Amazon Aurora
へのデータベースの移行

2016 年 *6* 月

© 2016, Amazon Web Services, Inc. or its affiliates. All rights reserved.

注意

本文書は、情報提供の目的のみのために提供されるものです。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。お客様は本文書の情報および AWS 製品の使用について独自に評価する責任を負うものとします。これらの情報は、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」提供されるものです。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 4

Amazon Aurora について 4

データベース移行に関する考慮事項 6

移行フェーズ 6

アプリケーションに関する考慮事項 6

シャーディングとリードレプリカに関する考慮事項 8

信頼性に関する考慮事項 8

コストとライセンスに関する考慮事項 9

移行に関するその他の考慮事項 9

データベース移行プロセスの計画 10

同種間の移行 10

異種間の移行 12

大規模データベースの Amazon Aurora への移行 13

Amazon Aurora でのパーティションとシャードの統合 14

移行オプション一覧 15

RDS スナップショット移行 16

データベーススキーマの移行 21

同種間のスキーマ移行 21

異種間のスキーマ移行 23

AWS Schema Conversion Tool を使用したスキーマ移行 24

データの移行 32

AWS DMS の概要と一般的なアプローチ 32

移行メソッド 33

移行手順 34

テストとカットオーバー 40

移行テスト 40

カットオーバー 41
まとめ 43
寄稿者 43
その他の資料 43
注意 43

要約

Amazon Aurora は、MySQL との互換性が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克服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、既存データベースの Amazon Aurora への移行のベストプラクティスを説明することです。ここでは、移行時の考慮事項と、アプリケーションへの影響を最小限に抑えながらオープンソースおよび商用データベースを Amazon Aurora に移行するための段階的なプロセスを紹介します。

Amazon Aurora について

従来のリレーショナルデータベースは、数十年もの間、データストレージと永続性に対する主な選択肢とされてきました。これらのデータベースは、依然としてモノリシックなアーキテクチャに依存し、クラウドインフラストラクチャを活用するようには設計されていません。これらのモノリシックアーキテクチャは、特にコスト、柔軟性、および可用性などの点において多くの課題を提起します。これらの課題に対処するために、AWS は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 との互換性があるリレーショナルデータベースエンジンで、高性能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と、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兼ね備えています。Aurora は、MySQL と比較して最大 5 倍の優れたパフォーマンス、そして高性能商用データベースに相当するパフォーマンスを実現します。Amazon Aurora の価格は商用エンジンのコストの 10 分の 1 に設定されています。

Amazon Aurora は、Amazon Relational Database Service (Amazon RDS) プラットフォームを介して利用することができます。他の Amazon RDS データベース

と同様に、Aurora もフルマネージドデータベースサービスです。Amazon RDS プラットフォームにより、ハードウェアのプロビジョニング、ソフトウェアパッチの適用、セットアップ、設定、モニタリング、およびバックアップなどのデータベース管理タスクの多くは完全に自動化されています。

Amazon Aurora はミッションクリティカルなワークロード向けに構築されており、高い可用性をデフォルトで持ち合わせています。Aurora データベースクラスターは、リージョン内の複数のアベイラビリティーゾーンにまたがり、追加設定なしで物理的なデータセンター全体のデータに耐久性と耐障害性を提供します。アベイラビリティーゾーンは、Amazon が運営する 1 つまたは複数の高可用性データセンターで構成されています。アベイラビリティーゾーンは互いに隔離されており、低レイテンシーのリンク経由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全体で 6 回レプリケートされます。

Aurora クラスターボリュームは、データベースのデータ量の増加に伴って、パフォーマンスまたは可用性に影響することなく自動的に増えるため、大量のデータベースストレージを事前に予測したり、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のサイズは、最大 64 テラバイト (TB) まで増やすことができます。課金対象となるのは、Aurora クラスターボリュームで使用する容量のみです。

Aurora の自動バックアップ機能はデータのポイントインタイムリカバリをサポートするため、直近 5 分前まで、保存期間内の任意の瞬間にデータベースを復元させることができます。自動バックアップは、99.999999999% の耐久性を持つように設計されている Amazon Simple Storage Service (Amazon S3) に保存されます。Amazon Aurora バックアップは自動的かつ継続的な増分バックアップで、データベースのパフォーマンスに影響を与えません。

読み取り専用のレプリカを必要とするアプリケーションでは、極めて短いレプリカラグで Aurora データベースごとに最大 15 個の Aurora レプリカを作成することができます。これらのレプリカはソースインスタンスと同じ基礎ストレージを共有するので、コストが削減され、レプリカノードで書き込みを実行する必要もありません。

Amazon Aurora は安全性が高く、ユーザーが AWS Key Management Service (AWS KMS) を使って作成および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 暗号化を使用して実行されるデータベースインスタンスでは、基礎ストレージ内の保存データに加え、同じクラスター内

にある自動バックアップ、スナップショット、およびレプリカも暗号化されます。Amazon Aurora は移動中のデータのセキュア化に SSL (AES-256) を使用します。

Aurora の機能の完全なリストについては、Amazon Aurora 製品ページを参照してください。[1]豊富な機能セットとコストパフォーマンスにより、Amazon Aurora は、ミッションクリティカルなアプリケーションのための主力データベースとしてますます注目されています。

データベース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において重要なコンポーネントとされています。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ける重大なイベントで、アプリケーションの機能、パフォーマンス、および信頼性に影響を及ぼす場合があります。Amazon Aurora への最初の移行プロジェクトに着手する前に、いくつかの重要な事柄を考慮するようにしてください。

移行フェーズ

データベースの移行は複雑になりやすいため、段階的な反復アプローチが推奨されます。

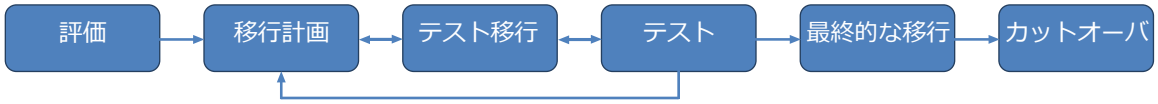


図 1: 移行フェーズ

アプリケーションに関する考慮事項

Aurora 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共に動作するよう設計できますが、お使いのアプリケーションが Amazon Aurora と動作することを確認するようにしてください。Amazon Aurora は MySQL 5.6 との接続互換性を持つよう設計されています。このため、現在 MySQL データベースと共に使用されているコード、アプリケーション、ドライバー、およびツールの多くは、変更なし、またはわずかな変更のみで Aurora と使用することができます。

ただし、MyISAM ストレージエンジンなどの特定の MySQL 機能を Amazon Aurora で使用することはできません。また、マネージド型という Aurora サービ

スの性質によってデータベースノードへの SSH アクセスが制限され、データベースホストへの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能力に影響を及ぼす場合があります。

パフォーマンスに関する考慮事項

データベースパフォーマンスは、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の主要考慮事項です。このため、成功したデータベース移行プロジェクトの多くが、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評価から始められています。Sysbench および TPC-C ベンチマークの実行で全体的なデータベースパフォーマンスをある程度把握することはできますが、これらのベンチマークはアプリケーションのデータアクセスパターンデータをエミュレートしません。より有益な結果を得るには、新しいプラットフォームでクエリ (またはクエリのサブセット) を直接実行することによって、時間依存型のワークロードに対するデータベースパフォーマンスをテストしてください。

以下の戦略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、ダウンタイムを伴う Amazon Aurora への移行を行い、アプリケーションのテストまたはステージングバージョンで、あるいは実稼働ワークロードを再現することによってパフォーマンステストを実行します。

- MySQL 非対応のエンジンの場合は、Amazon Aurora に最もビジーなテーブルを選択的にコピーし、これらのテーブルに対してクエリをテストします。このテストは良い出発点を提供しますが、当然ながら、完全なデータ移行後のテストでは、新しいプラットフォームでのアプリケーションの実稼働環境パフォーマンスの全容が提供されます。

Amazon Aurora は、商用エンジンに相当するパフォーマンスを実現し、MySQL パフォーマンスを大幅に上回ります。Aurora は、データベースエンジンと、データベースワークロード用に設計された SSD ベースの仮想ストレージレイヤーとを緊密に統合することによってこれを実現します。これによってストレージシステムへの書き込みが減り、ロック競合が最小限化されて、データベースのプロセススレッドによって生じる遅延が排除されます。r3.8xlarge インスタンスにおける SysBench を使ったテストでは、Amazon Aurora が 1 秒当たり 585,000 を超える読み取りと 107,000 書き込みを実現し、これは同じハードウェアで同じベンチマークを実行する MySQL よりも 5 倍の速さです。

Amazon Aurora が従来の MySQL を大幅に上回るひとつの分野は、同時並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には、多数の同時クエリを実行するようにアプリケーションを設計することが推奨されます。

シャーディングとリードレプリカに関する考慮事項

現在のデータベースが複数のノードにシャーディングされている場合、移行時にこれらのシャードを単一の Aurora データベースにまとめる機会があるかもしれません。単一の Amazon Aurora インスタンスは最大 64 TB まで拡張でき、何千ものテーブルをサポートすると共に、標準の MySQL データベースよりもはるかに多い読み取りおよび書き込み数をサポートします。

読み取り/書き込み量が多いアプリケーションの場合は、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ために Aurora リードレプリカを使用することを検討してください。これにより、書き込みに対するプライマリデータベースの同時並行性を向上させることができ、全体的な読み取りおよび書き込みパフォーマンスが改善されます。リードレプリカの使用は、データベースクラスターにフェイルオーバー機能を追加すると同時に、プライマリインスタンス用により小さいインスタンスを使用できる可能性があるため、マルチ AZ 構成のコストを削減することもできます。Aurora リードレプリカは、ほぼゼロのレプリケーション遅延を実現し、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

データベースの重要な考慮事項は、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) および RPO (目標復旧ポイント) 要件を判断してください。Amazon Aurora では、これらの要因の両方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 秒未満に短縮します。また、Aurora はバッファキャッシュをデータベースから移動させて、再起動時にすぐ利用できるようにします。ハードウェアと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、復旧はデータベースプラットフォームによって自動的に処理されます。

