインスタンスとは異なるアベイラビリティーゾーンに少なくとも 2 つのリードレプリカが必要です。
- 使用率メトリクスに基づいてリードレプリカを自動的にスケーリングします。詳細については、「Amazon Neptune DB クラスター内のレプリカの数の Auto-scaling」を参照してください。
- DB インスタンスのフェイルオーバーをテストすることで、そのプロセスでユースケースにかかる時間を把握します。
- アプリケーションが完全に AWS リージョン 停止した場合、DR プランの一部としてグローバルデータベースを使用することを検討してください。

すべての運用上の障害から学ぶ

自己修復インフラストラクチャは、まれな問題が発生したり、対応が意図したような効果を出さなかったりするたびに反復的に進化していく長期的な取り組みです。以下のプラクティスを採用することで、その目標に焦点を当てることができます。

- すべての障害から学ぶことで改善を推進します。
- チームと組織で教訓を共有します。組織内の複数のチームが Neptune を使用している場合は、共通のチャットルームまたはユーザーグループを作成して、学習とベストプラクティスを共有します。

ログ機能を使用して、不正または異常なアクティビティをモニタリングする

異常なパフォーマンスやアクティビティのパターンを観察するには、Amazon CloudWatch Logs にログを保存します。以下のベストプラクティスを考慮します。

- スロークエリロギングを有効にします。ログを定期的に確認し、特定のクエリが遅い理由を診断します。Gremlin、SPARQL、または openCypher 向けの Neptune の explain エンドポイントと profile エンドポイントを使用して、これらのクエリが遅い理由に関するインサイトを取得します。
- Neptune 監査ログを有効にし、ログに不正アクセスや異常がないか定期的に確認します。

- スロークエリログ記録または監査ログ記録を使用している場合は、CloudWatch Logs への発行を有効にします。これにより、インスタンスのディスク容量が不足するのを防ぐことができます。Neptune インスタンスのログストレージ容量は限られており、ログ容量を超えると古いログファイルは上書きされます。CloudWatch Logs は、ログの長期保持をサポートしています。CloudWatch Logs の強化されたモニタリング機能を使用すると、ログをクエリして問題を診断する能力が向上します。
- 監査ログの分析ツールを改善するため、監査ログデータを CloudWatch Logs のロググループに発行するように Neptune DB クラスターを設定できます。CloudWatch Logs を使用すると、ログデータのリアルタイム分析、CloudWatch を使用したアラームの作成およびメトリクスの表示、CloudWatch Logs を使用した、耐久性に優れたストレージへのログレコードの保存を行うことができます。詳細については、「Publishing Neptune logs to Amazon CloudWatch Logs」を参照してください。
- Neptune は、 AWS CloudTrailを使用したコントロールプレーンのアクションのログ記録をサポートしています。詳細については、「 を使用した Amazon Neptune API コールのログ記録 AWS CloudTrail」を参照してください。

セキュリティの柱

のクラウドセキュリティが最優先事項 AWS です。お客様は AWS 、セキュリティを最も重視する組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャを活用できます。

セキュリティは、 AWS お客様とお客様の間の責任共有です。責任共有モデルでは、これをクラウドのセキュリティおよびクラウド内のセキュリティとして説明しています。

- クラウドのセキュリティ – AWS は、 AWS のサービス で実行されるインフラストラクチャを保護する責任を担います AWS クラウド。 は、お客様が安全に使用できるサービス AWS も提供します。サードパーティーの監査者は、AWS コンプライアンスプログラムの一環として、 AWS セキュリティの有効性を定期的にテストおよび検証します。Amazon Neptune に適用されるコンプライアンスプログラムについては、コンプライアンスプログラムによるAWS 対象範囲内のサービスを参照してください。
- クラウドのセキュリティ – お客様の責任は、使用する によって決まり AWS のサービス ます。また、お客様は、お客様のデータの機密性、企業の要件、および適用可能な法律および規制などの他の要因についても責任を担います。データプライバシーの詳細については、「データプライバシーのよくある質問」を参照してください。欧州でのデータ保護の詳細については、「AWS Shared Responsibility Model and GDPR」のブログ記事を参照してください。

セキュリティの柱は、Neptune を使用する際に責任共有モデルを適用する方法を理解するのに役立ちます。以下のトピックでは、セキュリティおよびコンプライアンスの目的を達成するために Neptune を設定する方法を示します。また、Neptune リソースのモニタリングと保護 AWS のサービス に役立つ他の の使用方法についても説明します。

セキュリティの柱には、以下の主要な重点分野が含まれています。

- データセキュリティ
- ネットワークセキュリティ
- 認証と認可

データセキュリティの実装

データ漏洩やデータ侵害は顧客を危険にさらし、会社に大きな悪影響を与える可能性があります。以下のベストプラクティスは、顧客データを不注意による漏洩や悪意のある漏洩から保護するのに役立ちます。

- クラスター名、タグ、パラメータグループ、 AWS Identity and Access Management (IAM) ロール、およびその他のメタデータには、請求ログや診断ログに表示される可能性があるため、機密情報や機密情報を含めないでください。
- Neptune にデータとして保存されている外部サーバーへの URI またはリンクには、リクエストを検証するための認証情報を含めないでください。
- Neptune の暗号化されたインスタンスは、基になるストレージへの不正アクセスからデータを保護することで、データ保護のレイヤーを追加します。Neptune の暗号化を使用すると、クラウドにデプロイされたアプリケーションのデータ保護を強化できます。Neptune 暗号化は、保管中のデータのコンプライアンス要件を満たすために使用することもできます。

  新しい Neptune DB インスタンスの暗号化を有効にするには、Neptune コンソールの [暗号化の有効化] セクションで [はい] (デフォルトで選択) を選択するか、 CloudFormationで AWS::Neptune::DBCluster::StorageEncrypted プロパティを設定します。暗号化が有効になっている場合、Neptune はデフォルトで Amazon Relational Database Service (Amazon RDS) AWS マネージドキーを使用します。そうでない場合、カスタマーマネージドキーを作成できます。Neptune DB インスタンスの作成については、「Creating a new Neptune DB cluster」を参照してください。詳細については、「保管時の Neptune リソースの暗号化」を参照してください。自動スナップショットと手動スナップショットは、Neptune クラスターに選択したのと同じ暗号化を使用します。
- SPARQL 言語と openCypher 言語を使用する場合は、SQL インジェクションやその他の形式の攻撃を防ぐために、適切な入力検証手法およびパラメータ化の手法を実践してください。ユーザー指定の入力による文字列連結を使用するクエリは構築しないでください。パラメータ化されたクエリまたはプリペアドステートメントを使用して、入力パラメータをグラフデータベースに安全に渡します。詳細については、「openCypher のパラメーター化されたクエリの例」および「SPARQL インジェクション防御」を参照してください。
- Gremlin 言語の場合、潜在的なインジェクションの問題を回避するために、文字列ベースの Gremlin スクリプトを直接渡すのではなく Gremlin 言語バリアントを使用します。

