

<div>PostgreSQL 用 Amazon Aurora Labs</div> <div><div>▶ PostgreSQL 用 Amazon Aurora Labs</div><div>▶ すべての研究室の前提条件</div><div>▶ ラボ1.5 : Cloud9の設定とデータベースの初期化</div><div>▶ ラボ2 : 高速クローニング</div><div>▶ ラボ3 : クエリ・プランの管理</div><div>▶ ラボ4 : クラスターキャッシュの管理</div><div>▶ ラボ5 : データベース・アクティビティ・ストリーミング</div><div>▶ ラボ6 : RDSパフォーマンスインサイト</div><div>▶ ラボ7 : データセットの作成とオートスケール</div><div>▶ ラボ8 : フォールトトレランスのテスト</div><div>▶ ラボ9 : オーロラ・グローバル・データベース</div><div>▶ ラボ10: Auroraサーバーレスv1</div><div>▶ ラボ11 : オーロラ機械学習</div><div>▶ ラボ12 : グラビトン2とx86の比較</div><div>▼ ラボ13 : Auroraサーバーレスv2</div><div>▶ 研究室14: Aurora PostgreSQLのための信頼できる言語拡張</div><div>▶ PostgreSQLアドバンスラボ</div><div>▶ クリーンアップ</div><div>▶ 貢献者と改訂履歴</div><div>▶ その他のリソース</div></div>	<div>✕</div>
--	--------------

ラボ6 : RDSパフォーマンスインサイト

このラボでは、[Amazon RDS Performance Insights](#)の使用を実演します。Amazon RDS Performance Insights (RDS PI) は、Amazon RDS DB インスタンスの負荷を監視し、データベースのパフォーマンスを分析してトラブルシューティングできるようにします。

このラボには以下のタスクが含まれています：

1. サンプルデータをAurora PostgreSQL DBクラスタにロードします。
2. RDS Performance Insightsインターフェイスを理解する
3. RDS Performance Insightsを使用してパフォーマンスの問題を特定する
 1. pgbenchを使用したAurora DBクラスタへの大量のインサート負荷
 2. pgbenchを使用したAurora DBクラスタでの大量の更新負荷

前提条件

このラボでは、以下のラボモジュールを最初に完了する必要があります。

- [すべての研究室の前提条件](#)
- [ラボ1 : 新しいAuroraクラスタの作成](#)（「[すべてのラボの前提条件](#)」セクションで[Aurora PostgreSQLクラスタを使用しない素の最小ラボ環境](#)に従った場合は必須。それ以外は任意）
- [Cloud9を設定し、データベースを初期化](#)する（「[すべてのラボの前提条件](#)」セクションで[Aurora PostgreSQLクラスタを使用しないベア最小ラボ環境](#)に従った場合は必須。そうでない場合はスキップしてください）。

1. サンプルデータをAurora PostgreSQL DBクラスタにロードする。

まず、このラボで使用する必要なスクリプトをすべてダウンロードします。[Open Cloud9 Terminal Window](#)セクションを参照してcloud9ターミナルウィンドウを開き、以下のコマンドを貼り付けます。

```
cd
wget https://auroraworkshopassets.s3-us-west-2.amazonaws.com/labs-scripts/aurora-scrip
```

前述の前提条件のセクションに従った場合、ターミナルウィンドウでpsqlコマンドを実行すると、Aurora PostgreSQLクラスタに直接接続されるはずで：

```
psql
```

```
WSParticipantRole:~ $ psql
psql (14.8, server 15.3)
WARNING: psql major version 14, server major version 15.
Some psql features might not work.
SSL connection (protocol: TLSv1.2, cipher: AES128-SHA256, bits: 128, compression: off)
Type "help" for help.

mylab=>
```

psqlを終了するには、「\q」とタイプする。

Cloud9のターミナルウィンドウで以下のコマンドを実行してサンプルHRスキーマを作成します：

```
cd /home/ec2-user/aurora-scripts/scriptspsql -f postgres-hr.sql
```

それでは、先ほど作成したHRスキーマにサンプルデータをロードしてみよう：

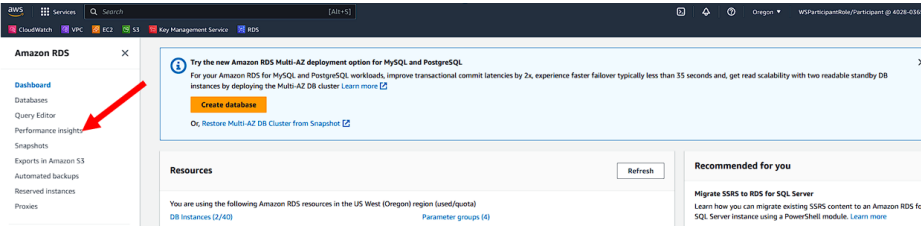
```
psqlset search_path='hr';
set session_replication_role to replica;
\i hr-load.sql
\i hrobject.sql
Select hr.add_employee_data(10000);
Select count(*) from hr.employees;
\q
```

2.RDS Performance Insightsインターフェイスを理解する

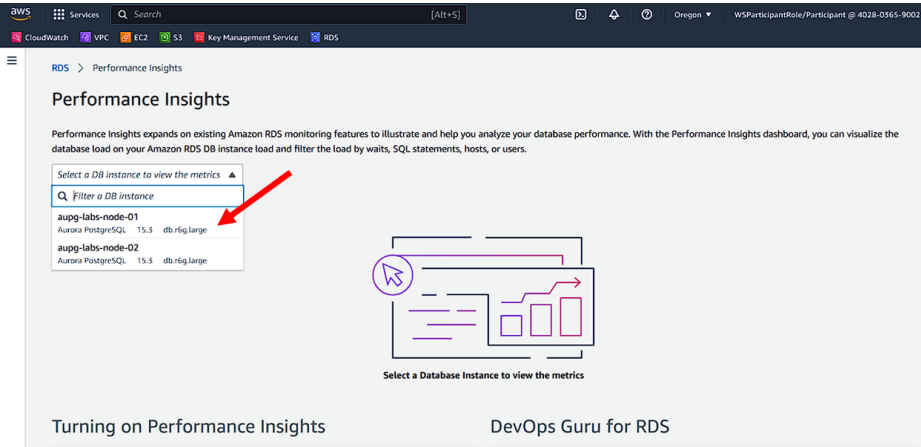
コマンドを実行している間に、まだ開いていなければ、新しいタブでAmazon RDSサービス コンソールを開く。

