Amazon Aurora へのデータベースの移行

2016 年 6 月

注意

本書は情報提供のみを目的としています。本書の発行時点における AWS の現行製品と慣行を表したものであり、それらは予告なく変更されることがあります。お客様は本書の情報、および AWS 製品またはサービスの利用について、独自の評価に基づき判断する責任を負います。いずれの AWS 製品またはサービスも、明示または黙示を問わずいかなる保証も伴うことなく、「現状のまま」提供されます。本書のいかなる内容も、AWS、その関係者、サプライヤ、またはライセンサーからの保証、表明、契約的責任、条件や確約を意味するものではありません。お客様に対する AWS の責任は、AWS 契約により規定されます。本書は、AWS とお客様の間で行われるいかなる契約の一部でもなく、そのような契約の内容を変更するものでもありません。

目次

要約

Amazon Aurora は MySQL と互換性のあるエンタープライズグレードのリレーショナルデータベースエンジンです。Amazon Aurora は従来のリレーショナルデータベースエンジンの制限の多くを解消する、クラウドネイティブなデータベースです。このホワイトペーパーの目的は、既存のデータベースを Amazon Aurora に移行するためのベストプラクティスを示すことです。移行の考慮事項を示すとともに、アプリケーションの中断を最小限に抑えながら、オープンソースの商用データベースを Amazon Aurora に移行するための詳しいプロセスを示します。

Amazon Aurora 入門

これまで何十年にもわたり、データストレージと永続性のためには従来のリレーショナルデータベースが主に選択されてきました。これらのデータベースシステムは依然としてモノリシックなアーキテクチャに依存しており、クラウドインフラストラクチャを活用する設計になっていません。こうしたモノリシックなアーキテクチャでは、特にコスト、柔軟性、可用性など特定の領域において多くの課題があります。AWS は、これらの課題に対応するため、リレーショナルデータベースをクラウドインフラストラクチャ用に再設計し、Amazon Aurora を発表しました。

Amazon Aurora は、MySQL と互換性のあるリレーショナルデータベースエンジンで、オープンソースデータベースのシンプルさとコスト効率性を備え、高性能の商用データベースのスピード、可用性、およびセキュリティをあわせ持ったエンジンです。Aurora は MySQL よりも最大 5 倍のスピードを提供し、高性能の商用データベースと同等のパフォーマンスを備えています。Amazon Aurora の料金は商用エンジンの 1/10 です。

Amazon Aurora は Amazon Relational Database Service (Amazon RDS) プラットフォームを通じて利用できます。Aurora は他の Amazon RDS データベースと同様、フルマネージドデータベースサービスです。Amazon RDS プラットフォームでは、ハードウェアのプロビジョニング、ソフトウェアのパッチ適用、セットアップ、設定、モニタリング、バックアップといったほとんどのデータベース管理タスクは完全に自動化されます。

Amazon Aurora はミッションクリティカルなワークロード用に作成されており、デフォルトで高い可用性を備えています。Aurora データベースクラスターはリージョンの複数のアベイラビリティーゾーンにまたがり、初期状態で耐久性と耐障害性を複数の物理的なデータセンター間のデータに提供します。アベイラビリティーゾーンは、Amazon が運営する 1 つ以上の可用性の高いデータセンターで構成されます。アベイラビリティーゾーン同士は相互に分離され、低レイテンシーのリンクで接続されています。データベースボリュームの各セグメントは、これらのアベイラビリティーゾーン間で 6 回レプリケートされます。

データベースのデータの量が増えると、Aurora クラスターボリュームも自動的に大きくなり、パフォーマンスや可用性への影響はありません。したがって、事前に大規模なデータベースストレージを予測し、プロビジョニングする必要はありません。Aurora クラスターボリュームは、最大 64 テラバイト (TB) のサイズまで増やすことができます。料金が発生するのは、Aurora クラスターボリュームで使用したスペースの分のみです。

Aurora の自動化されたバックアップ機能では、データのポイントインタイムリカバリがサポートされるため、最大で最後の 5 分前まで、保存期間中の任意の時点にデータベースを復元することができます。自動化されたバックアップは、ほぼ完全な耐久性を求めた設計になっている Amazon Simple Storage Service (Amazon S3) に保存されます。Amazon Aurora のバックアップは自動、差分、継続的で、データベースのパフォーマンスへの影響はありません。

読み取り専用レプリカが必要なアプリケーションでは、Aurora データベースごとに最大 15 個の Aurora レプリカを作成でき、レプリカの遅延は非常に低く抑えることができます。これらのレプリカはソースインスタンスと同じ基盤となるストレージを共有するため、コストが下がり、レプリカノードで書き込みを実行する必要がなくなります。

Amazon Aurora はセキュリティが高く、AWS Key Management Service (AWS KMS) を通じて作成、管理するキーを使用してデータベースを暗号化することができます。Amazon Aurora 暗号化で実行されるデータベースインスタンスでは、基盤となるストレージに保存された保管中のデータは暗号化されます。同じクラスター内の自動バックアップ、スナップショット、およびレプリカも同様です。Amazon Aurora は SSL (AES-256) を使用して伝送中のデータを保護します。

Aurora 機能の完全な一覧については、Amazon Aurora の製品ページを参照してください。[1]Amazon Aurora の豊富な機能と費用対効果により、ミッションクリティカルなアプリケーションに適したデータベースという見方がますます増えています。

データベースの移行に関する考慮事項

データベースは、ほとんどのアプリケーションのアーキテクチャで重要なコンポーネントとなります。新しいプラットフォームへのデータベースの移行は、アプリケーションのライフサイクルにおいて重要なイベントであり、アプリケーションの機能、パフォーマンス、および信頼性に大きな影響を与える可能性があります。Amazon Aurora への最初の移行プロジェクトを開始する前に、いくつかの重要な考慮事項を検討してください。

移行の段階

データベースの移行は複雑になる傾向があるため、段階的で反復性のある手法を採用することをお勧めします。

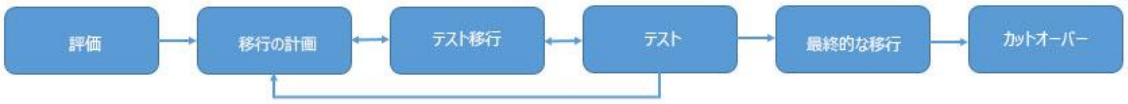

図 1: 移行の段階

アプリケーションに関する考慮事項

Aurora の機能の評価

ほとんどのアプリケーションは、多くのリレーショナルデータベースエンジンと連携するように構築できますが、アプリケーションが Amazon Aurora に対応することを確認してください。Amazon Aurora は MySQL 5.6 と互換性を持つよう設計されています。したがって、現在 MySQL データベースで使用されている大半のコード、アプリケーション、ドライバー、およびツールは、ほとんどまたはまったく変更することなく Aurora で使用できます。

ただし、MyISAM ストレージエンジンなど特定の MySQL 機能は、Amazon Aurora では使用できません。また、Aurora サービスのマネージド型の性質により、データベースノードへの SSH アクセスは制限されます。これにより、データベースホストにサードパーティー製ツールまたはプラグインをインストールする機能に影響が及ぶ可能性があります。

パフォーマンスに関する考慮事項

データベースを新しいプラットフォームに移行するときに、データベースのパフォーマンスが主要な考慮事項となります。したがって、データベースの移行に成功した多くのプロジェクトでは、新しいデータベースプラットフォームのパフォーマンスの評価から開始しています。Sysbench および TPC-C ベンチマークの実行により、全体的なデータベースパフォーマンスについてかなり理解できますが、これらのベンチマークでは、アプリケーションのデータアクセスパターンはエミュレートされません。より有益な結果を得るには、新しいプラットフォームで直接クエリ (またはクエリのサブセット) を実行して、時間が重要なワークロードのデータベースパフォーマンスをテストします。

以下の手法を検討してください。

- 現在のデータベースが MySQL である場合は、アプリケーションのテストバージョンまたはステージングバージョンでデータベースのダウンタイムとパフォーマンスをテストするか、実稼働のワークロードを再現して Amazon Aurora に移行します。

- MySQL と互換性がないエンジンを使用している場合は、最もビジー状態のテーブルを Amazon Aurora に選択的にコピーし、それらのテーブルのクエリをテストします。これが良い開始点となります。もちろん、完全なデータ移行後にテストを実行することで、新しいプラットフォームでのアプリケーションの実際のパフォーマンスを完全に把握することができます。

Amazon Aurora は商用エンジンと同等のパフォーマンスを提供し、MySQL のパフォーマンスを大幅に超えます。これを行うため、データベースのワークロード用に設計された SSD ベースの仮想化ストレージレイヤーを使用して、データベースエンジンと緊密に統合されます。これにより、ストレージシステムへの書き込みが減り、ロックの競合が最小化され、データベースプロセススレッドによって作成される遅延がなくなります。r3.8xlarge インスタンスでの SysBench のテストでは、Amazon Aurora は 1 秒あたり 585,000 を超える読み取りと、1 秒あたり 107,000 の書き込みを実現しました。これは、同じハードウェアで同じベンチマークを実行した MySQL より 5 倍高速です。

Amazon Aurora が従来の MySQL をはるかに上回る 1 つの領域が、同時実行性が高いワークロードです。Amazon Aurora でのワークロードのスループットを最大化するため、大量の同時実行クエリを実行するアプリケーションを作成することをお勧めします。

シャーディングおよびリードレプリカに関する考慮事項

データベースが複数のノード間でシャーディングされている場合、それらのシャードを移行中に 1 つの Aurora データベースに組み合わせる機会が発生することがあります。1 つの Amazon Aurora インスタンスは最大 64 TB までスケールアップし、数千のテーブルをサポートして、標準の MySQL データベースよりも多くの読み取りと書き込みをサポートします。

アプリケーションで読み込み/書き込みが多い場合は、Aurora リードレプリカを使用して、マスターデータベースノードから読み取り専用ワークロードをオフロードすることを検討してください。これを行うと、書き込みについてプライマリデータベースの同時実行が改善し、全体的な読み取りと書き込みのパフォーマンスが向上します。また、リードレプリカを使用すると、マルチ AZ 設定でコストを削減できます。これは、データベースクラスターでフェイルオーバー機能を追加しながら、プライマリインスタンスにより小さいインスタンスを使用できる場合があるためです。Aurora のリードレプリカでは、レプリケーションの遅延時間はゼロに近く、最大 15 個のリードレプリカを作成できます。

信頼性に関する考慮事項

データベースに関する重要な考慮事項は、高可用性と災害対策です。アプリケーションの RTO (目標復旧時間) と RPO (目標復旧時点) を決定します。Amazon Aurora では、これらの両方の要素を大幅に向上させることができます。

Amazon Aurora は、ほとんどのデータベースクラッシュシナリオで、データベースの再起動時間を 60 秒以下に減らします。Aurora はデータベースプロセスからバッファキャッシュを移動し、再起動時にすぐに利用可能にします。ハードウェアおよびアベイラビリティーゾーンの障害というまれなシナリオでは、データベースプラットフォームによって復元が自動的に処理されます。

Aurora は AWS リージョン内で RPO がゼロの復元を提供するよう設計されています。これはオンプレミスデータベースシステムに対する主要な機能強化です。Aurora は 3 つのアベイラビリティーゾーンにわたりデータの 6 個のコピーを維持し、データ損失なしで、正常な AZ でデータベースの復元を自動的に試みます。万一 Amazon Aurora ストレージでデータが利用できない場合、DB スナップショットから復元するか、新しいインスタンスへのポイントインタイムの復元オペレーションを実行できます。

コストおよびライセンスに関する考慮事項

データベースの所有と実行には、関連費用が発生します。データベースの移行を計画する前に、新しいデータベースプラットフォームの総所有コスト (TCO) の分析が必須になります。新しいデータベースプラットフォームへの移行により、総所有コストが減り、アプリケーションの機能は同等に保たれるか、向上することが理想です。オープンソースデータベースエンジン (MySQL、Postgres など) を実行している場合、コストは主にハードウェア、サーバー管理、およびデータベース管理アクティビティに関連して発生します。ただし、商用データベースエンジン (Oracle、SQL Server、DB2 など) を実行している場合、コストのかなりの部分はデータベースのライセンス料金です。