ネットワークを保護する

Amazon Neptune DB クラスターは、 AWS上の仮想プライベートクラウド (VPC) でのみ作成できます。Neptune 1.4.6.0 まで、Neptune DB クラスターのエンドポイントには、その VPC 内でのみアクセスできました。Neptune 1.4.6.0 以降では、Neptune インスタンスをインターネット経由でパブリックにアクセスできるように設定できます。開発者が Neptune に簡単にアクセスできるようにするには、この機能を非本番環境でのみ使用することがベストプラクティスです (ただし、パブリックアクセシビリティを有効にするには で IAM 認証が常に必要です) 。パブリックアクセシビリティ

が有効になっている場合は、データベースポートのインバウンドセキュリティグループルールを既知の IP アドレストラフィックのみに設定することを検討してください。本番環境または機密データを含むクラスターでは、パブリックアクセシビリティを許可せず、Neptune DB クラスターがある VPC へのアクセスを制限することで、Neptune データを保護します。詳細については、「Amazon Neptune へのグラフへの接続」を参照してください。

転送中のデータを保護するために、Neptune は安全なプロトコルと暗号を使用して、HTTPS を介した SSL 接続を任意のインスタンスまたはクラスターエンドポイントに強制します。Neptune は Neptune DB インスタンスの SSL 証明書を提供します。Neptune SSL 証明書は、クラスターエンドポイント、リーダーエンドポイント、インスタンスエンドポイントのホスト名のみをサポートします。

ロードバランサーまたはプロキシサーバー (HAProxy など) を使用している場合は、SSL ターミネーションを使用して独自の SSL 証明書をプロキシサーバーに保存する必要があります。提供された SSL 証明書はプロキシサーバーのホスト名と一致しないため、SSL パススルーは機能しません。SSL を使用して Neptune エンドポイントに接続する方法の詳細については、Using the HTTP REST endpoint to connect to a Neptune DB instance」を参照してください。

認証と認可を実装する

Neptune DB クラスターと DB インスタンスで Neptune 管理 アクションを実行できるユーザーを制御するには、IAM データベース認証を有効にし、IAM 認証情報を使用します。IAM 認証情報を使用して AWS に接続するとき、IAM ロールには、Neptune 管理操作を実行するためのアクセス許可を付与する IAM ポリシーが必要です。最小特権の原則に従って、タスクを完了するために必要なアクセス許可のみを付与してください。詳細については、「Neptune へのアクセスを制御するためのさまざまな種類の IAM ポリシーの使用」および「一時的な認証情報を使用した IAM 認証」を参照してください。

Neptune クラスターに接続してデータをクエリできるユーザーを制御するには、IAM を使用して Neptune DB インスタンスまたは DB クラスターを認証します。Neptune DB クラスターで IAM 認証を有効にした場合、DB クラスターにアクセスするすべてのユーザーを最初に認証する必要があります。詳細については、「Enabling IAM database authentication in Neptune」を参照してください。

IAM データベース認証が有効になっている場合、各リクエストは AWS 署名バージョン 4 を使用して署名する必要があります。IAM 認証が有効になっているすべての Neptune エンドポイントに署名付きリクエストを送信する方法については、「Connecting and Signing with AWS Signature Version 4」を参照してください。awscurl などの多くのライブラリやツールは、 AWS 署名バージョン 4 を既にサポートしています。

他の とやり取りするために AWS のサービス、Amazon Neptune は IAM サービスにリンクされたロールを使用します。サービスにリンクされたロールは、Neptune に直接リンクされた一意のタイプの IAM ロールです サービスリンクロールは、Neptune によって事前定義されており、サービスがユーザーに代わって他の AWS のサービス を呼び出すために必要なすべてのアクセス許可が含まれています。詳細については、「Using Service-Linked Roles for Neptune」を参照してください。

信頼性の柱

信頼性の柱には、ワークロードが意図した機能を正しく一貫して実行する能力が含まれます。これには、ワークロードのライフサイクル全体を通じてワークロードを運用およびテストする能力が含まれます。

信頼性の高いワークロードの実現は、ソフトウェアとインフラストラクチャの両方について事前に設計を決定することから始まります。アーキテクチャの選択は、 AWS Well-Architected のすべての柱にわたってワークロードの動作に影響します。高い信頼性を保つには、特定のパターンに従う必要があります。

信頼性の柱は、次の主要分野に焦点を当てています。

- サービスクォータとデプロイパターンを含むワークロードアーキテクチャ
- 変更管理
- 障害管理

Neptune サービスクォータを理解する

Neptune クラスターボリュームは、クォータが 64 TiB である中国と GovCloud を除くすべてのサポート対象の AWS リージョン で、最大 128 tebibytes (TiB) のサイズまで増やすことができます。

128 TiB のクォータでは、グラフに約 2,000～4,000 億個のオブジェクトを十分に保存することができます。ラベル付きプロパティグラフ (LPG) では、オブジェクトはノード、エッジ、またはノードもしくはエッジ上のプロパティです。Resource Description Framework (RDF) グラフでは、オブジェクトは四角形です。

任意の Neptune サーバーレスクラスターでは、Neptune 容量ユニット (NCU) の最小数と最大数の両方を設定します。各 NCU は、2 ギビバイト (GiB) のメモリと、関連する vCPU およびネットワークで構成されます。最小および最大 NCU 値は、クラスター内のサーバーレスインスタンスに適用されます。設定できる最高の最大 NCU 値は 128.0 NCU であり、最低の最小値は 1.0 NCU です。Amazon CloudWatch メトリクス `ServerlessDatabaseCapacity` および `NCUUtilization` を監視し、一般的に実行される範囲をキャプチャしてその範囲内の望ましくない動作やコストを関連付けることで、アプリケーションに最適な NCU 範囲を最適化します。多くのワークロードでは、1.0 NCU の開始点が低すぎるため、非アクティブ状態が続くと動作の信頼性が低下します。ワークロードが十分に速くスケーリングしない場合は、スケーリング中の最初のサージに対して十分な処理を提供するように最小 NCU を増やします。

各 AWS アカウント には、作成できるデータベースリソースの数に対する各リージョンのクォータがあります。これらのリソースには、DB インスタンスと DB クラスターが含まれます。リソースの制限に達すると、リソースを作成するための追加の呼び出しは、例外が発生して失敗します。一部のクォータはソフトクォータで、リクエストにより引き上げることができます。Amazon Neptune と Amazon RDS、Amazon Aurora、Amazon DocumentDB (MongoDB 互換) の間で共有されるクォータのリストと、クォータの引き上げをリクエストするためのリンク (利用可能な場合) については、「Amazon RDS のクォータ」を参照してください。

