Slog | Serge + Blog
the first and only official personal blog

Deploying a Meteor app to an AWS server with Meteor-Up, for beginners

Apr 10, 2015

We recently learned (https://news.ycombinator.com/item?id=9231200) from some people not related to Knotable (http://www.knotable.com), that we're building what is probably the largest Meteor app in existence.

If you're reading this, you either know me, or you know that Meteor (https://www.meteor.com/) is an open-source full-stack framework/platform for building modern, real-time, web and mobile applications. (How about that for some buzz-words?)

Over the past week, one of our developers, Gaurav Chikhale (https://github.com/gauravchl) has been working on converting our homepage (http://knotable.com) from an Angular.js app to a Meteor one. It seemed only fitting, given the fact that pretty much everything else is Meteor and the fact that everyone involved hated working with Jade (http://jade-lang.com/). He had gotten far enough in the process that it was time to deploy it to a server under our control, rather than one hosted on Github pages or Meteor.com.

Angus McLeod (https://twitter.com/anguspmcleod), another product manager and resident Operations consultant suggested using Arunoda Susiripala's Meteor-Up (https://github.com/arunoda/meteor-up) (here-after "mup"). The mup documentation is pretty straightforward and even provides a screencast of someone using it to set up a meteor app with Digital Ocean and Compose.io – tools I'm rather unfamiliar with. I thought that I'd write how to deploy a Meteor app using AWS (Amazon Web Services), which is probably a more common scenario, for a near total noob.

Getting started.

I do all of my (meager) development on a mac. If you're on a linux machine, this will work fine for you as well. If you're using Windows, there are probably utilities that might help you, but I'm not going to try.

The first thing you'll want to do is make sure you have Node.js installed on your machine. You'll need it because you need NPM (Node Package Manager). If you're reading this, you probably already do, but I suppose it doesn't hurt to preface this whole process with that. (If you don't know what I'm talking about, either install Node (https://nodejs.org/) or run like hell.)

If you're all set, you'll want to install mup. Open up terminal and type:

```
npm install -g mup
```

(The `-g` flag will install mup globally.)

Once that's out of the way, you'll want to do is get a local copy of your Meteor project. Chances are, you're cloning it from github. Navigate to the project directory (`cd /path/to/project`). Everything else we do with our terminal window will be here. If you're feeling curious, run `mup help` to get familiar with what mup does/can do.

Initializing and configuring mup

This stuff is pretty straightforward if you read mup's documentation, but I'll go over it again here really quickly.

In your project directory initialize mup by running:

```
mup init
```

As stated clearly in the documentation:

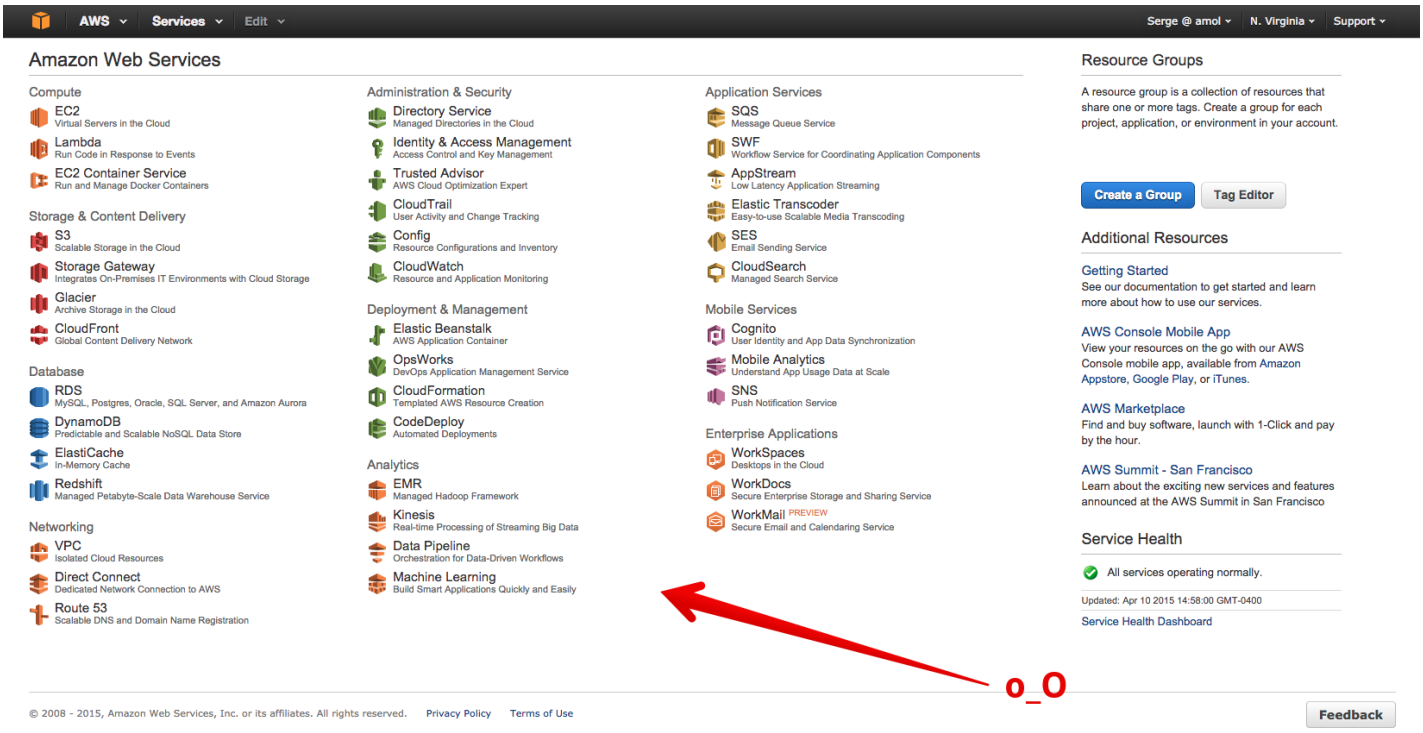
> *This will create two files in your Meteor Up project directory:*
>
> - *mup.json - Meteor Up configuration file*
> - *settings.json - Settings for Meteor's settings API (http://docs.meteor.com/#meteor_settings)*

