

This site uses cookies from Google to deliver its services, to personalise ads and to analyse traffic. Information about your use of this site is shared with Google. By using this site, you agree to its use of cookies.

[LEARN MORE](#) [GOT IT](#)

# Digital (dis)content

Julien Simon is a Principal Technical Evangelist at Amazon Web Services. He uses this blog to express personal opinions on digital co and whatever else keeps the adrenaline flowing

## Next talks

- [4/4 ComputerWorld.dk Summit \(Copenhagen\)](#)
- [12/4 AWS User Group \(Montpellier\)](#)
- [20/4 Open Tech Day \(Utrecht\)](#)
- [27/4 AWS Dev Day \(Lyon\)](#)
- [3/5 AWS Summit \(Stockholm\)](#)

## Follow me on:

- [Facebook](#)
- [Github](#)
- [LinkedIn](#)
- [Slideshare](#)
- [Twitter](#)
- [Youtube](#)

## Blog archive

- [2017](#) (4)
  - [2016](#) (23)
  - [2015](#) (23)
  - [2014](#) (2)
  - ▼ [2013](#) (28)
    - [October](#) (1)
    - [September](#) (2)
    - ▼ [August](#) (8)
      - [HOWTO: compiling ffmpeg + x264 + MP3 + AAC + Xvid ...](#)
      - [Arduino: LCD thermometer](#)
      - [Node.js, part 5.1: don't you C?](#)
      - [Node.js, part 5: Zero the hero!](#)
      - [Node.js, part 4: the Big Kahuna syslog!](#)
      - [Node.js + MongoDB, part 3: exit memcached, enter E...](#)
      - [Node.js + MongoDB, part 2: here comes memcached!](#)
      - [HOWTO AWS: mount S3 buckets from a Linux EC2 insta...](#)
  - [July](#) (9)
  - [June](#) (3)
  - [March](#) (2)
  - [February](#) (2)
  - [January](#) (1)
- [2012](#) (13)
  - [2011](#) (8)
  - [2010](#) (10)
  - [2009](#) (74)
  - [2008](#) (84)
  - [2007](#) (34)

Aug 3, 2013

## Node.js + MongoDB, part 2: here comes memcached!

Let's elaborate on the [Node.js + MongoDB](#) app. I lied: it's not really worth \$1 billion... yet. It surely will once we've added *memcached* to the mix ;)

Once again, we'll use a couple of EC2 instances running Ubuntu 12.04: one for the Node.js web server and one for the *memcached* server. MongoDB will still be served from MongoLab.

Start your instances and let's configure our *memcached* server first. Since it requires port 11211 to be open, we have to add a couple of rules for 11211/UDP and 11211/TCP in the security group attached to the instance.

Then, let's install *memcached*:

```
ubuntu@ip-10-234-177-74:~$ sudo apt-get install memcached
```

We also need to edit `/etc/memcached.conf` in order to set the `-l` parameter to the correct IP address (running *ifconfig eth0* will confirm the right one to use). In my case, it is 10.234.177.74.

Then, we need to restart *memcached*:

```
ubuntu@ip-10-234-177-74:~$ sudo service memcached restart
```

Now, let's go to the Node.js instance and check that we can access the *memcached* server:

```
ubuntu@ip-10-48-161-115:~$ echo stats|nc 10.234.177.74 11211
```

If you see a lot of stats like I do, you're good to go. If not, double-check the steps above (rules, config file, restart).

Now, let's install the *memcached* client for Node.js with *npm*, the Node.js package manager. There are several clients out there, *mc* looks pretty good and well-maintained :)

```
ubuntu@ip-10-48-161-115:~$ npm install mc
```

That's it. Now, let's write some code, yeah! Here's the idea:

- call the web server with a MongoDB ObjectId, e.g. `http://ec2-54-216-3-139.eu-west-1.compute.amazonaws.com:8080/?id=51e3ce08915082db3df32bf7`
- query the *memcached* server
- if we hit, job done!
- if we miss, query the MongoDB server and update the cache

This site uses cookies from Google to deliver its services, to personalise ads and to analyse traffic. Information about your use of this site is shared with Google. By using this site, you agree to its use of cookies.