Neptune デプロイパターンを理解する

Neptune DB クラスターには、1 つのプライマリ DB インスタンスと最大 15 の Neptune レプリカがあります。プライマリ DB インスタンスでは、読み書きオペレーションをサポートしており、クラスターボリュームに対してすべてのデータ変更を実行します。Neptune レプリカは、プライマリ DB インスタンスと同じストレージボリュームに接続し、読み取りオペレーションのみサポートしています。Neptune レプリカは、プライマリ DB インスタンスから読み取りワークロードをオフロードします。

高可用性を実現するには、リードレプリカを使用します。異なるアベイラビリティーゾーンで 1 つ以上のリードレプリカインスタンスを利用可能にすると、リードレプリカがプライマリインスタンスのフェイルオーバーターゲットとして機能するため、可用性が向上します。ライターインスタンスが失敗した場合、Neptune はリードレプリカインスタンスをプライマリインスタンスに昇格します。そうすると、プライマリインスタンスに対して行われた読み取り要求と書き込み要求が失敗して例外が発生すると、短時間の中断 (通常は 30 秒未満) が発生し、昇格したインスタンスは再起動されます。最高の信頼性を得るには、2 つのリードレプリカを別々のアベイラビリティーゾーンに使用することを検討してください。アベイラビリティーゾーン 1 のプライマリインスタンスがオフラインになった場合、アベイラビリティーゾーン 2 のインスタンスはプライマリに昇格されますが、その間はクエリを処理できません。したがって、移行中に読み取りクエリを処理するには、アベイラビリティーゾーン 3 のインスタンスが必要です。

Neptune サーバーレスを使用している場合、すべてのアベイラビリティーゾーンのリーダーインスタンスとライターインスタンスは、データベースの負荷に応じて、互いに独立してスケールアップおよびスケールダウンします。リーダーインスタンスの昇格階層を 0 または 1 に設定すると、ライターインスタンスの容量に合わせてスケールアップまたはスケールダウンできます。これにより、現在のワークロードをいつでも引き継ぐ準備が整います。

アプリケーションにグローバルなフットプリントがある場合、またはマルチリージョンフェイルオーバーが必要な場合は、Neptune グローバルデータベースの使用を検討してください。Amazon Neptune グローバルデータベースは複数のデータベースにまたがり AWS リージョン、低レイテン

シーのグローバル読み取りを可能にし、停止が 全体に影響を与えるまれなケースで高速リカバリを提供します AWS リージョン。Neptune グローバルデータベースは、1 つのリージョンにあるプライマリ DB クラスターと、異なるリージョンにある最大 5 つのセカンダリ DB クラスターで構成されます。

Neptune クラスターの管理とスケーリング

Neptune 自動スケーリングを使用すると、DB クラスター内の Neptune レプリカの数を自動的に調整して、CPU 利用率のしきい値に基づいて接続およびワークロードの要件を満たすことができます。自動スケーリングを使用すると、Neptune DB クラスターはワークロードの急激な増加を処理できます。ワークロードが減少すると、自動スケーリングは不要なレプリカを削除し、未使用の容量に対して料金が発生しないようにします。新しいインスタンスの起動には最大 15 分かかる場合があるため、自動スケーリングだけでは需要の急激な変化に対する十分なソリューションにはならないことに注意してください。

自動スケーリングは、既に 1 つのプライマリライターインスタンスと、少なくとも 1 つのリードレプリカインスタンスを持つ Neptune DB クラスターでのみ使用できます (「Amazon Neptune DB Clusters and Instances」を参照)。また、クラスター内のすべての read-replica インスタンスが使用可能な状態である必要があります。リードレプリカが使用可能以外の状態にある場合、クラスター内のすべてのリードレプリカが使用可能になるまで、Neptune 自動スケーリングは何もしません。

需要が急速に変化する場合は、サーバーレスインスタンスの使用を検討してください。サーバーレスインスタンスは短期間で垂直方向にスケールできる一方、自動スケーリングは長期間にわたり水平方向にスケールできます。この設定では、サーバーレスインスタンスが垂直方向にスケールし、自動スケーリングは新しいリードレプリカをインスタンス化して、単一のサーバーレスインスタンスの最大容量を超えるワークロードを処理するため、最適なスケーラビリティを提供します。Amazon Neptune サーバーレスの容量スケーリングの詳細については、「Neptune Serverless DB クラスタの容量スケーリング」を参照してください。

スケーリングのニーズが予測可能なタイミングで変化する場合は、最小インスタンス、最大インスタンス、しきい値に対する変更をスケジュールして、そうした変化するニーズをより適切に処理することができます。必要に応じて、少なくとも 15 分前にスケールアウトイベントをスケジュールして、インスタンスがオンラインになるようにしてください。

パラメータグループのパラメータを使用して、Amazon Neptune のデータベース設定を管理します。パラメータグループは、1 つ以上の DB インスタンスに適用されるエンジン設定値のコンテナとして機能します。パラメータグループのクラスターパラメータを変更するときは、静的パラメータと動的パラメータの違いと、それらを適用する方法とタイミングを理解してください。ステータスエンドポイントを使用して、現在適用されている設定を確認します。

バックアップとフェイルオーバーイベントを管理する

Neptune は、クラスターボリュームを自動的にバックアップし、バックアップ保持期間中、バックアップしたデータを保持します。Neptune のバックアップは連続的かつ増分的であるため、バックアップ保持期間内の任意の時点にすばやく復元できます。DB クラスターを作成または変更するときに、バックアップ保持期間を 1～35 日の間で指定できます。

バックアップ保持期間を超えたバックアップを保持するには、クラスターボリュームの中にデータのスナップショットを作成できます。スナップショットの保存には、Neptune の標準ストレージ料金がかかります。

DB クラスターの Amazon Neptune のスナップショットを作成すると、Neptune はクラスターのストレージボリュームのスナップショットを作成し、個々のインスタンスだけではなく、すべてのデータをバックアップします。新しい DB クラスターは、この DB クラスタースナップショットから復元することで後に作成できます。DB クラスターを復元する場合は、復元の元となる DB クラスタースナップショットの名前を指定し、復元によって作成される新しい DB クラスターの名前を指定します。

フェイルオーバーイベントに対するシステムの応答をテストします。Neptune API を使用してフェイルオーバーイベントを強制します。フェイルオーバーによる再起動が便利なのは、テスト用の DB インスタンスで障害をシミュレートするときや、フェイルオーバーの実行後にオペレーションを元のアベイラビリティーゾーンに復元するときです。詳細については、「マルチ AZ 配置の設定と管理」を参照してください。DB ライタークラスターを再起動すると、そのスタンバイレプリカにフェイルオーバーします。Neptune レプリカを再起動しても、フェイルオーバーは開始されません。