正しいAWSリージョンで作業していることを確認してください。

左側のメニューで、Performance Insightsメニューオプションをクリックします。



次に、パフォーマンス・メトリクスをロードするDBインスタンスを選択します。Aurora DBクラスタの場合、パフォーマンスメトリクスは個々のDBインスタンスベースで公開されます。クラスタを構成する異なるDBインスタンスは、異なるワークロードパターンを実行する可能性があり、すべてのPerformance Insightsが有効になっているとは限りません。このラボでは、ライター（プライマリ）DBインスタンスのみで負荷を生成します。名前が -node-01 または -instance-1 で終わる DB インスタンスを選択します。



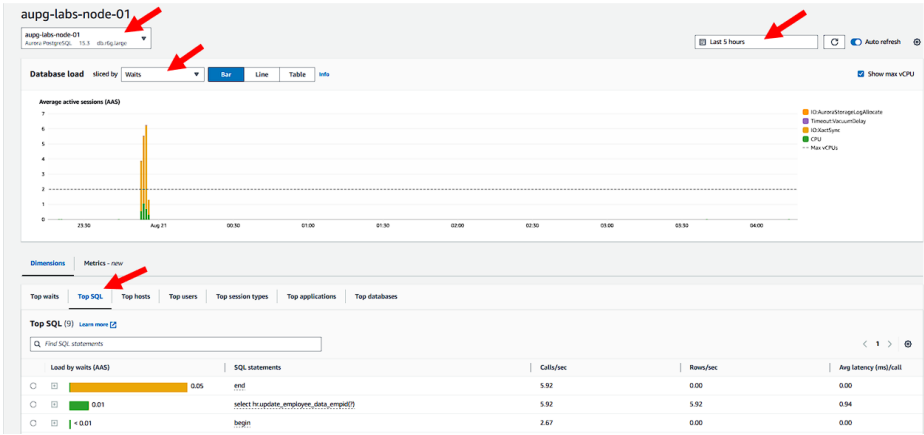
DBインスタンスが選択されると、RDS Performance Insightsのメインダッシュボードが表示されます。ダッシュボードは2つのセクションに分かれており、高レベルのパフォーマンス指標メトリクスから、個々の待ち時間、クエリ、ユーザー、負荷を発生させているホストまでドリルダウンできます。

PostgreSQL 用 Amazon Aurora Labs

- ▶ PostgreSQL 用 Amazon Aurora Labs
- ▶ すべての研究室の前提条件
 - ラボ1：新しいAuroraクラスタを手動で作成する
- ▶ ラボ1.5：Cloud9の設定とデータベースの初期化
 - ラボ2：高速クローニング
 - ラボ3：クエリ・プランの管理
- ▶ ラボ4：クラスターキャッシュの管理
- ▶ ラボ5：データベース・アクティビティ・ストリーミング
- ▶ ラボ6：RDSパフォーマンスインサイト
- ▶ ラボ7：データセットの作成とオートスケール
- ▶ ラボ8：フォールトトレランスのテスト
- ▶ ラボ9：オーロラ・グローバル・データベース
- ▶ ラボ10: Auroraサーバーレスv1
- ▶ ラボ11：オーロラ機械学習
- ▶ ラボ12：グラビトン2とx86の比較
- ▼ ラボ13：Auroraサーバーレスv2
 - Aurora Serverless v2 DBの作成
 - Aurora Serverless v2のスケールリング
 - サーバーレスv2にプロビジョニング
- ▶ 研究室14: Aurora PostgreSQLのための信頼できる言語拡張
- ▶ PostgreSQLアドバンスラボ
 - クリーンアップ
 - 貢献者と改訂履歴
 - その他のリソース

PostgreSQL 用 Amazon Aurora Labs

- ▶ PostgreSQL 用 Amazon Aurora Labs
- ▶ すべての研究室の前提条件
 - ラボ1：新しいAuroraクラスタを手動で作成する
- ▶ ラボ1.5：Cloud9の設定とデータベースの初期化
 - ラボ2：高速クローニング
 - ラボ3：クエリ・プランの管理
- ▶ ラボ4：クラスターキャッシュの管理
- ▶ ラボ5：データベース・アクティビティ・ストリーミング
- ▶ **ラボ6：RDSパフォーマンスインサイト**
 - ラボ7：データセットの作成とオートスケール
- ▶ ラボ8：フォールトトレランスのテスト
- ▶ ラボ9：オーロラ・グローバル・データベース
- ▶ ラボ10: Auroraサーバーレスv1
- ▶ ラボ11：オーロラ機械学習
- ▶ ラボ12：グラビトン2とx86の比較
- ▼ ラボ13：Auroraサーバーレスv2
 - Aurora Serverless v2 DBの作成
 - Aurora Serverless v2のスケールリング
 - サーバーレスv2にプロビジョニング
- ▶ 研究室14: Aurora PostgreSQLのための信頼できる言語拡張
- ▶ PostgreSQLアドバンスラボ
 - クリーンアップ
 - 貢献者と改訂履歴
 - その他のリソース



ダッシュボードで表示されるパフォーマンス・メトリクスは、移動する時間ウィンドウです。インターフェイスの右上隅にある表示時間をクリックし、相対範囲（5m、1h、5h、24h、1w、カスタム範囲）を選択するか、絶対範囲を指定することで、時間ウィンドウのサイズを調整できます。また、マウス・ポインターで選択し、グラフ上をドラッグすることで、特定の期間にズームインすることもできます。

