

Tech Challenge Results

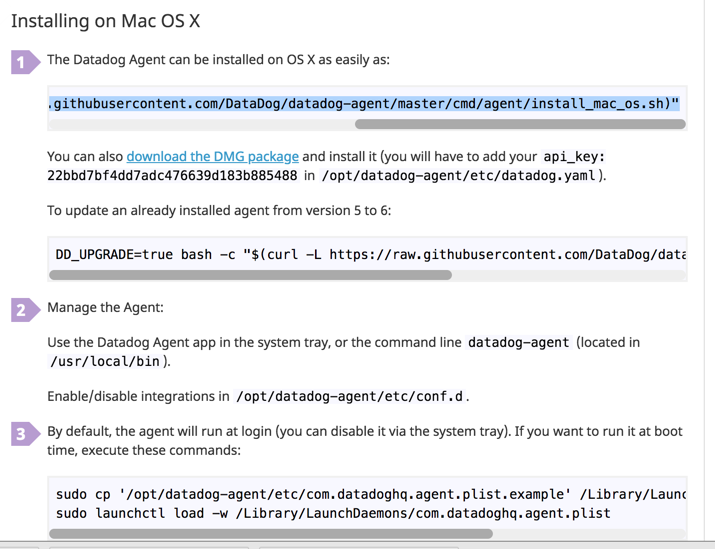
Charles D. “C.D.” Larson, Jr.

SE Applicant, Southwest

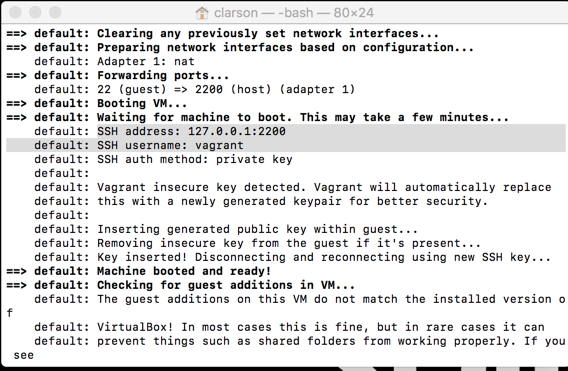
Hi, I’m Charles, and here are my efforts in the Tech Challenge, along with some notes of areas I found confusing.

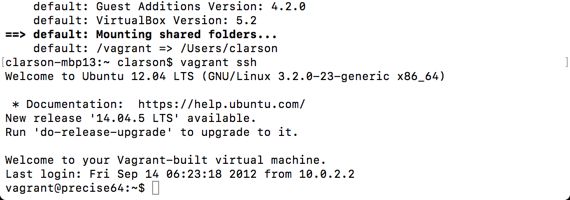
As required, I’ve included screenshots along the way.

Just for kicks, start with it on my Mac:

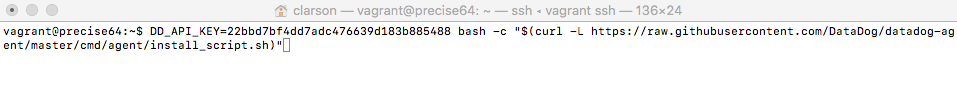


Setting up Vagrant

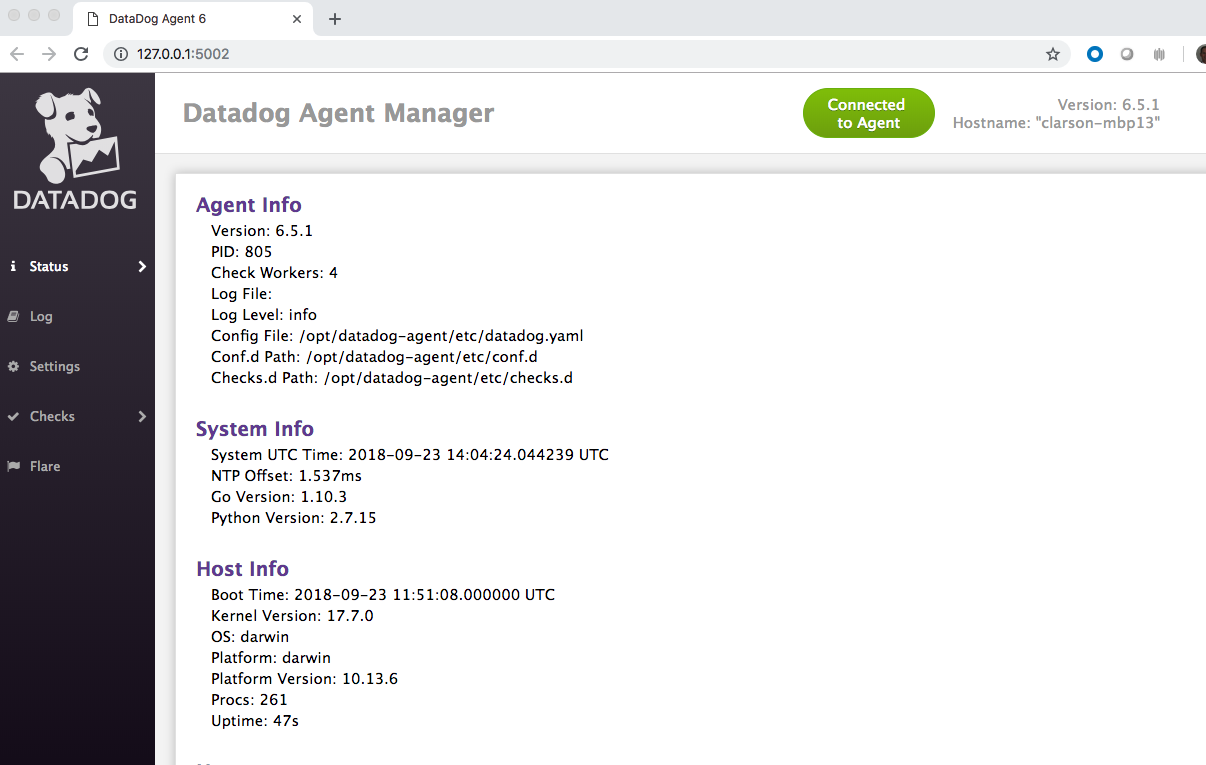


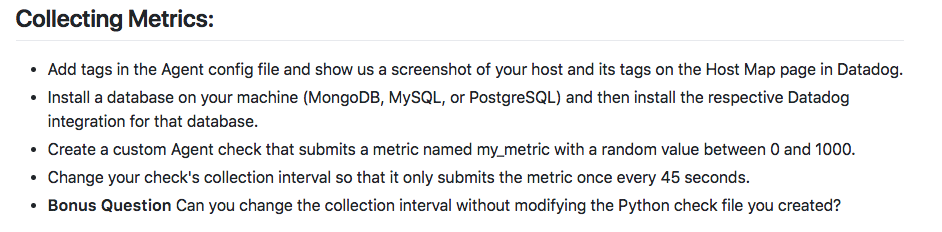


and install the agent on my vagrant vm

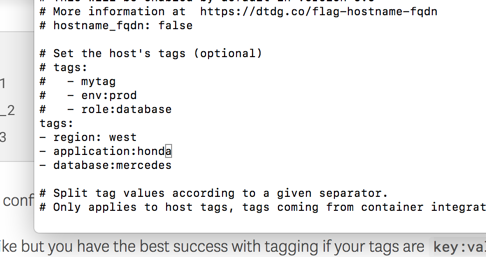


The agent GUI is friendly and informative:

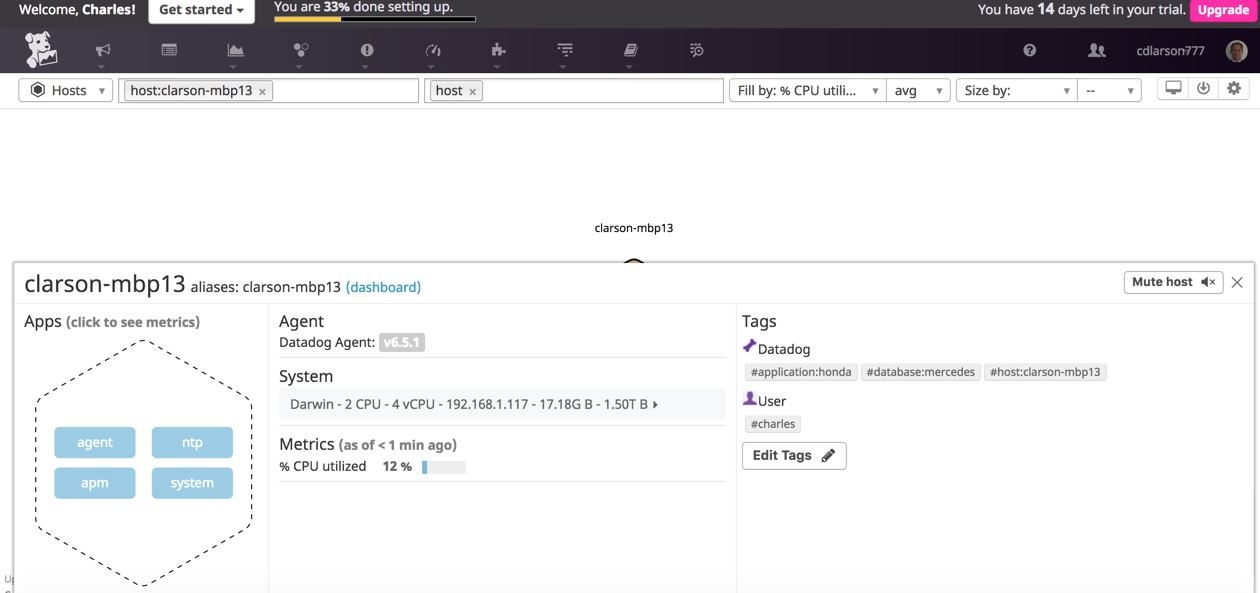


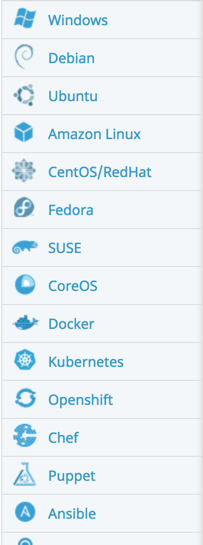


I created my environment and some tags…

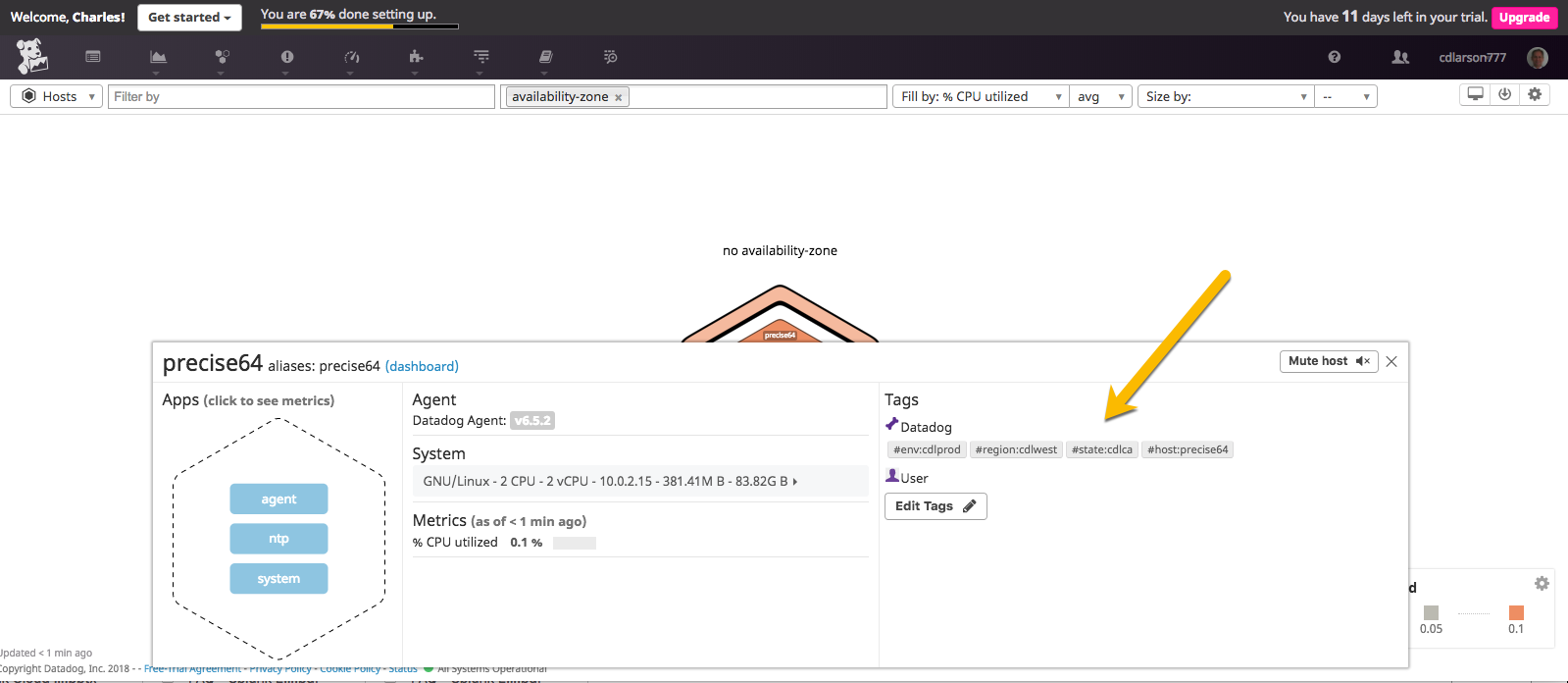


…first on my mac…

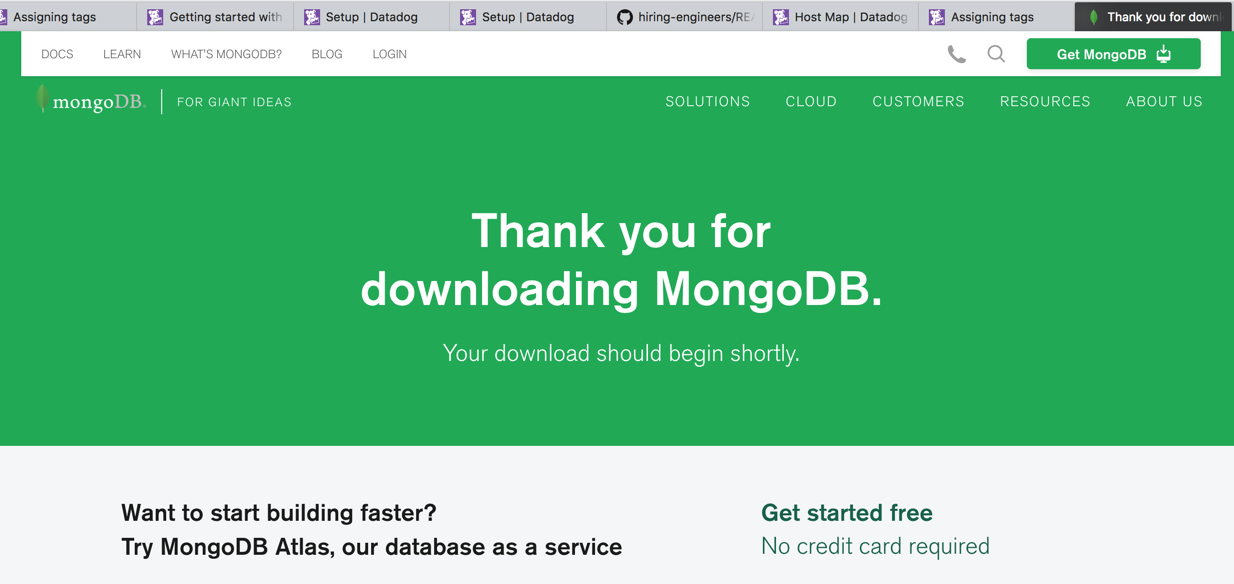




Here is my vagrant Ubuntu guest, showing my tags:



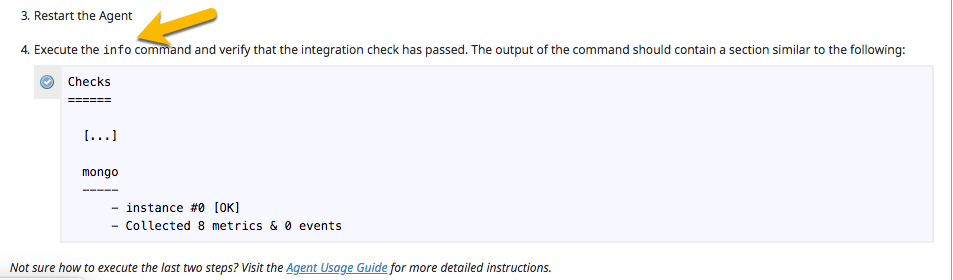
For my database, I chose MongoDB…



The parameters in Datadog mention V2 or V3, but V4 is the current product. After being an SE for many years, this raises questions in my mind. *Sometimes these are a big deal and sometimes they’re nothing*. I used the V3 parameters and forged ahead.

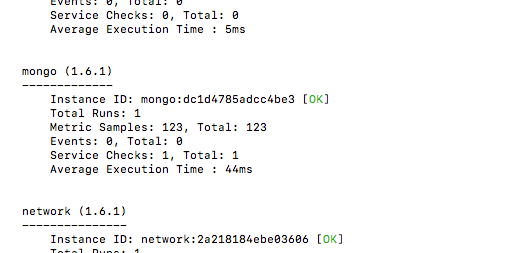


had to create the admin user first



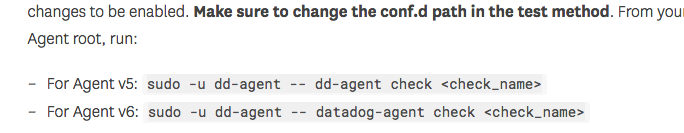
I eventually figured out whose “info” command was intended here, and saw that Mongo was

reporting properly.

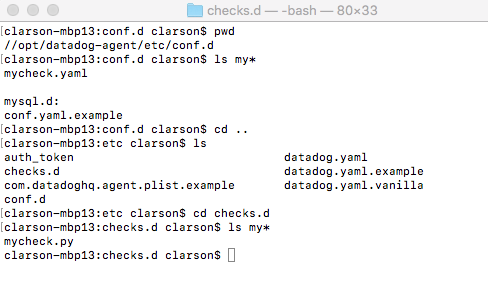




Agent check

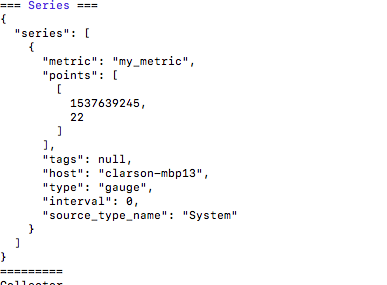


This warning was in bold print, but it was unclear to me what change and what test method were intended. But I did get the Agent check working.