クライアントの信頼性を確保するよう設計します。フェイルオーバーイベント中に動作をテストします。エクスポネンシャルバックオフロジックを使用して、クライアントに再試行ロジックを実装します。このロジックを実装するコード例は、AWS Lambda Amazon Neptune の関数例のドキュメントにあります。

複数のデータベースエンジンに適用される一般的なバックアップ要件のセットがある場合は、AWS Backup の使用を検討してください。

パフォーマンス効率の柱

AWS Well-Architected フレームワークのパフォーマンス効率の柱は、データの取り込みまたはクエリ中にパフォーマンスを最適化する方法に焦点を当てています。パフォーマンスの最適化は、以下を段階的かつ継続的に実行するプロセスです。

- ビジネス要件を確認する
- ワークロードのパフォーマンスを測定する
- パフォーマンスの低いコンポーネントを特定する
- ビジネスニーズに合わせてコンポーネントを調整する

パフォーマンス効率の柱は、使用する適切なグラフデータモデルとクエリ言語を特定するのに役立つ、ユースケース固有のガイドラインを提供します。また、Amazon Neptune へデータを取り込み、Amazon Neptune からデータを使用する際に従うべきベストプラクティスも含まれています。

パフォーマンス効率の柱は、次の主要分野に焦点を当てています。

- グラフモデリング
- クエリの最適化
- クラスターの適切なサイズ設定
- 書き込みの最適化

グラフモデリングを理解する

Labeled Property Graph (LPG) モデルと Resource Description Framework (RDF) モデルの違いを理解します。ほとんどの場合は好みの問題ですが、あるモデルが他のモデルよりも適しているユースケースがいくつかあります。グラフ内の 2 つのノードを接続するパスに関する知識が必要な場合は、LPG を選択します。Neptune クラスターまたは他のグラフのトリプルストア間でデータをフェデレーションする場合は、RDF を選択します。

Software as a Service (SaaS) アプリケーション、またはマルチテナンシーを必要とするアプリケーションを構築する場合は、クラスターごとに 1 つのテナントを設定するのではなく、データモデルにテナントの論理的な分離を組み込むことを検討してください。このタイプの設計を実現するには、SPARQL の名前付きグラフとラベル付け戦略を使用できます。例えば、ラベルに顧客識別子を付加したり、テナント識別子を表すプロパティのキーと値のペアを追加したりできます。論理的な分

離を維持するために、クライアントレイヤーがこれらの値を挿入することを確認します。マルチテナンシーの推奨事項の詳細については、Amazon Neptune データベースを実行する ISVs」を参照してください。

クエリのパフォーマンスは、クエリの処理で評価する必要があるグラフオブジェクト (ノード、エッジ、プロパティ) の数によって異なります。そのため、グラフモデルはアプリケーションのパフォーマンスに大きな影響を与える可能性があります。可能な限り細かいラベルを使用し、パスの決定またはフィルタリングに必要なプロパティのみを保存します。より高いパフォーマンスを実現するには、要約ノードの作成や共通パスを接続するより直接的なエッジの作成など、グラフの一部を事前に計算することを検討してください。

同じラベルを持つエッジの数が異常に多いノード間の移動はなるべく避けます。このようなノードには多くの場合、何千ものエッジがあります (大部分のノードのエッジカウントは数十個です)。その結果、コンピューティングとデータの複雑さがはるかに高くなります。これらのノードは、一部のクエリパターンでは問題にならない場合がありますが、特に中間ステップとしてノード間を移動する場合は、このようなノードを回避するためにデータを異なる方法でモデリングすることをお勧めします。スロークエリログを使用して、これらのノード間を移動するクエリを識別できます。特にデバッグモードを使用すると、レイテンシーとデータアクセスメトリクスが平均的なクエリパターンよりもはるかに高くなることがわかります。

Neptune を使用して ID にランダムな GUID 値を割り当てる代わりに、確定的なノード ID をノードとエッジに使用します (ユースケースでサポートされている場合)。ID によるノードへのアクセスが最も効率的な方法です。

クエリを最適化する

LPG モデルでは、openCypher 言語と Gremlin 言語を同じように使用できます。パフォーマンスが最大の懸念事項である場合は、2 つの言語を互換的に使用することを検討してください。特定のクエリパターンでは、ある言語が他の言語よりもパフォーマンスに優れる可能性があるためです。

Neptune は、代替クエリエンジン (DFE) への変換を進めています。openCypher は DFE でのみ実行されますが、オプションで、クエリ注釈を使用して Gremlin クエリと SPARQL クエリの両方を DFE で実行するように設定できます。DFE を有効にした状態でクエリをテストし、DFE を使用しない場合のクエリパターンのパフォーマンスと比較することを検討してください。

Neptune は、グラフ全体を評価する分析クエリではなく、単一ノードまたは一連のノードから開始し、そこからファンアウトするトランザクションタイプのクエリに最適化されています。分析クエリワークロードには、Neptune Analytics を使用します。Neptune Analytics は、データ、分析、アル

ゴリズム処理に高速反復を必要とする調査、調査、またはデータサイエンスのワークロードに最適です。また、グラフデータに対してベクトル検索を実行し、Neptune データベースインスタンスから直接データをロードすることもできます。Neptune Analytics がニーズを満たさない場合は、 AWS SDK for Pandas または と AWS Glueまたは Amazon EMR を組み合わせた neptune-export の使用を検討することもできます。

モデルとクエリの非効率やボトルネックを特定するには、各クエリ言語の `profile` および `explain` API を使用して、クエリプランとクエリメトリクスの詳細な説明を取得します。詳細については、「Gremlin profile」、「openCypher explain」、および「SPARQL explain」を参照してください。

クエリパターンを理解します。グラフ内の個別のエッジの数が多くなると、Neptune のデフォルトのアクセス方式は効率が悪くなる場合があります。以下のクエリは、非常に非効率になる可能性があります。

- エッジラベルが指定されていない場合にエッジ間を後方に移動するクエリ。
- Gremlin の `.both()` など、内部的に同じパターンを使用する句、または任意の言語でノードをドロップする句 (ラベルを認識せずに受信エッジをドロップする必要があります)。
- プロパティラベルを指定せずにプロパティ値にアクセスするクエリ。これらのクエリは、非常に非効率になる可能性があります。これが使用パターンと一致する場合は、OSGP インデックス (目的語、主語、グラフ、述語) を有効にすることを検討してください。

スロークエリログを使用して、遅いクエリを識別します。クエリの遅延は、最適化されていないクエリプランや不必要に多いインデックスルックアップが原因で発生する可能性があり、I/O コストが増加する可能性があります。Gremlin、SPARQL、または openCypher 向けの Neptune の explain エンドポイントと profile エンドポイントは、これらのクエリが遅い理由を理解するのに役立ちます。句には次のものが含まれる場合があります。