すべてのダッシュボード・ビューは時間同期されます。ズームインすると、下部の詳細ドリルダウンセクションを含むすべてのビューが調整されます。

以下は、RDS Performance Insightsコンソールのすべてのセクションの概要です。

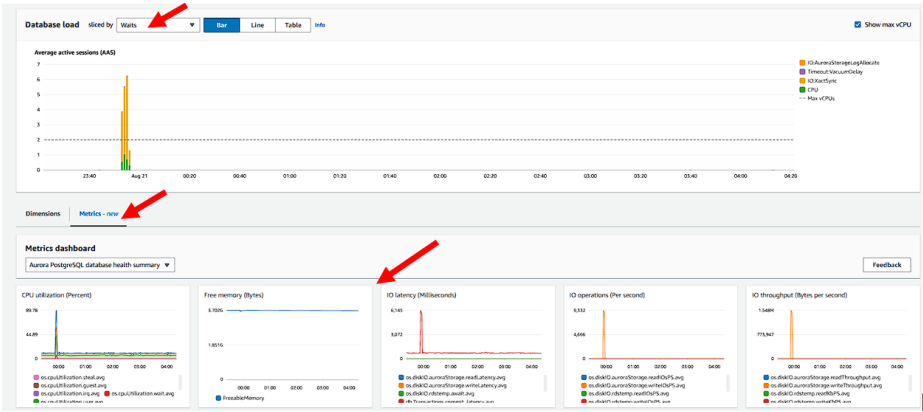
セクション	フィルター	説明
データベースの負荷	負荷は、待機時間（デフォルト）、アプリケーション、データベース、ホスト、セッションタイプ、SQLコマンド、ユーザーによってスライスできる。	このメトリックは、集約された負荷（選択したディメンションでスライス）と、そのDBインスタンスで利用可能な計算能力（vCPU数）を関連付けるように設計されています。負荷は、 平均アクティブ・セッション（AAS） メトリックを使用して集約および正規化されます。DB インスタンスの計算能力を超える AAS の数は、パフォーマンス問題の先行指標となります。
きめ細かなセッション活動	待ち時間、SQL（デフォルト）、ホスト、ユーザー、セッションタイプ、アプリケーション、データベースで並べ替え	個々のコマンドまで詳細なパフォーマンスデータを取得できるドリルダウン機能。Amazon Aurora PostgreSQL特有の待機イベントは、 Amazon Aurora PostgreSQLリファレンス ガイド に記載されています。
指標ダッシュボード	Dimensions "の横にある"Metrics"-新しいタブをクリックすると、カウンターの測定基準が表示されます。	このセクションでは、OSのメトリクス、データベースのメトリクス、CloudWatchのメトリクス（読み取りまたは書き込みの行数、コミットされたトランザクション数など）を一度にプロットします。これらのメトリクスは異常動作の原因を特定するのに便利です。

新しいメトリクスのダッシュボードはこんな感じです。

PostgreSQL 用
Amazon Aurora Labs



- ▶ PostgreSQL 用 Amazon Aurora Labs
- ▶ すべての研究室の前提条件
 - ラボ1：新しいAuroraクラスタを手動で作成する
- ▶ ラボ1.5：Cloud9の設定とデータベースの初期化
 - ラボ2：高速クローニング
 - ラボ3：クエリ・プランの管理
- ▶ ラボ4：クラスターキャッシュの管理
- ▶ ラボ5：データベース・アクティビティ・ストリーミング
 - ラボ6：RDSパフォーマンスインサイト
 - ラボ7：データセットの作成とオートスケール
- ▶ ラボ8：フォールトトレランスのテスト
- ▶ ラボ9：オーロラ・グローバル・データベース
- ▶ ラボ10: Auroraサーバーレスv1
- ▶ ラボ11：オーロラ機械学習
- ▶ ラボ12：グラビトン2とx86の比較
- ▼ ラボ13：Auroraサーバーレスv2
 - Aurora Serverless v2 DBの作成
 - Aurora Serverless v2のスケールリング
 - サーバーレスv2にプロビジョニング
- ▶ 研究室14: Aurora PostgreSQLのための信頼できる言語拡張
- ▶ PostgreSQLアドバンスラボ
 - クリーンアップ
 - 貢献者と改訂履歴
 - その他のリソース



3.RDS Performance Insightsを使用してパフォーマンスの問題を特定する

この演習では、Performance InsightsとPostgreSQL拡張を使用して、上位の待機イベントと性能問題を分析する方法を学びます。HRスキーマのemployeesテーブルに対して、pgbenchユーティリティを使用して挿入と更新の負荷テストケースを実行します。

3.1.pg_stat_statements拡張の作成

新しいpsqlセッションで、mylabデータベースに接続し、以下のSQLコマンドを実行する：

❗ そうしないと、pg_stat_statementsビューがhrスキーマの下に作成されてしまいます。

```
psql CREATE EXTENSION pg_stat_statements;
\q
```

さて、RDS Performance Insightsの機能を理解するために、Auroraインスタンスでいくつかの負荷を実行する準備が整いました。

3.2.pgbenchを使ったAurora DBクラスタの大量インサート負荷

1. cloud9のターミナル・ウィンドウで、以下のコマンドを使ってpgbenchワークロードを実行する：

```
pgbench -n -c 10 -T 300 -f /home/ec2-user/aurora-scripts/scripts/hrload1.sql > /tmp/...
```

hrload1.sqlのSQLスクリプトは、PL/pgSQL関数add_employee_dataを使用して従業員レコードを取り込みます。この関数はemployee_seqを使用して次のemployee_idを生成し、departmentsテーブルからfirst_name、salary、department_idを含むデータをランダムに生成します。各関数呼び出しは5レコードを挿入します。このテストは10人のクライアントで5分間実行されます。

2. PIダッシュボードを確認し、上位の待機イベント、期間中のAAS（平均アクティブセッション）をチェックする。

❗ 時間範囲が1時間に設定され、自動更新が有効になっていることを確認してください。

PostgreSQL 用
Amazon Aurora Labs

▶ PostgreSQL 用 Amazon Aurora Labs

▶ すべての研究室の前提条件

ラボ1：新しいAuroraクラスタを手動で作成する

▶ ラボ1.5：Cloud9の設定とデータベースの初期化

ラボ2：高速クローニング

ラボ3：クエリ・プランの管理

▶ ラボ4：クラスターキャッシュの管理

▶ ラボ5：データベース・アクティビティ・ストリーミング

ラボ6：RDSパフォーマンスインサイト

ラボ7：データセットの作成とオートスケール

▶ ラボ8：フォールトトレランスのテスト

▶ ラボ9：オーロラ・グローバル・データベース

▶ ラボ10: Auroraサーバーレスv1

▶ ラボ11：オーロラ機械学習

▶ ラボ12：グラビトン2とx86の比較

▼ ラボ13：Auroraサーバーレスv2

Aurora Serverless v2 DBの作成

Aurora Serverless v2のスケールリング

サーバーレスv2にプロビジョニング

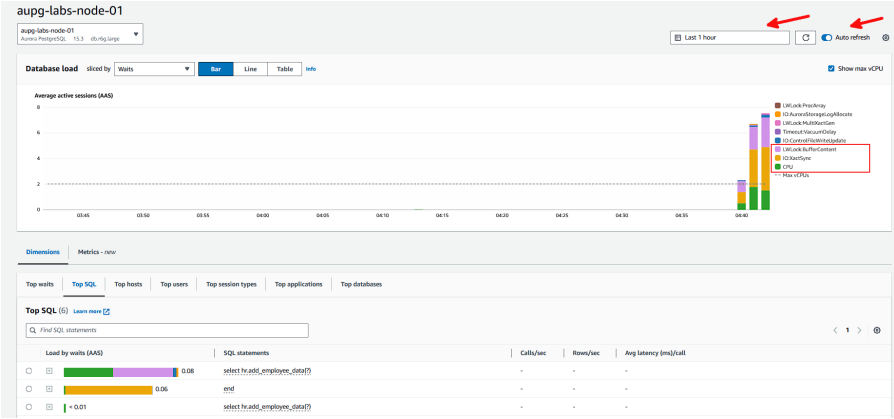
▶ 研究室14: Aurora PostgreSQLのための信頼できる言語拡張

▶ PostgreSQLアドバンスラボ

クリーンアップ

貢献者と改訂履歴

その他のリソース



ウェイティングイベントのトップ3は以下の通り：

IO:XactSync- この待機イベントでは、セッションがCOMMITまたはROLLBACKを発行し、現在のトランザクションの変更を永続化する必要があります。Aurora は、Aurora ストレージが永続化を承認するのを待っています。