Aurora は、AWS リージョン内でゼロ RPO の復旧を提供するように設計されており、これはオンプレミスデータベースシステムからの大きな改善です。Aurora は、3 つのアベイラビリティーゾーン全体にデータのコピーを 6 個維持

し、データを失うことなく、正常な AZ でのデータベースの復旧を自動的に試行します。万一、Amazon Aurora ストレージ内でデータが利用できなくなった場合は、DB スナップショットから復元する、または新しいインスタンスに対してポイントインタイム復元操作を行うことができます。

コストと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コストが伴います。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) を分析することが必要不可欠です。新しいデータベースプラットフォームへの移行は、アプリケーションに同様の機能、またはより優れた機能を提供しながら総所有コストを削減することが理想的です。オープンソースデータベースエンジン (MySQL、Postgres) 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するものです。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は、コストの大部分がデータベースのライセンス取得関連となります。

Aurora が商用エンジンの 10 分の 1 のコストで利用できるため、Aurora に移行される多くのアプリケーションは TCO を大幅に削減することができます。MySQL または Postgres といったオープンソースエンジンで実行している場合でも、Aurora の高パフォーマンスと 2 つの目的を兼ねたリードレプリカで、Amazon Aurora への移行によって有意義なコスト削減を実現できます。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 および信頼性要因を考慮した後は、新しいプラットフォームへの移行には何が必要になるかを検討するようにしてください。

コード変更工数の見積もり

データベースの Amazon Aurora への移行中に実行する必要があるコードおよびスキーマ変更の量を見積もることが大切です。MySQL との互換性があるデータベースからの移行時には、コード変更はほとんど必要ありませんが、MySQL 以外のエンジンからの移行時には、スキーマおよびコードの変更が必要となる場合があります。本ホワイトペーパーの後半で説明する AWS Schema Conversion Tool は、その工数を見積もるために役立ちます。

移行中におけるアプリケーションの可用性

Amazon Aurora への移行には、アプリケーションでの予測可能なダウンタイムを伴うアプローチ、またはダウンタイムがゼロに近いアプローチを取るオプションがあります。選択するアプローチは、データベースのサイズと、アプリケーションの可用性要件に応じて異なります。いずれの場合も、データベース移行を開始する前に、アプリケーションとビジネスに対する移行プロセスの影響を考慮することが推奨されます。次の数セクションでは、両方のアプローチ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 に移行する間に考慮する主な事柄についていくつか説明しました。Aurora がお使いのアプリケーションに適切であると判断した後のステップは、暫定的な移行アプローチを決定し、データベース移行計画を作成することです。

同種間の移行

ソースデータベースが MySQL 5.6 対応のデータベース (MySQL、MariaDB、Percona など) である場合、Aurora への移行は非常に簡潔です。

ダウンタイムを伴う同種間の移行

アプリケーションが、オフピーク時間内において予測可能な長さのダウンタイムに対応できる場合、ダウンタイムを伴う移行が最もシンプルなオプションであり、強く推奨されるアプローチです。大抵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ウィンドウがすでに存在するため、ほとんどのデータベース移行プロジェクトがこのカテゴリに当てはまります。ダウンタイムを伴った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 に移行することができます。ダウンタイムを伴う移行では、スナップショットおよび移行の進行中にアプリケーションを停止する、またはデータベースへの書き込みを停止する必要があります。移行に要する時間は主にデータベースのサイズに依存し、テスト移行を実行することによって、本番移行前に判断することができます。スナップショット移行オプションは、「RDS スナップショット移行」セクションで説明されています。スナップショット移行とバイナリログ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可能です。これは次のセクションで説明されています。

- **ネイティブ MySQL ツールを使用した移行。**Aurora へのデータとスキーマの移行には、ネイティブ MySQL ツールを使用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細かく制御する必要がある、ネイティブ MySQL ツールの使用に慣れている、および他の移行方法ではユースケースに対して良い成果が出せない場合に適したオプションです。このオプションを使用する時のベストプラクティスについては、ホワイトペーパー Amazon RDS for Aurora Export/Import Performance Best Practices 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。**AWS DMS を使用した 1 回限りの移行は、ソースデータベースを Amazon Aurora に移動させるためのもう一つのツールです。データ移動のために AWS DMS を使用する前に、ネイティブ MySQL ツールを使ってデータベーススキーマをソースからターゲットにコピーする必要があります。段階的なプロセスについては、「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最適なオプションです。

ダウンタイムがゼロに近い同種間の移行

シナリオによっては、最小限のダウンタイムでデータベースを Aurora に移行したい場合があります。これらはその 2 つの例です。

- データベースが比較的大きく、ダウンタイムを伴う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ウィンドウよりも長い場合
- テスト目的のためにソースデータベースとターゲットデータベースを並行して実行したい場合

このような場合は、レプリケーションを使用して、変更をソース MySQL データベースから Aurora にリアルタイムでレプリケートすることができます。これには、選択できるいくつかのオプションがあります。

- **MySQL バイナリログ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 バイナリログレプリケーションをサポートしています。MySQL データベースを実行しているユーザーは、典型的なバイナリログレプリケーションのセットアップについての詳しい知識を既に持っていると思われます。知識があり、移行プロセスをより詳細に制御したい場合は、バイナリログレプリケーションと共にネイ

ティブツールを使用した一回限りのデータベースのロードにより、Aurora への移行パスが使い慣れた形で提供されます。

- **AWS Database Migration Service (AWS DMS) を使用したダウンタイムがゼロに近い移行。**1 回限りの移行のサポートに加え、AWS DMS は、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でのデータレプリケーションもサポートしています。AWS DMS は、初回データコピー、レプリケーションの設定、およびレプリケーションの監視に関連する複雑さを解決します。初回データベース移行が完了した後も、ターゲットデータベースでは、ユーザーがソースとの同期化を選択する限り、同期状態が維持されます。バイナリログレプリケーションについて詳しい知識がない場合、Amazon Aurora への同種間のダウンタイムがゼロに近い移行には、AWS DMS が次の最適オプションとなります。「AWS DMS の概要と一般的なアプローチ」セクションを参照してください。
- **RDS スナップショットと MySQL バイナリログ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、そのデータベースのスナップショットを Amazon Aurora に移行して、スナップショット移行後にソースとターゲットの間のバイナリログ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 とのレプリケーション」を参照してください 。[4]

異種間の移行

MySQL 非対応のデータベース (Oracle、SQL Server、PostgresSQL など) の Amazon Aurora への移行を行う予定の場合、この移行を迅速かつ容易に完了するために役立つオプションがいつくかあります。

スキーマの移行

MySQL 非対応のデータベースから Amazon Aurora へのスキーマの移行は、AWS Schema Conversion Tool を使用して実現できます。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。AWS Schema Conversion Tool は、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に、ターゲットの Amazon RDS データベースに同等のスキーマを作成する方法についてのガイダンスを提供します。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、ダウンタイムがゼロに近い同種間移行をサポートすると同時に、異種データベース間の継続的なレプリケーションもサポートし、ソースデータベースのターゲットデータベースへの移行において、ダウンタイムを伴う移行とダウンタイムがゼロに近い移行の両方に対する推奨オプションです。移行が開始されると、AWS DMS は、移行プロセス中に生じるソースデータベースへのデータ変更がターゲットに自動的にレプリケートされることを確実にしながら、データ型の変換、圧縮、および並行転送 (より迅速なデータ転送のため) などの移行プロセスの複雑性のすべてに対応します。

AWS DMS 以外にも、Attunity Replicate、Tungsten Replicator、Oracle Golden Gate などの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最終的に決定する前に、パフォーマンスとライセンス取得のコストを考慮してください。

大規模データベースの Amazon Aurora への移行

大規模なデータセットの移行は、すべてのデータベース移行プロジェクトにおいて特有の課題をもたらします。成功した大規模データベース移行プロジェクトの多くでは、以下の戦略を組み合わせて使用しています。

- **継続的なレプリケーションでの移行。**大規模データベースには、通常、データのソースからターゲットへの移動中に長時間のダウンタイム要件があります。ダウンタイムを短縮するには、まず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、次にレプリケーションを有効化 (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) して変更を追いつかせます。
- **最初に静的テーブルをコピーする。**データベースが参照データを持つ大規模な静的テーブルに依存している場合、アクティブデータセットを移行する前に、これらの大規模テーブルをターゲットデータベースに移行することができます。テーブルを AWS DMS を活用して選択的にコピーする、またはこれらのテーブルを手動でエクスポートまたはインポートすることができます。
- **複数フェーズでの移行。**何千ものテーブルを持つ大規模なデータベースの移行は、複数のフェーズに分割することができます。たとえば、ソースデータベースがターゲットデータベースに完全に移行されるまで、週末ごとにクロス結合クエリなしでテーブルのセットを移行させることができます。これを実現させるため、データセットが 2 つの異なるノードにあるときに

は、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ることに注意してください。これは一般的な移行パターンではありませんが、オプションのひとつです。

- **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とテーブルが含まれています。多くの場合、開発者とデータベース管理者は、同じデータベース内にテーブルのバックアップコピーを保持しているか、単に使用されていないテーブルを削除し忘れています。いずれにしても、データベース移行プロジェクトは、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を提供します。使用されていないテーブルがある場合は、それらを削除するか、別のデータベースにアーカイブします。大きなテーブルから古いデータを削除する、またはそのデータをフラットファイルにアーカイブすることもできます。

Amazon Aurora でのパーティションとシャードの統合

高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は、単一の Aurora データベースでこれらのパーティションまたはシャードを統合することができます。単一の Amazon Aurora インスタンスは最大 64 TB まで拡張でき、何千ものテーブルをサポートすると共に、標準の MySQL データベースよりもはるかに多い読み取りおよび書き込み数をサポートします。単一の Aurora インスタンスでこれらのパーティションを統合することは、総所有コストを削減してデータベース管理を簡素化するだけでなく、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改善されます。

- **機能パーティション。**機能パーティショニングとは、異なるノードを異なるタスクの専用ノードにすることです。たとえば、e コマースアプリケーションでは、ひとつのデータベースノードを製品カタログデータ用に、別のデータベースノードを注文の保存と処理用にしている場合があります。その結果、これらのパーティションには、通常独特な重複しないスキーマがあります。