Aurora は商用エンジンの 1/10 の料金で使用できるため、Aurora に移行する多くのアプリケーションは、TCO を大幅に減らすことができます。MySQL や Postgres などのオープンソースエンジンを実行している場合でも、Aurora の高いパフォーマンスと二重目的のリードレプリカにより、Amazon Aurora に移行することで意味のある節約を実現できます。詳細については、「Amazon RDS for Aurora 料金表」ページを参照してください。[2]

移行に関するその他の考慮事項

アプリケーションの適合性、パフォーマンス、TCO、および信頼性の要素を考慮したら、新しいプラットフォームへの移行に必要なことを検討する必要があります。

コード変更の労力の予測

Amazon Aurora へのデータベースの移行に際しては、実行する必要があるコードおよびスキーマの変更の量を予測することが重要です。MySQL と互換性のあるデータベースからの移行では、コード変更はほとんど不要です。ただし、MySQL 以外のエンジンからの移行では、スキーマとコードの変更が必要になる可能性があります。後で説明する AWS スキーマ変換ツールを使用すると、この労力を予測するうえで役立ちます。

移行中のアプリケーションの可用性

Amazon Aurora への移行では、予測可能なダウンタイムの手法をアプリケーションで使用するか、ゼロに近いダウンタイムの手法を使用するオプションがあります。選択する手法は、データベースのサイズと、アプリケーションの可用性の要件によって決まります。いずれの場合も、アプリケーションとビジネスへの移行プロセスの影響について考慮してから、データベースの移行を開始することをお勧めします。次のいくつかのセクションでは、両方の手法について詳しく説明します。

データベース移行プロセスの計画

前のセクションでは、データベースを Amazon Aurora に移行する際に検討する主な考慮事項についていくつか説明しました。Aurora がアプリケーションに対して適切であると判断したら、次のステップとして移行の予備的な手法を決定し、データベース移行プランを作成します。

同機種の移行

ソースデータベースが MySQL 5.6 互換データベース (MySQL、MariaDB、Percona など) である場合、Aurora への移行は非常に単純です。

ダウンタイムがある同機種の移行

アプリケーションが、オフピーク時間中に予測される長いダウンタイムに対応できる場合、ダウンタイムがある移行が最も単純なオプションであり、強く推奨される手法です。ほとんどのデータベース移行プロジェクトは、このカテゴリに分類されます。これは、ほとんどのアプリケーションには明確に定義されたメンテナンス時間帯がすでにあるためです。ダウンタイムがあるデータベースの移行には、以下のオプションがあります。

- **RDS スナップショットの移行** – ソースデータベースが Amazon RDS MySQL 5.6 で実行されている場合は、単純にそのデータベースのスナップショットを Amazon Aurora に移行できます。ダウンタイムがある移行では、スナップショットと移行が進行中のときはアプリケーションを停止するか、データベースへの書き込みを停止する必要があります。移行するタイミングは主にデータベースのサイズによって異なり、テスト移行を実行して本番の移行前に決定できます。スナップショットの移行オプションは、「RDS スナップショットの移行」セクションで説明しています。また、スナップショットの移行と binlog のレプリケーションを使用して、ダウンタイムがゼロに近い移行を行うこともできます。これについては、次のセクションで説明します。

- **ネイティブ MySQL ツールを使用した移行。** MySQL ツールを使用して、データとスキーマを Aurora に移行することができます。これは、データベース移行プロセスをより詳細に制御する必要がある場合、ネイティブ MySQL ツールの使用に慣れている場合、および他の移行方法がユースケースに適していない場合に有益なオプションです。このオプションを使用する際のベストプラクティスについては、「Amazon RDS for Aurora のエクスポート/インポートのパフォーマンスに関するベストプラクティス」ホワイトペーパーをダウンロードしてください。[3]

- **AWS Database Migration Service (AWS DMS) を使用した移行。** ソースデータベースを Amazon Aurora に移行するための別のツールとして、AWS DMS を使用した 1 回限りの移行があります。AWS DMS を使用してデータを移動する前に、ネイティブ MySQL ツールを使用してソースからターゲットにデータベーススキーマをコピーする必要があります。詳しいステップについては、「データの移行」セクションを参照してください。AWS DMS の使用は、ネイティブ MySQL ツールの使用経験がない場合に有益なオプションです。

ダウンタイムがゼロに近い同機種の移行

シナリオによっては、最小のダウンタイムでデータベースを Aurora に移行したい場合もあります。ここでは、以下の 2 つの例を示します。

- データベースが比較的大きく、ダウンタイムオプションを使用した移行時間がアプリケーションのメンテナンス時間帯よりも長くなる
- ソースデータベースとターゲットデータベースをテスト目的で並列で実行したい

このような場合は、ソースの MySQL データベースから Aurora へ、レプリケーションを使用してリアルタイムで変更をレプリケートできます。いくつかのオプションを選択できます。

- **MySQL binlog レプリケーションを使用した、ダウンタイムがゼロに近い移行。**Amazon Aurora は従来の MySQL binlog レプリケーションをサポートします。MySQL データベースを実行している場合、お客様はすでに従来の binlog レプリケーションセットアップに慣れていると思われます。このような場合に、移行プロセスをより詳細に制御するには、binlog レプリケーションと併せてネイティブツールを使用して 1 回限りのデータベースの読み込みを行うと、使い慣れた方法で Aurora に移行できます。
- **AWS Database Migration Serrvice (AWS DMS) を使用した、ダウンタイムがゼロに近い移行。**AWS DMS では、1 回のみの移行がサポートされることに加えて、ソースからターゲットへの変更データキャプチャ (CDC) を使用したリアルタイムのデータレプリケーションもサポートされます。AWS DMS が初期のデータコピー、レプリケーションインスタンスの設定、およびレプリケーションのモニタリングに関連する複雑性に対応します。初期のデータベース移行が完了した後で、選択する限り、ターゲットデータベースはソースとの同期を維持します。binlog レプリケーションに精通していない場合、同機種のダウンタイムがゼロに近い Amazon Aurora への移行では、AWS DMS が次の最適なオプションとなります。「AWS DMS の概要と一般的な手法」セクションを参照してください。

- **RDS スナップショットの移行と MySQL binlog レプリケーションを使用した、ダウンタイムがゼロに近い移行。**ソースデータベースが Amazon RDS MySQL 5.6 で実行されている場合は、単純にそのデータベースのスナップショットを Amazon Aurora に移行し、スナップショットの移行後にソースとターゲット間で binlog レプリケーションを開始できます。この移行オプションの詳細については、『*Amazon RDS* ユーザーガイド』の「Amazon Aurora を使用したレプリケーション」を参照してください。[4]

異機種の移行

MySQL と互換性のないデータベース (Oracle、SQL Server、PostgresSQL など) を Amazon Aurora に移行する予定の場合、この移行を迅速で簡単に達成するための複数のオプションがあります。

スキーマの移行

MySQL と互換性のないデータベースから Amazon Aurora へのスキーマの移行は、AWS スキーマ変換ツールを使用して達成できます。このツールは、データベーススキーマを Oracle、Microsoft SQL Server、または PostgreSQL データベースから Amazon RDS MySQL DB インスタンスまたは Amazon Aurora DB クラスターに変換するために役立つデスクトップアプリケーションです。ソースデータベースからのスキーマを自動的かつ完全に変換できない場合、AWS スキーマ変換ツールにより、ターゲット Amazon RDS データベースで同等のスキーマを作成する方法が案内されます。詳細については、「データベーススキーマの移行」セクションを参照してください。

データの移行

AWS Database Migration Service (AWS DMS) は、ゼロに近いダウンタイムで同機種の移行をサポートする一方で、異機種データベース間の連続レプリケーションもサポートし、ソースデータベースをターゲットデータベースに移行するための推奨のオプションです。ダウンタイムがある移行と、ダウンタイムがゼロに近い移行の両方に使用できます。移行が開始されると、AWS DMS はデータ型の変換、圧縮、並行転送 (データ転送を高速化するため) といった移行プロセスの複雑性をすべて管理し、移行プロセス中に発生するソースデータベースへのデータ変更が、自動的にターゲットにレプリケートされるようにします。

AWS DMS の使用に加えて、Attunity Replicate、Tungsten Replicator、Oracle Golden Gate など、さまざまなサードパーティー製ツールを使用してデータを Amazon Aurora に移行できます。選択するツールにかかわらず、移行のためのツールセットを確定する前に、パフォーマンスやライセンスのコストを考慮してください。

Amazon Aurora への大規模データベースの移行

あらゆるデータベース移行プロジェクトで、大規模なデータセットの移行には独自の課題があります。成功している多くの大規模データベースの移行プロジェクトでは、以下の戦略が組み合わされて使用されています。

- **連続的なレプリケーションを使用した移行。**通常、大規模なデータベースでは、ソースからターゲットにデータを移動する際のダウンタイムに関して、より多くの要件があります。ダウンタイムを減らすには、最初にベースラインデータをソースからターゲットにロードし、変更が追いつくように (MySQL ネイティブツール、AWS DMS、またはサードパーティー製ツールを使用して) レプリケーションを有効にします。

- **まず、静的なテーブルをコピーする。**データベースが参照データのある大規模な静的テーブルに依存している場合、それらの大規模なテーブルをターゲットデータベースに移行してから、アクティブなデータセットを移行できます。AWS DMS を利用して選択的にテーブルをコピーするか、それらのテーブルを手動でエクスポートおよびインポートできます。
- **複数の段階の移行。**数千のテーブルがある大規模なデータベースの移行は、複数の段階に分けることができます。たとえば、ソースデータベースが完全にターゲットデータベースに移行するまで、毎週末にクロス結合クエリを使用せずにテーブルのセットを移動できます。これを達成するには、データセットが 2 つの個別のノードにあるときに、2 つのデータベースに同時に接続するようアプリケーションを変更する必要があります。これは一般的な移行パターンではありませんが、オプションの 1 つです。
- **データベースのクリーンアップ。**多くの大規模データベースには、使用されないままのデータやテーブルが含まれています。多くの場合、開発者や DBA は同じデータベースにテーブルのバックアップコピーを保持したり、使用されていないテーブルの削除を忘れたりします。どのような理由でも、データベースの移行プロジェクトでは、移行前に既存のデータベースをクリーンアップする機会があります。いくつかのテーブルが使用されていない場合は、それらを削除したり、別のデータベースにアーカイブしたりします。また、大規模なテーブルから古いデータを削除したり、そのデータをフラットファイルにアーカイブしたりすることもあります。

Amazon Aurora でのパーティションとシャードの統合

高いパフォーマンスを達成するためにデータベースの複数のシャードまたは機能パーティションを実行している場合、それらのパーティションまたはシャードを 1 つの Aurora データベースに統合する機会があります。1 つの Amazon Aurora インスタンスは最大 64 TB までスケールアップし、数千のテーブルをサポートして、標準の MySQL データベースよりも多くの読み取りと書き込みをサポートします。1 つの Aurora インスタンスでこれらのパーティションを統合すると、総所有コストが減り、データベースの管理が簡略化されるだけでなく、クロスパーティションクエリのパフォーマンスも大幅に向上します。

- **機能パーティション。**機能パーティションとは、各ノードを異なるタスク専用にすることを意味します。たとえば、e コマースアプリケーションでは、1 つのデータベースノードが製品カタログデータを提供していて、別のデータベースノードが注文をキャプチャし、処理しているとします。その結果、通常、これらのパーティションには明確で重複しないスキーマが存在します。

  統合戦略。各機能パーティションをターゲット Aurora インスタンスの明確なスキーマとして移行します。ソースデータベースが MySQL と互換性がある場合は、ネイティブ MySQL ツールを使用してスキーマを移行し、AWS DMS を使用してデータを移行します。1 回限り、またはレプリケーションを使用して連続的に行います。ソースデータベースが MySQL と互換性がない場合は、AWS スキーマ変換ツールを使用してスキーマを Aurora に移行し、1 回のロードまたは連続的なレプリケーション用に AWS DMS を使用します。

