Amazon RDSおよび Amazon Aurora での PostgreSQL パラメータのチューニング

AWS 規範ガイダンス

AWS 規範ガイダンス: Amazon RDSおよび Amazon Aurora での PostgreSQL パラメータのチューニング

Table of Contents

Amazon RDS と Amazon Aurora での PostgreSQL パラメータのチューニング

Sumana Yanamandra、Ramu Jagini、Rohit Kapoor、Amazon Web Services (AWS)

2024 年 2 月 (ドキュメント履歴)

Amazon Aurora PostgreSQL -Compatible Edition および Amazon Relational Database Service (Amazon RDS) for PostgreSQL は、幅広い機能を提供する高度なオープンソースのリレーショナルデータベースサービスです。これらのサービスを使用して、さまざまなプラットフォームやアプリケーションに PostgreSQL データベースを設定できます。

Aurora と Amazon RDS では、PostgreSQL データベースを簡単に管理および運用できます。これらは、データベースインフラストラクチャを管理し、アプリケーション開発に集中しながら、高可用性、耐久性、スケーラビリティを提供するように設計されています。ただし、これらのサービスのデフォルト設定は、すべてのワークロードに最適ではない場合があります。デフォルトでは、これらのサービスは可能な限り少ないリソースで、脆弱性を発生させることなく、あらゆる場所で実行されるように設定されています。パラメータを調整すると、パフォーマンスの向上、ダウンタイムの短縮、データベース全体の効率の向上に役立ちます。特定のワークロードのパラメータを最適化することで、Amazon RDS と Aurora が提供する機能を最大限に活用し、その利点を最大化できます。

例えば、Aurora と Amazon RDS for PostgreSQL を最適化し、それらのパラメータを設定することで、パフォーマンスを向上させることができます。データベースクエリを作成するときは、パフォーマンスも考慮する必要があります。データベース設定を最適化した場合でも、クエリがテーブルスキャン全体を実行した場合、インデックスを利用する場合、または高価な結合または集計オペレーションを実行した場合、システムはパフォーマンスが低下する可能性があります。

このガイドは、PostgreSQL データベースのメモリ、autovacuum、ロギング、論理レプリケーションパラメータを調整するデータベース開発者、データベースエンジニア、管理者を対象としています。このガイドでは、Amazon RDS for PostgreSQL および Aurora PostgreSQL 互換に固有のパラメータについても説明します。これらのパラメータを調整すると、データベースのパフォーマンスを最適化し、特定のワークロードのリソース使用量を削減できるため、パフォーマンスとコスト削減が向上します。

DB および DB クラスターパラメータグループの使用

Amazon RDS と Aurora は、データベースインスタンスのサイズに基づいて、特定の設定に最も適したパラメータ値を自動的に決定できます。また、データベースインスタンスとクラスターのパラメータグループによるパフォーマンスチューニングのパラメータカスタマイズもサポートしています。

DB および DB クラスターパラメータグループを使用して、メモリ使用量、ディスク I/O、ネットワーク、ロックなど、データベースエンジンの動作のさまざまな側面を制御するパラメータを変更できます。これらのパラメータを調整することで、特定のワークロードに合わせてデータベースエンジンを最適化し、パフォーマンスを向上させることができます。

、 AWS Command Line Interface（ AWS CLI ）、または Amazon RDS API を使用して AWS Management Console、DB および DB クラスターパラメータグループを作成および設定できます。このガイドでは、 を使用していることを前提としています AWS CLI。コンソールと API の手順については、Amazon RDS ドキュメントの「DB パラメータグループの使用」および「DB クラスターパラメータグループの使用」を参照してください。

> **Important**
>
> このガイドに記載されている AWS CLI コマンドを使用するには、まず をインストールして設定する必要があります AWS CLI。

DB パラメータグループを作成して設定するには：

```
# Create a new DB parameter group
aws rds create-db-parameter-group \
  --db-parameter-group-name mydbparamgroup \
  --db-parameter-group-family postgres13 \
  --description "My DB Parameter Group"

# Modify a parameter on the DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <param group name> \
  --parameters "ParameterName=max_connections<parameter-
name>,ParameterValue=<value>,ApplyMethod=immediate"

# Verify DB parameters
aws rds describe-db-parameters \
```

```
  --db-parameter-group-name aurora-instance-1
```

DB クラスターパラメータグループを作成して設定するには：

```
# Create a new DB cluster parameter group
aws rds create-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --db-parameter-group-family postgres12 \
  --description "My new parameter group"

# Modify a parameter on the DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name aws-guide-cluster \
  --parameters "ParameterName=<parameter-name>,ParameterValue=,ApplyMethod=immediate"

# Allocate the new DB cluster parameter to your cluster
aws rds modify-db-cluster  \
  --db-cluster-identifier  \
  --db-cluster-parameter-group-name=-cluster

# Verify cluster parameters
aws rds describe-db-cluster-parameters \
  --db-cluster-parameter-group-name=-cluster
```

> **Note**
>
> Aurora と Amazon RDS は、変更できない事前設定された値を持つデフォルトのパラメータグループを提供します。

パラメータグループは、静的または動的に設定できます。動的パラメータは、`ApplyMethod=immediate` オプションが有効になっているかどうかにかかわらず、すぐに適用されます。静的パラメータを有効にするには、手動で再起動する必要があります。

メモリパラメータのチューニング

メモリパラメータのチューニングは、Amazon RDS および Aurora PostgreSQL 互換データベースのパフォーマンスを最適化するために不可欠なタスクです。クエリの実行、ソート、インデックス作成、キャッシュなど、さまざまなデータベースオペレーションにメモリを適切に割り当てると、データベースのパフォーマンスが大幅に向上します。このセクションでは、デフォルト値、適切な値を計算する式、変更方法など、Amazon RDS および Aurora PostgreSQL -Compatible の最も重要なメモリパラメータについて説明します。パラメータの完全なリストについては、「Amazon RDS ドキュメント」の「RDS for PostgreSQL DB インスタンスでのパラメータの操作」および「Aurora ドキュメント」の「Amazon Aurora PostgreSQL パラメータ」を参照してください。

これらのパラメータを最適化するには、データベースのワークロードと、Aurora または Amazon RDS DB インスタンスで使用できるリソースを深く理解する必要があります。システムパフォーマンスは、重要なパラメータと条件付きパラメータの 2 つのカテゴリのパラメータの影響を受けます。

必須パラメータは、システムのパフォーマンスに大きく直接的な影響を与える不可欠なパラメータであり、最適な結果を達成するために不可欠です。

- shared_buffers
- temp_buffers
- effective_cache_size
- work_mem
- maintenance_work_mem

臨時パラメータは、シナリオおよびビジネス固有のパラメータです。これらは他の要因に左右され、重要なパラメータをサポートし、システム全体のパフォーマンスを最大化する上で間接的で重要な役割を担います。

- random_page_cost
- seq_page_cost
- track_activity_query_size
- idle_in_transaction_session_timeout
- statement_timeout
- 検索パス

- max_connections

これらのパラメータについては、以下のセクションで詳しく説明します。

shared_buffers

shared_buffers パラメータは、PostgreSQL がデータをメモリにキャッシュするために使用するメモリの量を制御します。このパラメータを適切な値に設定すると、クエリのパフォーマンスを向上させることができます。

Amazon RDS の場合、 のデフォルト値shared_buffersは、DB {DBInstanceClassMemory/32768} インスタンスで使用可能なメモリに基づいてバイトに設定されます。Aurora の場合、DB インスタンスで使用可能なメモリに基づいて{DBInstanceClassMemory/12038,-50003}、デフォルト値は に設定されます。このパラメータの最適な値は、データベースのサイズ、同時接続数、使用可能なインスタンスメモリなど、いくつかの要因によって異なります。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループshared_buffersに対して変更されます。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify shared_buffers on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=shared_buffers,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify shared_buffers on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters "ParameterName=shared_buffers,ParameterValue=<new-
value>,ApplyMethod=immediate"
```

タイプ: Static (変更を適用するには再起動が必要です)

デフォルト値: Amazon RDS for PostgreSQL の{DBInstanceClassMemory/32768}バイト、{DBInstanceClassMemory/12038,-50003}Aurora PostgreSQL - 互換。ほとんどの場合、この方程式はシステムのメモリの約 25% になります。このガイドラインに従って、パラメータグ

ループの shared_buffers設定は、PostgreSQL のデフォルトの単位である 8K バッファを使用して、バイトまたはキロバイトではなく設定されます。

shared_buffers パラメータ設定はパフォーマンスに大きな影響を与える可能性があるため、変更を徹底的にテストして、値がワークロードに適していることを確認することをお勧めします。

例

Amazon RDS または Aurora で PostgreSQL データベースを実行している金融サービスアプリケーションがあるとします。このデータベースは、顧客トランザクションデータを保存するために使用されます。多数のテーブルがあり、多数のサーバー上の複数のアプリケーションからアクセスされます。アプリケーションでは、クエリのパフォーマンスが低下し、CPU 使用率が高くなっています。shared_buffers パラメータを調整すると、パフォーマンスの向上に役立つ場合があります。

Amazon RDS for PostgreSQL では、 のデフォルト値shared_buffersは {DBInstanceClassMemory/32768} で使用可能なメモリのバイト数 db.r5.xlarge (3 GB など) に設定されます。に適切な値を決定するにはshared_buffers、使用可能なメモリのデフォルト値から始めて、値を徐々に増やしてshared_buffers、さまざまな値で一連のテストを実行します。テストごとに、データベースのクエリパフォーマンスと CPU 使用率を測定します。

テスト結果に基づいて、 の値を shared_buffers 8 GB に設定すると、ワークロードの全体的なクエリパフォーマンスと CPU 使用率が最大になると判断します。値は、データベースのサイズ、クエリの数と複雑さ、同時ユーザー数、使用可能なシステムリソースなど、ワークロード特性のテストと分析の組み合わせによって決まります。変更を加えると、モニタリングシステムはデータベースのパフォーマンスをチェックして、新しい値がワークロードに適していることを確認します。その後、必要に応じて追加のパラメータを微調整して、パフォーマンスをさらに向上させることができます。

temp_buffers

temp_buffers は、Aurora PostgreSQL -Compatible および Amazon RDS for PostgreSQL の主要な設定パラメータであり、一時テーブルでのソート、ハッシュ、集計オペレーションを含むワークロード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。このパラメータは、一時バッファに割り当てられるメモリの量を決定し、その結果、このようなオペレーションの効率と速度に影響します。に十分なメモリが割り当てられていない場合temp_buffers、システムはテンポラリテーブルのソート、ハッシュ、集約オペレーションに低速で効率の低い方法を使用する必要があり、パフォーマンスが最適ではない可能性があります。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループtemp_buffersに対して変更します。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify temp_buffers on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=temp_buffers,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify temp_buffers on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=temp_buffers,ParameterValue=<new_value>,ApplyMethod=immediate"
```

タイプ: 動的 (を設定すると変更がすぐに適用されるApplyMethod=immediate)

デフォルト値: 8 MB

このパラメータの詳細については、PostgreSQL ドキュメントの「リソース消費」を参照してください。 PostgreSQL

例

ワークロードに、一時テーブルに対するソート、ハッシュ、および集計操作が大量に含まれている場合、 は十分なメモリを割り当てないtemp_buffers可能性があります。この場合、システムは、インメモリソート、ハッシュ、集計オペレーションではなく、ディスクベースのメソッドが遅くなるテンポラリテーブルに対してソートオペレーションを実行しなければならない場合があります。これにより、特に大規模なデータセットを含むクエリでは、クエリのパフォーマンスが大幅に低下する可能性があります。の値を大きくtemp_buffersすると、このような操作をメモリ内で実行するために十分なメモリを確保できるため、パフォーマンスが大幅に向上します。

の最適な値を見つけるにはtemp_buffers、システムパフォーマンスをモニタリングし、最適でないパフォーマンスの領域を特定します。クエリの応答時間が遅い、または CPU 使用率が高い場合は、 の調整を検討してくださいtemp_buffers。例えば、ワークロードに多数の一時テーブルが含まれる場合、 の値を大きくtemp_buffersすると、これらのテーブルがメモリに保存されるようになります。これは、ストレージからの読み取り/書き込み I/O を使用するよりもはるかに高速です。

のさまざまな値をtemp_buffers少しずつ試し、変更のたびにシステムパフォーマンスを注意深くモニタリングします。パフォーマンスに対するさまざまな値の影響を分析し、ワークロードの特定の特性に基づいて設定を微調整します。

effective_cache_size

effective_cache_size パラメータは、PostgreSQL がデータのキャッシュに使用できると仮定するメモリ量を指定します。このパラメータを正しく設定すると、PostgreSQL が使用可能なメモリをより有効に活用できるようになるため、パフォーマンスが向上します。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループeffective_cache_sizeに対して変更されます。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify effective_cache_size on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=effective_cache_size,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify effective_cache_size on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=effective_cache_size,ParameterValue=<new_value>,ApplyMethod=immediate"
```

タイプ: 動的 (を設定すると変更がすぐに適用されるApplyMethod=immediate)

デフォルト値: SUM(DBInstanceClassMemory/12038,-50003) KB

例

オンライン学習プラットフォームには、ユーザーが頻繁にアクセスするコース資料、学生データ、その他のコンテンツの大規模なデータベースがあります。アプリケーションは、32 GB のメモリを持つ Amazon RDS for PostgreSQL db.r5.xlargeインスタンスで実行されます。ユーザーが頻繁にアクセスされるコンテンツを読み込もうとすると、アプリケーションのパフォーマンスが低下します。データベースサーバーのリソース使用量を分析した後、PostgreSQL が使用可能なメモリを最適に活用していないと判断します。

Amazon RDS for PostgreSQL の effective_cache_sizeパラメータは、サーバーがディスクキャッシュに使用するメモリ量を制御します。インスタンスクラス のデフォルト値は SUM({DBInstanceClassMemory/12038},-50003) KB に設定されていますがdb.r5.xlarge、このデフォルト値はすべてのワークロードに適しているとは限りません。この例では、データベー

スサーバーが頻繁にアクセスされる大量のコース資料と学生データを保存している可能性があります。effective_cache_size パラメータの値を大きくすると、より多くのデータがメモリにキャッシュされ、必要なディスク読み取りの数が減り、クエリのパフォーマンスが向上します。

クエリを実行すると、Amazon RDS for PostgreSQL はまず、クエリに必要なデータが既にキャッシュにあるかどうかを確認します。その場合は、ディスクから読み取られるのではなく、メモリから読み取ることができます。データがキャッシュにない場合は、ディスクから読み取る必要があり、処理が遅くなる可能性があります。

オンライン学習プラットフォームでは、テストと分析後に effective_cache_sizeを 16 GB (使用可能なメモリの半分) に設定することもできます。この値により、PostgreSQL は使用可能なメモリをより有効に活用できるため、必要なディスク読み取り回数が減り、クエリのパフォーマンスが向上します。

work_mem

work_mem パラメータは、ソートおよびハッシュ操作のためにクエリで使用されるメモリの量を制御します。デフォルト値は 4 MB です。クエリに複数のオペレーションが含まれている場合、オペレーションごとに最大 4 MB を使用できます。の値を大きくwork_memすると、ソートまたはハッシュを必要とするクエリのパフォーマンスが向上します。これらのオペレーションにはより多くのメモリが必要になるためです。ただし、このパラメータを高く設定しすぎると、メモリ使用量が過剰になり、パフォーマンスが低下する可能性があります。

の最適な値を計算するにはwork_mem、次の式を使用できます。

```
work_mem = (available_memory / (max_connections * work_mem_fraction))
```

ここで、 available_memory はサーバーで使用可能なメモリの合計量、max_connections は許可される接続の最大数、work_mem_fraction は各接続に割り当てる使用可能なメモリの量を決定する割合です。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループwork_memについて変更します。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify work_mem on a DB parameter group
aws rds modify-db-parameter-group \
   --db-parameter-group-name <parameter_group_name> \
```

```
   --parameters
 "ParameterName=work_mem,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify work_mem on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
   --db-cluster-parameter-group-name <parameter_group_name> \
   --parameters
 "ParameterName=work_mem,ParameterValue=<new_value>,ApplyMethod=immediate"
```

タイプ: 動的 (を設定すると変更がすぐに適用されるApplyMethod=immediate)

デフォルト値: 4 MB

例

ソーシャルメディア分析ツールは大量のデータを処理し、複雑なソートおよび結合操作を含むクエリは、ディスク I/O が高くなり、ディスクに流出します。の値を 4 MB work_memから 16 MB に増やすと、PostgreSQL はこれらのオペレーションにより多くのメモリを使用できます。これにより、I/O の量が減り、クエリのパフォーマンスが向上します。

maintenance_work_mem

maintenance_work_mem パラメータは、、VACUUM、ANALYZEインデックス作成などのメンテナンスオペレーションで使用されるメモリ量を制御します。Amazon RDS および Aurora のこのパラメータのデフォルト値は 64 MB です。

このパラメータの適切な値を計算するために、次の式を使用できます。

```
maintenance_work_mem = (total_memory - shared_buffers) / (max_connections * 5)
```

Aurora PostgreSQL -Compatible Edition と Amazon RDS for PostgreSQL は、次の式を適用して最適な値を設定します。

```
GREATEST({DBInstanceClassMemory/63963136*1024},65536)
```

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループmaintenance_work_memに対して変更されます。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify maintenance_work_mem on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=maintenance_work_mem,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify maintenance_work_mem on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=maintenance_work_mem,ParameterValue=<new_value>,ApplyMethod=immediate"
```

タイプ: 動的 (を設定すると変更がすぐに適用されるApplyMethod=immediate)

デフォルト値: 64 MB

例

大規模なアプリケーションは、Aurora または Amazon RDS でホストされている PostgreSQL データベースを使用します。バキューム処理やインデックス作成などのメンテナンス作業中に、データベースの速度が遅く、応答がないことに気付きました。メモリ使用量、メンテナンスオペレーション時間、CPU 使用率などのメトリクスをモニタリングして、 の現在の値が問題の原因maintenance_work_memとなっているかどうかを判断できます。

の最適な値を決定するにはmaintenance_work_mem、 パラメータを調整し、その影響をモニタリングします。メンテナンスオペレーション中にメモリ使用量が一貫して多い場合や、オペレーション時間が予想よりも長い場合は、 を増やすと役立つmaintenance_work_mem場合があります。逆に、メンテナンスオペレーション中に CPU 使用率が一貫して高い場合、 を減らすと役立つmaintenance_work_mem場合があります。調整とテストを繰り返すことで、メモリ使用量、メンテナンスオペレーション時間、CPU 使用量maintenance_work_memのバランスが最適な の最適な値を見つけることができます。

調査中に、 のデフォルト値である 64 MB maintenance_work_memがデータベースサイズに対して低すぎると判断したとします。その結果、メンテナンスオペレーションの完了に時間がかかり、過剰なダウンタイムが発生し、アプリケーションのパフォーマンスが低下します。この問題に対処するために、maintenance_work_memパラメータを 64 MB から 512 MB に増やすことで、パラメータを調整することもできます (最適な値として指定します) 。変更を適用すると、メンテナンスオペレーション時間が 3 分の 2 向上します。例えば、以前に完了までに 30 分かかったバキューム操作は、10 分しかかかりません。この最適化の結果、データベースはメンテナンスアクティビティをより効率的に処理できるようになりました。

random_page_cost

random_page_cost パラメータは、ランダムページアクセスを実行するコストを決定するのに役立ちます。Amazon RDS および Aurora のクエリプランナーは、このパラメータをテーブルに関する他の統計とともに使用して、クエリを実行するための最も効率的な計画を決定します。

seq_page_cost パラメータと random_page_costパラメータは密接に関連しており、通常はプランナーが一緒に使用して、さまざまなアクセス方法のコストを比較し、どちらが最も効率的かを判断します。したがって、これらのパラメータのいずれかを変更する場合は、他のパラメータを調整する必要があるかどうかも考慮する必要があります。

一般に、クエリプランナーはクエリの実行コストを最小限に抑えようとします。コストは、ディスクページの読み取り数と の値を組み合わせて決定されますrandom_page_cost。の値が大きいほどシーケンシャルスキャンが優先random_page_costされ、値が低いほどインデックススキャンが優先される傾向があります。また、値を小さくすると、ハッシュ結合ではなくネストされたループ結合が優先される傾向があります。

random_page_cost パラメータグループまたはローカルセッションで値が設定されていない限り、パラメータはデフォルトの PostgreSQL エンジン値 (4) を使用します。この値は、サーバーとワークロードの特定の特性に応じて調整できます。ワークロードで使用されるインデックスのほとんどがメモリまたは Aurora 階層型キャッシュに収まる場合は、 の値を に近いrandom_page_cost値に変更するのが適切seq_page_costです。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループrandom_page_costに対して変更されます。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify random_page_cost on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=random_page_cost,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify random_page_cost on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=random_page_cost,ParameterValue=<new_value>,ApplyMethod=immediate"
```

