Getting started with Node.js, Mongoose and MongoDB Part 2

by Micha Hernandez van Leuffen - Jun 21 2013

wercker | nodejs | mongodb | mongoose | tdd | javascript

This article is outdated. Please visit our Dev Center.

This is a followup from our previous post where we created an application with node.js, powered by 10gen's MongoDB. Not that we've set up our build pipeline with wercker, we are going to deploy our application to Heroku and use MongoLab, a MongoDB-as-a-Service cloud platform.

Archive »

Tags

Wercker (28)

Weekinreview (19)

Golang (17)

Boxes (16)

Opendelivery (15)

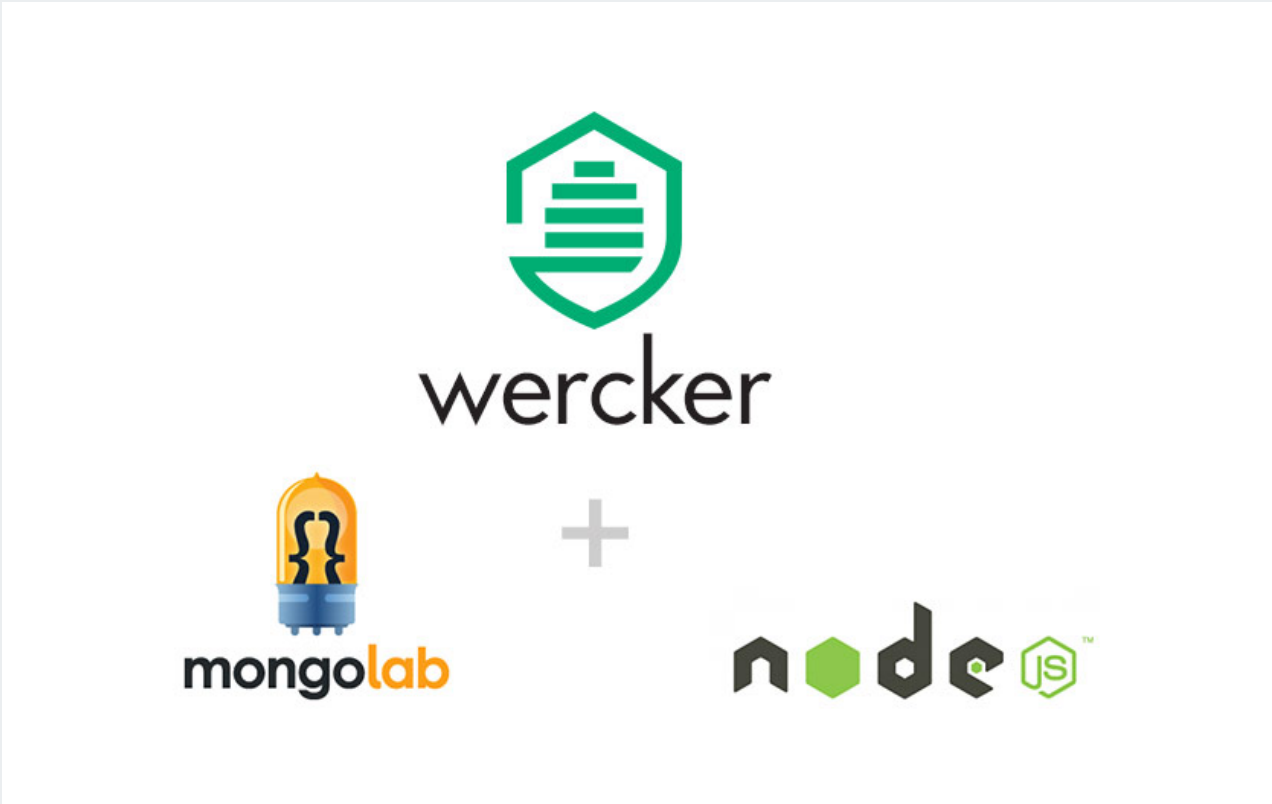
Ui (12)

Deployment (11)

Docker (10)

Design (9)

Python (9)



Introduction

In our previous post we created a node.js application powered by express and mongoose, a object document mapper for MongoDB. We created this application using test-driven development using Mocha and SuperTest. This article is also available on our dev center

You can visit the finished application on wercker here.

The code to this tutorial app is open source on GitHub so feel free to fork and clone it.

Now, let's deploy our app!

