THE HAPPY PATH

MIGRATION STRATEGIES FOR NODE.JS

Brian Anderson | @Brianmatic

Nic Jansma | @NicJ

Jason Sich | @JaSich

PHASE 1: BUILDING THE API

Goal: Create a Node.js API server that interacts with Amazon S3, and update the AngularJS UI to associate documents with tasks

PHASE 1: STEPS

1. Get up and running with Node...
2. Bootstrap an express project:
   ```
   > express node-server
   ```
3. Add a few simple REST routes for document creating, listing, retrieval and removal.
4. Integrate with Amazon's SDK for JavaScript:
   ```
   > npm install aws-sdk
   ```
5. Update the AngularJS application to point to the Node.js API server.
6. Hook your new node server into IIS server using iisnode.

PHASE 1: REST API

We want to create the following REST API routes:

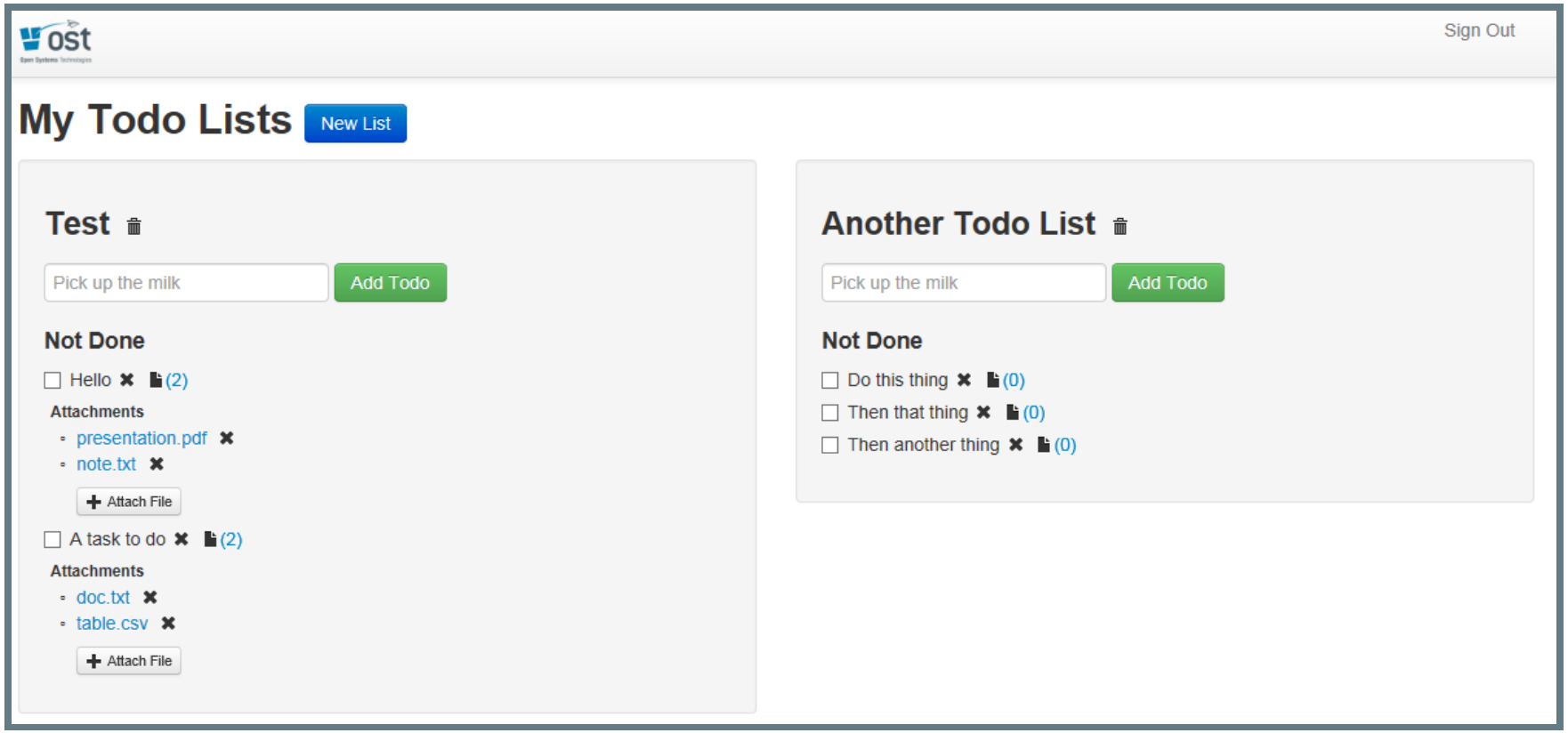
1. `GET /api/todo/:id/files`
   Get all of the file names for a todo
2. `GET /api/todo/:id/file/:name`
   Get a specific file
3. `DELETE /api/todo/:id/file/:name`
   Delete a specific file
4. `POST /api/todo/:id/files`
   Upload (overwrite) a file