タイプ: 動的 (を設定すると変更がすぐに適用されるApplyMethod=immediate)

デフォルト値: 4

例

インデックスが付いていない列のフィルターで頻繁にクエリされる大量のデータをテーブルに保存しているデータベースがあるとします。クエリの完了には時間がかかり、クエリプランナーはデータにアクセスするための最も効率的な計画を選択しません。

パフォーマンスを向上させる 1 つの方法は、 random_page_costパラメータを減らすことです。これを 1 に設定すると、ランダムページアクセスのコストはデフォルト値の 4 倍安くなります。デフォルト値の 4 random_page_costのままにすると、ランダムページアクセスはシーケンシャルページアクセスよりも 4 倍高価になります (デフォルトでは 1.0 である seq_page_costパラメータによって決まります) 。ただし、この特定のケースでは、ストレージタイプによっては、ランダムなページアクセスの方が実際にははるかにコストがかかる場合があります。

random_page_cost パラメータの値を減らすと、クエリプランナーがインデックスベースの計画を選択する可能性が高くなり、テーブルの特定の特性により適した別のアクセス方法を使用する可能性が高くなります。

パラメータを変更した後、クエリのパフォーマンスをモニタリングし、必要に応じて調整することをお勧めします。また、クエリプランナーを EXPLAINステートメントでチェックして、効率的な計画を選択しているかどうかも確認する必要があります。

これは 1 つの例にすぎません。最適な設定は、ワークロードの特定の特性によって異なります。また、これはパフォーマンスチューニングの一側面にすぎません。クエリのパフォーマンスに影響を与える可能性のある他のパラメータや設定オプションも考慮する必要があります。

seq_page_cost

seq_page_cost パラメータは、シーケンシャルページアクセスを実行するコストを決定するのに役立ちます。Amazon RDS および Aurora のクエリプランナーは、このパラメータをテーブルに関する他の統計とともに使用して、クエリを実行するための最も効率的な計画を決定します。

seq_page_cost パラメータと random_page_cost パラメータは密接に関連しており、通常はプランナーが一緒に使用して、さまざまなアクセス方法のコストを比較し、どちらが最も効率的かを判断します。したがって、これらのパラメータのいずれかを変更する場合は、他のパラメータを調整する必要があるかどうかも考慮する必要があります。

テーブルに順次アクセスすると、PostgreSQL はオペレーティングシステムのファイルシステムキャッシュを使用して、より高速なアクセスを提供できます。デフォルトでは、seq_page_costは 1.0 に設定されています。これは、シーケンシャルページアクセスが単一のディスクブロック読み取りと同じくらい安価であることを前提としています。主にシーケンシャル方式でアクセスされるテーブルがあるが、IOPS が少ないためにディスクアクセスが遅い場合は、ディスクへのアクセスの追加コストを反映するseq_page_costために の値を増やすことをお勧めします。

このパラメータの値を変更すると、システムで実行されるすべてのクエリに影響するため、クエリを異なる値でテストして、特定のユースケースに最適な値を決定することをお勧めします。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループseq_page_costに対して変更されます。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify seq_page_cost on a DB parameter group
aws rds modify-db-parameter-group \
   --db-parameter-group-name <parameter_group_name> \
   --parameters
 "ParameterName=seq_page_cost,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify seq_page_cost on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
   --db-cluster-parameter-group-name <parameter_group_name> \
   --parameters
 "ParameterName=seq_page_cost,ParameterValue=<new_value>,ApplyMethod=immediate"
```

タイプ: 動的 (を設定すると変更がすぐに適用されるApplyMethod=immediate)

デフォルト値: 1.0

例

主に順次アクセスされるテーブルに大量のデータを保存するデータベースがあるとします。テーブルは主にレポートに使用され、クエリの実行は非常に遅く、クエリプランナーはデータにアクセスするための最も効率的な計画を選択できません。

パフォーマンスを向上させる 1 つの方法は、 seq_page_costパラメータを減らすことです。デフォルト値は 1.0 です。これは、シーケンシャルページアクセスが 1 つのディスクブロック読み取りと同じくらい安価であることを前提としています。ただし、この特定のケースでは、ストレージタイプ (I/O によって異なります) により、シーケンシャルページアクセスの方が実際にはそれよりもコ

ストがかかる場合があります。seq_page_cost を 0.5 に設定すると、シーケンシャルページアクセスのコストはデフォルト値の半分になります。この変更により、クエリプランナーは、テーブルの特定の特性により適したシーケンシャルアクセス方式を使用する計画を選択する可能性が高くなります。

パラメータを変更した後、クエリのパフォーマンスをモニタリングし、必要に応じて調整することをお勧めします。また、クエリプランを EXPLAINステートメントでチェックして、効率的なプランを選択しているかどうかも確認する必要があります。

これは 1 つの例にすぎません。最適な設定は、ワークロードの特定の特性によって異なります。また、これはパフォーマンスチューニングの一側面にすぎません。クエリのパフォーマンスに影響する他のパラメータや設定オプションも考慮する必要があります。

track_activity_query_size

track_activity_query_size パラメータは、pg_stat_activityビュー内のアクティブなセッションごとに口グに記録されるクエリ文字列のサイズを制御します。デフォルトでは、クエリ文字列の最初の 1,024 バイトのみが Amazon RDS for PostgreSQL に記録され、4,096 バイトが Aurora PostgreSQL 互換に記録されます。長いクエリを口グに記録する場合は、このパラメータをより高い値に設定できます。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループtrack_activity_query_sizeについて変更します。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify track_activity_query_size on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=track_activity_query_size,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify track_activity_query_size on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=track_activity_query_size,ParameterValue=<new_value>,ApplyMethod=immediate"
```

タイプ: Static (変更を適用するには再起動が必要です)

デフォルト値: 1,024 バイト (Amazon RDS for PostgreSQL)、4,096 バイト (Aurora PostgreSQL 互換)

例

Amazon RDS for PostgreSQL データベースのクエリパフォーマンスが遅く、問題が長時間実行されるクエリに関連している可能性がある。より長いクエリを pg_stat_activityビューにログ記録することで、さらに調査できます。

track_activity_query_size パラメータの値を大きくすると、ログ記録が増加し、データベースのパフォーマンスに影響する可能性があります。問題が解決されたら、 パラメータをデフォルトの 1,024 値に戻すことをお勧めします。

idle_in_transaction_session_timeout

idle_in_transaction_session_timeout パラメータは、アイドル状態のトランザクションが停止するまでの待機時間を制御します。

Amazon RDS for PostgreSQL および Aurora PostgreSQL -Compatible のこのパラメータのデフォルト値は 86,400,000 ミリ秒です。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループidle_in_transaction_session_timeoutに対して変更されます。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify idle_in_transaction_session_timeout on a DB parameter group
aws rds modify-db-parameter-group \
   --db-parameter-group-name <parameter_group_name> \
   --parameters
 "ParameterName=idle_in_transaction_session_timeout,ParameterValue=<new_value>,ApplyMethod=imme

# Modify idle_in_transaction_session_timeout on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
   --db-cluster-parameter-group-name <parameter_group_name> \
   --parameters
 "ParameterName=idle_in_transaction_session_timeout,ParameterValue=<new_value>,ApplyMethod=imme
```

タイプ: 動的 (を設定すると変更がすぐに適用されるApplyMethod=immediate)

デフォルト値: 86,400,000 ミリ秒 (Aurora PostgreSQL 互換)

例

オンライン注文を処理する e コマースアプリケーションがあります。アプリケーションは、Amazon RDS または Aurora でホストされている PostgreSQL データベースを使用します。顧客が注文を行うたびに、アプリケーションは新しいトランザクションを開始してインベントリレコードと注文レコードを更新します。

トランザクションを長時間アイドル状態にしておくと、他のトランザクションが同じレコードにアクセスできなくなり、パフォーマンスの問題やアプリケーションのダウンタイムが発生する可能性があります。さらに、適切に停止されていないアイドル状態のトランザクションは、メモリや CPU などの貴重なシステムリソースを消費する可能性があります。

このような問題を回避するには、 idle_in_transaction_session_timeoutパラメータをアプリケーションに適した値に設定します。例えば、5 分 (300 秒) に設定すると、5 分以上アイドル状態のままになっているトランザクションが自動的に停止されます。これにより、システムリソースが効率的に使用され、アプリケーションが多数の注文を処理でき、速度が低下することはありません。に適切な値を設定することでidle_in_transaction_session_timeout、アプリケーションが Amazon RDS または Aurora で最適に動作するようにできます。

statement_timeout

statement_timeout パラメータは、クエリが停止するまでに実行できる最大時間を設定します。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループstatement_timeoutについて変更します。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify statement_timeout on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=statement_timeout,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify statement_timeout on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=statement_timeout,ParameterValue=<new_value>,ApplyMethod=immediate"
```

タイプ: 動的 (を設定すると変更がすぐに適用されるApplyMethod=immediate)

デフォルト値: 0 ミリ秒 (タイムアウトなし)

例

ウェブアプリケーションを使用すると、ユーザーは製品の大規模なデータベースを検索できます。検索クエリの完了に時間がかかる場合があり、ユーザーの応答時間が遅くなることがあります。この問題に対処するには、 statement_timeoutパラメータを 10 秒などの低い値に設定することで、10 秒を超えるクエリを強制的に停止できます。

これは劇的な対策のように思えるかもしれませんが、実際にはパフォーマンスの向上に非常に効果的です。多くの場合、実行時間が長いクエリは、最適化されていない SQL クエリや非効率的なインデックスが原因で発生します。statement_timeout 値を低く設定することで、これらの問題のあるクエリを特定し、それらを最適化する手順を実行できます。

例えば、特定の検索クエリが一貫してタイムアウトしていることが判明したとします。EXPLAIN やなどのツールを使用してクエリEXPLAIN ANALYZEを分析し、パフォーマンスのボトルネックを特定できます。問題を特定したら、新しいインデックスの追加、クエリの書き換え、または別の検索アルゴリズムを使用して、クエリを最適化する手順を実行できます。この方法で SQL クエリを継続的に分析して最適化することで、アプリケーションのパフォーマンスを大幅に向上させることができます。

search_path

search_path パラメータは、SQL ステートメントでスキーマがオブジェクトを検索する順序を決定します。デフォルト値は です。つまり$user, public、PostgreSQL は最初にユーザー名に一致するスキーマでオブジェクトを検索し、次にパブリックスキーマでオブジェクトを検索します。

スキーマの数が多い場合や、特定のスキーマ内のオブジェクトにアクセスする必要がある場合は、search_path パラメータを変更することでパフォーマンスを向上させることができます。を特定のスキーマsearch_pathに設定すると、PostgreSQL は複数のスキーマを検索しなくてもオブジェクトをより迅速に見つけることができます。

Amazon RDS および Aurora の search_pathパラメータを変更するには、次のコマンドをROLEレベルに使用できます。

```
ALTER ROLE <username> SET search_path = <schema>;
```

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループsearch_pathを変更します。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify search_path on a DB parameter group
aws rds modify-db-parameter-group \
   --db-parameter-group-name <parameter_group_name> \
   --parameters
 "ParameterName=search_path,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify search_path on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
   --db-cluster-parameter-group-name <parameter_group_name> \
   --parameters
 "ParameterName=search_path,ParameterValue=<new_value>,ApplyMethod=immediate"
```

タイプ: 動的 (を設定すると変更がすぐに適用されるApplyMethod=immediate)

デフォルト値: $user, public

例

Amazon RDS for PostgreSQL または Aurora PostgreSQL 互換データベースを使用するテナントごとに個別のスキーマを持つマルチテナントアプリケーションがあり、複数のスキーマからのデータの結合を含むクエリを実行する必要があります。

デフォルトでは、Amazon RDS と Aurora は検索パスを使用して、特定のテーブルに使用するスキーマを決定します。検索パスは、スキーマ名を修飾せずにテーブルを参照するときに PostgreSQL が順番に検索するスキーマ名のリストです。デフォルトでは、Amazon RDS と Aurora はまず現在のユーザーと同じ名前のテーブルをスキーマ内で検索し、次にパブリックスキーマを検索します。

、、tenant1tenant2および という名前の複数のスキーマからテーブルを結合するクエリを実行するとしますtenant3。テナントスキーマを使用するには、クエリにスキーマ名を含めることができます。

```
SELECT *
FROM tenant1.table1
JOIN tenant2.table2 ON tenant1.table1.id = tenant2.table2.id
JOIN tenant3.table3 ON tenant2.table2.id = tenant3.table3.id;
```

ただし、より効率的な方法は、AWS CLI 構文セクションのコマンドを使用して、テナントスキーマを含めるように search_pathパラメータを変更することです。PostgreSQL セッションで SET コマンドを使用することもできます。

```
SET search_path = tenant1, tenant2, tenant3, public;
```

その後、スキーマ名を修飾せずにクエリを記述できます。

```
SELECT *
FROM table1
JOIN table2 ON table1.id = table2.id
JOIN table3 ON table2.id = table3.id;
```

これにより、クエリがより簡潔で読みやすくなり、複数のスキーマからテーブルを結合するクエリが多数ある場合は、アプリケーションコードを簡素化することもできます。

max_connections

max_connections パラメータは、PostgreSQL データベースの同時接続の最大数を設定します。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループmax_connectionsについて変更します。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify max_connections on a DB parameter group
aws rds modify-db-parameter-group \
--db-parameter-group-name <parameter_group_name> \
--parameters
 "ParameterName=max_connections,ParameterValue=<new_value>,ApplyMethod=pending-reboot"

# Modify max_connections on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
--db-cluster-parameter-group-name <parameter_group_name> \
--parameters
 "ParameterName=max_connections,ParameterValue=<new_value>,ApplyMethod=pending-reboot"
```

タイプ: Static (変更を適用するには再起動が必要です)

デフォルト値: LEAST(DBInstanceClassMemory/9531392,5000)接続

Amazon RDS または Aurora max_connectionsでの の使用を最適化し、パフォーマンスへの影響を最小限に抑えるには、次のベストプラクティスを検討してください。

- 使用可能なシステムリソースに基づいてパラメータ値を設定します。
- 接続の使用状況をモニタリングして、制限にすばやく到達しないようにします。
- 接続プーリングを使用して、必要な接続の数を減らします。
- 接続プーリングには Amazon RDS Proxy を使用します。

Amazon RDS for PostgreSQL または Amazon Aurora PostgreSQL -Compatible max_connectionsで調整する場合は、使用可能なインスタンスタイプと割り当てられたリソースを検討し、メモリと CPU 容量に焦点を当てます。ストレージと I/O の詳細は によって管理されるため AWS、Amazon CloudWatch や Amazon RDS コンソールFreeableMemoryなどの一般的なワークロード特性とシステムメトリクスをモニタリングして、接続に十分なメモリがあることを確認できます。高いCPUUtilization値をモニタリングします。これは、調整が必要であることを示している可能性があります。設定が高max_connectionsすぎるとメモリ使用量に影響し、I/O に間接的に影響する可能性があるため、設定しすぎないようにしてください。各接続はメモリを消費することに注意してください。適切なバランスを見つけるには、徐々に増max_connectionsやして、それがシステムにどのように影響するかを確認します。パフォーマンスの低下や CPU 使用率が高い兆候がないか監視します。アプリケーションが引き続き正常に動作するかどうかを確認します。Aurora のリードレプリカなどの機能を使用して、読み取りトラフィックを分散し、プライマリインスタンスの負荷を軽減します。特定のリソース制約内で最適なデータベースパフォーマンスを確保するために、観測された使用パターンに基づいてmax_connections定期的に確認および調整します。

autovacuum パラメータのチューニング

Amazon RDS for PostgreSQL データベースと Aurora PostgreSQL 互換には、 のバキューム処理と呼ばれる定期的なメンテナンスが必要です。Autovacuum は、古いデータや不要なデータを削除してデータベース内の領域を解放する組み込み PostgreSQL ユーティリティです。autovacuum プロセスは、コマンドを定期的にバックグラウンドVACUUMで実行します。

autovacuum 設定のチューニングは、Amazon RDS for PostgreSQL または Aurora PostgreSQL 互換データベースシステムのパフォーマンス、安定性、可用性を維持する上で重要なステップです。ワークロードとデータベースサイズに合わせて autovacuum パラメータを調整することで、autovacuum プロセスのパフォーマンスを最適化し、システムリソースへの影響を軽減できるため、データベースの全体的なヘルスが向上します。

autovacuum 設定の調整に加えて、Amazon RDS と Aurora で利用可能なツールとメトリクスを使用して、データベースとそのコンポーネントのパフォーマンスをモニタリングすることが重要です。肥大化、空き領域、クエリの実行時間などのパフォーマンスメトリクスをモニタリングすることで、重大な問題になる前に潜在的な問題を特定し、解決するための適切なアクションを実行できます。

このセクションでは、以下の autovacuum トピックとパラメータについて説明します。

- VACUUM コマンドと ANALYZE コマンド
- 肥大化の確認
- autovacuum
- autovacuum_work_mem
- autovacuum_naptime
- autovacuum_max_workers
- autovacuum_vacuum_scale_factor
- autovacuum_vacuum_threshold
- autovacuum_analyze_scale_factor
- autovacuum_analyze_threshold
- autovacuum_vacuum_cost_limit

autovacuum の詳細については、次のリンクを参照してください。

- Amazon RDS for PostgreSQL 環境での autovacuum について (ブログ記事)

- 「Amazon RDS for PostgreSQL での PostgreSQL autovacuum の使用」(Amazon RDS ドキュメント)
- Amazon RDS for PostgreSQL と Amazon Aurora PostgreSQL での並列バキューム (ブログ記事)

VACUUM コマンドと ANALYZE コマンド

VACUUM ガベージコレクションはデータベースを収集し、オプションで分析します。ほとんどのアプリケーションでは、autovacuum デーモンにバキューム処理を実行させるだけで十分です。ただし、一部の管理者は、autovacuum のデータベースパラメータを変更したり、スケジューラに従って実行できる手動で管理されたVACUUMコマンドを使用してデーモンのアクティビティを補足または置き換えたりする場合があります。

VACUUM は、デッドタプルによって占有されているストレージを再利用します。標準の PostgreSQL オペレーションでは、更新によってタプルが削除されたり、廃止されたりしても、VACUUMオペレーションが実行されるまでテーブルからタプルは物理的に削除されません。したがって、特に頻繁に更新されるテーブルでは、VACUUM定期的に を実行することをお勧めします。

Amazon RDS for PostgreSQL および Aurora PostgreSQL -Compatible では、VACUUMパラメータのチューニングが特に重要です。これらのマネージドデータベースサービスは、セルフマネージド PostgreSQL データベースとは異なる特性を持つためです。これらの違いは、バキューム操作のパフォーマンスに影響を与える可能性があります。リソースの使用を最適化し、バキューム操作がデータベースシステムのパフォーマンスと可用性に悪影響を及ぼさないようにするには、VACUUMパラメータを調整することが不可欠です。

Aurora PostgreSQL -Compatible および Amazon RDS for PostgreSQL の VACUUM コマンドで使用できるパラメータの一部を次に示します。

- FULL
- FREEZE
- VERBOSE
- ANALYZE
- DISABLE_PAGE_SKIPPING
- table_name
- column_name

VACUUM ANALYZE は、選択したテーブルごとに VACUUM オペレーションを実行し、その後に ANALYZEオペレーションを実行します。定期的なメンテナンスを効率的に実行できます。

FULL オプションを指定せずに VACUUM コマンドを使用すると、再利用のために領域が再利用されます。テーブルに排他的ロックを必要としないため、標準の読み取りおよび書き込みオペレーション中にこのコマンドを実行できます。ただし、ほとんどの場合、 コマンドはオペレーティングシステムに余分なスペースを返すのではなく、同じテーブル内で再利用できるようにします。 はテーブルの内容を余分なスペースなしで新しいディスクファイルに書きVACUUM FULL換え、未使用のスペースをオペレーティングシステムに返すことができます。このフォームははるかに遅く、各テーブルにACCESS EXCLUSIVEロックが必要です。