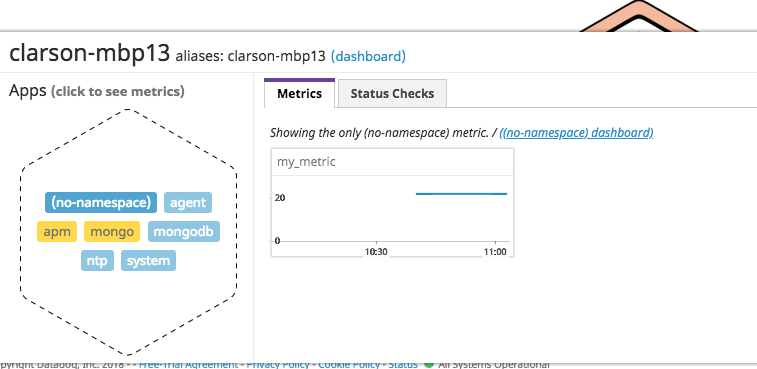


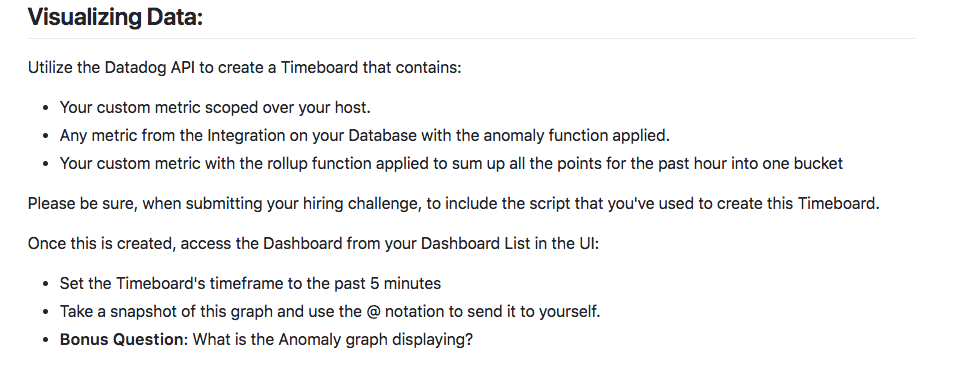


below I test out mycheck

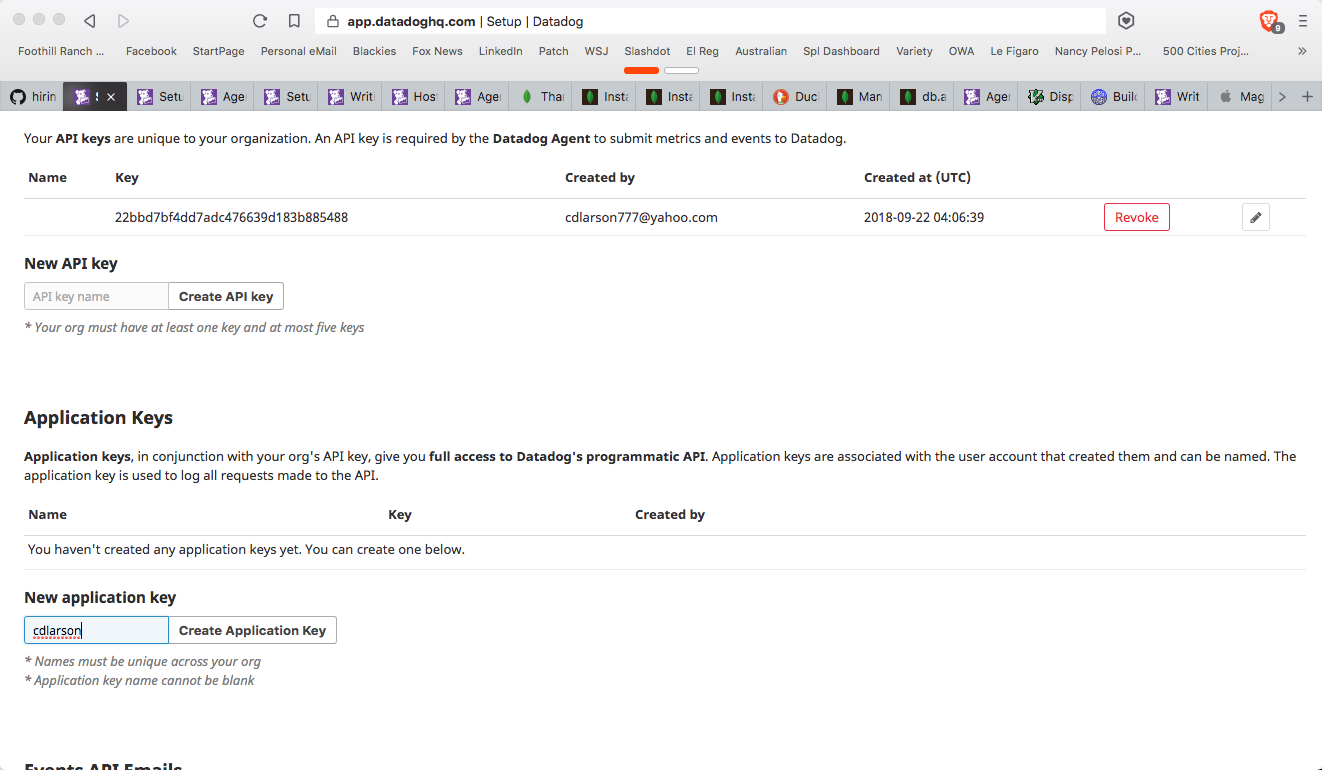


my\_metric is now showing up:

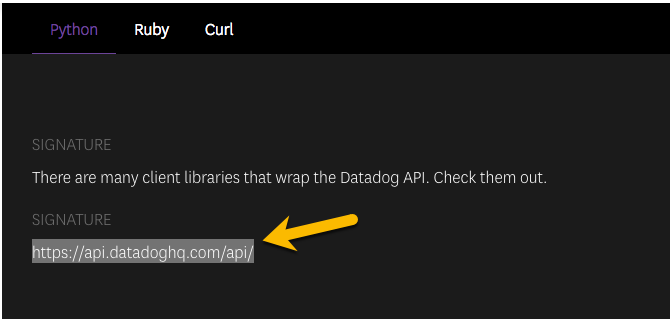




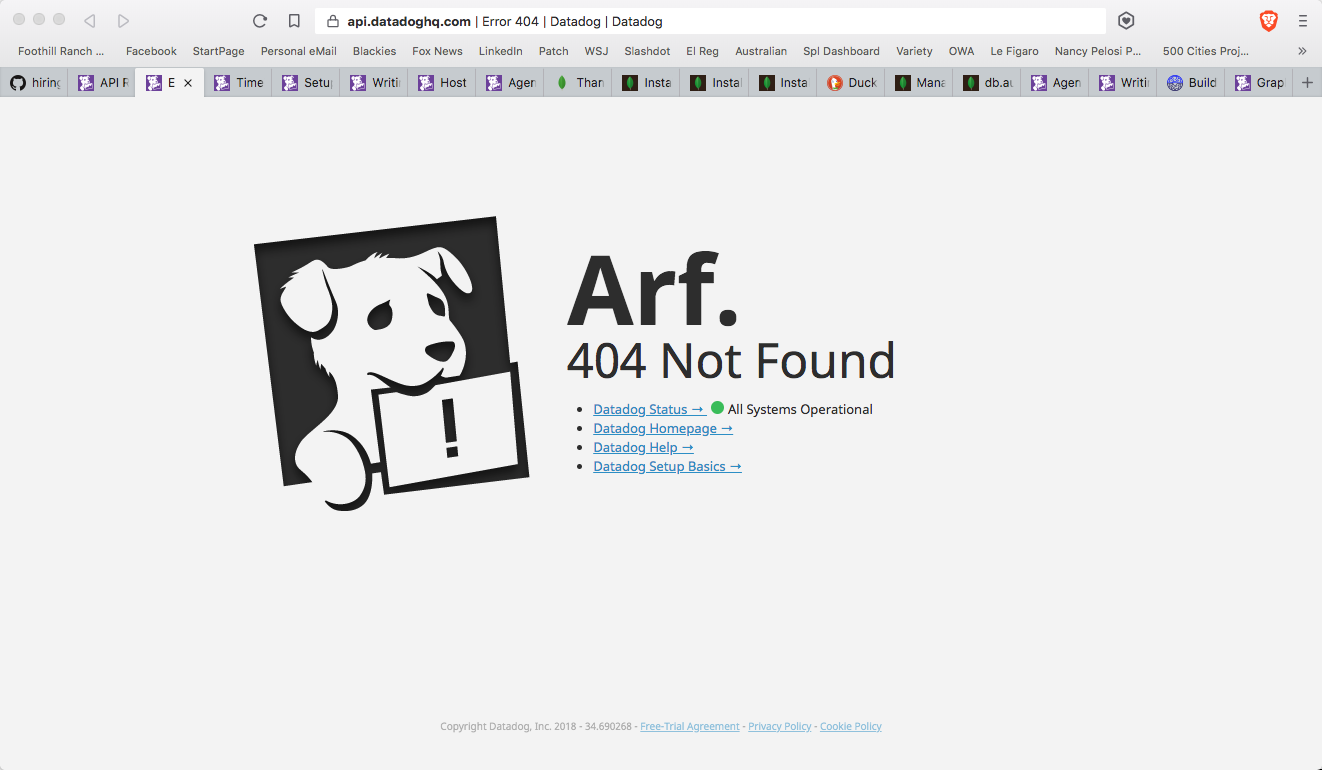
I got the API and Application keys.



Let’s do the Timeboard!



…but this link produces a 404; I reported this on the **github** page.