Deploying our app

We will deploy our application to Heroku, so make sure you have a (verified) Heroku account and have installed the Heroku Toolbelt. Let's first create an app:

```
heroku create

Creating guarded-atoll-9149... done, stack is cedar
http://guarded-atoll-9149.herokuapp.com/ | git@heroku.c
Git remote heroku added
```

Now we need a MongoDB instance in the cloud! Fortunately, Heroku has a Marketplace (wercker is on it by the way) filled with addons. One of which are our friends at Mongolab, so let's use their add-on to provision a MongoDB database:

```
heroku addons:add mongolab

Adding mongolab on guarded-atoll-9149... done, v3 (free
Welcome to MongoLab.  Your new subscription is ready fo
Use `heroku addons:docs mongolab` to view documentatio
```

For our `test` environment we used the `WERCKER_MONGODB_HOST` environment variable that was provided by wercker and was defined through the `wercker.yml`. We now need a similar environment variable for our production setting which is powered by Heroku and Mongolab. We can retrieve this environment variable, again via the Heroku command line interface:

```
heroku config | grep MONGOLAB_URI

MONGOLAB_URI: mongodb://heroku_app3489u7438034:ofs0gfj
```

We now need to use this environment variable, `MONGOLAB_URI`, in our actual application. Modify the **configure** section in your `app.js` file in the following way:

```
// Configure express
app.configure('development', function() {
  mongoose.connect('mongodb://localhost/todos');
});

app.configure('test', function() {
  mongoose.connect('mongodb://'+ process.env.WERCKER_M
});

app.configure('production', function() {
  mongoose.connect('mongodb://' + process.env.MONGOLAB_
});
```

Similarly to adding the `NODE_ENV=test` environment variable to wercker in the previous post, we need to do the same for Heroku, but now of course `NODE_ENV=production` as Heroku is our production environment.

```
heroku config:set NODE_ENV=production

Setting config vars and restarting guarded-atoll-9149.
NODE_ENV: production
```

We now have succesfully set up our production environment consisting of Heroku and MongoLab.

Creating a Heroku Procfile

Heroku needs to know which process to run on their cloud platform to actually launch your application. This is done throught the Heroku Procfile.

Create a file called `Procfile` in your project directory with the following line of code:

```
web: node app.js
```

Let's add this file to our repository and push it to our version control system:

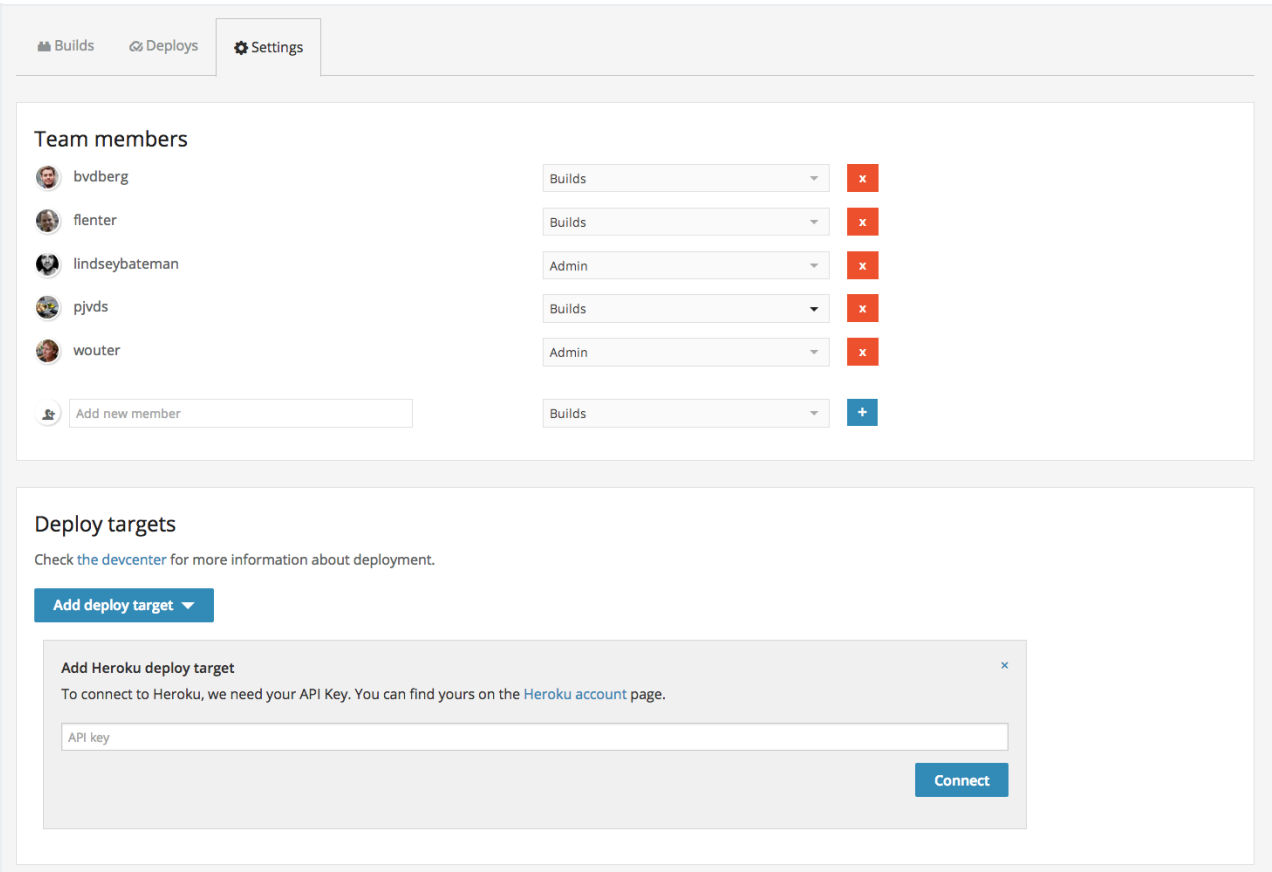
```
git add Procfile
git commit -am 'added Procfile'
git push origin master
```