これらのパラメータの詳細については、PostgreSQL ドキュメント を参照してください。

Aurora と Amazon RDS では、autovacuum はデーモン (バックグラウンドユーティリティ) プロセスであり、 VACUUMおよび ANALYZE コマンドを定期的に実行してデータベースとサーバーの冗長データをクリーンアップします。autovacuuming に依存していても、最適なパフォーマンスを確保するために、次のセクションで説明する autovacuum 設定を確認して調整することをお勧めします。

肥大化の確認

次の SQL クエリは、XML スキーマ内の各テーブルを調べ、ディスク領域を浪費するデッド行 (タプル) を識別します。

```
SELECT schemaname || '.' || relname as tuplename,
    n_dead_tup,
    (n_dead_tup::float / n_live_tup::float) * 100 as pfrag
FROM pg_stat_user_tables
WHERE schemaname = 'xml' and n_dead_tup > 0 and n_live_tup > 0 order by pfrag desc;
```

このクエリがデッドタプルの高い割合 (プラグ) を返す場合は、 VACUUMコマンドを使用してスペースを再利用できます。

トランザクションの前後にデータサイズをモニタリングするには、特定のデータベースに接続した後、シェルで次のクエリを実行します。

```
SELECT pg_size_pretty(pg_relation_size('table_name'));
```

autovacuum

autovacuum は、 autovacuum設定パラメータを使用してグローバルに設定することも、テーブルのautovacuum_enabled列をfalse特定のpg_classテーブルに対して trueまたは に設定することでテーブルごとに変更することもできます。

テーブルで autovacuum を有効にすると、データベースサーバーはデータベース管理者の介入なしに、テーブルのデッド行とタプルを定期的にスキャンしてバックグラウンドで削除します。これにより、テーブルを小さく保ち、クエリのパフォーマンスを向上させ、バックアップのサイズを小さくすることができます。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループautovacuumに対して を有効にします。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify autovacuum on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters "ParameterName=autovacuum,ParameterValue=true,ApplyMethod=immediate"

# Modify autovacuum on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters "ParameterName=autovacuum,ParameterValue=true,ApplyMethod=immediate"
```

タイプ: 動的 (を設定すると変更がすぐに適用されるApplyMethod=immediate)

デフォルト値: 有効

psql を使用して、特定のテーブルで autovacuum を無効化または有効にすることもできます。

```
ALTER TABLE <table_name> SET (autovacuum_enabled = true);
```

バキューム処理が多すぎるとパフォーマンスに影響する可能性があるため、自動バキュームプロセスのパフォーマンスとデータベースのパフォーマンスをモニタリングし、必要に応じて設定を調整することが重要です。

例

PostgreSQL データベースには、大量の書き込みおよび削除オペレーションを受け取るテーブルがあります。autovacuum がないと、このテーブルは最終的にデッド行 (つまり、削除対象としてマークされているが、テーブルから物理的に削除されていない行) でいっぱいになります。これらのデッド行は、ディスクの領域を占有し、クエリを遅くし、バックアップのサイズを増やします。テーブルで autovacuum を有効にすると、デッド行を自動的にスキャンして削除し、これらの問題を軽減できます。

autovacuum_work_mem

autovacuum_work_mem は、バキューム処理や分析などのテーブルメンテナンスタスクを実行するときに autovacuum プロセスが使用するメモリ量を制御する PostgreSQL 設定パラメータです。

Aurora と Amazon RDS では、 の値を調整しautovacuum_work_memてパフォーマンスを最適化できます。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループautovacuum_work_mem に対して を有効にします。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify autovacuum_work_mem on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_work_mem,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify autovacuum_work_mem on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_work_mem,ParameterValue=<new_value>,ApplyMethod=immediate"
```

タイプ: 動的 (を設定すると変更がすぐに適用されるApplyMethod=immediate)

デフォルト値: Aurora PostgreSQL の GREATEST({DBInstanceClassMemory/32768},131072)KB - 互換、Amazon RDS for PostgreSQL の 64 MB。 PostgreSQL ただし、デフォルト値は、使用している Amazon RDS または Aurora の特定のバージョンによって異なる場合があります。

例

Amazon RDS for PostgreSQL データベースには、頻繁に更新される大きなテーブルがあります。時間の経過とともに、データベースの速度が遅くなり、autovacuum の完了に時間がかかりすぎていると考えられます。

調査の一環として、システムログをチェックし、 pg_stat_activityビューを使用して現在実行されているクエリとプロセスを確認し、 pg_stat_user_tables ビューで各テーブルの統計を確認し、 pg_settingsビューを使用して の値をシステム上のautovacuum_work_mem使用可能なメモリと比較し、メモリ使用量のスパイクをモニタリングします。この情報を収集したら、ワークロードに必要な最適な値autovacuum_work_memに を設定できます。メモリ使用量とパフォーマンスの適切なバランスを見つけるには、システム上の使用可能なメモリの 4 分の 1 に設定することを決定できます。値を変更したら、データベースのパフォーマンスをモニタリングし、autovacuum が以前よりもはるかに速く完了し、データベース全体のパフォーマンスが速くなることがあります。

autovacuum_naptime

autovacuum_naptime パラメータは、autovacuum プロセスの連続実行間隔を制御します。デフォルト値は、Amazon RDS for PostgreSQL の場合は 15 秒、Aurora PostgreSQL -Compatible の場合は 5 秒です。

例えば、Amazon RDS for PostgreSQL データベースに大量の書き込みおよび削除オペレーションを受け取るテーブルがあるとします。デフォルト設定のままにすると、頻繁な autovacuum スキャンは、この高度にトランスアクティブなテーブルを破壊します。このパラメータを高い値に設定すると、連続するスキャンの間隔が長くなり、デッド行の削除頻度が低くなります。

autovacuum_naptime を使用して、特に CPU または I/O 負荷が高いビジーサーバーがある場合に、バキューム処理によって発生する負荷を管理できます。スリープ時間を設定する時間が長くなるほど、autovacuum の実行頻度が低くなり、サーバーの負荷が軽減されます。ただし、autovacuum_naptime を非常に高い値に設定すると、PostgreSQL テーブルが大きくなり、デッド行が蓄積され、パフォーマンスが低下する可能性があります。autovacuum プロセスのパフォーマンスをモニタリングし、必要に応じてautovacuum_naptime設定を調整することをお勧めします。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループautovacuum_naptimeに対して変更されます。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify autovacuum_naptime on a DB parameter group
aws rds modify-db-parameter-group \
```

```
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_naptime,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify autovacuum_naptime on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_naptime,ParameterValue=<new_value>,ApplyMethod=immediate"
```

タイプ: 動的 (を設定すると変更がすぐに適用されるApplyMethod=immediate)

デフォルト値: 15 秒 (Amazon RDS for PostgreSQL)、5 秒 (Aurora PostgreSQL 互換)

autovacuum_max_workers

autovacuum_max_workers パラメータは、autovacuum プロセスが作成できるワーカープロセスの最大数を制御します。各ワーカープロセスは、1 つのテーブルをバキューム処理または分析する責任があります。

例えば、頻繁に更新および削除される多数のテーブルを持つ大規模なデータベースがあるとします。autovacuum_max_workers を 1 などの低い値に設定すると、一度にバキューム処理できるテーブルは 1 つだけであり、すべてのテーブルのクリーンアップに時間がかかります。autovacuum_max_workers を 8 などの高い値に設定すると、最大 8 つのテーブルを同時にバキューム処理できます。これにより、多数のテーブルを含むデータベースのクリーニングプロセスが高速化されます。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループautovacuum_max_workersについて変更します。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify autovacuum_max_workers on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_max_workers,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify autovacuum_max_workers on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
```

```
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_max_workers,ParameterValue=<new_value>,ApplyMethod=immediate"
```

タイプ: Static (変更を適用するには再起動が必要です)

デフォルト値: GREATEST(DBInstanceClassMemory/64371566592,3)ワーカー

autovacuum_max_workers 設定を増やすとサーバーの負荷が増加し、十分なリソースがない場合にパフォーマンスに影響する可能性があります。最適な設定は、データベースの特定の要件、サイズ、およびデータベースに含まれるテーブルの数によって異なります。ユースケースに最適な設定を見つけるために、さまざまな値を試し、パフォーマンスをモニタリングすることをお勧めします。

autovacuum_vacuum_scale_factor

autovacuum_vacuum_scale_factor 設定パラメータは、テーブルをバキューム処理するときに autovacuum プロセスがどの程度攻撃的になるかを制御します。

バキュームスケール係数は、autovacuum がテーブルをクリーンアップする前に変更する必要があるテーブル内のタプルの合計数の一部です。デフォルト値は 0.1 です (つまり、タプルの 10% を変更する必要があります)。例えば、テーブルに 1,000,000 個のタプルがあり、そのうち 100,000 個のタプルがデッドまたは削除としてマークされている場合、autovacuum は制御係数としての autovacuum_vacuum_thresholdの値に応じてテーブルをバキューム処理します。

autovacuum_vacuum_scale_factor パラメータは、バキュームプロセスの実行頻度を制御するのに役立ちます。テーブルに大量の書き込みオペレーションがある場合は、バキュームスケール係数を下げて、autovacuum の実行頻度を高め、テーブルを小さく保つことができます。逆に、テーブルの書き込みオペレーションが少ない場合は、バキュームスケール係数を上げて、autovacuum の実行頻度を低くし、リソースを節約できます。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループautovacuum_vacuum_scale_factorを変更します。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify autovacuum_vacuum_scale_factor on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_vacuum_scale_factor,ParameterValue=<new_value>,ApplyMethod=immediate
```

```
# Modify autovacuum_vacuum_scale_factor on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_vacuum_scale_factor,ParameterValue=<new_value>,ApplyMethod=immediate
```

タイプ: 動的 (を設定すると変更がすぐに適用されるApplyMethod=immediate)

デフォルト値: 0.1

autovacuum_vacuum_scale_factor パラメータは、autovacuum_vacuum_threshold、autovacuum_vacuum_cost_limit, and autovacuum_naptimeパラメータと組み合わせて機能します。このパラメータの詳細については、 AWS ブログ記事「Amazon RDS for PostgreSQL 環境での autovacuum について」を参照してください。

autovacuum_vacuum_threshold

autovacuum_vacuum_threshold パラメータは、autovacuum がテーブルをバキュームする前にテーブルで実行する必要があるタプル更新または削除オペレーションの最小数を制御します。この設定は、これらのオペレーションのレートが高くないテーブルで不要なバキューム処理を防ぐのに役立ちます。Amazon RDS for PostgreSQL と Aurora PostgreSQL -Compatible の両方で、デフォルト値は PostgreSQL エンジンのデフォルトである 50 です。

例えば、100,000 行のテーブルがあり、 autovacuum_vacuum_threshold が 50 に設定されているとします。テーブルが 49 個の更新または削除のみを受信した場合、autovacuum はテーブルをバキューム処理しません。テーブルが 50 個以上の更新または削除を受け取った場合、autovacuum は、 の値に制御係数としてテーブル行数を掛けたautovacuum_vacuum_scale_factor値に応じて、テーブルをバキューム処理します。

このパラメータを高く設定しすぎると、テーブルが大きくなり、行が枯渇して蓄積され、パフォーマンスに影響する可能性があります。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループautovacuum_vacuum_thresholdに対して変更されます。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify autovacuum_vacuum_threshold on a DB parameter group
aws rds modify-db-parameter-group \
```

```
    --db-parameter-group-name <parameter_group_name> \
    --parameters
 "ParameterName=autovacuum_vacuum_threshold,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify autovacuum_vacuum_threshold on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name <parameter_group_name> \
    --parameters
 "ParameterName=autovacuum_vacuum_threshold,ParameterValue=<new_value>,ApplyMethod=immediate"
```

タイプ: 動的 (を設定すると変更がすぐに適用されるApplyMethod=immediate)

デフォルト値: 50 オペレーション

autovacuum_vacuum_threshold パラメータは、autovacuum_vacuum_scale_factor、autovacuum_vacuum_cost_limit, and autovacuum_naptimeパラメータと組み合わせて動作します。最適な設定は、データベースとテーブルサイズの特定の要件によって異なります。

このパラメータの詳細については、 AWS ブログ記事「Amazon RDS for PostgreSQL 環境での autovacuum の理解」を参照してください。

autovacuum_analyze_scale_factor

autovacuum_analyze_scale_factor パラメータは、分析 (テーブル内のデータの分布に関する統計を収集) 時に autovacuum プロセスがどの程度積極的に行われるかを制御します。

autovacuum プロセスは、このパラメータを使用して、テーブル内のタプル数に基づいてしきい値を計算します。タプルの挿入、更新、または削除の数がこのしきい値を超えると、autovacuum はテーブルを分析します。Amazon RDS for PostgreSQL と Aurora PostgreSQL -Compatible の両方のデフォルト値は 0.05 (つまり、タプルの 5% を変更する必要があります) PostgreSQL です。

例えば、テーブルに 1,000,000 タプルがあり、autovacuum_analyze_scale_factorデフォルト値を 0.05 のままにするとします。テーブルが 50,000 件以上の更新または削除を受け取った場合、autovacuum はautovacuum_analyze_threshold値に応じてテーブルの行数を制御要素として追加してバキューム処理します。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループautovacuum_analyze_scale_factorに対して変更されます。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify autovacuum_analyze_scale_factor on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_analyze_scale_factor,ParameterValue=<new_value>,ApplyMethod=immediat

# Modify autovacuum_analyze_scale_factor on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_analyze_scale_factor,ParameterValue=<new_value>,ApplyMethod=immediat
```

タイプ: 動的 (を設定すると変更がすぐに適用されるApplyMethod=immediate)

デフォルト値: 0.05 (5%)

データへのアクセス方法や整理方法など、情報に基づいた意思決定を行うためには、クエリプランナーが統計を収集することが不可欠です。そのため、autovacuum プロセスのパフォーマンスをモニタリングし、必要に応じて設定を調整して、統計が最新であることを確認することをお勧めします。

autovacuum_analyze_scale_factor パラメータは、autovacuum_analyze_threshold、autovacuum_analyze_cost_limit, and autovacuum_naptimeパラメータと組み合わせて機能します。最適な設定は、データベースとテーブルのサイズ、および更新の頻度の特定の要件によって異なります。このパラメータの詳細については、 AWS ブログ記事「Amazon RDS for PostgreSQL 環境での autovacuum の理解」を参照してください。

autovacuum_analyze_threshold

autovacuum_analyze_threshold パラメータは に似ていますautovacuum_vacuum_threshold。autovacuum がテーブルを分析する前にテーブルで発生する必要があるタプルの挿入、更新、または削除の最小数を制御します。この設定は、これらのオペレーションの頻度が高いテーブルで不要なバキューム処理を防ぐのに役立ちます。Amazon RDS for PostgreSQL と Aurora PostgreSQL -Compatible の両方で、デフォルト値は PostgreSQL エンジンのデフォルトである 50 です。

例えば、100,000 行のテーブルがあり、autovacuum_analyze_thresholdデフォルトを 50 のままにするとします。テーブルが 49 個の挿入、更新、または削除のみを受信した場合、autovacuum はそれを分析しません。テーブルが 50 個以上の挿入、更新、または削除を受け取る場合、autovacuum はそれを分析し、 の値をテーブル行数にautovacuum_analyze_scale_factor乗算した値を制御係数として保持します。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループautovacuum_analyze_thresholdに対して変更します。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify autovacuum_analyze_threshold on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_analyze_threshold,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify autovacuum_analyze_threshold on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_analyze_threshold,ParameterValue=<new_value>,ApplyMethod=immediate"
```

タイプ: 動的 (を設定すると変更がすぐに適用されるApplyMethod=immediate)

デフォルト値: 50 オペレーション

このパラメータは autovacuum_analyze_scale_factorパラメータと組み合わせて機能するため、autovacuum を設定するときは両方の設定を考慮してください。

クエリプランナーは、データへのアクセス方法や整理方法など、情報に基づいた意思決定を行うために統計情報を収集することが不可欠です。設定が高autovacuum_analyze_thresholdすぎると、統計が古くなり、パフォーマンスが低下する可能性があります。autovacuum プロセスのパフォーマンスをモニタリングし、必要に応じて設定を調整することをお勧めします。

このパラメータの詳細については、 AWS ブログ記事「Amazon RDS for PostgreSQL 環境での autovacuum の理解」を参照してください。

autovacuum_vacuum_cost_limit

autovacuum_vacuum_cost_limit パラメータは、autovacuum ワーカーが消費できる CPU および I/O リソースの量を制御します。

autovacuum プロセスのリソース使用量を制限すると、CPU またはディスク I/O の消費が過度になりすぎて、同じシステムで実行される他のクエリのパフォーマンスに影響する可能性があります。パラ

メータは、コスト制限 を指定します。これは、ワーカーが一時停止して、まだ制限を下回っているかどうかを確認する前に実行できる作業単位です。例えば、 パラメータが 2,000 に設定されている場合、ワーカーは一時停止する前に 2,000 ユニットの作業を処理できます。

autovacuum_vacuum_cost_limit パラメータは、PostgreSQL セッションで SET コマンドを使用するか、 AWS CLI コマンドを使用して設定できます。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループautovacuum_vacuum_cost_limitに対して変更されます。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify autovacuum_vacuum_cost_limit on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_vacuum_cost_limit,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify autovacuum_vacuum_cost_limit on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_vacuum_cost_limit,ParameterValue=<new_value>,ApplyMethod=immediate"
```

タイプ: 動的 (を設定すると変更がすぐに適用されるApplyMethod=immediate)

デフォルト値: 作業GREATEST({log(DBInstanceClassMemory/21474836480)*600},200)単位

値を高く設定autovacuum_vacuum_cost_limitしすぎると、autovacuum プロセスが消費するリソースが多すぎて、他のクエリが遅くなる可能性があります。これを低く設定しすぎると、autovacuum プロセスが十分なスペースを再利用できず、時間の経過とともにテーブルが大きくなる可能性があります。システムに適したバランスを見つけることが重要です。

このパラメータは autovacuum プロセスにのみ影響し、手動VACUUMコマンドには影響しません。また、 の autovacuum プロセスにのみ適用VACUUMされ、 の autovacuum プロセスには適用されませんANALYZE。

ログ記録パラメータの調整

PostgreSQL でログパラメータを調整すると、システムを圧倒する大きなログを生成せずに、適切な情報を収集できます。

ログパラメータの最適化は、ログの詳細とシステムパフォーマンスおよびディスク使用量のバランスを取るために重要です。次のログパラメータをカスタマイズして、ログに適切な詳細レベルをキャプチャし、問題を診断し、システムパフォーマンスとディスク使用量への影響を最小限に抑えながらインシデントを効果的に調査できます。

- rds.force_autovacuum_logging
- rds.force_admin_logging_level
- log_duration
- log_min_duration_statement
- log_error_verbosity
- log_statement
- log_statement_stats
- log_min_error_statement
- log_min_messages
- log_temp_files
- log_connections
- log_disonnections

これらのパラメータについては、以下のセクションで詳しく説明します。

> **Warning**
>
> これらのパラメータの最適な設定は、組織のポリシーとコンプライアンス要件によって異なります。ただし、ログパラメータを有効にすると、大量のログやメッセージが生成される可能性があり、ストレージが使い果たされ、特にビジー状態のデータベースではパフォーマンスに影響する可能性があります。これらのパラメータは慎重に使用することをお勧めします。例えば、パフォーマンスの低い SQL ステートメントの問題の絞り込みを一時的に有効にし、モニタリング期間が終了したらオフにすることができます。

rds.force_autovacuum_logging

rds.force_autovacuum_logging パラメータ (Amazon RDS for PostgreSQL でのみ使用可能) は、autovacuum アクションがサーバーログに記録されるかどうかを制御します。その値はdisabled、、debug5、debug4、debug3、debug2、debug1、、info、notice、warning、、errorですpanic。デフォルト値は、warningです。

を有効にするとrds.force_autovacuum_logging、プロセスの開始時、終了時、バキューム処理する行数など、autovacuum プロセスのすべてのアクションがログに記録されます。これは、autovacuum のパフォーマンス問題のデバッグまたはトラブルシューティングに役立ちます。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループrds.force_autovacuum_loggingに対して変更されます。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify rds.force_autovacuum_logging on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=rds.force_autovacuum_logging,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify rds.force_autovacuum_logging on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=rds.force_autovacuum_logging,ParameterValue=<new_value>,ApplyMethod=immediate"
```