  統合戦略。各機能パーティションを個別のスキーマとしてターゲット Aurora インスタンスに移行します。ソースデータベースが MySQL 対応である場合、ネイティブ MySQL ツールを使用してスキーマを移行し、次に AWS DMS を使用してデータの一回限りの移行、またはレプリケーションでの継続的な移行を行います。ソースデータベースが MySQL 非対応である場合、AWS Schema Conversion Tool を使用してスキーマを Aurora に移

行し、一回限りのロードまたは継続的なレプリケーションのために AWS DMS を使用します。

- **データシャード。**複数のノード間に個別のデータセットを持つ同一のスキーマがある場合、データベースシャーディングを活用していることになります。たとえば、トラフィックが多い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、ユーザーアクティビティとデータを複数のデータベースシャードにシャーディングする場合があります。

  統合戦略。シャードはすべて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は一度作成すればよいだけです。MySQL 対応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 以外のデータベースを使用している場合は、AWS Schema Conversion Tool 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の移行が完了したら、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 回限りのデータのロードを使用して、個々のシャードを Aurora に移行することが最善です。アプリケーションへの書き込みを長時間停止できない場合でも、レプリケーションで AWS DMS を使用できますが、これは適切な計画とテストを実施した後でのみ使用してください。

移行オプション一覧

ソースデータベースタイプ	ダウンタイムを伴う移行	ダウンタイムがゼロに近い移行
Amazon RDS MySQL	オプション 1: RDS スナップショット移行	オプション 1: ネイティブツール + バイナリログレプリケーションを使用した移行
	オプション 2: ネイティブツールを使用した手動での移行*	オプション 2: RDS スナップショット移行 + バイナリログレプリケーション
	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と AWS DMS を使用したデータのロード	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 でのデータ移動
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 1: ネイティブツール + バイナリログレプリケーションを使用した移行
	オプション 2: ネイティブツールを	オプション 2: ネイティブツールを使

ソースデータベースタイプ	ダウンタイムを伴う移行	ダウンタイムがゼロに近い移行
	使用したスキーマの移行 + AWS DMS でのデータロード	用したスキーマの移行 + AWS DMS でのデータ移動
Oracle/SQL サーバー	オプション 1: AWS Schema Conversion Tool + AWS DMS (推奨) オプション 2: 手動またはサードパーティー製ツールでのスキーマ変換 + ターゲットでの手動またはサードパーティーのデータロード	オプション 1: AWS Schema Conversion Tool + AWS DMS (推奨) オプション 2: 手動またはサードパーティー製ツールでのスキーマ変換 + ターゲットでの手動またはサードパーティーのデータロード + サードパーティー製ツールでのレプリケーション
その他 MySQL 以外のデータベース	オプション: 手動またはサードパーティー製ツールでのスキーマ変換 + ターゲットでの手動またはサードパーティーのデータロード	オプション: 手動またはサードパーティー製ツールでのスキーマ変換 + ターゲットでの手動またはサードパーティーのデータロード + サードパーティー製ツールでのレプリケーション (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 スナップショット移行

Aurora への移動に RDS スナップショット移行を使用するには、MySQL データベースが Amazon RDS MySQL 5.6 で実行されていなくてはならず、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。この移行メソッドは、オンプレミスのデータベースまたは Amazon Elastic Compute Cloud (Amazon EC2) で実行されるデータベースでは機能しません。また、5.6 より前のバージョンで 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。

この移行メソッドの最大の利点は、最もシンプルで、必要なステップの数が少ないことです。特に、このメソッドは、すべてのデータベースデータと共に、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もすべて移行します。

バイナリログレプリケーションなしでのスナップショット移行中、ソースデータベースはオフラインまたは読み取り専用モードにしておく必要があります (これは、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す)。ダウンタイムの見積りは、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実施するだけで行えます。移行時間がダウンタイム要件と一致する場合は、これが最適のメソッドとなり得ます。場合によっては、AWS DMS またはネイティブ移行ツールを使用した移行がスナップショット移行よりも早い可能性があることに留意してください。

長時間のダウンタイムを許容できない場合でも、まず、ソースデータベースが引き続きアップデートされることを可能にしたままで RDS データベースのスナップショットを Amazon Aurora に移行することにより、ダウンタイムがゼロに近い移行を実現することができます。その後、MySQL から Aurora への MySQL バイナリログレプリケーションを使用してデータベースを最新状態にします。

DB スナップショットは、手動または自動化されたもののどちらでも移行することができます。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判断します。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 インスタンスがあるリージョンにスナップショットを作成します。
3. コンソールの [**Migrate Database**] 機能を使用し、MySQL 5.6 の元の DB インスタンスからの DB スナップショットを使用してデータを投入する Amazon Aurora DB クラスターを作成します。

注意: MyISAM テーブルには変換時にエラーが発生するものもあり、手動での変更が必要と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を複合キーの一部にすることが許可されません。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ための容量要件の見積り

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、スナップショットを移行する前に Amazon Elastic Block Store (Amazon EBS) ボリュームを使用してスナップショットからのデータをフ

ォーマットします。移行のためのデータのフォーマット用に追加容量が必要となる場合があります。移行中に容量問題を起こす可能性がある 2 つの機能は、MyISAM テーブル、および `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機能のいずれかを使用していなければ容量問題は起こらないため、このセクションは省略できます。移行中、MyISAM テーブルが InnoDB に変換され、圧縮されたテーブルがすべて展開されます。このため、このようなテーブルの追加コピーのために十分な容量が必要となります。

移行ボリュームのサイズは、スナップショットの作成元であるソース MySQL データベースの割当済サイズに基づきます。したがって、データベースの全体的なサイズのごく一部を占める MyISAM テーブルまたは圧縮テーブルがあり、元のデータベースに空き容量がある場合は、容量問題を生じることなく移行が成功するはずです。ただし、変換された MyISAM テーブルのコピー、および圧縮テーブルの別の (展開済み) コピーを保存するために十分な容量が元のデータベースにない場合、移行ボリュームの大きさも十分ではなくなります。このような場合は、これらのテーブルの追加コピーのための容量を空けるためにソース 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、データベースの新しいスナップショットを作成してから、この新しいスナップショットを移行する必要があります。

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に従ってください。

- 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を ト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、このサイズは最大 6 TB に制限されます。
- ソースデータベース内の MyISAM 以外のテーブルのサイズは、最大 6 TB にすることができます。ただし、変換中における追加容量要件のため、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でサイズが 3 TB を超えるものがないことを確認してください。

Amazon Aurora にデータベーススキーマを移行する前に、これを変更 (MyISAM テーブルを InnoDB に変換して `ROW_FORMAT=COMPRESSED` を削除) してもよいでしょう。これは、次のような場合に役立ちます。

- 移行プロセスを迅速化したい。
- どれだけの容量をプロビジョニングする必要があるかよくわからない。

- データを移行しようとしたが、プロビジョニングされた容量の不足が原因で移行に失敗した。

これらの変更は、実稼働 Amazon RDS MySQL データベースではなく、本番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で行うようにしてください。これを実行することに関する詳細については、*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 インスタンスからのデータが投入されます。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ており、暗号化されていない必要があります。DB スナップショットの作成に関する情報については、*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 クラスターを設置したいリージョンにない場合は、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情報については、*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、次の手順を実行します。

1. https://console.aws.amazon.com/rds/ で AWS マネジメントコンソールにサインインして Amazon RDS コンソールを開きます。
2. [**Snapshots**] を選択します。
3. [**Snapshots**] ページで、Aurora DB クラスターに移行させたい 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、以下の図にあるように、お使いの環境に一致する値と処理要件を指定します。これらのオプションの説明については、*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]

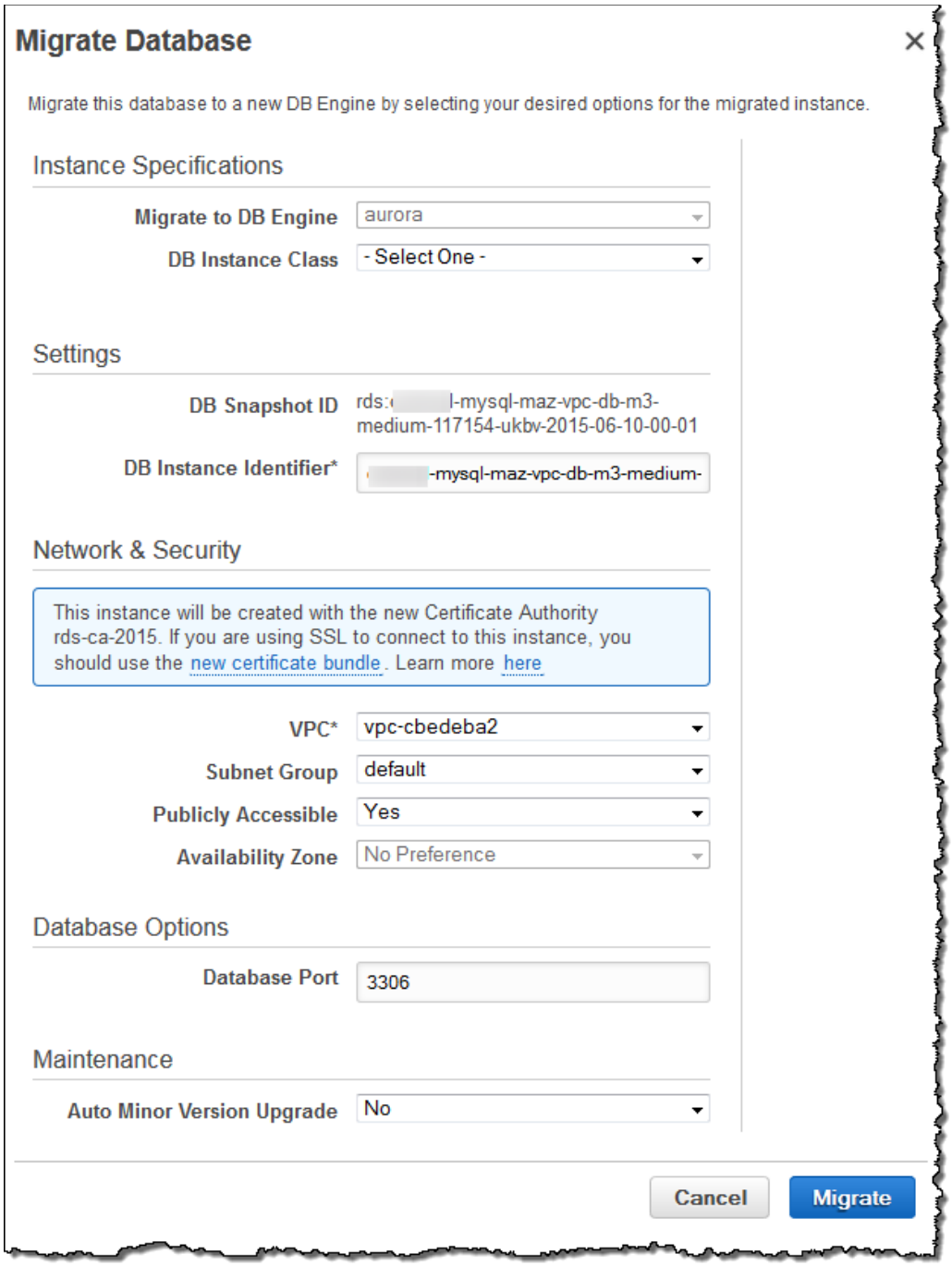


図 **2:** スナップショット移行

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

インスタンスのリストで、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、移行の進行状況を監視します。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たクラスターのエンドポイントが表示されます。Amazon Aurora DB クラスターへの接続に関する詳細につい

