THE HAPPY PATH

MIGRATION STRATEGIES FOR NODE.JS

Brian Anderson | @Brianmatic

Nic Jansma | @NicJ

Jason Sich | @JaSich

INTRODUCTION

1. OST Todos and NDriven
2. What is Node.js?
3. What is the most popular stack for building web based Node.js apps?
4. Why are large companies adopting it?
5. Why do we care? Why should you care?
6. Show me

OST TODOS

We are going to use this app for our migrations demos.

It is a reference sample application that we built and released open source last year at GLSEC.

WHAT IS NODE.JS?

Node.js is a software platform for scalable server-side and networking applications. Node.js applications are written in JavaScript, and can be run within the Node.js runtime on Windows, Mac OS X and Linux with no changes.

TECHNICAL DETAILS

- Node.js runs **JavaScript**, using Google's V8 runtime (C++).
- Node.js is event-driven, asynchronous, single threaded and **non-blocking**, which makes it able to scale more efficiently than thread-per-connection models (eg. ASP).

- **Node.js =** base .Net Framework and runtime
- **Modules =** libraries like .Net namespace assemblies
- **NPM =** package manager like Nuget

MOST POPULAR STACK FOR WEB APPS?

MEAN

1. **M**ongoDB - Document-based database server
2. **E**xpress - Web framework for Node.js
3. **A**ngular - JavaScript MVW framework
4. **N**ode.js - Server side JavaScript

ENTERPRISE PROCESS READY

- **Grunt, Gulp =** build tool (MSBuild)
- **Jenkins, Strider =** continuous integration (TeamCity, Bamboo or Cruise Control still work)
- **NodeUnit, Jasmine, Mocha, Vows =** unit testing (nUnit)
- **Bower =** front end package manager by Twitter

MAJOR COMPANIES ARE ADOPTING THESE TECHNOLOGIES

PAYPAL

*" We are moving **every product & every site within PayPal to Node**. We started the journey by using it as **prototyping framework**... by end of 2014 we hope to have **all major experiences** redesigned, and rewritten on Node.*

*We are seeing big scale gains, **performance boosts** and big developer productivity."*

- Built almost twice as fast with fewer people
- 33% fewer lines of code
- 40% less files

Bill Scott - http://www.quora.com/Node-js/What-companies-are-using-Node-js-in-production

DOW JONES

*"The simple truth is Node has **reinvented** the way we create websites. Developers build critical functionality in **days, not weeks**."*

Michael Yormark

WSJD Product Operations

More examples here: http://nodejs.org/industry/

MORE EXAMPLES

- LinkedIn
- eBay
- Yahoo
- Walmart
- Supported by Microsoft (Visual Studio and Azure).
- The list is long and keeps growing...

WHY SHOULD YOU CARE?

- **Less training, less code**: One language on the client and server.
- **Less abstraction**: simplifies layers and removes need for ORM if you think NoSQL.
- **Faster prototyping** and bootstrapping.
- **Modern** and makes you think in terms of modern web apps.
- A **large community** with a lot of active development. These technologies seem to be beloved by developers.