タイプ: 動的 (を設定すると変更がすぐに適用されるApplyMethod=immediate)

デフォルト値: warning

例

rds.force_autovacuum_logging パラメータを使用して、書き込みレートが非常に高いテーブルで autovacuum のパフォーマンスを分析できます。例えば、テーブルが 1 秒あたりに多数の書き込みおよび削除オペレーションを受信し、パフォーマンスが低下する場合は、 パラメータを有効にして各 autovacuum 実行の開始時刻と終了時刻をログに記録し、バキューム処理された行数を判断できます。これにより、autovacuum の実行頻度、

実行にかかる時間、バキューム処理する行数に関する貴重な情報が得られます。その後、この情報を使用して、、、autovacuum_vacuum_scale_factor、 などの autovacuum 設定autovacuum_naptimeを微調整しautovacuum_vacuum_threshold、パフォーマンスを最適化できます。

rds.force_admin_logging_level

rds.force_admin_logging_level パラメータ (Amazon RDS for PostgreSQL でのみ使用可能) は、バキューム処理、分析、インデックスの再作成などの管理オペレーションによって生成されるログの詳細レベルを制御します。値debug5、debug4、、debug3、debug2、debug1、、log、infonotice、、warning、、fatalおよび off （ デフォルト) error logを受け入れます。最適な設定はユースケースによって異なります。例えば、問題をトラブルシューティングする場合は、 パラメータをデバッグレベルに設定することができます。それ以外の場合は、、log、infoまたは warning設定を使用できます。

rds.force_admin_logging_level を に設定するとdebug1、開始時刻と終了時刻、処理された行数、プロセス中に発生したエラーや警告など、インデックス再作成オペレーションの詳細情報をログに記録できます。これにより、インデックス再作成プロセスの実行方法に関する貴重な情報が提供され、発生した問題のトラブルシューティングに役立ちます。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループrds.force_admin_logging_levelに対して変更されます。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify rds.force_admin_logging_level on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=rds.force_admin_logging_level,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify rds.force_admin_logging_level on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=rds.force_admin_logging_level,ParameterValue=<new_value>,ApplyMethod=immediate"
```

タイプ: 動的 (を設定すると変更がすぐに適用されるApplyMethod=immediate)

デフォルト値: off

例

を使用してrds.force_admin_logging_level、大規模なデータベース内の複数のテーブルにわたる管理オペレーションのパフォーマンスをモニタリングおよび分析できます。例えば、多数のテーブルを持つ大規模なデータベースがあり、それらのテーブルに対して定期的にバキューム処理と分析オペレーションを実行して、これらのテーブルのパフォーマンスを最適化したいとします。rds.force_admin_logging_level パラメータを infoまたは に設定することでlog、各オペレーションの開始時刻と終了時刻、および影響を受けたテーブルをログに記録できます。この情報を使用して、さまざまなテーブルにわたる管理オペレーションのパフォーマンスを追跡し、より頻繁または積極的なメンテナンスを必要とする可能性のあるテーブルを特定できます。

一部のログ記録レベルは、特にビジー状態のデータベースがある場合に、ディスク容量をすばやく埋めることができるログファイルとメッセージを多数生成します。このパラメータは慎重に使用し、モニタリング期間が終了したらオフにすることをお勧めします。

log_duration

log_duration パラメータは、各クエリの継続時間 (つまり、実行にかかる時間) がクエリに記録されるかどうかを制御します。このパラメータを に設定するとon、各クエリの実行にかかる時間が、クエリテキストとともにログ出力に含まれます。時間はミリ秒単位で測定されます。

log_duration パラメータの主なユースケースは、パフォーマンスのチューニングとトラブルシューティングを支援することです。各クエリの所要時間をログに記録することで、実行に時間がかかるクエリを特定し、それらのクエリの最適化に集中できます。これにより、パフォーマンスのボトルネックを特定して修正し、データベースの全体的なパフォーマンスを向上させることができます。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループlog_durationに対して変更されます。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify log_duration on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_duration,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify log_duration on a DB cluster parameter group
```

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_duration,ParameterValue=<new_value>,ApplyMethod=immediate"
```

タイプ: 動的 (を設定すると変更がすぐに適用されるApplyMethod=immediate)

デフォルト値: off

例

このパラメータは、特定のクエリまたは一連のクエリがパフォーマンスの問題を引き起こしている可能性がある場合に使用できます。log_duration パラメータを有効にして口グ出力を調べることで、どのクエリの実行に最も時間がかかるかを確認し、インデックスの最適化、新しいインデックスの追加、クエリの書き換えなどの適切なアクションを実行できます。

を有効にすると、ログ出力のボリュームが増加するlog_duration可能性があります。ストレージがいっぱいになったり、口グが読みにくくなったりしないように、必要な場合にのみ使用し、標準オペレーション中はオフにすることをお勧めします。

log_min_duration_statement

log_min_duration_statement パラメータは、SQL ステートメントが口グに記録される前に実行される最小時間をミリ秒単位で制御します。

このパラメータは、パフォーマンスの問題を引き起こす可能性のある実行時間が長いクエリを特定するのに役立ちます。しきい値 (特定のワークロードに対して長すぎると見なされるランタイム) に設定して、そのしきい値を超えるクエリをキャプチャし、潜在的なパフォーマンスのボトルネックを特定できます。ユースケースの例については、このガイドの後半の「ログパラメータを使用してバインド変数をキャプチャする」を参照してください。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループlog_min_duration_statementを変更します。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify log_min_duration_statement on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
```

```
  --parameters
 "ParameterName=log_min_duration_statement,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify log_ min_duration_statement on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_min_duration_statement,ParameterValue=<new_value>,ApplyMethod=immediate"
```

タイプ: 動的 (を設定すると変更がすぐに適用されるApplyMethod=immediate)

デフォルト値: 1 (PostgreSQL エンジンのデフォルトである無効)

例

次のコマンドは、実行に 100 ミリ秒以上かかるステートメントをすべて口グに記録します。

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name <parameter_group_name> \
   --parameters
 "ParameterName=log_min_duration_statement,ParameterValue=100,ApplyMethod=immediate"
```

log_error_verbosity

log_error_verbosity パラメータは、エラーレベル 以上でログに記録されるエラーやメッセージのログ出力に含まれる詳細レベルを制御します。このパラメータは、、terse、defaultまたは の 3 つの値のいずれかを取ることができますverbose。

- terse には、メッセージテキスト、エラーレベル、およびエラーが発生したファイルと行番号のみが含まれます。
- default には、メッセージテキスト、エラーレベル、ファイルと行番号、エラーコンテキストが含まれます。
- verbose には、メッセージテキスト、エラーレベル、ファイルと行番号、エラーコンテキスト、および完全なエラーメッセージが含まれます。

パラメータを verboseに設定して、非本番環境でのトラブルシューティングとデバッグに関する最も詳細な情報を取得します。実稼働環境では、必須情報のみterseを提供し、口グストレージにあまり詳細を埋めないように、 defaultまたは に設定することをお勧めします。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループlog_error_verbosityに対して変更されます。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify log_error_verbosity on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_error_verbosity,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify log_error_verbosity on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_error_verbosity,ParameterValue=<new_value>,ApplyMethod=immediate"
```

タイプ: 動的 (を設定すると変更がすぐに適用されるApplyMethod=immediate)

デフォルト値: default

log_statement

log_statement パラメータは、サーバーログに記録される SQL ステートメントを制御します。パラメータには、次のいずれかの値を指定できます。

- none (デフォルト) はステートメントを記録しません
- ddl は、 CREATE TABLEや などのデータ定義言語 (DDL) ステートメントのみを口グに記録します。 ALTER TABLE
- mod は、、INSERT、 などのデータ変更ステートメントのみを口グUPDATEに記録します。 DELETE
- all すべての SQL ステートメントを口グに記録する

log_statement パラメータを使用して、ユースケースに関連する特定のタイプのステートメントのみをドキュメント化することで、ログに書き込まれる情報の量を制御することができます。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループlog_statementに対して変更されます。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify log_statement on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_statement,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify log_statement on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_statement,ParameterValue=<new_value>,ApplyMethod=immediate"
```

タイプ: 動的 (を設定すると変更がすぐに適用されるApplyMethod=immediate)

デフォルト値: none

例

実稼働環境では、DDL ステートメントのみをログに記録し、データベーススキーマに加えられた変更を追跡log_statementddlするように を に設定することができます。開発環境では、デバッグとトラブルシューティングに役立つすべてのステートメントallをログに記録するように パラメータを に設定することもできます。別のユースケースの例については、このガイドの後半の「ログパラメータを使用してバインド変数をキャプチャする」を参照してください。

を有効にすると、ログ出力の量が増大log_statementする可能性があるため、必要な場合にのみ使用してオフにし、ストレージがいっぱいになったり、ログの読み取りが困難になったりしないようにします。

システムを監視し、このパラメータの値を調整して、ログに記録される情報量とシステムのストレージとパフォーマンスの適切なバランスを取ることをお勧めします。

log_statement_stats

log_statement_stats パラメータは、SQL ステートメントの実行に関連付けられた統計が ステートメントにログ記録されるかどうかを制御します。このパラメータをオンにすると、影響を受ける行数、読み書きされるディスクブロック数、ステートメントの実行にかかる時間などの統計がログ出力に含まれます。

log_statement_stats パラメータを使用して、個々のステートメントのパフォーマンスと全体的なワークロードに関する追加情報を収集できます。ステートメント統計をログに記録することで、クエリのパフォーマンスとリソースの使用状況のパターンを特定し、その情報を使用してデータベースを最適化し、全体的なパフォーマンスを向上させることができます。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループlog_statement_statsについて変更します。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify log_statement_stats on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_statement_stats,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify log_statement_stas on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_statement_stats,ParameterValue=<new_value>,ApplyMethod=immediate"
```

タイプ: 動的 (を設定すると変更がすぐに適用されるApplyMethod=immediate)

デフォルト値: off (PostgreSQL エンジンのデフォルト) 。0 または 1 (ブール値) を使用してパラメータグループに設定します。

例

log_statement_stats を使用して、特定のクエリの動作を分析し、CPU、メモリ、ディスク I/O などのリソースがどのように使用されるかを確認し、クエリを最適化できるかどうかを特定できます。このパラメータを使用して、特定のテーブルが頻繁に読み取られているかどうか (特定の列にインデックスを作成する必要があることを示している可能性がある) 、またはテーブルがスキャン頻度が高すぎるかどうかを確認することもできます。

を有効にすると、ログ出力のボリュームlog_statement_statsが増加する可能性があるため、必要なときにのみ使用してオフにし、ストレージがいっぱいになったり、ログの読み取りが難しくなったりしないようにします。

log_min_error_statement

log_min_error_statement パラメータは、エラーの原因となるSQL ステートメントをログに記録するかどうかを制御します。その値はdebug5、、debug4、debug3、debug2、debug1、info、、notice、warning、error、log、fata および ですpanic。これらの設定は、ログに書き込まれる情報の量を制御するため、重要度の低いメッセージをフィルタリングできます。このパラメータを重要度の高いレベルに設定すると、ログ出力の量を減らし、重要なメッセージを見つけやすくなります。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループlog_min_error_statementに対して変更されます。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify log_min_error_statement on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_min_error_statement,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify log_min_error_statement on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_min_error_statement,ParameterValue=<new_value>,ApplyMethod=immediate"
```

タイプ: 動的 (を設定すると変更がすぐに適用されるApplyMethod=immediate)

デフォルト値: error (PostgreSQL エンジンのデフォルト)

例

特定の問題をトラブルシューティングし、エラーの原因となっている SQL ステートメントからのエラーメッセージを表示log_min_error_statementしたい場合は、 の使用を検討してください。

log_min_messages

log_min_messages パラメータは、ログに書き込まれる重要度レベルを制御します。パラメータは、debug5、、debug4、debug3、debug2、debug1、info、、notice、warning、error、fatalま

たは logに設定できますpanic。これらの設定は、ログに書き込まれる情報の量を制御するため、重要度の低いメッセージをフィルタリングできます。このパラメータを重要度の高いレベルに設定すると、ログ出力の量を減らし、重要なメッセージを見つけやすくなります。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループlog_min_messagesに対して変更されます。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify log_min_messages on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_min_messages,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify log_min_messages on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_min_messages,ParameterValue=<new_value>,ApplyMethod=immediate"
```

タイプ: 動的 (を設定すると変更がすぐに適用されるApplyMethod=immediate)

デフォルト値: notice

例

特定の問題をトラブルシューティングしていて、すべてのエラーメッセージを表示したい場合は、エラーと上位レベルの重要度の問題のみをログerrorに記録するように、このパラメータを に設定することができます。システムのパフォーマンスのモニタリングに関心がある場合は、このパラメータを に設定infoして、各ステートメントの期間や統計などのより詳細な情報を表示できます。

をより高い重要度レベルlog_min_messagesに設定すると、ログの量が減ります。このパラメータは、特定のユースケース、チェックするログのサイズ、およびディスク容量に応じて調整することをお勧めします。

log_temp_files

log_temp_files パラメータは、一時ファイル名とサイズのログ記録を制御します。これは、ソート、ハッシュ、一時クエリ結果などの目的で作成された一時ファイルに適用されます。このパラメータを有効にすると、ファイルサイズを含む削除時に一時ファイルごとにログエントリがバイト単位

で生成されます。このパラメータを 0 (ゼロ) に設定すると、すべての一時ファイル情報が包括的にログに記録されます。また、そのサイズ (単位が指定されていない場合はキロバイト単位) を超えるログファイルの正の値に設定できます。これは、一時ストレージに関連するパフォーマンスのボトルネックやその他の問題を特定して解決するのに役立ちます。デフォルトでは、一時ファイルのログ記録は無効になっています。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループlog_temp_filesに対して変更されます。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify log_temp_files on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_temp_files,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify log_temp_files on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_temp_files,ParameterValue=<new_value>,ApplyMethod=immediate"
```

タイプ: 動的 (を設定すると変更がすぐに適用されるApplyMethod=immediate)

デフォルト値: -1 (PostgreSQL エンジンのデフォルト)

例

システムが一時ストレージを使いすぎている、または一時ファイルが正しく削除されていないと思われる場合は、このパラメータを有効にできます。ログ出力を調べると、一時ファイルを生成しているクエリやオペレーション、およびこれらのファイルがどのように使用されているかを確認できます。

一部のクエリまたはオペレーションでは、多数の一時ファイルが作成されるため、 を有効にするとシステム全体のパフォーマンスに影響するlog_temp_files可能性があります。

log_connections

log_connections パラメータは、データベースへの接続がログに記録されるかどうかを制御します。このパラメータを に設定するとon、ログには、クライアントの IP アドレス、ユーザー名、データベース名、接続日時など、データベースへの成功した各接続に関する情報が含まれます。

log_connections パラメータを使用して、データベースへの接続をモニタリングおよびトラブルシューティングできます。データベースに接続するユーザー、アプリケーション、ターミナル、ボット、接続元、頻度を確認できます。この情報は、接続関連の問題の特定と解決、または使用パターンの追跡に役立ちます。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループlog_connectionsについて変更します。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify log_connections on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_connections,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify log_connections on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_connections,ParameterValue=<new_value>,ApplyMethod=immediate"
```

タイプ: 動的 (を設定すると変更がすぐに適用されるApplyMethod=immediate)

デフォルト値: off (PostgreSQL エンジンのデフォルト)

例

このパラメータは、データベースへの接続が多すぎる、または接続頻度の高い特定のユーザーまたは IP アドレスがパフォーマンスに影響を与えている可能性がある場合に使用できます。log_connections パラメータを有効にして ログ出力を調べることで、すべての接続の数と詳細を確認できます。

このパラメータを有効にする前に、組織のポリシーを確認し、IP アドレスとユーザー名のログ記録によるセキュリティへの影響を考慮してください。

log_disconnections

log_disconnections パラメータは、データベースからの切断のログ記録を制御します。このパラメータを に設定するとon、クライアントの IP アドレス、ユーザー名、データベース名、切断日時など、各セッションの終了に関する情報を ログに記録します。

log_disconnections パラメータを使用して、データベースセッションの終了をモニタリングおよびトラブルシューティングできます。データベースから切断するユーザー、アプリケーション、ターミナル、ボット、タイミング、理由を確認できます。例えば、クラッシュや管理者主導の切断など、予期しない終了を確認できます。この情報は、切断に関連する問題の特定と解決、または使用パターンの追跡に役立ちます。

AWS CLI 構文

次のコマンドは、特定の DB パラメータグループlog_disconnectionsに対して変更されます。この変更は、 パラメータグループを使用するすべてのインスタンスまたはクラスターに適用されます。

```
# Modify log_disconnections on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_disconnections,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify log_disconnections on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_disconnections,ParameterValue=<new_value>,ApplyMethod=immediate"
```

タイプ: 動的 (を設定すると変更がすぐに適用されるApplyMethod=immediate)

デフォルト値: off (PostgreSQL エンジンのデフォルト)

例

データベースから切断しているユーザーが多すぎる、または特定のユーザーまたは IP アドレスが頻繁に切断されていると思われるlog_disconnections場合は、 を使用できます。log_disconnections パラメータを有効にして ログ出力を調べることで、切断前にエラーが発生したユーザー、タイミング、エラーが発生したかどうかなど、すべての切断の数と詳細を確認できます。

このパラメータを有効にする前に、組織のポリシーを確認し、IP アドレスとユーザー名のログ記録によるセキュリティへの影響を考慮してください。

ログ記録パラメータを使用したバインド変数のキャプチャ

PostgreSQL でバインド変数をキャプチャする一般的なユースケースは、SQL クエリをデバッグしてパフォーマンスを調整することです。バインド変数を使用すると、実行時にクエリにデータを渡すことができます。バインド変数をキャプチャすることで、クエリに渡された入力データを確認できます。これにより、データまたはクエリのパフォーマンスに関する問題を特定できます。バインド変数をキャプチャすると、入力データを監査し、潜在的なセキュリティリスクや悪意のあるアクティビティを検出するのにも役立ちます。

PostgreSQL のバインド変数をキャプチャするには、いくつかの方法があります。1 つの方法は、debug_print_parse および debug_print_rewrittenパラメータを有効にすることです。これにより、PostgreSQL は SQL ステートメントの解析済みバージョンと書き換え済みバージョンを、バインドされた変数とともにサーバーログに送信します。

- debug_print_parse: このパラメータを有効にすると、受信クエリの解析ツリーがサーバーログに出力されます。これは、クエリの構造とバインドされたパラメータの値を理解するのに役立ちます。
- debug_print_rewritten: このパラメータを有効にすると、書き換えられた形式の受信クエリがサーバーログに出力されます。これは、クエリプランナーがクエリを解釈する方法と、バインドされたパラメータの値を理解するのに役立ちます。

Amazon RDS と Aurora で 2 つの追加パラメータを使用して、PostgreSQL データベースのバインド変数をキャプチャできます。

- log_min_duration_statement: このパラメータは、ステートメントがログに記録されるまでの最小時間をミリ秒単位で設定します。ステートメントが指定された期間よりも時間がかかる場合、そのバインド値はログ出力に含まれます。
- log_statement: このパラメータは、ログに記録される SQL ステートメントを制御します。このパラメータを に設定するallか、バインドしてログにバインドされた値を含めます。ログ記録レベルを上げるとパフォーマンスに影響するため、トラブルシューティング後に変更を元に戻すことをお勧めします。

pg_stat_statements 拡張機能を使用することもできます。拡張機能は、クエリテキストやバインド値など、サーバーによって実行されるすべての SQL ステートメントのパフォーマンス統計を提供します。この拡張機能を使用すると、pgAdmin または同様のツールを使用してクエリのパフォーマンスをモニタリングおよび分析できます。

もう 1 つのオプションは、 pg_bind_parameter_status()関数を使用して準備済みステートメントからバインドされたパラメータの値を取得するか、 pg_get_parameter_status (paramname)関数を使用して特定のランタイムパラメータのステータスまたは値を取得することです。