Chances are, you won't really have to deal with the settings.json file for a simple app, so we're just going to focus on setting up `mup.json` for now. This is the part where I wish I'd had my own help. But before we set this up, we've got to start a new AWS instance (server), which can be a little intimidating.

Selecting and launching the correct AWS instance

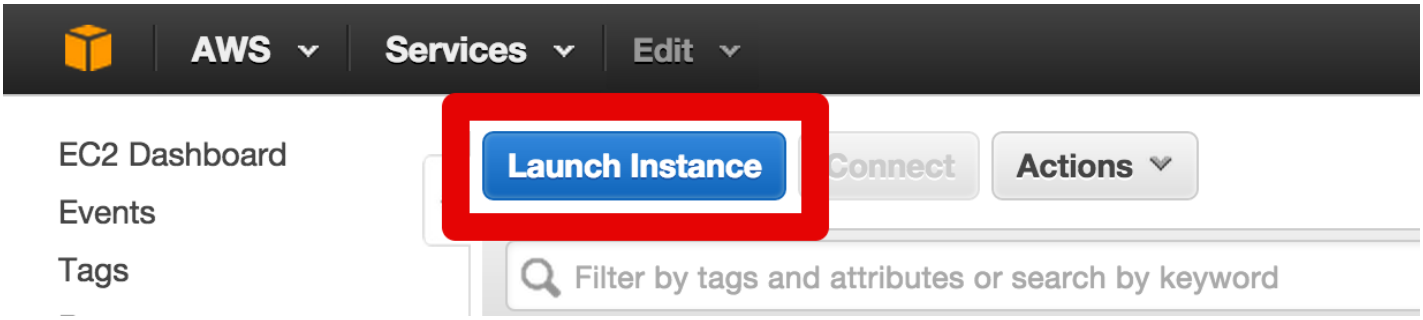
Step 0: Don't Panic

Once you log into the AWS console, you're faced with an overwhelming amount of products and features.



