SquadStack

PGConf India

Ujjwal Gupta

Senior Product Engineer @ SquadStack

linkedin.com/in/ujjwalgupta983/
medium.com/@ujjwal.gupta_21067

SquadStack

PGConf India

Use Connection Pooling to Enable Postgres Proxy and to Improve Database Performance

Agenda

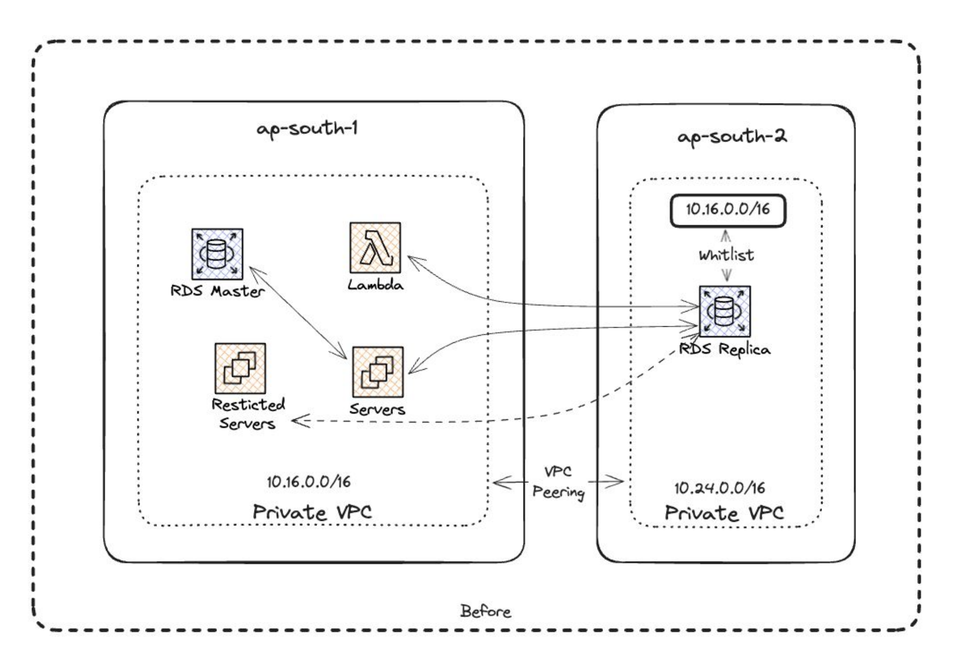
1. Our Use case - Why we decided to use a Connection Pooler?
2. Deep dive into how Postgres establishes connection
3. How does the number of connections affect database server resources?
4. What is Connection Pooling?
5. Pros & Cons of Connection Pooling
6. Performance Benchmarks
7. Pgbouncer vs Pgpool-II
8. Conclusion

1. Our Use Case

Why we decided to use a Connection Pooler?

