Day 27: Restify–Build Correct REST Web Services in Node.js

NOVEMBER 24, 2013 BY SHEKHAR GULATI (HTTPS://BLOG.OPENSHIFT.COM/AUTHOR/SHEKHARGULATI/)

Today for my 30 day challenge (/learning-30-technologies-in-30-days-a-developer-challenge) I'm learning a Node.js module called restify (http://mcavage.me/node-restify/). This module makes it easy to write correct RESTful APIs in Node.js and provides out-of-the-box support for features like versioning, error handling, CORS, and content negotiation. Restify intentionally borrows from Express (http://expressjs.com/) as it's the de facto API for writing web applications on top of node.js. We will develop a RESTful API for storing jobs and store the data in MongoDB (http://www.mongodb.org/).

Restify Prerequisites

Restify requires NodeJS and the NPM package manager which comes by default with node.js installations. Download the latest version of NodeJS from official website (http://nodejs.org/). Once you have node.js and NPM installed, we will use the NPM system to install Harp.

This application uses MongoDB as data storage choice. Download the latest MongoDB release (http://www.mongodb.org/downloads) for your operation system.

Install Restify

Create a new directory at any convenient directory on your file system.

```
$ mkdir myapp
$ cd myapp
```

To install restify module issue this command:

```
$ npm install restify
```

We will use MongoJS as the MongoDB driver. Install the mongojs module:

```
$ npm install mongojs
```

Writing RESTful API

Now we have restify and mongojs installed, let's write code. Create a new file called app.js file.

```
$ touch app.js
```

Copy and paste the following content to app.js.

```
var restify = require('restify');
var mongojs = require("mongojs");
```

The two lines show above load the restify and mongojs modules using the require function and assign them to variables.

Now create a new server using restify API:

```
var restify = require('restify');
var mongojs = require("mongojs");

var ip_addr = '127.0.0.1';
var port    =  '8080';

var server = restify.createServer({
    name : "myapp"
});

server.listen(port ,ip_addr, function(){
    console.log('%s listening at %s ', server.name , server.url);
});
```

The code shown above creates a new server. The createServer() function takes an options object. We passed ***myapp*** as the name of the server in options object. You can view the full list of options in the documentation (http://mcavage.me/node-restify/#Creating-a-Server). After create the server instance, we call the listen function passing port, ip address, and a callback function.

Run the application by typing the following command.

```
$ node app.js
```

You will see following on the command line terminal.

```
myapp listening at http://127.0.0.1:8080
```

Configure Plugins

The restify module has a lot of built in plugins which we can use. Copy and paste the following in app.js. These should be added before the `server.listen()` function. Refer to documentation (http://mcavage.me/node-restify/#Bundled-Plugins) to learn about all the supported plugins.

```
server.use(restify.queryParser());
server.use(restify.bodyParser());
server.use(restify.CORS());
```

The three lines shown above :

1. The `restify.queryParser()` plugin is used to parse the HTTP query string (i.e., /jobs?skills=java,mysql). The parsed content will always be available in req.query.
2. The `restify.bodyParser()` takes care of turning your request data into a JavaScript object on the server automatically.
3. The `restify.CORS()` configures CORS (http://en.wikipedia.org/wiki/Cross-origin_resource_sharing) support in the application.

Configure MongoDB

Before adding the routes, let's add code to connect to ***myapp*** to the MongoDB database.

```
var connection_string = '127.0.0.1:27017/myapp';
var db = mongojs(connection_string, ['myapp']);
var jobs = db.collection("jobs");
```

In the code shown above, we connect to local MongoDB instance. Next, we get the jobs collection using database object.

Writing CRUD API

Now, we have the server and database part ready. We still need routes to define the behaviour of the API. Copy and paste the following code to app.js.

```
var PATH = '/jobs'
server.get({path : PATH , version : '0.0.1'} , findAllJobs);
server.get({path : PATH +'/:jobId' , version : '0.0.1'} , findJob);
server.post({path : PATH , version: '0.0.1'} ,postNewJob);
server.del({path : PATH +'/:jobId' , version: '0.0.1'} ,deleteJob);
```

The code shown above does the following:

1. When a user makes a GET request to '/jobs', then findAllJobs callback will be invoked. The another interesting part is the use of versioned routes. A client can specify the version using `Accept-Version` header.
2. When a user makes a GET request to '/jobs/123', then findJob callback will be invoked.
3. When a user makes POST request to '/jobs', then postNewJob callback will be invoked.