- **データシャード。**複数のノードにわたる明確なデータのセットを持つ同じスキーマがある場合、データベースシャーディングを利用しています。たとえば、高トラフィックのブログサービスでは、同じテーブルスキーマを維持しながら、複数のデータベースシャードにユーザーアクティビティとデータをシャードする場合があります。

  統合戦略。すべてのシャードは同じデータベーススキーマを共有するため、ターゲットスキーマを 1 回作成するだけで済みます。MySQL 互換データベースを使用している場合は、ネイティブツールを使用してデータベーススキーマを Aurora に移行します。MySQL と互換性のないデータベースを使用している場合は、AWS スキーマ変換ツールを使用してデータベーススキーマを Aurora に移行します。データベーススキーマを移行したら、データベースシャードへの書き込みを停止し、ネイティブツールまたは AWS DMS の 1 回限りのデータロードを使用して個別のシャードを Aurora に移行するのが最適です。アプリケーションへの書き込みを長時間にわたり停止できない場合、AWS DMS とレプリケーションを使用できますが、適切な計画とテストを実行した後にしてください。

移行オプションの概要

ソースデータベースのタイプ	ダウンタイムがある移行	ダウンタイムがほぼゼロの移行
Amazon RDS MySQL	オプション 1: RDS スナップショットの移行 オプション 2: ネイティブツールを使用した手動の移行* オプション 3: ネイティブツールを使用したスキーマの移行と AWS DMS を使用したデータのロード	オプション 1: ネイティブツールを使用した移行 + binlog のレプリケーション オプション 2: RDS スナップショットの移行 + binlog のレプリケーション オプション 3: ネイティブツールを使用したスキーマの移行 + AWS DMS を使用したデータの移動
MySQL Amazon EC2 またはオンプレミス	オプション 1: ネイティブツールを使用した移行 オプション 2: ネイティブツールを使用したスキーマの移行 + AWS DMS を使用したデータのロード	オプション 1: ネイティブツールを使用した移行 + binlog のレプリケーション オプション 2: ネイティブツールを使用したスキーマの移行 + AWS DMS を使用したデータの移動
Oracle/SQL サーバー	オプション 1: AWS スキーマ変換ツール + AWS DMS (推奨) オプション 2: スキーマ変換用の手動またはサードパーティー製ツール + ターゲットでの手動またはサードパーティーによるデータのロード	オプション 1: AWS スキーマ変換ツール + AWS DMS (推奨) オプション 2: スキーマ変換用の手動またはサードパーティー製ツール + ターゲットでの手動またはサードパーティーによるデータのロード + レプリケーション用のサードパーティー製ツール
その他の MySQL 以外のデータベース	オプション: スキーマ変換用の手動またはサードパーティ製ツール + ターゲットでの手動またはサードパーティーによるデータのロード	オプション: スキーマ変換用の手動またはサードパーティ製ツール + ターゲットでの手動またはサードパーティーによるデータのロード + レプリケーション用のサードパーティー製ツール (GoldenGate など)

*Mysql ネイティブツール: mysqldump、SELECT INTO OUTFILE、mydumper/myloader などのサードパーティー製ツール

RDS スナップショットの移行

RDS スナップショットの移行を使用して Aurora に移行するには、MySQL データベースが Amazon RDS MySQL 5.6 で実行されていて、データベースの RDS スナップショットを作成する必要があります。この移行方法は、オンプレミスデータベースまたは Amazon Elastic Compute Cloud (Amazon EC2) で実行されているデータベースでは有効ではありません。また、5.6 以前のバージョンの Amazon RDS MySQL データベースを実行している場合は、前提条件として 5.6 にアップグレードする必要があります。

この移行方法の最大の利点は、これが最もシンプルで少ないステップで済むということです。特に、すべてのデータベースデータとともに、すべてのスキーマオブジェクト、セカンダリインデックス、およびストアドプロシージャが移行されます。

binlog レプリケーションを行わないスナップショットの移行中に、ソースデータベースはオフラインであるか、読み取り専用モード (移行中にソースデータベースに変更が行われないようにするため) である必要があります。ダウンタイムを予測するには、データベースの既存のスナップショットを使用してテスト移行を行います。移行時間がダウンタイムの要件に合う場合、これが最適の方法である可能性があります。場合によっては、AWS DMS またはネイティブな移行ツールを使用した移行が、スナップショットの移行よりも高速である可能性があります。

ダウンタイムが長くなることが許容できない場合は、ソースデータベースの更新を継続しながら、最初に RDS データベースのスナップショットを Amazon Aurora に移行することで、ダウンタイムがゼロに近い移行を達成できます。次に、MySQL から Aurora への MySQL binlog レプリケーションを使用して、これを最新の状態にします。

手動で作成された DB スナップショットと自動的に作成された DB スナップショットのどちらも移行できます。実行する必要がある一般的な手順は次のとおりです。

1. Amazon RDS MySQL 5.6 インスタンスを Aurora DB クラスターに移行するために必要な容量を決定します。詳細については、次のセクションを参照してください。
2. Amazon RDS コンソールを使用して、Amazon RDS MySQL 5.6 インスタンスが配置されているリージョン内にスナップショットを作成します。
3. コンソールで**データベースの移行**機能を使用して、MySQL 5.6 の元の DB インスタンスの DB スナップショットを使用して入力される Amazon Aurora DB クラスターを作成します。

注意: 一部の MyISAM テーブルではエラーなしで変換されず、手動の変更が必要になる場合があります。たとえば、InnoDB エンジンでは自動増分フィールドが複合キーの一部となることは許可されません。また、空間インデックスは現在サポートされていません。

スナップショット移行のスペース要件の予測

MySQL DB インスタンスのスナップショットを Aurora DB クラスターに移行するとき、Aurora は、スナップショットのデータを移行する前に Amazon Elastic Block Store (Amazon EBS) ボリュームを使用してそのデータの書式を設定します。移行するデータの書式を設定するために追加容量が必要になる場合があります。移行中に容量の問題を発生させる可能性のある 2 つの機能は、MyISAM テーブルと `ROW_FORMAT=COMPRESSED` オプションの使用です。ソースデータベースでこれらのいずれの機能も使用していない場合は、容量の問題は発生しないため、このセクションを省略できます。移行中に、MyISAM テーブルは InnoDB に変換され、圧

縮されたテーブルは圧縮解除されます。その結果、このようなテーブルの追加のコピー用の適切な領域が必要になります。

移行ボリュームのサイズは、スナップショットの作成元のソース MySQL データベースの割り当てられたサイズに基づきます。したがって、全体的なデータベースサイズの小さい割合を占める MyISAM または圧縮されたテーブルがあり、元のデータベースに利用可能なスペースがある場合、移行は容量の問題が発生することなく成功します。ただし、変換された MyISAM テーブルのコピーと、圧縮されたテーブルの別の (圧縮されていない) コピーを保存するための十分な領域が元のデータベースにない場合、移行ボリュームは十分に大きくなりません。この状況では、ソースの Amazon RDS MySQL データベースを変更してデータベースのサイズ割り当てを増やし、これらのテーブルの追加のコピー用の領域を作成します。また、データベースの新しいスナップショットを作成し、新しいスナップショットを移行します。

DB クラスターにデータを移行するときは、以下のガイドラインと制限を確認してください。

- Amazon Aurora は最大 64 TB のストレージをサポートしますが、Aurora DB クラスターにスナップショットを移行するプロセスはスナップショットの Amazon EBS ボリュームのサイズによって制限されるため、最大サイズは 6 TB に制限されます。
- ソースデータベースの MyISAM 以外のテーブルの最大サイズは 6 TB です。ただし、変換中の追加のスペース要件により、MySQL DB インスタンスから移行される MyISAM テーブルおよび圧縮テーブルのサイズがいずれも 3 TB を超えていないことを確認してください。

Amazon Aurora に移行する前に、データベーススキーマを変更する (MyISAM テーブルを InnoDB に変換し、`ROW_FORMAT=COMPRESSED` を削除する) ことをお勧めします。これは、次のような場合に便利です。

- 移行プロセスを高速化する必要がある場合。
- プロビジョニングするために必要な領域の量がわからない場合。
- データを移行しようとしたが、プロビジョニング済み領域の不足で移行が失敗した場合。

これらの更新は、本稼働 Amazon RDS MySQL データベースではなく、本稼働スナップショットから復元されたデータベースインスタンスに対して実行します。この詳細については、『*Amazon RDS* ユーザーガイド』の「Amazon Aurora にデータを移行するために必要な領域の縮小」を参照してください。[5]

コンソールを使用した DB スナップショットの移行

Amazon RDS MySQL DB インスタンスの DB スナップショットを移行して、Aurora DB クラスターを作成することができます。新しい DB クラスターには、元の Amazon RDS MySQL DB インスタンスのデータが入力されます。DB スナップショットは、MySQL 5.6 を実行している RDS DB インスタンスから作成され、暗号化される必要があります。DB スナップショットの作成に関する詳細については、『*Amazon RDS* ユーザーガイド』の「DB スナップショットの作成」を参照してください。[6]

DB スナップショットが Aurora DB クラスターを検索するリージョン内にない場合は、Amazon RDS コンソールを使用して DB スナップショットをそのリージョンにコピーします。DB スナップショットのコピーに関する詳細については、『*Amazon RDS* ユーザーガイド』の「DB スナップショットのコピー」を参照してください。[7]

コンソールを使用して MySQL 5.6 DB スナップショットを移行するには、次の操作を行います。

1. AWS マネジメントコンソールにサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. [**Snapshots**] を選択します。
3. [**Snapshots**] ページで、Aurora DB クラスターに移行する Amazon RDS MySQL 5.6 スナップショットを選択します。
4. [**Migrate Database**] を選択します。
5. [**Migrate Database**] ページで、次の図に示すように、環境と処理の要件に合った値を指定します。これらのオプションの詳細については、『*Amazon RDS* ユーザーガイド』の「コンソールを使用した DB スナップショットの移行」を参照してください。[8]

