# リアルタイムダッシュボード

Hyperscale (Citus) は、複数のワーカー/リソース間でクエリを並列化して、パフォーマンスを大きく向上させることができます。並列処理の機能の恩恵を受けることができるワークロードの1つは、イベントデータのリアルタイムダッシュボードの機能です。

たとえば、他の企業がHTTPトラフィックを監視できるようにするクラウドサービスプロバイダーになることができます。顧客の1社がHTTP要求を受け取るたびに、サービスはログレコードを受け取ります。これらのレコードをすべて取り入れ、HTTP運用分析ダッシュボードを作成すると、サイトが提供したHTTPエラーの数などの洞察が顧客に提供されます。顧客がサイトの問題を解決できるように、このデータができるだけ少ない待機時間で表示されることが重要です。また、ダッシュボードに過去の傾向のグラフを表示することも重要です。

また、広告ネットワークを構築し、キャンペーンのクリック率を顧客に表示することもできます。この例では、待機時間も重要であり、生データ量も多く、履歴データとライブデータの両方が重要です。

この経験では、Azure Database for PostgreSQLのHyperscale (Citus) を使用して、リアルタイムおよびスケーリングの問題に対処する方法について説明します。

1. このウインドウの右下にある**Next**をクリックします。

# ハイパースケール(Citus)を使ってリアルタイムアプリケーションを作成する

最初に提供されたクレデンシャルを用いてAzureポータルにログインします。

# Azureポータルにサイン・インする

□1. 既にAzureポータルにログインしている場合、次のページに進みます。このウインドウの右下にある**Next**をクリックします。

□2. ブラウザで <https://portal.azure.com> を開き、ブラウザのウインドウを最大化します。

□3. Pick an accountというダイアログが表示されたら、+ Use another accountを選択します。

□4. Sign inダイアログの、Email, phone or Skypeフィールドに  
xxxx@cloudplatimmersionlabs.onmicrosoft.com を入力しNextをクリックします。

□5. **Password**フィールドに **xxxxxxxx** を入力します。

□6. **Sign in**をクリックします。

□7. **Stay signed in?** とタイトルがついた**No**と**Yes**ボタンがあるポップアップが表示されるかもしれません。**No**を選択します。

□8. Welcome to Microsoft AzureとタイトルがついたStart TourとMaybe Laterボタンがあるポップアップが表示されるかもしれません。Maybe Laterを選択します。

□9. このウインドウの右下にある**Next**をクリックします。

# ハイパースケール(Citus)の利用を始める

Azure Portal クラウド シェルを使用するには、ストレージ アカウントを作成する必要があります。ストレージ アカウントを使用すると、クラウド シェルに関連付けられたファイルを保存できるため、スクリプトを実行して Azure リソースを管理するさまざまな Azure ポータルアクティビティで使用できます。

# Cloud Shellを作成する

□1. ポータルのバナーでCloud Shellのアイコンをクリックします。

□2. Welcome to Azure Cloud Shellで**Bash**をクリックします。

□3. You have no storage mountedの画面で、Show advanced settingsをクリックします。

□4. サブスクリプションとリージョンのデフォルト値を使います。

□5. リソースグループは既存の**rg000000**を使うようにしてください。

□6. ストレージアカウントには、Create newを選択し、**sg000000shell**をペーストします。

□7. ファイルシェアには、Create newを選択し、**sg000000shell**を入力してください。

□8. Create Storageをクリックします。

|  |
| --- |
| 注: Cloud Shellを作成・開始するのに1分程度を要します。 |

□9. 次の手順でファイアウォールを構成するには、Cloud Shell のクライアント IP アドレスが必要です。コマンド プロンプトで次のコマンドを入力し、return キーを押してから、クラウド シェルの IP アドレスをコピーまたはメモします。

|  |
| --- |
| curl -s https://ifconfig.co |

|  |
| --- |
| 注: bashコンソールでペーストするには右クリック後にpasteを選択します。 |

□11. このウインドウの右下にある**Next**をクリックします。

# PostgreSQL用のハイパースケール(Citus)拡張の利用を始める

Azure Database for PostgreSQLハイパースケール (Citus) サービスは、サーバー レベルでファイアウォールを使用します。既定では、ファイアウォールはすべての外部アプリケーションとツールがコーディネータ ノードおよび内部のデータベースに接続するのを防ぎます。特定の IP アドレス範囲のファイアウォールを開くルールを追加する必要があります。

