Concurrent Connection

Last updated by | Lisa Liu | Nov 6, 2020 at 10:35 AM PST

Concurrent Connection

Monday, January 13, 2020
4:28 PM

Difference SKU has different max connection setting (https://docs.microsoft.com/en-us/azure/postgresql/concepts-limits). If the max connection is hit, the server will encounter the connectivity issue. Too many concurrent connections might indicate the server is under heavy load too.

Customer can check that at their end by running show process list more information here:

select * from pg_stat_activity;

Too many connections can impact server performance in terms if high CPU utilization and slow connection establishment, use the following two queries to troubleshoot this scenario:

Below query will result in the number of connections created per second more than 20 per second, more than 20 newly established connection per second is considered high and in this case we recommend customers to use connection pooling as a best practice.

```
//let TimeCheckStart = datetime('2019-07-30T11:50:00');
//let TimeCheckEnd = datetime('2019-07-30T19:09:00');
MonLogin
| where (logical_server_name =~ "{ServerName}" )
| where AppTypeName == "Gateway.MySQL"
//| where originalEventTimestamp >= TimeCheckStart and originalEventTimestamp <= TimeCheckEnd
//| where originalEventTimestamp >= datetime({StartTime}) and originalEventTimestamp <=
datetime({EndTime})
| where event == "process_login_finish"
| where is_success
| summarize count() by bin(originalEventTimestamp, 1s)
| where count_ > 20
```

The following query shows the latency establishing connection with the database, this might be caused by slow client, high CPU on the Database or issues while doing the SSL handshake.

```
//latency establishing a connection with the server MySQL
MonLogin
| where TIMESTAMP >= ago(1d)
| where originalEventTimestamp >= datetime({StartTime}) and originalEventTimestamp <= datetime({EndTime})
| where (logical_server_name =~ "{ServerName}" )
| where event == "process_login_finish" or event == "connection_accept"
| where AppTypeName == "Gateway.PG" or AppTypeName == "Host.PG"
| where is_success == true
| where connection_id != "00000000-0000-0000-0000-000000000000"
| summarize start=min(originalEventTimestamp), end=max(originalEventTimestamp) by connection_id,
logical_server_name
| extend duration = bin(end - start, 1tick)
| summarize latency_seconds=tolong(avg(duration))/10000000.0 by TIMESTAMP=bin(start, 1m),
logical_server_name
```

To get the concurrent connection count of the server, run following query

```
let StartTime = ago(2d);

let EndTime = ago(1d);

let ServerName = "servername";

MonRdmsServerMetrics

| where originalEventTimestamp > StartTime and originalEventTimestamp < EndTime

| where LogicalServerName == ServerName
```

```
| where metric_name contains "active_connections"
| summarize max(metric_value) by bin(originalEventTimestamp, 1m), metric_name
| render timechart
```

Attempt connection: means how many connection is issued to the server. You can see attempt connection from two place.

From the PostgreSQL metric:

```
let StartTime = ago(2d);
let EndTime = ago(1d);
let ServerName = "servername";
MonRdmsServerMetrics
| where originalEventTimestamp > StartTime and originalEventTimestamp < EndTime
| where LogicalServerName == ServerName
| where metric_name contains "Connections"
| summarize sum(metric_value) by bin(originalEventTimestamp, 1m), metric_name
| render timechart
```

From the Gateway log

```
let StartTime = ago(2d);
let EndTime = ago(1d);
let ServerName = "howang57standard-westeurope";
MonLogin
| where originalEventTimestamp > StartTime and originalEventTimestamp < EndTime
| where logical_server_name == ServerName
| where AppTypeName == "Gateway.PG" and event == "process_login_finish"
| summarize count() by bin(originalEventTimestamp, 1m)
| render timechart
```

Customers can check this from psql or pgAdmin by running the following sql queries

1. Query all active server processes:
   ```
   SELECT * FROM pg_stat_activity WHERE STATE = 'active';
   ```
2. Cancel or terminate:
   ```
   SELECT pg_cancel_backend(pid);
   SELECT pg_terminate_backend(pid);
   ```
3. Terminate all connections on database:
   ```
   SELECT pg_terminate_backend(pid) FROM pg_stat_activity
   WHERE datname = 'dbname';
   ```

Created with Microsoft OneNote 2016.

How good have you found this content?