PHASE 1: CODE

```
// imports
var http = require('http'), fs = require('fs'),
    express = require('express'), AWS = require('aws-sdk');

// configure AWS
AWS.config.loadFromPath('./aws.json');
var s3 = new AWS.S3();

// startup the HTTP server
app = express();
var httpServer = http.createServer(app);
httpServer.listen(80);

// one of the routes
app.get('/api/todo/:id/files/:name', function(req, res) {
    s3.getObject({
        Bucket: 'glsec-2014',
        Key: 'todos/' + req.params.id + '/' + req.params.name + '.txt'
    }, function(err, data) {
        if (err || !data) { return res.send(500, err); }
        var buff = new Buffer(data.Body, "binary");
        res.send(buff);
    });
});
```

This is a simplified (but runnable) version of the code.

PHASE 1: DEMO



phase1.foofactory.net

PHASE 2: BUILDING THE ADMIN INTERFACE

Goal: Create an Admin interface so the system can be monitored in real-time.

PHASE 2: STEPS

1. Create a new AngularJS view for Admins.
2. Add Socket.IO to the node server and AngularJS client.
3. Add Socket.IO events for log events, API hits and periodic server statistics.
4. Use the Rickshaw JavaScript charting library to display real-time charts.

PHASE 2: CODE

Server

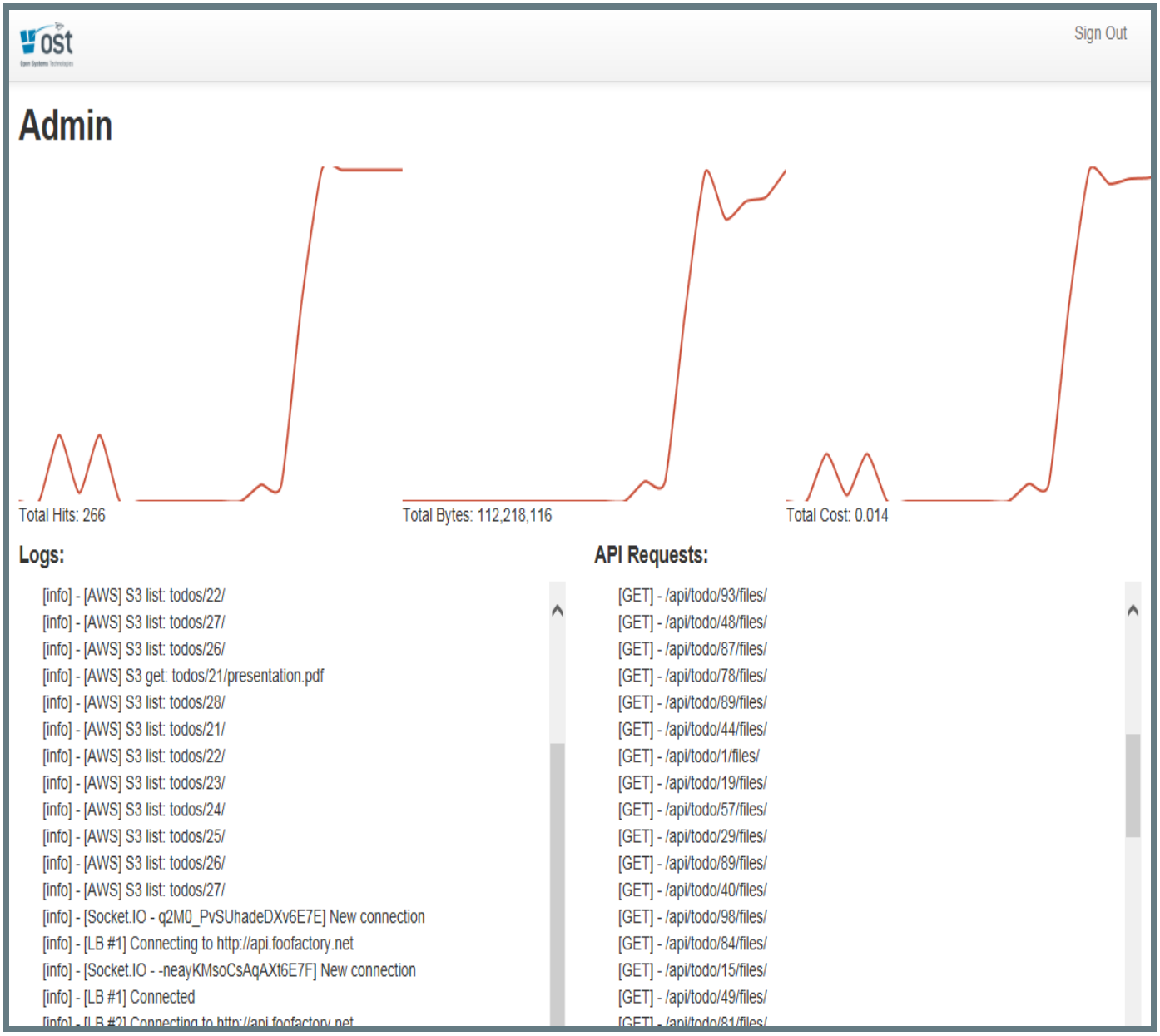
```
// ... continued from Phase 1 server code ...
var socketIo = require('socket.io');

var sockets = [];

io = socketIo.listen(httpServer);
io.sockets.on('connection', function(socket) {
    sockets.push(socket);

    socket.emit('log', { msg: 'Hello' });
});
```