このラボでは、4 つの vCore と 16 GB の RAM を備えた 1 つのコーディネータと 2つのワーカーを備えた基本的な本番グレードのハイパースケール (Citus) クラスターを事前にプロビジョニングしました。

# サーバーレベルのファイアウォールのルールを設定する

□1. Azureポータルの左上にある**Home**をクリックします。

□2. Azureサービスの下にあるAzure Database for PostgreSQL serversをクリックします。

□3. **sgxxxxxx**をクリックします。

□4. Securityの下のOverviewペインの左のナビゲーションで**Firewall**をクリックします。

□5. Cloud Shellで確認した**IPアドレス**を**START IP**と**END IP**に入力します。

□6. FIREWALL RULE NAMEにCloudShellと入力します。

□7. ペインの左上にある**Save**をクリックします。

|  |
| --- |
| 注: ハイパースケール(Citus)サーバはポート5432を介して通信します。企業ネットワーク内から接続しようとしている場合、ポート 5432 を超える送信トラフィックは、ネットワークのファイアウォールで許可されない場合があります。その場合は、IT 部門がポート 5432 を開かない限り、ハイパースケール (Citus) サーバーに接続できません。 |

□8. このウインドウの右下にある**Next**をクリックします。

# Azure Database for PostgreSQLのハイパースケール (Citus)に接続する

ハイパースケール(Citus)を作成すると、**citus** という名前の既定のデータベースが作成されます。データベース サーバーに接続するには、接続文字列と管理者パスワードが必要です。最初の接続には最大 2 分かかる場合があります。何らかの理由でシェルがタイムアウトして再起動した場合は、curl -s https://ifconfig.co コマンドをもう一度実行し、ファイアウォールが新しい IP アドレスで更新されていることを確認する必要があります。

# Psqlでデータベースに接続する

□1. Cloud Shellの右上にある**最大化**ボックスをクリックして全画面にします。

□2. Bashプロンプトで、Psqlユーティリティを用いてAzure Database for PostgreSQLに接続します。最初の接続には最大 2 分かかる場合があります。以下のコマンドをコピー＆ペーストして[enter]を押します。

|  |
| --- |
| psql "host=sg000000-c.postgres.database.azure.com port=5432 dbname=citus user=citus password='xxxxxxxx' sslmode=require" |

□3. このウインドウの右下にある**Next**をクリックします。

# データモデル

私たちがこれから扱うデータは、ハイパースケール (Citus) に直接挿入されるログデータで事後に変更されることのないストリームです。また、ログデータを最初にKafkaのようなサービスにルーティングすることも一般的です。Kafkaには、大量のデータを管理できるように、データを事前に集計できるなど、多くの利点があります。

このページでは、HTTPイベントデータを取り込み、シャードし、読み込みとクエリを作成するための単純なスキーマを作成します。

# アプリケーションのテーブルを作成する

http\_requests のテーブル、分単位の集計、および最後のロールアップの位置を維持するテーブルを作成してみましょう。

□1. Psqlコンソールに以下のCREATE TABLEコマンドをコピー＆ペーストしてテーブルを作成します。

|  |
| --- |
| -- this is run on the coordinator  CREATE TABLE http\_request (  site\_id INT,  ingest\_time TIMESTAMPTZ DEFAULT now(),  url TEXT,  request\_country TEXT,  ip\_address TEXT,  status\_code INT,  response\_time\_msec INT  );  CREATE TABLE http\_request\_1min (  site\_id INT,  ingest\_time TIMESTAMPTZ, -- which minute this row represents  error\_count INT,  success\_count INT,  request\_count INT,  average\_response\_time\_msec INT,  CHECK (request\_count = error\_count + success\_count),  CHECK (ingest\_time = date\_trunc('minute', ingest\_time))  );  CREATE INDEX http\_request\_1min\_idx ON http\_request\_1min (site\_id, ingest\_time);  CREATE TABLE latest\_rollup (  minute timestamptz PRIMARY KEY,  CHECK (minute = date\_trunc('minute', minute))  ); |

