My Datadog Experience

Michael Shaughnessy

I'm about to tell you about the experience I had during my test of the datadog agent as part of the application process for the position of solutions engineer. In order to make my explanations as clear as possible, I'm going to present the directions for each challenge one-by-one along with screenshots and some commentary about how I completed the challenge and dealt with any issues that came up.

Off we go to step one!

Add tags in the Agent config file and show us a screenshot of your host and its tags on the Host Map page in Datadog.

This part proved to be more difficult than I anticipated. For my first try, I added the tags directly through environment variables when calling the docker command from the command line. This seemed to work well enough but some tags didn't show up and I noticed some odd behavior in the host map. I found out later that the problem came from a combination of factors including the way I was passing the environment variables and the fact that I was attempting to build the docker image each time I ran it.

If my understanding is correct, the result of this was that I ended up with several containers running at the same time and the Datadog Agent considered each of them to be individual hosts. The screenshots starting on the next page show my web host once all my tags were applied and, further down, there is a screenshot of a much saner layout after I switched to a non-dockerized version of the agent on my Macbook.
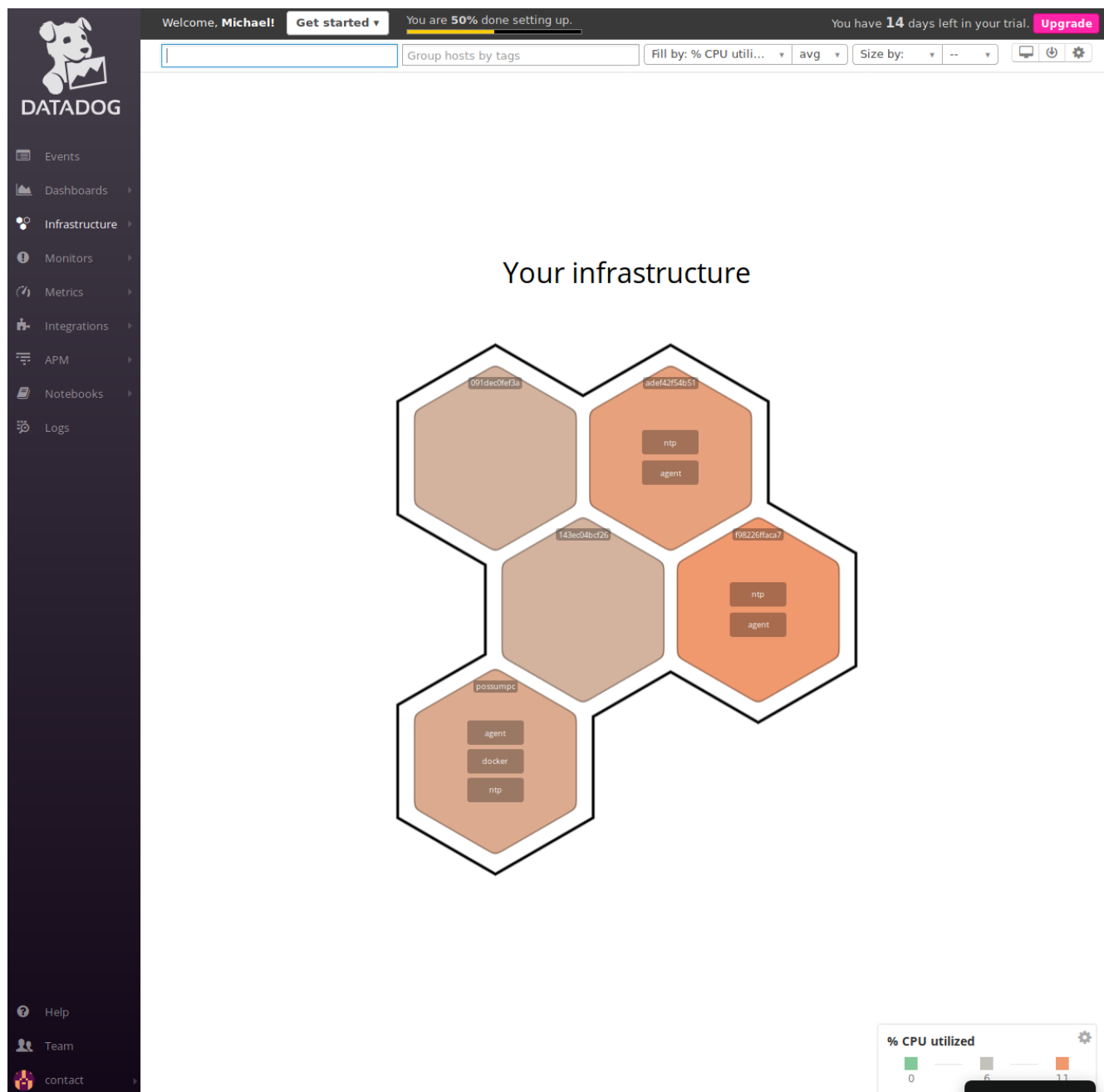
Illustration 1: *Some of my tags ended up running as individual hosts, resulting in the overcomplicated setup shown here.*
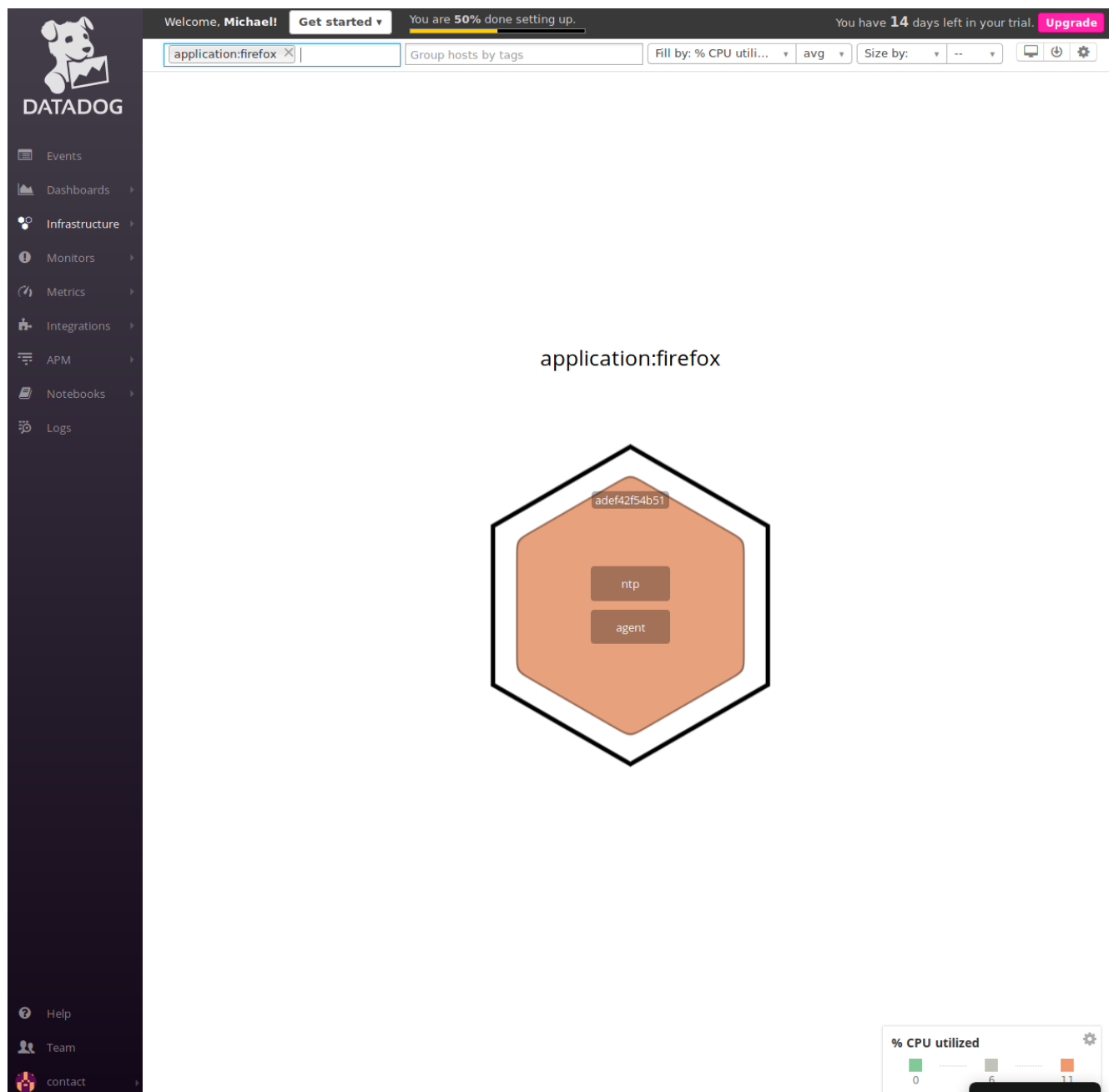
Illustration 2: *Example of a tag running under its own host node.*