さらに、pgBadger などのサードパーティーツールを使用して PostgreSQL ログを分析し、バインド変数やその他の情報を抽出してさらに分析できます。

レプリケーションパラメータのチューニング

PostgreSQL では、物理的なファイルベースのレプリケーションの代わりに論理レプリケーションを使用することで、ある PostgreSQL データベースから別の PostgreSQL データベースにデータ変更をレプリケートできます。論理レプリケーションは、ログ先行書き込み (WAL) を使用して変更をキャプチャし、選択したテーブルまたはデータベース全体のレプリケーションをサポートします。

Amazon RDS for PostgreSQL と Aurora PostgreSQL 互換はどちらも論理レプリケーションをサポートしているため、複数のソースからの読み取りおよび書き込みトラフィックを処理できる可用性とスケーラブルなデータベースアーキテクチャを設定できます。これらのサービスは、オープンソースの PostgreSQL 拡張機能である pglogical を使用して論理レプリケーションを実装します。

Aurora と Amazon RDS で論理レプリケーションを調整することは、最適なパフォーマンス、スケーラビリティ、可用性を実現する上で重要です。pglogical 拡張機能のパラメータを調整して、論理レプリケーションのパフォーマンスを管理できます。例えば、以下のことが可能です。

- ワーカープロセスの数を増やすか、メモリ割り当てを調整することで、レプリケーションのパフォーマンスを向上させます。
- ソースデータベースとレプリカデータベース間の同期頻度を調整することで、レプリケーションの遅延のリスクを軽減します。
- ワーカープロセスのメモリと CPU 割り当てを調整して、リソースの使用を最適化します。
- レプリケーションプロセスがソースデータベースのパフォーマンスに過度の影響を与えないことを確認します。

Aurora と Amazon RDS で次のパラメータを使用して、論理レプリケーションを制御および設定できます。

- `max_replication_slots` は、サーバーで作成できるレプリケーションスロットの最大数を設定します。レプリケーションスロットは、レプリカに WAL データを送信するためのレプリケーション接続の名前付き永続予約です。
- `max_wal_senders` は、同時に接続される WAL 送信者プロセスの最大数を設定します。WAL 送信者プロセスは、プライマリサーバーからレプリカに WAL をストリーミングするために使用されます。
- `wal_sender_timeout` は、WAL 送信者がレプリカからの応答を待ってから、あきらめて再接続するまでの最大時間をミリ秒単位で設定します。

- wal_receiver_timeout は、レプリカがプライマリデータベースからの WAL データを待機してからタイムアウトするまでの最大時間をミリ秒単位で設定します。
- log_replication_commandsを に設定するとon、レプリケーション関連の SQL ステートメントが実行されます。

rds.logical_replication パラメータを有効にすると (1 に設定) 、 wal_levelパラメータはに設定されます。つまりlogical、データベースに加えられたすべての変更は、読み取りとレプリカへの適用が可能な形式で WAL に書き込まれます。この設定は、論理レプリケーションを有効にするために必要です。この設定では、 SELECTステートメントのレプリケーションも可能です。

wal_level を に設定すると、WAL、つまりディスクに書き込まれるデータの量logicalが増え、システムのパフォーマンスに影響する可能性があります。論理レプリケーションを有効にするときは、使用可能なディスク容量とシステムパフォーマンスを考慮することをお勧めします。

例

バックアップとディザスタリカバリの目的で、プライマリデータベースからセカンダリデータベースにデータをレプリケートする場合。ただし、セカンダリデータベースには大量の読み取りオペレーションがあるため、データの整合性を損なうことなく、レプリケーションプロセスが可能な限り高速かつ効率的に行われるようにする必要があります。

Amazon RDS と Aurora の論理レプリケーションのデフォルト値は、パフォーマンスよりも一貫性を優先するため、このユースケースには適していない可能性があります。論理レプリケーション設定を最適化して速度と効率を向上させるには、パラメータを次のようにカスタマイズします。

- 将来の成長とレプリケーションの潜在的なニーズに対応するために、10 (Amazon RDS のデフォルト) または 20 (Aurora のデフォルト) max_replication_slotsから 30 に増やします。
- レプリケーションの需要に追いつくのに十分な WAL 送信者プロセスがあることを確認するために、10 (デフォルト) max_wal_sendersから 20 に増やします。
- アイドル状態の WAL 送信者プロセスをより迅速に終了できるように、30 秒 (デフォルト) wal_sender_timeoutから 15 秒に減らします。これにより、アクティブなレプリケーションのためにリソースが解放されます。
- アイドル状態の WAL レシーバープロセスをより迅速に終了できるように、30 秒 (デフォルト) wal_receiver_timeoutから 15 秒に減らします。これにより、アクティブなレプリケーションのためにリソースが解放されます。

- レプリケーションの需要に追いつくのに十分な論理レプリケーションワーカープロセスがあることを確認するために、4 (デフォルト) max_logical_replication_workersから 8 に増やします。

これらの最適化により、データの整合性とセキュリティを維持しながら、より高速で効率的なデータレプリケーションが可能になります。

例えば、災害が発生し、プライマリデータベースが使用できなくなった場合、セカンダリデータベースには、最適化されたレプリケーションプロセスにより、すでに最新のデータが利用可能になります。これにより、ビジネスオペレーションは中断することなく重要なサービスを提供し続けることができます。

ベストプラクティス

巨大なワークロードで論理レプリケーションを調整するのは、データセットのサイズ、レプリケートされるテーブルの数、レプリカの数、使用可能なリソースなど、さまざまな要因に依存する複雑なタスクです。巨大なワークロードで論理レプリケーションを調整するための一般的なヒントをいくつか紹介します。

- レプリケーションラグをモニタリングします。レプリケーションラグは、プライマリサーバーとスタンバイサーバー間の時間差です。レプリケーションの遅延をモニタリングすることで、潜在的なボトルネックを特定し、レプリケーションのパフォーマンスを向上させるためのアクションを実行できます。pg_current_wal_lsn() 関数を使用して、現在のレプリケーションラグを確認できます。
- WAL 設定を調整します。pg_logical 拡張機能は WAL を使用して、プライマリサーバーからスタンバイサーバーに変更を送信します。WAL 設定が適切に調整されていない場合、レプリケーションが遅くなり、信頼性が低下する可能性があります。ワークロードに応じて、 パラメータmax_wal_sendersと max_replication_slotsパラメータを適切な値に設定してください。
- インデックス作成戦略がある。プライマリサーバーに適切なインデックスを設定すると、論理レプリケーションのパフォーマンスが向上し、プライマリサーバーの I/O が減少し、システムの負荷が軽減されます。
- 並列レプリケーションを使用します。並列レプリケーションを使用すると、複数の並列ワーカープロセスでデータをレプリケートできるため、レプリケーション速度が向上します。この機能は PostgreSQL 12 以降で使用できます。

次のステップ

Amazon RDS for PostgreSQL または Aurora PostgreSQL 互換データベースのメモリ、レプリケーション、autovacuum、ログ記録パラメータを最適化したら、データベースのパフォーマンスをさらに向上させるため、次のステップを検討してください。

- データベースをモニタリングします。組み込みのモニタリングツールまたはサードパーティーソリューションを使用して、データベースのパフォーマンスを経時的に追跡します。CPU 使用率、ディスク I/O、メモリ使用率、クエリランタイムなどの主要なパフォーマンスメトリクスをモニタリングして、潜在的なボトルネックと改善が必要な領域を特定します。
- パラメータを継続的に調整します。ワークロードが進化するにつれて、データベースパラメータのモニタリングと調整を継続し、最適なパフォーマンスを確保します。システムログ、エラーメッセージ、パフォーマンスメトリクスを定期的にチェックして、新しい調整の機会を特定します。
- キャッシュを実装します。キャッシュを使用して、データベースにヒットするクエリの数を減らします。Memcached や Redis などのツールを使用してアプリケーションレベルでキャッシュを実装することも、Amazon を使用してデータベースにインメモリキャッシュ ElastiCache を提供することもできます。
- クエリを最適化します。設計の悪いクエリは、データベース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。`EXPLAIN` およびその他のクエリ調整ツールを使用して、スロークエリを特定し、最適化して、不要なクエリを排除します。

これらのガイドラインに従うことで、Aurora または Amazon RDS for PostgreSQL データベースのパフォーマンスを最適化し、データベースのパフォーマンスの向上、信頼性の向上、ダウンタイムの短縮、セキュリティの向上、コスト削減により、アプリケーションのニーズを確実に満たすことができます。ワークロードに合わせて設定パラメータを最適化することで、データベースが効率的に実行され、リソースを効果的に使用できるようになり、パフォーマンスが向上し、アプリケーションの応答性が向上します。さらに、パラメータを適切に設定することで、エラーや脆弱性の可能性が軽減され、信頼性が向上し、セキュリティが向上します。これにより、メンテナンスとダウンタイムの短縮、全体的なユーザーエクスペリエンスと満足度の向上の観点からコスト削減につながります。

リソース

- Amazon Aurora PostgreSQL パラメータ、パート 1: メモリとクエリプランの管理 (AWS ブログ記事)
- Amazon Aurora PostgreSQL パラメータ、パート 2: レプリケーション、セキュリティ、ログ記録 (AWS ブログ記事)
- Amazon Aurora PostgreSQL パラメータ、パート 3: オプティマイザパラメータ (AWS ブログ記事)
- Amazon Aurora PostgreSQL パラメータ、パート 4: ANSI 互換性オプション (AWS ブログ記事)
- Amazon Aurora PostgreSQL の使用 (AWS ドキュメント)
- Amazon RDS for PostgreSQL の使用 (AWS ドキュメント)
- Amazon RDS での Performance Insights による DB ロードのモニタリング (AWS ドキュメント)
- Amazon CloudWatch メトリクスの使用 (AWS ドキュメント)
- pg_stats_statements PostgreSQL ドキュメント)

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
メモリパラメータと autovacuum パラメータに関する情報を更新しました	random_page_cost パラメータの説明を更新しました。メモリパラメータと autovacuum パラメータのデフォルト値に欠落している単位を追加しました。max_connections パラメータの AWS CLI 構文を更新しました。	2024 年 2 月 27 日
に関する情報を更新しました `autovacuum`	autovacuum のデフォルト設定を修正しました (有効) 。	2023 年 12 月 27 日
に関する情報を更新しました `max_connections`	max_connections セクションを更新し、このパラメータの調整に関する新しいガイダンスを追加しました。	2023 年 11 月 15 日
初版発行	—	2023 年 10 月 31 日

AWS 規範的ガイダンスの用語集

以下は、 AWS 規範的ガイダンスが提供する戦略、ガイド、パターンで一般的に使用される用語です。エントリを提案するには、用語集の最後のフィードバックの提供リンクを使用します。

数字

7 Rs

アプリケーションをクラウドに移行するための 7 つの一般的な移行戦略。これらの戦略は、ガートナーが 2011 年に特定した 5 Rs に基づいて構築され、以下で構成されています。

- リファクタリング/アーキテクチャの再設計 — クラウドネイティブ特徴を最大限に活用して、俊敏性、パフォーマンス、スケーラビリティを向上させ、アプリケーションを移動させ、アーキテクチャを変更します。これには、通常、オペレーティングシステムとデータベースの移植が含まれます。例: オンプレミスの Oracle データベースを Amazon Aurora PostgreSQL 互換エディションに移行します。
- リプラットフォーム (リフトアンドリシェイプ) — アプリケーションをクラウドに移行し、クラウド機能を活用するための最適化レベルを導入します。例: オンプレミスの Oracle データベースを の Oracle 用 Amazon Relational Database Service (Amazon RDS) に移行します AWS クラウド。
- 再購入 (ドロップアンドショップ) — 通常、従来のライセンスから SaaS モデルに移行して、別の製品に切り替えます。例: カスタマーリレーションシップ管理 (CRM) システムを Salesforce.com に移行します。
- リホスト (リフトアンドシフト) — クラウド機能を活用するための変更を加えずに、アプリケーションをクラウドに移行します。例: オンプレミスの Oracle データベースを の EC2 インスタンス上の Oracle に移行します AWS クラウド。
- 再配置 (ハイパーバイザーレベルのリフトアンドシフト) – 新しいハードウェアを購入したり、アプリケーションを書き換えたり、既存の運用を変更したりすることなく、インフラストラクチャをクラウドに移行できます。サーバーをオンプレミスプラットフォームから同じプラットフォームのクラウドサービスに移行します。例: Microsoft Hyper-Vアプリケーションを に移行します AWS。
- 保持 (再アクセス) — アプリケーションをお客様のソース環境で保持します。これには、主要なリファクタリングを必要とするアプリケーションや、お客様がその作業を後日まで延期したいアプリケーション、およびそれらを移行するためのビジネス上の正当性がないため、お客様が保持するレガシーアプリケーションなどがあります。

- 使用停止 — お客様のソース環境で不要になったアプリケーションを停止または削除します。

A

ABAC

「属性ベースのアクセスコントロール」を参照してください。

抽象化されたサービス

「 マネージドサービス」を参照してください。

ACID

「原子性、一貫性、分離性、耐久性」を参照してください。

アクティブ - アクティブ移行

(双方向レプリケーションツールまたは二重書き込み操作を使用して) ソースデータベースとターゲットデータベースを同期させ、移行中に両方のデータベースが接続アプリケーションからのトランザクションを処理するデータベース移行方法。この方法では、1 回限りのカットオーバーの必要がなく、管理された小規模なバッチで移行できます。アクティブ/パッシブ移行よりも柔軟ですが、より多くの作業が必要です。

アクティブ - パッシブ移行

ソースデータベースとターゲットデータベースを同期させながら、データがターゲットデータベースにレプリケートされている間、接続しているアプリケーションからのトランザクションをソースデータベースのみで処理するデータベース移行の方法。移行中、ターゲットデータベースはトランザクションを受け付けません。

集計関数

行のグループを操作し、グループの単一の戻り値を計算する SQL 関数。集計関数の例としては、 SUMや などがありますMAX。

AI

「人工知能」を参照してください。

AIOps

「人工知能オペレーション」を参照してください。

匿名化

データセット内の個人情報を完全に削除するプロセス。匿名化は個人のプライバシー保護に役立ちます。匿名化されたデータは、もはや個人データとは見なされません。

アンチパターン

繰り返し起こる問題に対して頻繁に用いられる解決策で、その解決策が逆効果であったり、効果がなかったり、代替案よりも効果が低かったりするもの。

アプリケーションコントロール

マルウェアからシステムを保護するために、承認されたアプリケーションのみを使用できるようにするセキュリティアプローチ。

アプリケーションポートフォリオ

アプリケーションの構築と維持にかかるコスト、およびそのビジネス価値を含む、組織が使用する各アプリケーションに関する詳細情報の集まり。この情報は、ポートフォリオの検出と分析プロセス の需要要素であり、移行、モダナイズ、最適化するアプリケーションを特定し、優先順位を付けるのに役立ちます。

人工知能 (AI)

コンピューティングテクノロジーを使用し、学習、問題の解決、パターンの認識など、通常は人間に関連づけられる認知機能の実行に特化したコンピュータサイエンスの分野。詳細については、「人工知能 (AI) とは何ですか?」を参照してください。

AI オペレーション (AIOps)

機械学習技術を使用して運用上の問題を解決し、運用上のインシデントと人の介入を減らし、サービス品質を向上させるプロセス。 AWS 移行戦略での AIOps の使用方法については、オペレーション統合ガイド を参照してください。

非対称暗号化

暗号化用のパブリックキーと復号用のプライベートキーから成る 1 組のキーを使用した、暗号化のアルゴリズム。パブリックキーは復号には使用されないため共有しても問題ありませんが、プライベートキーの利用は厳しく制限する必要があります。

原子性、一貫性、分離性、耐久性 (ACID)

エラー、停電、その他の問題が発生した場合でも、データベースのデータ有効性と運用上の信頼性を保証する一連のソフトウェアプロパティ。

属性ベースのアクセス制御 (ABAC)

部署、役職、チーム名など、ユーザーの属性に基づいてアクセス許可をきめ細かく設定する方法。詳細については、 AWS Identity and Access Management (IAM) ドキュメントの「 の ABAC AWS」を参照してください。

信頼できるデータソース

最も信頼性のある情報源とされるデータのプライマリーバージョンを保存する場所。匿名化、編集、仮名化など、データを処理または変更する目的で、信頼できるデータソースから他の場所にデータをコピーすることができます。

アベイラビリティーゾーン

他のアベイラビリティーゾーンの障害から AWS リージョン 隔離され、同じリージョン内の他のアベイラビリティーゾーンへの低コストで低レイテンシーのネットワーク接続を提供する 内の別の場所。

AWS クラウド導入フレームワーク (AWS CAF)

組織がクラウドに正常に移行 AWS するための効率的で効果的な計画を立てるのに役立つ、 からのガイドラインとベストプラクティスのフレームワーク。 AWS CAF は、ビジネス、人材、ガバナンス、プラットフォーム、セキュリティ、運用という 6 つの重点分野にガイダンスを編成します。ビジネス、人材、ガバナンスの観点では、ビジネススキルとプロセスに重点を置き、プラットフォーム、セキュリティ、オペレーションの視点は技術的なスキルとプロセスに焦点を当てています。例えば、人材の観点では、人事 (HR)、人材派遣機能、および人材管理を扱うステークホルダーを対象としています。この観点から、 AWS CAF は、組織がクラウド導入を成功させるための準備に役立つ人材開発、トレーニング、コミュニケーションに関するガイダンスを提供します。詳細については、AWS CAF ウェブサイト と AWS CAF のホワイトペーパー を参照してください。

AWS ワークロード認定フレームワーク (AWS WQF)

データベース移行ワークロードを評価し、移行戦略を推奨し、作業見積もりを提供するツール。AWS WQF は AWS Schema Conversion Tool （ AWS SCT) に含まれています。データベーススキーマとコードオブジェクト、アプリケーションコード、依存関係、およびパフォーマンス特性を分析し、評価レポートを提供します。

B

不正なボット

個人または組織に混乱や損害を与えることを目的としたボット。

BCP

「事業継続計画」を参照してください。

動作グラフ

リソースの動作とインタラクションを経時的に示した、一元的なインタラクティブビュー。Amazon Detective の動作グラフを使用すると、失敗したログオンの試行、不審な API 呼び出し、その他同様のアクションを調べることができます。詳細については、Detective ドキュメントのData in a behavior graphを参照してください。

ビッグエンディアンシステム

最上位バイトを最初に格納するシステム。エンディアンネス も参照してください。

二項分類

バイナリ結果 (2 つの可能なクラスのうちの 1 つ) を予測するプロセス。例えば、お客様の機械学習モデルで「この E メールはスパムですか、それともスパムではありませんか」などの問題を予測する必要があるかもしれません。または「この製品は書籍ですか、車ですか」などの問題を予測する必要があるかもしれません。

ブルームフィルター

要素がセットのメンバーであるかどうかをテストするために使用される、確率的でメモリ効率の高いデータ構造。

ブルー/グリーンデプロイ

2 つの異なる同一の環境を作成するデプロイ戦略。現在のアプリケーションバージョンは 1 つの環境 (青) で実行し、新しいアプリケーションバージョンは他の環境 (緑) で実行します。この戦略は、影響を最小限に抑えながら迅速にロールバックするのに役立ちます。

ボット

インターネット経由で自動タスクを実行し、人間のアクティビティやインタラクションをシミュレートするソフトウェアアプリケーション。インターネット上の情報のインデックスを作成するウェブクローラーなど、一部のボットは有用または有益です。悪質なボット と呼ばれる他のボッ

トの中には、個人や組織に混乱を与えたり、損害を与えたりすることを意図しているものがあります。

ボットネット

マルウェアに感染し、ボットのヘルダーまたはボットオペレーター と呼ばれる、単一関係者の管理下にあるボットのネットワーク。ボットは、ボットとその影響をスケールするための最もよく知られているメカニズムです。

ブランチ

コードリポジトリに含まれる領域。リポジトリに最初に作成するブランチは、メインブランチといいます。既存のブランチから新しいブランチを作成し、その新しいブランチで機能を開発したり、バグを修正したりできます。機能を構築するために作成するブランチは、通常、機能ブランチと呼ばれます。機能をリリースする準備ができたら、機能ブランチをメインブランチに統合します。詳細については、「ブランチについて (GitHub ドキュメント) 」を参照してください。

