How-To: Node.js and MongoDB on Ubuntu

Written by: Brandon L. Clark
July 24, 2013

This is a three part tutorial of how to develop with Node.js using MongoDB under Ubuntu (12.04 LTS to 13.04) and deploy with AWS using Juju. It is intended to help small application developers build a service and scale quickly if their service becomes popular.

- Part 1: Local Development Environment Setup
- Part 2: Node.js application using MongoDB via Mongoosejs
- Part 3: Deploying Node.js / MongoDB Application on AWS

Notes:
The lessons learned here are compatible with other cloud deployment services but focuses on AWS deployment.

I will make a effort to ensure this guide is updated for changes to the corresponding environments and services so please report any changes you may need to take in the process to support@themindspot.com.

Part 1: Local Development Environment Setup

Here you will learn how to install Node.js and MongoDB on Ubuntu from your terminal console. The packages in the Advanced Packaging Tool (AptGet) do not work always or are outdated at times on Ubuntu 12.04LTS to 13.04. You must install from their source to develop with the newest versions at the moment (7-23-13).

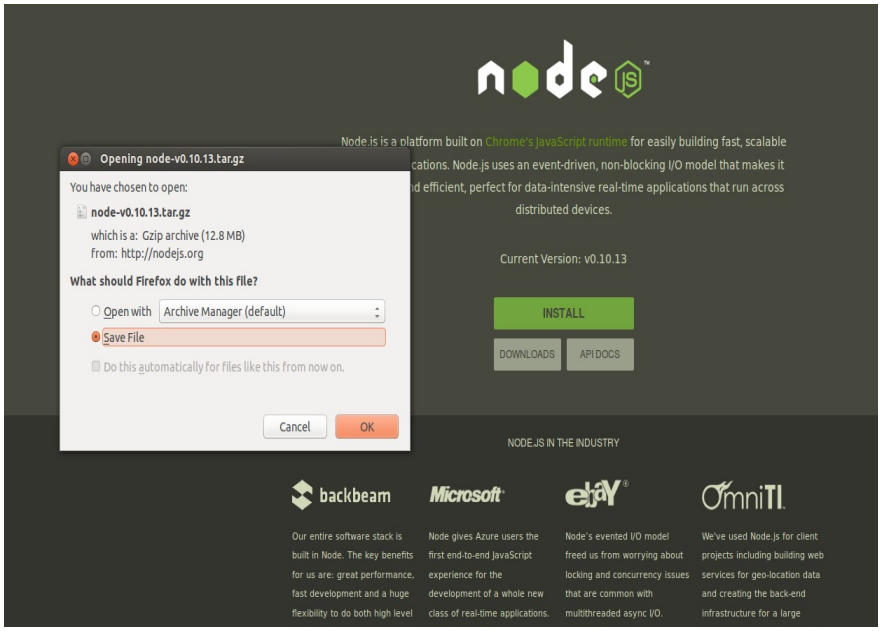
What you need!

Here are some tools you will need to install:

```
sudo apt-get install build-essential lamp-server^
```

What you need to download.

First download Node.js and the tool NPM. To do this download the latest version from nodejs.org. Do **not** download your platform specific version on the Downloads page (i.e. X64 for Linux).



Install Node.js

Decompress Node.js and enter the directory using the terminal console.

```
tar -xvzf node-v{the version number you downloaded}.tar.gz
cd node-v{the version number you downloaded}
```

Now lets configure the package for your system and build a package and install it.

```
./configure
sudo make
sudo make install
```

Check to see if it works.

```
node -v
```

This should display the version number you just installed.

Install MongoDB

To do this we will be using specified instruction for install on Ubuntu from the MongoDB website. Special thanks to 10gen.com for making this package available. *I am sorry for showing people how to get around your AWS packages; but their expensive!*

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv 7F0CEB10
echo 'deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen' | sudo tee /etc/apt/sources.list.d/10gen.list
sudo apt-get update
sudo apt-get install mongodb-10gen
```

Check to see if works.