I prepared my first cut at an API timeboard in hope that the libraries would materialize:

from datadog import initialize, api

options = {

'api\_key': '22bbd7bf4dd7adc476639d183b885488',

'app\_key': '963976d8595083b9af5a37379898557981817319'

}

initialize(\*\*options)

title = "CDL Timeboard"

description = "An informative timeboard."

graphs = [{

"definition": {

"events": [],

"requests": [

{"q": "avg:system.mem.free{\*}"}

],

"viz": "timeseries"

},

"title": "Average Memory Free"

}]

template\_variables = [{

"name": "host1",

"prefix": "host",

"default": "host:my-host"

}]

read\_only = True

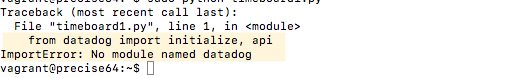
api.Timeboard.create(title=title,

description=description,

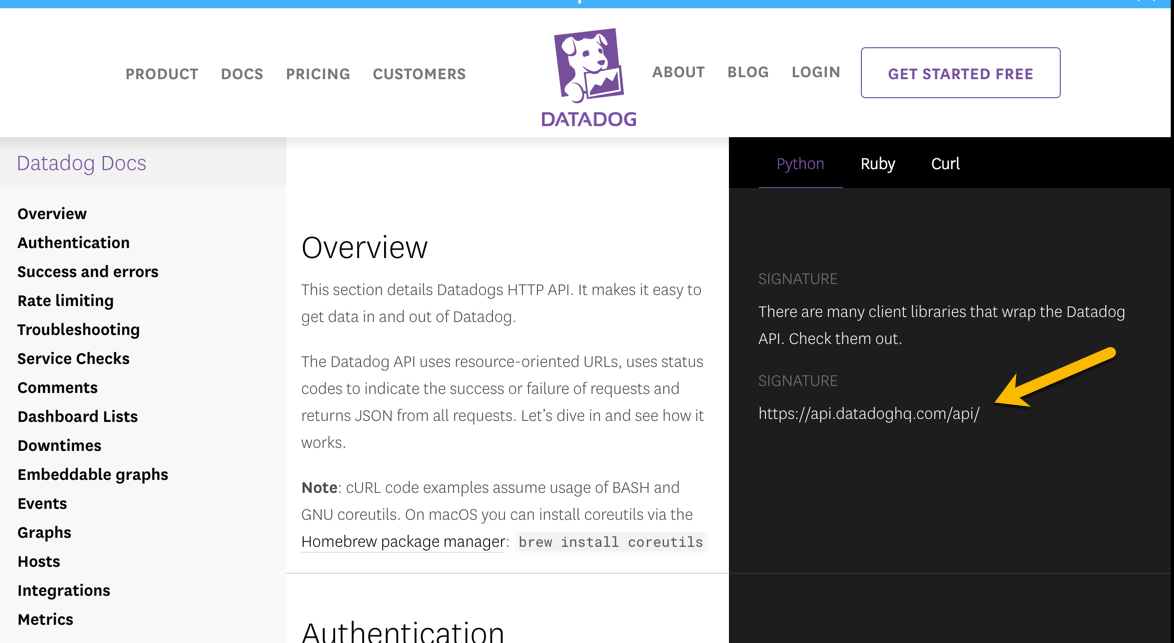
graphs=graphs,

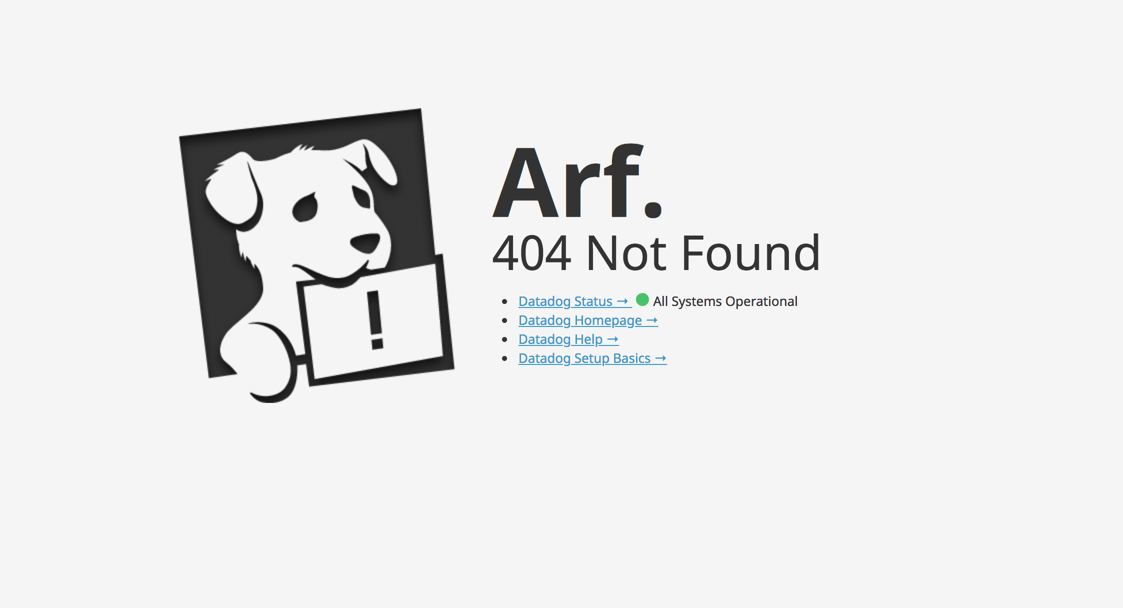
template\_variables=template\_variables,

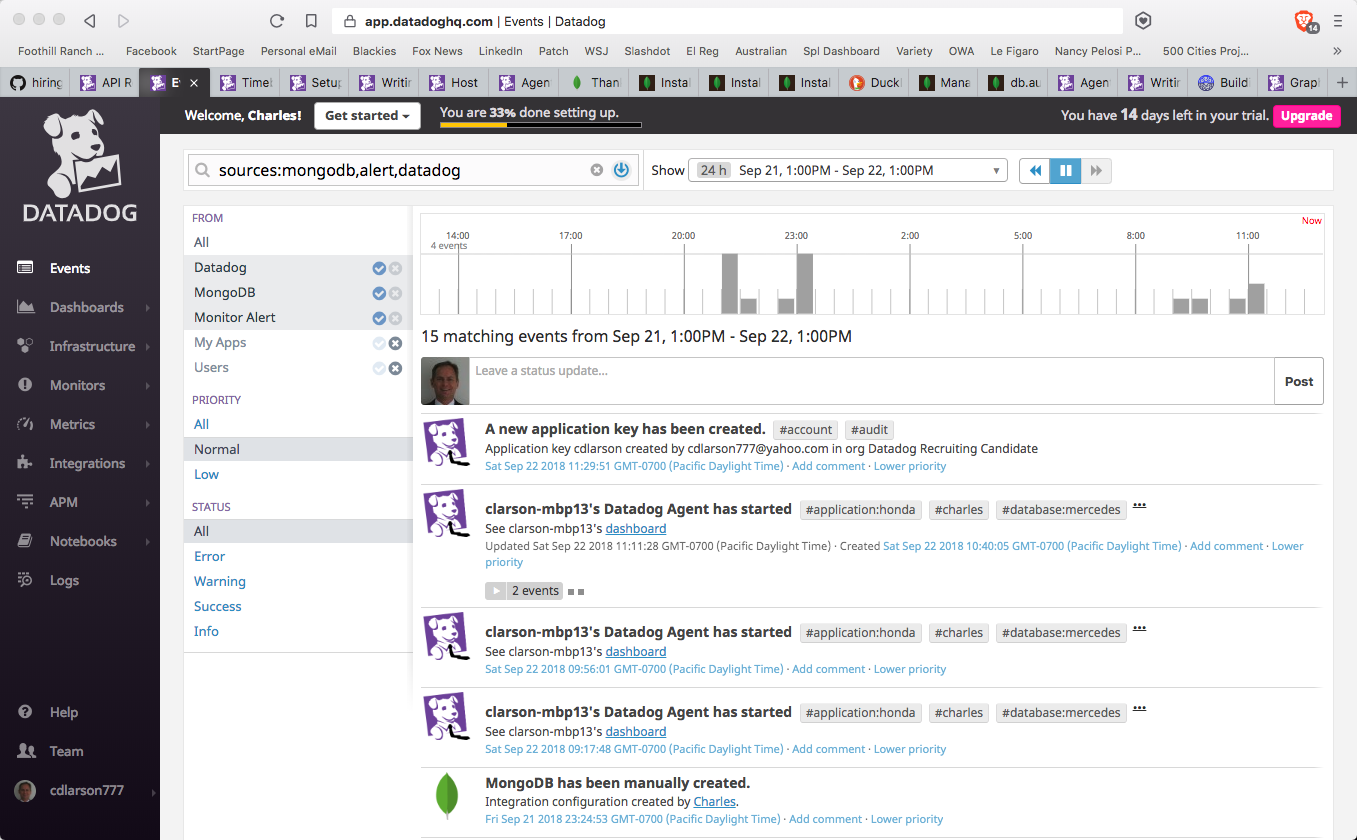
read\_only=read\_only)