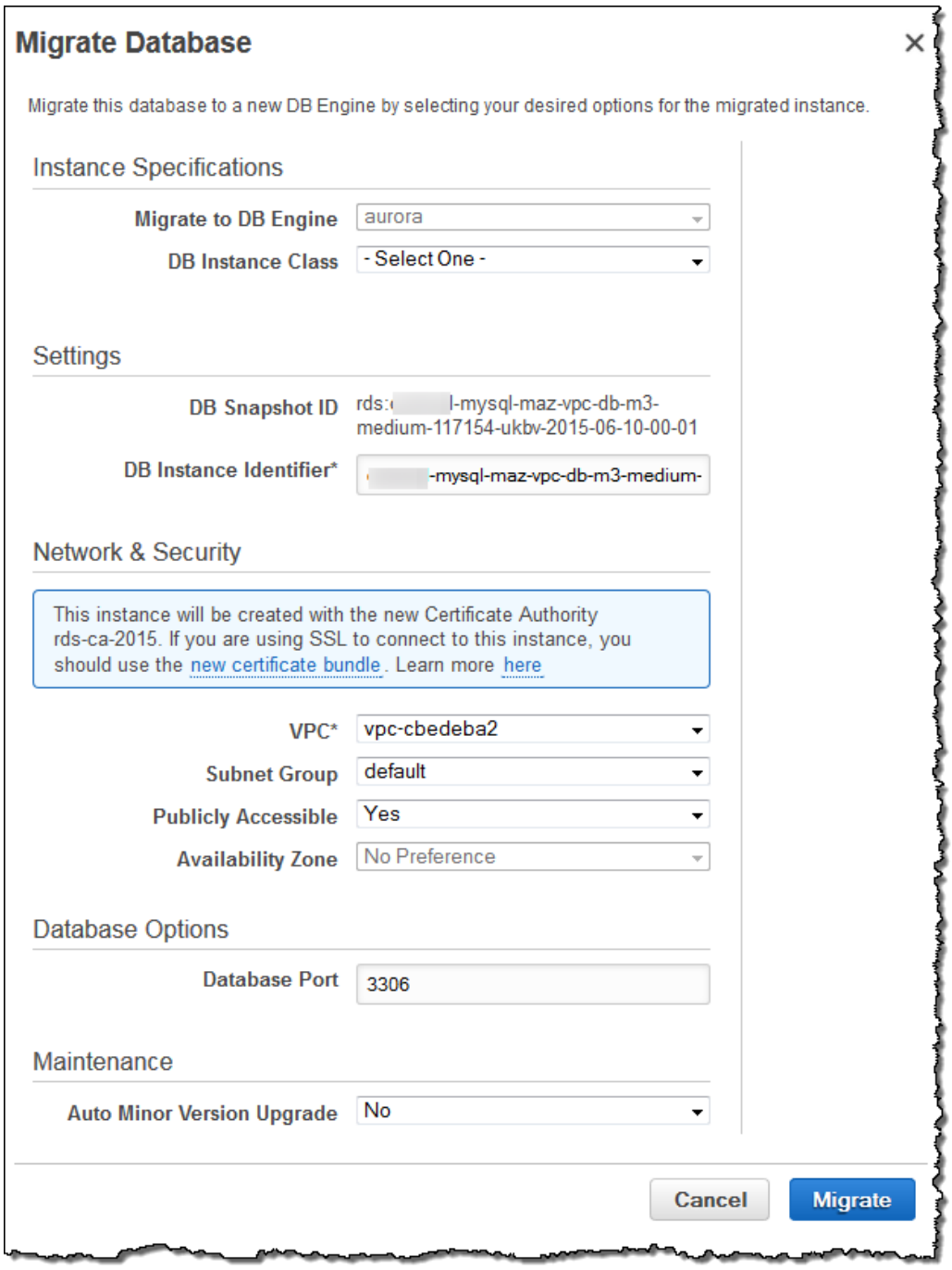

図 2: スナップショットの移行

6. [**Migrate**] をクリックして、DB スナップショットを移行します。

インスタンスの一覧で、適切な矢印アイコンをクリックして DB クラスターの詳細を表示し、移行の進行状況をモニタリングします。この詳細パネルには、DB クラスターのプライマリインスタンスへの接続に使用されているクラスターエンドポイントが表示されます。Amazon Aurora DB クラスターに接続する方法の詳細については、『*Amazon RDS* ユーザーガイド』の「Amazon Aurora DB クラスターへの接続」を参照してください。[9]

データベーススキーマの移行

RDS DB スナップショットの移行では、完全なスキーマとデータが新しい Aurora インスタンスに移行されます。ただし、ソースデータベースの場所またはアプリケーションの稼働時間の要件により RDS スナップショットの移行の使用が許可されない場合、実際のデータを移動する前に、まずデータベーススキーマをソースデータベースからターゲットデータベースに移行する必要があります。データベーススキーマは、データベース全体の論理ビューを表すスケルトン構造であり、通常は以下が含まれます。

- データベースストレージオブジェクト: テーブル、列、制約、インデックス、シーケンス、ユーザー定義型、およびデータ型
- データベースオブジェクト: 関数、プロシージャ、パッケージ、トリガー、ビュー、マテリアライズドビュー、イベント、SQL スカラー関数、SQL インライン関数、SQL テーブル関数、属性、変数、定数、テーブル型、パブリック型、プライベート型、カーソル、例外、パラメーター、およびその他のオブジェクト

ほとんどの状況で、データベーススキーマは相対的に静的なままになるため、データベーススキーマの移行ステップ中にダウンタイムは必要ありません。ソースデータベースからのスキーマは、ソースデータベースの稼働中に、パフォーマンスに影響を与えることなく抽出できます。アプリケーションまたは開発者がデータベーススキーマを頻繁に変更する場合は、それらの変更が移行中に一時停止されるか、スキーマの移行プロセス中に考慮されるようにします。

ソースデータベースの型に応じて、次のセクションで説明するテクニックを使用してデータベーススキーマを移行できます。スキーマ移行の前提条件として、ターゲットの Aurora データベースを作成し、利用可能にしておく必要があります。

同機種のスキーマの移行

ソースデータベースが MySQL 5.6 に対応していて、Amazon RDS、Amazon EC2 で実行されているか、AWS 外部で実行されている場合は、ネイティブ MySQL ツールを使用してスキーマをエクスポートおよびインポートできます。

- **データベーススキーマのエクスポート。**mysqldump クライアントユーティリティを使用してデータベーススキーマをエクスポートできます。このユーティリティを実行するには、ソースデータベースに接続し、`mysqldump` コマンドの出力をフラットファイルにリダイレクトする必要があります。`-no-data` オプションでは、実際のテーブルデータなしで、データベーススキーマのみがエクスポートされます。完全な `mysqldump` コマンドのリファレンスについては、「mysqldump— データベースバックアッププログラム」を参照してください。[10]

```
mysqldump –u source_db_username –p --no-data --routines --
triggers –databases source_db_name > DBSchema.sql
```

- **データベーススキーマの Aurora へのエクスポート。**スキーマを Aurora インスタンスにインポートするには、MySQL コマンドラインクライアント (または対応する Windows クライアント) から Aurora データベースに接続し、エクスポートファイルのコンテンツを MySQL にダイレクトします。

```
mysql –h aurora-cluster-endpoint –u username –p <
DBSchema.sql
```

次のことに注意してください。

- ソースデータベースにストアドプロシージャ、トリガー、およびビューが含まれる場合、ダンプファイルから `DEFINER` 構文を削除する必要があります。これを行うシンプルな Perl コマンドを次に示します。これを行うと、現在の接続されたユーザーが `DEFINER` として、すべてのトリガー、ビュー、およびストアドプロシージャが作成されます。必ず、これによって発生する可能性のあるセキュリティへの影響を評価してください。

```
$perl -pe 's/\sDEFINER=`[^`]+`@`[^`]+`//' < DBSchema.sql >
DBSchemaWithoutDEFINER.sql
```

- Amazon Aurora は InnoDB テーブルのみをサポートします。ソースデータベースに MyISAM テーブルがある場合、`CREATE TABLE` コマンドが実行されると、Aurora は自動的にエンジンを InnoDB に変更します。

- Amazon Aurora では、圧縮テーブル (ROW_FORMAT=COMPRESSED を使用して作成されたテーブル) をサポートしていません。ソースデータベースに圧縮されたテーブルがある場合、CREATE TABLE コマンドが実行されると、Aurora は自動的に ROW_FORMAT を COMPACT に変更します。

MySQL 5.6 互換ソースデータベースから Amazon Aurora にスキーマを正常にインポートしたら、次のステップではソースからターゲットに実際のデータをコピーします。詳細については、後の「AWS DMS の概要と一般的な手法」を参照してください。

異機種のスキーマの移行

ソースデータベースが MySQL と互換性がない場合、Amazon Aurora と互換性のある形式にスキーマを変換する必要があります。1 つのデータベースエンジンから別のデータベースエンジンへのスキーマの変換は簡単ではないタスクであり、データベースおよびアプリケーションコードの特定の部分の再記述が必要になる場合があります。スキーマを Amazon Aurora に変換および移行するためには、2 つの主要なオプションがあります。

- **AWS スキーマ変換ツール。**AWS スキーマ変換ツールは、ソースデータベーススキーマと大部分のカスタムコード (ビュー、ストアドプロシージャ、関数を含む) をターゲットデータベースと互換性のある形式に自動的に変換して、異機種のデータベース移行を容易にします。[11]自動的に変換できないコードは明確にマークされるため、手動で変換できます。このツールを使用して、Oracle または Microsoft SQL Server で実行されているソースデータベースを、Amazon Aurora、MySQL、または Amazon RDS あるいは Amazon EC2 の PostgreSQL ターゲットデータベースに変換できます。AWS スキーマ変換ツールを使用して、Oracle、SQL Server、または PostgreSQL スキーマを Aurora 互換形式に変換する方法をお勧めします。

- **手動のスキーマの移行とサードパーティー製ツール。**ソースデータベースが Oracle、SQL Server、または PostgreSQL でない場合は、手動でソースデータベーススキーマを Aurora に移行するか、サードパーティー製ツールを使用して、MySQL 5.6 と互換性のある形式にスキーマを移行できます。手動のスキーマの移行は、ソーススキーマのサイズと複雑さによっては、かなり複雑なプロセスとなる場合があります。ただし、ほとんどの場合、手動のスキーマ変換は、Amazon Aurora による費用の節約、パフォーマンスの利点、その他の改善を考慮すると実行する価値があります。

AWS スキーマ変換ツールを使用したスキーマの移行

AWS スキーマ変換ツールは、ソースデータベースのデータベーススキーマを、Amazon Aurora と互換性のある形式に自動的に変換するためのプロジェクトベースのユーザーインターフェイスを提供します。AWS スキーマ変換ツールを使用してデータベースの移行の労力を評価し、実際の本番移行を実行する前に、パイロット移行を行うことを強くお勧めします。

以下の説明では、AWS スキーマ変換ツールの使用方法の概要を示しています。詳細な手順については、『*AWS Schema Conversion Tool User Guide*』の「Getting Started」セクションを参照してください。[12]

1. 最初に、ツールをインストールします。AWS スキーマ変更ツールは、Microsoft Windows、Mac OS X、Ubuntu Linux、および Fedora Linux で利用できます。

   ダウンロードとインストールの詳細な手順は、ユーザーガイドの「インストールおよび更新」セクションを参照してください。[13]AWS スキーマ変換ツールをインストールする場所も重要です。このツールは、スキーマを変換して適用するには、ソースデータベースとターゲットデータベースに直接接続する必要があります。AWS スキーマ変更ツールをインストールするデスクトップで、ソースデータベースとターゲットデータベースの両方とのネットワーク接続が可能であることを確認します。

2. JDBC ドライバをインストールします。AWS スキーマ変更ツールは、JDBC ドライバを使用してソースデータベースとターゲットデータベースに接続します。このツールを使用するには、これらの JDBC ドライバをローカルデスクトップにダウンロードする必要があります。ドライバのダウンロードの手順については、『*AWS Schema Conversion Tool User Guide*』の「Required Database Drivers」セクションを参照してください。[14]また、さまざまなデータベースエンジン用に JDBC ドライバを設定する手順について、AWS スキーマ変換ツールに関する AWS フォーラムを参照してください。[15]

3. ターゲットデータベースを作成します。Amazon Aurora ターゲットデータベースを作成します。Amazon Aurora データベースを作成する手順については、『*Amazon RDS* ユーザーガイド』の「Amazon Aurora DB クラスターの作成」を参照してください。[16]

4. AWS スキーマ変換ツールを開き、**New Project ウィザード**を起動します。

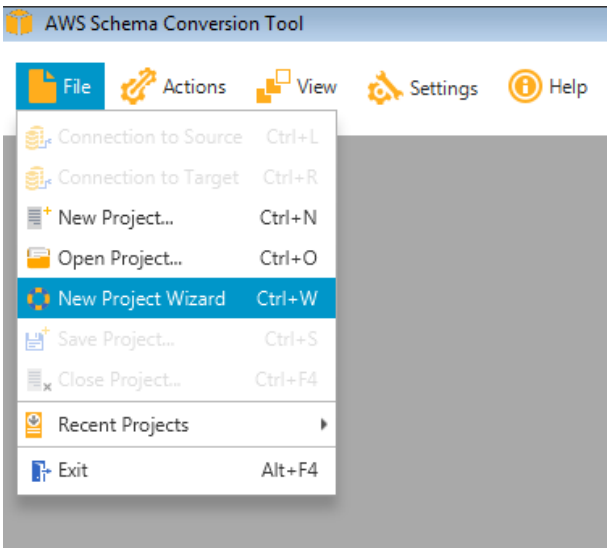

図 3: 新しい AWS スキーマ変換ツールプロジェクトの作成

5. ソースデータベースを設定し、AWS スキーマ変更ツールとソースデータベースの間の接続をテストします。このためには、ソースデータベースはデスクトップから到達可能である必要があるため、適切なネットワーク設定およびファイアウォール設定が適用されていることを確認します。