□2. Psqlコンソールに以下をコピー＆ペーストして作成したものを確認します。

|  |
| --- |
| \dt |

# ノード間にテーブルをシャードする

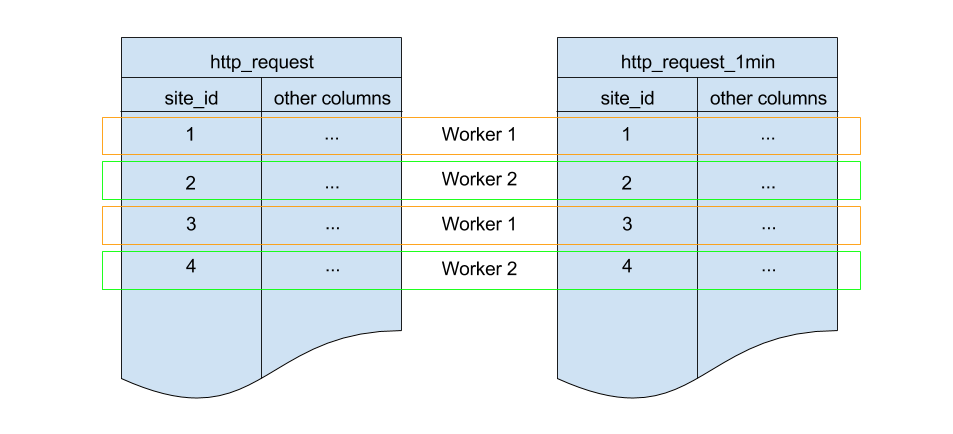
ハイパースケールをデプロイすると、ユーザーが指定した列の値に基づいて、異なるノードにテーブルの行が格納されます。この「分散列」は、ノード間でデータをシャードする方法を示します。分散列をsite\_id、つまりシャードキーに設定してみましょう。

□3. Psqlコンソールに以下をコピー＆ペーストしてテーブルをシャードします。

|  |
| --- |
| SELECT create\_distributed\_table('http\_request', 'site\_id');  SELECT create\_distributed\_table('http\_request\_1min', 'site\_id'); |

上記のコマンドは、ワーカーノード間の2つのテーブルのシャードを作成します。シャードは、一連のサイトを保持するPostgreSQLテーブルにすぎません。テーブルの特定のサイトのすべてのデータは、同じシャードに保持されます。

両方のテーブルがsite\_idでシャードされていることに注意してください。したがって、http\_requestシャードとhttp\_request\_1minシャード、つまり同じサイトのセットを保持する両方のテーブルのシャードが同じワーカーノード上にある、1対1の対応があります。これは**コロケーション**と呼ばれています。コロケーションを使用すると、結合などのクエリがより高速になり、ロールアップが可能になります。次の図では、両方のテーブルのsite\_id 1と3がワーカー1にあり、site\_id 2と4 がWorker2 にあるコロケーションの例が表示されます。



|  |
| --- |
| 注: create\_distributed\_table UDF (ユーザー定義関数) は、シャードカウントのデフォルト値を使用します。デフォルトは32です。HTTPトラフィックの監視と同様のリアルタイム分析のユースケースでは、クラスター内のCPUコアと同じ数のシャードを使用することをお勧めします。これにより、新しいワーカーノードを追加した後、クラスター全体でデータのバランスを取り直すことができます。シャードカウントは、citus.shard\_countを使用して設定できます。これは、create\_distributed\_tableコマンドを実行する前に構成する必要があります。 |