LEARN MORE GOT IT

```
var ObjectId = require('mongodb').ObjectId;
var collection;

function onRequest(request, response) {
  // No id parameter --> do nothing
  // id=0 --> list all documents
  // id=SOME_OBJECT_ID --> show the 'x' value for this document
  var query = url.parse(request.url, true).query;
  console.log("Request received, id="+query.id);

  response.writeHead(200, {"Content-Type": "text/html"});
  response.write("<html>");

  // XXX Missing check: query.id must be a proper ObjectId, i.e. 12-byte hex value

  if (query.id == null) {
    // No id parameter --> do nothing
    response.write("Nothing to do, bleh");
    response.write("</HTML>");
    response.end();
  }
  else if (query.id == "0") {
    // id=0 --> list all documents
    collection.find().toArray(function (err, items) {
      console.log ("Finding all documents");
      for (var i = 0; i < items.length; i++) {
        response.write(JSON.stringify(items[i])+"<br>");
      }
      response.write("</HTML>");
      response.end();
    });
  }
  else {
    // id=SOME_OBJECT_ID --> show the 'x' value for this document
    // Check the cache first
    MemcacheClient.get(query.id, function(err, result) {
      if (!err) {
        // Key found, display value
        console.log("Cache hit, key="+query.id+", value="+result[query.id]);
        response.write("x="+result[query.id]);
        response.write("</HTML>");
        response.end();
      }
      else {
        // Key not found, fetch value from DB
        console.log("Cache miss, key "+query.id+" Querying...");
        collection.findOne({"_id": new ObjectId(query.id)}, function (err, item) {
          if (item == null) {
            response.write("Item does not exist, duh");
          }
          else {
            console.log("Item found: "+JSON.stringify(item));
            // Display value
            response.write(JSON.stringify(item));
            // Store value in cache with a 60 second TTL
            MemcacheClient.set(query.id, item.x, { flags: 0, exptime: 60}, function(err,
status) {
              if (!err) {
                console.log("Stored key="+query.id+", value="+item.x);
              }
              else {
                console.log("Couldn't store key="+query.id+", error="+err);
              }
            });
            response.write("</HTML>");
            response.end();
          }
        });
      }
    });
  }
}

// Connect to the memcached server with its private AWS IP address
MemcacheClient = new mc.Client("10.234.177.74");
MemcacheClient.connect(function() {
  console.log("Connected to memcache");
});

// Connect to the MongoDB server hosted at MongoLab
MongoClient.connect("mongodb://USER:PASSWD@ds051067.mongolab.com:51067/mongolab
-test", function(err, db) {
  if (!err) {
    console.log("Connected to database");
    collection = db.collection("collection1");
  }
  else {
    console.log("Dude, where's my DB?");
  }
});

// Start the web server on port 8080 and wait for incoming requests
http.createServer(onRequest).listen(8080);
console.log("Web server started");
```

Alright, let's run this and hit it with some requests :

```
Mac:~ julien$ curl http://ec2-54-216-3-139.eu-west-1.compute.amazonaws.com:8080/?id=0
{"_id":"51e3ce08915082db3df32bf0","x":1}
{"_id":"51e3ce08915082db3df32bf1","x":2}
{"_id":"51e3ce08915082db3df32bf2","x":3}
{"_id":"51e3ce08915082db3df32bf3","x":4}
{"_id":"51e3ce08915082db3df32bf4","x":5}
{"_id":"51e3ce08915082db3df32bf5","x":6}
{"_id":"51e3ce08915082db3df32bf6","x":7}
{"_id":"51e3ce08915082db3df32bf7","x":8}
{"_id":"51e3ce08915082db3df32bf8","x":9}
{"_id":"51e3ce08915082db3df32bf9","x":10}
{"_id":"51e3ce08915082db3df32bfA","x":11}
```

This site uses cookies from Google to deliver its services, to personalise ads and to analyse traffic. Information about your use of this site is shared with Google. By using this site, you agree to its use of cookies.