- グラフ内の平均ノードと比較してエッジ数が異常に多いノード (数十に対して数千など) は、計算の複雑さが増し、レイテンシーが長くなり、リソースの消費量が増える可能性があります。これらのノードが正しくモデル化されているかどうか、または横断する必要があるエッジの数を減らすためにアクセスパターンを改善できるかどうかを決定します。
- 最適化されていないクエリには、特定のステップが最適化されていないという警告が含まれます。これらのクエリを書き換えて最適化されたステップを使用すると、パフォーマンスが向上する可能性があります。
- 冗長フィルターは、不要なインデックスルックアップを引き起こす可能性があります。同様に、冗長パターンは重複したインデックスルックアップを引き起こす可能性があり、これはクエリを

改善することで最適化できます (プロファイル出力の「`Index Operations - Duplication ratio`」を参照)。

- Gremlin などの一部の言語には厳密に型指定された数値がなく、代わりに型の上位変換を使用します。例えば、値が 55 の場合、Neptune は、整数型、ロング型、浮動小数点型、およびその他の 55 に相当する数値型の値を検索します。これにより、追加のオペレーションが発生します。型が一致することが事前にわかっている場合は、クエリヒントを使用してこれを回避できます。
- グラフモデルはパフォーマンスに大きな影響を与える可能性があります。より細かいラベルを使用するか、マルチホップ線形パスへのショートカットを事前に計算して、評価する必要があるオブジェクトの数を減らすことを検討してください。

クエリの最適化だけではパフォーマンス要件を満たすことができない場合は、Neptune でさまざまなキャッシュ手法を使用してこれらの要件を満たすことを検討してください。

Neptune のパフォーマンスは、バージョンごとに継続的に向上しています。リリースノートを確認して、各リリースの改善点の詳細を確認してください。最適なパフォーマンスを実現するために、Neptune DB クラスターの定期的な更新を計画することを検討してください。新しいバージョンでは、新しいインスタンスもサポートされています。`r8g` インスタンスを利用できるようにするには、1.4.5.0 以降にアップグレードすることを検討してください。これがワークロードのパフォーマンスを向上させる方法の詳細については、「Amazon Amazon Neptune v1.4.5 を使用した AWS Graviton4 R8g インスタンスでの書き込みクエリの料金パフォーマンスが 4.7 倍向上しました」を参照してください。

クラスターを適切なサイズにする

同時実行要件とスループット要件に合わせてクラスターのサイズを設定します。クラスター内の各インスタンスで処理できる同時クエリの数は、そのインスタンスの仮想 CPU (vCPU) の数の 2 倍です。すべてのワーカースレッドが占有されている間に到着する追加のクエリは、サーバー側のキューに入れられます。これらのクエリは、ワーカースレッドが利用可能になったときに先入れ先出し (FIFO) ベースで処理されます。`MainRequestQueuePendingRequests` Amazon CloudWatch メトリクスには、各インスタンスの現在のキューの深さが表示されます。この値が頻繁に 0 を超える場合は、より多くの vCPU を持つインスタンスの選択を検討してください。キューの深さが 8,192 を超えると、Neptune は`ThrottlingException` エラーを返します。

各インスタンスの RAM の約 65% がバッファキャッシュ用に予約されています。バッファキャッシュは、データのワーキングデータセット (グラフ全体ではなく、クエリ対象のデータのみ) を保持します。ストレージではなくバッファキャッシュから取得されるデータの割合を確認するには、CloudWatch メトリクスの `BufferCacheHitRatio` をモニタリングします。このメトリクスが

99.9% を下回ることがよくある場合は、より多くのメモリを持つインスタンスを試して、レイテンシーと I/O コストが減少するかどうか判断することを検討してください。

リードレプリカは、ライターインスタンスと同じサイズである必要はありません。ただし、書き込みワークロードが多いと、レプリケーションに追いつくことができないため、小さいレプリカは遅くなり、再起動する可能性があります。したがって、レプリカのサイズはライターインスタンスと同じかそれ以上にすることをお勧めします。

リードレプリカに自動スケーリングを使用する場合は、新しいリードレプリカをオンラインにするのに最大 15 分かかる場合があることに注意してください。クライアントトラフィックが急に、ただし予測可能な範囲で増加する場合は、スケジュールに基づくスケーリングを使用して、その初期化時間を考慮してリードレプリカの最小数を高く設定することを検討してください。

サーバーレスインスタンスは、いくつかの異なるユースケースとワークロードをサポートしています。以下のシナリオでは、プロビジョニングインスタンスよりもサーバーレスを優先して検討します。

- ワークロードが 1 日を通して頻繁に変動する。
- 新しいアプリケーションを作成したが、ワークロードのサイズが不明。
- 開発とテストを実行している。

サーバーレスインスタンスは、同等のプロビジョニングされたインスタンスよりも、GB 単位の RAM ベースでコストがかかることに注意してください。各サーバーレスインスタンスは、2 GB の RAM と関連する vCPU およびネットワークで構成されます。想外の請求を回避するため、オプションごとのコストを分析します。一般的に、大量のワークロードが発生するのは毎日数時間のみで、残りの時間はほぼゼロの場合、またはワークロードが 1 日を通して大幅に変動する場合にのみ、サーバーレスでコスト削減を達成できます。

Amazon Neptune 料金計算ツールを使用して、1 queries-per-second (QPS) 要件などの要因に基づいてクラスターの正しい設定を評価します。

書き込みの最適化

書き込みを最適化するには、次の点を考慮してください。

- Neptune Bulk Loader は、最初にデータベースをロードしたり、既存のデータに追加したりする場合に最適な方法です。Neptune ローダーはトランザクションではなく、データを削除できないため、データの削除が要件である場合は使用しないでください。

- トランザクションの更新は、サポートされているクエリ言語を使用して行うことができます。書き込み I/O オペレーションを最適化するには、コミットごとに 50～100 個のオブジェクトを一括でデータを書き込みます。オブジェクトとは、LPG のノード、エッジ、またはノードもしくはエッジ上のプロパティ、または RDF のトリプルストアまたはクワッドです。
- すべてのトランザクション Neptune 書き込みオペレーションは、接続ごとにシングルスレッドになります。Neptune に大量のデータを送信するときは、それぞれがデータを書き込む複数の並列接続を設定することを検討してください。Neptune でプロビジョニングされたインスタンスを選択すると、インスタンスサイズは vCPU の数に関連付けられます。Neptune はインスタンス上の vCPU ごとに 2 つのデータベーススレッドを作成するため、最適な並列接続を決定するためにテストする際は vCPU の数の 2 倍から開始します。 サーバーレスインスタンスは、4 NCU ごとにおよそ 1 のレートで vCPU の数をスケーリングします。

