Fearby Software

Developing since 1996

How to setup pooled MySQL connections in Node JS that don't disconnect

August 29, 2015 by Simon Fearby (http://fearby.com/article/author/simon-fearby/)

Using NodeJS/NGINX to power a custom API or website is quite simple. At some stage you will want to connect NodeJS to a MySQL database.

I have a dedicated Ubuntu server on Digital Ocean. You can easily deploy an SSD cloud server in 55 seconds for $5 a month. Sign up using my link and receive $10 in credit: https://www.digitalocean.com/?refcode=99a5082b6de5 (https://www.digitalocean.com /?refcode=99a5082b6de5).

Here is a quick sample of the minimum code needed to get a Node Server up with a API target of "/api/database/status". What better way to know if a database is up than to read a setting from a table to confirm it is up.

```
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());

var db_config = { host : 'localhost', user : 'your_db_username', password : 'your_db_password', datab
var connection = mysql.createConnection(db_config);

app.get('/api/database/status',function(req,res){
    var retvalSettingValue = "?";
    connection.query('SELECT SettingValue FROM your_status_table WHERE SettingKey =\'DatabaseStatus\'
        if (err) {
            var data = { "Time":"", "DatabaseStatus":"" };
            data["Time"] = (new Date()).getTime();
            data["DatabaseStatus"] = "Down";
            res.json(data);
        } else {
            var dbretval = rows[0].SettingValue;
            if (dbretval == 1 ) {
                var data = { "Time":"", "DatabaseStatus":"" };
                data["Time"] = (new Date()).getTime();
                data["DatabaseStatus"] = "Up";
```

This should return the following JSON data.

```
{"Time":1440835363892,"DatabaseStatus":"Up"}
```

The problem for me with the code above was that MySQL kept disconnecting after a few days. Adjusting timeouts, setting up keep alive cron jobs failed to keep the connection up. I then tried introducing the following code solve the disconnected MySQL connection.

```
function handleDisconnect() {
    console.log('handleDisconnect()');
    connection.destroy();
    connection = mysql.createConnection(db_config);
    connection.connect(function(err) {
        if(err) {
            console.log(' Error when connecting to db  (DBERR001):', err);
            setTimeout(handleDisconnect, 1000);
        }
    });

}
```

```
        console.log('Connection is asleep (time to wake it up): ', err);
        setTimeout(handleDisconnect, 1000);
        handleDisconnect();
    }
    });
```

I ended up swapping over to using pooled MySQL connections (https://github.com/felixge/node-mysql/) and this code has been working for days. Pooled connections reserves a number of connections ready for use in the background. You simply ask for a connection and when you are done you release it. The pooled connection closes and refreshed the connection and puts it at the end of the queue.

```
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());

var mysql_pool  = mysql.createPool({
  connectionLimit : 100,
  host            : 'localhost',
  user            : 'your_database_user',
  password        : 'your_database_password',
  database        : 'your_database'
});

app.get('/api/database/status',function(req,res) {
    console.log('API CALL: /api/database/status');
    var retvalSettingValue = "?";
    mysql_pool.getConnection(function(err, connection) {
        if (err) {
            connection.release();
            console.log(' Error getting mysql_pool connection: ' + err);
            throw err;
        }
        connection.query('SELECT SettingValue FROM your_database_table WHERE SettingKey =\'DatabaseSt
            if (err2) {
                var data = { "Time":"", "DatabaseStatus":"" };
                data["Time"] = (new Date()).getTime();
                data["DatabaseStatus"] = "Down";
                res.json(data);
            } else {
                var dbretval = rows[0].SettingValue;
                if (dbretval == 1 ) {
                    var data = { "Time":"", "DatabaseStatus":"" };
                    data["Time"] = (new Date()).getTime();
                    data["DatabaseStatus"] = "Up";
                    res.json(data);
                } else {
                    var data = { "Time":"", "DatabaseStatus":"" };
                    data["Time"] = (new Date()).getTime();
                    data["DatabaseStatus"] = "Down";
                    res.json(data);
                }
            }
            console.log(' mysql_pool.release()');
            connection.release();
        });
    });
});