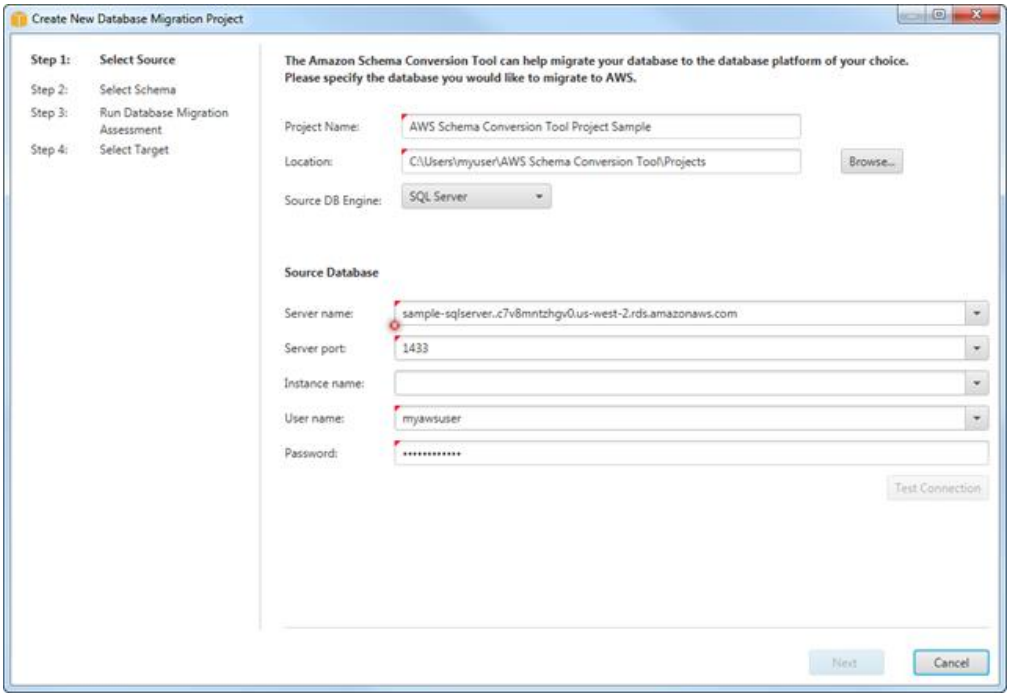

図 4: 新しいデータベース移行プロジェクトウィザードの作成

6. 次の画面で、Amazon Aurora に変換するソースデータベースのスキーマを選択します。

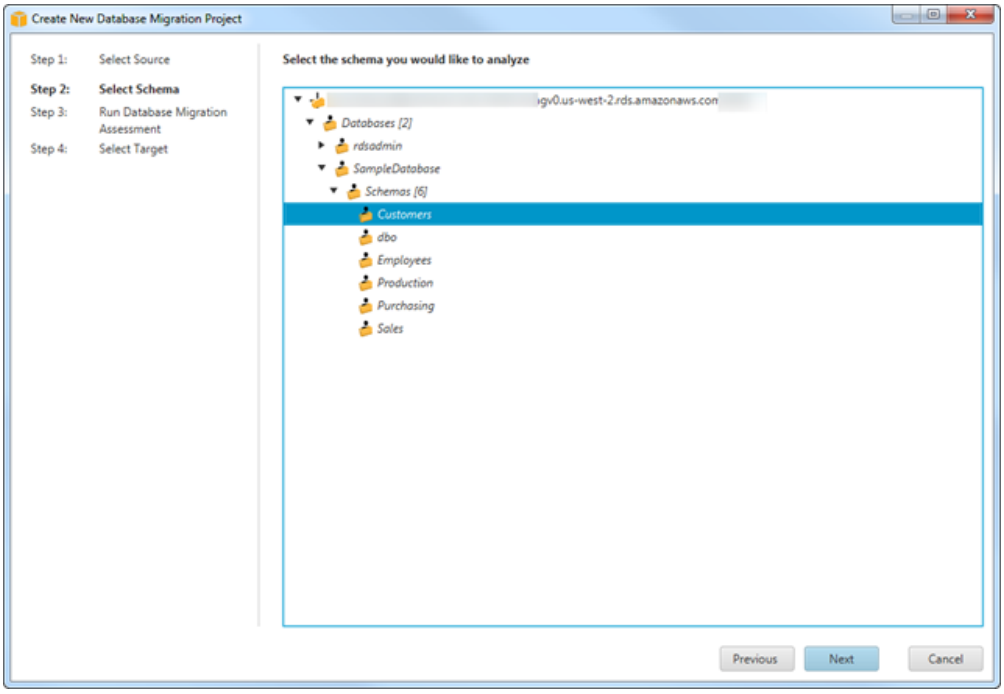

図 5: 移行ウィザードのスキーマステップの選択

7. データベース移行評価レポートを実行します。このレポートでは、ソースデータベースからターゲットの Amazon Aurora インスタンスへのスキーマの変換に関する重要な情報が提供されます。すべてのスキーマ変換タスクの概要と、自動的に Aurora に変換できないスキーマ部分のアクション項目の詳細が示されます。このレポートには、自動的に変換されなかったターゲットデータベースで同等のコードを記述するために必要な労力の予測も含まれます。

   [Next] をクリックしてターゲットデータベースを設定します。後でこの移行レポートをもう一度表示できます。

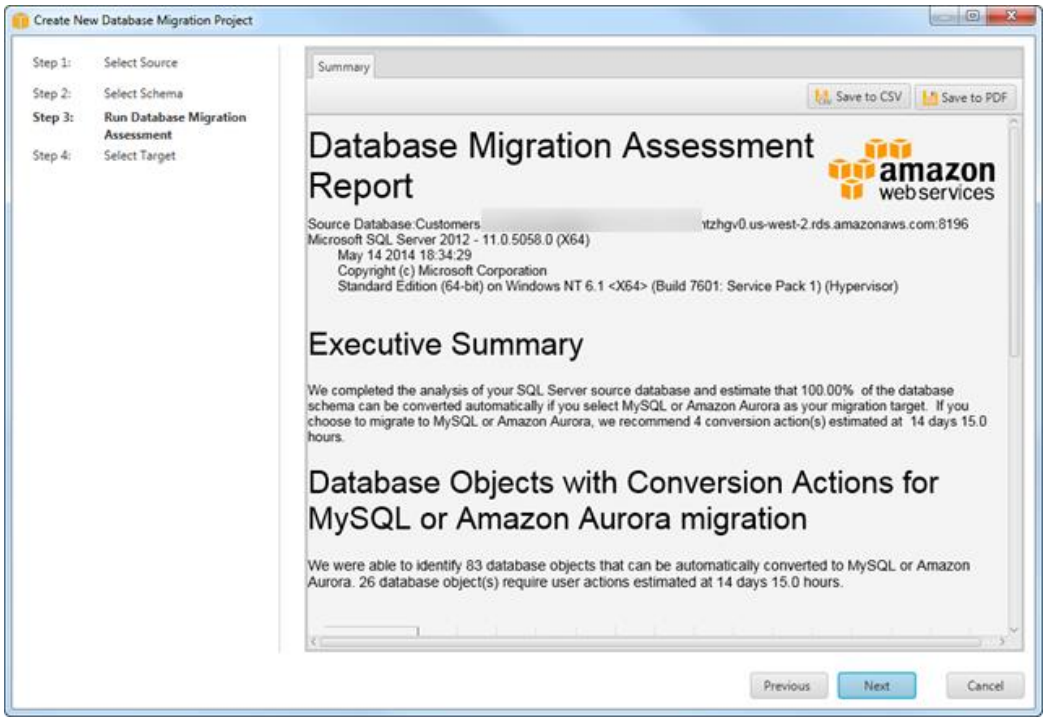

図 6: 移行レポート

8. ターゲットの Amazon Aurora データベースを設定し、AWS スキーマ変更ツールとソースデータベースの間の接続をテストします。このためには、ターゲットデータベースはデスクトップから到達可能である必要があるため、適切なネットワーク設定およびファイアウォール設定が適用されていることを確認します。[**Finish**] をクリックしてプロジェクトウィンドウに移動します。

9. プロジェクトウィンドウに移動すると、ソースデータベースおよびターゲットデータベースへの接続が確立されています。これで、詳細な評価レポートを評価し、スキーマを移行する準備ができました。

10. ソースデータベースからスキーマを表示する左のパネルで、評価レポートを作成するスキーマオブジェクトを選択します。オブジェクトを右クリックし、[**Create Report**] を選択します。

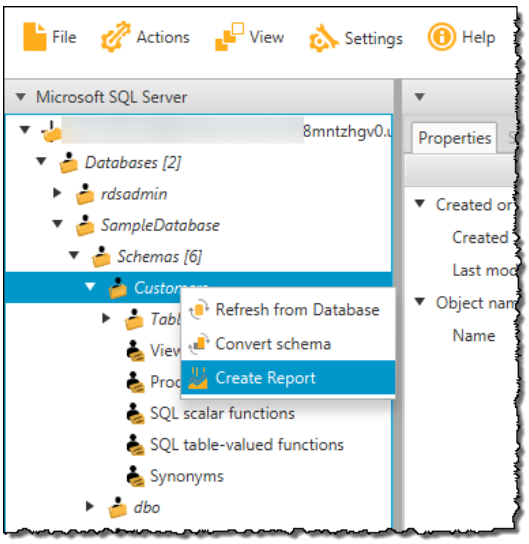

図 7: 移行レポートの作成

[**Summary**] タブには、データベース移行評価レポートからの概要情報が表示されます。自動的に変換された項目と、自動的に変換できなかった項目が表示されます。

ターゲットデータベースエンジンに自動的に変換できなったスキーマ項目について、概要にはターゲット DB インスタンスでソースデータベースと同等のスキーマを作成するために必要な労力の予測が含まれます。レポートには、これらのスキーマ項目を変換するための予測時間が次のように分類されます。

- **Simple** – 1 時間以内に完了できるアクション。
- **Medium** – より複雑で、1～4 時間以内に完了できるアクション。
- **Significant** – 非常に複雑で、完了に 4 時間以上かかるアクション。

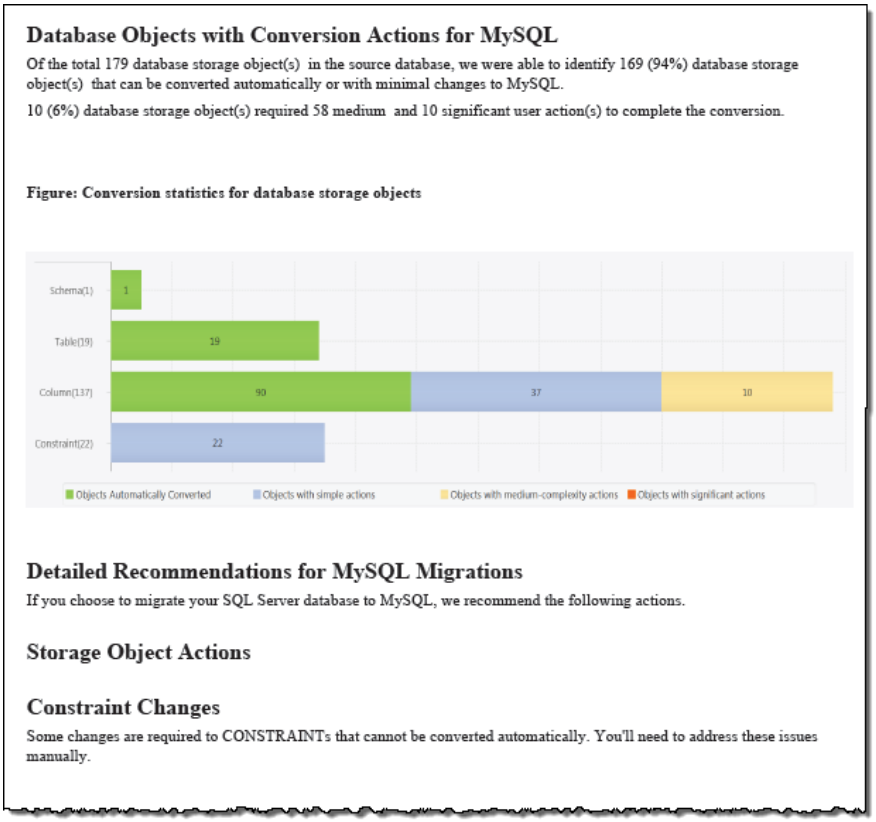

図 8: 移行レポート

重要: データベース移行プロジェクトで必要な労力を評価している場合、この評価レポートは検討する重要なアーティファクトとなります。評価レポートを詳細に調べて、データベーススキーマで必要なコードの変更と、アプリケーションの機能と設計への変更の影響について確認します。