# データの生成

システムはデータを受け入れ、クエリを提供する準備が整いました。次の一連の命令では、この資料の他のコマンドを続行しながら、バックグラウンドでPsqlコンソールで次のループを実行し続けます。それは1秒または2秒ごとに偽のデータを生成します。

□4. クラウドシェルのPsqlコンソールに以下をコピー＆ペーストしてbashコンソールから抜けます。

|  |
| --- |
| \q |

□5. クラウドシェルのバナー上の編集アイコンをクリックします。

壁, 室内 が含まれている画像

自動的に生成された説明

□6. クラウドシェルのエディターに、以下をコピー＆ペーストして（エディターにペーストするには、**Control + V**を使います）、http\_requestの負荷を生成させます。

|  |
| --- |
| -- loop continuously writing records every 1/4 second  DO $$  BEGIN LOOP  INSERT INTO http\_request (  site\_id, ingest\_time, url, request\_country,  ip\_address, status\_code, response\_time\_msec  ) VALUES (  trunc(random()\*32), clock\_timestamp(),  concat('http://example.com/', md5(random()::text)),  ('{China,India,USA,Indonesia}'::text[])[ceil(random()\*4)],  concat(  trunc(random()\*250 + 2), '.',  trunc(random()\*250 + 2), '.',  trunc(random()\*250 + 2), '.',  trunc(random()\*250 + 2)  )::inet,  ('{200,404}'::int[])[ceil(random()\*2)],  5+trunc(random()\*150)  );  COMMIT;  PERFORM pg\_sleep(random() \* 0.25);  END LOOP;  END $$; |

□7. クラウドシェルのエディターの右上にある、省略記号のアイコンをクリックし、**Close Editor**を選びます。

□8. “Do you want to save”ダイアログで、**Save**をクリックします。

□9. ファイル名として以下を入力し、**Save**をクリックします。

|  |
| --- |
| load.sql |

□10. クラウドシェルのbashコンソールに以下をコピー＆ペーストして、バックグラウンドでload.sqlを実行するために[Enter]を押します。

|  |
| --- |
| psql "host=sgxxxxxx-c.postgres.database.azure.com port=5432 dbname=citus user=citus password='spxxxxxxxx' sslmode=require" -f load.sql & |

# ダッシュボードのクエリー

ハイパースケール(Citus)ホスティングオプションを使用すると、複数のノードがクエリを並列処理して高速化できます。たとえば、データベースはワーカーノードのSUMやCOUNTなどの集計を計算し、結果を最終的な回答に結合します。

□11. クラウドシェルのbashコンソールに以下をコピー＆ペーストして、再度Psqlを実行するために[Enter]を押します。

|  |
| --- |
| psql "host=sgxxxxxx-c.postgres.database.azure.com port=5432 dbname=citus user=citus password='spxxxxxxxx' sslmode=require" |

□12. クラウドシェルのPsqlコンソールに以下のコマンドを入力し、リアルタイムの負荷が生成されているかを検証します。

|  |
| --- |
| Select Count(\*) from http\_request; |

□13. クラウドシェルのPsqlコンソールに以下のコマンドを複数回入力し、カウントが増加していることを確認します。

|  |
| --- |
| Select Count(\*) from http\_request; |

このクエリを実行して、1 分あたりの Web 要求といくつかの統計情報をカウントします。

□14. Psqlコンソールに以下をコピー＆ペーストし、サイトに対する平均応答時間を確認します。

|  |
| --- |
| SELECT  site\_id,  date\_trunc('minute', ingest\_time) as minute,  COUNT(1) AS request\_count,  SUM(CASE WHEN (status\_code between 200 and 299) THEN 1 ELSE 0 END) as success\_count,  SUM(CASE WHEN (status\_code between 200 and 299) THEN 0 ELSE 1 END) as error\_count,  SUM(response\_time\_msec) / COUNT(1) AS average\_response\_time\_msec FROM http\_request  WHERE date\_trunc('minute', ingest\_time) > now() - '5 minutes'::interval  GROUP BY site\_id, minute  ORDER BY minute ASC  LIMIT 15; |