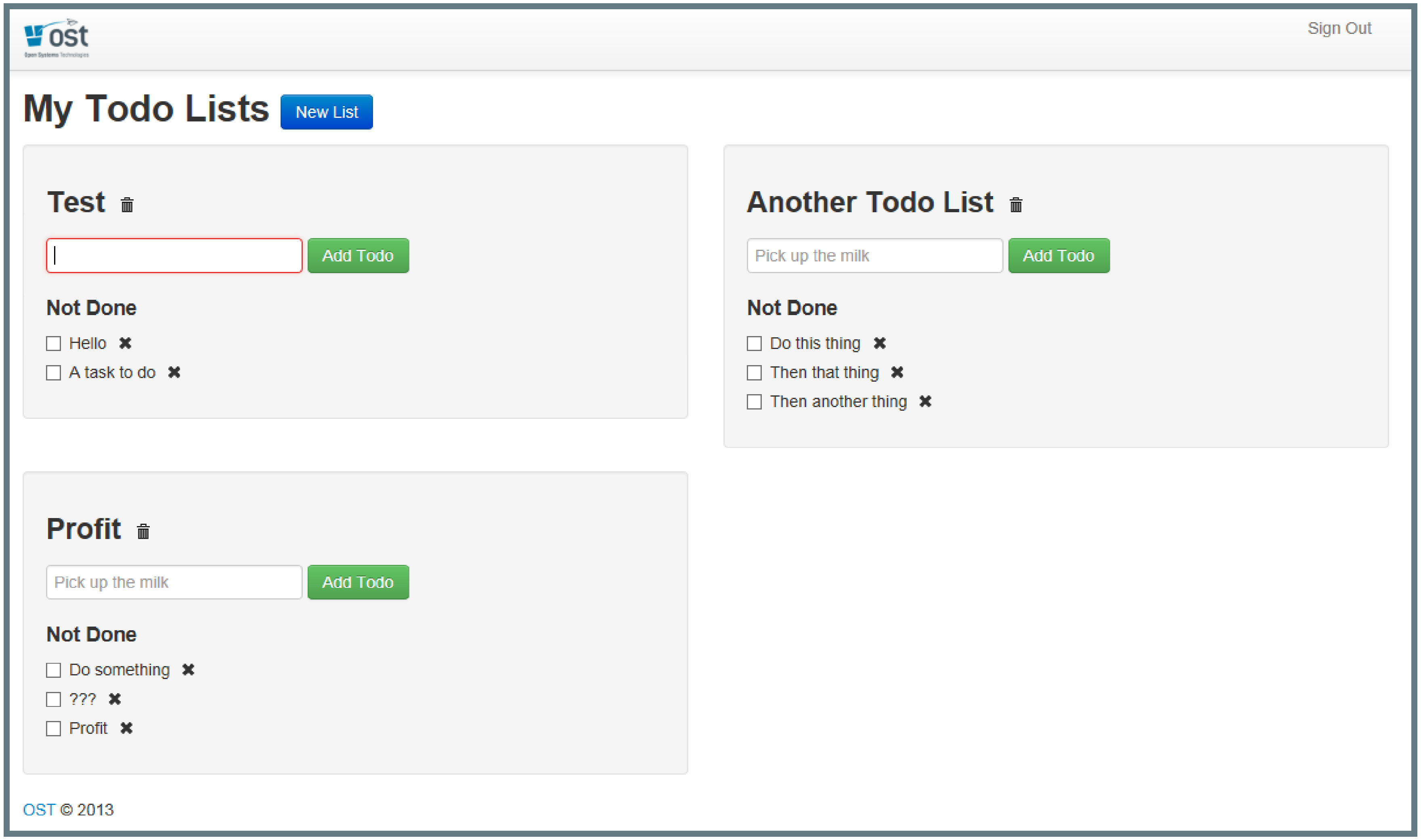
SHOW ME

Take it over Nic and Jason

THE SCENARIO

- You are a full-stack developer working at the *FooFactory*.
- The FooFactory has an internal task management system based off OST Todos (ASP.NET MVC 4, AngularJS, ndriven).
- Your boss, *BossFoo*, is interested in rapidly prototyping new features for the task management system as it is becoming an integral part of the FooFactory's daily operations.
- BossFoo has read about Node.js and is interested in possibly using it to speed up and unify FooFactory's web development stacks.

OST TODOS DEMO


Sign Out

My Todo Lists New List

Test

Add Todo

Not Done

- ☐ Hello ✖
- ☐ A task to do ✖

Another Todo List

Pick up the milk Add Todo

Not Done

- ☐ Do this thing ✖
- ☐ Then that thing ✖
- ☐ Then another thing ✖

Profit

Pick up the milk Add Todo

Not Done

- ☐ Do something ✖
- ☐ ??? ✖
- ☐ Profit ✖

OST © 2013

phase0.foofactory.net

BOSSFOO'S REQUIREMENTS

BossFoo wants you to add the following features to the *Todos* application:

- A **document storage** and retrieval system where users can attach documents to their tasks.
- An **admin interface** that can monitor the system in real-time to show its load and operational costs.

YOUR CONSTRAINTS

However, you are constrained by the following criteria:

- BossFoo is under a **time and budget crunch** to get these features implemented and would prefer to see a prototype of how it will work before he invests in big hardware to run it full time.
- There is **no short-term budget for dedicated hardware** or storage until it has been proven to work, so an on-demand infrastructure-as-a-service (IaaS) should be used.
- The prototype has to **integrate seamlessly** into the existing *Todos* application.