11. 次のステップは、スキーマの変換です。変換されたスキーマは、ターゲットデータベースにすぐに適用されません。変換されたスキーマをターゲットデータベースに明示的に適用するまで、ローカルに保存されます。ソースデータベースからスキーマを変換するには、プロジェクトの左のパネルで、変換するスキーマオブジェクトを選択します。次の図に示すように、オブジェクトを右クリックし、[**Convert schema**] を選択します。

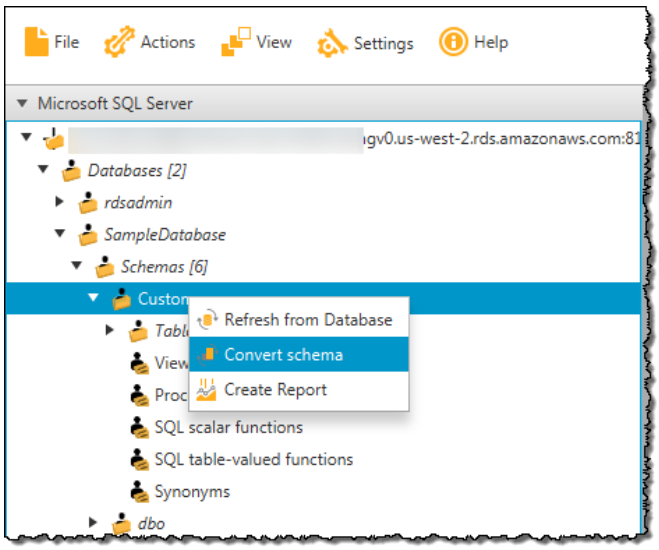

図 9: スキーマの変換

このアクションでは、変換されたスキーマがプロジェクトウィンドウの右のパネルに追加され、AWS スキーマ変換ツールによって自動的に変換されたオブジェクトが表示されます。

12. 評価レポートのアクション項目には、さまざまな方法で対応できます。

- 同等のスキーマを手動で追加します。プロジェクトの右側のパネルで [**Apply to database**] を選択すると、ターゲット DB インスタンスに自動的に変換できるスキーマの部分を記述できます。ターゲット DB インスタンスに書き込まれるスキーマには、自動的に変換できなかった項目は含まれません。これらの項目は、データベース移行評価レポートで一覧表示されます。

  スキーマをターゲット DB インスタンスに適用したら、自動的に変換できなかった項目について、ターゲット DB インスタンスでスキーマを手動で作成できます。場合によっては、ターゲット DB インスタンスで同等のスキーマを作成できないことがあります。アプリケーションとデータベースの一部を再設計し、DB エンジンから利用できる機能をターゲット DB インスタンスに対して使用しなければならない場合があります。その他の場合、自動的に変換できないスキーマは無視できます。

注意: ターゲット DB インスタンスでスキーマを手動で作成できない場合、完了した手動の作業のコピーを保存するまでは、[**Apply to database**] を選択しないでください。プロジェクトからターゲット DB インスタンスにスキーマを適用すると、ターゲット DB インスタンスの同じ名前のスキーマが上書きされ、手動で追加した更新プログラムは失われます。

- ソースデータベーススキーマを変更し、プロジェクトでスキーマを更新します。一部の項目について、ソースデータベースのデータベーススキーマを、アプリケーションアークテクチャと互換性のあるスキーマに変更し、ターゲット DB インスタンスの DB エンジンに自動的に変換できるようにすると最適な場合があります。ソースデータベースでスキーマを更新し、更新プログラムがアプリケーションと互換性のあることを確認したら、プロジェクトの左のパネルの [**Refresh from Database**] を選択して、ソースデータベースからスキーマを更新します。次に、更新されたスキーマを変換し、もう一度データベース移行評価レポートを生成できます。更新されたスキーマのアクション項目は表示されなくなります。

13. 変換されたスキーマをターゲット Aurora インスタンスに適用する準備ができたら、プロジェクトの右側のパネルからスキーマ要素を選択します。次の図に示すように、スキーマ要素を右クリックし、[**Apply to database**] を選択します。

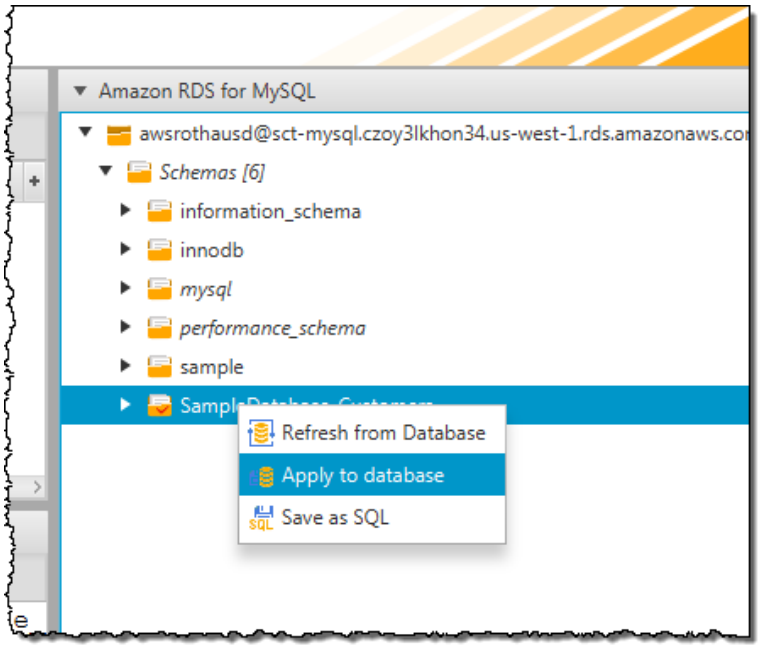

図 10: データベースへのスキーマの適用

注意: 変換されたスキーマを初めてターゲット DB インスタンスに適用するときに、AWS スキーマ変換ツールは追加のスキーマ (`AWS_ORACLE_EXT` または `AWS_SQLSERVER_EXT`) をターゲット DB インスタンスに追加します。このスキーマは、変換されたスキーマをターゲット DB インスタンスに書き込むときに必要なソースデータベースのシステム関数を実装します。このスキーマは変更しないでください。変更すると、ターゲット DB インスタンスに書き込まれる、変換されたスキーマで予期しない結果が発生する可能性があります。スキーマがターゲット DB インスタンスに完全に移行され、AWS スキーマ変換ツールが必要なくなったときは、`AWS_ORACLE_EXT` または `AWS_SQLSERVER_EXT` スキーマを削除できます。

AWS スキーマ変更ツールは、移行ツールキットに簡単に追加できます。AWS スキーマ変換ツールに関連するその他のベストプラクティスについては、『*AWS Schema Conversion Tool User Guide*』の「Best Practices」トピックを参照してください。[17]

データの移行

データベーススキーマがソースデータベースからターゲットの Aurora データベースにコピーされたら、次のステップではソースからターゲットに実際のデータを移行します。データ移行はさまざまなツールを使用して達成できますが、AWS DATABASE MIGRATION SERVICE(AWS DMS) を使用してデータを移動することをお勧めします。このサービスでは、タスクで必要なわかりやすさと機能の両方が提供されます。

AWS DMS の概要と全般的な手法

AWS DATABASE MIGRATION SERVICE(AWS DMS) を使用すると、お客様はプロダクションデータベースを最小のダウンタイムで簡単に AWS に移行できます。データベースの移行中に、アプリケーションの実行を続行できます。さらに、AWS データベース移行サービスでは、移行中および移行後に発生するソースデータベースへのデータ変更が、連続してターゲットにレプリケートされます。移行タスクは、AWS マネジメントコンソールから、数分で設定できます。AWS データベース移行サービスでは、Oracle、SQL Server、MySQL、PostgreSQL、Amazon Aurora、MariaDB、Amazon Redshift など、幅広く使用されているデータベースプラットフォームとの間でデータを移行できます。

このサービスでは、Oracle から Oracle への同機種の移行をサポートしているほか、Oracle から Amazon Aurora、SQL Server から MySQL など、異なるデータベースプラットフォーム間の異機種の移行もサポートしています。1 回のみの移行を実行するか、複雑なソフトウェアをインストールまたは設定しなくても、データベースとの間で連続的なレプリケーションを維持できます。

AWS DMS は、オンプレミスのデータベース、Amazon EC2 で実行されるデータベース、または Amazon RDS で実行されるデータベースと連携します。ただし、AWS DMS は、ソースデータベースとターゲットデータベースの両方がオンプレミスである状況では動作しません。1 つのエンドポイントが AWS 内にある必要があります。

AWS DMS は、Oracle、SQL Server、Amazon Aurora、MySQL、および PostgreSQL の特定のバージョンをサポートします。現在サポートされているバージョンについては、『AWS Database Migration Service User Guide』を参照してください。[18]このホワイトペーパーでは、移行ターゲットとして Amazon Aurora について説明しているのみです。

移行方法

AWS DMS では、データの移行のための 3 つの方法があります。

既存のデータを移行する。この方法では、ターゲットデータベースにテーブルを作成し、ターゲットで必要なメタデータを自動的に定義して、ソースデータベースのデータをテーブルに入力します (「フルロード」とも呼ばれます)。テーブルのデータは、効率を高めるために並列でロードされます。テーブルが作成されるのは異機種の移行の場合のみで、AWS DMS によって自動的にセカンダリインデックスを作成することはできません。詳細については、以下をお読みください。

既存のデータを移行し、継続的な変更をレプリケートする。この方法では、上記のフルロードを行い、それに加えて、フルロード中にソースデータベースに加えられた継続的な変更をキャプチャし、それらをレプリケーションインスタンスに保存します。フルロードが完了すると、保存された変更はソースデータベースに対して最新の状態になるまで、ターゲットデータベースに適用されます。さらに、ソースデータベースに対して行われた継続的な変更は、同期を保つためにターゲットデータベースに対して継続してレプリケートされます。この移行方法は、ほとんどダウンタイムを発生させずにデータベース移行を実行する場合に非常に有効です。

データ変更のみをレプリケートする。この方法では、ソースデータベースの復旧ログファイルから変更を読み取り、それらの変更を継続的にターゲットデータベースに適用します。ターゲットデータベースが利用できない場合、それらの変更は、ターゲットが利用可能になるまでレプリケーションインスタンスでバッファされます。

AWS DMS がフルロード移行を行っているときに、ソースデータベースのテーブルに負荷がかかります。これにより、同時にこのデータベースを対象とするアプリケーションのパフォーマンスに影響が及ぶ場合があります。これが問題となり、移行中にアプリケーションをシャットダウンできない場合は、以下の手法を検討してください。

- データベースへのアプリケーションの負荷が最も低いポイントのときに移行を実行する。
- ソースデータベースのリードレプリカを作成し、リードレプリカから AWS DMS 移行を実行する。

移行手順

AWS DMS を使用する一般的な概要は、次のとおりです。

1. ターゲットデータベースを作成します。
2. スキーマをコピーします。
3. AWS DMS レプリケーションインスタンスを作成します。
4. データベースのソースとターゲットエンドポイントを定義します。
5. 移行タスクを作成し、実行します。