This will trigger a new build on wercker (which should pass as we didn't change anything dramatic) and we're now ready to deploy our application.

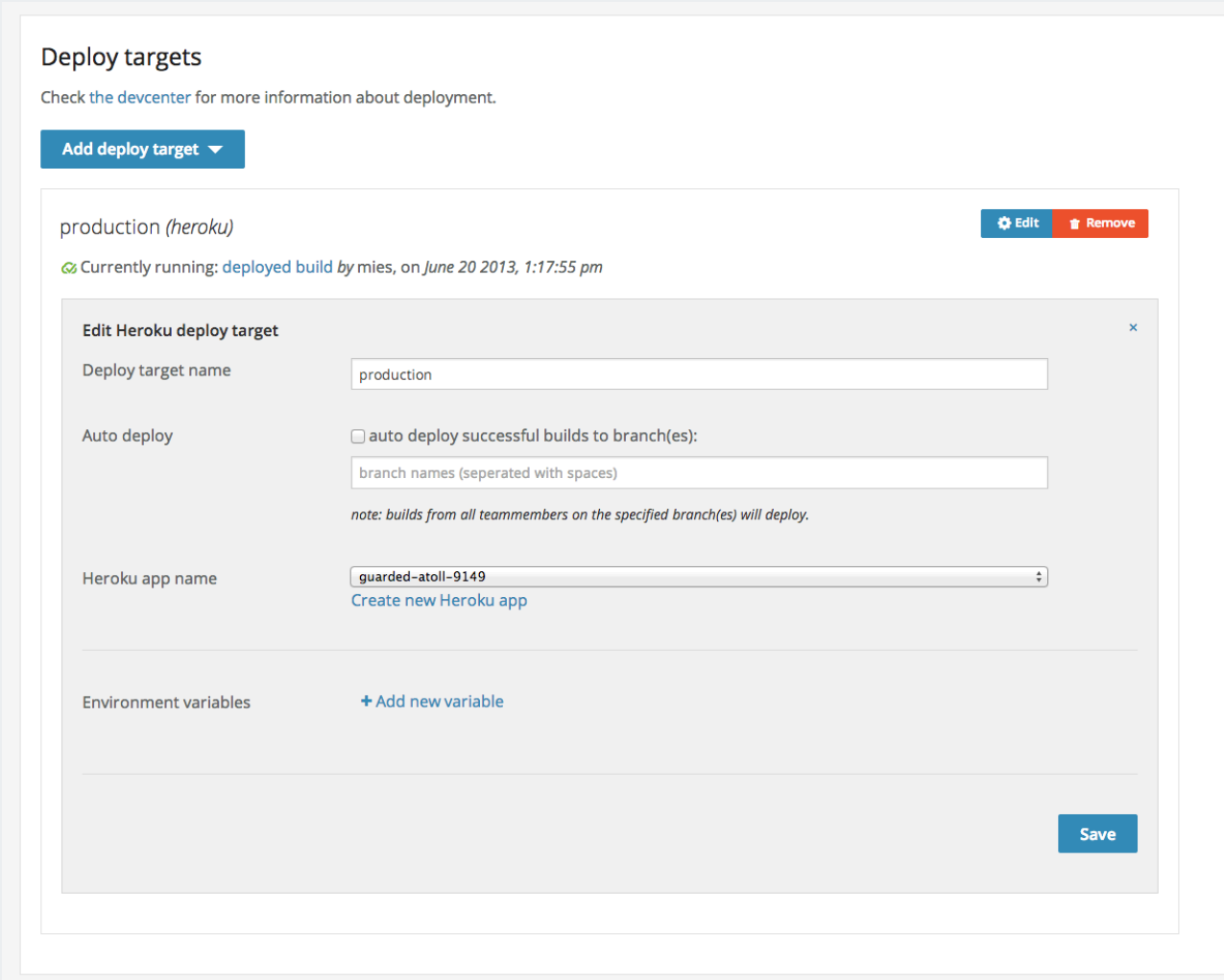
Add deploy target

First, we add a deploy target. Go to your settings tab for your application on wercker and look for the **deploy targets** section. Although wercker has an [add-on] on the Heroku Marketplace, making deployment even easier, we are going

to add Heroku manually as a deploy target.



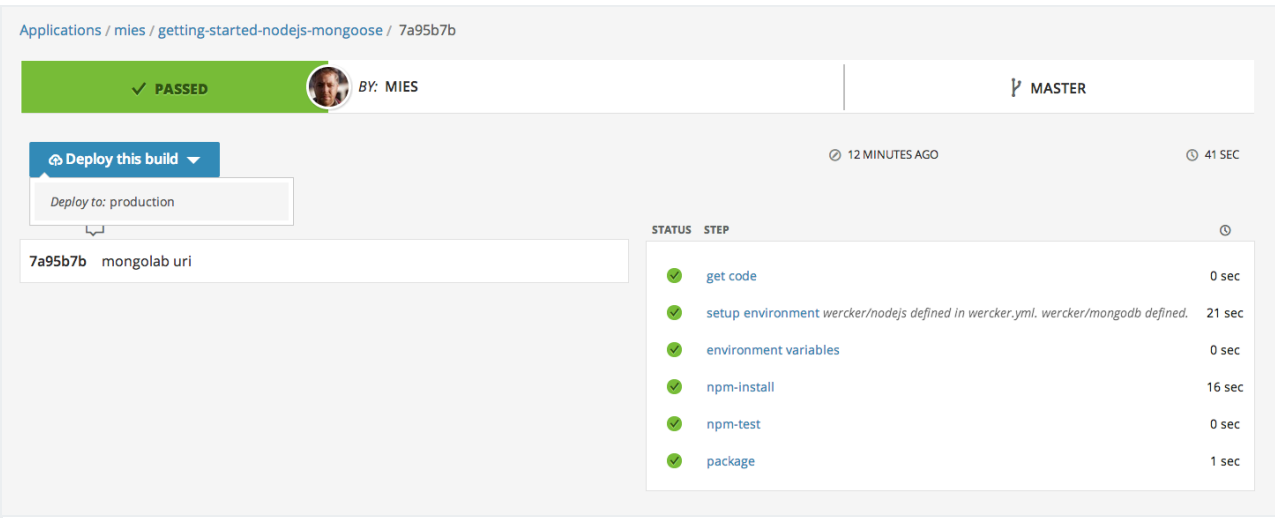
As the page indicates, retrieve your Heroku API key from your Heroku Dashboard and paste it in the form. Next, you are presented with a form where you can *name* your deploy target (let's go with *production*) and are able to auto deploy specific branches. We've previously written a post on auto-deployment here. We are also able to either select an existing Heroku app that we want to deploy to, or create one. I'm going to pick the application I've previously created using the `heroku create` command and to which I've also added the MongoLab add-on