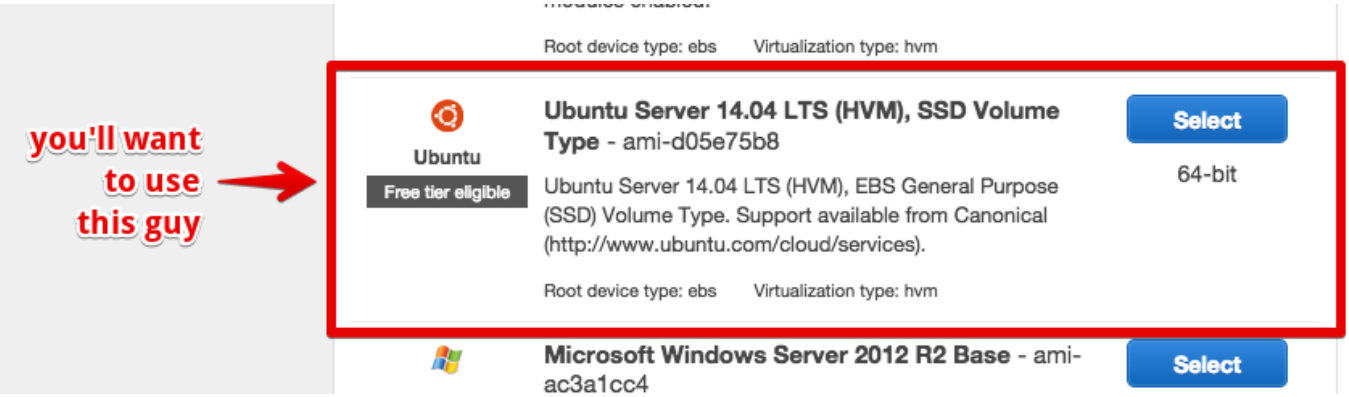
Luckily, we're focused on only one (or two) of them: EC2 (their virtual servers product), and maybe Route53 (their DNS and Domain Name Registration service).

Navigate into EC2 and click the big blue "Launch Instance" button.



Step 1: Select an AMI

You'll see a page where you're asked to select an AMI, or "Amazon Machine Instance," which is little more than a fancy way of saying "virtual server." AWS offers a slew of different server types, but mup only works on Debian/Ubuntu and Open Solaris flavors of Linux. For most of the work we do at Knotable, we use the Amazon Linux AMI, which is a Redhat flavor of Linux. I don't know much, but I did learn that Redhat and Debian/Ubuntu flavors use different package managers (yum or rpm, and apt respectively) the hard way. Save yourself the hassle and pick the right type right away.



Step 2: Select Instance Type

Once you've selected your AMI, they're going to want you to select the an instance type, which is little more than a fancy way of saying, "select how powerful you want your virtual server to be." Depending on your budget, how resource intensive your app is, and on the traffic which it will receive, you will want a different instance type. AWS has a decent blog post (https://aws.amazon.com/blogs/aws/choosing-the-right-ec2-instance-type-for-your-application/) on selecting the right one for your app, but I'm not going to reccommend anything specific. For what it's worth, we went for the m3.medium.

Steps 3, 4 & 5:

Next, you're going to configure your instance details. Chances are that you're not going to need to change anything here until you're ready to get really fancy.

So go ahead and move on to Step 4: Add Storage. Once you're here, you can add more storage space to your digital server. Unless your app is huge, if you selected your instance type properly in Step 2, you should be fine with whatever the default is.

Move on to Step 5: Tag instance. Here, give your instance a name by which you will identify it. The name of your meteor project is probably fine.

Step 6: Configure Security Group:

As AWS explains, “a security group is a set of firewall rules that control the traffic for your instance.” You may have heard of firewalls and ports. Basically, if an IP address defines the “building” that is your server, the “port” defines the floor. Different business is conducted on the different floors, and unless you explicitly allow traffic to visit those floors explicitly, people won’t be able to conduct certain types of business.

Now this is important, becuase unless your instance is in a VPC (Virtual Private Cloud), you won’t be able to change your security groups without terminating your instance. You don’t particularly want to terminate your instance because it means that you have to go back to Step 1. You especially don’t want to do this if you’ve already moved past this stage and deployed your Meteor app. There’s nothing wrong with terminating an instance – you’re just going to be frustrated.

It’s time to learn from my mistakes again. Chances are, you’re going to want to allow at least two ports:

- **Port 22**: SSH
  - This is the floor of our building that conducts SSH business. It will allow you to get into your virtual server at AWS remotely. If you can’t do this, you’ll be really limited.
  - This port is open by default in a new security group. Don’t close it.
- **Port 80**: HTML
  - This is the floor of our building that conducts regular old internet business. If you want to serve any webpages that people can access from their browser, you will want to have this port open.

Step 7: Review your launch, and setup a key pair:

Once you’ve set your security groups, review your instance to make sure everything’s in order and click “Launch.” You will be prompted to select or created a key pair.

The reason it’s called a key pair, is because there’s two (shocker!) – a public key and a private key. When you generate a key pair, one half, the public key, is stored on your AWS instance, while the other is available for you to download. In order to SSH into your AWS instance, you need to specify which key to use. No key, no business on Port 22.

Your private key comes in the form of a `.pem` file. This key is called a private key for a reason. Anyone who has access to this `.pem` file will be able to access your server if they know the address. Don’t give it to people you don’t trust with your server. If something bad does happen, you may be able to get rid of the old public key and create a new key pair, but it’s a bit of a hassle.