  > **Note**
  >
  > これは一括ロード API には適用されず、直接接続にのみ適用されます。

- 任意の時間に単一の接続だけがデータを書き込んでいる場合でも、すべての書き込みプロセス中の ConcurrentModificationExceptions に備えて計画を立て、効率的に処理します。`ConcurrentModificationExceptions` が発生しても信頼性を損なわないようクライアントを設計します。
- すべてのデータを削除する場合は、同時削除クエリを発行するのではなく、高速リセット API を使用することを検討してください。後者は前者に比べてはるかに時間がかかり、かなりの I/O コストが発生します。
- ほとんどのデータを削除する場合は、neptune-export を使用して新しいクラスターにデータをロードすることで、保持するデータをエクスポートすることを検討してください。その後、元のクラスターを削除します。

コスト最適化の柱

AWS Well-Architected フレームワークのコスト最適化の柱は、不要なコストを回避することに重点を置いています。以下の推奨事項は、コスト最適化の設計原則とアーキテクチャのベストプラクティスを Amazon Neptune に実行するのに役立ちます。

コスト最適化の柱は、次の主要分野に焦点を当てています。

- 長期的な支出を把握し、資金配分を管理する
- 適切なタイプおよび数量のリソースを選択する
- 過剰な支出なしでスケールし、ビジネスニーズを満たす

必要な使用状況パターンとサービスの理解

データモデルに識別可能なグラフ構造があり、クエリで関係を調べて複数のホップを通過する必要がある場合、Neptune はワークロードに適しています。グラフデータベースは、次のパターンには適していません。

- 主にシングルホップクエリ (データをオブジェクトの属性として表現した方が適切かどうかを検討してください)
- プロパティとして保存される JSON または BLOB データ
- 多数のノードにわたる数値プロパティの合計の計算など、データセット全体で集計されるクエリ

特定のアクセスパターンに複数の目的別データベースを一緒に使用することで、すべてのニーズに対応できるかどうかを検討してください。例えば、次のようになります。

- 複雑なグラフナビゲーションをあまり必要とせず、単一ノードのプロパティを高度に同時取得する必要がある API では、Neptune、DynamoDB、または Amazon DocumentDB の 1 つ以上を使用することが最適な場合があります。
- リレーショナルデータベースは Neptune と共存して既存の機能を維持できますが、Neptune は、リレーショナルデータベースでうまく実行およびスケーリングされないマルチホップトラバーサルにのみ使用できます。

Neptune とやり取りするサービスや Neptune を補完するサービスに付随する以下のようなコストを把握してください。

- Neptune に一括ロードされるデータファイルの Amazon Simple Storage Service (Amazon S3) ストレージコスト
- クエリの挿入またはアップサート、クエリの読み取り、Neptune ストリーム処理に使用される Lambda 関数
- Amazon API Gateway または で (データベースに直接接続する代わりに) クライアントアプリケーションとやり取りするために Neptune 上に構築された API レイヤー AWS AppSync
- AWS Glue Neptune との間でデータを転送するために使用する ジョブ
- ストリーミングデータを受信して Neptune にほぼリアルタイムで取り込む Amazon Kinesis または Amazon Managed Streaming for Apache Kafka (Amazon MSK) インスタンス。
- AWS Database Migration Service Neptune へのリレーショナルデータの移行用
- Jupyter Notebook と Deep Graph Library 機械学習モデルの Amazon SageMaker ランタイムコスト

コストに注意してリソースを選択する

Neptune の料金は、時間単位のインスタンスコスト (またはサーバーレスに消費される Neptune Compute Units)、データ I/O、ストレージ使用量に基づいています。インスタンスは平均してコスト全体の 85% を占めるため、適切なサイズを選ぶことがコストに大きな影響を与える可能性があります。インスタンスのサイズを適正化する最善の方法は、さまざまなインスタンスでアプリケーションのパフォーマンスをテストし、次の要因を比較することです。

- `MainRequestQueuePendingRequests` CloudWatch メトリクスは一貫してゼロに近い低い数値のままですか?
- `BufferCacheHitRatio` CloudWatch メトリクスはほとんどの時間で 99.9% 以上を維持していていますか?
- インスタンスコストおよび関連するデータ I/O コストのコストとパフォーマンス曲線はどうですか? ストレージとのバッファキャッシュのスワップを頻繁に必要とするサイズの小さいインスタンスでは、データ読み取りコストが大幅に増加する可能性があります。`BufferCacheHitRatio` は、これらのシナリオでは頻繁に低下します。

インスタンスコストは、同じインスタンスファミリー内のサイズに応じて直線的にスケーリングされます。`db.r6i.2xlarge` インスタンスの時間単位のコストは `db.r6i.xlarge` インスタンスの 2 倍で、リソース割り当ても 2 倍です。`db.r6i.24xlarge` インスタンスは、`db.r6i.xlarge` インスタンスの時間単位のコストの 24 倍です。

サポートする必要がある同時クエリの数を見積もります。読み取り専用クエリを処理するために、0 ~ 15 個のリードレプリカを持つことができます。要件が時間帯、週、または月によって異なる場合は、複数の小さなインスタンスを使用してスケジュールに基づいてスケーリングできます。インスタンスの各 vCPU には、同時クエリを処理するための 2 つのスレッドが用意されています。それぞれ 4 つの vCPU を持つ 3 つの `db.r6i.xlarge` リードレプリカは、24 個の同時クエリを処理できます。

そうではなく、トラフィック量が 1 秒あたりのクエリ数 (QPS) で測定される場合は、実験でクエリの平均レイテンシーを決定する必要があります。Neptune クラスターがサポートできる 1 秒あたりのクエリ数は、`vCPU × 2 × (1 second/average query latency)` に等しくなります。例えば、4 つの vCPU があり、クエリレイテンシーが 100 ミリ秒 (0.1 秒) の場合、`QPS = 4 × 2 × (1s/0.1s) = 80 queries per second` となります。

プロビジョニングされたインスタンスは、継続的で安定した、予測可能なワークロードに対して、サーバーレスよりも安価です。サーバーレスは、1 日あたり数時間のみ非常に高い使用量を必要とするワークロード (`db.r6i.4xlarge` など) があり、残りの時間はほとんどトラフィックがない (1 Neptune Compute Unit など) 場合に、コスト最適化の機会が得られます。数時間スケールアップし、その後スケールダウンするサーバーレスインスタンスは、プロビジョニングされた `db.r6i.4xlarge` インスタンスを終日使用するよりも安価になります。

Neptune 1.4.5.0 以降にアップグレードし、 `r8g` インスタンスを使用して、 `r7g`や などの旧世代のインスタンスよりも低コストで読み取りと書き込みのスループットを向上させることを検討してくださいr6g。詳細については、Amazon Neptune v1.4.5 を使用した AWS Graviton4 R8g インスタンスでの書き込みクエリの料金パフォーマンスが 4.7 倍向上する」(AWS ブログ記事) を参照してください。