Illustration 3*: After I switched to my Macbook, I decided to try adding the tags through the datadog config file. Here is the entire Host Map of my Macbook along with the three tags I added in the configs (shown bottom right).*

```
Install a database on your machine (MongoDB, MySQL, or PostgreSQL) and
then install the respective Datadog integration for that database.
```

This part went smoothly. After installing a PostgreSQL database on my computer, I installed the integration through the agent gui. The process was simple and intuitive. The preset dashboards for my PostgreSQL integration can be found at the following addresses:

- https://app.datadoghq.com/screen/integration/235/postgres---overview

- https://app.datadoghq.com/dash/integration/17/postgres---metrics?
live=true&page=0&is_auto=false&from_ts=1522757930583&to_ts=152276
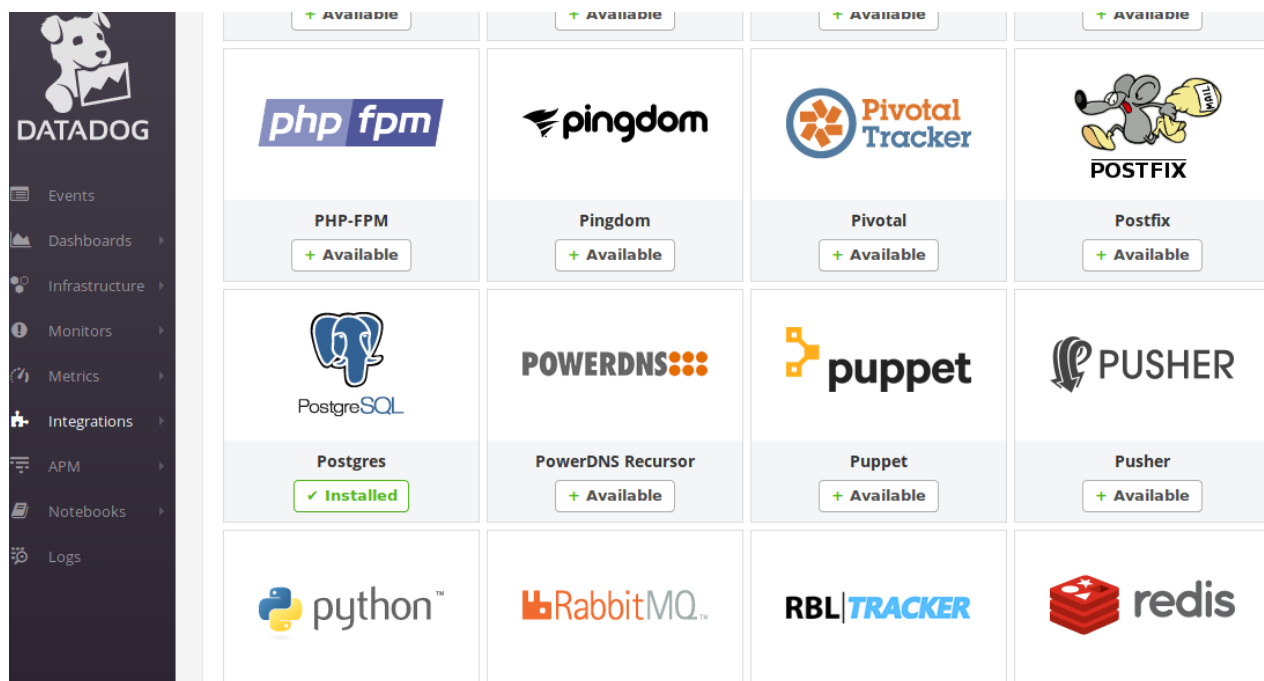1530583&tile_size=m

Illustration 4*: The instructions given in the integrations section are clear and precise, making the installation process as close to a one-click install as it possibly can be. Disclaimer: in my excitement, I may have clicked more than once.*
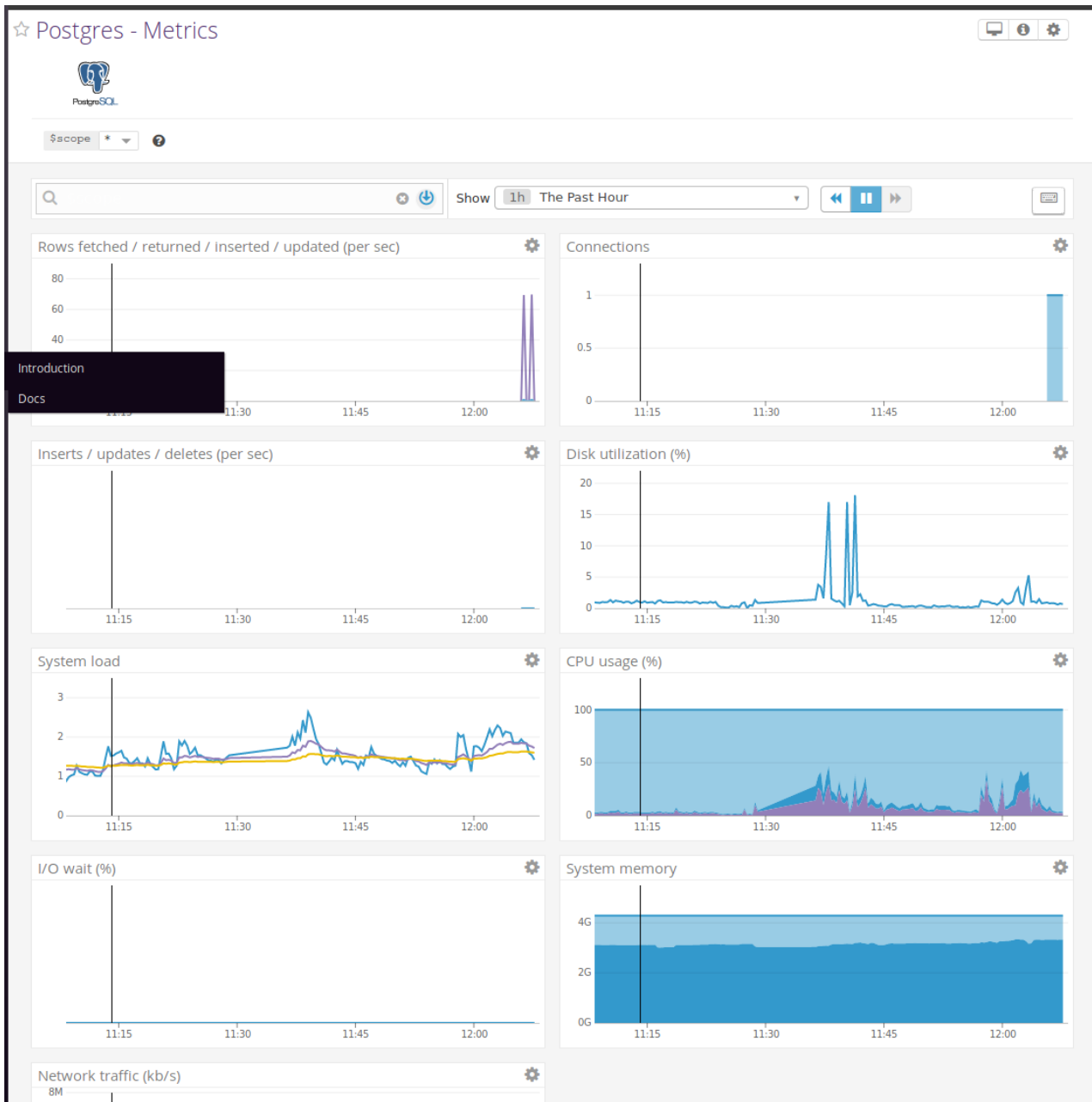
Illustration 5: *Postgres dashboard showing all the metrics collected from my database.*

```
Create a custom Agent check that submits a metric named my_metric with
a random value between 0 and 1000.
```