You can proceed without creating or selecting a key, but I recommend that you don’t.

Also, please note, that if you decide to use an existing key pair, but you don’t have access to the private key of the key pair you have selected, you won’t be able to ssh into your server.

So let’s say you’ve followed my suggestion and created a new key pair and downloaded a private key. You can keep this anywhere on your computer, but keys are generally kept in the hidden `.ssh` folder that’s in your home directory `~/`.

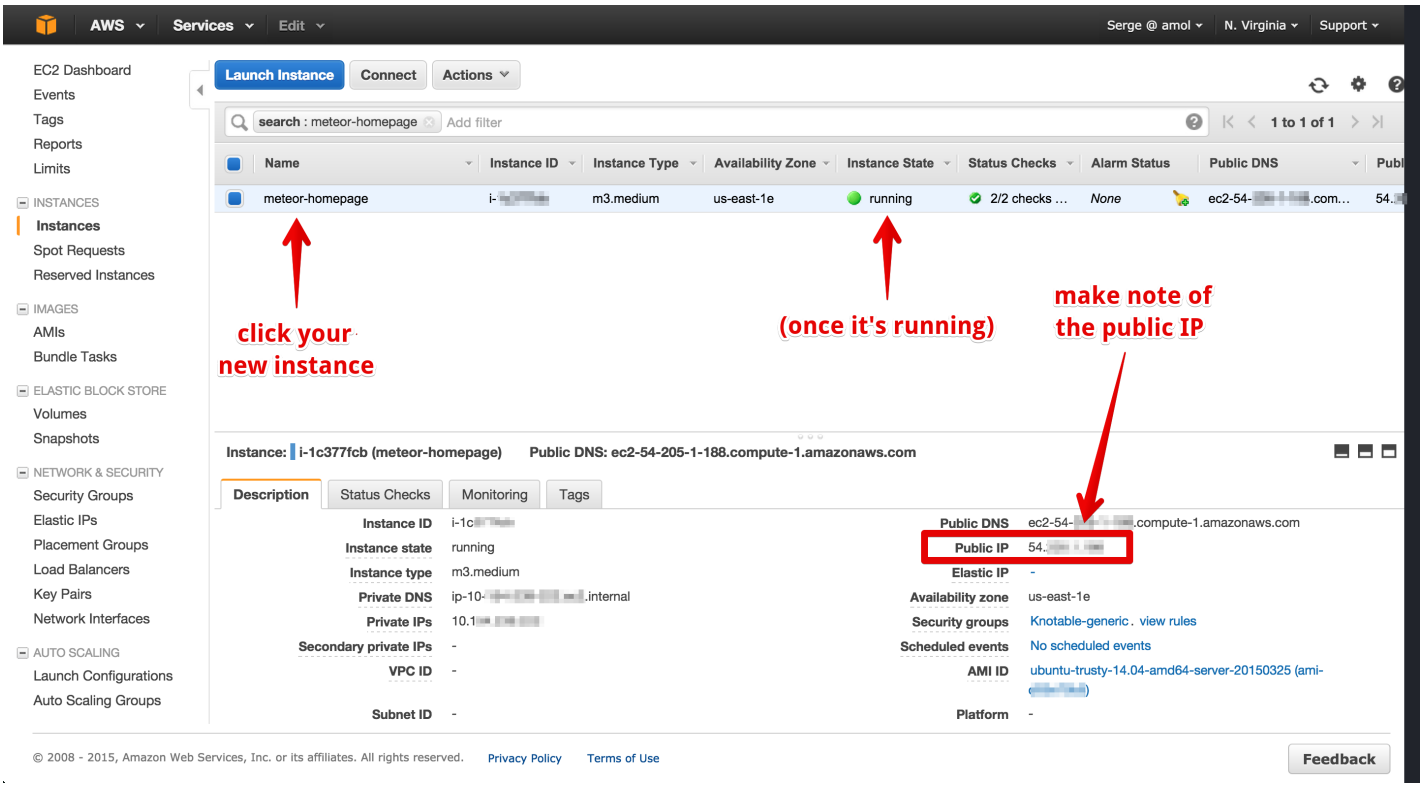
IMPORTANT: Change the local permissions for your `.pem` file. Navigate to whatever directory you're key file is in (e.g. with the command `cd ~/.ssh/`), and execute the following command:

```
chmod 400 keyname.pem
```

where "keyname" is the name of your pem file.

Key in hand (hehe), launch your instance and click "View Instances".