CPU

LWLock:Buffer_content- この待機イベントでは、他のセッションがそのページを書き込み用にロックしている間に、セッションがメモリ上のデータ・ページの読み取りまたは書き込みを待機している。

3. pgbenchの出力にあるレイテンシ平均やtpsなどの主要なメトリクスをメモしておくこと。

```
cat /tmp/pgload1-run1.log
```

```
WSParticipantRole:~/environment $
WSParticipantRole:~/environment $ cat /tmp/pgload1-run1.log
pgbench (14.8, server 15.3)
transaction type: /home/ec2-user/aurora-scripts/scripts/hrload1.sql
scaling factor: 1
query mode: simple
number of clients: 10
number of threads: 1
duration: 300 s
number of transactions actually processed: 151094
latency average = 19.843 ms
initial connection time = 210.942 ms
tps = 503.964233 (without initial connection time)
WSParticipantRole:~/environment $
```

4. では、実行時間とCPU消費量の上位5つのクエリを確認してみましょう。
pg_stat_statements拡張を使用した上記のpgbench実行による負荷を理解するために、以下のSQLクエリを実行してください。

```
psql -c"SELECT substring(query, 1, 50) AS short_query, round(total_exec_time::numeric, 2) AS total_exec_time, calls, round(mean_exec_time::numeric, 2) AS mean_exec_time, round(100 * total_exec_time / sum(total_exec_time::numeric) OVER ())::numeric, 2) AS percentage_cpu FROM pg_stat_statements ORDER BY total_exec_time DESC LIMIT 5;"
```

5. 同じ関数を1回の実行で50回インサートして再実行し、待ちイベントへの影響をチェックしてみましょう。この実行にはhrload2.sqlを使用してください。

```
pgbench -n -c 10 -T 300 -f /home/ec2-user/aurora-scripts/scripts/hrload2.sql
```

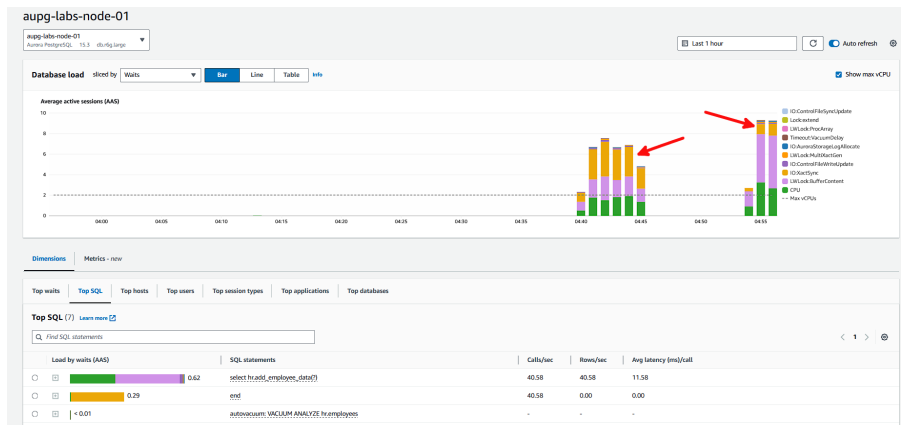

PostgreSQL 用 Amazon Aurora Labs



- ▶ PostgreSQL 用 Amazon Aurora Labs
- ▶ すべての研究室の前提条件
 - ラボ1：新しいAuroraクラスタを手動で作成する
- ▶ ラボ1.5：Cloud9の設定とデータベースの初期化
 - ラボ2：高速クローニング
 - ラボ3：クエリ・プランの管理
- ▶ ラボ4：クラスターキャッシュの管理
- ▶ ラボ5：データベース・アクティビティ・ストリーミング
 - ラボ6：RDSパフォーマンスインサイト**
 - ラボ7：データセットの作成とオートスケール
- ▶ ラボ8：フォールトトレランスのテスト
- ▶ ラボ9：オーロラ・グローバル・データベース
- ▶ ラボ10: Auroraサーバーレスv1
- ▶ ラボ11：オーロラ機械学習
- ▶ ラボ12：グラビトン2とx86の比較
- ▼ ラボ13：Auroraサーバーレスv2
 - Aurora Serverless v2 DBの作成
 - Aurora Serverless v2のスケールリング
 - サーバーレスv2にプロビジョニング
- ▶ 研究室14: Aurora PostgreSQLのための信頼できる言語拡張
- ▶ PostgreSQLアドバンスラボ
 - クリーンアップ
 - 貢献者と改訂履歴
 - その他のリソース

6. PIダッシュボードにアクセスし、上位の待機イベントと上位のSQLをチェックし、変更があるかどうかを確認してください。

- ③ データベース・ロード・セクションに新しいアクティビティが表示されない場合は、時間範囲を**直近5分**に変更し、**適用**をクリックします。その後、**直近1時間**に戻し、**適用**をクリックします。



7. pg_stat_statementsクエリを再実行して、リソースの消費量を確認してください。

```
psql -c "SELECT substring(query, 1, 50) AS short_query, round(total_exec_time::numeric, 2) AS total_exec_time, calls, round(mean_exec_time::numeric, 2) AS mean_exec_time, round(stddev_exec_time::numeric, 2) AS stddev_exec_time FROM pg_stat_statements ORDER BY total_exec_time DESC LIMIT 5;"
```

```
SELECT hradd_employees_data() | 3247881.36 | 179071 | 1 | 17321.43 | 0.23
```

```
copy pgbench_accounts from stdin | 17321.43 | 1 | 17321.43 | 0.46
```

```
alter table pgbench_accounts add primary key (aid) | 5824.65 | 1 | 5824.65 | 0.13
```

```
vacuum analyze pgbench_accounts | 2736.55 | 1 | 2736.55 | 0.07
```

```
WITH upsert AS (UPDATE public.rds_heartbeat2 SET v | 985.47 | 9690 | 0.18 | 0.03
```

```
(5 rows)
```