ブレークグラスアクセス

例外的な状況や承認されたプロセスを通じて、ユーザーが通常アクセス許可を持たない AWS アカウント にすばやくアクセスできるようにします。詳細については、 Well-Architected ガイダンスの「ブレークグラスプロシージャの実装」インジケータ AWS を参照してください。

ブラウンフィールド戦略

環境の既存インフラストラクチャ。システムアーキテクチャにブラウンフィールド戦略を導入する場合、現在のシステムとインフラストラクチャの制約に基づいてアーキテクチャを設計します。既存のインフラストラクチャを拡張している場合は、ブラウンフィールド戦略とグリーンフィールド戦略を融合させることもできます。

バッファキャッシュ

アクセス頻度が最も高いデータが保存されるメモリ領域。

ビジネス能力

価値を生み出すためにビジネスが行うこと (営業、カスタマーサービス、マーケティングなど)。マイクロサービスのアーキテクチャと開発の決定は、ビジネス能力によって推進できます。詳細については、ホワイトペーパー AWSでのコンテナ化されたマイクロサービスの実行 の ビジネス機能を中心に組織化 セクションを参照してください。

ビジネス継続性計画 (BCP)

大規模移行など、中断を伴うイベントが運用に与える潜在的な影響に対処し、ビジネスを迅速に再開できるようにする計画。

C

CAF

AWS 「クラウド導入フレームワーク」を参照してください。

Canary デプロイ

エンドユーザーへのバージョンの低速かつ増分的なリリース。確信できたら、新しいバージョンをデプロイし、現在のバージョン全体を置き換えます。

CCoE

「Cloud Center of Excellence」を参照してください。

CDC

「データキャプチャの変更」を参照してください。

変更データキャプチャ (CDC)

データソース (データベーステーブルなど) の変更を追跡し、その変更に関するメタデータを記録するプロセス。CDC は、ターゲットシステムでの変更を監査またはレプリケートして同期を維持するなど、さまざまな目的に使用できます。

カオスエンジニアリング

障害や破壊的なイベントを意図的に導入して、システムの耐障害性をテストします。AWS Fault Injection Service （ AWS FIS ） を使用して、 AWS ワークロードに負荷をかけ、その応答を評価する実験を実行できます。

CI/CD

「継続的インテグレーションと継続的デリバリー」を参照してください。

分類

予測を生成するのに役立つ分類プロセス。分類問題の機械学習モデルは、離散値を予測します。離散値は、常に互いに区別されます。例えば、モデルがイメージ内に車があるかどうかを評価する必要がある場合があります。

クライアント側の暗号化

ターゲットがデータ AWS サービス を受信する前に、ローカルでデータを暗号化します。

Cloud Center of Excellence (CCoE)

クラウドのベストプラクティスの作成、リソースの移動、移行のタイムラインの確立、大規模変革を通じて組織をリードするなど、組織全体のクラウド導入の取り組みを推進する学際的なチーム。詳細については、 AWS クラウド エンタープライズ戦略ブログのCCoE の投稿」を参照してください。

クラウドコンピューティング

リモートデータストレージと IoT デバイス管理に通常使用されるクラウドテクノロジー。クラウドコンピューティングは、一般的にエッジコンピューティングテクノロジーに接続されています。

クラウド運用モデル

IT 組織において、1 つ以上のクラウド環境を構築、成熟、最適化するために使用される運用モデル。詳細については、「クラウド運用モデルの構築」 を参照してください。

導入のクラウドステージ

組織が に移行するときに通常実行する 4 つのフェーズ AWS クラウド :

- プロジェクト — 概念実証と学習を目的として、クラウド関連のプロジェクトをいくつか実行する
- 基礎固め — お客様のクラウドの導入を拡大するための基礎的な投資 (ランディングゾーンの作成、CCoE の定義、運用モデルの確立など)
- 移行 — 個々のアプリケーションの移行
- 再発明 — 製品とサービスの最適化、クラウドでのイノベーション

これらのステージは、 AWS クラウド エンタープライズ戦略ブログのブログ記事「クラウドファーストへのジャーニー」と「導入のステージ」で Stephen Orban によって定義されました。移行戦略とどのように関連しているかについては、 AWS 「移行準備ガイド」を参照してください。

CMDB

「設定管理データベース」を参照してください。

コードリポジトリ

ソースコードやその他の資産 (ドキュメント、サンプル、スクリプトなど) が保存され、バージョン管理プロセスを通じて更新される場所。一般的なクラウドリポジトリには、 GitHub または が含まれます AWS CodeCommit。コードの各バージョンはブランチと呼ばれます。マイクロサー

ビスの構造では、各リポジトリは 1 つの機能専用です。1 つの CI/CD パイプラインで複数のリポジトリを使用できます。

コールドキャッシュ

空である、または、かなり空きがある、もしくは、古いデータや無関係なデータが含まれているバッファキャッシュ。データベースインスタンスはメインメモリまたはディスクから読み取る必要があり、バッファキャッシュから読み取るよりも時間がかかるため、パフォーマンスに影響します。

コールドデータ

めったにアクセスされず、通常は過去のデータです。この種類のデータをクエリする場合、通常は低速なクエリでも問題ありません。このデータを低パフォーマンスで安価なストレージ階層またはクラスに移動すると、コストを削減することができます。

コンピュータビジョン (CV)

機械学習を使用してデジタルイメージやビデオなどのビジュアル形式から情報を分析および抽出する AI の分野。例えば、 AWS Panorama はオンプレミスのカメラネットワークに CV を追加するデバイスを提供し、Amazon SageMaker は CV の画像処理アルゴリズムを提供します。

設定ドリフト

ワークロードの場合、設定は想定した状態から変化します。これにより、ワークロードが非準拠になる可能性があり、通常は段階的かつ意図的ではありません。

構成管理データベース (CMDB)

データベースとその IT 環境 (ハードウェアとソフトウェアの両方のコンポーネントとその設定を含む) に関する情報を保存、管理するリポジトリ。通常、CMDB のデータは、移行のポートフォリオの検出と分析の段階で使用します。

コンフォーマンスパック

コンプライアンスチェックとセキュリティチェックをカスタマイズするためにアセンブルできる AWS Config ルールと修復アクションのコレクション。YAML テンプレートを使用して、コンフォーマンスパックを AWS アカウント および リージョンで単一のエンティティとしてデプロイすることも、組織全体にデプロイすることもできます。詳細については、 AWS Config ドキュメントの「コンフォーマンスパック」を参照してください。

継続的インテグレーションと継続的デリバリー (CI/CD)

ソフトウェアリリースプロセスのソース、ビルド、テスト、ステージング、本番の各ステージを自動化するプロセス。CI/CD は一般的にパイプラインと呼ばれます。プロセスの自動化、生産性

の向上、コード品質の向上、配信の加速化を可能にします。詳細については、「継続的デリバリーの利点」を参照してください。CD は継続的デプロイ (Continuous Deployment) の略語でもあります。詳細については「継続的デリバリーと継続的なデプロイ」を参照してください。

CV

「コンピュータビジョン」を参照してください。

D

保管中のデータ

ストレージ内にあるデータなど、常に自社のネットワーク内にあるデータ。

データ分類

ネットワーク内のデータを重要度と機密性に基づいて識別、分類するプロセス。データに適した保護および保持のコントロールを判断する際に役立つため、あらゆるサイバーセキュリティのリスク管理戦略において重要な要素です。データ分類は、 AWS Well-Architected フレームワークのセキュリティの柱のコンポーネントです。詳細については、データ分類を参照してください。

データドリフト

実稼働データと ML モデルのトレーニングに使用されたデータとの間に有意な差異が生じたり、入力データが時間の経過と共に有意に変化したりすることです。データドリフトは、ML モデル予測の全体的な品質、精度、公平性を低下させる可能性があります。

転送中のデータ

ネットワーク内 (ネットワークリソース間など) を活発に移動するデータ。

データメッシュ

一元化された管理とガバナンスにより、分散型の分散型データ所有権を提供するアーキテクチャフレームワーク。

データ最小化

厳密に必要なデータのみを収集し、処理するという原則。でデータ最小化を実践 AWS クラウドすることで、プライバシーリスク、コスト、分析のカーボンフットプリントを削減できます。

データ境界

AWS 環境内の一連の予防ガードレール。信頼できる ID のみが、期待されるネットワークから信頼できるリソースにアクセスしていることを確認できます。詳細については、「 でのデータ境界の構築 AWS」を参照してください。

データの前処理

raw データをお客様の機械学習モデルで簡単に解析できる形式に変換すること。データの前処理とは、特定の列または行を削除して、欠落している、矛盾している、または重複する値に対処することを意味します。

データ出所

データの生成、送信、保存の方法など、データのライフサイクル全体を通じてデータの出所と履歴を追跡するプロセス。

データ件名

データを収集、処理している個人。

データウェアハウス

分析などのビジネスインテリジェンスをサポートするデータ管理システム。データウェアハウスには通常、大量の履歴データが含まれており、クエリや分析によく使用されます。

データベース定義言語 (DDL)

データベース内のテーブルやオブジェクトの構造を作成または変更するためのステートメントまたはコマンド。

データベース操作言語 (DML)

データベース内の情報を変更 (挿入、更新、削除) するためのステートメントまたはコマンド。

DDL

「データベース定義言語」を参照してください。

ディープアンサンブル

予測のために複数の深層学習モデルを組み合わせる。ディープアンサンブルを使用して、より正確な予測を取得したり、予測の不確実性を推定したりできます。

ディープラーニング

人工ニューラルネットワークの複数層を使用して、入力データと対象のターゲット変数の間のマッピングを識別する機械学習サブフィールド。

defense-in-depth

一連のセキュリティメカニズムとコントロールをコンピュータネットワーク全体に層状に重ねて、ネットワークとその内部にあるデータの機密性、整合性、可用性を保護する情報セキュリティの手法。この戦略を に採用するときは AWS、 AWS Organizations 構造の異なるレイヤーに複数のコントロールを追加して、リソースの安全性を確保します。例えば、 defense-in-depth アプローチでは、多要素認証、ネットワークセグメンテーション、暗号化を組み合わせることができます。

委任管理者

では AWS Organizations、互換性のあるサービスが AWS メンバーアカウントを登録して組織のアカウントを管理し、そのサービスのアクセス許可を管理できます。このアカウントを、そのサービスの委任管理者と呼びます。詳細、および互換性のあるサービスの一覧は、 AWS Organizations ドキュメントのAWS Organizationsで使用できるサービスを参照してください。

デプロイメント

アプリケーション、新機能、コードの修正をターゲットの環境で利用できるようにするプロセス。デプロイでは、コードベースに変更を施した後、アプリケーションの環境でそのコードベースを構築して実行します。

開発環境

「環境」を参照してください。

検出管理

イベントが発生したときに、検出、ログ記録、警告を行うように設計されたセキュリティコントロール。これらのコントロールは副次的な防衛手段であり、実行中の予防的コントロールをすり抜けたセキュリティイベントをユーザーに警告します。詳細については、Implementing security controls on AWSのDetective controlsを参照してください。

開発バリューストリームマッピング (DVSM)

ソフトウェア開発ライフサイクルのスピードと品質に悪影響を及ぼす制約を特定し、優先順位を付けるために使用されるプロセス。DVSM は、もともとリーンマニュファクチャリング・プラクティスのために設計されたバリューストリームマッピング・プロセスを拡張したものです。ソフトウェア開発プロセスを通じて価値を創造し、動かすために必要なステップとチームに焦点を当てています。

デジタルツイン

建物、工場、産業機器、生産ラインなど、現実世界のシステムを仮想的に表現したものです。デジタルツインは、予知保全、リモートモニタリング、生産最適化をサポートします。

ディメンションテーブル

スタースキーマ では、ファクトテーブル内の量的データに関するデータ属性を含む小さなテーブル。ディメンションテーブル属性は通常、テキストフィールドまたはテキストのように動作する離散数値です。これらの属性は、クエリの制約、フィルタリング、結果セットのラベル付けに一般的に使用されます。

ディザスタ

ワークロードまたはシステムが、導入されている主要な場所でのビジネス目標の達成を妨げるイベント。これらのイベントは、自然災害、技術的障害、または意図しない設定ミスやマルウェア攻撃などの人間の行動の結果である場合があります。

ディザスタリカバリ (DR)

災害によるダウンタイムとデータ損失を最小限に抑えるために使用する戦略とプロセス。詳細については、 AWS Well-Architected フレームワークの「 でのワークロードのディザスタリカバリ AWS: クラウドでのリカバリ」を参照してください。

DML

「データベース操作言語」を参照してください。

ドメイン駆動型設計

各コンポーネントが提供している変化を続けるドメイン、またはコアビジネス目標にコンポーネントを接続して、複雑なソフトウェアシステムを開発するアプローチ。この概念は、エリック・エヴァンスの著書、Domain-Driven Design: Tackling Complexity in the Heart of Software (ドメイン駆動設計:ソフトウェアの中心における複雑さへの取り組み) で紹介されています (ボストン: Addison-Wesley Professional、2003)。strangler fig パターンでドメイン駆動型設計を使用する方法の詳細については、コンテナと Amazon API Gateway を使用して、従来の Microsoft ASP.NET (ASMX) ウェブサービスを段階的にモダナイズ を参照してください。

DR

「ディザスタリカバリ」を参照してください。

ドリフト検出

ベースライン設定からの偏差の追跡。例えば、 AWS CloudFormation を使用してシステムリソース のドリフトを検出したり、 を使用して AWS Control Tower ガバナンス要件への準拠に影響を与える可能性のあるランディングゾーンの変更を検出したりできます。

DVSM

「開発値ストリームマッピング」を参照してください。

E

EDA

「探索的データ分析」を参照してください。

エッジコンピューティング

IoT ネットワークのエッジにあるスマートデバイスの計算能力を高めるテクノロジー。クラウドコンピューティング と比較すると、エッジコンピューティングは通信レイテンシーを短縮し、応答時間を短縮できます。

暗号化

人間が読み取り可能なプレーンテキストデータを暗号文に変換するコンピューティングプロセス。

暗号化キー

暗号化アルゴリズムが生成した、ランダム化されたビットからなる暗号文字列。キーの長さは決まっておらず、各キーは予測できないように、一意になるように設計されています。

エンディアン

コンピュータメモリにバイトが格納される順序。ビッグエンディアンシステムでは、最上位バイトが最初に格納されます。リトルエンディアンシステムでは、最下位バイトが最初に格納されます。

エンドポイント

「サービスエンドポイント」を参照してください。

エンドポイントサービス

仮想プライベートクラウド (VPC) 内でホストして、他のユーザーと共有できるサービス。を使用してエンドポイントサービスを作成し AWS PrivateLink 、他の AWS アカウント または AWS Identity and Access Management (IAM) プリンシパルにアクセス許可を付与できます。これらのアカウントまたはプリンシパルは、インターフェイス VPC エンドポイントを作成することで、エンドポイントサービスにプライベートに接続できます。詳細については、Amazon Virtual Private Cloud (Amazon VPC) ドキュメントの「エンドポイントサービスを作成する」を参照してください。

エンタープライズリソースプランニング (ERP)

エンタープライズの主要なビジネスプロセス (アカウンティング、MES、プロジェクト管理など) を自動化および管理するシステム。

エンベロープ暗号化

暗号化キーを、別の暗号化キーを使用して暗号化するプロセス。詳細については、 AWS Key Management Service （ AWS KMS) ドキュメントの「エンベロープ暗号化」を参照してください。

環境

実行中のアプリケーションのインスタンス。クラウドコンピューティングにおける一般的な環境の種類は以下のとおりです。

- 開発環境 — アプリケーションのメンテナンスを担当するコアチームのみが使用できる、実行中のアプリケーションのインスタンス。開発環境は、上位の環境に昇格させる変更をテストするときに使用します。このタイプの環境は、テスト環境と呼ばれることもあります。
- 下位環境 — 初期ビルドやテストに使用される環境など、アプリケーションのすべての開発環境。
- 本番環境 — エンドユーザーがアクセスできる、実行中のアプリケーションのインスタンス。CI/CD パイプラインでは、本番環境が最後のデプロイ環境になります。
- 上位環境 — コア開発チーム以外のユーザーがアクセスできるすべての環境。これには、本番環境、本番前環境、ユーザー承認テスト環境などが含まれます。

エピック

アジャイル方法論で、お客様の作業の整理と優先順位付けに役立つ機能カテゴリ。エピックでは、要件と実装タスクの概要についてハイレベルな説明を提供します。例えば、 AWS CAF セキュリティエピックには、ID とアクセスの管理、検出コントロール、インフラストラクチャセキュリティ、データ保護、インシデント対応が含まれます。 AWS 移行戦略のエピックの詳細については、プログラム実装ガイド を参照してください。

ERP

「エンタープライズリソース計画」を参照してください。

探索的データ分析 (EDA)

データセットを分析してその主な特性を理解するプロセス。お客様は、データを収集または集計してから、パターンの検出、異常の検出、および前提条件のチェックのための初期調査を実行します。EDA は、統計の概要を計算し、データの可視化を作成することによって実行されます。

F

ファクトテーブル

スタースキーマ の中央テーブル。事業運営に関する定量的データを保存します。通常、ファクトテーブルには、メジャーを含む列とディメンションテーブルへの外部キーを含む列の 2 種類の列が含まれます。

フェイルファスト

開発ライフサイクルを短縮するために頻繁で段階的なテストを使用する哲学。これはアジャイルアプローチの重要な部分です。

障害分離境界

では AWS クラウド、障害の影響を制限し、ワークロードの耐障害性を向上させるアベイラビリティーゾーン AWS リージョン、コントロールプレーン、データプレーンなどの境界です。詳細については、AWS 「障害分離境界」を参照してください。

機能ブランチ

「ブランチ」を参照してください。

特徴量

お客様が予測に使用する入力データ。例えば、製造コンテキストでは、特徴量は製造ラインから定期的にキャプチャされるイメージの可能性もあります。

特徴量重要度

モデルの予測に対する特徴量の重要性。これは通常、Shapley Additive Deskonations (SHAP) や積分勾配など、さまざまな手法で計算できる数値スコアで表されます。詳細については、「 を使用した機械学習モデルの解釈可能性：AWS」を参照してください。

機能変換

追加のソースによるデータのエンリッチ化、値のスケーリング、単一のデータフィールドからの複数の情報セットの抽出など、機械学習プロセスのデータを最適化すること。これにより、機械学習モデルはデータの恩恵を受けることができます。例えば、「2021-05-27 00:15:37」の日付を「2021 年」、「5 月」、「木」、「15」に分解すると、学習アルゴリズムがさまざまなデータコンポーネントに関連する微妙に異なるパターンを学習するのに役立ちます。

FGAC

「きめ細かなアクセスコントロール」を参照してください。

きめ細かなアクセス制御 (FGAC)

複数の条件を使用してアクセス要求を許可または拒否すること。

フラッシュカット移行

段階的なアプローチを使用するのではなく、変更データキャプチャによる継続的なデータレプリケーションを使用して、可能な限り短時間でデータを移行するデータベース移行方法。目的はダウンタイムを最小限に抑えることです。

G

ジオブロッキング

「地理的制限」を参照してください。

地理的制限 (ジオブロッキング)

Amazon では CloudFront、特定の国のユーザーがコンテンツディストリビューションにアクセスできないようにするオプションです。アクセスを許可する国と禁止する国は、許可リストまたは禁止リストを使って指定します。詳細については、 CloudFront ドキュメントの「コンテンツの地理的ディストリビューションの制限」を参照してください。

Gitflow ワークフロー

下位環境と上位環境が、ソースコードリポジトリでそれぞれ異なるブランチを使用する方法。Gitflow ワークフローはレガシーと見なされ、トランクベースのワークフローはモダンで推奨されるアプローチです。

グリーンフィールド戦略

新しい環境に既存のインフラストラクチャが存在しないこと。システムアーキテクチャにグリーンフィールド戦略を導入する場合、既存のインフラストラクチャ (別名ブラウンフィールド) との互換性の制約を受けることなく、あらゆる新しいテクノロジーを選択できます。既存のインフラストラクチャを拡張している場合は、ブラウンフィールド戦略とグリーンフィールド戦略を融合させることもできます。

ガードレール

組織単位 (OU) 全般のリソース、ポリシー、コンプライアンスを管理するのに役立つ概略的なルール。予防ガードレールは、コンプライアンス基準に一致するようにポリシーを実施します。これらは、サービスコントロールポリシーと IAM アクセス許可の境界を使用して実装