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 スナップショット移行は、完全なスキーマとデータの両方を新しい Aurora インスタンスに移行させます。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アップタイム要件のために RDS スナップショットを使用できない場合は、実際のデータを移動させる前に、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、通常は以下のオブジェクトが含まれています。

- 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、ユーザー定義タイプ、およびデータ型
- データベースコードオブジェクト: 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、public 型、private 型、カーソル、例外、パラメータ、およびその他オブジェクト

多くの場合、データベーススキーマは比較的静的な状態のままであるため、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によってデータベーススキーマが頻繁に変更される場合は、移行処理中にこれらの変更が中断する、またはスキーマ移行プロセス中にこれらの変更を確認するようにしてください。

ソースデータベースのタイプに応じて、次のセクションで説明する手法を使用してデータベーススキーマを移行することができます。スキーマ移行の前提条件として、ターゲット Aurora データベースが作成され、利用可能である必要があります。

同種間のスキーマ移行

ソースデータベースが MySQL 5.6 対応で、Amazon RDS、Amazon EC2 で実行されている、または AWS 外で実行されている場合、ネイティブ MySQL ツールを使用してスキーマのエクスポートとインポートを行うことができます。

- **データベーススキーマのエクスポート。**データベーススキーマは、mysqldump クライアントユーティリティを使用してエクスポートすることができます。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` オプションは、実際のテーブルデータを一切エクスポートせずに、データベーススキーマのみがエクスポートされることを確実にします。完全な `mysqldump` コマンド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
```

- **Aurora へのデータベーススキーマのインポート。**Aurora インスタンスにスキーマを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 に向けます。

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

次の点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ている場合、ダンプファイルから `DEFINER` 構文を削除する必要があります。これを実行するためのシンプルな Perl コマンドを以下に示します。これを行うことにより、現在接続されているユーザーを `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。これによって生じる可能性があるセキュリティへの影響を評価するようにしてください。

```
$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内に MyISAM テーブルがある場合、Aurora は `CREATE TABLE` コマンドの実行時にエンジンを InnoDB に自動変更します。
- Amazon Aurora は圧縮テーブル (つまり、`ROW_FORMAT=COMPRESSED` で作製されたテーブル) をサポートしません。ソースデータベース内に圧縮されたテーブルがある場合、Aurora は `CREATE TABLE` コマンドの実行時に `ROW_FORMAT` を `COMPACT` に自動変更します。

MySQL 5.6 対応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、次のステップはソースからターゲットへの実際のデータのコピーです。詳細については、本ホワイトペーパー後半の「AWS DMS の概要と一般的なアプローチ」を参照してください。

異種間のスキーマ移行

ソースデータベースに MySQL との互換性がない場合は、スキーマを Amazon Aurora との互換性があるフォーマットに変換する必要があります。ひと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変換は容易なタスクではなく、データベースおよびアプリケーションコードの特定箇所の書き換えが関与する場合があります。スキーマの変換と Amazon Aurora への移行には、2 つの主なオプションがあります。

- **AWS Schema Conversion Tool。**AWS Schema Conversion Tool は、ソースデータベーススキーマと、ビュー、ストアドプロシージャ、および関数を含むカスタムコードの大部分をターゲットデータベースと互換性があるフォーマットに自動変換することによって、異種データベース間の移行を容易にします。[11]自動変換できないコードには明確な印が付けられるため、これらを手動で変換することができます。このツールを使用して、Oracle または Microsoft SQL Server のいずれか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 または Amazon EC2 のいずれかにある PostgreSQL ターゲットデータベースに変換できます。Oracle、SQL Server、または PostgreSQL スキーマを AWS Schema Conversion Tool を使用して Aurora との互換性があるフォーマットに変換する方法が推奨されるメソッドです。
- **手動のスキーマ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 ではない場合は、ソースデータベーススキーマを手動で Aurora に移行する、またはサードパーティー製ツールを使用して MySQL 5.6 との互換性があるフォーマットにスキーマを移行することができます。手動のスキーマ移行は、ソーススキーマのサイズと複雑性に応じてかなり複雑なプロセスになり得ます。しかし、ほと

んどの場合、手動のスキーマ変換は、Amazon Aurora に付随するコスト削減、パフォーマンスメリット、およびその他の機能改善を考えると、実行する価値があります。

AWS Schema Conversion Tool を使用したスキーマ移行

AWS Schema Conversion Tool は、ソースデータベースのデータベーススキーマを Amazon Aurora との互換性があるフォーマットに自動変換するために、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。データベース移行工数を評価するため、および実際の本番移行前のパイロット移行のために AWS Schema Conversion Tool を使用することが強く推奨されます。

次の説明は、AWS Schema Conversion Tool を使用するための高度な手順の段階的な説明です。詳細手順については、*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. まず、ツールをインストールします。AWS Schema Conversion Tool は Microsoft Windows、Mac OS X、Ubuntu Linux、および Fedora Linux で利用可能です。

   詳細なダウンロードおよびインストール手順は、AWS Schema Conversion Tool User Guide の「installation and update」 に記載されています。[13]AWS Schema Conversion Tool 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ために、ソースデータベースとターゲットデータベースの両方に直接接続する必要があります。AWS Schema Conversion Tool をインストールしたデスクトップに、ソースデータベースとターゲットデータベースとのネットワーク接続があることを確認してください。

2. JDBC ドライバーをインストールします。AWS Schema Conversion Tool は、JDBC ドライバー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 ドライバー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。ドライバーをダウンロードする手順は、*AWS Schema Conversion Tool User Guide* の「Required Database Drivers」に記載されています。[14]また、「AWS forum for AWS Schema Conversion Tool」で異なるデータベースエンジンのための JDBC ドライバーのセットアップ手順を確認してください。[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 Schema Conversion Tool を開き、[**New Project Wizard**] を起動します。

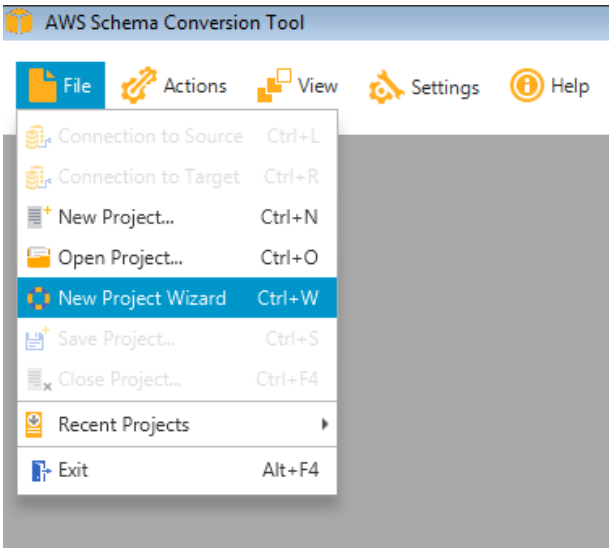


図 3: 新しい AWS Schema Conversion Tool プロジェクトの作成

5. ソースデータベースを設定して、AWS Schema Conversion Tool とソースデータベース間の接続をテストします。これを機能させるには、デスクトップからソースデータベースへのアクセスが可能である必要があるため、適切なネットワークとファイアウォール設定が整っていることを確認するようにしてください。

