Applifier and Node.JS

Juho Mäkinen
Software Architect at Applifier

Frontend Finland
11.5.2011

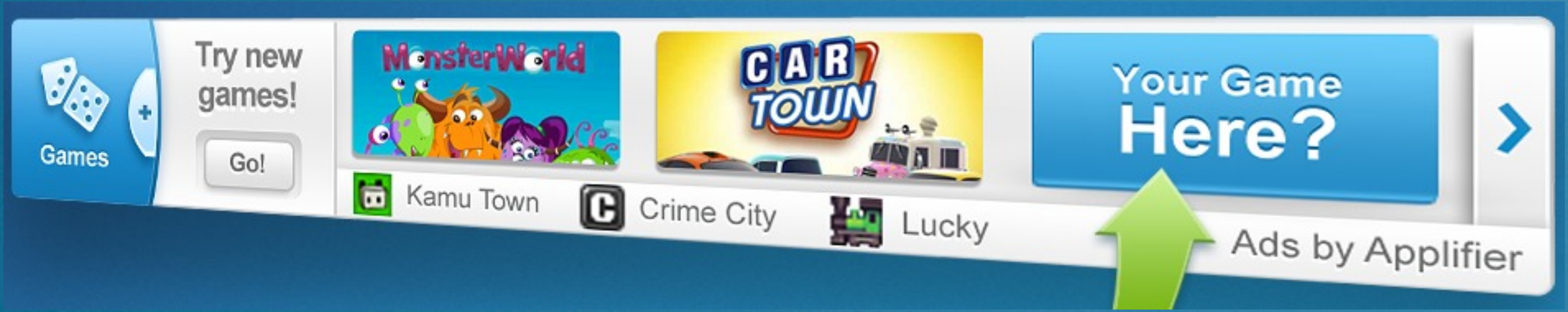
Agenda

1. What Applifier does?
2. Our Delivery Platform
3. Tips for Node.JS
4. Questions

What does Applifier do?

Juho Mäkinen
Software Architect

Cross-promo network



- Key product is a cross-promotion network for games.
- Operates mostly inside Facebook
- Helps game publishers to get more traffic

applifier

Cross-promo network

- About 1000-1200 requests per second.
- 700M ads served per day
- Content delivery network delivers 6TB of data per day. Peaks at 1 Gbps

Our Delivery Platform

Using Javascript as a platform

- It's possible to write entire application stack with Javascript
- From browser all the way to the datastore
- Imagine that you can send a JS object from the browser to the backend, store it to the database and retrieve it later without any unnecessary transformations.

Why Javascript

- Fast development cycle
- Easy data transformations as the storage and delivery is always JSON
- It’s possible to share code between browser and the backend

Our stack

- Javascript client library running inside users browser
- Uses HTTP to request JSON from the delivery backend
- Delivery backend implemented with Javascript on top of Node.JS
- MongoDB JSON document storage as db

applifier

Why Node.JS

- Perfect for low latency, high IO applications
  - Load balancing, API backend, comet / longpoll servers, streaming...
- It's easy to process the request, close the client socket and continue some tasks after the request has been processed.
- It's easy to cache data between requests

Where Node isn’t good

- Harder to code (than say PHP).
- Need to take perfect care of all resources to avoid leaks.
- Less libraries and bindings because everything must be asynchronous and non-blocking.

Our Node.js backend

- Uses *express* and *connect* middlewares for HTTP server handling.
- node-mongodb-native for MongoDB database connection. Except that we don't use the native part from it due to bugs.
- Few native extensions like geoip library
- 2/3 of the lines are actual code, 1/3 are tests

Why MongoDB

- Document oriented database
- Stores JSON documents which can be indexed on arbitrary keys inside documents.
- Fast and scales well
- Good drivers for Node.JS
- All operations (queries, scripts etc) are in JS

MongoDB example

```
> var doc = { name : "Garo", company : "Applifier" };

> db.test.save(doc);

> db.test.find({name : "Garo"});
{ "_id" : ObjectId("4dca539f422946adf7afe034"),
"name" : "Garo", "company" : "Applifier" }
```

Tips for Node.JS

What have we learned?

Buy a mac

- We can run our entire development environment in our mac laptops.
  - node
  - mongodb
  - apache, php (if you wish...)

Watch out for leaks

- The node server stays intact between requests.
- Thus resources, memory and database connections can easily leak.
- Db connection leaks are most common.
    - Our testing framework and database api helps detecting and avoiding them

applifier

Detect socket leaks

- You can use “netstat -np” to display socket connections to the node process.
  - Run a bunch of requests to the node and then look if the database connections are closed correctly.

Detect resource leaks

- Leaked sockets, database connections or objects stored into scopes might leak memory.
- Run node with "--expose-gc --trace_gc" to get memory consumption data.
  - Beware, the GC makes this hard to read
    - Run 100000 queries and see that the memory consumption is still about the same.

Clustering

- Node is single threaded
- Using multiple cores needs additional effort. There are two good options
  - Spawn multiple instances, one for each core and use load balancer to drive traffic into them. "Forever" makes this easy. https://github.com/indexzero/forever

Clustering

- Another option: Use Node Cluster (http://learnboost.github.com/cluster/) to fork worker threads.

Test Driven Development

- The nature of Javascript makes it easy to write mock objects
- Because your code is split into short callback functions, it's easy to test them individually with a unit testing framework.
- We use https://github.com/caolan/nodeunit

Avoid try..catch blocks

- As node is event driven, you use a lot of callbacks.
- try..catch blocks doesn’t behave well with callbacks and they’re slow.
- Stick with the node callback notation where you pass error as first parameter and handle errors this way.

Use async.js library

- Makes easier to write asynchronous code and makes it look prettier.
- Helps avoiding nesting callbacks
- https://github.com/caolan/async

Where to get modules?

- npm stands for “node package manager”
    - It’s your apt-get / CPAN / yum for all your node module needs. Learn it.
- Github. Full of node related modules.

Avoind anonymous functions

- Give all your functions a name so that you can get better stacktraces.
- fs.rename(p1, p2, function **rename_cb**() {
  // callback name is rename_cb
  // instantly better stacktrace upon error
  }

Log everything

- Log all your data and store it somewhere when you can query it.
- Use some form of metrics system to trace counters and to get history graphs out of them.
- All this helps debugging, performance tuning and diagnostics.

Log everything

- We use Scribe to feed performance and log data from our machines into a hadoop cluster. https://github.com/facebook/scribe
  - Hadoop is used to crunch the data into usable metrics.
- There's also tools for getting realtime statistics into usable graphs.
  - Zabbix, statsd / graphite

Questions?

Questions?

ps. we are hiring

http://careers.applifier.com

Web developers, frontend developers, Javascript gurus...