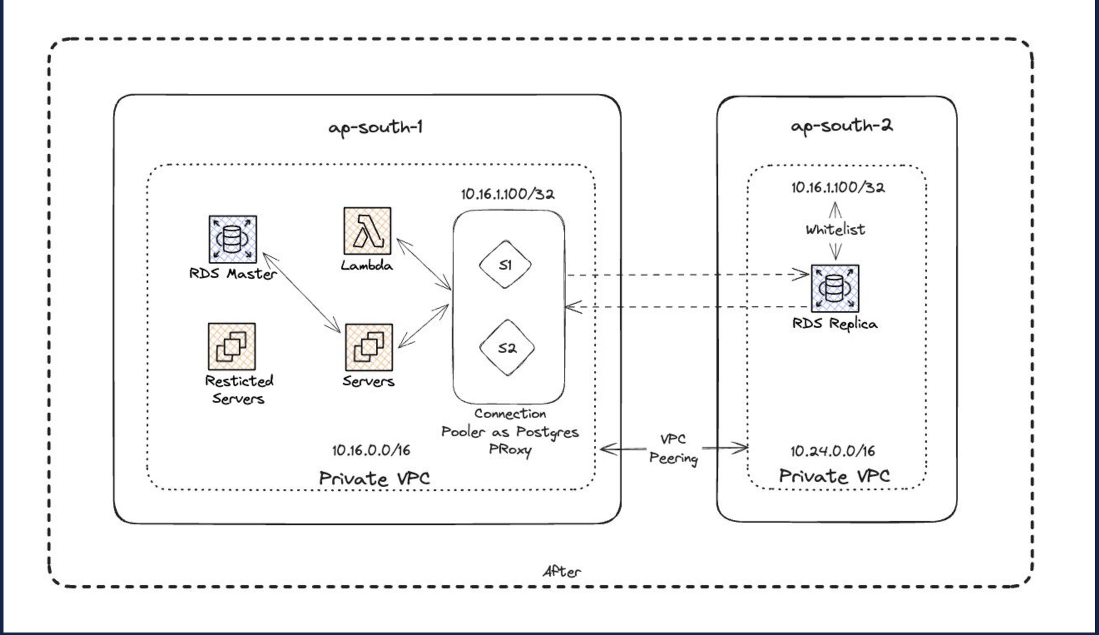
Problems before Connection Pooling

1. High Resource Consumption
2. Security - All servers within the VPC can only access read replica
3. Network Cost Overhead



After Connection Pooling

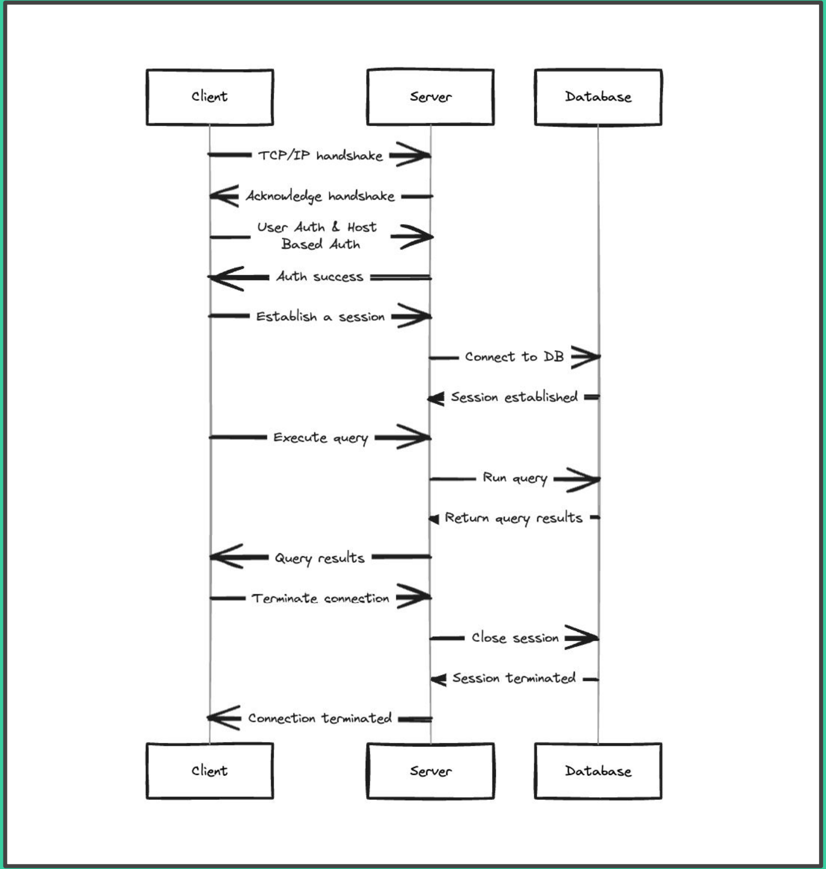
1. Optimized Resource Usage
2. Postgres proxy to handle security
3. Less network cost as compared to previous ones