8. 2つのpgbench実行の待機イベントを比較すると、最新の実行ではIO:XactSync関連の待機が減少していることがわかる。
9. pgbenchが報告するスループットとレイテンシを実行間で比較することで、全体的なスループット（挿入数）が増加したかどうかを検証できますか？

```
cat /tmp/pgload1-run2.log
```

```
WSParticipantRole:~/environment $ cat /tmp/pgload1-run2.log
```

```
pgbench (14.8, server 15.3)
```

```
transaction type: /home/ec2-user/aurora-scripts/scripts/hrload2.sql
```

```
scaling factor: 1
```

```
query mode: simple
```

```
number of clients: 10
```

```
number of threads: 1
```

```
duration: 300 s
```

```
number of transactions actually processed: 27539
```

```
latency average = 108.892 ms
```

```
initial connection time = 201.346 ms
```

```
tps = 91.833962 (without initial connection time)
```

```
WSParticipantRole:~/environment $
```

hrload2.sql 実行で達成されたTPSは、hrload1.sql 実行よりも低いことに注意。しかし、hrload2の各トランザクションは50レコードをロードし、hrload1の各トランザクションは5レコードをロードしたことに留意されたい。では、どちらがより多くのレコードをロードしたのだろうか？

3.3.pgbenchを使用したAurora DBクラスタの大量更新負荷


この演習では、update_employee_data_fname関数とupdate_employee_data_empid関数を使用して従業員テーブルの更新を実行します。

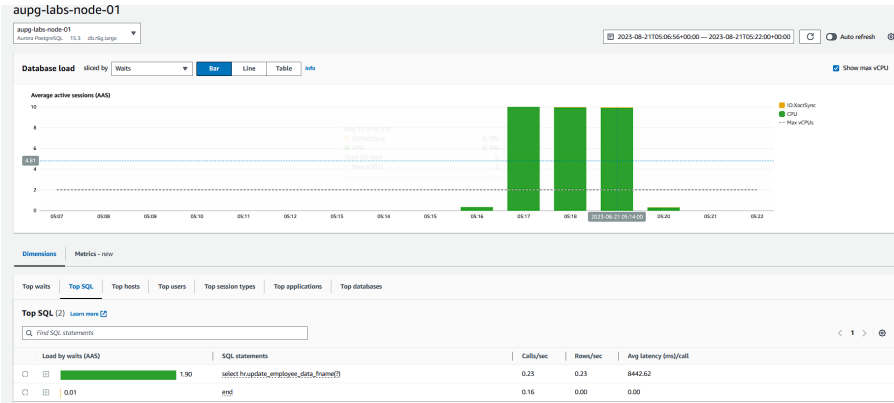
1. cloud9のターミナルウィンドウで、以下のコマンドを使ってpgbench update workloadを実行する：

```
pgbench -n -c 10 -T 180 -f /home/ec2-user/aurora-scripts/scripts/hrupname.sql
```

hrupname.sqlのSQLスクリプトは、PL/pgSQL関数update_employee_data_fnameを使用してemployeesテーブルの従業員の給与の詳細を更新します。この関数は従業員のレコードをランダムに選択し、その給与が範囲内（その仕事の最低給与と最高給与）かどうかをチェックします。各関数呼び出しはランダムに5つのレコードを選択します。このテストは10人のクライアントで3分間実行されます。

2. RDS PIダッシュボードに移動します。実行期間中の上位の待機イベントとAASを確認します。

 データベース・ロード・セクションに新しいアクティビティが表示されない場合は、時間範囲を直近5分に変更し、適用をクリックします。その後、直近1時間に戻し、適用をクリックします。



ウェイティングイベントのトップは

CPU

また、Monitoringタブを選択し、cpuを検索し、CPUUtilizationグラフを展開することで、AuroraクラスタのCPU使用率Cloudwatchメトリクスを確認します。

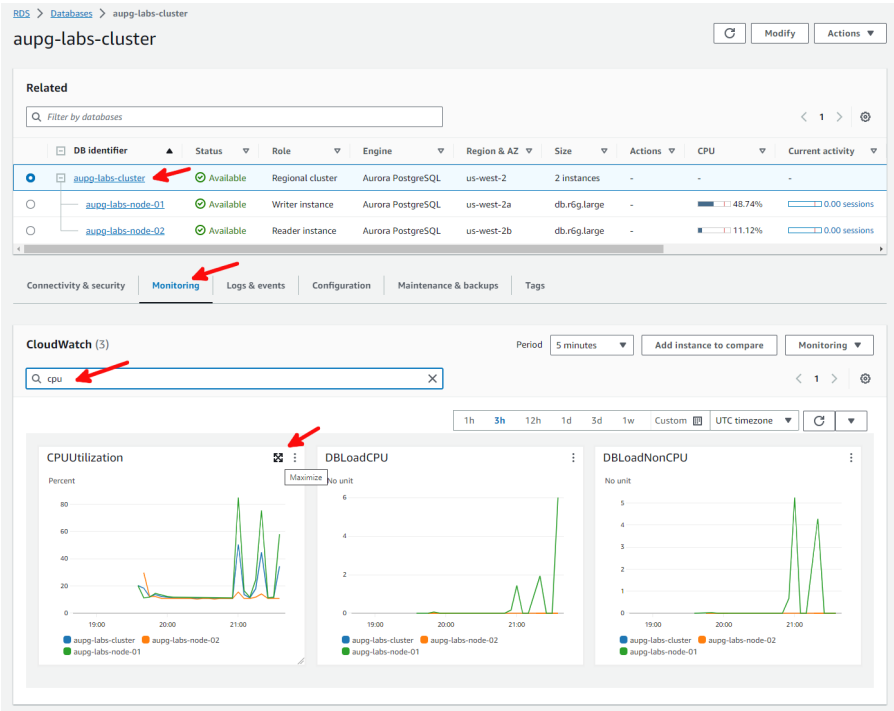
PostgreSQL 用 Amazon Aurora Labs

- ▶ PostgreSQL 用 Amazon Aurora Labs
- ▶ すべての研究室の前提条件
 - ラボ1：新しいAuroraクラスタを手動で作成する
- ▶ ラボ1.5：Cloud9の設定とデータベースの初期化
 - ラボ2：高速クローニング
 - ラボ3：クエリ・プランの管理
- ▶ ラボ4：クラスターキャッシュの管理
- ▶ ラボ5：データベース・アクティビティ・ストリーミング
- ▶ ラボ6：RDSパフォーマンスインサイト
 - ラボ7：データセットの作成とオートスケール
- ▶ ラボ8：フォールトトレランスのテスト
- ▶ ラボ9：オーロラ・グローバル・データベース
- ▶ ラボ10: Auroraサーバーレスv1
- ▶ ラボ11：オーロラ機械学習
- ▶ ラボ12：グラビトン2とx86の比較
- ▼ ラボ13：Auroraサーバーレスv2
 - Aurora Serverless v2 DBの作成
 - Aurora Serverless v2のスケールリング
 - サーバーレスv2にプロビジョニング
- ▶ 研究室14: Aurora PostgreSQLのための信頼できる言語拡張
- ▶ PostgreSQLアドバンスラボ
 - クリーンアップ
 - 貢献者と改訂履歴
 - その他のリソース