Neptune クラスターは、デフォルトで標準ストレージで作成されます (コンソールを使用して を作成すると、デフォルトで I/O 最適化ストレージが選択されます) 。I/O 最適化ストレージでは、ストレージとインスタンスのコストがわずかに高くなりますが、I/O コストはありません。これにより、予測可能な経常コストが発生しますが、I/O 使用率が一般的に低い場合は、標準ストレージを利用する方がコスト効率が良い可能性があります。最初に大量のデータをロードする場合は、I/O 最適化ストレージを選択し、初期データロードを実行してから、標準ストレージに切り替えることでコストを最適化できます。ストレージタイプは請求モデルにのみ影響し、Neptune DB クラスターまたはインスタンス設定に技術的な違いはありません。ストレージタイプは 30 日に 1 回変更できます。30 日後、詳細な Neptune コストを確認し、Neptune 料金ページを使用して、I/O 最適化ストレージを使用してコストが高くなったかどうかを計算します。そうであれば、引き続き標準ストレージを使用し、そうでない場合は I/O 最適化に切り替えます。

ワークロードに最適な Neptune インスタンス設定を選択する

2025 年 7 月 15 日より AWS アカウント 前に を作成した場合は、Neptune でのエントリレベルの実験に AWS 無料利用枠を使用できます。db.t3.medium および db.t4g.medium インスタンスの無料利用時間である 750 時間で、Neptune を低スケールで十分に理解できます。クラスターは無料トライアル期間終了後も残りますが、それ以降の使用量に対しては課金されます。

db.t3.medium および db.t4g.mediumインスタンスは、openCypher、Graph Explorer、またはさまざまな生成 AI 統合を使用していない低コストの開発環境に適しています。これらのインスタンスの RAM-to-vCPUの比率 (2:1) は、Rファミリーインスタンス (8:1) またはXファミリーインスタンス (16:1) よりも小さくなります。これにより、比率が削減され、openCypher パフォーマンス、GenAI 統合 (LLM にグラフスキーマを通知するため)、Graph Explorer を有効にする DFE エンジン統計が使用できなくなります。パフォーマンスプロファイルは、Tファミリーインスタンスを使用する場合、特に前述のワークロードでは大きく異なる場合があります。これらのインスタンスは、クエリがグラフの大部分をナビゲートOutOfMemoryExceptionsする場合の の発生を増やすこともできます。後者の条件が影響を受ける可能性があるかどうかを判断するには、BufferCacheHitRatio CloudWatch メトリクスを確認します。

本番環境を示唆しない一貫性のない結果が発生する可能性があるため、Tファミリーインスタンスでのパフォーマンステストや負荷テストを行わないことを強くお勧めします。

プロビジョニングされたインスタンスは、ワークロードがかなり安定していて予測可能な場合に、コストとパフォーマンスの最適な組み合わせを提供します。インスタンスサイズは、必要なリクエストの同時実行数とクエリの複雑さに基づいて選択します。同時実行数を増やすには、より多くの vCPU が必要です。クエリの複雑さが高いほど、より多くの RAM が必要になります。前者の影響を判断するには、MainRequestQueuePendingRequests CloudWatch メトリクスを使用します (0 より大きい場合は、同時リクエストの数が処理可能な数よりも多いことを表します)。後者の影響を判断するには、BufferCacheHitRatio CloudWatch メトリクスを使用します。比率が頻繁に 99.9% を下回る場合、評価対象のグラフの作業部分を含むのに十分な RAM がなく、キャッシュスワップの頻度が高くなることを示します。インスタンスの R ファミリーに十分な同時実行性があるものの、十分な RAM がない場合は、インスタンスのXファミリーを試すことを検討してください。

サーバーレスインスタンスの理想的なユースケースについては、Neptune ドキュメントで説明しています。プロビジョニングとサーバーレスのどちらが最適かが不明で、コストが主な懸念事項である場合は、サーバーレスでワークロードをテストして、使用される NCU の数を判断し、プロビジョニング (N hours × hourly provisioned cost) のコストをサーバーレス (sum of NCUs × hourly cost per NCU) と比較します。同等のサイズのプロビジョニングインスタンスがわからない場合、1 NCU は、約 2 GB の RAM および関連する vCPU とネットワークに相当します。プロ

ビジョニングされたインスタンスが r6iファミリーのものである場合、比率は 8 GB の RAM あたり 1 vCPU、または関連するネットワークとともに 4 NCUs。Amazon Neptune 料金計算ツールには、最適なコスト設定を決定するのに役立つ比較も用意されています。

プライマリインスタンスとレプリカインスタンスにサーバーレスを使用する場合、昇格階層 0 と 1 のリードレプリカは、フェイルオーバーイベントが発生した場合に適切にスケーリングされるように、ライターインスタンスに合わせて NCU をスケーリングすることに注意してください。これらのインスタンスの NCU の制限は、ライターまたはリーダーのどちらのインスタンスが最も多くのトラフィックを受信するかに基づいて設定します。

クラスターが 1 日 24 時間、週 7 日必要でない環境では、使用していないときに Neptune インスタンスをオフにし、使用する前に再度起動するスクリプトを記述することを検討してください。必要なメンテナンスアップデートが適用されるように、Neptune インスタンスは 7 日ごとに自動的に再起動されます。インスタンスを長期間オフのままにする場合は、週次スクリプトを使用して再度シャットダウンします。

適切なサイズのデータストレージと転送

より効率的なクエリ (触れる必要のあるグラフ内のノード、エッジ、プロパティが少ないクエリなど) では、必要な I/O 転送が少なくなり、バッファキャッシュが少なくて済むため、より小さなインスタンスを使用できる可能性があります。クエリ言語のプロファイルエンドポイントまたは説明エンドポイントを使用してクエリを最適化し、クエリのパフォーマンスに合わせてグラフモデルを最適化することを検討してください。

Neptune は大きな文字列にディクショナリエンコードを使用し、そのディクショナリは効率ではなくパフォーマンスに最適化されています。プロパティに対して大きな BLOBs、JSON、または頻繁に変更される文字列がある場合は、それらを Neptune の外部の Amazon S3、Amazon DynamoDB、または Amazon DocumentDB に保存し、Neptune ノードには参照のみを保存することを検討してください。

場合によっては、インスタンスサイズを大きくするとコストを下げることができます。`BufferCacheHitRatio` が低いために I/O コストが非常に高い場合、バッファキャッシュが大きくすると、そのコストが大幅に削減される可能性があります。これは、データがストレージから頻繁にスワップされ、I/O 転送レートが発生する代わりに、すべてのデータがキャッシュに収まるためです。

Neptune はコピーオンライトでクローンを作成します。クローンを作成してグラフを複数のシャードに分割する場合、クローンされたクラスター上の不要なデータを削除しない方が効率的です。これ

には新しいデータページの作成が含まれ、ストレージコストが増加するためです。クローン作成イベント前から変更されていないデータは、2 つのクラスター間で共有される単独のデータページに存在し、その単独のコピーに対してのみ課金されます。