```
Mongo
```

This will start your MongoDB terminal application where you can manual edit and search items. To exit the mono terminal type “Crtl + C”.

Your Done!

Now you can move on to the next section if you would like to learn more about writing your first **Node.js Application using MongoDB via Mongoosejs**. If you are already familiar writing your own applications with this framework you can skip ahead to **Deploying Node.js / MongoDB Applications on AWS**.

Part 2: Node.js Application using MongoDB via Mongoosejs

If you have not setup your development environment you may want to go back and review **Local Development Environment Setup**. We will assume those reading this tutorial know JavaScript and are not programming beginners. If you need to learn more about Node.js and how it works please visit this Q&A article How do I get started with Node.js for some helpful tips.

Here you will learn how to quickly deploy a sample application via Node.js on our local machine. We will use this sample application to quickly add Mongoose support for connecting to your local MongoDB framework.

Make and launch a Node.js with Express sample application.

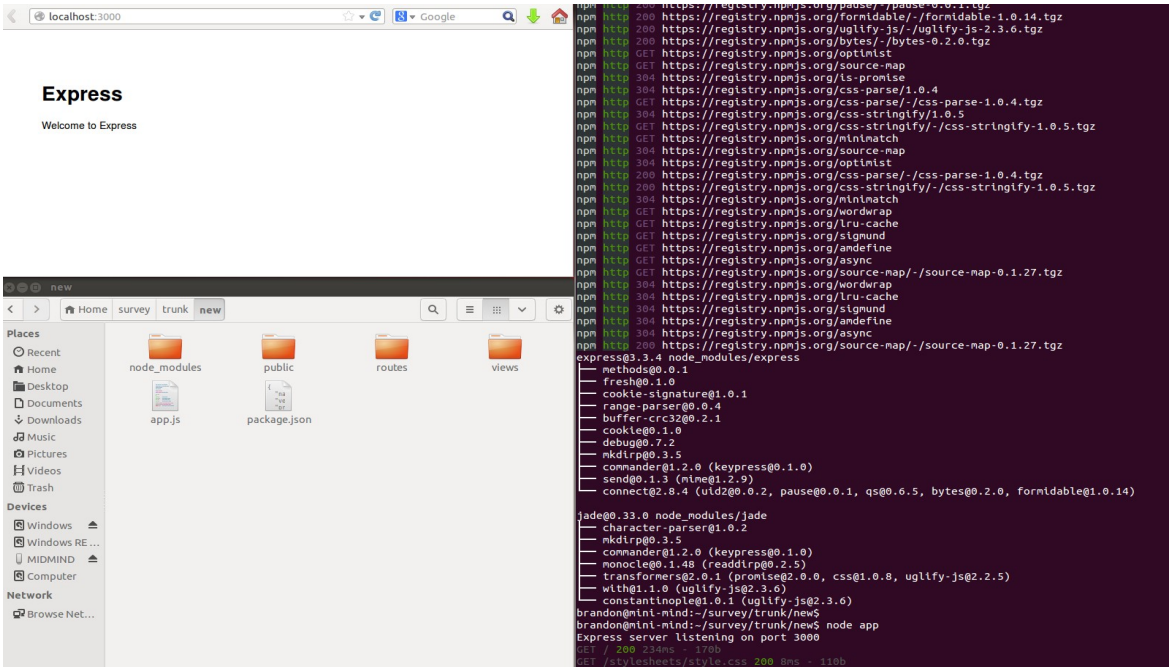
Node.js has many modules you can use to make programming easier. The most common of these is Express which has a built in feature to quickly make a "Hello World" boilerplate. It packages some of the most commonly used modules for Node.js to simplify the coding process. The boilerplate uses a Jade Template Engine and is extremely useful when writing Node.js applications.

Update your module registry and setup Express globally. Then make a sample application.