This was the first moment where I started to run into some trouble. I don't have a lot of experience with docker and as I read the Datadog documentation I struggled to understand how to go about passing files from my local environment to the docker container. Without being able to see the file structure inside the docker container, I had a hard time figuring out what folders I needed to inject my custom files into. I gave it a valiant effort but, in the end, I decided that it would be easier to create a local install of Datadog on my Macbook so that I could see the file structure directly and interact with it more easily. As luck would have it, this also solved the small problem I had above with adding tags. Here is a link to a dashboard showing all the values of my_metric since its creation: https://app.datadoghq.com/dash/753870/my-metric-raw-values?live=true&page=0&is_auto=false&from_ts=1522528842609&to_ts=1522701642 609&tile_size=m

```python
1 from checks import AgentCheck
2 from random import randint
3
4 class MyCheck(AgentCheck):
5     def check(self, instance):
6         randomNumber = randint(0, 1000)
7         self.gauge('my_metric', randomNumber)
```

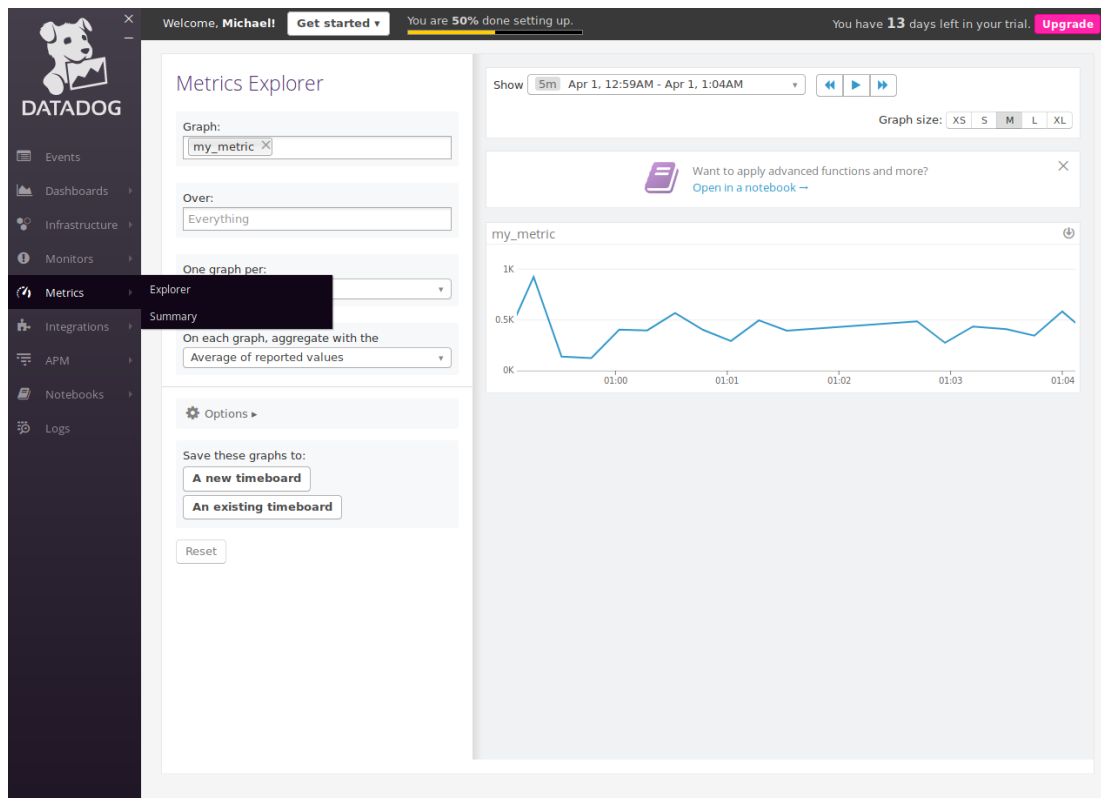***Illustration 6****: The custom agent check that I created.*

Illustration 7: The values collected for my custom check over a 5 minute period. By default, the check is running every 20 seconds.

```
Change your check's collection interval so that it only submits the
metric once every 45 seconds.
```

This step was easy once I figured out how to add the check. I simply had to find the associated config file (conf.yaml.default) under conf.d/myCheck.d/ and edit its instance variable to collect every 45 seconds.

Illustration 8: min_collection_interval modifies the collection interval from its default of 20 seconds.
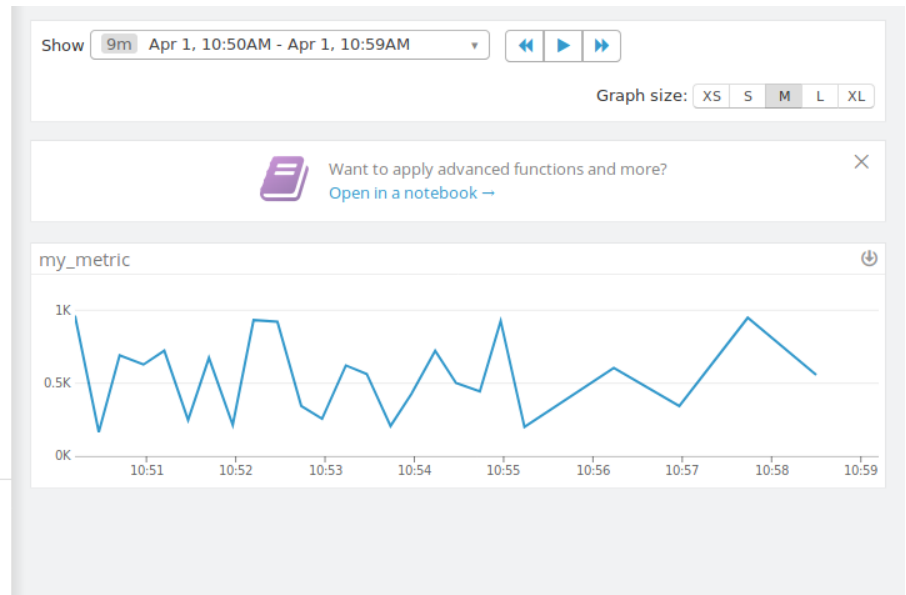
Illustration 9*: Here we can see that my custom metric started reporting less frequently after I changed the minimum collection interval.*

```
Utilize the Datadog API to create a Timeboard that contains:
- Your custom metric scoped over your host.
- Any metric from the Integration on your Database with the anomaly
function applied.
- Your custom metric with the rollup function applied to sum up all
the points for the past hour into one bucket
- Please be sure, when submitting your hiring challenge, to include
the script that you've used to create this Timemboard.
```

This part of the challenge was very straightforward. The gui for creating timeboards is easy to follow and provides all the options you need. The ability to switch into json-editing mode is a nice feature too and makes it easy to tweak the timeboard to your exact preferences. Here is the link to the timeboard that I created:

- https://app.datadoghq.com/dash/750319/my-metric?live=true&page=0&is_auto=false&from_ts=1522617338505&to_ts=1522703738505&tile_size=m
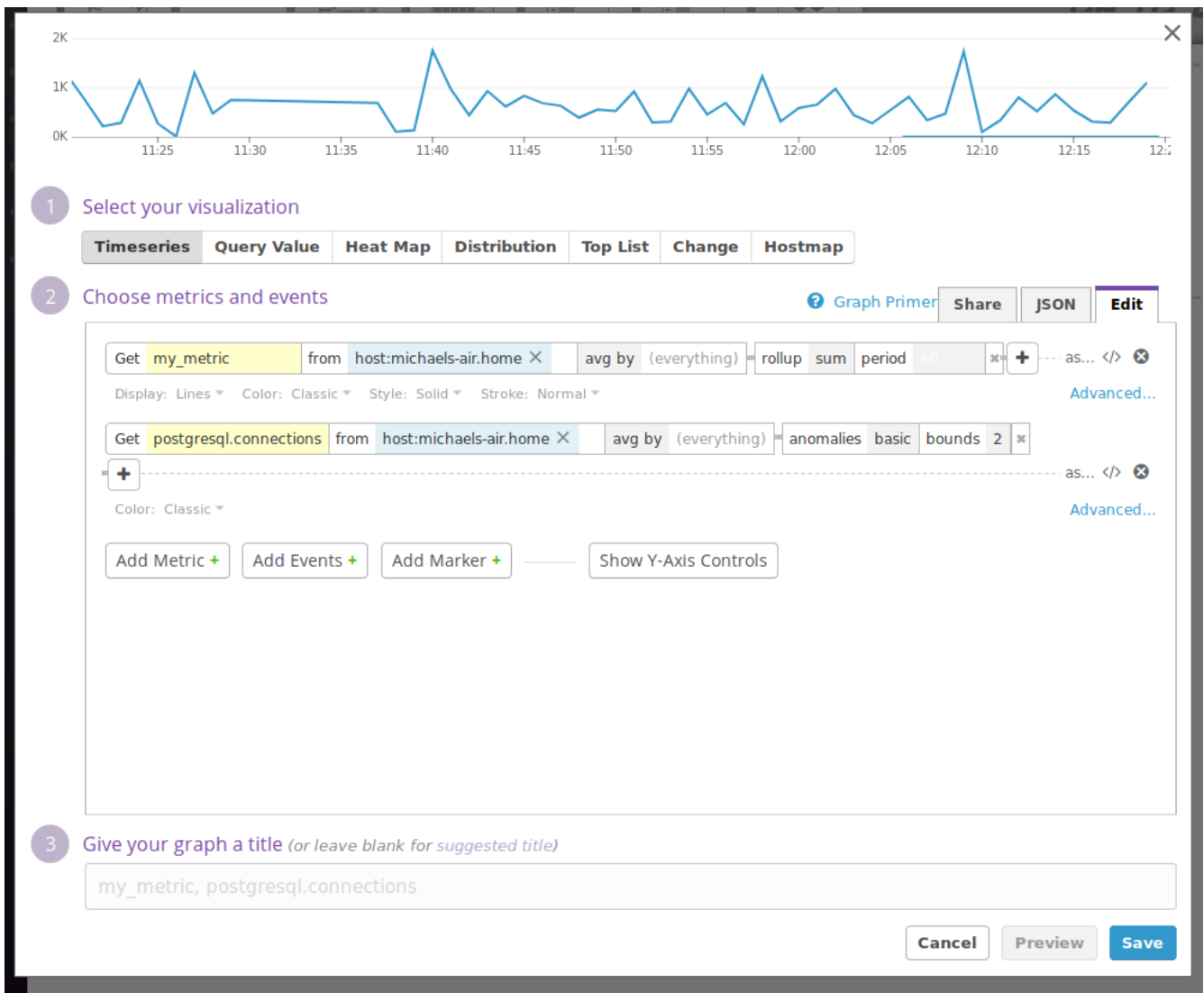
Illustration 10*: The timeboard representation screen. The lines under "Choose metrics and events" show the metric in field 1 followed by its scope in field 2. Field 3 indicates that we want to see an average of values and, finally, the last field specifies a function to be applied to the metric.*

```
 1 {
 2   "requests": [
 3     {
 4       "q": "avg:my_metric{host:michaels-air.home}.rollup(sum, 60)",
 5       "type": "line",
 6       "style": {
 7         "palette": "dog_classic",
 8         "type": "solid",
 9         "width": "normal"
10       },
11       "conditional_formats": [],
12       "aggregator": "avg"
13     },
14     {
15       "q": "anomalies(avg:postgresql.connections{host:michaels-air.home}, 'b
   asic', 2)",
16       "type": "line",
17       "style": {
18         "palette": "dog_classic",
19         "type": "solid",
20         "width": "normal"
21       }
22     }
23   ],
24   "viz": "timeseries",
25   "autoscale": true
26 }
```

Illustration 11: *The same as the above screenshot but this time represented in JSON format.*

```
Once this is created, access the Dashboard from your Dashboard List in
the UI:
Set the Timeboard's timeframe to the past 5 minutes
Take a snapshot of this graph and use the @ notation to send it to
yourself.
```

Another easy challenge. I've gotten pretty used to @ notifications in apps such as Slack and Discord so this step felt very intuitive. Setting the timeboard's timeframe is also simple and intuitive: you just click and drag the mouse while highlighting to select the section of the timeboard that you want to zoom in on.
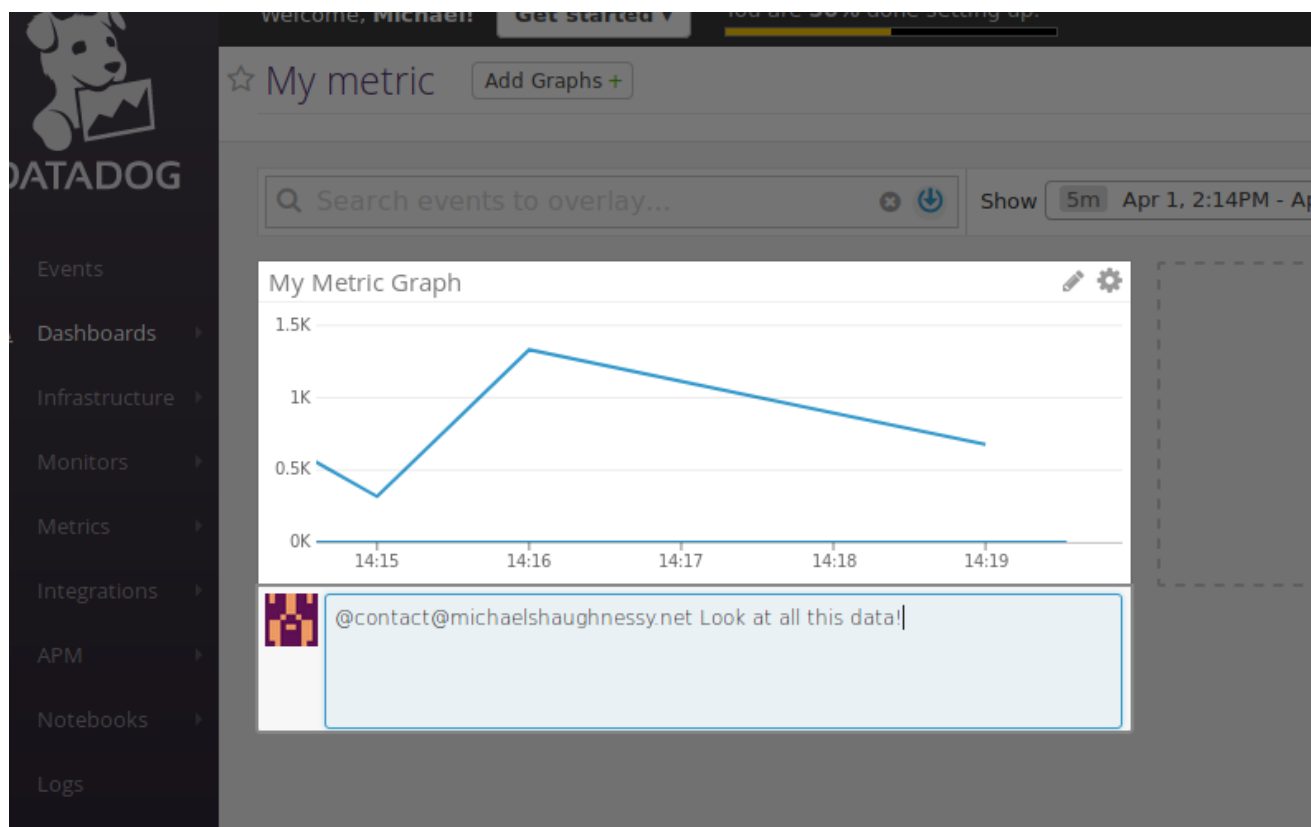
Illustration 12*: This data was so exciting I just had to tell myself about it!*
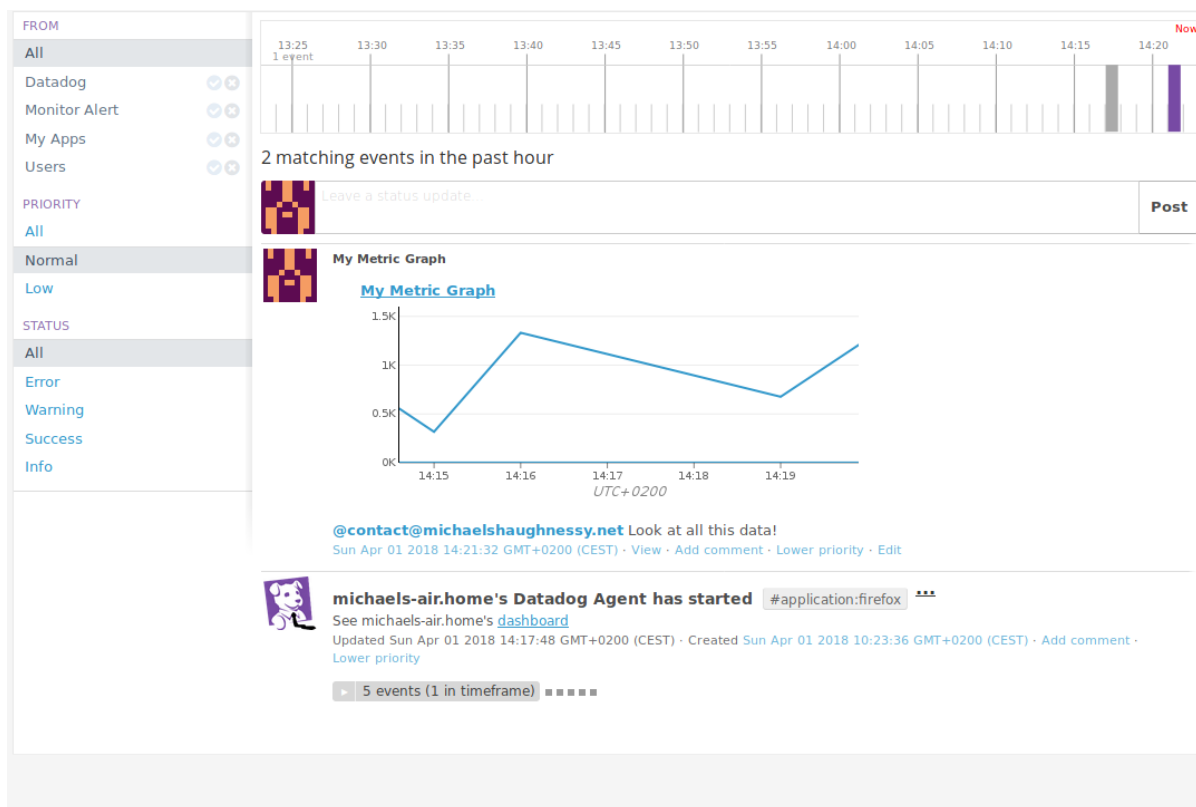