ターゲットデータベースの作成

『*Amazon RDS* ユーザーガイド』の「Amazon Aurora DB クラスターの作成」に示している手順を使用して、ターゲット Amazon Aurora データベースクラスターを作成します。[19]リージョンで、ビジネス要件に合ったインスタンスタイプを使用してターゲットデータベースを作成します。また、移行のパフォーマンスを向上させるため、ターゲットデータベースでマルチ AZ 配置が有効でないことを確認します。ロードが終了したら、これを有効にできます。

スキーマのコピー

さらに、このターゲットデータベースでスキーマを作成する必要があります。AWS DMS は、テーブルとプライマリキーの作成を含めて、基本的なスキーマの移行をサポートします。ただし、AWS DMS は、ターゲットデータベースのセカンダリインデックス、外部キー、ストアドプロシージャ、ユーザーアカウントなどは自動的に作成しません。完全なスキーマ移行の詳細については、「データベーススキーマの移行」セクションを参照してください。

AWS DMS レプリケーションインスタンスの作成

AWS DMS サービスを使用するには、VPC で実行される AWS DMS レプリケーションインスタンスを作成する必要があります。このインスタンスはソースデータベースからデータを読み取り、指定されたテーブルマッピングを実行して、ターゲットデータベースにデータを書き込みます。通常、大きなレプリケーションインスタンスサイズを使用すると、データベースの移行の速度が上がります (ただし、移行はソースデータベースとターゲットデータベースの容量、接続のレイテンシーなどその他の要素による影響を受ける場合もあります)。また、データベースの移行が完了したら、レプリケーションインスタンスを停止できます。

図 11: AWS データベース移行サービス

現在、AWS DMS はレプリケーションインスタンス用に T2 および C4 インスタンスクラスをサポートしています。T2 インスタンスはベースラインレベルの CPU パフォーマンスを提供しながら、そのベースラインレベルを超えてバーストする機能を備えるよう設計された、低コストのスタンダードインスタンスです。このインスタンスは、データベース移行プロセスの開発、設定、テスト、および CPU のバースト機能の利点を活用できる定期的なデータ移行タスクに適しています。C4 インスタンスクラスは、最高レベルのプロセッサパフォーマンスを提供するとともに、非常に高いパケット毎秒 (PPS) パフォーマンス、低いネットワークジッター、および低いネットワークのレイテンシーを実現します。大きなデータベースを移行し、移行時間を最小化したい場合は、C4 インスタンスクラスを使用してください。

通常、フルロードを実行しても、AWS DMS レプリケーションインスタンスで多くのインスタンスストレージを必要としません。ただし、フルロードでレプリケーションを行っている場合、ソースデータベースへの変更は、フルロードの実行中は AWS DMS レプリケーションインスタンスに保存されます。したがって、移行の進行中に多くの更新を受け取る非常に大きなソースデータベースを移行する場合、大量のインスタンスストレージが消費される可能性があります。C4 インスタンスファミリーには 100 GB のインスタンスストレージがあり、T2 インスタンスファミリーには 50 GB のインスタンスストレージがあります。通常、これらのストレージ容量は、ほとんどの移行シナリオに対して十分以上です。

また、非常に高いトランザクションレートを持つ非常に大きいデータベースがレプリケーションを有効にして移行されている場合、AWS DMS レプリケーションが時間に追い付かない可能性があります。このような状況が発生した場合、レプリケーションが追いつき、アプリケーションがターゲットの Aurora DB を再度指すまでに、数分間ソースデータベースへの変更を停止しなければならない可能性があります。

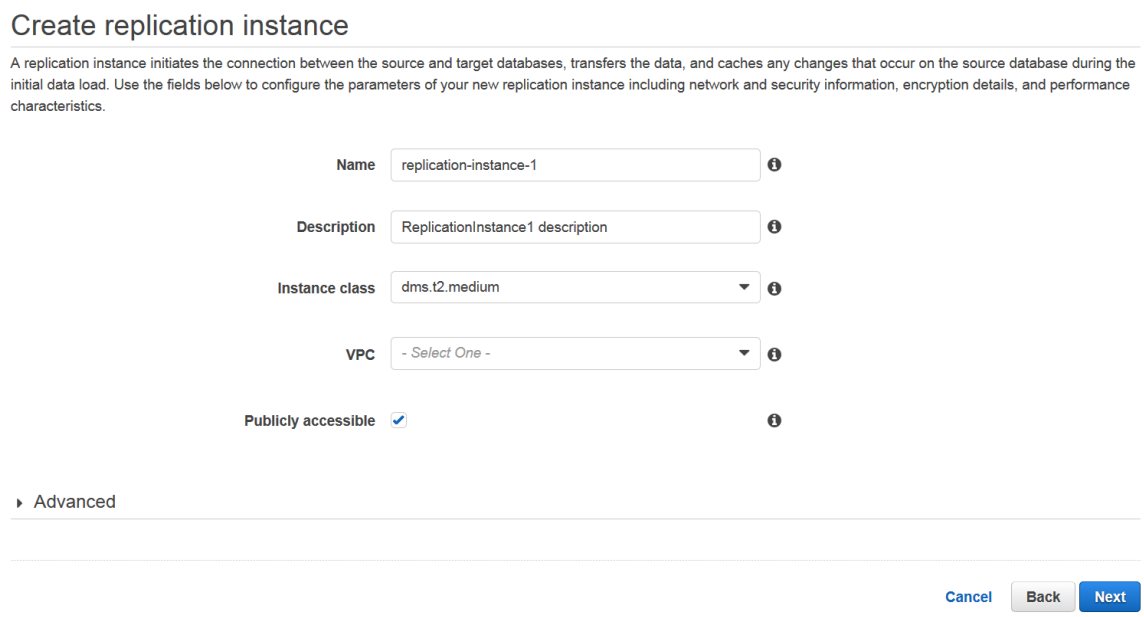

図 12: AWS DMS コンソールでのレプリケーションインスタンスページの作成

データベースのソースとターゲットエンドポイントの定義

データベースエンドポイントは、データベースに接続するためにレプリケーションインスタンスによって使用されます。データベース移行を実行するには、ソースデータベースエンドポイントとターゲットデータベースエンドポイントの両方を作成する必要があります。指定するデータベースエンドポイントは、オンプレミス、Amazon EC2 で実行、Amazon RDS で実行するものとすることができますが、ソースとターゲットの両方をオンプレミスとすることはできません。

定義後に、データベースエンドポイント接続をテストすることを強くお勧めします。後で説明するように、データベースエンドポイントの作成に使用されたのと同じページを、このテストに使用できます。

注意: ソーススキーマに外部キー制約がある場合、ターゲットエンドポイントを作成するときは、[**Advanced**] セクションの [**Extra connection attributes**] に以下を入力する必要があります。

```
initstmt=SET FOREIGN_KEY_CHECKS=0
```

これにより、ターゲットテーブルのロード中に外部キーのチェックが無効になります。その結果、一部がロードされたテーブルで失敗した外部キーチェックによってロードが中断されることがなくなります。

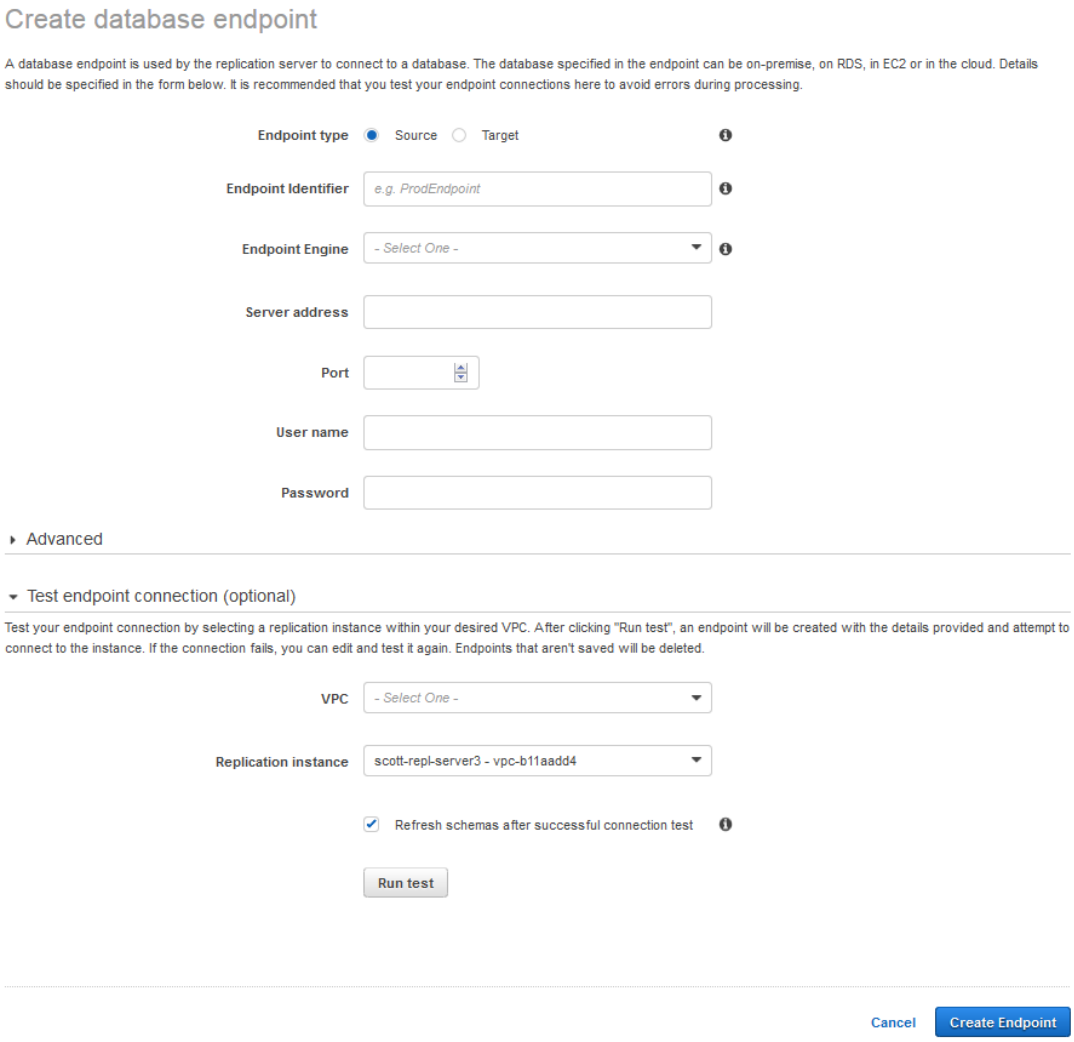

図 13: AWS DMS コンソールでのデータベースエンドポイントページの作成

移行タスクの作成と実行

これでソースデータベースエンドポイントとターゲットデータベースエンドポイントを作成およびテストしたので、データ移行を実行するタスクを作成できます。タスクを作成するときは、Amazon Aurora データベースクラスター用に作成したレプリケーションインスタンス、データベースの移行方法の種類 (前に説明)、ソースデータベースポイント、およびターゲットデータベースエンドポイントを指定します。

また、[**Task Settings**] で、すでにターゲットデータベースでフルスキーマを作成している場合は、[**Target table preparation mode**] でデフォルト値の [**Drop tables on target**] を使用するのではなく、[**Do nothing**] に変更する必要があります。デフォルト値を使用すると、テーブルを削除し、再作成するときに、外部キー制約などのスキーマ定義の一部が失われる場合があります。

タスクを作成するときに、ソーススキーマと、ターゲットエンドポイントに移行する対応テーブルを指定するテーブルマッピングを作成できます。デフォルトのマッピング方法では、すべてのソーステーブルが、存在する場合は同じ名前のターゲットテーブルに移行されます。それ以外の場合は、ターゲット (タスク設定による) でソーステーブルが作成されます。さらに、特定のテーブルのみを移行するか、フィールドおよびテーブルのマッピングプロセスをさらに管理する場合は、カスタムマッピングを (JSON ファイルを使用して) 作成できます。また、ソースエンドポイントの 1 つのスキーマのみ、またはすべてのスキーマを移行する選択もできます。