```
npm update
npm install express -g
npm sample test
```

The last line runs a script that will create a sample application. Follow the instructions given by the script when it has finished. We will review what the `npm` and `node` commands do later in this tutorial. This will build and launch your sample application. To see it in action open your browser and visit http://localhost:3000 after running the command below. To stop node press "Ctrl + C".

```
npm install && node app.js
```



Using Mongoose framework for MongoDB.

We are going to add Mongoose support for MongoDB and make a simple name list. With this we will be able to submit a name and view those names we have submitted.

If you need to learn more about Mongoose I suggest you visit mongoosejs.com; their Getting Started and API guides are very comprehensive.

With your programming editor of choice (I tend to use Gedit or VIM), open the `package.json` file. This is used to describe and setup the application sever. You will be adding to the *dependencies* section to configure our application to work with Mongoose along with making several changes to make sure we are running the latest module updates.

```
{
  "name": "mongonode-app",
  "version": "0.0.1",
  "private": true,
  "scripts": {
    "start": "node app.js"
  },
  "dependencies": {
    "express": "latest",
    "jade": "latest" ,
    "mongoose": "latest"
  }
}
```

Save and close.

Create a folder named `lib` and in this folder create a JavaScript file named `names.js`. In here you are going to write your own module with Mongoose that allows you to create and read database entries.

Open `lib/names.js` and create a schema and model like so:

```
var mongoose = require('mongoose'),
Schema = mongoose.Schema;

var NameSchema =  new Schema({
     name:       {type: String, lowercase: true, required: true, sparse: true, unique:true}
});

var NameModel = mongoose.model('names', NameSchema);
```

Now you need to write the code needed to submit data to the database. Add the following code to to the file:

```
module.exports.addName = function (name, cb) {
    mongoose.model('names', NameSchema);
    mongoose.connect('mongodb://localhost/list');
    var db = mongoose.connection;
    db.on('error', console.error.bind(console, 'connection error: '));

    var Name = new NameModel({name: name});
    Name.save(function(err) {
            if (err) {
                    cb(err);
            } else {
                    cb(null);
            }
            mongoose.disconnect();
    });
};
```

You also need to write the code needed to retrieve the data. Add the following code to the file:

```
function getList (cb){
    mongoose.model('names', NameSchema);
    mongoose.connect('mongodb://localhost/list');
    var db = mongoose.connection;
    db.on('error', console.error.bind(console, 'connection error: '));
    db.once('open', function () {
    NameModel.find({}, cb);
   });
};

module.exports.nameList = function (cb) {
    getList(function(err, names) {
            mongoose.disconnect();
            if (err) {
                    cb(null, err);
                    return;
            } else {
                    cb(names, null);
                    return;
            }
    });
};
```

Before we you go any further lets have you remove `routes/user.js` because it does not apply to this application. Open the `app.js` file next and delete the following lines:

```
, user = require('./routes/user')
app.use(express.favicon());
app.get('/users', user.list);
```

Now you can setup your application to use the module you just wrote and add it to `app.js`. To load the module write the following under you dependencies:

```
var Name = require('./lib/names.js');
```

Next you want to set the module us to read and write data. Replace the following line:

```
app.get('/register', routes.register);
```

with:

```
app.get('/', function(req, res) {
    Name.nameList(function(names, err){
            if (err) {
                    console.log(err);
            } else {
                    res.render('index', { nameList: names, title: 'MongoDB Sample App' });
                    console.log(names);
            }
    });
});

app.post('/add', function(req, res) {
    Name.addName(req.body.name, function(err){
            if (err) {
                    console.log(err);
            } else {
                    console.log('Added name: ' + req.body.name);
                    res.redirect('/');
            }
    });
});
```

Open `views/index.jade` so users can view and add names. This is where Jade template engine handy by writing parse code right into it. Replace all the code in this file with the following:

```
extends layout

block content
    form(method='POST', action='/add')
        label(for='name') Add Name
        br
        input(name='name')
        input(type='submit', value='Add')
    for names in nameList
        p #{names.name}
```

Now run it and open it in your browser at **http://localhost:3000**!