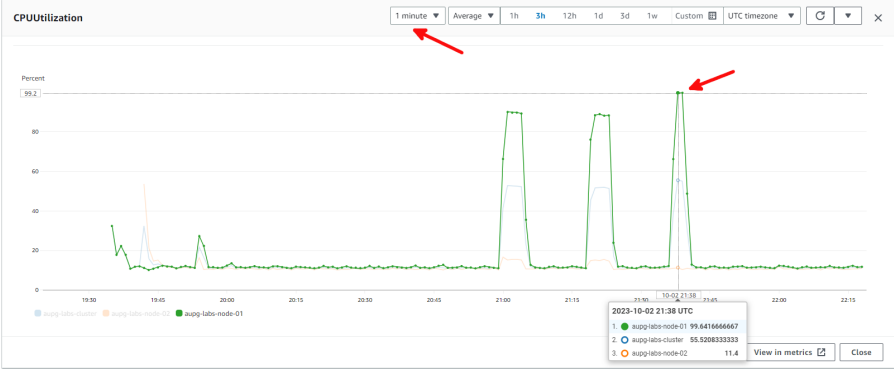
PostgreSQL 用
Amazon Aurora Labs



- ▶ PostgreSQL 用 Amazon Aurora Labs
- ▶ すべての研究室の前提条件
 - ラボ1：新しいAuroraクラスタを手動で作成する
- ▶ ラボ1.5：Cloud9の設定とデータベースの初期化
 - ラボ2：高速クローニング
 - ラボ3：クエリ・プランの管理
- ▶ ラボ4：クラスターキャッシュの管理
- ▶ ラボ5：データベース・アクティビティ・ストリーミング
- ▶ **ラボ6：RDSパフォーマンスインサイト**
- ▶ ラボ7：データセットの作成とオートスケール
- ▶ ラボ8：フォールトトレランスのテスト
- ▶ ラボ9：オーロラ・グローバル・データベース
- ▶ ラボ10: Auroraサーバーレスv1
- ▶ ラボ11：オーロラ機械学習
- ▶ ラボ12：グラビトン2とx86の比較
- ▼ ラボ13：Auroraサーバーレスv2
 - Aurora Serverless v2 DBの作成
 - Aurora Serverless v2のスケールリング
 - サーバーレスv2にプロビジョニング
- ▶ 研究室14: Aurora PostgreSQLのための信頼できる言語拡張
- ▶ PostgreSQLアドバンスラボ
 - クリーンアップ
 - 貢献者と改訂履歴
 - その他のリソース



グラフを更新して**1分間の平均**を表示する。ご覧の通り、更新負荷テスト中にCPUUtilizationが〜100%に達しました。



3. pg_stat_statements拡張機能を使用してパフォーマンス統計を見てみましょう。

1. 以下のコマンドを実行し、CPUを消費する上位5つのクエリを観察する。

```
psql -c "SELECT substring(query, 1, 50) AS short_query, round(total_exec_time::numeric, 2) AS total_exec_time, round(mean_exec_time::numeric, 2) AS mean_exec_time, round(percent_cpu::numeric, 2) AS percent_cpu FROM pg_stat_statements ORDER BY total_exec_time DESC LIMIT 5;"
```

```
select hr_add_employee_data($1) | 3747081.36 | 179071 | 20.93 | 67.23
select hr_update_employee_data($1) | 1796592.13 | 372 | 4829.55 | 32.23
copy pgbench_accounts from stdin | 17321.49 | 1 | 17321.49 | 0.31
alter table pgbench_accounts add primary key (aid) | 5804.45 | 1 | 5804.45 | 0.09
vacuum analyze pgbench_accounts | 2736.95 | 1 | 2736.95 | 0.05
```

2. PL/pgSQL関数のSQL文で使用される説明計画を見てみましょう。説明計画をログに記録するためには、セッションレベルで以下のDBパラメータを設定してください。

```
psqlset auto_explain.log_nested_statements=1;
set auto_explain.log_min_duration=10;
```

これは、10ms以上かかっているネストされたSQL文を含む全てのSQL文を、対応する説明プランと共にerror/postgres.logに記録します。

3. EXPLAIN ANALYZEを実行し、クエリを実行するだけでなく、説明計画を取り込みます。

PostgreSQL 用
Amazon Aurora Labs

▶ PostgreSQL 用 Amazon Aurora Labs

▶ すべての研究室の前提条件

ラボ1：新しいAuroraクラスタを手動で作成する

▶ ラボ1.5：Cloud9の設定とデータベースの初期化

ラボ2：高速クローニング

ラボ3：クエリ・プランの管理

▶ ラボ4：クラスターキャッシュの管理

▶ ラボ5：データベース・アクティビティ・ストリーミング

ラボ6：RDSパフォーマンスインサイト

ラボ7：データセットの作成とオートスケール

▶ ラボ8：フォールトトレランスのテスト

▶ ラボ9：オーロラ・グローバル・データベース

▶ ラボ10: Auroraサーバーレスv1

▶ ラボ11：オーロラ機械学習

▶ ラボ12：グラビトン2とx86の比較

▼ ラボ13：Auroraサーバーレスv2

Aurora Serverless v2 DBの作成

Aurora Serverless v2のスケールリング

サーバーレスv2にプロビジョニング

▶ 研究室14: Aurora PostgreSQLのための信頼できる言語拡張

▶ PostgreSQLアドバンスラボ

クリーンアップ

貢献者と改訂履歴

その他のリソース

EXPLAIN ANALYZE SELECT hr.update_employee_data_fname(10);
! 「従業員データ名」を選択します。

```
mylab=> set auto_explain.log_nested_statements=1;
SET
mylab=> set auto_explain.log_min_duration=10;
SET
mylab=> EXPLAIN ANALYZE SELECT hr.update_employee_data_fname(10);
QUERY PLAN
-----
Result (cost=0.00..0.26 rows=1 width=4) (actual time=1834.839..1834.839 rows=1 loops=1)
Planning Time: 0.048 ms
Execution Time: 1836.215 ms
(3 rows)

mylab=>
```