[LEARN MORE](#) [GOT IT](#)

```
{ "_id": "51e3ce08915082db3df32bff", "x": 16 }
{ "_id": "51e3ce08915082db3df32c00", "x": 17 }
{ "_id": "51e3ce08915082db3df32c01", "x": 18 }
{ "_id": "51e3ce08915082db3df32c02", "x": 19 }
{ "_id": "51e3ce08915082db3df32c03", "x": 20 }
{ "_id": "51e3ce08915082db3df32c04", "x": 21 }
{ "_id": "51e3ce08915082db3df32c05", "x": 22 }
{ "_id": "51e3ce08915082db3df32c06", "x": 23 }
{ "_id": "51e3ce08915082db3df32c07", "x": 24 }
{ "_id": "51e3ce08915082db3df32c08", "x": 25 }
```

```
Mac:~ julien$ curl http://ec2-54-216-3-139.eu-west-1.compute.amazonaws.com:8080/?id=51e3ce08915082db3df32bfc
{ "_id": "51e3ce08915082db3df32bfc", "x": 13 }
```

#### Console output:

```
Request received, id=51e3ce08915082db3df32bfc
Cache miss, key 51e3ce08915082db3df32bfc. Querying...
Item found: { "_id": "51e3ce08915082db3df32bfc", "x": 13 }
Stored key=51e3ce08915082db3df32bfc, value=13
```

#### Memcached stats:

```
ubuntu@ip-10-234-177-74:~$ echo stats|nc 10.234.177.74 11211|grep [s,g]et
STAT cmd_get 1
STAT cmd_set 1
STAT get_hits 0
STAT get_misses 1
```

#### Let's try the same request again (within 60 seconds!): +1 get, +1 hit!

```
Request received, id=51e3ce08915082db3df32bfc
Cache hit, key=51e3ce08915082db3df32bfc, value=13
```

```
STAT cmd_get 2
STAT cmd_set 1
STAT get_hits 1
STAT get_misses 1
```

#### And 60 seconds later, the memcached item should have disappeared: +1 get, +1 miss, +1 set

```
Cache miss, key 51e3ce08915082db3df32bfc. Querying...
Item found: { "_id": "51e3ce08915082db3df32bfc", "x": 13 }
Stored key=51e3ce08915082db3df32bfc, value=13
```

```
STAT cmd_get 3
STAT cmd_set 2
STAT get_hits 1
STAT get_misses 2
```

Pretty cool, huh? A basic Node.js + *memcached* + MongoDB app in less than 100 lines of code, comments and logging included.

However, the really great stuff is what you DON'T see:

- Node.js automatically handles asynchronous requests and callbacks,
- Building a *memcached* cluster will hardly have any impact on the application code,
- Building a MongoDB cluster (replica sets, sharding) will be completely transparent.
- And of course, you can easily create your own pool of Node.js servers and load balance them.

That's A LOT of scalability for free as far as the application developer is concerned.

Food for thought... Something tells me this isn't the last post on these topics. I hope you're enjoying this as much as I am. Time for a drink, I'm exhausted. Cheers!



[G+1](#) Recommend this on Google

Tags: [aws](#), [code](#), [open source](#), [virtualization](#)

No comments:

Post a Comment

This site uses cookies from Google to deliver its services, to personalise ads and to analyse traffic. Information about your use of this site is shared with Google. By using this site, you agree to its use of cookies.

[LEARN MORE](#) [GOT IT](#)

Comment as: Gene (Google) ▾

Sign out

Publish

Preview

☐ Notify me

[Newer Post](#)[Home](#)[Older Post](#)

Subscribe to: [Post Comments \(Atom\)](#)