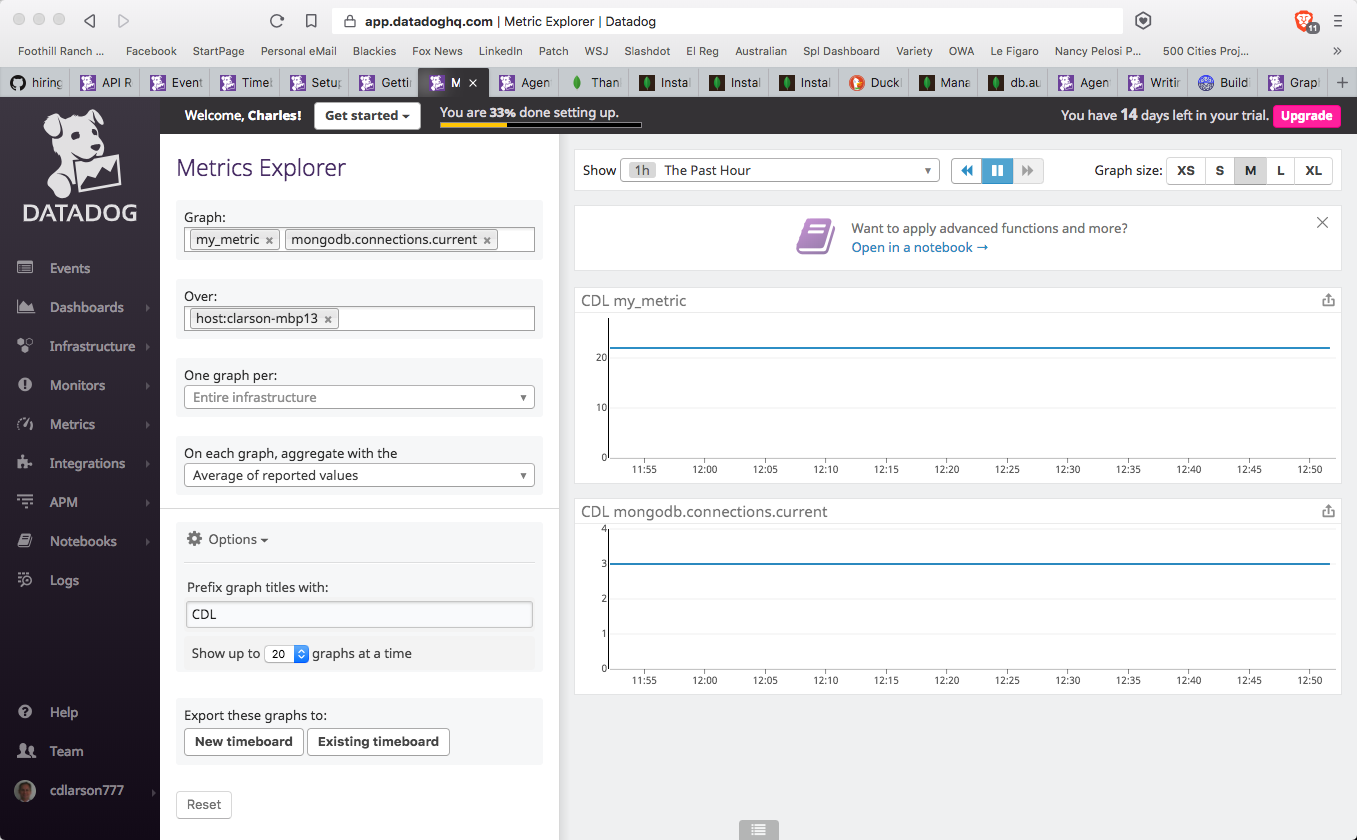
Hmmm. Or should I say “Arf”. Looks like I need those libraries. If I follow the given link to where I think the libraries I need should be…



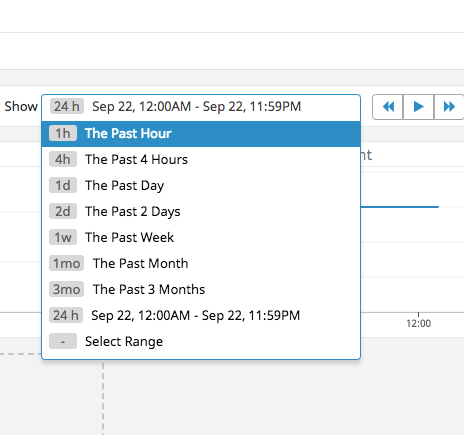
  
  
…Arf! There’s our friend again.

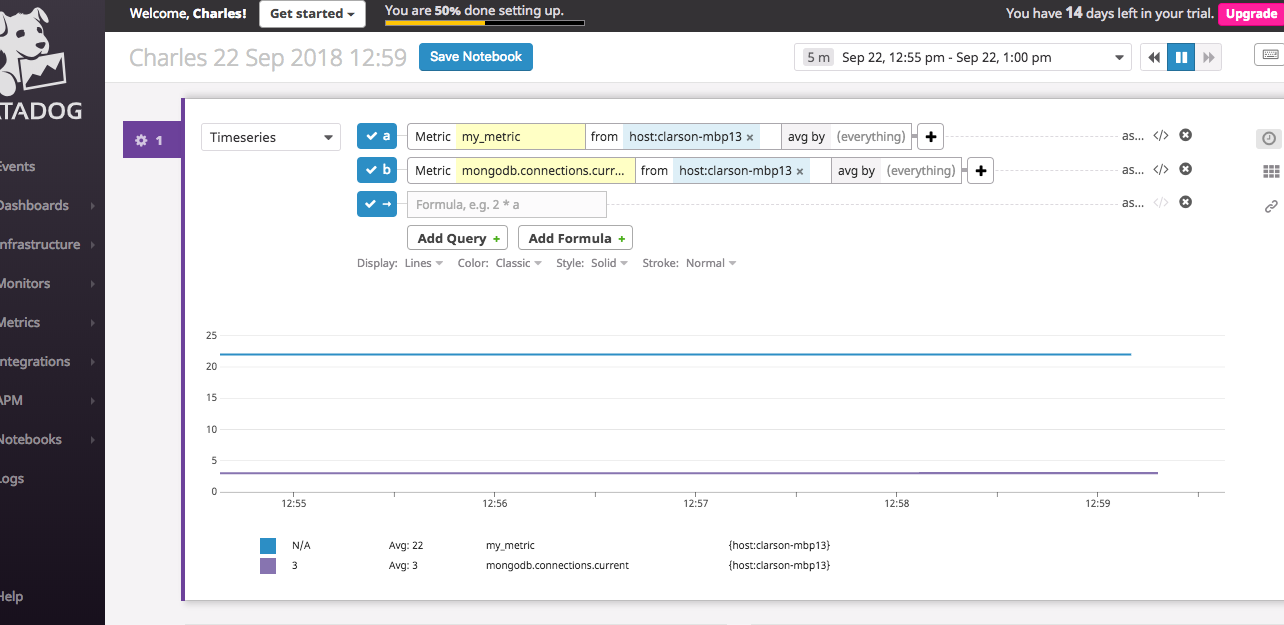


Since the API is proving a bit tough to figure out, create the Timeboard using the GUI for now:



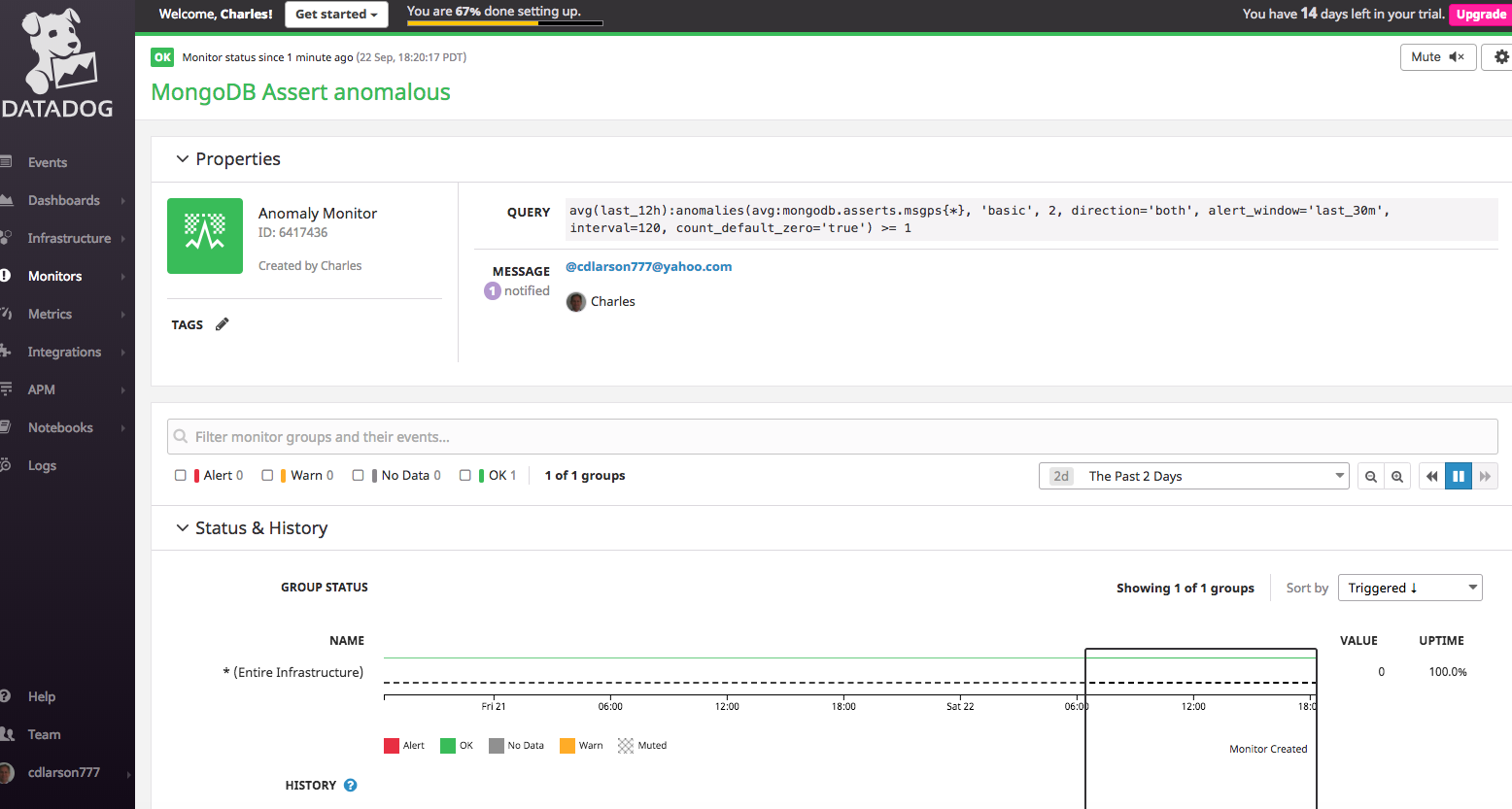
It’s called CDL Timeboard 1



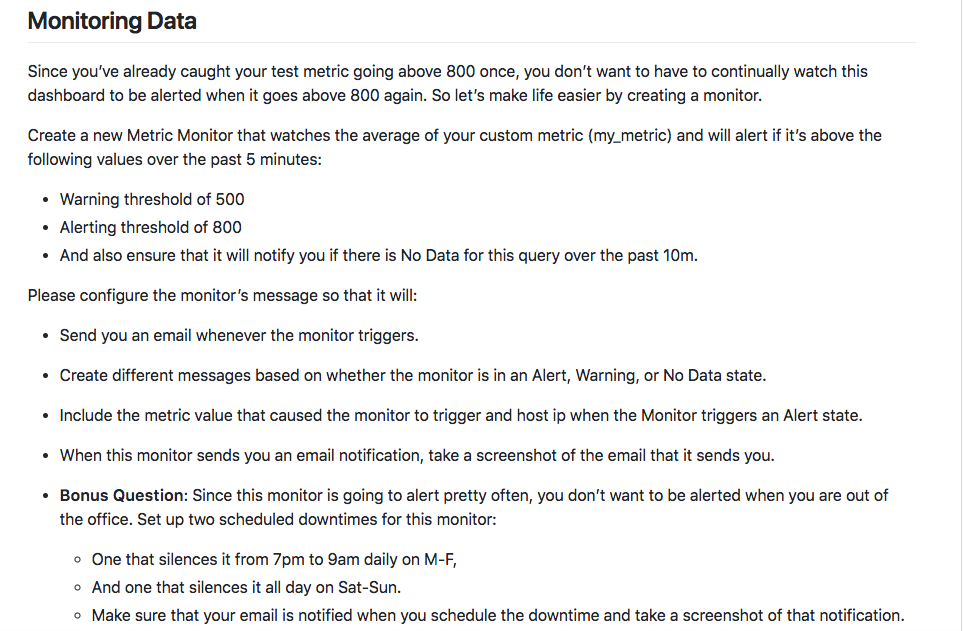




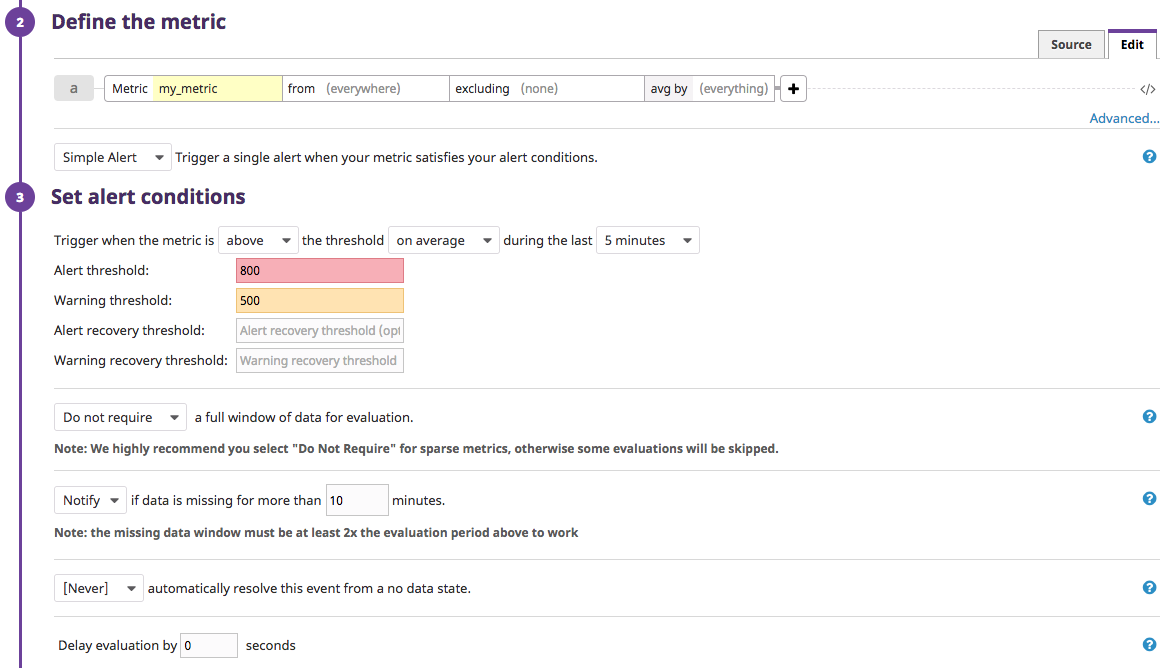
Now to create an Anomaly Monitor use case.

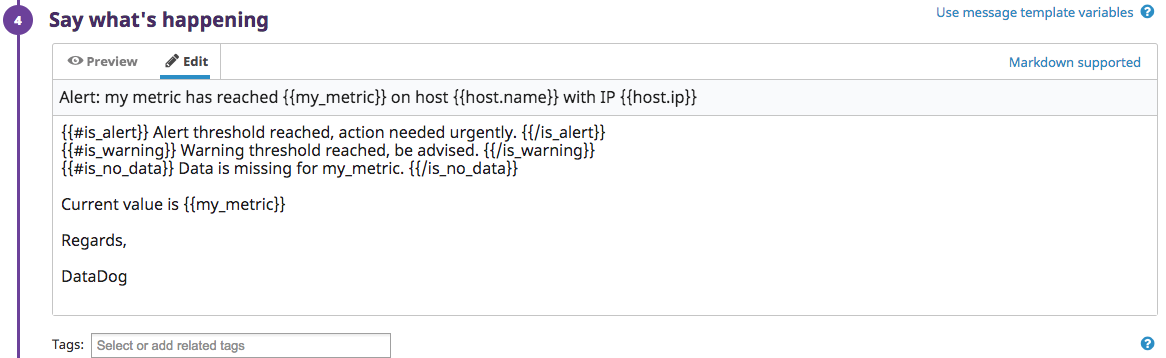


Above I’ve created an anomalous study of a MongoDB metric, but it’s unclear how to install it on my Timeboard.