Once your instance is done initializing, click on your instance, and make note of your public IP.



Configuring mup.json

Now you're ready to really get started. The good news is that the difficult stuff is really pretty much out of the way.

In your terminal or in finder, navigate to your folder directory that now has a file called `mup.json` and open it up in your favorite text editor. (Please don't use "Text Edit", you're going to have a bad time.) (Really? You're using "Text Edit"? Do yourself a favor and set it to plaintext mode (http://www.tekrevue.com/tip/textedit-plain-text-mode/) and turn off smart everything.)

You'll see a file that looks very much like the one in the documentation (https://github.com/arunoda/meteor-up#example-file). Here's exactly what you have to change:

- `"host":"hostname"` should be: `"host":"[your IP address]"`
  - e.g.: `"host":"54.291.1.1"`
- `"username":"root"` should be: `"username":"ubuntu"`
- `"password":"password"` should get commented out: `//"password":"password"`
- `//"pem":"~/.ssh/id_rsa"` should get uncommented: `"pem":"path/to/your/pemfile"`
- `"appName":"meteor"` should be: `"appName":"[your-app-name]"`
- `"ROOT_URL":"http://myapp.com"` should be: `"ROOT_URL":"[whatever url your app will eventually live at]"`
  - An IP address should be fine here.

Save your work and head on back to your terminal.

SSH into your AWS server once

```
ssh -i /path/to/your/keyfile.pem ubuntu@0.0.0.0
```

Replace `0.0.0.0` above with your AWS server's IP address.

You may have to enter your password, or type `y` or `yes` if prompted. If you're successful, you will be inside your AWS server. You don't need to do anything in here. Simply run:

```
exit
```

and you will be back in your very own computer (even though it may feel as though you never left).

Setting up mup on your AWS server

Make sure that you're still in your project directory, and run:

```
mup setup
```

If you've done everything right up to this point – you should see some cool success screens and maybe a bit of advertising for Arunoda's other project, Kadira.io (http://www.kadira.io).

That was it for setting up mup on your AWS server. This is why mup is awesome. You normally would have to have struggled through manually installing a whole lot more stuff to get meteor running on a basically fresh server.

Deploying your meteor app

Make sure that you're still in your project directory, and run:

```
mup deploy
```

Again, if you've done everything right, you should see some happy looking success screens and some more advertising for Kadira, and your app should be deployed and running. If you run into some errors and can't google them to figure out what to do, feel free to leave them in the comments.

Visit your app on the internet.com

Type your AWS server's IP address into your browser's address bar and you should see your app (assuming you've done everything correctly and your Meteor app displays something on port 80).

Everything should more or less be working. You haven't hooked your app up to any external databases, so it will be empty at first, but that's likely out of your wheelhouse. If something is still not working and you've followed this guide very closely, I'm likely out of my wheelhouse and you'll probably need to find a developer to help you out.

127.0.0.1, which is my app ip. and I have allready tried setting different: HTTP_FORWARDED_COUNT environment variable

• Reply • Share ›

mkhait ➚ Jay Rocc • 25 days ago

hey **@Jay Rocc** , I'm going through the same exact thing. Have you found a solution? I understand what the problem is, but I can't find a solution just.

• Reply • Share ›

Christopher Waitforit Stark • 2 months ago

Just a little observation.

In the step 6 I think is HTTP not HTML

• Reply • Share ›

bordalix • 4 months ago

Great tutorial Serge, thanks. I think you have a small error, since it is necessary to configure in the mup.json the line "app": "/path/to/the/app" with the correct path to the meteor project. Additionally, I had some errors on the deploy, corrected when I match the node versions in the mup.json and in the project ("nodeVersion": "0.12.7",).

• Reply • Share ›

Neil Alexander • 5 months ago

After typing mup setup or mup init,, I get "/usr/bin/env: node: No such file or directory" error..I've tried sudo'ing, but that doesn't work..

• Reply • Share ›

Ross Newton • 6 months ago

Great writeup. This helped me a lot. Got my app up for testing no problem.

• Reply • Share ›

David Roberto Venegas • 6 months ago

After I upload it, should I just continue adding features on my local meteor app and then deploy it in the same location to "update" it, or how would the update flow be?

• Reply • Share ›

David Roberto Venegas • 6 months ago

Awesome! Thanks Serge, I was struggling on how to upload my meteor app, but I found the way.

• Reply • Share ›

Carelessly made using Bootstrap (http://www.getbootstrap.com) and Jekyll (http://jekyllrb.com/). Proudly hosted on DigitalOcean (http://digitalocean.com).