|  |
| --- |
| 注: 結果ビューでスタックした場合は、「q」と入力し、「Enter」を押してビューモードを終了します。 |

上記のセットアップは機能しますが、欠点があります。

* HTTP 運用分析ダッシュボードは、グラフを生成する必要があるたびに全ての行を確認する必要があります。たとえば、クライアントが過去1年間の傾向に関心を持っている場合、クエリは過去1年間の全ての行を最初から集計します。
* ストレージコストは、読み込み速度とクエリ可能な履歴の長さに比例して増加します。実際には、生のイベントを短い期間 (1か月) だけ保持し、より長い期間 (年) については履歴グラフだけを見たいはずです。

□15. このウインドウの右下にある**Next**をクリックします。

# ロールアップ

データが増加しても、パフォーマンスを維持することを考えましょう。生データを集計テーブルに定期的にロールアップすることで、ダッシュボードの高速化を保証します。集計期間を試すことができます。この例では、1**分あたりの集計**テーブルを使用しますが、代わりにデータを5分、15分、または60分に分割できます。

このロールアップをより簡単に実行するには、plpgsql関数に配置します。

**http\_request\_1min**を設定するには、SELECTに挿入を定期的に実行します。これは、テーブルがコロケーションされているために可能となります。次の関数は、便宜上ロールアップクエリをラップします。

□1. Psqlコンソールに以下をコピー＆ペーストし、rollup\_http\_request関数を作成します。

|  |
| --- |
| -- initialize to a time long ago  INSERT INTO latest\_rollup VALUES ('10-10-1901');  -- function to do the rollup  CREATE OR REPLACE FUNCTION rollup\_http\_request() RETURNS void AS $$  DECLARE  curr\_rollup\_time timestamptz := date\_trunc('minute', now());  last\_rollup\_time timestamptz := minute from latest\_rollup;  BEGIN  INSERT INTO http\_request\_1min (  site\_id, ingest\_time, request\_count,  success\_count, error\_count, average\_response\_time\_msec  ) SELECT  site\_id,  date\_trunc('minute', ingest\_time),  COUNT(1) as request\_count, SUM(CASE WHEN (status\_code between 200 and 299) THEN 1 ELSE 0 END) as success\_count, SUM(CASE WHEN (status\_code between 200 and 299) THEN 0 ELSE 1 END) as error\_count, SUM(response\_time\_msec) / COUNT(1) AS average\_response\_time\_msec  FROM http\_request  -- roll up only data new since last\_rollup\_time  WHERE date\_trunc('minute', ingest\_time) <@  tstzrange(last\_rollup\_time, curr\_rollup\_time, '(]')  GROUP BY 1, 2;  -- update the value in latest\_rollup so that next time we run the  -- rollup it will operate on data newer than curr\_rollup\_time  UPDATE latest\_rollup SET minute = curr\_rollup\_time;  END;  $$ LANGUAGE plpgsql; |

□2. Psqlコンソールに以下をコピー＆ペーストし、ロールアップ関数を実行します。

|  |
| --- |
| SELECT rollup\_http\_request(); |

|  |
| --- |
| 注: 上記の関数は毎分呼び出す必要があります。これを行うには、**pg\_cron**という PostgreSQL 拡張機能を使用して、データベースから直接定期的なクエリをスケジュールできます。たとえば、上記のロールアップ関数は、以下のコマンドで毎分呼び出すことができます。 |

|  |
| --- |
| SELECT cron.schedule('\* \* \* \* \*','SELECT rollup\_http\_request();'); |

以前のダッシュボードクエリよりもずっと良くなっています。１分間の集計ロールアップテーブルを照会して、以前と同じレポートを取得できます。

□3. Psqlコンソールに以下をコピー＆ペーストし、１分毎の集計テーブルでのクエリを実行します。

|  |
| --- |
| SELECT site\_id, ingest\_time as minute, request\_count,  success\_count, error\_count, average\_response\_time\_msec  FROM http\_request\_1min  WHERE ingest\_time > date\_trunc('minute', now()) - '5 minutes'::interval  LIMIT 15; |