THE PATH TO NODE.JS

You carefully consider what you will need to prototype the new features:

- A **document storage system** that will scale gracefully as usage increases.
- A **new API** that allows for document listing, creation, retrieval and deletion.
- An **update to the *Todos* AngularJS application** that will integrate with the new API.

DOCUMENT STORAGE

Since you have no hardware budget, you choose to use Amazon S3, a cloud-based file system that scales seamlessly.

API - THE CONTENDERS

There are at least two stacks you could use to build the API:

- **C# / ASP.net MVC**: Extend the existing MVC app, add data models, build a new API, update all the layers of the onion, integrate with S3, update the AngularJS UI.
- **Node.js**: Add a new Node.js API server, have it integrate with S3, update the AngularJS UI.

AND THE WINNER IS... NODE.JS

- Node.js is great for **prototyping** since there is no build process, JavaScript is a dynamic language, server and client code look the same, etc.
- Amazon S3 introduces a high latency link. Node.js is **non-blocking** so it can scale better.
- Node.js and Socket.IO make it easy to build **real-time** applications.

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:

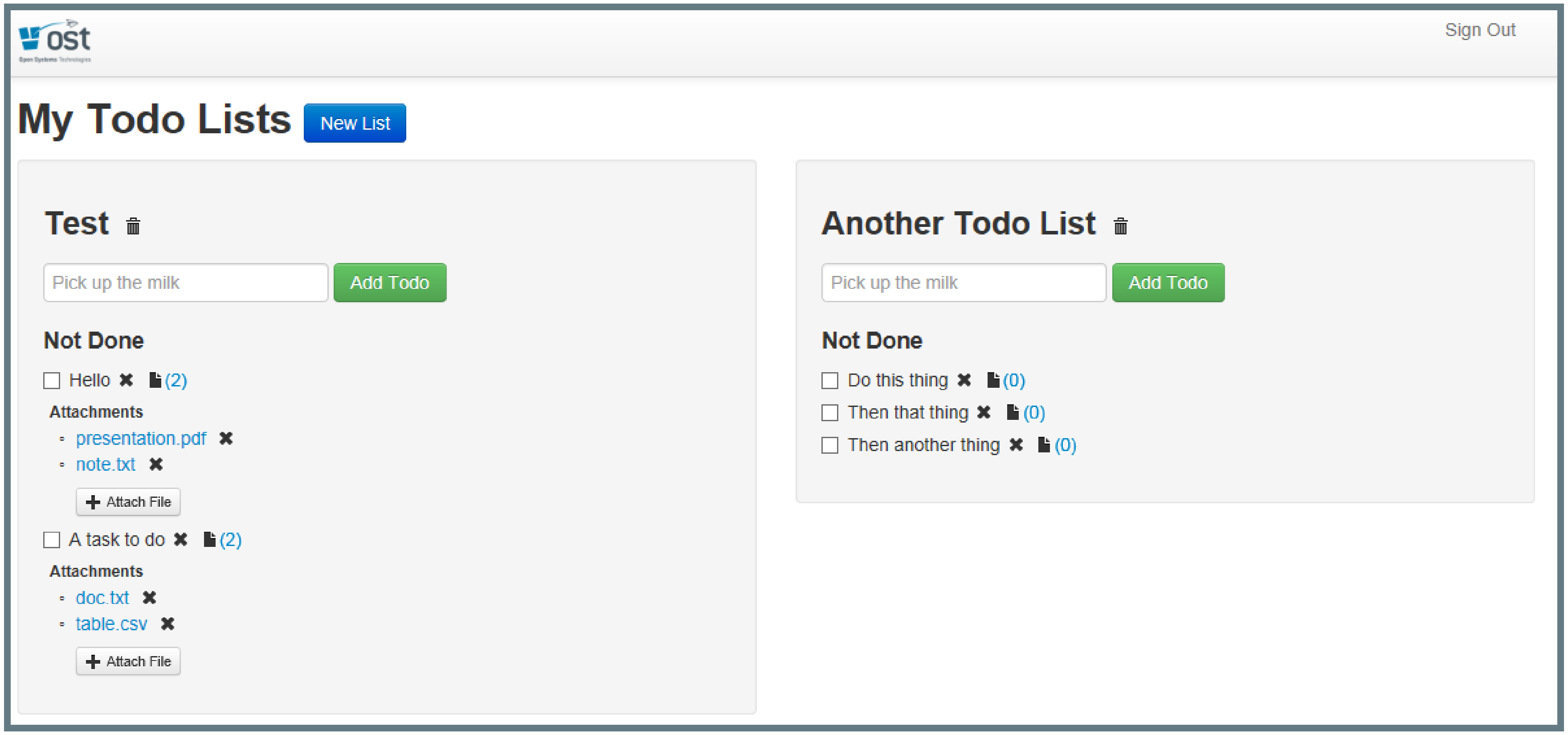
1. `GET /api/todo/:id/files`
   Get all of the file names for a todo
2. `GET /api/todo/:id/file/:name`
   Get a specific file
3. `DELETE /api/todo/:id/file/:name`
   Delete a specific file
4. `POST /api/todo/:id/files`
   Upload (overwrite) a file

