Useful Kusto Queries regarding PostgreSQL Performance

Last updated by | Hamza Aqel | Nov 24, 2021 at 2:59 AM PST

Useful Kusto Queries regarding PostgreSQL Performance

Tuesday, September 4, 2018
2:51 PM

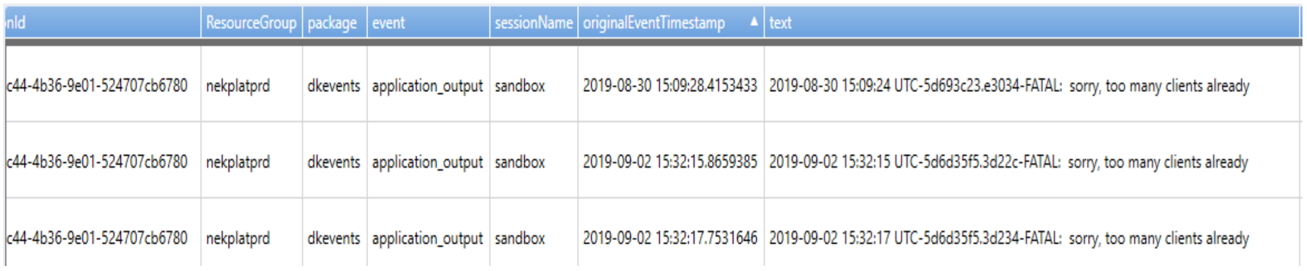
PostgreSQL Performance TSG >> read this carefully, this is very useful as a good start to troubleshoot PostgreSQL Perfromance related incidents

Check resource utilization using the following query

check for too many connections which means the customer maxed out the number of allowed connections, limits are documented here

```
MonRdmsPgSqlSandbox
| where LogicalServerName =~ "{ServerName}"
| where text contains "too many clients"
```

Snippet for output



nId	ResourceGroup	package	event	sessionName	originalEventTimestamp	text
c44-4b36-9e01-524707cb6780	nekplatprd	dkevents	application_output	sandbox	2019-08-30 15:09:28.4153433	2019-08-30 15:09:24 UTC-5d693c23.e3034-FATAL: sorry, too many clients already
c44-4b36-9e01-524707cb6780	nekplatprd	dkevents	application_output	sandbox	2019-09-02 15:32:15.8659385	2019-09-02 15:32:15 UTC-5d6d35f5.3d22c-FATAL: sorry, too many clients already
c44-4b36-9e01-524707cb6780	nekplatprd	dkevents	application_output	sandbox	2019-09-02 15:32:17.7531646	2019-09-02 15:32:17 UTC-5d6d35f5.3d234-FATAL: sorry, too many clients already