されます。検出ガードレールは、ポリシー違反やコンプライアンス上の問題を検出し、修復のためのアラートを発信します。これらは、 AWS Config、、Amazon AWS Security Hub、GuardDuty、Amazon Inspector AWS Trusted Advisor、およびカスタム AWS Lambda チェックを使用して実装されます。

H

HA

「高可用性」を参照してください。

異種混在データベースの移行

別のデータベースエンジンを使用するターゲットデータベースへお客様の出典データベースの移行 (例えば、Oracle から Amazon Aurora)。異種間移行は通常、アーキテクチャの再設計作業の一部であり、スキーマの変換は複雑なタスクになる可能性があります。AWS は、スキーマの変換に役立つ AWS SCTを提供します。

ハイアベイラビリティ (HA)

課題や災害が発生した場合に、介入なしにワークロードを継続的に運用できること。HA システムは、自動的にフェイルオーバーし、一貫して高品質のパフォーマンスを提供し、パフォーマンスへの影響を最小限に抑えながらさまざまな負荷や障害を処理するように設計されています。

ヒストリアンのモダナイゼーション

製造業のニーズによりよく応えるために、オペレーションテクノロジー (OT) システムをモダナイズし、アップグレードするためのアプローチ。ヒストリアンは、工場内のさまざまなソースからデータを収集して保存するために使用されるデータベースの一種です。

同種データベースの移行

お客様の出典データベースを、同じデータベースエンジンを共有するターゲットデータベース (Microsoft SQL Server から Amazon RDS for SQL Server など) に移行する。同種間移行は、通常、リホストまたはリプラットフォーム化の作業の一部です。ネイティブデータベースユーティリティを使用して、スキーマを移行できます。

ホットデータ

リアルタイムデータや最近の翻訳データなど、頻繁にアクセスされるデータ。通常、このデータには高速なクエリ応答を提供する高性能なストレージ階層またはクラスが必要です。

ホットフィックス

本番環境の重大な問題を修正するために緊急で配布されるプログラム。緊急性のため、通常、修正は一般的な DevOps リリースワークフローの外で行われます。

ハイパーケア期間

カットオーバー直後、移行したアプリケーションを移行チームがクラウドで管理、監視して問題に対処する期間。通常、この期間は 1～4 日です。ハイパーケア期間が終了すると、アプリケーションに対する責任は一般的に移行チームからクラウドオペレーションチームに移ります。

I

IaC

「Infrastructure as Code」を参照してください。

ID ベースのポリシー

AWS クラウド 環境内のアクセス許可を定義する 1 つ以上の IAM プリンシパルにアタッチされたポリシー。

アイドル状態のアプリケーション

90 日間の平均的な CPU およびメモリ使用率が 5～20% のアプリケーション。移行プロジェクトでは、これらのアプリケーションを廃止するか、オンプレミスに保持するのが一般的です。

IIoT

「産業モノのインターネット」を参照してください。

イミュータブルインフラストラクチャ

既存のインフラストラクチャを更新、パッチ適用、または変更するのではなく、本番ワークロード用の新しいインフラストラクチャをデプロイするモデル。イミュータブルなインフラストラクチャは、本質的にミュータブルなインフラストラクチャ よりも一貫性、信頼性、予測性が高くなります。詳細については、 AWS Well-Architected フレームワークの「イミュータブルインフラストラクチャを使用したデプロイ」のベストプラクティスを参照してください。

インバウンド (受信) VPC

AWS マルチアカウントアーキテクチャでは、アプリケーションの外部からネットワーク接続を受け入れ、検査し、ルーティングする VPC。AWS Security Reference Architecture では、アプリ

ケーションとより広範なインターネット間の双方向のインターフェイスを保護するために、インバウンド、アウトバウンド、インスペクションの各 VPC を使用してネットワークアカウントを設定することを推奨しています。

増分移行

アプリケーションを 1 回ですべてカットオーバーするのではなく、小さい要素に分けて移行するカットオーバー戦略。例えば、最初は少数のマイクロサービスまたはユーザーのみを新しいシステムに移行する場合があります。すべてが正常に機能することを確認できたら、残りのマイクロサービスやユーザーを段階的に移行し、レガシーシステムを廃止できるようにします。この戦略により、大規模な移行に伴うリスクが軽減されます。

インダストリー 4.0

接続、リアルタイムデータ、自動化、分析、AI/ML の進歩を通じて、 のビジネスプロセスのモダナイズを指すために 2016 年に Klaus Schwab によって導入された用語。

インフラストラクチャ

アプリケーションの環境に含まれるすべてのリソースとアセット。

Infrastructure as Code (IaC)

アプリケーションのインフラストラクチャを一連の設定ファイルを使用してプロビジョニングし、管理するプロセス。IaC は、新しい環境を再現可能で信頼性が高く、一貫性のあるものにするため、インフラストラクチャを一元的に管理し、リソースを標準化し、スケールを迅速に行えるように設計されています。

産業分野における IoT (IIoT)

製造、エネルギー、自動車、ヘルスケア、ライフサイエンス、農業などの産業部門におけるインターネットに接続されたセンサーやデバイスの使用。詳細については、「Building an industrial Internet of Things (IIoT) digital transformation strategy」を参照してください。

インスペクション VPC

AWS マルチアカウントアーキテクチャでは、VPC (同一または異なる 内 AWS リージョン)、インターネット、オンプレミスネットワーク間のネットワークトラフィックの検査を管理する一元化された VPCs。AWS Security Reference Architecture では、アプリケーションとより広範なインターネット間の双方向のインターフェイスを保護するために、インバウンド、アウトバウンド、インスペクションの各 VPC を使用してネットワークアカウントを設定することを推奨しています。

IoT

インターネットまたはローカル通信ネットワークを介して他のデバイスやシステムと通信する、センサーまたはプロセッサが組み込まれた接続済み物理オブジェクトのネットワーク。詳細については、「IoT とは」を参照してください。

解釈可能性

機械学習モデルの特性で、モデルの予測がその入力にどのように依存するかを人間が理解できる度合いを表します。詳細については、「AWS を使用した機械学習モデルの解釈」を参照してください。

IoT

「モノのインターネット」を参照してください。

IT 情報ライブラリ (ITIL)

IT サービスを提供し、これらのサービスをビジネス要件に合わせるための一連のベストプラクティス。ITIL は ITSM の基盤を提供します。

IT サービス管理 (ITSM)

組織の IT サービスの設計、実装、管理、およびサポートに関連する活動。クラウドオペレーションと ITSM ツールの統合については、オペレーション統合ガイド を参照してください。

ITIL

「IT 情報ライブラリ」を参照してください。

ITSM

「IT サービス管理」を参照してください。

L

ラベルベースアクセス制御 (LBAC)

強制アクセス制御 (MAC) の実装で、ユーザーとデータ自体にそれぞれセキュリティラベル値が明示的に割り当てられます。ユーザーセキュリティラベルとデータセキュリティラベルが交差する部分によって、ユーザーに表示される行と列が決まります。

ランディングゾーン

ランディングゾーンは、スケーラブルで安全な、適切に設計されたマルチアカウント AWS 環境です。これは、組織がセキュリティおよびインフラストラクチャ環境に自信を持ってワークロー

ドとアプリケーションを迅速に起動してデプロイできる出発点です。ランディングゾーンの詳細については、安全でスケーラブルなマルチアカウント AWS 環境のセットアップ を参照してください。

大規模な移行

300 台以上のサーバの移行。

LBAC

「ラベルベースのアクセスコントロール」を参照してください。

最小特権

タスクの実行には必要最低限の権限を付与するという、セキュリティのベストプラクティス。詳細については、IAM ドキュメントの最小特権アクセス許可を適用するを参照してください。

リフトアンドシフト

「7 R」を参照してください。

リトルエンディアンシステム

最下位バイトを最初に格納するシステム。エンディアンネス も参照してください。

下位環境

「環境」を参照してください。

M

機械学習 (ML)

パターン認識と学習にアルゴリズムと手法を使用する人工知能の一種。ML は、モノのインターネット (IoT) データなどの記録されたデータを分析して学習し、パターンに基づく統計モデルを生成します。詳細については、「機械学習」を参照してください。

メインブランチ

「ブランチ」を参照してください。

マルウェア

コンピュータのセキュリティまたはプライバシーを侵害するように設計されているソフトウェア。マルウェアは、コンピュータシステムの中断、機密情報の漏洩、不正アクセスにつながる

可能性があります。マルウェアの例としては、ウイルス、ワーム、ランサムウェア、トロイの木馬、スパイウェア、キーロガーなどがあります。

マネージドサービス

AWS サービス がインフラストラクチャレイヤー、オペレーティングシステム、プラットフォーム AWS を運用し、ユーザーがエンドポイントにアクセスしてデータを保存および取得します。Amazon Simple Storage Service (Amazon S3) と Amazon DynamoDB は、 マネージドサービスの例です。これらは抽象化されたサービス とも呼ばれます。

製造実行システム (MES)

生産プロセスを追跡、モニタリング、文書化、制御するためのソフトウェアシステム。これにより、加工品を現場の完成製品に変換します。

MAP

「移行促進プログラム」を参照してください。

メカニズム

ツールを作成し、ツールの導入を推進し、調整のために結果を検査する完全なプロセス。メカニズムとは、動作中にそれ自体を強化して改善するサイクルです。詳細については、 AWS 「 Well-Architected フレームワーク」の「メカニズムの構築」を参照してください。

メンバーアカウント

内の組織の一部である管理アカウント AWS アカウント 以外のすべて AWS Organizations。アカウントが組織のメンバーになることができるのは、一度に 1 つのみです。

MES

「製造実行システム」を参照してください。

メッセージキューイングテレメトリトランスポート (MQTT)

リソースに制約のある IoT デバイス用の、パブリッシュ/サブスクライブパターンに基づく軽量の machine-to-machine (M2M) 通信プロトコル。

マイクロサービス

明確に定義された API を介して通信し、通常は小規模な自己完結型のチームが所有する、小規模で独立したサービスです。例えば、保険システムには、販売やマーケティングなどのビジネス機能、または購買、請求、分析などのサブドメインにマッピングするマイクロサービスが含まれる場合があります。マイクロサービスの利点には、俊敏性、柔軟なスケーリング、容易なデプロ

イ、再利用可能なコード、回復力などがあります。詳細については、AWS 「サーバーレスサービスを使用したマイクロサービスの統合」を参照してください。

マイクロサービスアーキテクチャ

各アプリケーションプロセスをマイクロサービスとして実行する独立したコンポーネントを使用してアプリケーションを構築するアプローチ。これらのマイクロサービスは、軽量 API を使用して、明確に定義されたインターフェイスを介して通信します。このアーキテクチャの各マイクロサービスは、アプリケーションの特定の機能に対する需要を満たすように更新、デプロイ、およびスケーリングできます。詳細については、「 でのマイクロサービスの実装 AWS」を参照してください。

Migration Acceleration Program (MAP)

コンサルティングサポート、トレーニング、サービスを提供する AWS プログラム。組織がクラウドへの移行のための強固な運用基盤を構築し、移行の初期コストを相殺するのに役立ちます。MAP には、組織的な方法でレガシー移行を実行するための移行方法論と、一般的な移行シナリオを自動化および高速化する一連のツールが含まれています。

大規模な移行

アプリケーションポートフォリオの大部分を次々にクラウドに移行し、各ウェーブでより多くのアプリケーションを高速に移動させるプロセス。この段階では、以前の段階から学んだベストプラクティスと教訓を使用して、移行ファクトリー チーム、ツール、プロセスのうち、オートメーションとアジャイルデリバリーによってワークロードの移行を合理化します。これは、AWS 移行戦略 の第 3 段階です。

移行ファクトリー

自動化された俊敏性のあるアプローチにより、ワークロードの移行を合理化する部門横断的なチーム。移行ファクトリーチームには、通常、オペレーション、ビジネスアナリストと所有者、移行エンジニア、デベロッパー、スプリントに取り組む DevOps プロフェッショナルが含まれます。エンタープライズアプリケーションポートフォリオの 20～50% は、ファクトリーのアプローチによって最適化できる反復パターンで構成されています。詳細については、このコンテンツセットの移行ファクトリーに関する解説とCloud Migration Factory ガイドを参照してください。

移行メタデータ

移行を完了するために必要なアプリケーションおよびサーバーに関する情報。移行パターンごとに、異なる一連の移行メタデータが必要です。移行メタデータの例には、ターゲットサブネット、セキュリティグループ、 AWS アカウントなどがあります。

移行パターン

移行戦略、移行先、および使用する移行アプリケーションまたはサービスを詳述する、反復可能な移行タスク。例: Application Migration Service を使用して Amazon EC2 AWS への移行をリホストします。

Migration Portfolio Assessment (MPA)

に移行するためのビジネスケースを検証するための情報を提供するオンラインツール AWS クラウド。MPA は、詳細なポートフォリオ評価 (サーバーの適切なサイジング、価格設定、TCO 比較、移行コスト分析) および移行プラン (アプリケーションデータの分析とデータ収集、アプリケーションのグループ化、移行の優先順位付け、およびウェーブプランニング) を提供します。MPA ツール (ログインが必要) は、すべての AWS コンサルタントと APN パートナーコンサルタントが無料で利用できます。

移行準備状況評価 (MRA)

AWS CAF を使用して、組織のクラウド対応状況に関するインサイトを取得し、長所と短所を特定し、特定されたギャップを埋めるためのアクションプランを構築するプロセス。詳細については、移行準備状況ガイド を参照してください。MRA は、AWS 移行戦略の第一段階です。

移行戦略

ワークロードを に移行するために使用されるアプローチ AWS クラウド。詳細については、この用語集の「7 Rs エントリ」と「組織を動員して大規模な移行を加速する」を参照してください。

ML

「機械学習」を参照してください。

モダナイゼーション

古い (レガシーまたはモノリシック) アプリケーションとそのインフラストラクチャをクラウド内の俊敏で弾力性のある高可用性システムに変換して、コストを削減し、効率を高め、イノベーションを活用します。詳細については、「」の「アプリケーションをモダナイズするための戦略 AWS クラウド」を参照してください。

モダナイゼーション準備状況評価

組織のアプリケーションのモダナイゼーションの準備状況を判断し、利点、リスク、依存関係を特定し、組織がこれらのアプリケーションの将来の状態をどの程度適切にサポートできるかを決定するのに役立つ評価。評価の結果として、ターゲットアーキテクチャのブループリント、モダナイゼーションプロセスの開発段階とマイルストーンを詳述したロードマップ、特定され

たギャップに対処するためのアクションプランが得られます。詳細については、「」の「アプリケーションのモダナイゼーション準備状況の評価 AWS クラウド」を参照してください。

モノリシックアプリケーション (モノリス)

緊密に結合されたプロセスを持つ単一のサービスとして実行されるアプリケーション。モノリシックアプリケーションにはいくつかの欠点があります。1 つのアプリケーション機能エクスペリエンスの需要が急増する場合は、アーキテクチャ全体をスケーリングする必要があります。モノリシックアプリケーションの特徴を追加または改善することは、コードベースが大きくなると複雑になります。これらの問題に対処するには、マイクロサービスアーキテクチャを使用できます。詳細については、モノリスをマイクロサービスに分解する を参照してください。

MPA

「移行ポートフォリオ評価」を参照してください。

MQTT

「Message Queuing Telemetry Transport」を参照してください。

多クラス分類

複数のクラスの予測を生成するプロセス (2 つ以上の結果の 1 つを予測します)。例えば、機械学習モデルが、「この製品は書籍、自動車、電話のいずれですか?」 または、「このお客様にとって最も関心のある商品のカテゴリはどれですか?」と聞くかもしれません。

変更可能なインフラストラクチャ

本番ワークロードの既存のインフラストラクチャを更新および変更するモデル。Well-Architected AWS Framework では、一貫性、信頼性、予測可能性を向上させるために、イミュータブルインフラストラクチャの使用をベストプラクティスとして推奨しています。

O

OAC

「オリジンアクセスコントロール」を参照してください。

OAI

「オリジンアクセスアイデンティティ」を参照してください。

OCM

「組織変更管理」を参照してください。

オフライン移行

移行プロセス中にソースワークロードを停止させる移行方法。この方法はダウンタイムが長くなるため、通常は重要ではない小規模なワークロードに使用されます。

OI

「 オペレーション統合」を参照してください。

OLA

「運用レベルの契約」を参照してください。

オンライン移行

ソースワークロードをオフラインにせずにターゲットシステムにコピーする移行方法。ワークロードに接続されているアプリケーションは、移行中も動作し続けることができます。この方法はダウンタイムがゼロから最小限で済むため、通常は重要な本番稼働環境のワークロードに使用されます。

OPC-UA

「Open Process Communications - Unified Architecture」を参照してください。

オープンプロセス通信 - 統合アーキテクチャ (OPC-UA)

産業用オートメーション用の machine-to-machine (M2M) 通信プロトコル。OPC-UA は、データの暗号化、認証、認可スキームを備えた相互運用性標準を提供します。

オペレーショナルレベルアグリーメント (OLA)

サービスレベルアグリーメント (SLA) をサポートするために、どの機能的 IT グループが互いに提供することを約束するかを明確にする契約。

運用準備状況レビュー (ORR)

インシデントや潜在的な障害の理解、評価、防止、または範囲の縮小に役立つ質問とそれに関連するベストプラクティスのチェックリスト。詳細については、 AWS Well-Architected フレームワークの「運用準備状況レビュー (ORR)」を参照してください。

運用テクノロジー (OT)

産業運用、機器、インフラストラクチャを制御するために物理環境と連携するハードウェアおよびソフトウェアシステム。製造では、OT と情報技術 (IT) システムの統合が、Industry 4.0 トランスフォーメーションの主要な焦点です。

オペレーション統合 (OI)

クラウドでオペレーションをモダナイズするプロセスには、準備計画、オートメーション、統合が含まれます。詳細については、オペレーション統合ガイド を参照してください。

組織の証跡

の組織 AWS アカウント 内のすべての のすべてのイベントを口グ AWS CloudTrail に記録する によって作成された証跡 AWS Organizations。証跡は、組織に含まれている各 AWS アカウント に作成され、各アカウントのアクティビティを追跡します。詳細については、 ドキュメントの「組織の証跡の作成」を参照してください。 CloudTrail

組織変更管理 (OCM)

人材、文化、リーダーシップの観点から、主要な破壊的なビジネス変革を管理するためのフレームワーク。OCM は、変化の導入を加速し、移行問題に対処し、文化や組織の変化を推進することで、組織が新しいシステムと戦略の準備と移行するのを支援します。 AWS 移行戦略では、クラウド導入プロジェクトに必要な変化のスピードから、このフレームワークは人材アクセラレーション と呼ばれます。詳細については、OCM ガイド を参照してください。

オリジンアクセスコントロール (OAC)

では CloudFront、Amazon Simple Storage Service (Amazon S3) コンテンツを保護するためのアクセスを制限するための拡張オプションです。OAC は、すべての のすべての S3 バケット AWS リージョン、 AWS KMS (SSE-KMS) によるサーバー側の暗号化、S3 バケットへの動的 PUTおよび DELETEリクエストをサポートします。

オリジンアクセスアイデンティティ (OAI)

では CloudFront、Amazon S3 コンテンツを保護するためのアクセスを制限するオプションです。OAI を使用すると、 は Amazon S3 が認証できるプリンシパル CloudFront を作成します。認証されたプリンシパルは、特定の CloudFront ディストリビューションを介してのみ S3 バケット内のコンテンツにアクセスできます。OACも併せて参照してください。OAC では、より詳細な、強化されたアクセスコントロールが可能です。

ORR

「運用準備状況レビュー」を参照してください。

OT

「運用技術」を参照してください。

アウトバウンド (送信) VPC

AWS マルチアカウントアーキテクチャでは、アプリケーション内から開始されるネットワーク接続を処理する VPC。AWS Security Reference Architecture では、アプリケーションとより広範なインターネット間の双方向のインターフェイスを保護するために、インバウンド、アウトバウンド、インスペクションの各 VPC を使用してネットワークアカウントを設定することを推奨しています。

P

アクセス許可の境界

ユーザーまたはロールが使用できるアクセス許可の上限を設定する、IAM プリンシパルにアタッチされる IAM 管理ポリシー。詳細については、IAM ドキュメントのアクセス許可の境界を参照してください。

個人を特定できる情報 (PII)