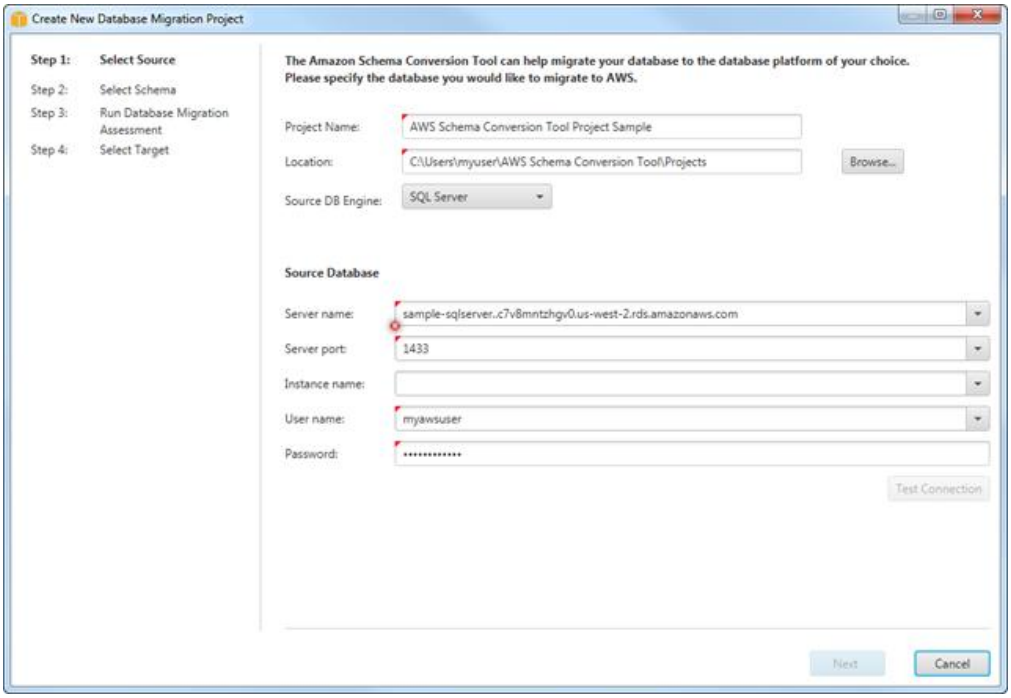


図 **4:** 新しいデータ移行プロジェクトの作成ウィザード

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。

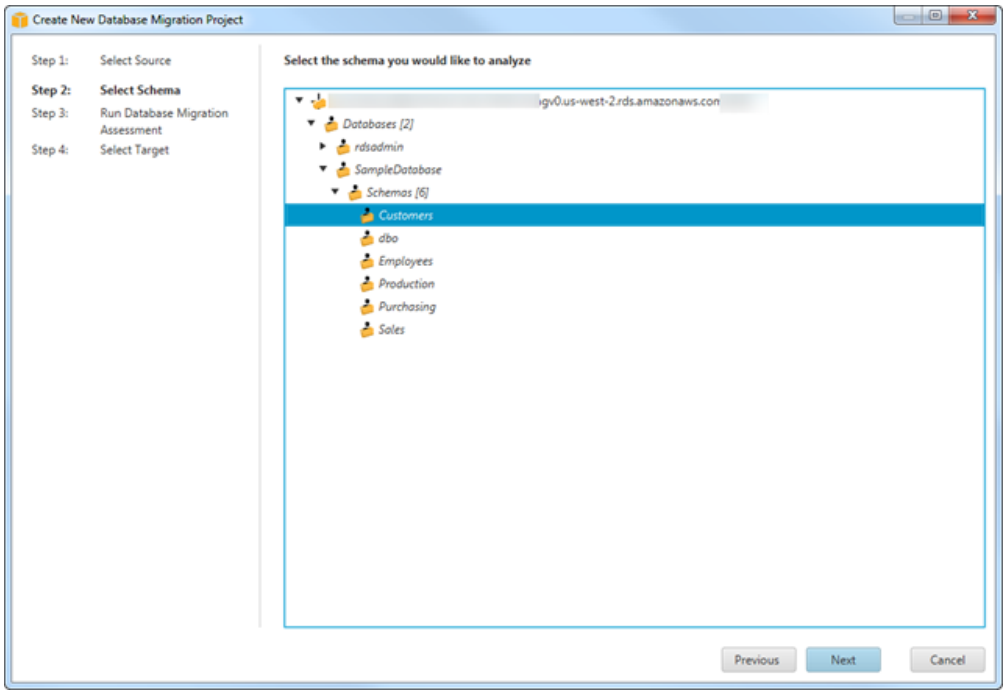


図 **5:** 移行ウィザードのスキーマの選択ステップ

7. データベース移行評価レポートを実行します。このレポートは、ソースデータベースからターゲット 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を提供します。すべてのスキーマ変換タスクが要約され、Aurora に自動変換できないスキーマ部分のためのアクションアイテムが詳しく説明されています。レポートには、自動変換できなかったものと同等のコードをターゲットデータベースに書き込むためにかかる工数の見積もりも含まれています。

   [**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。このレポートは、後で再度表示することができます。

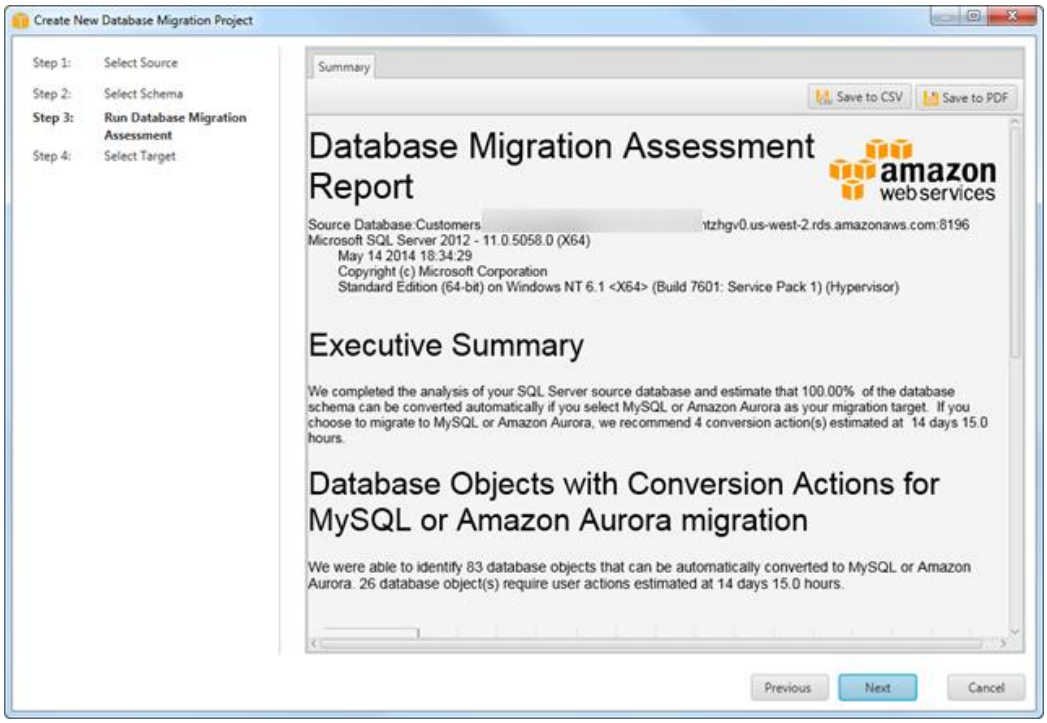


図 **6:** 移行レポート

8. ターゲット Amazon Aurora データベースを設定して、AWS Schema Conversion Tool とソースデータベース間の接続をテストします。これを機能させるには、デスクトップからターゲットデータベースへのアクセスが可能である必要があるため、適切なネットワークとファイアウォール設定が整っていることを確認するようにしてください。[**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が表示されたら、ソースおよびターゲットデータベースへの接続がすでに確立されており、詳細アセスメントレポートを評価する準備が整います。

10. ソースデータベースからのスキーマが表示される左パネルで、アセスメントレポートの対象とな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**] をクリックします。

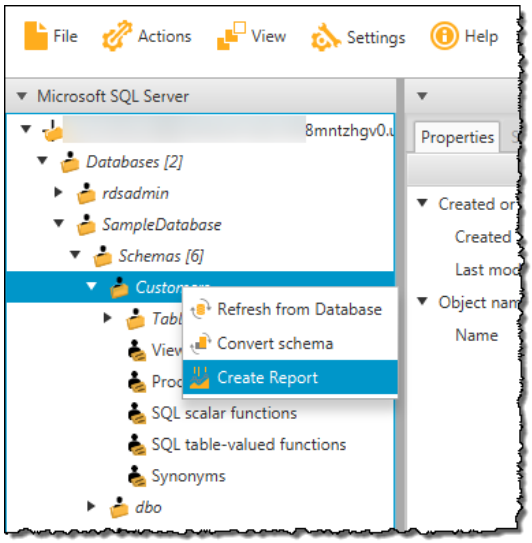
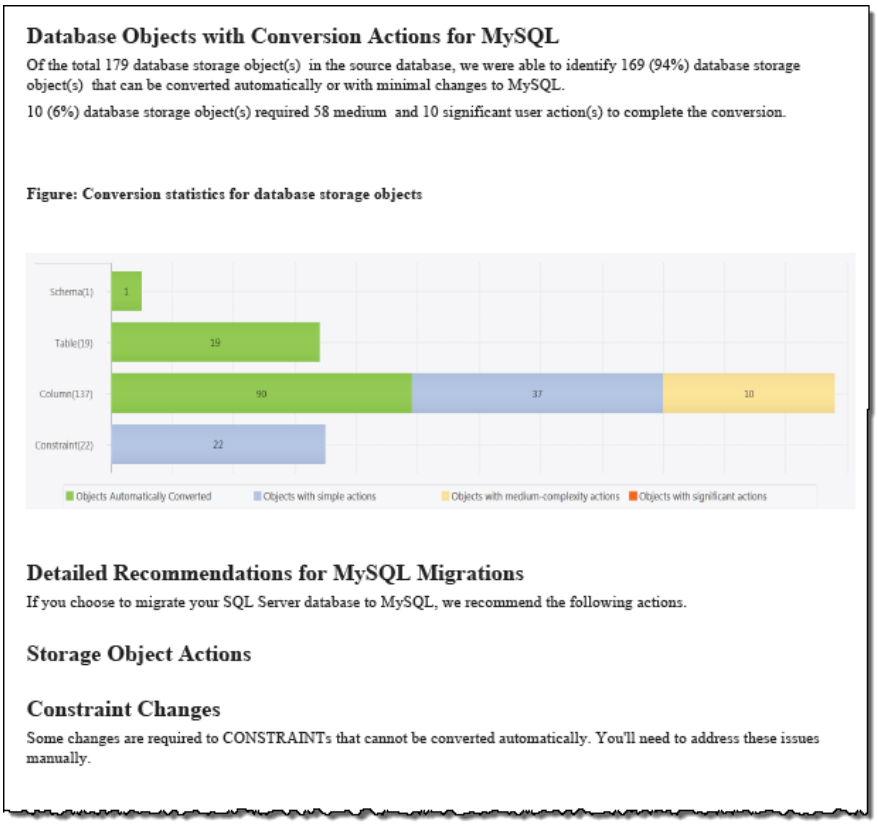


図 7: 移行レポートの作成

[**Summary**] タブには、データベース移行アセスメントレポートからの要約情報が表示されます。これには、自動変換されたアイテムと、自動変換できなかったアイテムが示されています。

ターゲットデータベースエンジンに自動変換できなかったスキーマアイテムについては、要約情報に、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かかる工数の見積りが含まれています。このレポートは、これらのスキーマアイテムを変換する推定時間を次のように分類します。

- **Simple –** 1 時間未満で完了できるアクション。
- **Medium –** より複雑で、1～4 時間で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。



Database Objects with Conversion Actions for MySQL

Of the total 179 database storage object(s) in the source database, we were able to identify 169 (94%) database storage object(s) that can be converted automatically or with minimal changes to MySQL.