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



Sign Out

My Todo Lists New List

Test

Pick up the milk Add Todo

Not Done

- [ ] Hello ✖ (2)

Attachments

- presentation.pdf ✖
- note.txt ✖

Attach File

- [ ] A task to do ✖ (2)

Attachments

- doc.txt ✖
- table.csv ✖

Attach File

Another Todo List

Pick up the milk Add Todo

Not Done

- [ ] Do this thing ✖ (0)
- [ ] Then that thing ✖ (0)
- [ ] Then another thing ✖ (0)

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

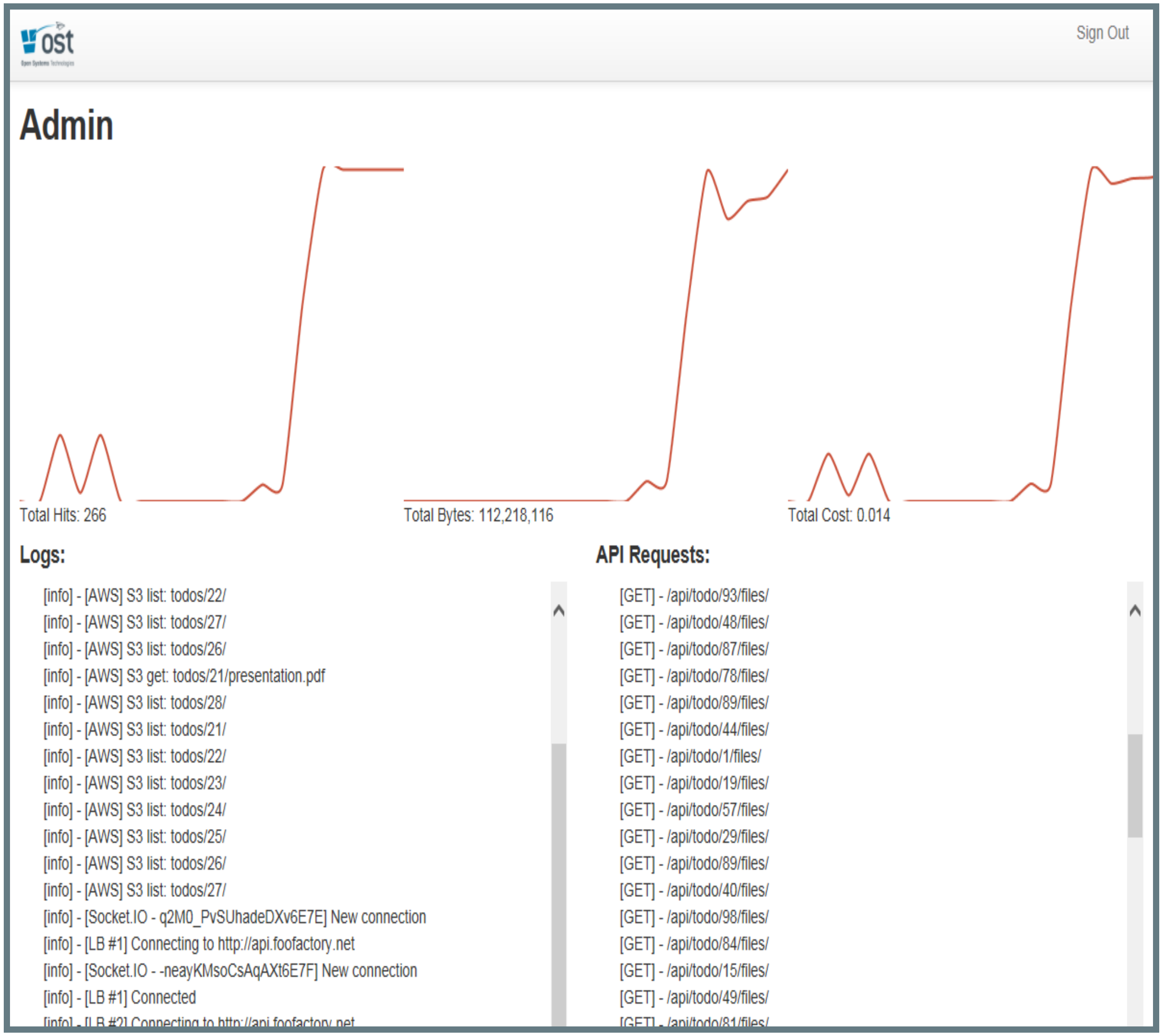
```
// ... continued from Phase 1 server code ...
var socketIo = require('socket.io');

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



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
[info] - [LB #2] Connecting to http://api.foofactory.net
```