4. When a user makes DELETE request to '/jobs/123', then postNewJob callback will be invoked.

Now we will write the callbacks. Copy and paste the following to app.js.

```
function findAllJobs(req, res , next){
    res.setHeader('Access-Control-Allow-Origin','*');
    jobs.find().limit(20).sort({postedOn : -1} , function(err , success){
        console.log('Response success '+success);
        console.log('Response error '+err);
        if(success){
            res.send(200 , success);
            return next();
        }else{
            return next(err);
        }

    });

}

function findJob(req, res , next){
    res.setHeader('Access-Control-Allow-Origin','*');
    jobs.findOne({_id:mongojs.ObjectId(req.params.jobId)} , function(err , success){
        console.log('Response success '+success);
        console.log('Response error '+err);
        if(success){
            res.send(200 , success);
            return next();
        }
        return next(err);
    })
}

function postNewJob(req , res , next){
    var job = {};
    job.title = req.params.title;
    job.description = req.params.description;
    job.location = req.params.location;
    job.postedOn = new Date();

    res.setHeader('Access-Control-Allow-Origin','*');

    jobs.save(job , function(err , success){
        console.log('Response success '+success);
        console.log('Response error '+err);
        if(success){
            res.send(201 , job);
            return next();
        }else{
            return next(err);
```

```
        }
    });
}

function deleteJob(req , res , next){
    res.setHeader('Access-Control-Allow-Origin','*');
    jobs.remove({_id:mongojs.ObjectId(req.params.jobId)} , function(err , success){
        console.log('Response success '+success);
        console.log('Response error '+err);
        if(success){
            res.send(204);
            return next();
        } else{
            return next(err);
        }
    })

}
```

The code shown above is self explanatory. We are using mongojs API to perform CRUD operations.

We can test the web services using curl. To create a new job, type the command shown below.

```
$ curl -i -X POST -H "Content-Type: application/json" -d '{"title":"NodeJS Developer Requi
red" , "description":"NodeJS Developer Required" , "location":"Sector 30, Gurgaon, India"}
' http://127.0.0.1:8080/jobs
```

To find all the jobs

```
$ curl -is http://127.0.0.1:8080/jobs
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: application/json
Content-Length: 187
Date: Sun, 24 Nov 2013 16:17:27 GMT
Connection: keep-alive

[{"title":"NodeJS Developer Required","description":"NodeJS Developer Required","location"
:"Sector 30, Gurgaon, India","postedOn":"2013-11-24T16:16:16.688Z","_id":"52922650aab61073
20000001"}]
```

Deploy to Cloud

Before we deploy the application to OpenShift, we'll have to do few setup tasks :

1. Sign up for an OpenShift Account (https://www.openshift.com/app/account/new). It is completely free, and Red Hat gives every user three free Gears on which to run your applications. At the time of this writing, the combined resources allocated for each user is 1.5

GB of memory and 3 GB of disk space.

2. Install the rhc client tool (https://openshift.redhat.com/community/get-started#cli) on the machine. The rhc is a ruby gem so you need to have ruby 1.8.7 or above on your machine. To install rhc type: `sudo gem install rhc`
   If you already have one, make sure it is the latest one. To update the rhc, execute the command shown below. `sudo gem update rhc`
   For additional assistance setting up the rhc command-line tool, see the following page: https://openshift.redhat.com/community/developers/rhc-client-tools-install
3. Setup the OpenShift account using `rhc setup` command. This command will help us create a namespace and upload your ssh keys to OpenShift server.

After setup, create a new OpenShift application by running the following command:

```
$ rhc create-app day27demo nodejs-0.10 mongodb-2 --from-code https://github.com/shekhargul
ati/day27-restify-openshift-demo.git
```

It will do all the stuff from creating an application, to setting up public DNS, to creating private git repository, and then finally deploying the application using code from my Github repository. The app is running here http://day27demo-{domain-name}.rhcloud.com//

That's it for today. Keep giving feedback.

