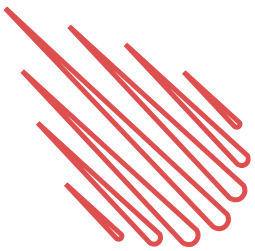
Menu

How to deploy a Meteor app to Galaxy using CI (part 1)

May 21, 2016 by Paul Dowman – Meteor, Deployment, Testing

We think Galaxy is the best place to host a Meteor app and we have started hosting our clients' apps there.

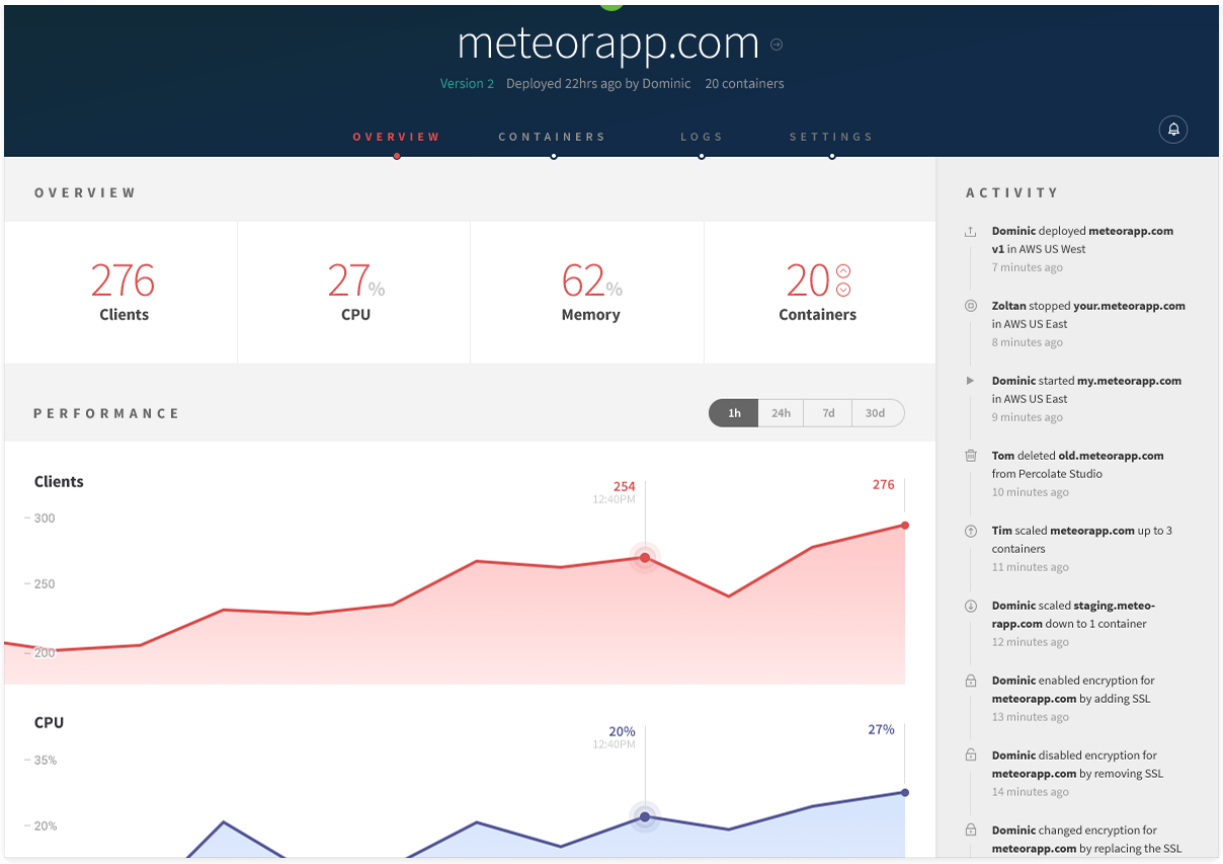


We have always used "Platform as a Service" (PaaS) hosting because even though they cost a bit more than DIY alternatives like Digital Ocean or AWS the total cost to our clients is *much* lower when people's time is taken into account.

Scaling up the server capacity takes no effort, and we don't have to worry about OS updates and all the little things that take work and can cause downtime (do you automate your log file rotation and archiving or will your app go down one day because the disk is full?). And we could never respond as quickly to critical security issues like DROWN or Heartbleed.

This is what managing your production app should look like:

Menu



But deployment should be automated using a Continuous Integration (CI) service because:

1. There's no risk of deploying the wrong thing. Deploying the wrong branch or uncommitted local changes can not only introduce bugs but also make them harder to reproduce for debugging since you won't know what exactly was deployed.
2. There's no risk of specifying the wrong Meteor settings with the deployment command.
3. Tests are run automatically and you won't deploy with failing tests.
4. You can ensure that your test or staging server is always up to date with the latest changes automatically.
5. You can automatically reset your test or staging server's data to a clean state (we actually reset it to a copy of production data, more

Menu

How to automate your Galaxy deployment

There are many CI services, we use Semaphore.

Most CI services will separate this into two steps: testing and deployment. If the testing phase doesn't fail then the service will deploy, usually to a specific server depending on which branch was built.

We have written a bit about testing here, so this article is going to focus on the deployment part only, and in **part 2** I'll show how we reset our staging server data to a copy of production.

Your CI service will have settings to deploy a build when there are new commits on a specific branch. In Semaphore you can add a new server and choose the "Generic Deployment" strategy.

Then you enter the commands to deploy, somewhat like you would do yourself on the command-line. (So first, make sure you can successfully deploy to Galaxy from the command-line).

with your meteor account. On your development machine it will save this login info, but on the CI server that won't work.

It's not a problem though. The `meteor deploy` command looks for an environment variable called `METEOR_SESSION_FILE` where it can find a saved authentication token.

To create that file use: `METEOR_SESSION_FILE=deployment_token.json meteor login`. (We have a separate Meteor/Galaxy account just for this, that's optional but recommended.) This will create the file `deployment_token.json` which you can save as a stored config file in your CI service. In Semaphore that file will be available to your deployment commands as `/home/runner/deployment_token.json`.

`meteor deploy` can then use the same `METEOR_SESSION_FILE` environment variable and it will read the token, like so:

```
export METEOR_SESSION_FILE=/home/runner/meteor-login.js
export DEPLOY_HOSTNAME=galaxy.meteor.com
meteor deploy myapp-staging.meteorapp.com
```

Or the one-line equivalent:

`METEOR_SESSION_FILE=/home/runner/meteor-login.json DEPLOY_HOSTNAME=galaxy.meteor.com meteor deploy myapp-staging.meteorapp.com`

Meteor settings and env vars

to put in your Meteor settings).

So you will also want to create another stored config file in your CI service. You will create one for each server environment (production, staging, etc.).

So then your deployment commands will look like this:

```
curl https://install.meteor.com/ | sh
export METEOR_SESSION_FILE=/home/runner/meteor-login.js
export DEPLOY_HOSTNAME=galaxy.meteor.com
meteor deploy myapp-staging.meteorapp.com --settings /h
```

That's it! Look out for part 2 of this post to see how we reset the test or staging servers to a copy of production data, and in the meantime go automate your deployments to Galaxy!

Menu

SERVICES

Enablement

Mobile

Web

Training

LEARN

Blog

Training

Tech Talks and Open Source

Guides

SOCIAL

Twitter

GitHub

YouTube

Facebook

Instagram

CONTACT

Work With Us

hello@okgrow.com