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/
[GET] - /api/todo/81/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

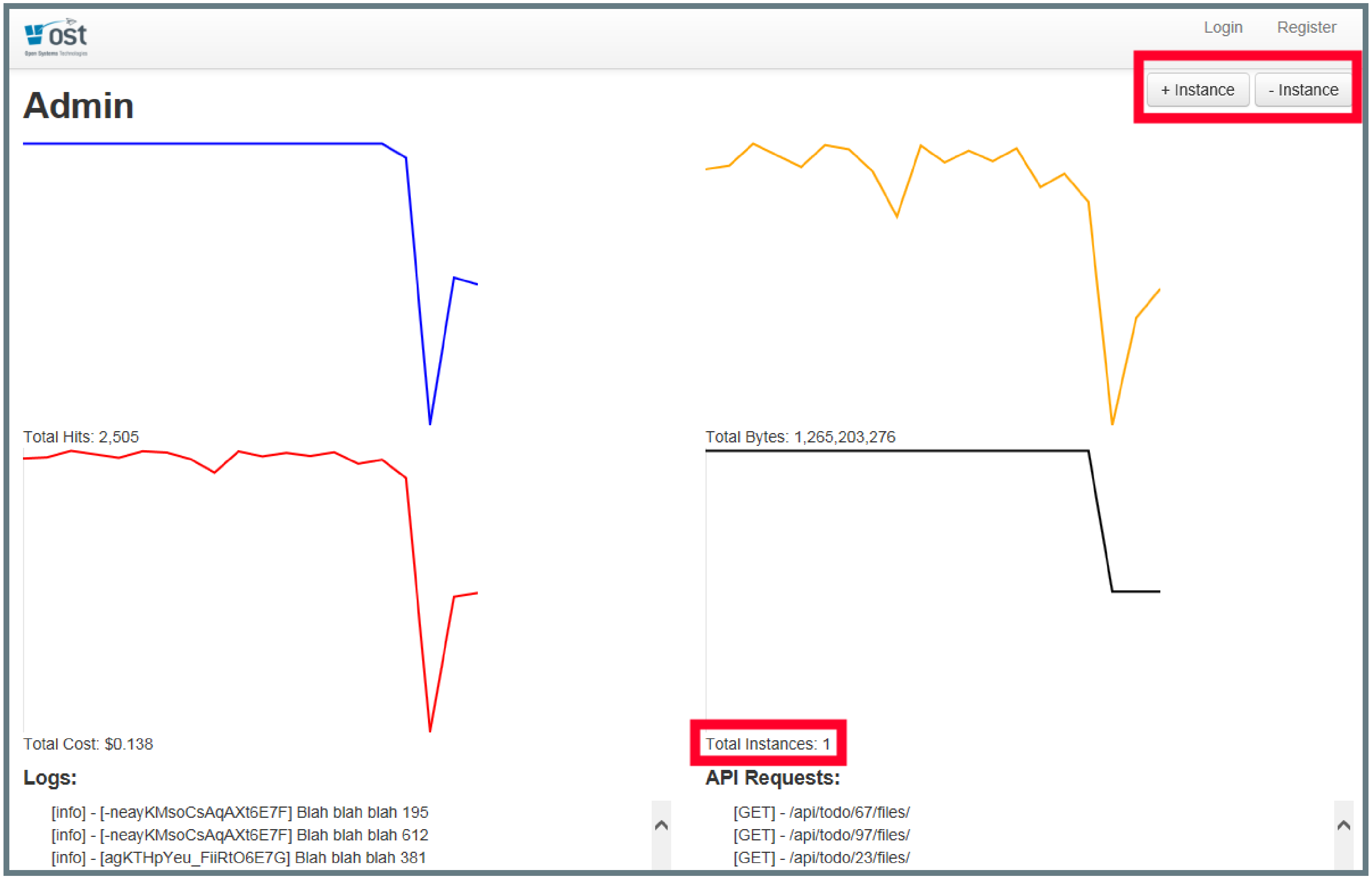
```
// workerInstance.js: Server instances also connect
// to master API server via Socket.IO
var socketIoClient = require('socket.io-client');

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



Admin

Total Hits: 2,505

Total Bytes: 1,265,203,276

Total Cost: $0.138

Total Instances: 1

Logs:

```
[info] - [-neayKMsoCsAqAXt6E7F] Blah blah blah 195
[info] - [-neayKMsoCsAqAXt6E7F] Blah blah blah 612
[info] - [agKTHpYeu_FiiRtO6E7G] Blah blah blah 381
```

API Requests:

```
[GET] - /api/todo/67/files/
[GET] - /api/todo/97/files/
[GET] - /api/todo/23/files/
```

phase3.foofactory.net

PHASE 4: REWRITING THE ASP.NET MVC API

Goal: For comparison, migrate all of the existing ASP.net MVC API to Node.js.

PHASE 4: STEPS

1. Create a new REST API in the Node.js server that mimics the ASP.NET MVC REST API.
2. Simplify by using a NoSQL solution such as MongoDB for todo storage.

PHASE 4: REST API

Leave the client alone. Migrate the following REST APIs to Node.js:

1. `GET /api/todolist/:id/todos`: Get todos for a list
2. `POST /api/todolist/:id/todos`: Post a todo to a list
3. `GET /api/todolists`: Get all of the todos for the user
4. `GET /api/todolist/:id`: Get a specific todolist
5. `DELETE /api/todolist/:id`: Deletes a specific todolist
6. `POST /api/todolist`: Creates a todolist

PHASE 4: CODE: DATA MODEL

```
// todolist.js
var mongoose = require('mongoose');

var todoSchema = mongoose.Schema({
  Title: String,
  Completed: Boolean
});

var todoListSchema = mongoose.Schema({
  Name: String,
  OwnerId: String,
  Todos: [todoSchema]
});

// CommonJS exports
exports.Todo = mongoose.model('Todo', todoSchema);
exports.TodoList = mongoose.model('TodoList', todoListSchema);
```

This is a simplified version of the code.

PHASE 4: CODE: CONTROLLER

```
var TodoList = require('./todolist').TodoList;

app.get('/api/todolists', function(req, res) {
    var userId = req.get('userId');
    // MongoDB / Mongoose ORM
    TodoList.find({ OwnerId: userId }, function(err, lists) {
        if (err) { return res.send(500, err); }