2. Deep dive into how postgres establishes connection

Steps involved in establishing a postgres connection

1. DNS lookup
2. Three-way handshake
3. TLS handshake
4. Session established
5. Authentication
6. Authorization
7. Executes query and return results
8. Tear down connections



3. How does the number of connections affect database server resources?

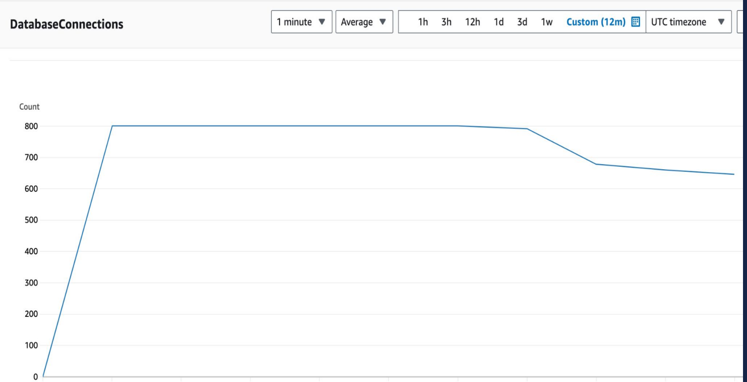
Resource Consumed by Postgres Connections

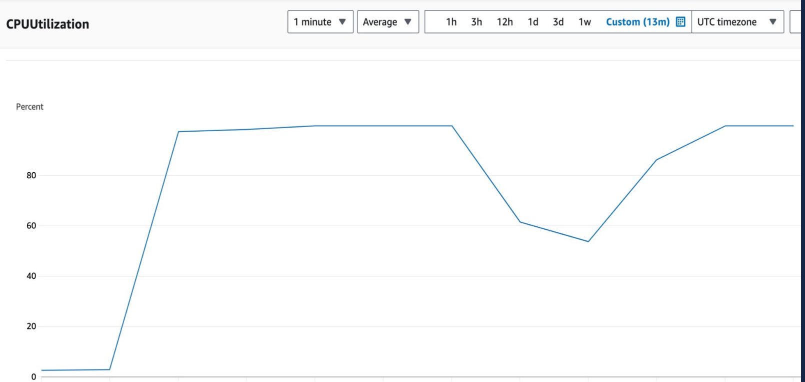
1. **Utilizes both process memory and shared memory**
    1. Shared memory is communal: caches data, locks, and configuration across all connections
    2. Process memory is private: holds code, data, and structures for each connection
2. **CPU to maintain the state of the new connections**
3. **Increase in connections can lead to:**
    1. **Rapid increase in the server's overall process memory usage**
    2. **Impact shared memory - will Increase data page cache, locks and semaphores**
    3. **High CPU consumptions**

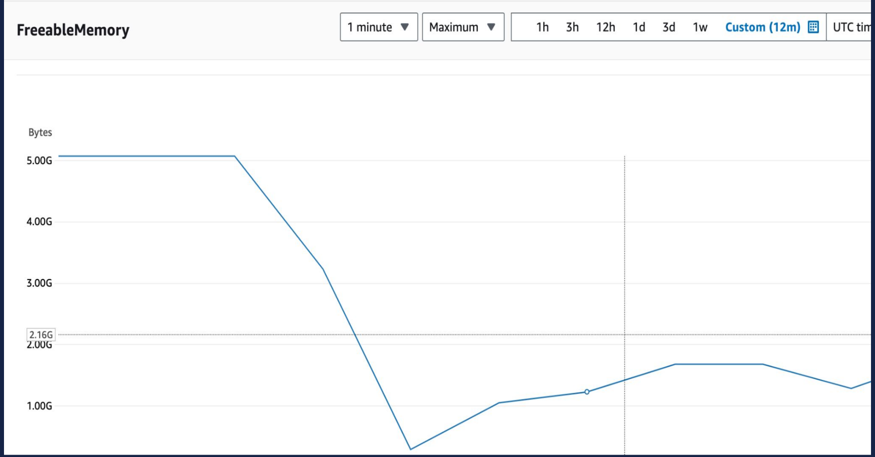
Performed Resource Utilization Tests

1. Instance of size m5.large (2 vCPUs, 8GB RAM)
2. 30GB GP2 storage
3. 800 Connections
4. Created a temporary table and inserted 1 million rows
5. Dropped the temporary table
6. Repeat these steps for all 800 connections
7. Leave the connections idle for 5 minutes
8. Close the connections