4. Aurora writer (Primary) インスタンスを選択し、**Logs & events**タブを選択します。Logs セクションにスクロールダウンし、**Last written**列でソートし、最新のログを特定します。

Amazon RDS

Related

Filter by databases

DB Identifier	Status	Role	Engine	Region & AZ	Size	Actions	CPU	Current activity	Maintenance	VPC
aupg-labs-cluster	Available	Regional cluster	Aurora PostgreSQL	us-west-2	2 instances					
aupg-labs-node-01	Available	Writer instance	Aurora PostgreSQL	us-west-2a	db.r5g.large		11.21%	0.00 sessions	none	vpc
aupg-labs-node-02	Available	Reader instance	Aurora PostgreSQL	us-west-2a	db.r5g.large		11.80%	0.00 sessions	none	vpc

Connectivity & securityMonitoring**Logs & events**ConfigurationMaintenanceTags

CloudWatch alarms (0)

Filter by Alarms

Name	State	More options
No Alarms found		

Logs (3)

Filter by DB Logs

Name	Last written	Size
error/postgresql.log-2023-10-02-1937	October 02, 2023, 15:35 (UTC-07:00)	76.5 kB
error/postgres.log	October 02, 2023, 13:04 (UTC-07:00)	4.8 kB
error/postgresql.log-2023-10-02-1936	October 02, 2023, 12:57 (UTC-07:00)	7.6 kB

5. 最新のログファイルを選択し、「ダウンロード」をクリックしてログファイルをダウンロードします。

Logs (3)

Filter by DB Logs

Name	Last written	Size
error/postgresql.log-2023-10-02-1937	October 02, 2023, 15:35 (UTC-07:00)	76.5 kB
error/postgres.log	October 02, 2023, 13:04 (UTC-07:00)	4.8 kB
error/postgresql.log-2023-10-02-1936	October 02, 2023, 12:57 (UTC-07:00)	7.6 kB

6. ダウンロードしたログファイルで、PL/pgSQL関数で更新されている更新文の説明プランを確認してください。説明プランに問題はありますか？

```
2036 --> Seq Scan on employees (cost=0.00..62468.16 rows=1 width=20)
2037   Filter: ((first_name)::text = '1933348-fname')::text
2038 2023-08-20 23:20:00 UTC:10.0.0.65(48844):masteruser@mylab:[30528]:CONTEXT: SQL statement "update hr.employees set salary= minsal where first_name=nam"
2039 PL/pgSQL function hr.update_employee_data_fname(numeric) line 37 at SQL statement
2040 2023-08-20 23:20:00 UTC:10.0.0.65(48844):masteruser@mylab:[30528]:LOG: duration: 236.277 ms plan:
2041   Query Text: update hr.employees set salary= minsal where first_name=nam
2042   Update on employees (cost=0.00..62468.16 rows=1 width=20)
2043 --> Seq Scan on employees (cost=0.00..62468.16 rows=1 width=20)
2044   Filter: ((first_name)::text = (9)::text)
2045 2023-08-20 23:20:00 UTC:10.0.0.65(48844):masteruser@mylab:[30528]:CONTEXT: SQL statement "update hr.employees set salary= minsal where first_name=nam"
2046 PL/pgSQL function hr.update_employee_data_fname(numeric) line 37 at SQL statement
2047 2023-08-20 23:20:00 UTC:10.0.0.65(48844):masteruser@mylab:[30528]:LOG: duration: 1594.198 ms plan:
2048   Query Text: EXPLAIN ANALYZE SELECT hr.update_employee_data_fname(10);
2049   Result (cost=0.00..0.26 rows=1 width=4)
```

updateステートメントがemployeesテーブルの完全なシーケンシャルスキャンを行っており、データのクエリにインデックスを使用していないことに注意してください。

7. では、SQLスクリプト hrupdid.sql を使用してemployee_idカラムを使用し、employeeテーブルを更新するためにロードを再実行してみましょう。

8. cloud9のターミナル・ウィンドウで、以下のコマンドを使ってpgbenchワークロードを実行する。

https://catalog.us-east-1.prod.workshops.aws/workshops/098605dc-8eee-4e84-85e9-c5c6c9e43de2/en-US/lab6-perf-insights

9/11

PostgreSQL 用
Amazon Aurora Labs

- ▶ PostgreSQL 用 Amazon Aurora Labs
- ▶ すべての研究室の前提条件
 - ラボ1：新しいAuroraクラスタを手動で作成する
- ▶ ラボ1.5：Cloud9の設定とデータベースの初期化
 - ラボ2：高速クローニング
 - ラボ3：クエリ・プランの管理
- ▶ ラボ4：クラスターキャッシュの



ビティ・ストリーミング

ラボ6：RDSパフォーマンス
サイト

ラボ7：データセットの作成とオートスケール

- ▶ ラボ8：フォールトトレランスのテスト
- ▶ ラボ9：オーロラ・グローバル・データベース
- ▶ ラボ10: Auroraサーバーレスv1
- ▶ ラボ11：オーロラ機械学習
- ▶ ラボ12：グラビトン2とx86の比較
- ▼ ラボ13：Auroraサーバーレスv2

Aurora Serverless v2 DBの作成

Aurora Serverless v2のスケールリング

サーバーレスv2にプロビジョニング

- ▶ 研究室14: Aurora PostgreSQLのための信頼できる言語拡張

- ▶ PostgreSQLアドバンスラボ
クリーンアップ

貢献者と改訂履歴

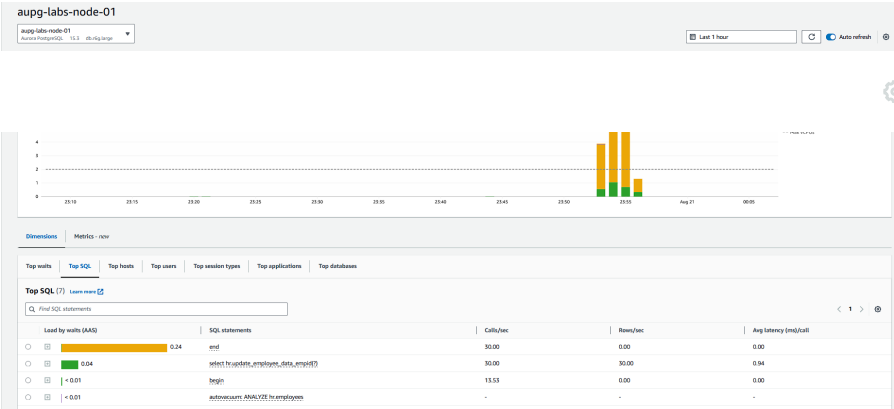
その他のリソース

```
pgbench -n -c 10 -T 180 -f /home/ec2-user/aurora-scripts/scripts/hrupdid.sql > /tmp/
```