□4. このウインドウの右下にある**Next**をクリックします。

# 古いデータを期限切れにする

ロールアップによってクエリが高速になりますが、ストレージコストが無限に増大することを回避するために古いデータを期限切れにする必要があります。粒度ごとにデータを保持する期間を決定し、標準クエリを使用して期限切れのデータを削除します。次の例では、生データを１日、１分単位の集計を１か月間、保持することにしました。期限切れになる古いデータがないため、これらのコマンドを今すぐ実行する必要はありません。

|  |
| --- |
| DELETE FROM http\_request WHERE ingest\_time < now() - interval '1 day';  DELETE FROM http\_request\_1min WHERE ingest\_time < now() - interval '1 month'; |

本番環境では、これらのクエリを関数にラップし、cronジョブで毎分呼び出すことができます。

データの有効期限は、Hyperscale (Citus) によるシャーディングに加えて、PostgreSQLの最新の時間パーティショニング機能を使用することで、さらに高速に実行できます。また、pg\_partmanなどの拡張機能を使用して、時間パーティションの作成と保守を自動化することもできます。

これらは基本に過ぎません！ HTTPイベントを取り込み、これらのイベントを事前に集約された形式にロールアップするアーキテクチャを提供しました。これにより、生のイベントを保存し、１秒未満のクエリを使用して分析ダッシュボードをより強力にすることもできます。

次のセクションでは、基本的なアーキテクチャについて説明し、よく出てくる質問を解決する方法を示します。

□1. このウインドウの右下にある**Next**をクリックします。

# おおよその個別の数

HTTP運用分析におけるよくある質問は、おおよその個別の数を扱います: 先月のサイトを訪問したユニーク訪問者数はいくつですか。この質問に正確に答えるには、以前にサイトを訪れたすべての訪問者のリストをロールアップテーブルに格納する必要があります。しかし、おおよその答えははるかに管理しやすくなります。

ハイパーログログ (HLL) と呼ばれるデータ型は、クエリにほぼ答えることができます。セット内のユニークな要素の数を知るには、驚くほど少ないスペースで十分です。その正確さは調節することができます。1280バイトのみで、最大2.2%のエラーがあるものの、何百億という単位のユニーク訪問者数を数えることができるものを使用します。

先月にクライアントのサイトを訪問したユニークなIPアドレスの数など、グローバルクエリを実行する場合は、同様の問題が発生します。HLLがない場合、このクエリには、ワーカーからコーディネータに重複除外するIPアドレスの一覧が含まれます。これは、多くのネットワークトラフィックと計算の両方が必要になってしまいます。HLLを使用すると、クエリの速度を大幅に向上できます。

**ハイパースケール以外 (Citus)**をインストールする場合は、まずHLL拡張機能をインストールして有効にする必要があります。Psqlコマンド**CREATE EXTENSION hll**を実行します。この場合、すべてのノードで実行する必要があります。ハイパースケール (Citus) には、他の便利な拡張機能と共にHLLが既にインストールされているので、この作業はAzureでは必要となりません。

これで、HLLを使用したロールアップでIPアドレスを追跡する準備ができました。最初にロールアップ テーブルに列を追加します。

□1. Psqlコンソールに以下をコピー＆ペーストし、http\_request\_1minテーブルを変更します。

|  |
| --- |
| ALTER TABLE http\_request\_1min ADD COLUMN distinct\_ip\_addresses hll; |