Metrics

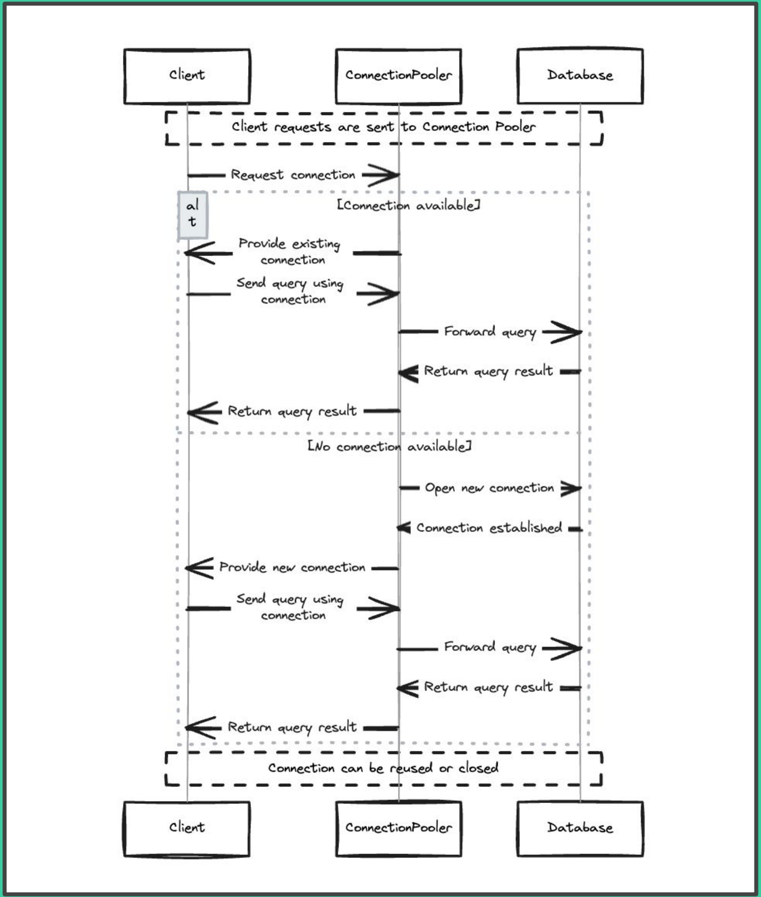

DatabaseConnections


CPUUtilization


FreeableMemory

4. What is Connection Pooling?

Connection pooling is a strategy of recycling database connections for multiple requests instead of closing them immediately when a query has been resolved.



5. Pros & Cons of Connection Pooling

Pros

1. Improved Performance by reducing connection overhead and reduced context switching
2. Limit resource usage
3. Can act as an Postgres proxy
4. Scalability

And many more…

Cons

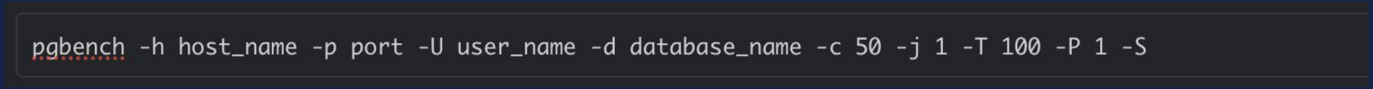
1. Can lead to latency issues if not configured properly
2. Connection leaks
3. Increased Complexity