10 (6%) database storage object(s) required 58 medium and 10 significant user action(s) to complete the conversion.

Figure: Conversion statistics for database storage objects

Detailed Recommendations for MySQL Migrations

If you choose to migrate your SQL Server database to MySQL, we recommend the following actions.

Storage Object Actions

Constraint Changes

Some changes are required to CONSTRAINTs that cannot be converted automatically. You'll need to address these issues manually.

図 **8:** 移行レポート

重要: データベース移行プロジェクトに必要な工数を評価している場合、このアセスメントレポートは、検討すべき重要なアーティファクトです。データスキーマで必要なコード変更、およびその変更がアプリケーションの機能性と設計に及ぼす可能性がある影響を判断するために、アセスメントレポートを細部にわたって検討してください。

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るわけではありません。その代わり、変換されたスキーマは、ターゲットデータベースに明示的に適用され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のスキーマを変換するには、プロジェクトの左パネルから変換するスキーマオブジェクトを選択します。以下の図に示されるように、オブジェクトを右クリックして [**Convert schema**] を選択します。

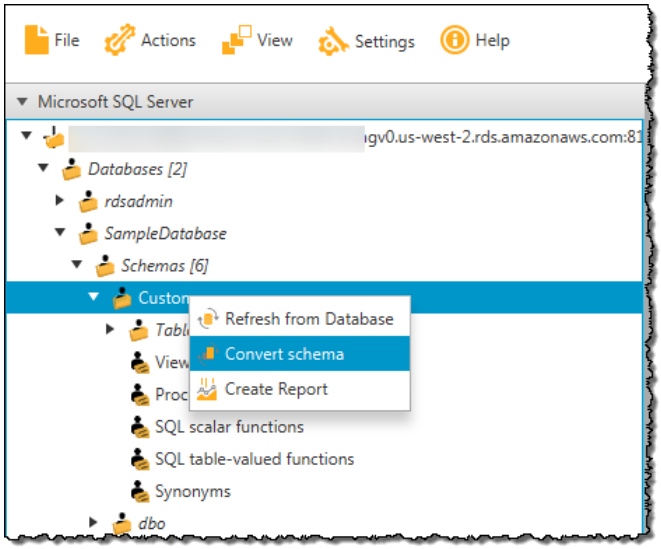


図 9: スキーマの変換

このアクションは、変換されたスキーマをプロジェクトウィンドウの右パネルに追加し、AWS Schema Conversion Tool によって自動変換されたオブジェクトを表示します。

12. アセスメントレポートにあるアクションアイテムには異なる方法で対応することができます。

- 同等のスキーマを手動で追加します。プロジェクトの右パネルにある [**Apply to database**] を選択することによって、ターゲット DB インスタンスに自動変換できるスキーマの一部を記述することができます。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、自動変換できなかったアイテムは含まれません。これらのアイテムは、データベース移行アセスメントレポートにリストされます。

  ターゲット DB インスタンスにスキーマを適用したら、自動変換できなかったアイテムのために、スキーマをターゲット DB インスタンスに手動で作成することができます。場合によっては、ターゲット DB インスタンスに同等のスキーマを作成することができないことがあります。DB エンジンから利用できる機能性をターゲット DB インスタンスのために使用するには、アプリケーションとデータベースの一部を再設計することが必要となる場合があります。これ以外の場合、自動変換できないスキーマは単に無視することができます。

  警告: ターゲット DB インスタンスにスキーマを手動で作成する場合は、行った手動作業すべてのコピーを保存するまで [**Apply to database**] を選択しないでください。プロジェクトからのスキーマをターゲット DB イ

ンスタンスに適用すると、ターゲット DB インスタンスにある同じ名前のスキーマが上書きされ、手動で追加したアップデートのすべてが失われます。

- ソースデータベーススキーマを変更して、プロジェクトでスキーマを更新します。一部のアイテムについては、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キテクチャと互換性があり、ターゲット DB インスタンスの DB エンジンに自動変換できるスキーマに変更することが最善策である場合があります。ソースデータベースでスキーマをアップデートし、アップデートにアプリケーションとの互換性があることを確認した後で、プロジェクトの左パネルで [**Refresh from Database**] を選択し、ソースデータベースからのスキーマをアップデートします。この後、アップデートされたスキーマを変換し、移行アセスメントレポートを再度生成することができます。アップデートされたスキーマのアクションアイテムは表示されなくなります。

13. 変換し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、プロジェクトの右パネルからスキーマ要素を選択します。以下の図にある通り、スキーマ要素を右クリックして [**Apply to database**] を選択します。

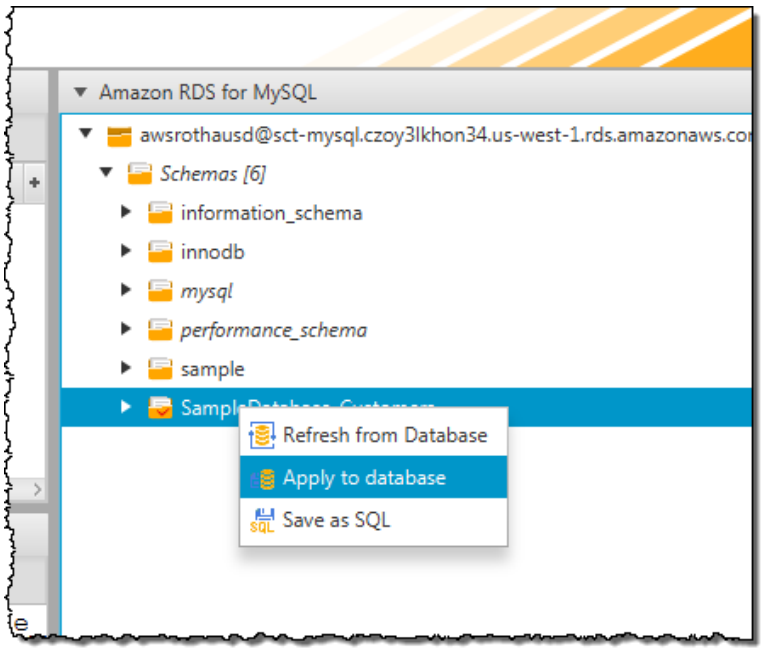


図 10: データベースへのスキーマの適用

注意: 変換したスキーマをターゲット DB インスタンスに初めて適用するときは、AWS Schema Conversion Tool が追加のスキーマ (`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、変換し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となる、ソースデータベースの system 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た変換済みスキーマで予期しない結果が生じる場合があります。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 Schema Conversion Tool が不要になったら、`AWS_ORACLE_EXT` または `AWS_SQLSERVER_EXT` スキーマを削除できます。

AWS Schema Conversion Tool は、移行ツールキットに対する使いやすい追加機能です。AWS Schema Conversion Tool に関連する追加のベストプラクティスについては、*AWS Schema Conversion Tool User Guide* のトピック「Best Practices」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 Aurora データベースにコピーされたら、次のステップは、実際のデータのソースからターゲットへの移行です。データ移行はさまざまなツールを使用して実行できますが、AWS Database Migration Service (AWS DMS) は、シンプルさと、目下のタスクに必要な機能の両方を提供することから、これを使用してデータを移動することが推奨されます。

AWS DMS の概要と一般的なアプローチ

AWS Database Migration Service (DMS) により、顧客は最小限のダウンタイムで本番データベースを AWS に簡単に移行することが可能になります。アプリケーションは、データベースの移行中も実行させておくことができます。さらに、AWS Database Migration Service は、移行中とその後に生じるソースデータへの

変更が継続的にターゲットにレプリケートされることを確実にします。移行タスクは、AWS マネジメントコンソールを使って数分でセットアップすることができます。AWS Database Migration Service は、Oracle、SQL Server、MySQL、PostgreSQL、Amazon Aurora、MariaDB、および Amazon Redshift などの広く利用されているデータベースプラットフォーム間でデータを移行させることができます。

このサービスでは、Oracle から Oracle のような同種のデータベース間の移行も、Oracle から Amazon Aurora または SQL から MySQL といった異なるデータベースプラットフォーム間の移行もサポートされます。顧客に複雑なソフトウェアをインストールして設定させることなく、データベース間で 1 回限りの移行を実行したり、継続的なレプリケーションを維持することができます。

AWS DMS は、オンプレミスのデータベース、Amazon EC2 または Amazon RDS で実行されるデータベースとの使用が可能です。ただし、AWS DMS は一方のエンドポイントが AWS であることを必要とし、ソースデータベースとターゲットデータベースがオンプレミスである場合は使用できません。

AWS DMS は、特定のバージョンの Oracle、SQL Server、Amazon Aurora、MySQL、および PostgreSQL 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 ユーザーガイドを参照してください。[18]ただし、このホワイトペーパーは、移行ターゲットとしての Amazon Aurora のみに焦点を当てています。

移行メソッド

AWS DMS はデータを移行するために 3 つのメソッドを提供します。

既存データの移行。このメソッドはターゲットデータベースにテーブルを作成し、ターゲットで必要とされるメタデータを自動定義して、ソースデータベースからのデータをテーブルに投入します (これは「フルロード」とも呼ばれます)。テーブルからのデータは、効率性向上のために並行してロードされます。テーブルは同種間移行の場合のみに作成され、セカンダリインデックスは AWS DMS によって自動的に作成されません。詳細については、この後の説明をお読みください。

既存データの移行と進行中の変更のレプリケート。このメソッドは、前に説明したフルロードを行いますが、それに加え、フルロード中にソースデータベースに対して行われる進行中の変更のすべてをキャプチャし、それらをレプリケーションインスタンスに保存します。フルロードが完了したら、宛先データベースがソースデータベースと同じ状態になるまで、保存された変更が宛先データベースに

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ている進行中の変更は引き続き宛先データベースにレプリケートされ、同期化された状態を維持します。この移行メソッドは、極めて少ないダウンタイムでデータベース移行を行いたい場合に非常に便利です。