次に、カスタム集計を使用して列を設定します。

□2. Psqlコンソールに以下をコピー＆ペーストし、ロールアップ関数のクエリに追加します。

|  |
| --- |
| -- function to do the rollup  CREATE OR REPLACE FUNCTION rollup\_http\_request() RETURNS void AS $$  DECLARE  curr\_rollup\_time timestamptz := date\_trunc('minute', now());  last\_rollup\_time timestamptz := minute from latest\_rollup;  BEGIN  INSERT INTO http\_request\_1min (  site\_id, ingest\_time, request\_count,  success\_count, error\_count, average\_response\_time\_msec,  distinct\_ip\_addresses  ) SELECT  site\_id,  date\_trunc('minute', ingest\_time),  COUNT(1) as request\_count,  SUM(CASE WHEN (status\_code between 200 and 299) THEN 1 ELSE 0 END) as success\_count,  SUM(CASE WHEN (status\_code between 200 and 299) THEN 0 ELSE 1 END) as error\_count,  SUM(response\_time\_msec) / COUNT(1) AS average\_response\_time\_msec,  hll\_add\_agg(hll\_hash\_text(ip\_address)) AS distinct\_ip\_addresses  FROM http\_request  -- roll up only data new since last\_rollup\_time  WHERE date\_trunc('minute', ingest\_time) <@  tstzrange(last\_rollup\_time, curr\_rollup\_time, '(]')  GROUP BY 1, 2;  -- update the value in latest\_rollup so that next time we run the  -- rollup it will operate on data newer than curr\_rollup\_time  UPDATE latest\_rollup SET minute = curr\_rollup\_time;  END;  $$ LANGUAGE plpgsql; |

INSERT INTOステートメントに**distinct\_ip\_address**が追加され、  
SELECTには**hll\_add\_agg(hll\_hash\_text(ip\_address)) AS distinct\_ip\_address**がrollup\_http\_request関数に追加されました。

□3. Psqlコンソールに以下をコピー＆ペーストし、更新された関数を実行します。

|  |
| --- |
| SELECT rollup\_http\_request(); |

ダッシュボードクエリはもう少し複雑です。hll\_cardinality関数を呼び出すことによって、異なるIPアドレスの数を読み取る必要があります。

□4. Psqlコンソールに以下をコピー＆ペーストし、hll\_cardinality関数を利用したレポートを生成します。

|  |
| --- |
| SELECT site\_id, ingest\_time as minute, request\_count,  success\_count, error\_count, average\_response\_time\_msec,  hll\_cardinality(distinct\_ip\_addresses)::bigint AS distinct\_ip\_address\_count  FROM http\_request\_1min  WHERE ingest\_time > date\_trunc('minute', now()) - interval '5 minutes'  LIMIT 15; |

HLLは単に高速なだけではなく、以前はできなかったことができます。ロールアップを実行したが、HLL を使用する代わりに、正確な一意のカウントを保存したとします。これは正常に動作しますが、「この1週間に、生データを破棄したセッションはいくつあったか」などのクエリには答えられません。

HLLを使用すれば簡単です。次のクエリを使用して、一定期間における個別のIP数を計算できます。

□5. Psqlコンソールに以下をコピー＆ペーストし、期間中の異なるIP数を計算します。

|  |
| --- |
| SELECT hll\_cardinality(hll\_union\_agg(distinct\_ip\_addresses))::bigint  FROM http\_request\_1min  WHERE ingest\_time > date\_trunc('minute', now()) - '5 minutes'::interval  LIMIT 15; |

□6. このウインドウの右下にある**Next**をクリックします。

# JSONBの非構造化データ

ハイパースケール (Citus) は、Postgresに組み込みでサポートされている非構造化データ型とうまく機能します。これを実証するために、各国から来た訪問者数を追跡してみましょう。半構造化データ型を使用すると、個々の国ごとに列を追加する必要がなくなります。PostgreSQLには、JSONデータを格納するためのJSONBデータ型とJSONデータ型があります。データ型としてJSONBが推奨される理由は、a) JSONと比較してJSONBにはインデックス作成機能 (GINおよびGIST) があり、b) JSONBはバイナリ形式であるため圧縮機能が提供される、ためです。ここでは、JSONB列をデータモデルに組み込む方法を示します。

□1. Psqlコンソールに以下をコピー＆ペーストし、ロールアップのテーブルにJSONB列を新たに追加します。

|  |
| --- |
| ALTER TABLE http\_request\_1min ADD COLUMN country\_counters JSONB; |