        if (!lists) { return res.send(404); }

        return res.send(lists);
    });
});
```

PHASE 4: CODE: CLIENT

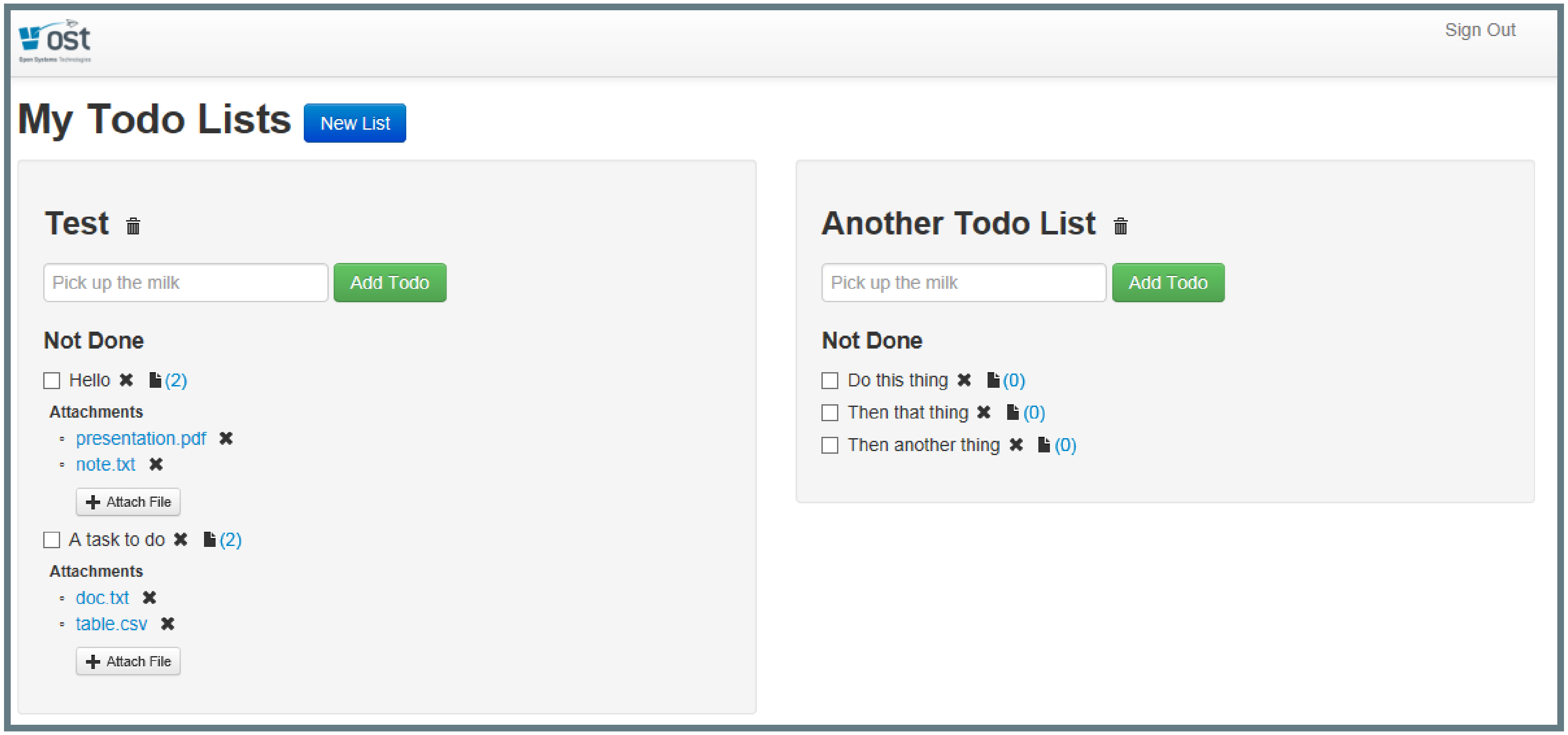
Before

```
angular.module("todos.services", ['ngResource']).
  factory("TodoList", ['$resource', function ($resource) {
    return $resource('/api/todolists/:id', { id: '@Id' });
  }]).
}]).
```

After

```
angular.module("todos.services", ['ngResource']).
  factory("TodoList", ['$resource', function ($resource) {
    return $resource('http://api.foofactory.net/api/todolists/:id', { id: '@I
  }]).
}]).
```

PHASE 4: DEMO



Sign Out

My Todo Lists

New List

Test

Pick up the milk | Add Todo

Not Done

- ☐ Hello ✖ (2)

  Attachments
  - presentation.pdf ✖
  - note.txt ✖

  + Attach File
- ☐ A task to do ✖ (2)

  Attachments
  - doc.txt ✖
  - table.csv ✖

  + Attach File

Another Todo List

Pick up the milk | Add Todo

Not Done

- ☐ Do this thing ✖ (0)
- ☐ Then that thing ✖ (0)
- ☐ Then another thing ✖ (0)

This should look the same as Phase 2, now powered by Node.js!

phase4.foofactory.net

MIGRATION STRATEGIES

- **Start small**: Consider implementing a small new feature in Node.js before you decide what parts of your architecture make sense to use Node.js.
- **Learn to love prototyping**: Node is great for quickly experimenting on new features. Prototype first before engineering a fully-baked solution.
- **Hook Node.js into IIS**: Consider using iisnode to easily hook Node.js into your IIS server instead of adding a separate Node.js server.
- **Use it alongside your existing apps**: Give it a try

CLOSING

Links:

- Code: github.com/hello-mean/glsec-2014
- Presentation: github.com/hello-mean/glsec-2014/presentation
- Demo: phase1.foofactory.net, phase1.foofactory.net, phase1.foofactory.net, phase1.foofactory.net

Brian Anderson | @Brianmatic

Nic Jansma | @NicJ

Jason Sich | @JaSich