```
npm install  && node app.js
```

Your Done!

Now you can move on to the next section **Deploying Node.js / MongoDB Applications on AWS**.

Part 3: Deploying Node.js / MongoDB Applications on AWS

In this part of our tutorial I will show you how to setup deploy your Node.js application using MongoDB to Amazon Web Services using the power of Juju. Juju is a popular tool used to deploy cloud services to a number of cloud systems and services. It's not just for AWS! Visit juju.ubuntu.com if you would like to learn more about deploying to your cloud service or your own cloud system.

Juju deploys what are called 'charms'. These are environments that have been setup by industry professionals to run on cloud systems like AWS. You will be deploying three charms to assist you with this:

- node-app
- mongodb
- haproxy

Install Juju Tools

You will need to open terminal console and first setup your SSH key. For the command line below hit ENTER for each prompt, you do not need to enter anything unless you wish to. This identifies your identity with AWS when you begin deploying instances.

```
ssh-keygen -t rsa -b 2048
```

Now install Juju from PPA.

```
sudo add-apt-repository ppa:juju/devel
sudo apt-get update && sudo apt-get install juju-core
```

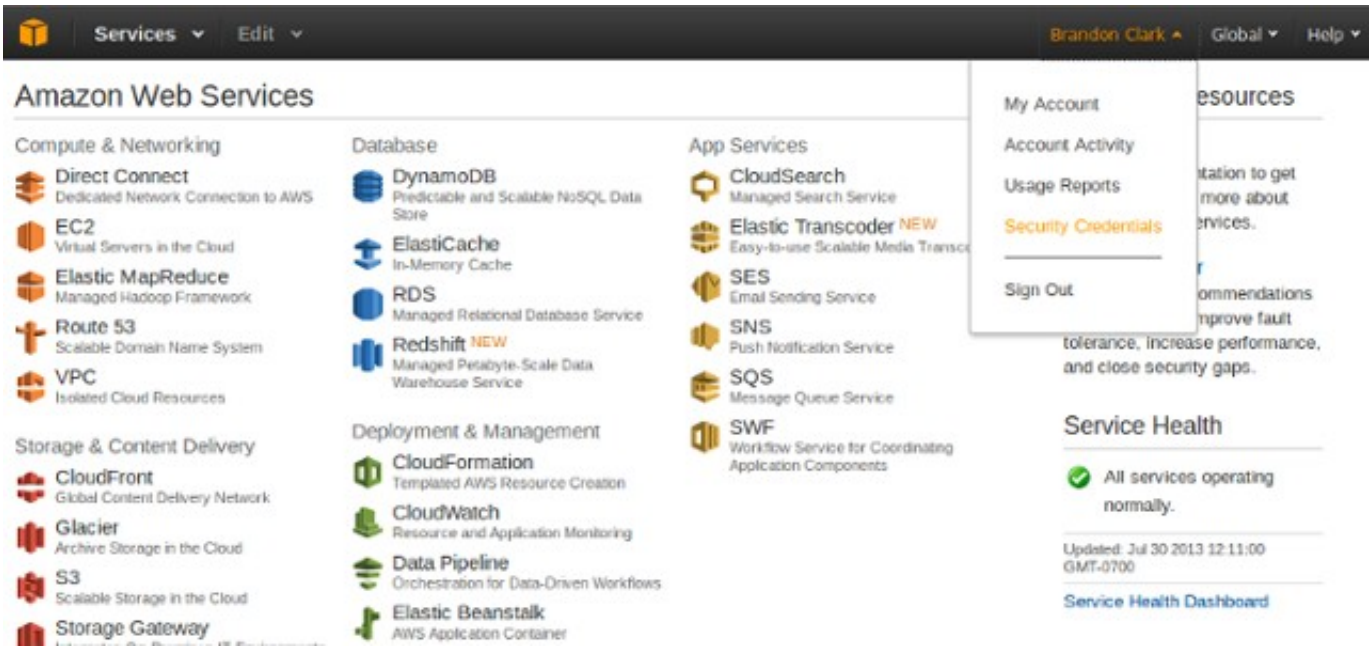
When it is done installing you want to setup your configuration. The following will create a folder and configuration file `~/.juju/environments.yaml`.