□2. Psqlコンソールに以下をコピー＆ペーストし、rollup\_http\_requestをcountry\_countersで更新します。

|  |
| --- |
| -- function to do the rollup  CREATE OR REPLACE FUNCTION rollup\_http\_request() RETURNS void AS $$  DECLARE  curr\_rollup\_time timestamptz := date\_trunc('minute', now());  last\_rollup\_time timestamptz := minute from latest\_rollup;  BEGIN  INSERT INTO http\_request\_1min (  site\_id, ingest\_time, request\_count,  success\_count, error\_count, average\_response\_time\_msec,  distinct\_ip\_addresses,  country\_counters  ) SELECT  site\_id,  date\_trunc('minute', ingest\_time),  COUNT(1) as request\_count,  SUM(CASE WHEN (status\_code between 200 and 299) THEN 1 ELSE 0 END) as success\_count,  SUM(CASE WHEN (status\_code between 200 and 299) THEN 0 ELSE 1 END) as error\_count,  SUM(response\_time\_msec) / COUNT(1) AS average\_response\_time\_msec,  hll\_add\_agg(hll\_hash\_text(ip\_address)) AS distinct\_ip\_addresses,  jsonb\_object\_agg(request\_country, country\_count) AS country\_counters  FROM (  SELECT \*,  count(1) OVER (  PARTITION BY site\_id, date\_trunc('minute', ingest\_time), request\_country  ) AS country\_count  FROM http\_request  )h  -- roll up only data new since last\_rollup\_time WHERE date\_trunc('minute', ingest\_time) <@  tstzrange(last\_rollup\_time, curr\_rollup\_time, '(]')  GROUP BY 1, 2;  -- update the value in latest\_rollup so that next time we run the  -- rollup it will operate on data newer than curr\_rollup\_time  UPDATE latest\_rollup SET minute = curr\_rollup\_time;  END;  $$ LANGUAGE plpgsql; |

□3. Psqlコンソールに以下をコピー＆ペーストし、更新した関数を実行します。

|  |
| --- |
| SELECT rollup\_http\_request(); |

ダッシュボードでアメリカから送信されたリクエストの数を取得する場合は、ダッシュボードクエリを次のように変更できます。

□4. Psqlコンソールに以下をコピー＆ペーストし、アメリカからのリクエストを確認します。

|  |
| --- |
| SELECT  request\_count, success\_count, error\_count, average\_response\_time\_msec, COALESCE(country\_counters->>'USA', '0')::int AS american\_visitors  FROM http\_request\_1min WHERE ingest\_time > date\_trunc('minute', now()) - '5 minutes'::interval  LIMIT 15; |

□5. このウインドウの右下にある**Next**をクリックします。

# 結論

このチュートリアルでは、Azure Database for PostgreSQLのハイパースケール(Citus)を使い以下をどのように実行するかを学びました。

* バックグラウンドでリアルタイムの負荷を生成する
* Psql 関数を作成して更新する
* アプリケーションがスケールできるようにデータをロールアップする
* 古いデータの有効期限を切る
* 個別のカウントに関するレポート
* 非構造化データ (JSONB) を使用するようにモデルを更新する

追加の参照情報

* Citus and pg\_partman: Creating a scalable time series database on Postgres ([https://www.citusdata.com/blog/2018/01/24/citus-and- pg-partman-creating-a-scalable-time-series-database-on-PostgreSQL/](https://www.citusdata.com/blog/2018/01/24/citus-and-%20pg-partman-creating-a-scalable-time-series-database-on-PostgreSQL/))
* GitHub - PostgreSQL Cron job (<https://github.com/citusdata/pg_cron>)
* GitHub - HLL HyperLogLog (<https://github.com/citusdata/postgresql-hll>)
* When to use unstructured datatypes in Postgres–Hstore vs. JSON vs. JSONB ([https://www.citusdata.com/blog/2016/07/14/choosing- nosql-hstore-json-jsonb/](https://www.citusdata.com/blog/2016/07/14/choosing-%20nosql-hstore-json-jsonb/))

1. この経験がどのぐらい素晴らしかったか**Feedback**をクリックして教えてください。