6. Performance Benchmarks

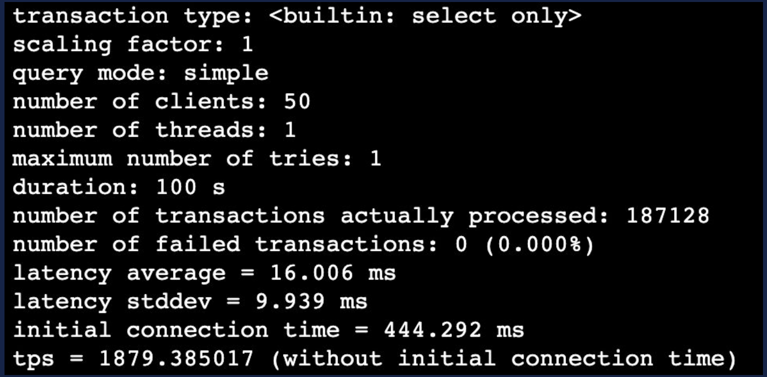
SquadStack

Benchmark using pgbench – with and without pgbouncer

1. Pool size = 50, client connections = 50

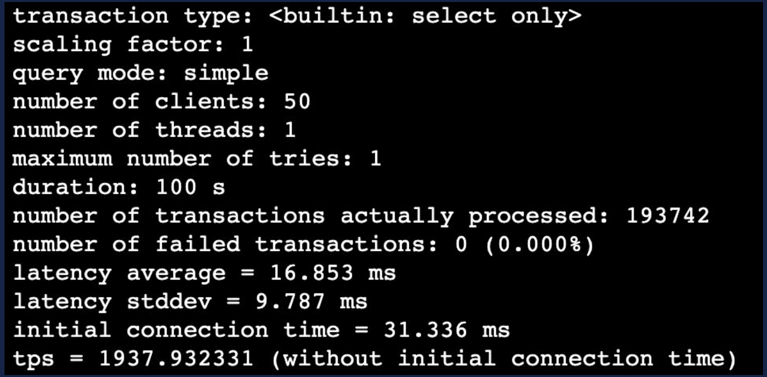


```
pgbench -h host_name -p port -U user_name -d database_name -c 50 -j 1 -T 100 -P 1 -S
```



```
transaction type: <builtin: select only>
scaling factor: 1
query mode: simple
number of clients: 50
number of threads: 1
maximum number of tries: 1
duration: 100 s
number of transactions actually processed: 187128
number of failed transactions: 0 (0.000%)
latency average = 16.006 ms
latency stddev = 9.939 ms
initial connection time = 444.292 ms
tps = 1879.385017 (without initial connection time)
```

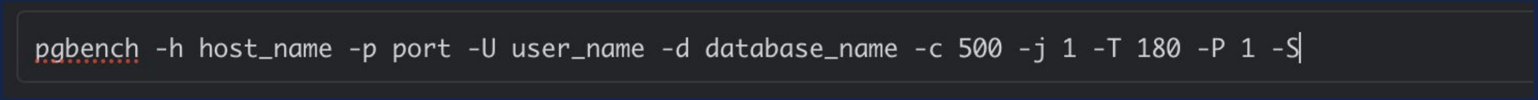
Without Pooling



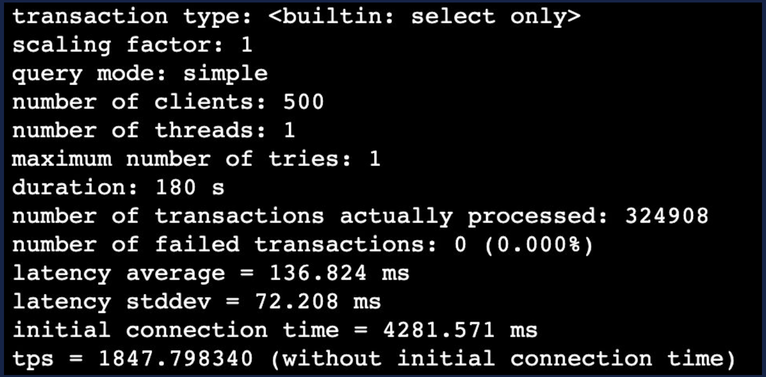
```
transaction type: <builtin: select only>
scaling factor: 1
query mode: simple
number of clients: 50
number of threads: 1
maximum number of tries: 1
duration: 100 s
number of transactions actually processed: 193742
number of failed transactions: 0 (0.000%)
latency average = 16.853 ms
latency stddev = 9.787 ms
initial connection time = 31.336 ms
tps = 1937.932331 (without initial connection time)
```