OSGP インデックスを有効にしたり、R5d インスタンスを使用したりしないでください。ただし、ワークロードに大きな違いが生じることをテストで確認している場合は除きます。どちらもめったに発生しないシナリオ向けに設計されており、メリットがほとんどないにもかかわらずコストが増加する可能性があります。

持続可能性の柱

持続可能性の柱は、クラウドワークロードの実行による環境への影響を最小限に抑えることに焦点を当てています。主なトピックは、持続可能性に関する責任共有モデル、影響の把握、必要なリソースを最小限に抑えてダウンストリームへの影響を軽減するための使用率の最大化です。

持続可能性の柱には、以下の主要な重点分野が含まれています。

- ユーザーによる影響
- 持続可能性の目標
- 使用率の最大化
- より効率的な新しいハードウェアおよびソフトウェア製品の予測および採用
- マネージドサービスの使用
- ダウンストリームの影響軽減

このガイドでは、ユーザーによる影響に焦点を当てます。その他の持続可能性設計原則の詳細については、AWS 「Well-Architected フレームワーク」を参照してください。

ユーザーによる選択と要件は、環境に影響を及ぼします。ユーザーが炭素集約度の低い AWS リージョン を選択し、ユーザーの要件が単に稼働時間と耐久性を最大化するのでなく、実際のワークロードのニーズを反映している場合、ワークロードの持続可能性が向上します。ワークロードの設計および継続的な運用に、次のセクションで説明するベストプラクティスと考慮事項を採用すると、環境に良い影響を及ぼすことができます。

AWS リージョン 選択

一部の AWS リージョン は Amazon 再生可能エネルギープロジェクトの近くにあるか、グリッドの炭素強度が他のものよりも低い公開されている場所にあります。リージョンを選ぶ際は、ワークロードにとって有効な持続可能性への影響を考慮し、リストを Neptune が利用可能なリージョンと相互参照します。

ユーザー行動パターンに基づく使用量

ユーザーのトラフィックと動作に合わせて使用量を適切なサイズにすることで、 AWS が環境に与える影響を最小限に抑えることができます。ソリューションを設計するときは、次のベストプラクティスを考慮してください。

- CPUUtilization、MainRequestQueuePendingRequests、TotalRequestsPerSec などの Amazon CloudWatch メトリクスをモニタリングして、需要が最も高いときと最も低いときを判断し、そうしたタイミングにおいてクラスターリソースのサイズが適切に設定されていることを確認します。
- 使用していない時間帯は非本番環境を停止するよう自動化します。詳細については、「Automate the stopping and starting of Amazon Neptune environment resources using resource tags」を参照してください。
- トラフィックパターンの変動が頻繁かつ予測不可能な場合は、ピークトラフィック用にプロビジョニングされたインスタンスを使用する代わりに、需要に応じてスケールアップ/ダウンする Neptune サーバーレスインスタンスを使用することを検討してください。
- サービスレベル契約を、ビジネス継続性の目標だけでなく持続可能性の目標に合わせることを検討してください。マルチリージョンディザスタリカバリ、高可用性、長期バックアップ保持などの簡単な要件については、特に非本番環境やミッションクリティカルでないワークロードの場合、これらの目標を達成するために必要なリソースの量を減らすことができます。

ソフトウェア開発とアーキテクチャパターンを最適化する

無駄を防ぐには、モデルとクエリを最適化し、コンピューティングリソースを共有して、Neptune インスタンスとクラスターで使用可能なすべてのリソースを活用します。具体的なベストプラクティスは次のとおりです。

- 開発者に、それぞれが独自のインスタンスを作成するのではなく、Neptune インスタンスと Jupyter Notebook アプリケーションのインスタンスを共有してもらいます。マルチテナンシーパーティショニング戦略を使用して、各デベロッパーに単一の Neptune クラスターに独自の論理パーティションを提供し、単一の Jupyter インスタンスで各デベロッパーに個別のノートブックフォルダを作成します。
- リソースの使用率を最大化し、アイドル時間を最小限に抑えるパターンを実装します。例えば、データをロードするための並列スレッドや、レコードをより大きなトランザクションにまとめてバッチ処理するなどです。
- クエリとグラフモデルを最適化して、結果の計算に必要なリソースを最小限に抑えます。
- Gremlin クエリの結果については、結果キャッシュ機能を使用して、ページ分割されたクエリまたは頻繁に繰り返されるクエリの再計算に費やされるリソースを最小限に抑えます。
- Neptune 環境を最新の状態に保ちます。Neptune の最新バージョンは、Graviton などの最新の Amazon EC2 インスタンスをより効率的にサポートします。また、クエリ最適化の改善とバグ修正により、クエリの計算に必要なリソースの量を削減できます。

リソース

リファレンス

- AWS Well-Architected
- AWS Well-Architected フレームワークのドキュメント
- Neptune の最新更新
- ベストプラクティス: Neptune を最大限に活用する
- Amazon Neptune 料金計算ツール

ブログ記事

- Automated testing of Amazon Neptune data access with Apache TinkerPop Gremlin
- Automate the stopping and starting of Amazon Neptune environment resources using resource tags
- Fine Grained Access Control for Amazon Neptune data plane actions
- Amazon Neptune v1.4.5 を使用した AWS Graviton4 R8g インスタンスでの書き込みクエリの料金パフォーマンスが 4.7 倍向上
- Orca Security が Amazon Neptune データベースのパフォーマンスを最適化する方法
- Amazon Neptune パブリックエンドポイントを使用してグラフアプリケーションを迅速に構築する
- 新しい Amazon Neptune エンジンバージョンはopenCypher クエリパフォーマンスのために最大 9 倍高速および 10 倍高いスループットを提供します。

無料 AWS スキルビルダーコース

- Amazon Neptune の開始方法
- Building Applications on Amazon Neptune
- Data Modeling for Amazon Neptune

寄稿者

このガイドの寄稿者は次のとおりです。

- Brian O'Keefe、プリンシパル Neptune ソリューションアーキテクト、 AWS
- Abhishek Mishra、シニア Neptune ソリューションアーキテクト、 AWS
- Ganesh Sawhney、戦略的パートナーサクセスソリューションアーキテクトチームリーダー AWS
- マイケル・ヘイビー、シニア Neptune ソリューションアーキテクト、 AWS
- Kevin Phillips、Neptune ソリューションアーキテクト、 AWS
- Melissa Kwok、Neptune ソリューションアーキテクト、 AWS
- プリンシパルソリューションアーキテクト、Sakti Mishra AWS
- Javed Ali、シニアソリューションアーキテクト、 AWS

ドキュメント履歴

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

変更	説明	日付
Neptune リリースの更新	Amazon Neptune 1.4.6.0 以降に関する情報を含めるようにドキュメントを更新しました。	2026 年 1 月 2 日
初版発行	—	2023 年 9 月 27 日

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。