PL/pgSQL関数`update_employee_data_empid`を使用して、従業員の給与の詳細を更新します。この関数は従業員のレコードをランダムに選択し、その給与が範囲内（その仕事の最低給与と最高給与）かどうかをチェックします。各関数呼び出しはランダムに5つのレコードを実行します。このテストは10人のクライアントで3分間実行されます。

9. RDS PIのダッシュボードを確認し、上位の待機イベントをもう一度確認してください。上位の待機イベントに変化はありますか？主な見解は何ですか？

③ データベースのロード・セクションに新しいアクティビティが表示されない場合は、時間範囲を**直近5分**に変更し、**適用**をクリックします。その後、**直近1時間**に戻し、**適用**をクリックします。



ウェイトタイプのトップ2は以下の通り：

IO:XactSync- この待機イベントでは、セッションがCOMMITまたはROLLBACKを発行し、現在のトランザクションの変更を永続化することを要求している。

CPU

10. 再度`pg_stat_statements`クエリを使用して実行結果を比較します。

```
psql -c "SELECT substring(query, 1, 50) AS short_query, round(total_exec_time::numeric, 2) AS total_exec_time, calls, round(mean_exec_time::numeric, 2) AS mean_exec_time, round((sum * total_exec_time / sum(total_exec_time::numeric) OVER ()):numeric, 2) AS percentage_cpu FROM pg_stat_statements ORDER BY total_exec_time DESC LIMIT 5;"
```

```
select hr.add_employee_data($1) | 3587264.38 | 172025 | 20.74 | 63.85
select hr.update_employee_data($1) | 1796979.20 | 414 | 4340.53 | 31.99
select hr.update_employee_data2_empid($1) | 144818.98 | 145477 | 0.40 | 2.59
copy pgbench_accounts from stdin | 18124.99 | 1 | 18124.99 | 0.32
WITH upsert AS (UPDATE public.rds_heartbeat2 SET v | 13957.92 | 218099 | 0.06 | 0.25
(0 rows)
```

11. `pgbench`が報告するスループットとレイテンシを実行間で比較する。

```
cat /tmp/pgload2-run1.logcat /tmp/pgload2-run2.log
```

PostgreSQL 用
Amazon Aurora Labs

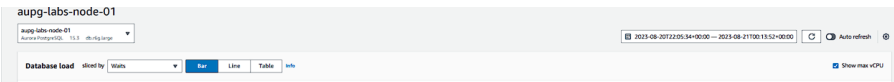


- ▶ PostgreSQL 用 Amazon Aurora Labs
- ▶ すべての研究室の前提条件
 - ラボ1：新しいAuroraクラスタを手動で作成する
- ▶ ラボ1.5：Cloud9の設定とデータベースの初期化
- ▶ ラボ2：高速クローニング

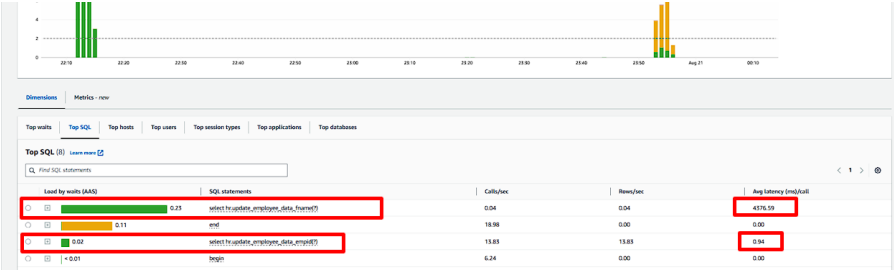
管理

- ▶ ラボ5：データベース・アクティビティ・ストリーミング
- ▶ **ラボ6：RDSパフォーマンスインサイト**
- ▶ ラボ7：データセットの作成とオートスケール
- ▶ ラボ8：フォールトトレランスのテスト
- ▶ ラボ9：オーロラ・グローバル・データベース
- ▶ ラボ10: Auroraサーバーレスv1
- ▶ ラボ11：オーロラ機械学習
- ▶ ラボ12：グラビトン2とx86の比較
- ▼ ラボ13：Auroraサーバーレスv2
 - Aurora Serverless v2 DBの作成
 - Aurora Serverless v2のスケールリング
 - サーバーレスv2にプロビジョニング
- ▶ 研究室14: Aurora PostgreSQLのための信頼できる言語拡張
- ▶ PostgreSQLアドバンスラボ
 - クリーンアップ
 - 貢献者と改訂履歴
 - その他のリソース

```
WSParticipantRole:~/aurora-scripts/scripts $ cat /tmp/pgload2-run1.log
pgbench (14.8, server 15.3)
transaction type: /home/ec2-user/aurora-scripts/scripts/hrupdtype.sql
scaling factor: 1
query mode: simple
number of clients: 10
number of threads: 1
duration: 180 s
number of transactions actually processed: 372
latency average = 4896.739 ms
initial connection time = 165.693 ms
tps = 2.042175 (without initial connection time)
WSParticipantRole:~/aurora-scripts/scripts $ cat /tmp/pgload2-run2.log
pgbench (14.8, server 15.3)
transaction type: /home/ec2-user/aurora-scripts/scripts/hrupdtype.sql
scaling factor: 1
query mode: simple
number of clients: 10
number of threads: 1
duration: 180 s
number of transactions actually processed: 166111
latency average = 10.826 ms
initial connection time = 179.142 ms
tps = 923.689684 (without initial connection time)
WSParticipantRole:~/aurora-scripts/scripts $
```



© 2008 - 2024, Amazon Web Services, Inc.またはその関連会社。無断複写・転載を禁じます。 [プライバシーポリシー](#) [利用規約](#) [クッキーの設定](#)



*employee_id*カラムにインデックスを付け、*first_name*カラムの代わりに*employee_id*カラムを使用して*employee*テーブルを更新することで、TPSは確実に向上し、クエリの実行時間は減少した。