//connection latency :

```
  let startTime = datetime(xxxx-xx-xx xx:xx);
  let endTime = datetime(xxxx-xx-xx xx:xx);
  let serverName = "pg-server-name";
  MonLogin
  | where originalEventTimestamp > startTime and originalEventTimestamp < endTime
  | where AppTypeName == "Gateway.PG"
  | where logical_server_name == serverName
  | where event == "process_login_finish"
  | where is_success == true
  | where connection_id != "00000000-0000-0000-0000-000000000000"
  | extend gw_start = datetime_add('millisecond', -total_time_ms, originalEventTimestamp)
  | project gw_start, connection_id
  | join kind = inner(
  MonLogin
  | where originalEventTimestamp > startTime and originalEventTimestamp < endTime
  | where AppTypeName == "Host.PG"
  | where logical_server_name == serverName
  | where event == "process_login_finish"
  | where is_success == true
  | where connection_id != "00000000-0000-0000-0000-000000000000"
  | project host_done = originalEventTimestamp, connection_id
  ) on $left.connection_id == $right.connection_id
  | join kind = inner(
  MonRdmsPgSqlSandbox
  | where originalEventTimestamp > startTime and originalEventTimestamp < endTime
  | where LogicalServerName == serverName
  | where text contains "azure connection id"
  | where text !contains "00000000-0000-0000-0000-000000000000"
  | parse text with * "LOG:  Azure Connection ID: \"" ConnectionId "\", VNET \"" VNet "\", SSL " Protocol "\", state " error_state ", time
  | extend connection_id=toupper(ConnectionId)
  | extend backend_done = originalEventTimestamp
  ) on $left.connection_id == $right.connection_id
  | extend gw_to_sockdup = datetime_diff('millisecond', host_done, gw_start)
  | extend connectivity_start_to_end = datetime_diff('millisecond', backend_done, gw_start)
  //| where connectivity_start_to_end > 500
  | project gw_start,
      connectivity_start_to_end,
      gw_to_sockdup,
      guc_comp, //If this time is high, then the issue is likely to be slow storage.
      auth_comp, con_comp, // opening the database files for the connection. If any of these are slow, then it would indicate a storage is
      be_start, be_ready, //indicate how long the backend processes are taking to start and become ready
      tls_init_comp, //If this is high, then the issue could be time taken to load certificates.
      ssl_end, //If this is high, there might be high latency in the Azure network in the region, or the gateway might be overloaded.
      load_hba, load_id, //indicate time taken to load firewall files
      auth_start, auth_done, //is the password challenge and response.
      conn_recv, //how long it takes to get to printing the "connection received" log message. If this time is high, it could be because s
      sema // if it is a long time, the server is taking a long time to create new processes to handle connections.
  | render timechart
```

```
//CPU
MonResourceMetricsProvider
| where LogicalServerName =~ "{ServerName}"
| where TIMESTAMP >= ago(7d)
| extend cpu_percentage = cpu_load_percent, memory_percentage = working_set_percent
| project TIMESTAMP, cpu_percentage, memory_percentage
```

```
//IOPS(PFS)
MonRdmsServerMetrics
//| where AppTypeName == "Worker.PAL.MySQL"
| where LogicalServerName == "{ServerName}"
| where TIMESTAMP >= ago(5d)
//| where TIMESTAMP >= datetime({StartTime}) and TIMESTAMP <= datetime({EndTime})
| where metric_name contains "io_consumption_percent"
| summarize max(metric_value) by bin(originalEventTimestamp, 1m)
```

```
//Latency establishing a connection to the Azure Database for PostgreSQL instance
MonLogin
| where TIMESTAMP >= ago(4d)
| 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

 //active connections
MonRdmsServerMetrics
| where TIMESTAMP >= ago(7d)
```

```
//| where TIMESTAMP >= datetime({StartTime}) and TIMESTAMP <= datetime({EndTime})
| where LogicalServerName =~ "{ServerName}"
| where metric_name == "active_connections"
| summarize max(metric_value) by bin(originalEventTimestamp, 1m), metric_name
| render timechart

//query count
MonRdmsServerMetrics
| where TIMESTAMP >= ago(1h)
| where LogicalServerName =~ "{ServerName}"
| where metric_name == "Queries"
| summarize sum(metric_value) by bin(originalEventTimestamp, 1s), metric_name

//number of connections created per second - lists only more than 25
MonLogin
| where (logical_server_name =~ "{ServerName}" )
| where AppTypeName == "Gateway.PG"
| where event == "process_login_finish"
//| where is_success
| summarize count() by bin(originalEventTimestamp, 3s)
| where count_ > 25



// number of throttling events on a specific db file (XIO)
let TimeCheckStart = datetime('08/11/2020 09:34:00');
let TimeCheckEnd = datetime('08/14/2020 09:34:00');
let TimeRange = TimeCheckEnd - TimeCheckStart;
let Intervals = iff(TimeRange <= 4h, 1s, iff(TimeRange <= 25h, 5s, 30s));
let Throttling = MonSQLSbs
| where originalEventTimestamp >= TimeCheckStart and originalEventTimestamp <= TimeCheckEnd
| where LogicalServerName =~ 'pg_server_name' and event == 'xio_failed_request' and errorcode == 'ServerBusy';
Throttling
| extend mdf_read_cnt = iff(file_path hassuffix 'sbs.mdf' and request_type == 'XIOTypeRead', 1, 0)
| extend mdf_write_cnt = iff(file_path hassuffix 'sbs.mdf' and request_type == 'XIOTypeWrite', 1, 0)
| extend ldf_read_cnt = iff(file_path hassuffix 'sbs_log.ldf' and request_type == 'XIOTypeRead', 1, 0)
| extend ldf_write_cnt = iff(file_path hassuffix 'sbs_log.ldf' and request_type == 'XIOTypeWrite', 1, 0)
| project Timestamp = originalEventTimestamp, mdf_read_cnt, mdf_write_cnt, ldf_read_cnt, ldf_write_cnt
| summarize mdf_read = sum(mdf_read_cnt),
mdf_write = sum(mdf_write_cnt),
ldf_read = sum(ldf_read_cnt),
ldf_write = sum(ldf_write_cnt) by bin(Timestamp, 1s)
| summarize ['data file read beyond limit'] = max(mdf_read),
['data file write beyond limit (no perf impact)'] = max(mdf_write),
['log file read beyond limit'] = max(ldf_read),
['log file write beyond limit'] = max(ldf_write) by bin(Timestamp, Intervals)
| sort by Timestamp asc
//QueryName:'Azure Database for Postgres Disk Storage IO beyond limit'

//Connection Status
 MonDmPgSqlConnectionStats | where LogicalServerName =~ "{ServerName}"
| where TIMESTAMP >= datetime({StartTime}) and TIMESTAMP <= datetime({EndTime})


//Sessions
 MonDmPgSqlSessions |  where  LogicalServerName =~ "{ServerName}"
| where TIMESTAMP >= datetime({StartTime}) and TIMESTAMP <= datetime({EndTime})


//Connections
 MonDmPgSqlAppConnection | where  LogicalServerName =~ "{ServerName}"
| where TIMESTAMP >= datetime({StartTime}) and TIMESTAMP <= datetime({EndTime})

 //Transactions
 MonDmPgSqlTransactionStats
| where TIMESTAMP >= ago(2d)
//| where TIMESTAMP >= datetime({StartTime}) and TIMESTAMP <= datetime({EndTime})
//| where database_name  == "groundcontrols"
| where  LogicalServerName =~ "{ServerName}"
```

Created with Microsoft OneNote 2016.

How good have you found this content?