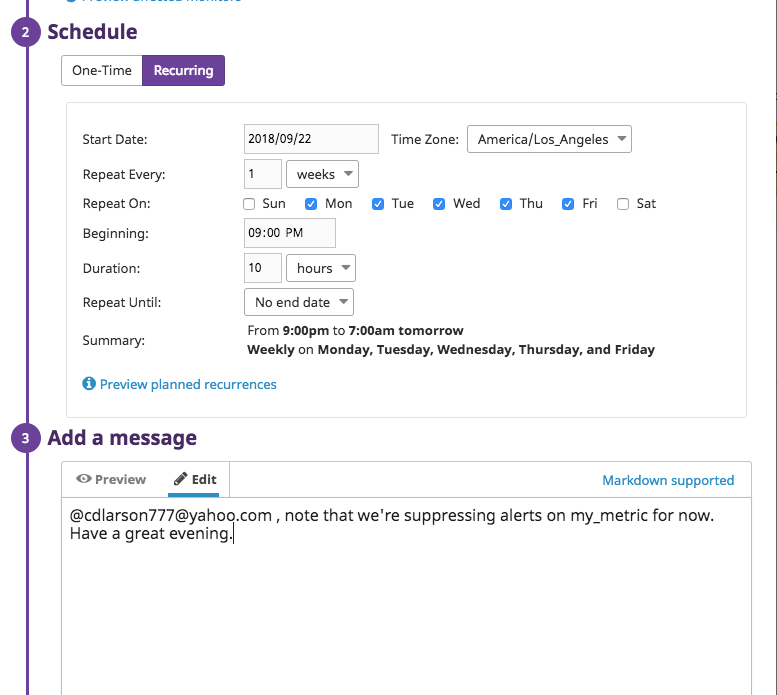


So now I create my monitors

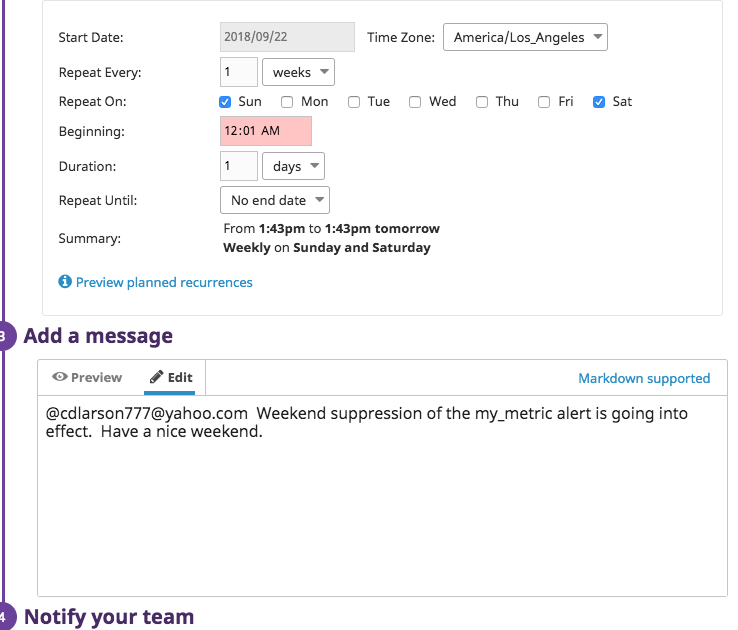




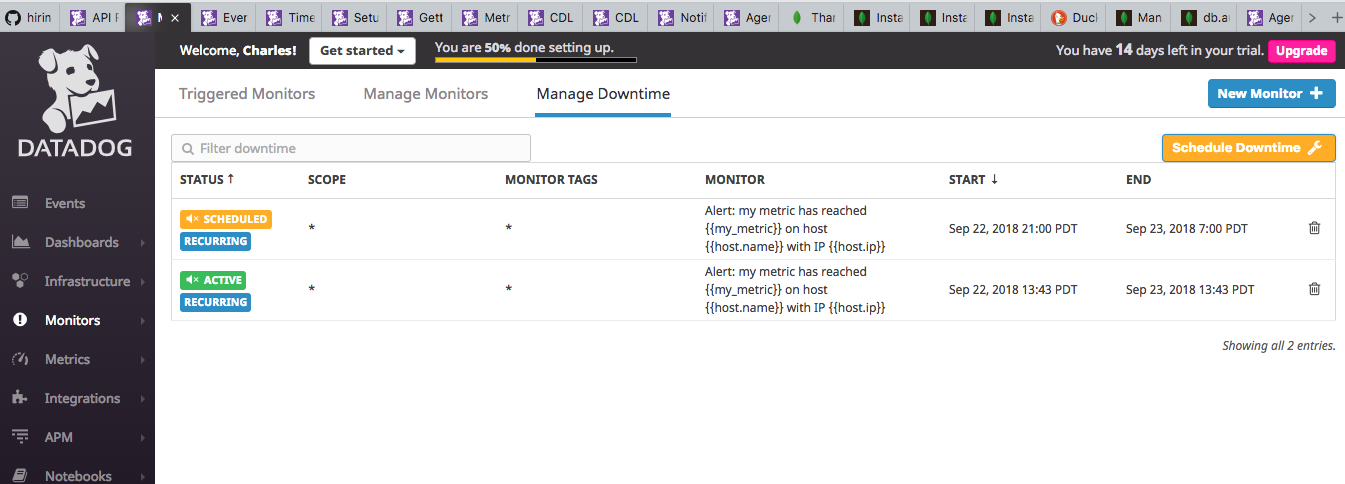
Extra Credit: Here we set up schedule exclusions, plus a suppression message:



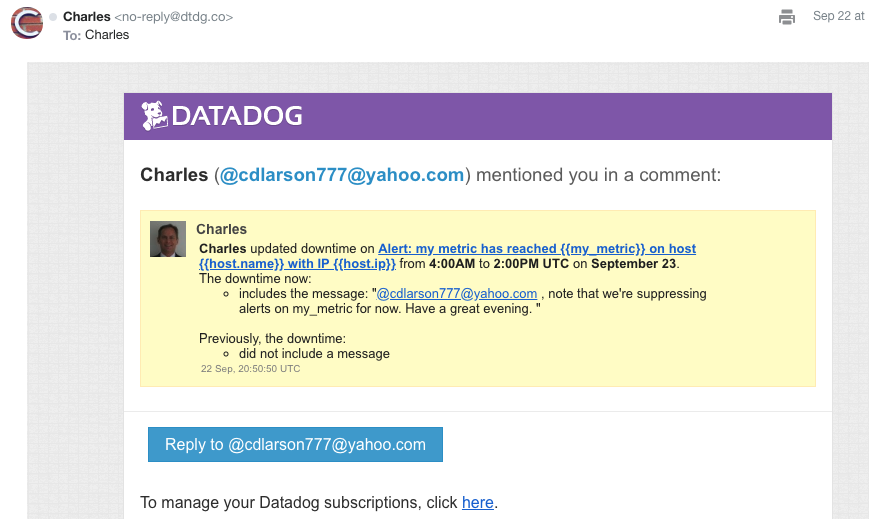
…and here is the second.

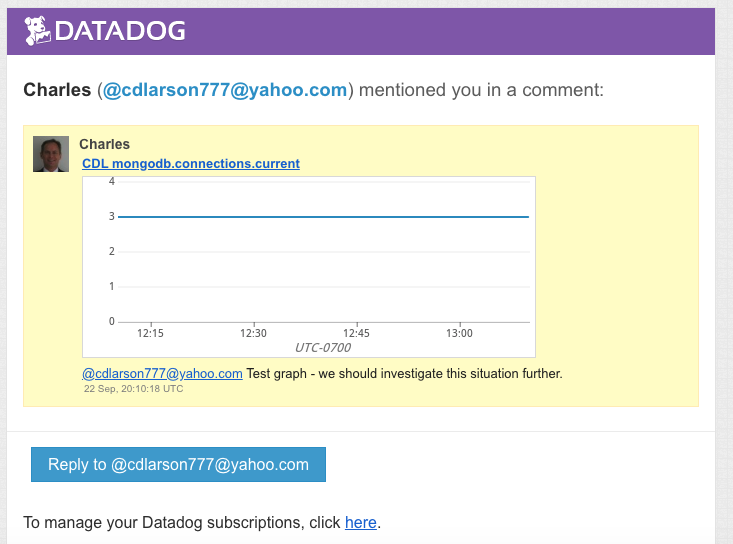


Here’s how they appear when completed and active:

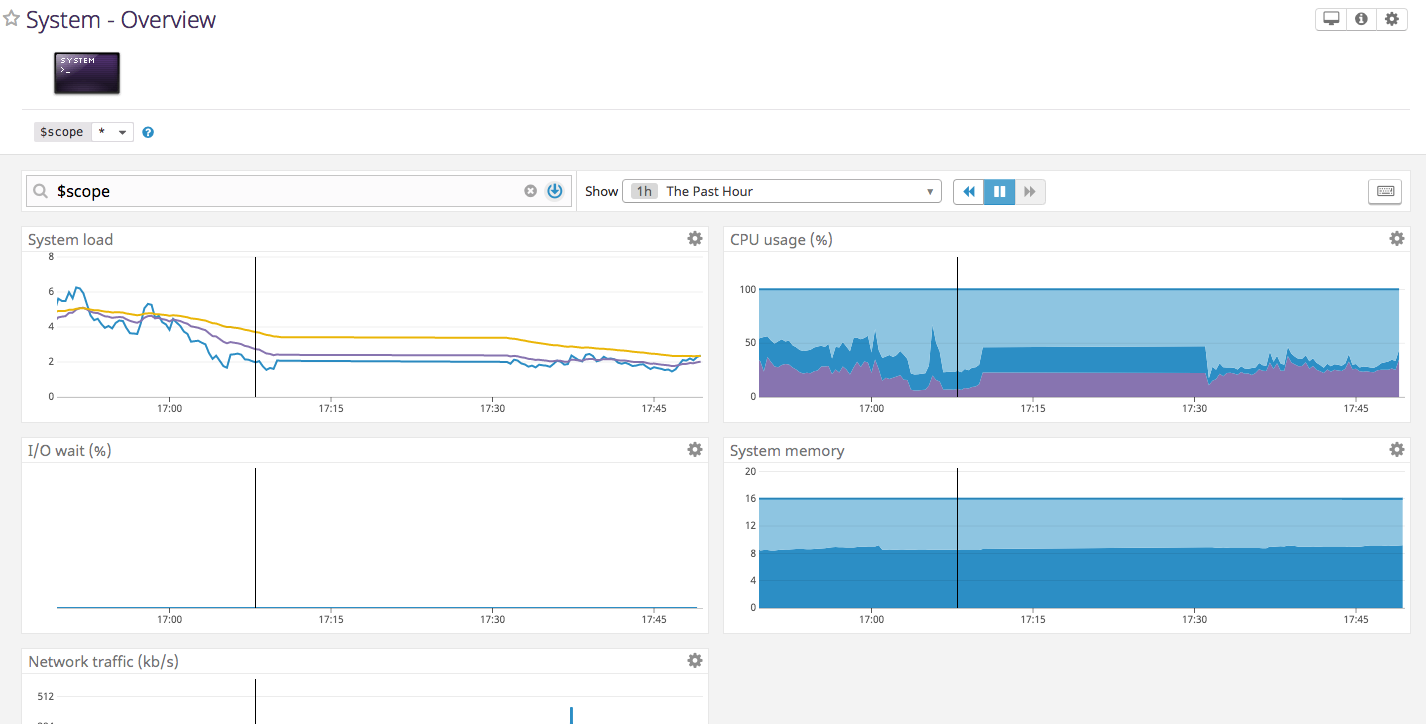


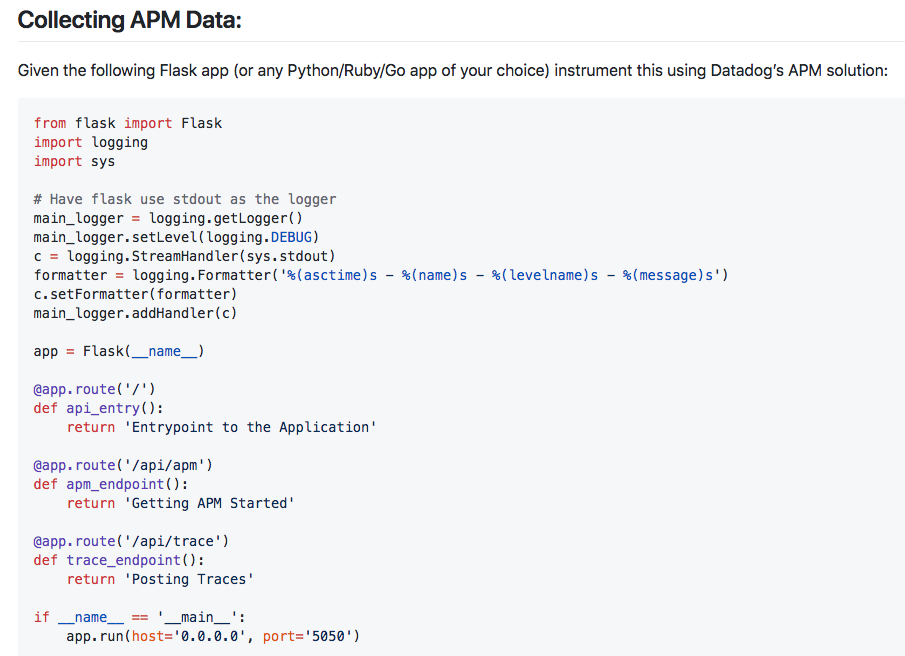
..and here’s the notification to my personal email:

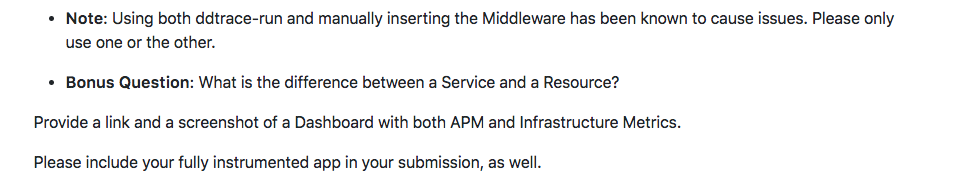


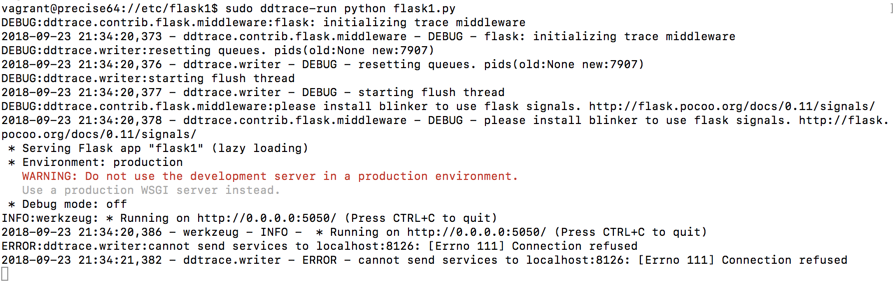


How cool is this.

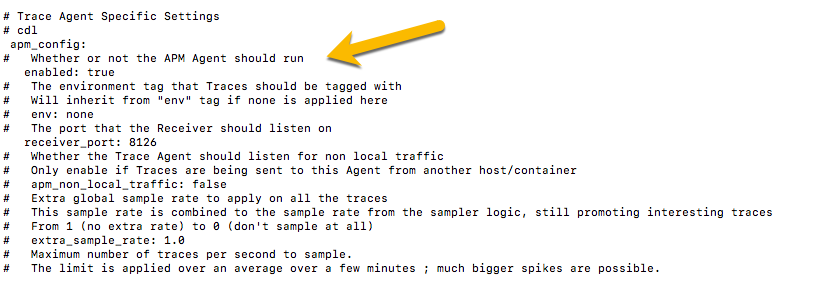




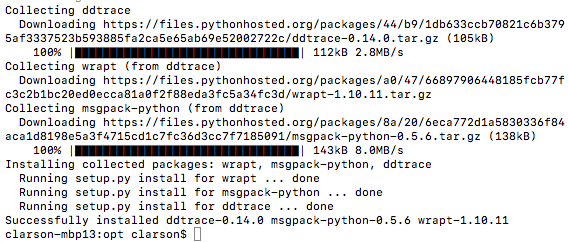


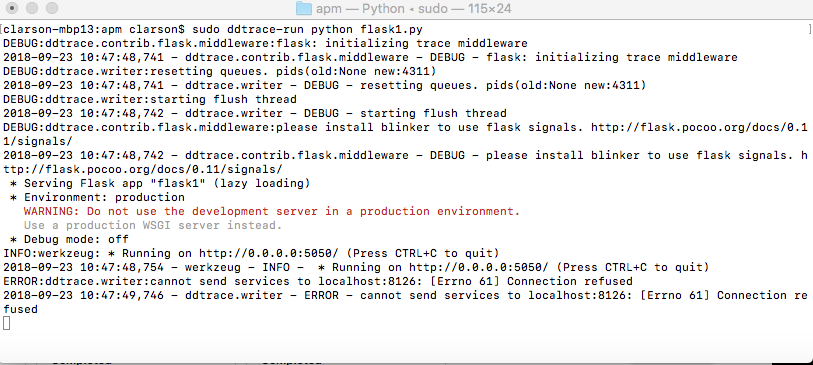


Enabling the APM options in yaml



installing ddtrace:





I elected the ddtrace-run method, so I didn’t introduce middleware into it. So here’s my “fully instrumented app”

from flask import Flask

import logging

import sys

# Have flask use stdout as the logger

main\_logger = logging.getLogger()

main\_logger.setLevel(logging.DEBUG)

c = logging.StreamHandler(sys.stdout)

formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')

c.setFormatter(formatter)

main\_logger.addHandler(c)

app = Flask(\_\_name\_\_)

@app.route('/')

def api\_entry():

return 'Entrypoint to the Application'

@app.route('/api/apm')

def apm\_endpoint():

return 'Getting APM Started'

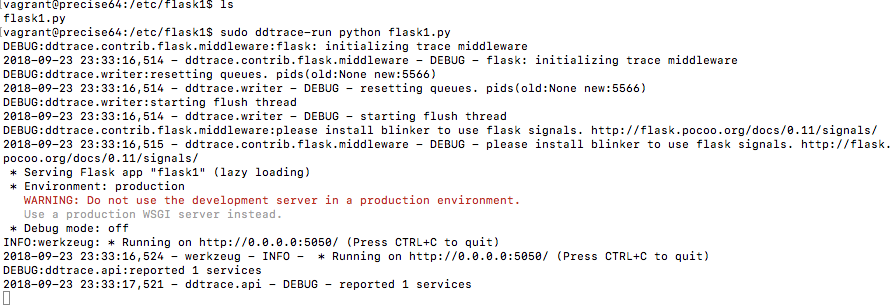
@app.route('/api/trace')

def trace\_endpoint():

return 'Posting Traces'

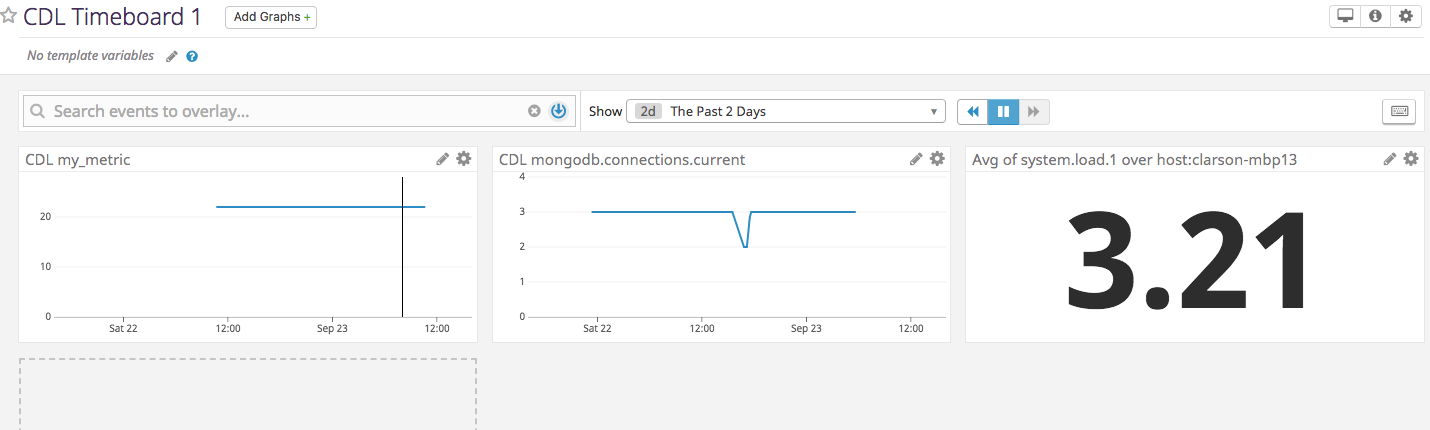
if \_\_name\_\_ == '\_\_main\_\_':

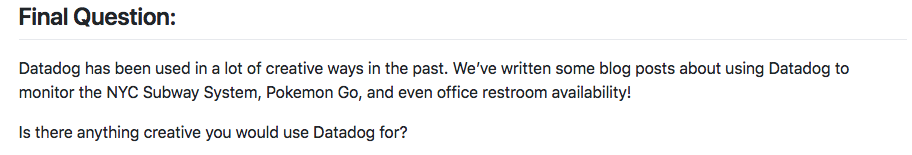
app.run(host='0.0.0.0', port='5050')



Still not reporting, though. Apparently this rev of Flask doesn’t [easily integrate with DDtrace](http://pypi.datadoghq.com/trace/docs/#module-ddtrace.contrib.flask).

Checking back at my [TimeBoard,](https://app.datadoghq.com/dash/924157/cdl-timeboard-1?live=true&page=0&is_auto=false&from_ts=1537571508815&to_ts=1537744308815&tile_size=m) where I’ve done some minor customization:





>> Track the spread of communicable diseases, particularly in developing countries, to see how life-saving strategies could be improved. Data, when available, has vast power for good.

Respectfully Submitted,

*Charles D. Larson, Jr.*