```
juju generate-config -w
```

Get your AWS access key.

This process requires you to have an Amazon Web Services (AWS) account. If you have not signed up for one yet, it can obtained at http://aws.amazon.com.

You can retrieve these values easily from your AWS Management Console at http://console.aws.amazon.com. Click on your name in the top-right and then the "Security Credentials" link from the drop down menu.



Under the "Access Keys" heading click the "Create New Root Key" button. You will be prompted to "Download Key File" which by default is named `rootkey.csv`. This file contains your AWS access-key and secret-key for the `~/.juju/environments.yaml` configuration file you created earlier.

Open the configuration file and uncomment the access-key and secret-key and copy the values from `rootkey.csv` to the appropriate fields. Here is a example of how it should look but with your info:

```
amazon:
    type: ec2
    admin-secret: 772b97c4c31c6b5883475e396d9a6d32
    control-bucket: juju-a29403f89d8223343d3cab01f1ca5a4d
    default-series: precise
    region: us-east-1
    access-key:  AKSDXSCIY67DFBJNB8
    secret-key: HGJKHGV7TOF8OOGO
```

Set your application to use AWS.

First you need to create a configuration module for our Node.js charm. This is where your Node.js application looks to find its assigned mongodb instance on AWS.

Make a folder in the root of the application, `config`. Create a file in that folder called `config.js`. Open your configuration file and write the following module:

```
module.exports = config = {
   "name" : "mongonode"
  ,"listen_port" : 8000
  ,"mongo_host" : "localhost"
  ,"mongo_port" : 27017
}
```

Now setup you `lib/names.js` file to use these modules. Add these lines to your dependencies.

```
var config = require('./../config/config.js');
```

You can now use this module and edit you `mongoose.connect` setting like so:

```
mongoose.connect('mongodb://' + config.mongo_host + ':' + config.mongo_port + '/list');
```

Lastly you need to change your application servers file name from `app.js` to `server.js`. The node-app charm requires this.

You should create a git repository and upload your application now. I have one setup for this tutorial where you can review the code if you have any questions or problems. Be sure to delete your node_modules folder before you do, it will not be needed.

• http://github.com/TheMindCompany/mongonode-app.git

Deploy your AWS charms.

First your going to need to clone the node-app from http://github.com/TheMindCompany/node-app. You will need it to be located in a specific folder structure like `~/charms/precise/node-app`.

Now lets edit `node-app/config.yaml`. You need to edit the application name and repository location at the very least. I have named our application mongonode-app and directed git to my applications git repository. Now its time to get AWS started up:

```
juju bootstrap
```

To find out if the service is ready for you to begin deploying service instances run:

```
juju status
```

First we will deploy the node-app charm you cloned along with mongodb, and haproxy. The node-app charm may take 20 minute to startup the first time because it builds Node.js from source like we did in Part 1 of this tutorial.

```
juju deploy –repository charms local:node-app mongonode-app
juju deploy mongodb
juju deploy haproxy
```

Now you want to make these services communicate with each other:

```
juju add-relation mongodb mongonode-app
juju add-relation mongonode-app haproxy
```

You expose haproxy to the public so people can access the application.

```
juju expose haproxy
```

Run a status check and when the services are done you will see a URL location like:

- ec2-54-212-57-198.us-west-2.compute.amazonaws.com

Your Done!

Now go off and make something great. Good luck!

Note:
You could use the charm stores version it can be slightly difficult to submit the configuration .yaml file to.