Client

```
<script src="socket.io.js"></script>
<script>
// ... AngularJS client code ...
var socket = io.connect('http://api.foofactory.net');
socket.on('log', function (data) {
    // log message
});
</script>
```

This is a simplified (but runnable) version of the code.

PHASE 2: DEMO


Sign Out
Admin
Total Hits: 266
Total Bytes: 112,218,116
Total Cost: 0.014
Logs:
[info] - [AWS] S3 list: todos/22/
[info] - [AWS] S3 list: todos/27/
[info] - [AWS] S3 list: todos/26/
[info] - [AWS] S3 get: todos/21/presentation.pdf
[info] - [AWS] S3 list: todos/28/
[info] - [AWS] S3 list: todos/21/
[info] - [AWS] S3 list: todos/22/
[info] - [AWS] S3 list: todos/23/
[info] - [AWS] S3 list: todos/24/
[info] - [AWS] S3 list: todos/25/
[info] - [AWS] S3 list: todos/26/
[info] - [AWS] S3 list: todos/27/
[info] - [Socket.IO - q2M0_PvSUhadeDXv6E7E] New connection
[info] - [LB #1] Connecting to http://api.foofactory.net
[info] - [Socket.IO - -neayKMsoCsAqAXt6E7F] New connection
[info] - [LB #1] Connected
API Requests:
[GET] - /api/todo/93/files/
[GET] - /api/todo/48/files/
[GET] - /api/todo/87/files/
[GET] - /api/todo/78/files/
[GET] - /api/todo/89/files/
[GET] - /api/todo/44/files/
[GET] - /api/todo/1/files/
[GET] - /api/todo/19/files/
[GET] - /api/todo/57/files/
[GET] - /api/todo/29/files/
[GET] - /api/todo/89/files/
[GET] - /api/todo/40/files/
[GET] - /api/todo/98/files/
[GET] - /api/todo/84/files/
[GET] - /api/todo/15/files/
[GET] - /api/todo/49/files/

PHASE 3: SCALING

Goal: Allow the system to be easily scaled by Admins.

PHASE 3: STEPS

1. Add buttons for Admins to increase or decrease the number of Node.js API server instances (enterprise or cloud).
2. Add a new controller that will manage internal VMs, Amazon EC2 or Azure Node.js load-balanced instances.
3. Allow Admins to monitor the number of instances.
4. Have all Node.js API server instances communicate with the master server, sending stats, logs and API hits in real-time.

For this phase, we created mock instances that pretend to see user activity.