With Pooling

2. Pool size = 50 and client connections = 500

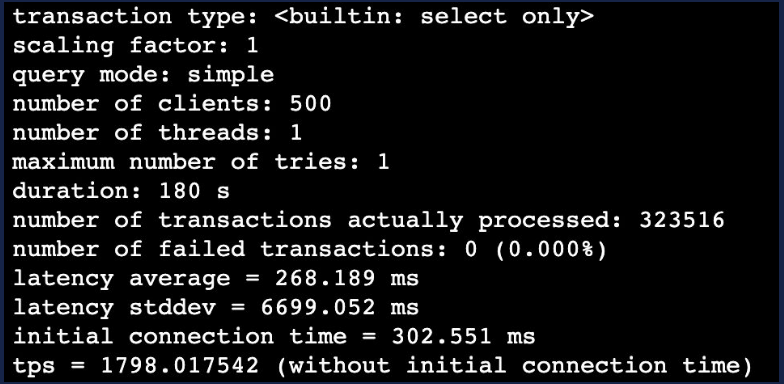


```
pgbench -h host_name -p port -U user_name -d database_name -c 500 -j 1 -T 180 -P 1 -S
```



```
transaction type: <builtin: select only>
scaling factor: 1
query mode: simple
number of clients: 500
number of threads: 1
maximum number of tries: 1
duration: 180 s
number of transactions actually processed: 324908
number of failed transactions: 0 (0.000%)
latency average = 136.824 ms
latency stddev = 72.208 ms
initial connection time = 4281.571 ms
tps = 1847.798340 (without initial connection time)
```

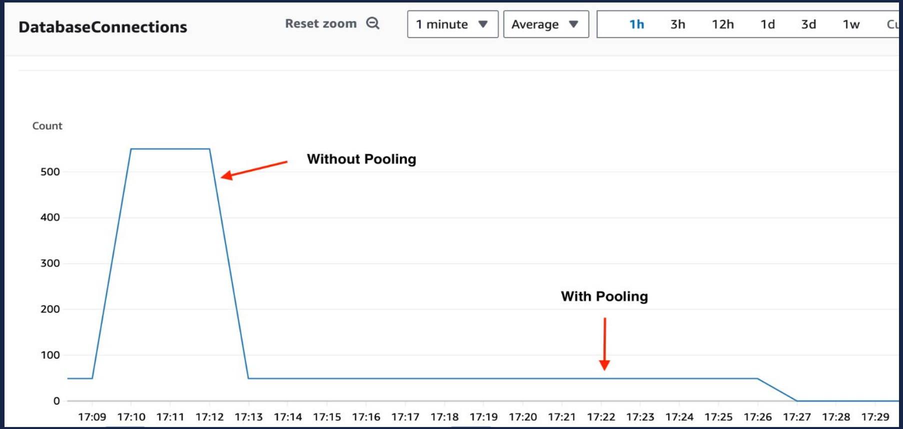
Without Pooling



```
transaction type: <builtin: select only>
scaling factor: 1
query mode: simple
number of clients: 500
number of threads: 1
maximum number of tries: 1
duration: 180 s
number of transactions actually processed: 323516
number of failed transactions: 0 (0.000%)
latency average = 268.189 ms
latency stddev = 6699.052 ms
initial connection time = 302.551 ms
tps = 1798.017542 (without initial connection time)
```