Next Steps

- Sign up for OpenShift Online (https://www.openshift.com/app/account/new) and try this out yourself
- Promote and show off your awesome app in the OpenShift Application Gallery (https://www.openshift.com/application-gallery) today.

Automatic Updates

Stay informed and learn more about OpenShift by receiving email updates (http://openshift.us3.list-manage.com/subscribe?u=979c70339150d05eec1531104&id=c528e5e48e).

Get blog updates by email
(http://openshift.us3.list-manage.com/subscribe?u=979c70339150d05eec1531104&id=c528e5e48e)
(http://openshift.us3.list-manage.com/subscribe?u=979c70339150d05eec1531104&id=c528e5e48e)

Categories:
MongoDB (https://blog.openshift.com/category/technologies/mongodb/), Node.js (https://blog.openshift.com/category/technologies/node-js/)

Tags:
mongodb (https://blog.openshift.com/tag/mongodb-2/), Node.js (https://blog.openshift.com/tag/node-js/), rest (https://blog.openshift.com/tag/rest/)

Comments are closed.

Search the blog

POPULAR POSTS

- Custom URL names for your PaaS applications (host forwarding and cnames) (https://blog.openshift.com/custom-url-names-for-your-paas-applications-host-forwarding-and-cnames-the-openshift-way/)
- Day 12: OpenCV–Face Detection for Java Developers (https://blog.openshift.com/day-12-opencv-face-detection-for-java-developers/)
- How to build Java WebSocket Applications Using the JSR 356 API (https://blog.openshift.com/how-to-build-java-websocket-applications-using-the-jsr-356-api/)
- Learning 30 Technologies in 30 Days: A Developer Challenge (https://blog.openshift.com/learning-30-technologies-in-30-days-a-developer-challenge/)
- OpenShift V3 Deep Dive Tutorial | The Next Generation of PaaS (https://blog.openshift.com/openshift-v3-deep-dive-docker-kubernetes/)
- OpenShift v3 Platform Combines Docker, Kubernetes, Atomic and More (https://blog.openshift.com/openshift-v3-platform-combines-docker-kubernetes-atomic-and-more/)
- 10 Reasons OpenShift is the Best Place for Node.js Aps (https://blog.openshift.com/10-reasons-openshift-is-the-best-place-to-host-your-nodejs-app/)

CATEGORIES

- Events (https://blog.openshift.com/category/events/)
- General (https://blog.openshift.com/category/general/)
  - Educators (https://blog.openshift.com/category/general/educators/)
  - News (https://blog.openshift.com/category/general/news/)
  - Thought Leadership (https://blog.openshift.com/category/general/thought-leadership/)
- Products (https://blog.openshift.com/category/products/)
  - OpenShift Enterprise (https://blog.openshift.com/category/products/openshift-enterprise/)

OpenShift Online (https://blog.openshift.com/category/products/openshift-online/)
OpenShift Origin (https://blog.openshift.com/category/products/openshift-origin/)

Technologies (https://blog.openshift.com/category/technologies/)
Java (https://blog.openshift.com/category/technologies/java/)
JBoss (https://blog.openshift.com/category/technologies/jboss/)
Jenkins (https://blog.openshift.com/category/technologies/jenkins/)
MongoDB (https://blog.openshift.com/category/technologies/mongodb/)
MySQL (https://blog.openshift.com/category/technologies/mysql/)
Node.js (https://blog.openshift.com/category/technologies/node-js/)
Perl (https://blog.openshift.com/category/technologies/perl/)
PHP (https://blog.openshift.com/category/technologies/php/)
PostgreSQL (https://blog.openshift.com/category/technologies/postgresql/)
Python (https://blog.openshift.com/category/technologies/python/)
Ruby (https://blog.openshift.com/category/technologies/ruby/)
Tomcat (https://blog.openshift.com/category/technologies/tomcat/)
Zend Server (https://blog.openshift.com/category/technologies/zend-server/)

Videos (https://blog.openshift.com/category/videos/)

USEFUL LINKS

App Gallery (https://www.openshift.com/application-gallery)
Developer Center (https://www.openshift.com/developers)
FAQs (https://help.openshift.com/hc/en-us/categories/200177370-Frequently-Asked-Questions)
Upcoming Events (https://www.openshift.com/events)
What is Openshift? (https://www.openshift.com/products)

ARCHIVES

Select Month ▼