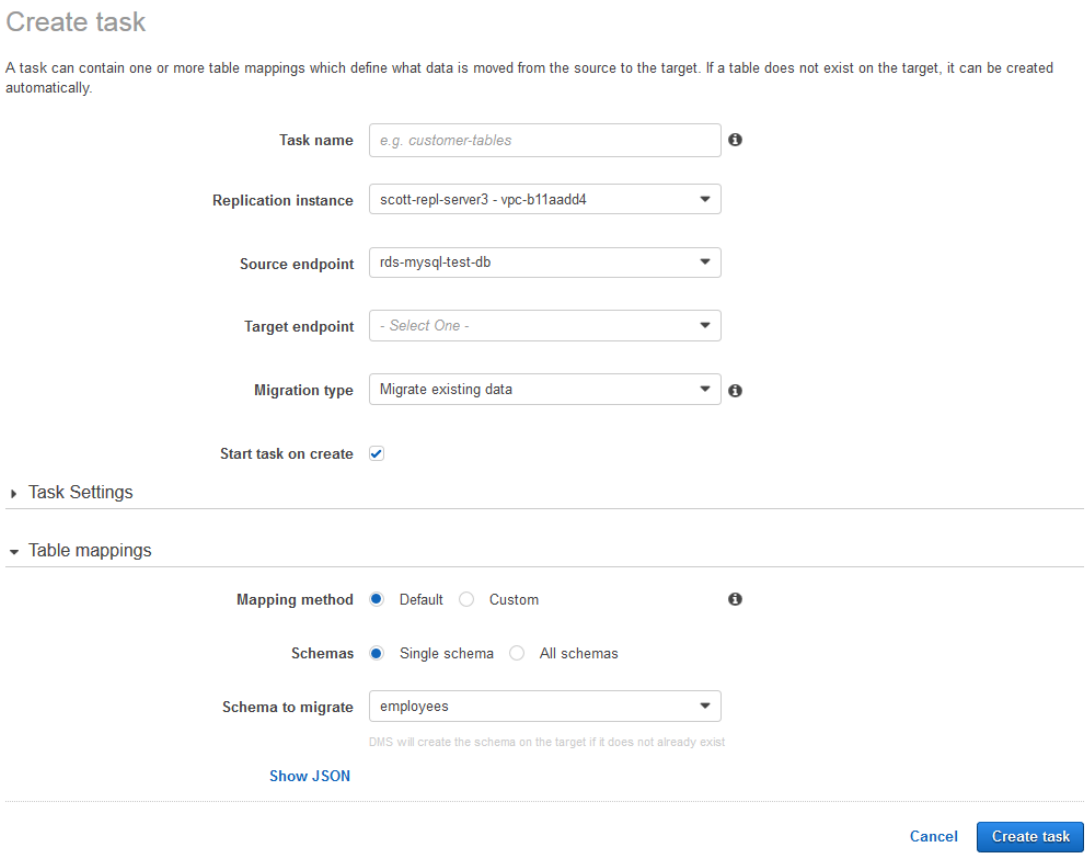

図 14: AWS DMS コンソールのタスクの作成ページ

AWS マネジメントコンソールを使用して、AWS DATABASE MIGRATION SERVICE(AWS DMS) タスクの進行状況をモニタリングできます。また、使用中のリソースとネットワーク接続をモニタリングすることもできます。AWS DMS コンソールには、次の図に示すように、タスクのステータス、完了率、経過時間、テーブルの統計を含めて、各タスクの基本的な統計が表示されます。

さらに、タスクを選択し、スループット、毎秒あたり移行されたレコード数、ディスクとメモリの使用量、レイテンシーを含む、そのタスクのパフォーマンスメトリックスを表示できます。

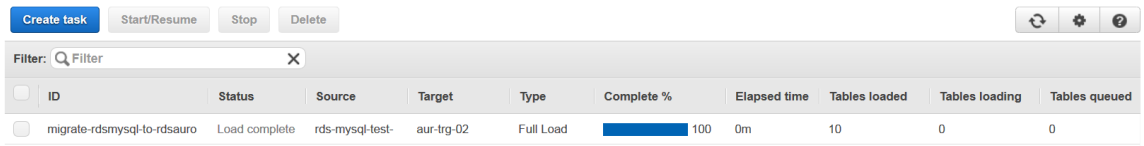

図 15: AWS DMS コンソールでのタスクのステータス

テストとカットオーバー

スキーマとデータがソースデータベースから Amazon Aurora に正常に移行されたら、移行プロセスのエンドツーエンドのテストを実行できます。テスト方法は、毎回のテスト移行の後に改善し、最終的な移行計画には、移行されたデータベースの適切なテストを確実にするテスト計画が含まれている必要があります。

移行テスト

テストカテゴリ	目的
基本的な受け入れのテスト	これらのカットオーバー前のテストは、データ移行プロセスの完了時に自動的に実行される必要があります。この主な目的は、データ移行が正常に行われたかどうか確認することです。以下に、これらのテストの一般的な出力をいくつか示します。 • 処理された項目の総数 • インポートされた項目の総数 • スキップされた項目の総数 • 警告の総数 • エラーの総数 テストで報告されたこれらのいずれかの総数が、予想される値を逸脱している場合、移行は成功せず、プロセスの次のステップまたは次のラウンドのテストに移る前に、問題を解決する必要があることを意味します。
機能テスト	これらのカットオーバー前のテストでは、データストレージ用に Aurora を使用してアプリケーションの機能を実行します。これには、自動および手動テストの組み合わせが含まれます。機能テストの主な目的は、Aurora へのデータの移行によって発生したアプリケーションの問題を識別することにあります。

テストカテゴリ	目的
機能以外のテスト	これらのカットオーバー後のテストでは、さまざまなレベルの負荷がかかった場合のパフォーマンスなど、アプリケーションの機能以外の特性を評価します。
ユーザー受け入れのテスト	これらのカットオーバー後のテストは、最終的なデータ移行とカットオーバーが完了した後で、アプリケーションのエンドユーザーによって実行される必要があります。これらのテストの目的は、アプリケーションが組織の主な機能を満たすために十分に役立つかどうか、エンドユーザーが判断することにあります。

カットオーバー

最終的な移行とテストを完了したら、アプリケーションで Amazon Aurora データベースを指します。移行のこの段階をカットオーバーと呼びます。計画およびテスト段階が正しく実行されていれば、カットオーバーで予期しない問題は発生しません。

カットオーバー前のアクション

- カットオーバーの時間帯の選択: ビジネスの中断を最小にして、新しいデータベースへのカットオーバーを達成できる時間帯を確認します。通常は、データベースのアクティビティが低い期間を選択します (通常は夜間、週末、またはその両方)。
- 変更が追いついていることの確認: ソースデータベースからターゲットデータベースへのデータベース変更のレプリケートに、ダウンタイムがゼロに近い移行方法が使用された場合は、すべてのデータベース変更が追いつき、ターゲットデータベースでソースデータベースから著しい遅延が発生していないことを確認します。

- アプリケーション設定を変更するスクリプトの準備: カットオーバーを達成するには、アプリケーションの設定ファイルでデータベース接続の詳細を変更する必要があります。大規模で複雑なアプリケーションでは、複数の場所で接続の詳細への更新が必要になる場合があります。接続設定を迅速かつ信頼性の高い方法で更新するために必要なスクリプトが準備できていることを確認します。
- アプリケーションの停止: ソースデータベースでアプリケーションプロセスを停止し、ソースデータベースを読み取り専用モードにして、それ以上の書き込みがソースデータベースに対して行われないようにします。ソースデータベースの変更がターゲットデータベースに完全に追いついていない場合は、それらの変更がターゲットデータベースに完全に伝播するまでしばらく待ちます。
- カットオーバー前のテストの実行: カットオーバー前のテストを実行して、データの移行が正常に行われたことを確認します。

カットオーバー

- カットオーバーの実行: カットオーバー前のチェックが正常に完了した場合は、アプリケーションで Amazon Aurora を指すことができます。カットオーバー前の段階で作成されたスクリプトを実行し、アプリケーション設定を変更して新しい Aurora データベースを指すようにします。
- アプリケーションの起動: この時点で、アプリケーションを起動することができます。アプリケーションの実行中にユーザーがアプリケーションにアクセスしないようにできる場合は、カットオーバー後のチェックの実行を終了するまで、そのオプションを使います。

カットオーバー後のチェック

- カットオーバー後のテストの実行: 事前定義された、自動または手動のテストケースを実行し、新しいデータベースに対してアプリケーションが予期どおり動作することを確認します。最初にデータベースの読み取り専用機能のテストを開始し、次にデータベースへのその書き込みテストを実行することをお勧めします。
- ユーザーアクセスの有効化と綿密なモニタリング: テストケースが正常に実行された場合、移行プロセスを完了するために、ユーザーにアプリケーションへのアクセスを許可できます。この時点で、アプリケーションとデータベースの両方を綿密にモニタリングする必要があります。

まとめ

Amazon Aurora は、クラウド用に構築された、パフォーマンスと可用性が高い、エンタープライズ級のデータベースです。Amazon Aurora を利用すると、他のオープンソースデータベースよりもパフォーマンスと可用性が高まり、ほとんどの商業グレードデータベースよりも費用が下がります。このホワイトペーパーでは、データベースを Amazon Aurora に移行するための最適な方法を確認するための手法を提案し、そうした移行を計画、実行するための手順の詳細を示しました。特に、AWS DATABASE MIGRATION SERVICE(AWS DMS) と AWS スキーマ変換ツールは、異機種の移行シナリオで推奨されるツールです。これらの強力なツールにより、データベース移行の費用と複雑さを大幅に減らすことができます。

寄稿者

本書の執筆に当たり、次の人物および組織が寄稿しました。

- Puneet Agarwal、ソリューションアーキテクト、アマゾン ウェブ サービス
- Scott Williams、ソリューションアーキテクト、アマゾン ウェブ サービス

詳細情報

追加のヘルプについては、次の資料を参照してください。

- Amazon Aurora 製品の詳細
- Amazon Aurora に関するよくある質問
- Amazon データベース移行サービス
- Amazon データベース移行サービスに関するよくある質問

注意

[1] https://aws.amazon.com/rds/aurora/

[2] http://aws.amazon.com/rds/aurora/pricing/

[3] https://do.awsstatic.com/product-marketing/Aurora/Aurora_Export_Import_Best_Practices_v1-3.pdf

[4] http://docs.aws.amazon.com/pt_br/AmazonRDS/latest/UserGuide/Aurora.Replication.html#Aurora.Overview.Replication.MySQLReplication

[5] http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Aurora.Migrate.html#USER_ImportAurora.PreImport

[6] http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_CreateSnapshot.html

[7] http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_CopySnapshot.html

[8] http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Aurora.Migrate.html#USER_ImportAuroraCluster.Console

[9] http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Aurora.Connect.html

[10] https://dev.mysql.com/doc/refman/5.6/en/mysqldump.html

[11] http://docs.aws.amazon.com/SchemaConversionTool/latest/userguide/Welcome.html

[12] http://docs.aws.amazon.com/SchemaConversionTool/latest/userguide/CHAP_SchemaConversionTool.GettingStarted.html

[13] http://docs.aws.amazon.com/SchemaConversionTool/latest/userguide/CHAP_SchemaConversionTool.Installing.html

[14] http://docs.aws.amazon.com/SchemaConversionTool/latest/userguide/CHAP_SchemaConversionTool.Installing.html#CHAP_SchemaConversionTool.Installing.JDBCDrivers

[15] https://forums.aws.amazon.com/forum.jspa?forumID=208

[16] http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Aurora.CreateInstance.html

[17] http://docs.aws.amazon.com/SchemaConversionTool/latest/userguide/CHAP_SchemaConversionTool.BestPractices.html

[18] http://docs.aws.amazon.com/dms/latest/userguide/CHAP_Source.html

[19] http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Aurora.CreateInstance.html