Illustration 13*: When I received the notification I was just as excited as when I sent it.*

Create a new Metric Monitor that watches the average of your custom
metric (my_metric) and will alert if it's above the following values
over the past 5 minutes:
- Warning threshold of 500
- Alerting threshold of 800
- And also ensure that it will notify you if there is No Data for this
query over the past 10m.
Please configure the monitor's message so that it will:
- Send you an email whenever the monitor triggers.
- Create different messages based on whether the monitor is in an
Alert, Warning, or No Data state.
- Include the metric value that caused the monitor to trigger and host
ip when the Monitor triggers an Alert state.
- When this monitor sends you an email notification, take a screenshot
of the email that it sends you.

This part was similar to the timeboard creation section in that the process was fairly straightforward in the gui. I only had one minor issue caused by the fact that I forgot to change the threshold from the default "on average" to "at least once". As a result, I spent nearly an hour waiting for an alert e-mail only to realize that I would never get one due to my mistake since the average would never be high enough to reach the alert threshold.

DATADOG

Events

Dashboards

Infrastructure

Monitors

Metrics

Integrations

APM

Notebooks

Logs

a   Get my_metric   from host:michaels-air.home ✕   excluding (none)   avg by (everything) ✚   </>

Advanced...

Simple Alert ▾   Trigger a single alert when your metric satisfies your alert conditions.   ❓

3 Set alert conditions

Trigger when the metric is above ▾ the threshold on average ▾ during the last 5 minutes ▾

Alert threshold: 800   (0.8K)

Warning threshold: 500   (0.5K)

Alert recovery threshold: Alert recovery threshold (o

Warning recovery threshold: Warning recovery threshol

Require ▾ a full window of data for evaluation.   ❓

Note: We highly recommend you select "Do Not Require" for sparse metrics, otherwise some evaluations will be skipped.

Do not not... ▾ if data is missing.   ❓

Note: the missing data window must be at least 2x the evaluation period above to work

[Never] ▾ automatically resolve this event from a triggered state.   ❓

Delay evaluation by [   ]⇕ seconds   ❓

4 Say what's happening   Use message template variables ❓

👁 Preview   ✏ Edit   Markdown supported

The value is too high!

3 - Did you try turning it off and back on again?
4 - Call IT.
{{/is_alert}}
{{#is_warning}} The value might be too high.
No panic. It's not critical yet. Let's just keep an eye on it for now.
{{/is_warning}}
{{#is_no_data}} We've misplaced the data.
We'll be darned if we can't find it. Probably turn up in an hour or so.
{{/is_no_data}}
@contact@michaelshaughnessy.net

Tags: Select or add related tags   ❓

[Never] ▾ renotify if the monitor has not been resolved.   ❓

5 Notify your team

Michael Shaughnessy ✕

Do not not... ▾ alert recipients when this alert is modified   ❓

Do not restrict ▾ editing this monitor to its creator or administrators   ❓

Illustration 14: The monitor creation form with custom messages for Alert, Warning and No Data cases. The thresholds are set to 800 for alert and 500 for warning and, to my chagrin, the threshold type in the second field of the line after "Set alert conditions" is set to take an average of the values over a 5 minute period instead of reporting upon each occurrence of an inappropriate value.

Help

Team

Set alert conditions

Trigger when the metric is [above ▾] the threshold [at least once ▾] during the last [5 minutes ▾]

Alert threshold: [800]

Warning threshold: [500]

Alert recovery threshold: [ ]

Warning recovery threshold: [ ]

Illustration 15: *That's better! Now I'll be notified as soon as any value exceeds one of my two thresholds.*

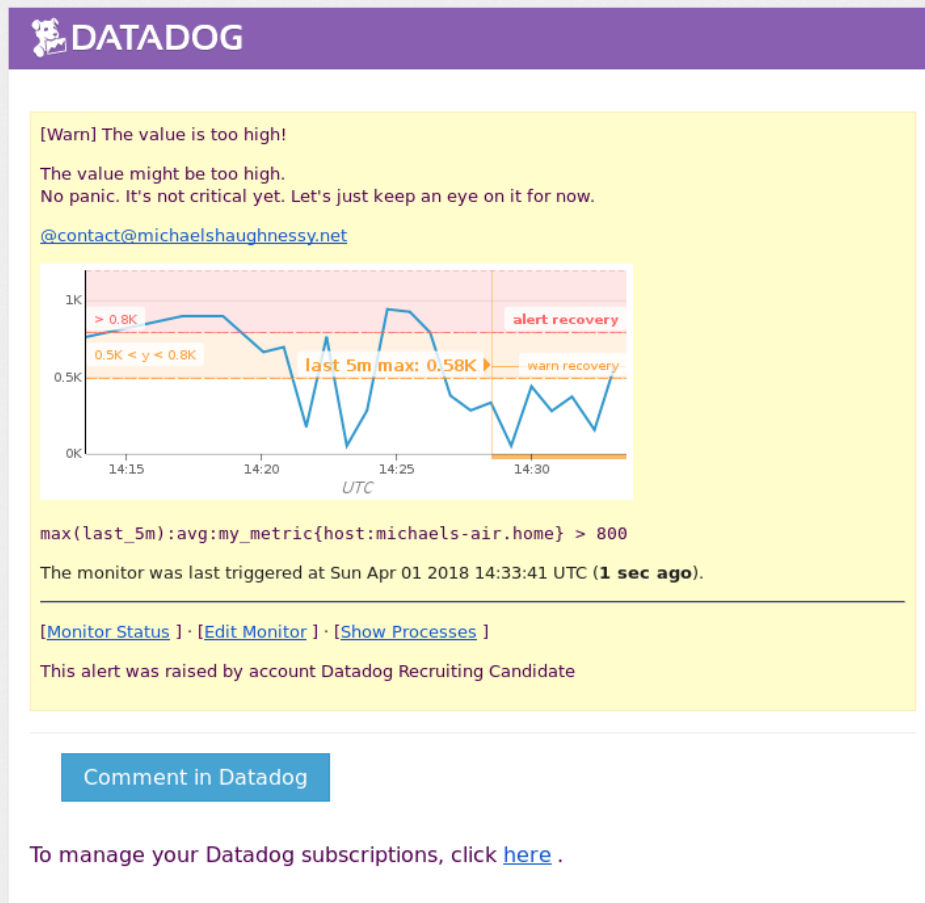Datadog Alerting alert@datadoghq.com via outbound.dtdg.co   4:33 PM (9 minutes ago) ☆ ↩
to me ▾

DATADOG

[Warn] The value is too high!