データ変更のみのレプリケート。このメソッドは、ソースデータベースの復旧ログファイルからの変更のみを読み取り、これらの変更をターゲットデータベースに継続的に適用します。ターゲットデータベースが使用できない場合、これらの変更はターゲットデータベースが使用可能になるまでレプリケーションインスタンスにバッファされます。

AWS DMS がフルロード移行を実行するときは、移行処理がソースデータベース内のテーブルに負荷をかけるため、このデータベースを同時にヒットするアプリケーションのパフォーマンスに影響を与える可能性があります。これが問題であり、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、以下のアプローチを検討することができます。

- データベースに対するアプリケーション負荷が最も低い時点で移行を実行する。
- ソースデータベースのリードレプリカを作成し、そのリードレプリカから AWS DMS 移行を実行する。

移行手順

AWS DMS を使用するための概要は次の通りです。

1. ターゲットデータベースを作成する。
2. スキーマをコピーする。
3. AWS DMS レプリケーションインスタンスを作成する。
4. データベースのソースとターゲットのエンドポイントを定義する。
5. 移行タスクを作成して実行する。

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」で説明され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]ターゲットデータベースは、ビジネス要件に一致するリージョン内に、一致するインスタンスタイプで作成するようにしてください。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 配置が有効化されていないことを確認してください。これはロード終了後に有効化できます。

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む、基本的なスキーマ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に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を自動で作成しません。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。一般的に、より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ことでデータベース移行が迅速化されます (ただし、移行はソースおよびターゲットのデータベースの容量、接続レイテンシーなどのその他要因によって妨げられる可能性もあります)。また、レプリケーションインスタンスはデータベース移行が完了したら停止することができます。

図 11: AWS Database Migration Service

AWS DMS では現在、レプリケーションインスタンスのために 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 インスタンスクラスは、ベースラインレベルの CPU パフォーマンスを提供するように設計された低コストの標準インスタンスで、パフォーマンスをベースラインを超えるレベルにバーストさせる機能を備えています。これらは、データベース移行プロセスの開発、設定、およびテストだけでなく、CPU のバースト機能が役に立つ定期的なデータ移行タスクにも適しています。C4 インスタンスクラスは、最高レベルのプロセッサー

パフォーマンスを提供するように設計されており、T2 よりもはるかに高い PPS (packet per second) 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レイテンシーを実現します。C4 インスタンスクラスは、大規模なデータベースを移行しており、かつ移行時間を最小限に抑えたい場合に使用するようにしてください。

通常、フルロードの実施は、AWS DMS レプリケーションインスタンスに大量のインスタンスストレージを必要としません。ただし、フルロードと共にレプリケーションを実行している場合は、フルロードが行われている間、ソースデータベースへの変更が AWS DMS レプリケーションインスタンスに保存されます。このため、非常に大きく、移行進行中に受け取るアップデートも多いソースデータベースを移行する場合は、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、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備わっています。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、トランザクション速度が極めて高い、非常に大きなデータベースがレプリケーションを有効にした状態で移行されているという極端な例では、AWS DMS レプリケーションが時間内に追いつけない可能性もあります。この状況が発生した場合は、アプリケーションをターゲット Aurora DB に再度ポイントする前に、ソースデータベースへの変更を何分間か停止して、レプリケーションに遅れを取り戻させることが必要となる場合があります。

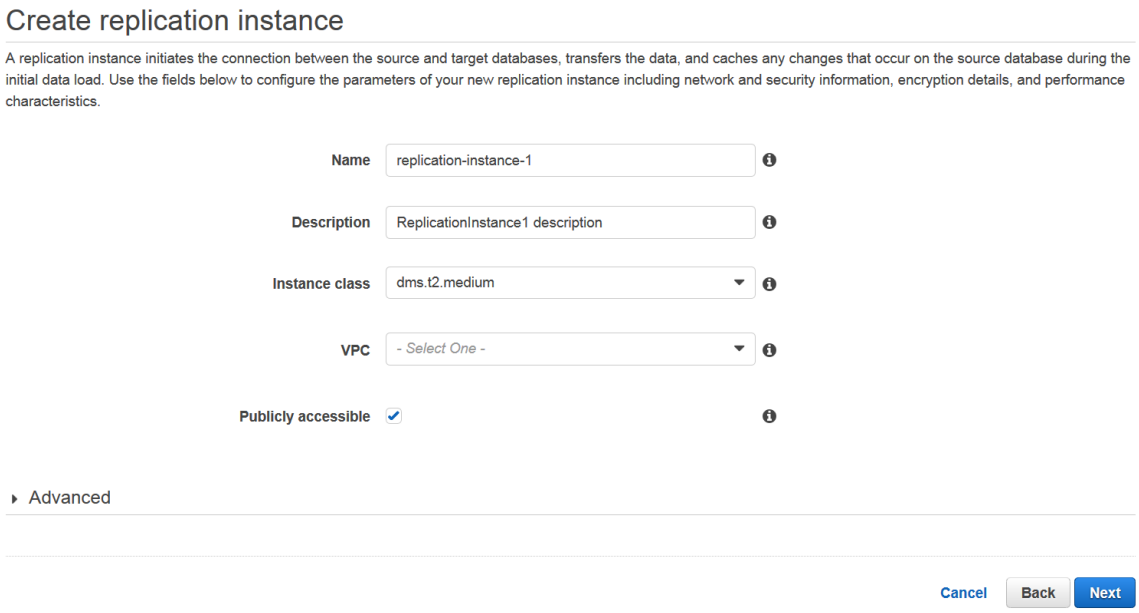


図 12: AWS DMS コンソールのレプリケーションインスタンスの作成ページ

データベースのソースおよび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、データベースへの接続の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。指定されたデータベースエンドポイントは、オンプレミスのもの、Amazon EC2 または Amazon RDS で実行されているものに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にすることはできません。

定義後は、データベースエンドポイント接続をテストすることが強く推奨されます。このホワイトペーパーで後ほど説明する通り、データベースエンドポイントを作成するために使用したものと同じページを、エンドポイントをテストするために使用することもできます。