直接閲覧した場合、または他の関連データと組み合わせた場合に、個人の身元を合理的に推測するために使用できる情報。PII の例には、氏名、住所、連絡先情報などがあります。

PII

個人を特定できる情報を参照してください。

プレイブック

クラウドでのコアオペレーション機能の提供など、移行に関連する作業を取り込む、事前定義された一連のステップ。プレイブックは、スクリプト、自動ランブック、またはお客様のモダナイズされた環境を運用するために必要なプロセスや手順の要約などの形式をとることができます。

PLC

「プログラム可能なロジックコントローラー」を参照してください。

PLM

「製品ライフサイクル管理」を参照してください。

ポリシー

アクセス許可の定義 (アイデンティティベースのポリシー を参照) 、アクセス条件の指定 (リソースベースのポリシー を参照) 、または の組織内のすべてのアカウントに対する最大アクセス許可の定義 AWS Organizations (サービスコントロールポリシー を参照) が可能なオブジェクト。

多言語の永続性

データアクセスパターンやその他の要件に基づいて、マイクロサービスのデータストレージテクノロジーを個別に選択します。マイクロサービスが同じデータストレージテクノロジーを使用している場合、実装上の問題が発生したり、パフォーマンスが低下する可能性があります。マイクロサービスは、要件に最も適合したデータストアを使用すると、より簡単に実装でき、パフォーマンスとスケーラビリティが向上します。詳細については、マイクロサービスでのデータ永続性の有効化 を参照してください。

ポートフォリオ評価

移行を計画するために、アプリケーションポートフォリオの検出、分析、優先順位付けを行うプロセス。詳細については、「移行準備状況ガイド」を参照してください。

述語

`true` または を返すクエリ条件。`false`通常は `WHERE`句にあります。

述語プッシュダウン

転送前にクエリ内のデータをフィルタリングするデータベースクエリ最適化手法。これにより、リレーショナルデータベースから取得して処理する必要があるデータの量が減少し、クエリのパフォーマンスが向上します。

予防的コントロール

イベントの発生を防ぐように設計されたセキュリティコントロール。このコントロールは、ネットワークへの不正アクセスや好ましくない変更を防ぐ最前線の防御です。詳細については、Implementing security controls on AWSのPreventative controlsを参照してください。

プリンシパル

アクションを実行し AWS 、リソースにアクセスできる のエンティティ。このエンティティは通常、、IAM ロール AWS アカウント、またはユーザーのルートユーザーです。詳細については、IAM ドキュメントのロールに関する用語と概念内にあるプリンシパルを参照してください。

プライバシーバイデザイン

エンジニアリングプロセス全体を通してプライバシーを考慮に入れたシステムエンジニアリングのアプローチ。

プライベートホストゾーン

1 つ以上の VPC 内のドメインとそのサブドメインへの DNS クエリに対し、Amazon Route 53 がどのように応答するかに関する情報を保持するコンテナ。詳細については、Route 53 ドキュメントの「プライベートホストゾーンの使用」を参照してください。

プロアクティブコントロール

非準拠のリソースのデプロイを防止するように設計されたセキュリティコントロール。これらのコントロールは、プロビジョニング前にリソースをスキャンします。リソースがコントロールに準拠していない場合、プロビジョニングされません。詳細については、 AWS Control Tower ドキュメントの「 コントロールリファレンスガイド」および「 でのセキュリティコントロールの実装」の「プロアクティブコントロール」を参照してください。 AWS

製品ライフサイクル管理 (PLM)

設計、開発、発売から成長と成熟まで、製品のデータとプロセスのライフサイクル全体にわたる管理。

本番環境

「環境」を参照してください。

プログラミング可能ロジックコントローラー (NAL)

製造では、マシンをモニタリングし、承認プロセスを自動化する、信頼性が高く適応可能なコンピュータです。

仮名化

データセット内の個人識別子をプレースホルダー値に置き換えるプロセス。仮名化は個人のプライバシー保護に役立ちます。仮名化されたデータは、依然として個人データとみなされます。

パブリッシュ/サブスクライブ (pub/sub)

マイクロサービス間の非同期通信を可能にするパターン。スケーラビリティと応答性を向上させます。例えば、マイクロサービスベースの MES では、マイクロサービスは他のマイクロサービスがサブスクライブできるチャネルにイベントメッセージを発行できます。システムは、公開サービスを変更せずに新しいマイクロサービスを追加できます。

Q

クエリプラン

SQL リレーショナルデータベースシステムのデータにアクセスするために使用される手順などの一連のステップ。

クエリプランのリグレッション

データベースサービスのオプティマイザーが、データベース環境に特定の変更が加えられる前に選択されたプランよりも最適性の低いプランを選択すること。これは、統計、制限事項、環境設

定、クエリパラメータのバインディングの変更、およびデータベースエンジンの更新などが原因である可能性があります。

R

RACI マトリックス

[責任、説明責任、相談、通知 (RACI)] を参照してください。

ランサムウェア

決済が完了するまでコンピュータシステムまたはデータへのアクセスをブロックするように設計された、悪意のあるソフトウェア。

RASCI マトリックス

[責任、説明責任、相談、通知 (RACI)] を参照してください。

RCAC

[「行と列のアクセスコントロール」] を参照してください。

リードレプリカ

読み取り専用に使用されるデータベースのコピー。クエリをリードレプリカにルーティングして、プライマリデータベースへの負荷を軽減できます。

再構築

[「7 Rs」を参照してください]。

目標復旧時点 (RPO)

最後のデータリカバリポイントからの最大許容時間です。これにより、最後の回復時点からサービスが中断されるまでの間に許容できるデータ損失の程度が決まります。

目標復旧時間 (RTO)

サービスの中断から復旧までの最大許容遅延時間。

リファクタリング

[「7 Rs」を参照してください]。

リージョン

地理的エリア内の AWS リソースのコレクション。各 AWS リージョン は、耐障害性、安定性、耐障害性を提供するために、他の とは分離され、独立しています。詳細については、AWS リージョン 「 を使用できるアカウントを指定する」を参照してください。

回帰

数値を予測する機械学習手法。例えば、「この家はどれくらいの値段で売れるでしょうか?」という問題を解決するために、機械学習モデルは、線形回帰モデルを使用して、この家に関する既知の事実 (平方フィートなど) に基づいて家の販売価格を予測できます。

リホスト

「7 Rs」を参照してください。

リリース

デプロイプロセスで、変更を本番環境に昇格させること。

再配置

「7 R」を参照してください。

プラットフォーム変更

「7 R」を参照してください。

再購入

「7 R」を参照してください。

回復性

中断に耐えたり、中断から回復したりするアプリケーションの機能。で障害耐性を計画する場合、高可用性とディザスタリカバリが一般的な考慮事項です AWS クラウド。詳細については、AWS クラウド 「レジリエンス」を参照してください。

リソースベースのポリシー

Amazon S3 バケット、エンドポイント、暗号化キーなどのリソースにアタッチされたポリシー。このタイプのポリシーは、アクセスが許可されているプリンシパル、サポートされているアクション、その他の満たすべき条件を指定します。

実行責任者、説明責任者、協業先、報告先 (RACI) に基づくマトリックス

移行活動とクラウド運用に関わるすべての関係者の役割と責任を定義したマトリックス。マトリックスの名前は、マトリックスで定義されている責任の種類、すなわち責任 (R) 、説明責任

(A) 、協議 (C) 、情報提供 (I) に由来します。サポート (S) タイプはオプションです。サポートを含めると、そのマトリックスは RASCI マトリックスと呼ばれ、サポートを除外すると RACI マトリックスと呼ばれます。

レスポンシブコントロール

有害事象やセキュリティベースラインからの逸脱について、修復を促すように設計されたセキュリティコントロール。詳細については、Implementing security controls on AWSのResponsive controlsを参照してください。

保持

「7 R」を参照してください。

廃止

「7 Rs」を参照してください。

ローテーション

攻撃者が認証情報にアクセスすることをより困難にするために、シークレットを定期的に更新するプロセス。

行と列のアクセス制御 (RCAC)

アクセスルールが定義された、基本的で柔軟な SQL 表現の使用。RCAC は行権限と列マスクで構成されています。

RPO

「目標復旧時点」を参照してください。

RTO

「目標復旧時間」を参照してください。

ランブック

特定のタスクを実行するために必要な手動または自動化された一連の手順。これらは通常、エラー率の高い反復操作や手順を合理化するために構築されています。

S

SAML 2.0

多くの ID プロバイダー (IdPs) が使用するオープンスタンダード。この機能により、フェデレーティッドシングルサインオン (SSO) が有効になるため、ユーザーは にログイン AWS

Management Console したり、組織内のすべてのユーザーを IAM で作成しなくても AWS API オペレーションを呼び出すことができます。SAML 2.0 ベースのフェデレーションの詳細については、IAM ドキュメントのSAML 2.0 ベースのフェデレーションについてを参照してください。

SCADA

「監視コントロールとデータ収集」を参照してください。

SCP

「サービスコントロールポリシー」を参照してください。

シークレット

では AWS Secrets Manager、暗号化された形式で保存するパスワードやユーザー認証情報などの機密情報または制限付き情報。シークレット値とそのメタデータで構成されます。シークレット値は、バイナリ、1 つの文字列、または複数の文字列にすることができます。詳細については、Secrets Manager ドキュメントの「Secrets Manager シークレットの内容」を参照してください。

セキュリティコントロール

脅威アクターによるセキュリティ脆弱性の悪用を防止、検出、軽減するための、技術上または管理上のガードレール。セキュリティコントロールには、予防的 、検出的 、???応答的 、プロアクティブ の 4 つの主なタイプがあります。

セキュリティ強化

アタックサーフェスを狭めて攻撃への耐性を高めるプロセス。このプロセスには、不要になったリソースの削除、最小特権を付与するセキュリティのベストプラクティスの実装、設定ファイル内の不要な機能の無効化、といったアクションが含まれています。

Security Information and Event Management (SIEM) システム

セキュリティ情報管理 (SIM) とセキュリティイベント管理 (SEM) のシステムを組み合わせたツールとサービス。SIEM システムは、サーバー、ネットワーク、デバイス、その他ソースからデータを収集、モニタリング、分析して、脅威やセキュリティ違反を検出し、アラートを発信します。

セキュリティレスポンスの自動化

セキュリティイベントに自動的に応答または修正するように設計された、事前定義されたプログラムされたアクション。これらの自動化は、セキュリティのベストプラクティスの実装に役立つ検出的または応答的な AWS セキュリティコントロールとして機能します。自動レスポンスア

クションの例としては、VPC セキュリティグループの変更、Amazon EC2 インスタンスへのパッチ適用、認証情報のローテーションなどがあります。

サーバー側の暗号化

送信先にあるデータの、それを受け取る AWS サービス による暗号化。

サービスコントロールポリシー (SCP)

AWS Organizationsの組織内の、すべてのアカウントのアクセス許可を一元的に管理するポリシー。SCP は、管理者がユーザーまたはロールに委任するアクションに、ガードレールを定義したり、アクションの制限を設定したりします。SCP は、許可リストまたは拒否リストとして、許可または禁止するサービスやアクションを指定する際に使用できます。詳細については、 AWS Organizations ドキュメントの「サービスコントロールポリシー」を参照してください。

サービスエンドポイント

のエントリポイントの URL AWS サービス。ターゲットサービスにプログラムで接続するには、エンドポイントを使用します。詳細については、AWS 全般のリファレンスの「AWS サービス エンドポイント」を参照してください。

サービスレベルアグリーメント (SLA)

サービスのアップタイムやパフォーマンスなど、IT チームがお客様に提供すると約束したものを明示した合意書。

サービスレベルインジケータ (SLI)

エラー率、可用性、スループットなど、サービスのパフォーマンス側面の測定。

サービスレベルの目標 (SLO)

サービスレベルのインジケータ によって測定される、サービスの状態を表すターゲットメトリクス。

責任共有モデル

クラウドのセキュリティとコンプライアンス AWS について と共有する責任を説明するモデル。AWS はクラウドのセキュリティを担当しますが、お客様はクラウドのセキュリティを担当します。詳細については、責任共有モデルを参照してください。

SIEM

「セキュリティ情報とイベント管理システム」を参照してください。

単一障害点 (SPOF)

システムを中断させる可能性のあるアプリケーションの単一の重要なコンポーネントの障害。

SLA

「サービスレベルアグリーメント」を参照してください。

SLI

「サービスレベルインジケータ」を参照してください。

SLO

「サービスレベルの目標」を参照してください。

split-and-seed モデル

モダナイゼーションプロジェクトのスケーリングと加速のためのパターン。新機能と製品リリースが定義されると、コアチームは解放されて新しい製品チームを作成します。これにより、お客様の組織の能力とサービスの拡張、デベロッパーの生産性の向上、迅速なイノベーションのサポートに役立ちます。詳細については、「」の「アプリケーションをモダナイズするための段階的アプローチ AWS クラウド」を参照してください。

SPOF

単一障害点 を参照してください。

star スキーマ

トランザクションデータまたは測定データを保存するために 1 つの大きなファクトテーブルを使用し、データ属性を保存するために 1 つ以上の小さなディメンションテーブルを使用するデータベースの組織構造。この構造は、データウェアハウスまたはビジネスインテリジェンスの目的で使用するように設計されています。

strangler fig パターン

レガシーシステムが廃止されるまで、システム機能を段階的に書き換えて置き換えることにより、モノリシックシステムをモダナイズするアプローチ。このパターンは、宿主の樹木から根を成長させ、最終的にその宿主を包み込み、宿主に取って代わるイチジクのつるを例えています。そのパターンは、モノリシックシステムを書き換えるときのリスクを管理する方法として Martin Fowler により提唱されました。このパターンの適用方法の例については、コンテナと Amazon API Gateway を使用して、従来の Microsoft ASP.NET (ASMX) ウェブサービスを段階的にモダナイズを参照してください。

サブネット

VPC 内の IP アドレスの範囲。サブネットは、1 つのアベイラビリティーゾーンに存在する必要があります。

監視統制とデータ収集 (SCADA)

製造では、ハードウェアとソフトウェアを使用して物理アセットと生産オペレーションをモニタリングするシステム。

対称暗号化

データの暗号化と復号に同じキーを使用する暗号化のアルゴリズム。

合成テスト

ユーザーインタラクションをシミュレートして潜在的な問題を検出したり、パフォーマンスをモニタリングしたりする方法でシステムをテストします。Amazon CloudWatch Synthetics を使用してこれらのテストを作成できます。

T

タグ

AWS リソースを整理するためのメタデータとして機能するキーと値のペア。タグは、リソースの管理、識別、整理、検索、フィルタリングに役立ちます。詳細については、「AWS リソースのタグ付け」を参照してください。

ターゲット変数

監督された機械学習でお客様が予測しようとしている値。これは、結果変数 のことも指します。例えば、製造設定では、ターゲット変数が製品の欠陥である可能性があります。

タスクリスト

ランブックの進行状況を追跡するために使用されるツール。タスクリストには、ランブックの概要と完了する必要のある一般的なタスクのリストが含まれています。各一般的なタスクには、推定所要時間、所有者、進捗状況が含まれています。

テスト環境

「環境」を参照してください。

トレーニング

お客様の機械学習モデルに学習するデータを提供すること。トレーニングデータには正しい答えが含まれている必要があります。学習アルゴリズムは入力データ属性をターゲット (お客様が予測したい答え) にマッピングするトレーニングデータのパターンを検出します。これらのパター

ンをキャプチャする機械学習モデルを出力します。そして、お客様が機械学習モデルを使用して、ターゲットがわからない新しいデータでターゲットを予測できます。

トランジットゲートウェイ

VPC とオンプレミスネットワークを相互接続するために使用できる、ネットワークの中継ハブ。詳細については、 AWS Transit Gateway ドキュメントの「トランジットゲートウェイとは」を参照してください。

トランクベースのワークフロー

デベロッパーが機能ブランチで機能をローカルにビルドしてテストし、その変更をメインブランチにマージするアプローチ。メインブランチはその後、開発環境、本番前環境、本番環境に合わせて順次構築されます。

信頼されたアクセス

ユーザーに代わって AWS Organizations およびそのアカウントで組織内でタスクを実行するために指定するサービスへのアクセス許可を付与します。信頼されたサービスは、サービスにリンクされたロールを必要なときに各アカウントに作成し、ユーザーに代わって管理タスクを実行します。詳細については、 ドキュメントの「 AWS Organizations を他の AWS のサービスで使用する AWS Organizations 」を参照してください。

チューニング

機械学習モデルの精度を向上させるために、お客様のトレーニングプロセスの側面を変更する。例えば、お客様が機械学習モデルをトレーニングするには、ラベル付けセットを生成し、ラベルを追加します。これらのステップを、異なる設定で複数回繰り返して、モデルを最適化します。

ツーピザチーム

2 つのピザを食べることができる小さな DevOps チーム。ツーピザチームの規模では、ソフトウェア開発におけるコラボレーションに最適な機会が確保されます。

U

不確実性

予測機械学習モデルの信頼性を損なう可能性がある、不正確、不完全、または未知の情報を指す概念。不確実性には、次の 2 つのタイプがあります。認識論的不確実性は、限られた、不完全なデータによって引き起こされ、弁論的不確実性 は、データに固有のノイズとランダム性によって引き起こされます。詳細については、深層学習システムにおける不確実性の定量化 ガイドを参照してください。

未分化なタスク

ヘビーリフティングとも呼ばれ、アプリケーションの作成と運用には必要だが、エンドユーザーに直接的な価値をもたらさなかったり、競争上の優位性をもたらしたりしない作業です。未分化なタスクの例としては、調達、メンテナンス、キャパシティプランニングなどがあります。

上位環境

「環境」を参照してください。

V

バキューミング

ストレージを再利用してパフォーマンスを向上させるために、増分更新後にクリーンアップを行うデータベースのメンテナンス操作。

バージョンコントロール

リポジトリ内のソースコードへの変更など、変更を追跡するプロセスとツール。

VPC ピアリング

プライベート IP アドレスを使用してトラフィックをルーティングできる、2 つの VPC 間の接続。詳細については、Amazon VPC ドキュメントの「VPC ピア機能とは」を参照してください。

脆弱性

システムのセキュリティを脅かすソフトウェアまたはハードウェアの欠陥。

W

ウォームキャッシュ

頻繁にアクセスされる最新の関連データを含むバッファキャッシュ。データベースインスタンスはバッファキャッシュから、メインメモリまたはディスクからよりも短い時間で読み取りを行うことができます。

ウォームデータ

アクセス頻度の低いデータ。この種類のデータをクエリする場合、通常は適度に遅いクエリでも問題ありません。

ウィンドウ関数

現在のレコードに関連する行のグループに対して計算を実行する SQL 関数。ウィンドウ関数は、移動平均の計算や、現在の行の相対位置に基づく行の値へのアクセスなどのタスクの処理に役立ちます。

ワークロード

ビジネス価値をもたらすリソースとコード (顧客向けアプリケーションやバックエンドプロセスなど) の総称。

ワークストリーム

特定のタスクセットを担当する移行プロジェクト内の機能グループ。各ワークストリームは独立していますが、プロジェクト内の他のワークストリームをサポートしています。たとえば、ポートフォリオワークストリームは、アプリケーションの優先順位付け、ウェーブ計画、および移行メタデータの収集を担当します。ポートフォリオワークストリームは、これらの設備を移行ワークストリームで実現し、サーバーとアプリケーションを移行します。

WORM

「書き込み 1 回」を参照し、多くの を読み取ります。

WQF

「AWS ワークロード認定フレームワーク」を参照してください。

Write Once, Read Many (WORM)

データを 1 回書き込み、データの削除や変更を防ぐストレージモデル。承認されたユーザーは、必要な回数だけデータを読み取ることができますが、変更することはできません。このデータストレージインフラストラクチャはイミュータブルな と見なされます。

Z

ゼロデイエクスプロイト

ゼロデイ脆弱性 を利用する攻撃、通常はマルウェア。

ゼロデイ脆弱性

実稼働システムにおける未解決の欠陥または脆弱性。脅威アクターは、このような脆弱性を利用してシステムを攻撃する可能性があります。開発者は、よく攻撃の結果で脆弱性に気付きます。

ゾンビアプリケーション

平均 CPU およびメモリ使用率が 5% 未満のアプリケーション。移行プロジェクトでは、これらのアプリケーションを廃止するのが一般的です。

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。