PHASE 3: CODE

Load-Balanced Instances

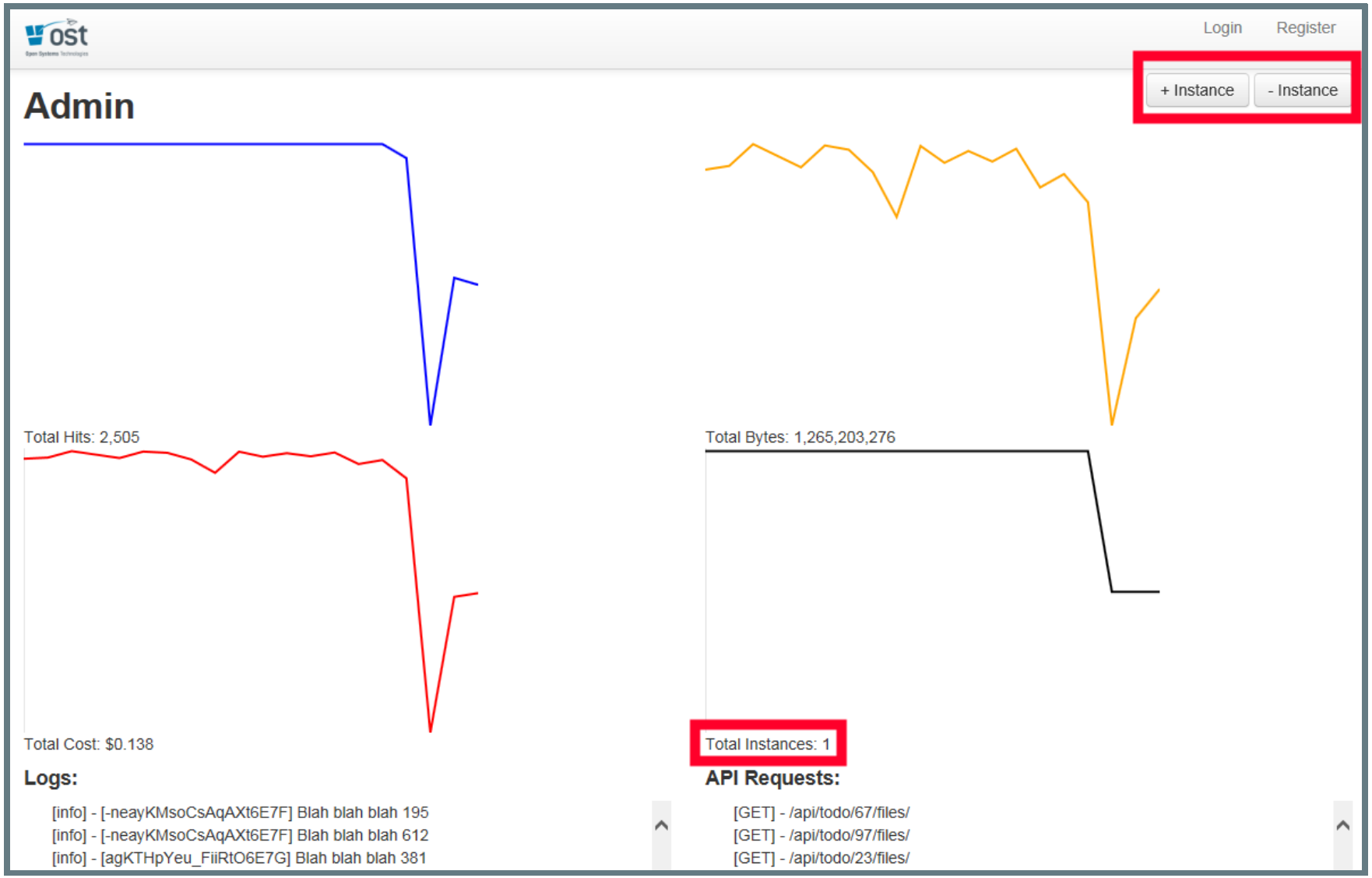
```
// workerInstance.js: Server instances also connect
// to master API server via Socket.IO
var socketIoClient = require('socket.io-client');

var io = socketIoClient.connect('http://api.foofactory.net');

client.on('connect', function() {
    client.emit('log', { msg: 'I connected' });
});
```

Master Server

```
// ... continued from Phase 2 server code ...

// collect log events from other server instances
socket.on('log', function(data) {
    // repeat log to all Admin clients
    for (var i = 0; i < sockets.length; i++) {
        sockets[i].emit('log', data);
    }
});
```

This is a simplified version of the code.

PHASE 3: DEMO



phase3.foofactory.net