The value might be too high.
No panic. It's not critical yet. Let's just keep an eye on it for now.

@contact@michaelshaughnessy.net
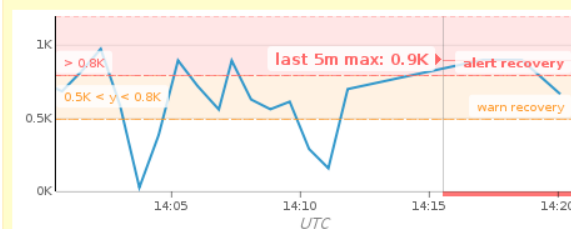
```
max(last_5m):avg:my_metric{host:michaels-air.home} > 800
```

The monitor was last triggered at Sun Apr 01 2018 14:33:41 UTC (**1 sec ago**).

[Monitor Status] · [Edit Monitor] · [Show Processes]

This alert was raised by account Datadog Recruiting Candidate

[Comment in Datadog]

To manage your Datadog subscriptions, click here .

Illustration 16: *The message I received in my Gmail inbox when a value went over the warning threshold.*

Illustration 17: *The message I received in my inbox after a value exceeded the alert threshold. This message provides the actual value and suggests some steps the user can take in order to correct the problem.*

This was by far the hardest step. I ran into several blocking bugs and errors that caused me to scratch my head for a while. Most notably, port usage turned out to be a problem. Both the Datadog agent and the APM agent wanted to use port 5000 on localhost so I had to specify a new port for the APM agent (I chose 5050). That got things working so that I could run both agents at the same time but, unfortunately, that wasn't where my troubles ended. Once I launched my Flask app using the APM agent (ddtrace) I started seeing "connection refused" errors on port 8126. According to Datadog's documentation, this is the port that the agent uses to listen to APM requests so I deduced that the problem was either with ddtrace's permissions or Datadog's configurations. Looks like it's time to go digging through the docs!

After a thorough excavation of Datadog's documentation and a good deal of playing around, I finally stumbled upon a solution. Indeed, according to the documentation here (https://docs.datadoghq.com/tracing/setup/), the APM agent needs to be enabled in the datadog.yaml file. I had already done this but I got the syntax wrong. The documentation had led me to believe that the proper syntax was as follows:

```
apm_config:
        enabled: true
```

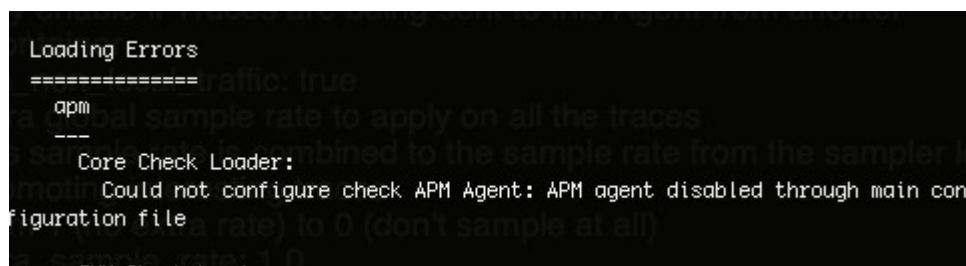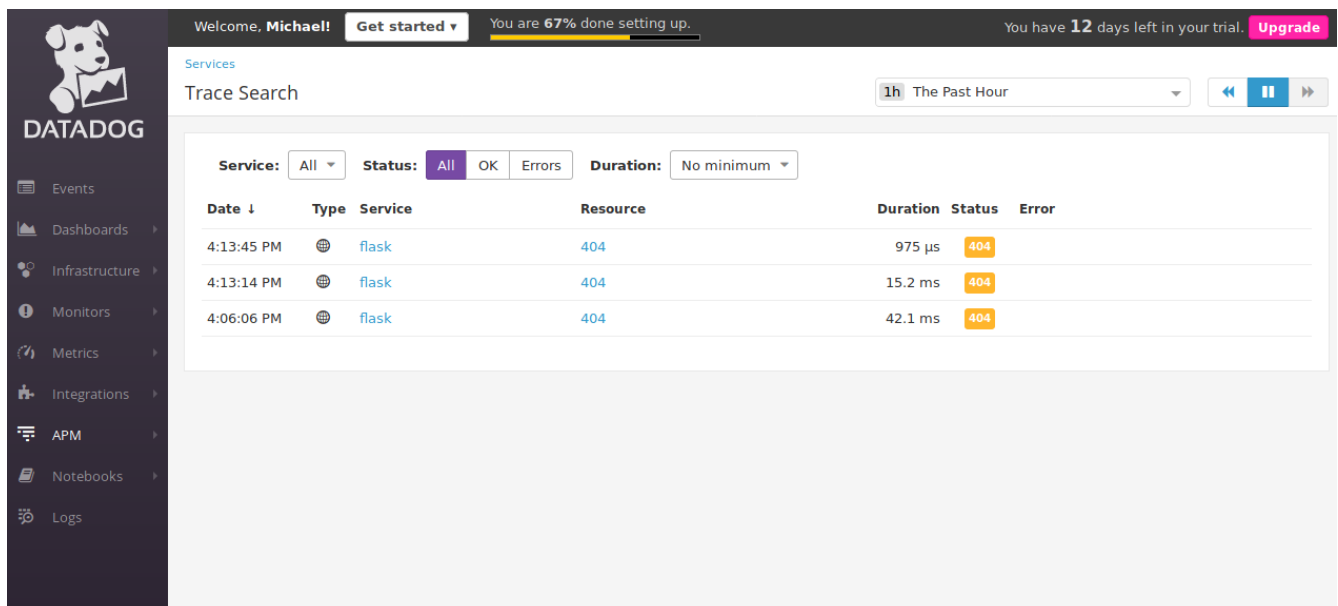However, this syntax caused a load error for APM.

Illustration 18: APM wouldn't load using the options as they appear in the default, example config file.

Once I realized this, I tried a different syntax (the one specified under the "File setting" column in the documentation that I linked to above). The result? It works!