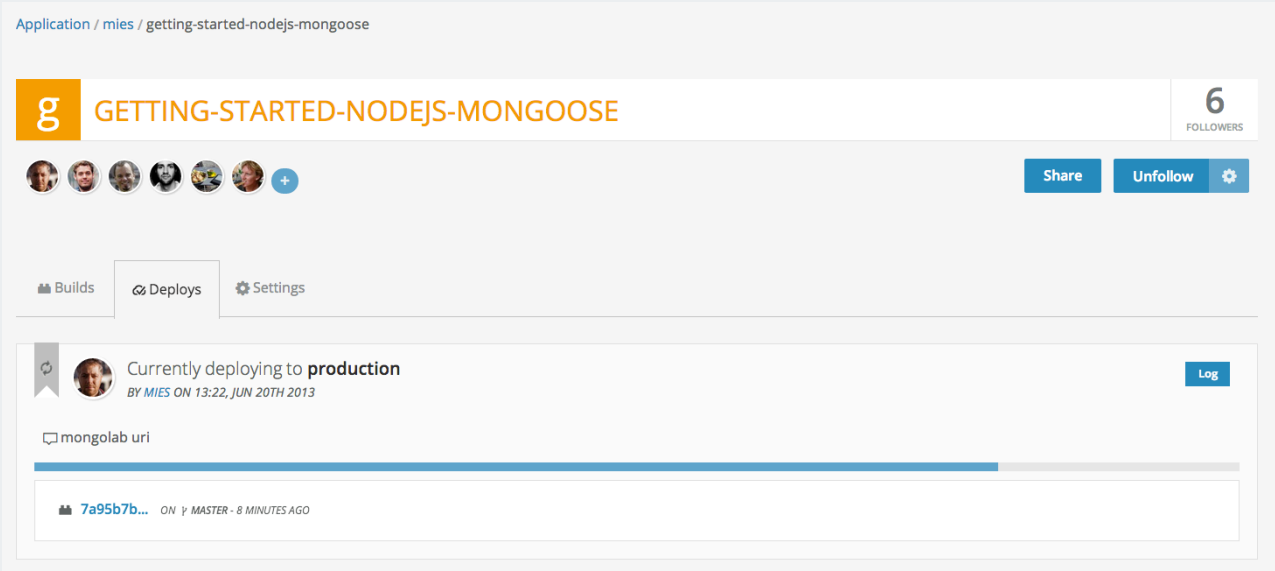
Let's deploy our application!

Deploying our application

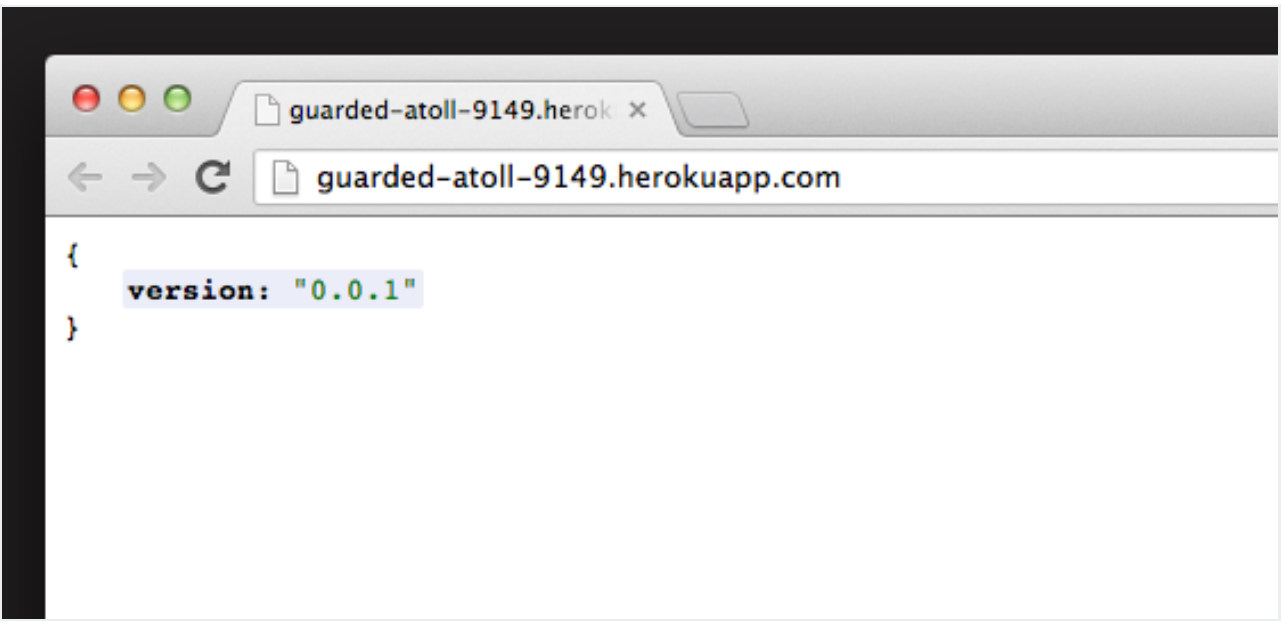
Go to the latest green build on wercker and hit the *deploy this build* button. A pull down menu will appear and you can select the target ('production') that you've just created,



If you go to the **deploys** tab you can see your deploy in action:



If we visit the root of our application on Heroku we can see it in action:



Congratulations! You've succesfully built a REST API using test-driven development and leveraged wercker for continuous delivery. Subscribe to the feed of our blog and follow us on twitter for more updates.

Earn some stickers!

Let us know about the applications you build with wercker. Don't forget to tweet out a screenshot of your first green build with **#wercker** and we'll send you some @wercker stickers.

Don't forget to sign up for wercker for free here.

Share this post on:

Tweet Google + Facebook