注意: ソーススキーマに外部キー制約がある場合は、ターゲットエンドポイントの作成時、 [**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
```

これはターゲットテーブルのロード中における外部キーチェックを無効化します。その結果、部分的に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るロードの中断が阻止されます。

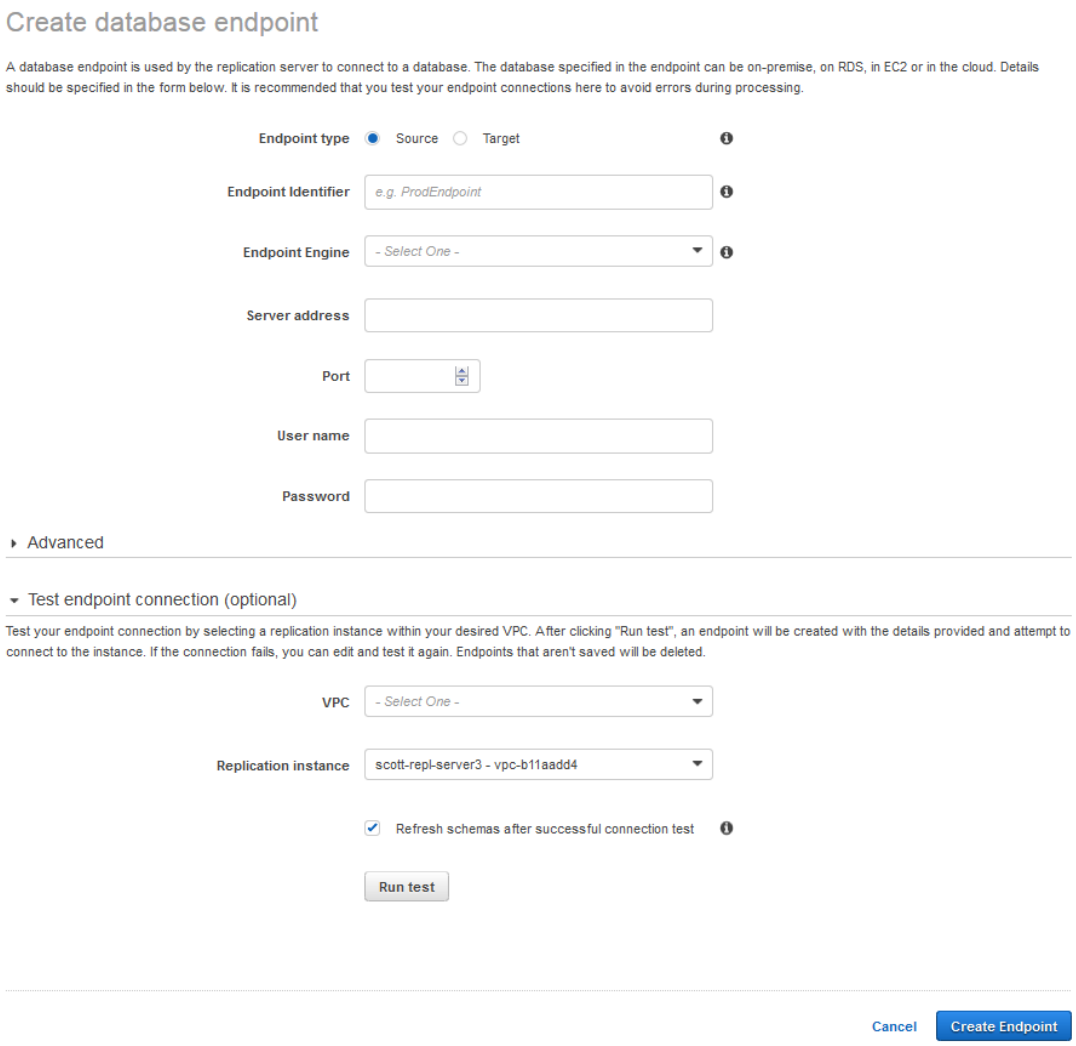


図 13: AWS DMS コンソールのデータベースエンドポイントの作成ページ

移行タスクの作成と実行

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作成とテストを終えたら、データ移行を行うためのタスクを作成できます。タスクを作成するときは、作成したレプリケーションインスタンス、データベース移行メソッドのタイプ (前述)、ソースデータベースエンドポイント、および Amazon Aurora データベースクラスター用の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、ターゲットデータベースで完全なスキーマを既に作成した場合は、[**Task Settings**] で、[**Target table preparation mode**] にデフォルト値の [**Drop tables on target**] を使用するのではなく、それを [**Do nothing**] に変更するようにしてください。[**Drop tables on target**] は、これがテーブルを削除して再

作成するときに、外部キー制約などのスキーマ定義のアスペクトが失われる原因となる場合があります。

タスクを作成するときは、ターゲットエンドポイントに移行されるソーススキーマと共に対応するテーブルを指定するテーブルマッピングを作成することができます。デフォルトのマッピングメソッドは、同じ名前のターゲットテーブルが存在する場合に、すべてのソーステーブルをそれらに移行します。それ以外の場合は、ソーステーブルをターゲットに作成します (タスク設定に応じて異なります)。さらに、特定のテーブルのみを移行したい、またはフィールドおよびテーブルマッピングプロセスをより詳細に制御したい場合は、カスタムマッピングを作成することもできます (JSON ファイルを使用)。ソースエンドポイントから 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ように選択することもできます。

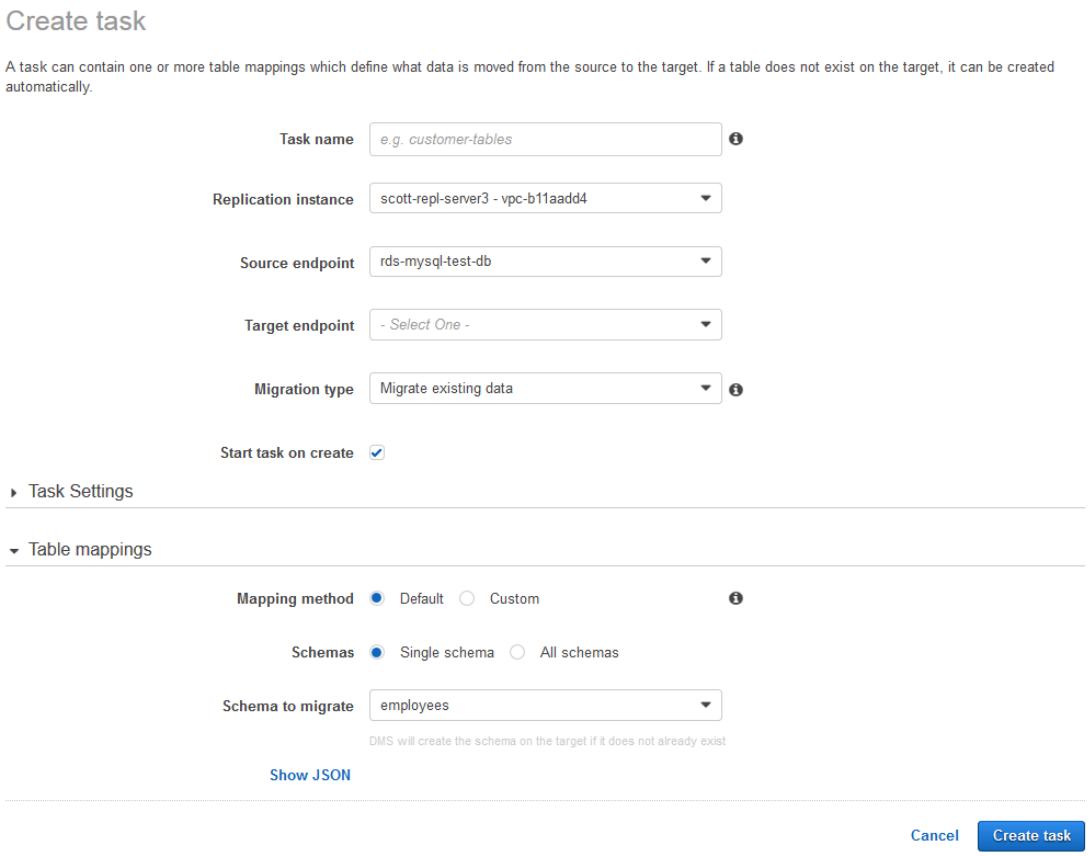


図 14: AWS DMS コンソールのタスクの作成ページ

AWS Database Migration Service (AWS DMS) タスクの進捗状況を監視するには、AWS マネジメントコンソールを使用することができます。また、使用されたリソースとネットワーク接続を監視することもできます。AWS DMS コンソールは、

以下の図にある通り、タスクステータス、完了率、経過時間、およびテーブル統計を含む各タスクの基本的な統計を表示します。

さらに、タスクを選択し、そのタスクのパフォーマンスメトリックスを表示することもできます。これには、スループット、1 秒毎に移行されたレコード数、ディスクとメモリの使用率、およびレイテンシーが含まれます。

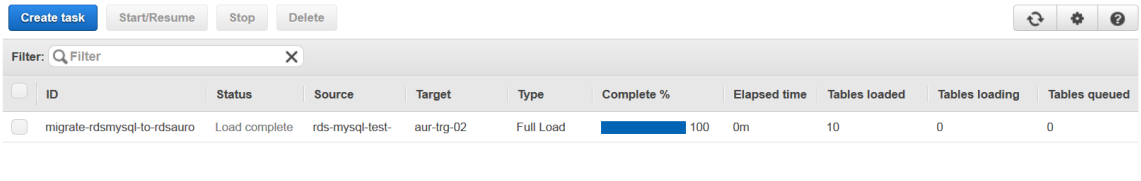


図 15: AWS DMS コンソールのタスクステータス

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テストを実行する準備が整います。テストアプローチは各テスト移行後に調整する必要があり、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を含めるようにします。

移行テスト

テストカテゴリ	目的
基本的な受け入れテスト	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完了時に自動で実行される必要があります。これらの主な目的は、データ移行が正常に行われたかどうかを確認することです。以下は、これらのテストからの一般的な出力の一部です。 • 処理されたアイテムの合計数 • インポートされたアイテムの合計数 • スキップされたアイテムの合計数 • 警告の合計数 • エラーの合計数 テストによって報告されたこれらの合計数のいずれかが期待値から逸脱する場合は、移行が正常に行われなかったことを意味し、プロセスの次のス

テストカテゴリ	目的
	テップ、または次回のテストに進む前に問題を解決する必要があります。
機能テスト	これらのカットオーバー後のテストは、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動作させます。これらには、自動テストと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生じたアプリケーションの問題を特定することです。
非機能テスト	これらのカットオーバー後のテストは、様々なレベルの負荷下におけるパフォーマンスなど、アプリケーションの非機能特性を評価します。
ユーザー受け入れテスト	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ときにアプリケーションのエンドユーザーによって実施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内での主な機能を満たすための使用に十分適しているかどうかをエンドユーザーが判断することです。

カットオーバー

最終的な移行とテストを完了したら、アプリケーションを Amazon Aurora データベースをポイントします。この移行フェーズは カットオーバーと呼ばれています。計画およびテストフェーズが適切に実行されていれば、カットオーバーで予期しない問題が生じることはありません。

カットオーバー前のアクション

- カットオーバーの時間枠を選択します。ビジネスに対する支障を最小限に抑えながら、新しいデータへのカットオーバーを達成できる時間ブロックを特定します。通常は、データベースの低アクティビティ期間を選択します (一般に夜間および/または週末)。
- 変更の遅れが取り戻されていることを確認します。ソ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ションにダウンタイムがゼロに近い移行アプローチ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いており、ターゲットデータベースがソースデータベースに大きな遅れを取っていないことを確認してください。
- アプリケーションの設定変更を行うスクリプトを準備します。カットオーバーを達成するには、アプリケーション設定ファイルでデータベース接続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には、

複数箇所で接続詳細をアップデートすることが必要となる場合があります。接続設定を迅速かつ確実にアップデートするため、必要なスクリプトの準備ができていることを確認してください。

- アプリケーションを停止します。ソースデータベースに対してさらに書き込みが行われないようにするため、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ます。ターゲットデータベースがソースデータベースの変更に完全に追いついていない場合、これらの変更がターゲットデータベースに完全に反映されるまでしばらく待ちます。
- カットオーバー前のテストを実行します。自動化されたカットオーバー前のテストを実行して、データ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

- カットオーバーを実行します。カットオーバー前のチェックが正常に完了した場合は、アプリケーションを Amazon Aurora にポイントすることができます。カットオーバー前のフェーズで作成したスクリプトを実行し、新しい Aurora データベースをポイントするようアプリケーション設定を変更します。
- アプリケーションを開始します。アプリケーションは、この時点で開始することができます。アプリケーション実行中にユーザーによるアプリケーションへのアクセスを停止できる場合は、カットオーバー後のチェックを実行し終えるまでそのオプションを利用します。

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を実行します。事前定義された自動または手動テストケースを実行し、アプリケーションが新しいデータベースで期待どおりに動作することを確認します。データベースへの書き込みを行うテストを実行する前に、まずデータベースの読み取り専用機能のテストを開始することが良い戦略です。
- ユーザーアクセスを有効化して、綿密に監視します。テストケースが正常に実行された場合、ユーザーにアプリケーションへのアクセスを許可して移行プロセスを完了することができます。アプリケーションとデータベースはどちらも、この時に綿密に監視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高パフォーマンス、高可用性のエンタープライズグレードのデータベースです。Amazon Aurora を活用することによって、他のオープンソースデータベースよりも優れたパフォーマンスと可用性、そしてほとんどの商用グレードのデータベースよりも低いコストを実現することができます。このホワイトペーパーは、データベースを Amazon Aurora に移行する最良の方法を識別するための戦略を提案し、これらの移行を計画して実施するための手順を詳しく説明します。特に、AWS Database Migration Service (AWS DMS) および AWS Schema Conversion Tool は、異種間移行シナリオに対する推奨ツールです。これらの強力なツールは、データベース移行に伴うコストと複雑さを大幅に削減することができます。

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。

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

その他の資料

追加情報については、次の資料を参照してください。

- 製品の詳細 – Amazon Aurora
- よくある質問 – Amazon Aurora
- Amazon Database Migration Service
- よくある質問 – Amazon Database Migration Service

注意

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

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

[3] https://d0.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