Illustration 19: *The working code from the datadog.yaml config file. The previous syntax is commented out on lines 446 and 447.*

Illustration 20: *So close! APM is reporting to the Datadog Agent but all my http requests are returning 404 errors.*

Unfortunately, over the course of my playing around with configurations, I managed to introduce an error into my Flask configuration so that I only received 404 errors every time I tried to connect to the app. Fortunately, the error was just a minor mistake of syntax in my Flask app and once I corrected it everything worked smoothly.

```python
1  from flask import Flask
2  import logging
3  import sys
4  import blinker as _
5
6  from ddtrace import tracer
7  from ddtrace.contrib.flask import TraceMiddleware
8
9  tracer.configure(hostname='localhost', port=8126)
10
11 # Have flask use stdout as the logger
12 main_logger = logging.getLogger()
13 main_logger.setLevel(logging.DEBUG)
14 c = logging.StreamHandler(sys.stdout)
15 formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
16 c.setFormatter(formatter)
17 main_logger.addHandler(c)
18
19 app = Flask(__name__)
20 app.config['DEBUG'] = False
21
22 @app.route('/')
23 def api_entry():
24     return 'Entrypoint to the Application'
25
26 @app.route('/api/apm')
27 def apm_endpoint():
28     return 'Getting APM Started'
29
30 @app.route('/api/trace')
31 def trace_endpoint():
32     return 'Posting Traces'
33
34 if __name__ == '__main__':
35 app.run(host = 'localhost', port=5050)
```

Illustration 21: *The final code for my Flask application. At some point during my configurations I added a second call to app.run() on line 21 which caused the @app.routes below it to return 404 errors whenever I queried them. It was an obvious mistake that I overlooked for an embarrassingly long time but, in the end, I fixed it and managed to conclude my Datadog adventure with a happy ending.*
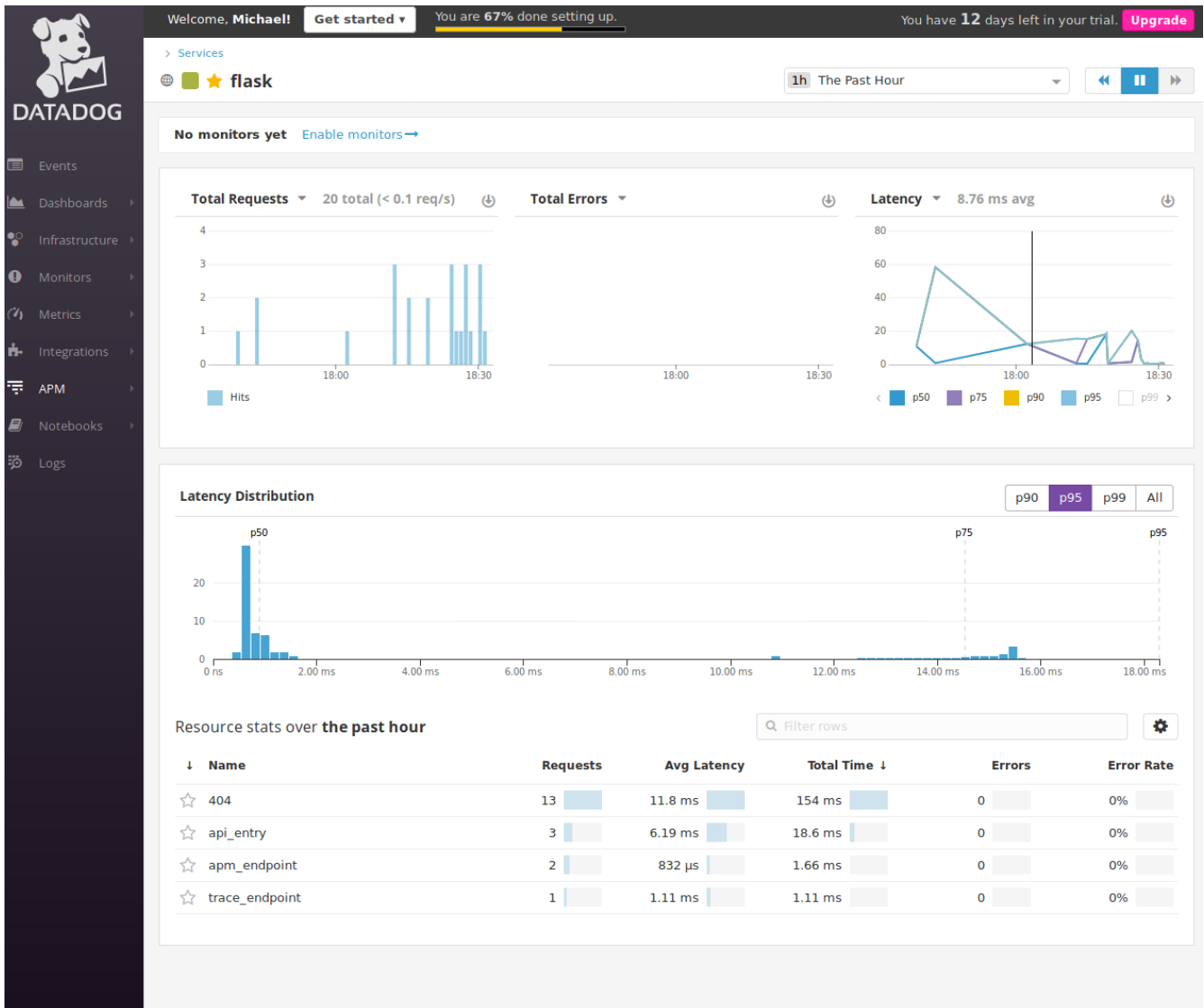
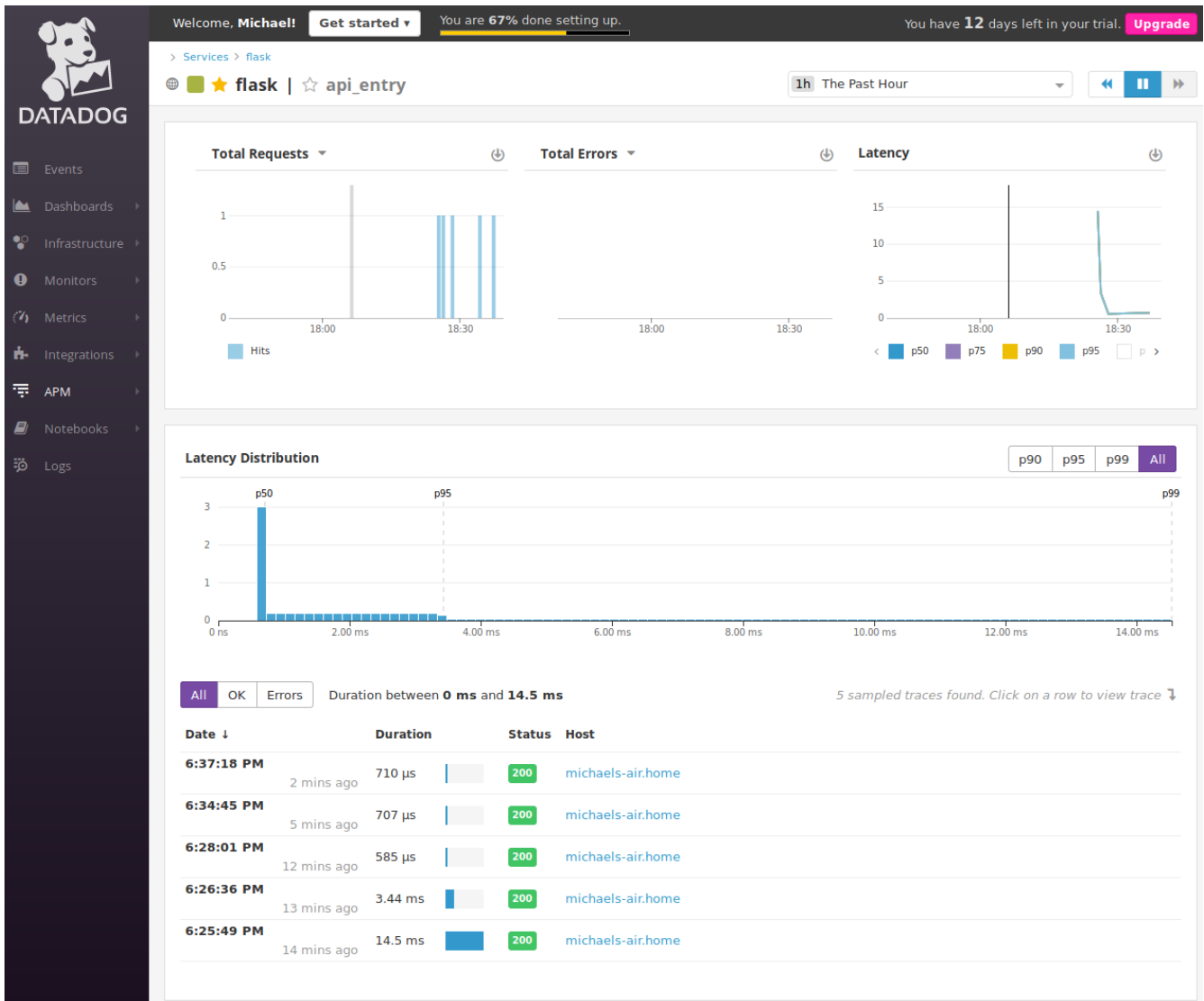**Illustration 22**: *Hurray! We have successful queries and data.*