```

To test the connections/throughput it's time to fire up more than 100 connections using siege (https://www.joedog.org/siege-home) over 60 seconds. I am using HTTPS connections so his would slow down the throughput a fair bit (but I am sticking with HTTPS for customer security/privacy). Read my guide here on adding a A+ level SSL cert to a digital ocean VM (http://fearby.com/article /adding-a-commercial-ssl-certificate-to-a-digital-ocean-vm).

```
#!/bin/bash
free -m

siege -b -c1 -t1M https://yourserver.com/api/database/status
cat /var/log/siege.log

free -m
```

Here are the results from a siege single client hitting the NodeJS/pooled connections for 60 seconds. I am able to get about **6,150** database reads in 60 seconds with no dropped connections.

```

** SIEGE 3.0.5
** Preparing 1 concurrent users for battle.
The server is now under siege...
Lifting the server siege...      done.

Transactions:                   6150 hits
Availability:                 100.00 %
Elapsed time:                  59.69 secs
Data transferred:               0.26 MB
Response time:                  0.01 secs
Transaction rate:             103.03 trans/sec
Throughput:                     0.00 MB/sec
Concurrency:                    0.97
Successful transactions:        6150
Failed transactions:               0
Longest transaction:            0.05
Shortest transaction:           0.00

FILE: /var/log/siege.log
```

2 siege clients hitting the server delivers **8,120** database reads with 0 dropped connections.

```
      Date & Time,  Trans,  Elap Time,  Data Trans,  Resp Time,  Trans Rate,  Throughput,  Concurrent,
2015-08-29 04:25:13,   8120,      59.83,           0,       0.01,      135.72,        0.00,        1.9
```

10 siege clients hitting the server delivers **16,383** database reads with 0 dropped connections.

```
      Date & Time,  Trans,  Elap Time,  Data Trans,  Resp Time,  Trans Rate,  Throughput,  Concurrent,
2015-08-29 04:29:50,  16383,      59.51,           0,       0.04,      275.30,        0.00,        9.8
```

20 siege clients hitting the server delivers 15440 database reads with 0 dropped connections. This looks like the max the server can handle (*257 delivered requests a second (19.7 concurrent connections)*).

```
      Date & Time,  Trans,  Elap Time,  Data Trans,  Resp Time,  Trans Rate,  Throughput,  Concurrent,
2015-08-29 04:33:06,  15440,      59.93,           0,       0.08,      257.63,        0.00,       19.7
```

My SQL, Linux Memory and Cache tweaks

I have tweaked my stock MySQL config a bit (*changed values below*)

```
...
key_buffer      = 32M
key_buffer_size = 32M
max_allowed_packet  = 16M
max-connect-errors             = 1000000

#max_connections        = 100
max-connections = 500
thread_stack        = 256K
thread_cache_size       = 8
myisam-recover         = FORCE,BACKUP
#table_cache            = 64
#thread_concurrency     = 10
...
query_cache_limit   = 1M
query_cache_size        = 0
...
expire_logs_days    = 10
max_binlog_size         = 100M
...
```

Some may notice a 20mb memory jump in memory usage after I benchmarked, I have set my ubuntu to use as much memory as possible and not release too soon. I also have a 4GB (https://help.ubuntu.com/community/SwapFaq) swap file (https://www.digitalocean.com/community/tutorials/how-to-add-swap-on-ubuntu-12-04) (that is ready in reserve but not used much).

- swappiness this control is used to define how aggressively the kernel swaps out anonymous memory relative to pagecache and other caches. Increasing the value increases the amount of swapping. The default value is 60.
- vfs_cache_pressure this variable controls the tendency of the kernel to reclaim the memory which is used for caching of VFS caches, versus pagecache and swap. Increasing this value increases the rate at which VFS caches are reclaimed.

/etc/sysctl.conf

```
vm.swappiness = 10
vm.vfs_cache_pressure = 50
vm.min_free_kbytes= 131072
```

More performance tweaks here: https://gist.github.com/dakull/5629740 (https://gist.github.com/dakull/5629740)

0 Comments fearby.com 1 Login

Recommend Share Sort by Best

Start the discussion…

Be the first to comment.

Subscribe Add Disqus to your site Add Disqus Add Privacy DISQUS