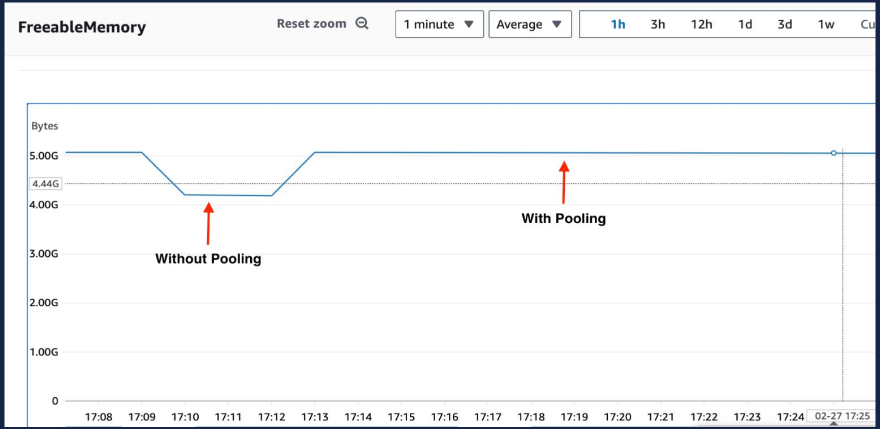
With Pooling

Metrics



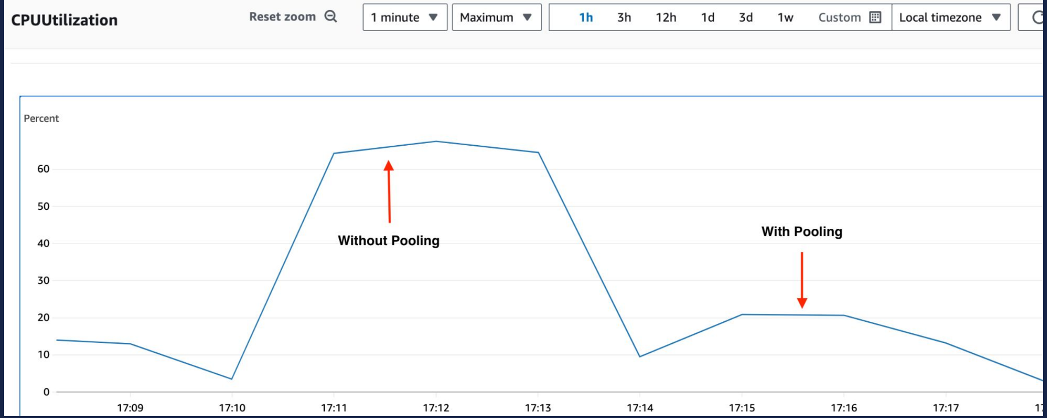
Without Pooling

With Pooling



Without Pooling

With Pooling



Without Pooling

With Pooling

7. Pgbouncer vs Pgpool - II

Feature	Pgbouncer	Pgpool-II
Resource Consumption	It uses only one process which makes it very lightweight.	If we require N parallel connections, this forks N child processes.
Types of pooling Supported	Transaction, Session and Statement	Only Session
High availability (HA)	No	Yes, supports failover and standby servers
Connection limiting	Per user, database, or pool	Overall number of connections and per-user
Load balancing	Not by default, but we can do by using external Load balancers	Yes, automatic read/write splitting
Management interface	Virtual database with statistics	GUI and detailed administration interface
Complexity	Simpler, easier to configure	More complex, requires fine-tuning

JFYI you can also look into other connection poolers like PgCat, Supavisor

8. Conclusion

1. **Faster response times:** Reuses connections, saving time on establishing and closing them.
2. **Reduced resource usage:** Saves resources on both application and database server.
3. **Improved scalability:** Handles more concurrent requests effectively.
4. **Simplified development:** Easier connection management for developers.
5. Can cause latency issues if not configured correctly

Thank You

SquadStack