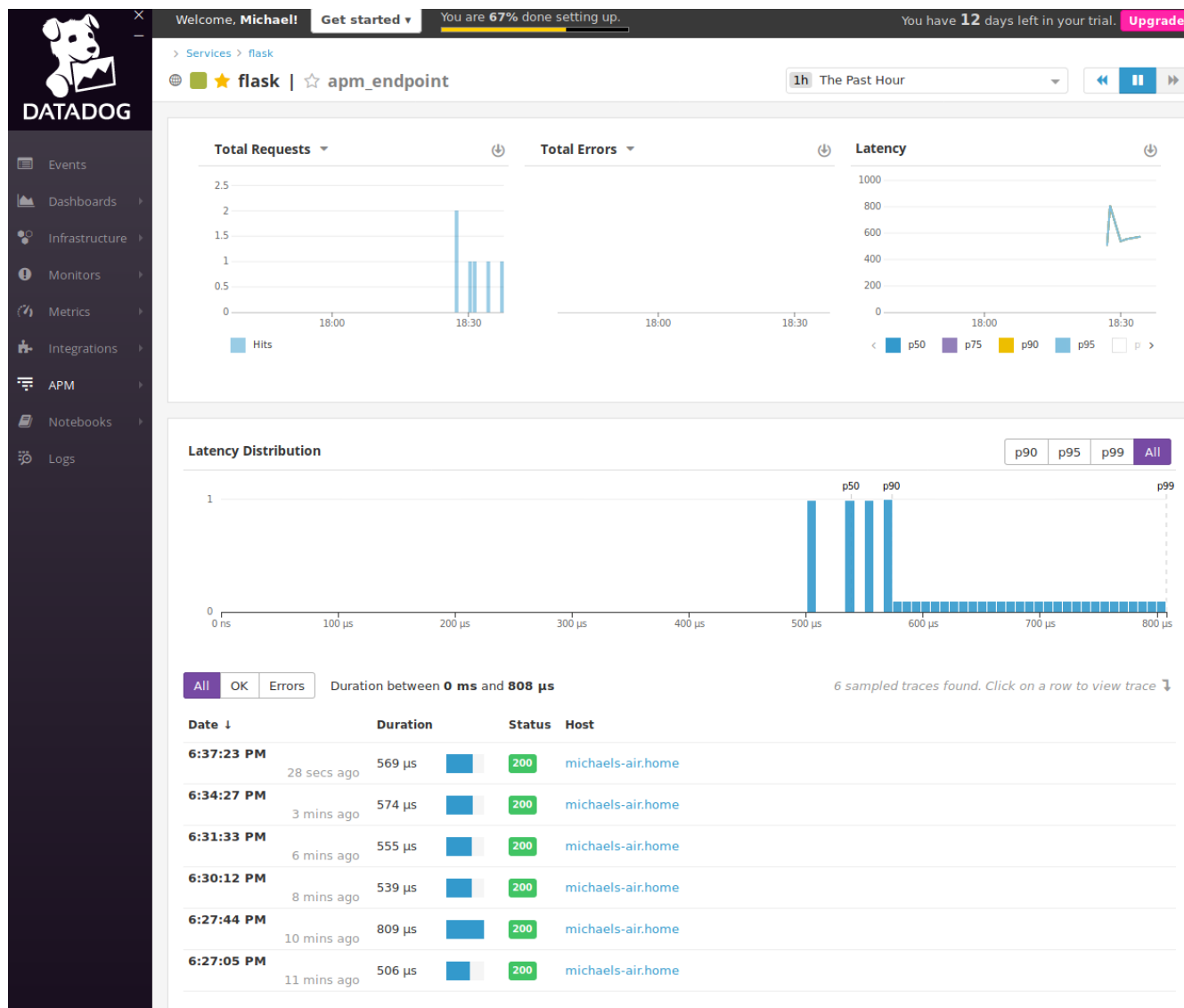***Illustration 23****: This page displays a history of the queries I made to localhost:5050 .*

***Illustration 24**: This page displays a history of the queries I made to localhost:5050/api/apm .*
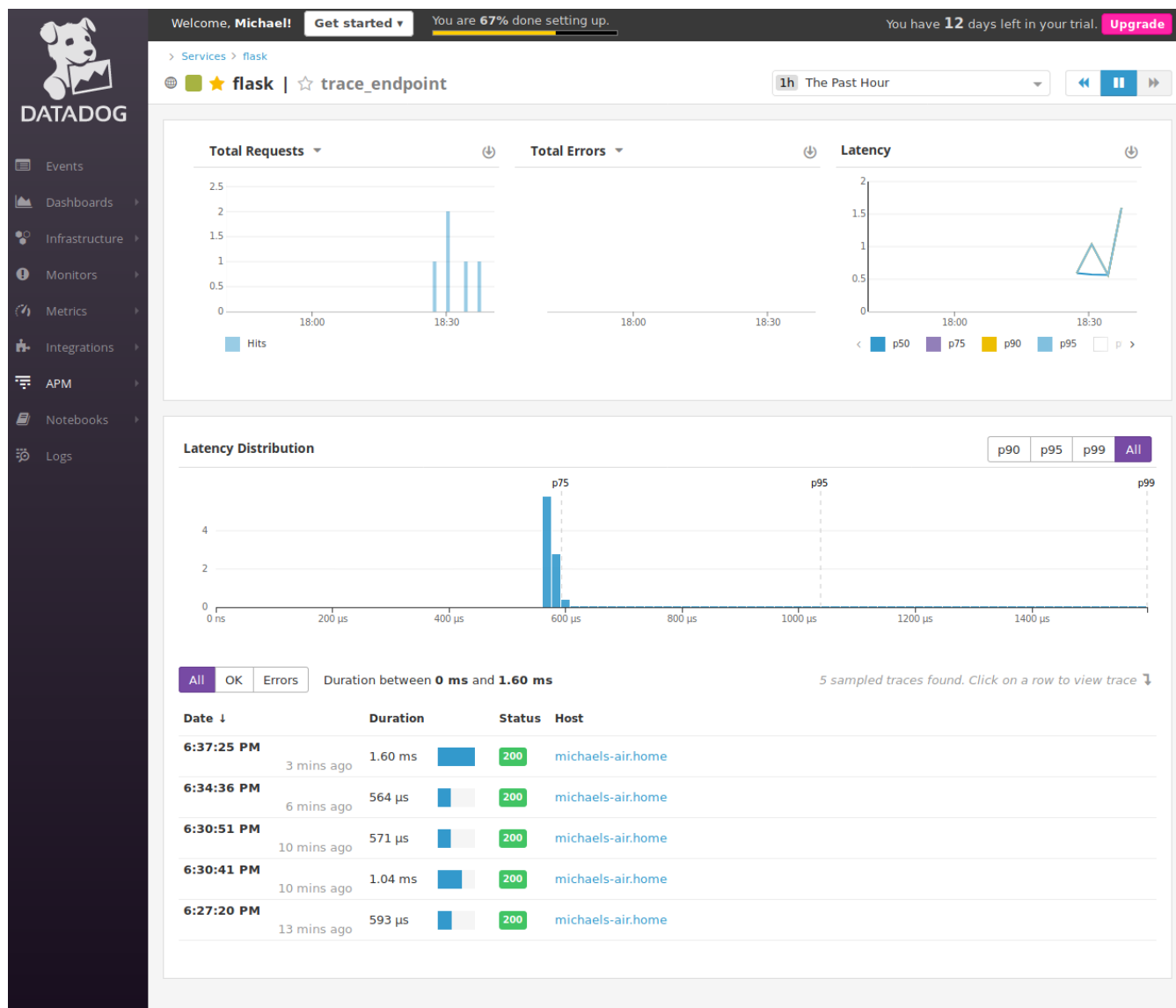
Illustration 25*: This page displays a history of the queries I made to localhost:5050/api/trace .*

That concludes my Datadog adventure. I enjoyed having the chance to explore all the possibilities that Datadog's software offers and I'm eager to learn more. Thank you for reading this far and allowing me to share my experience with you.