

Datadog Recruitment Candidate Exercise

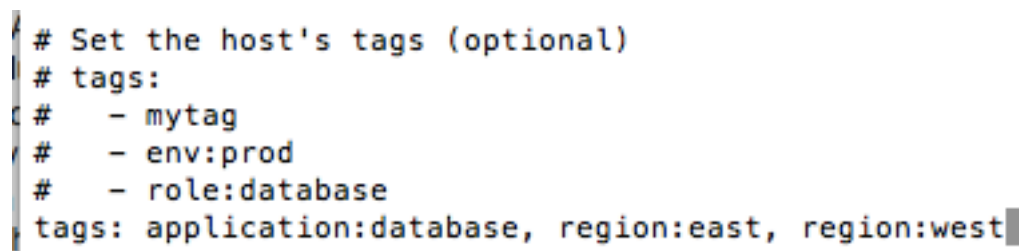
Cathy Swanton
05/09/2018

1. Collecting Metrics

1.1. Adding Tags

To add tags using the Agent config file, I followed the instructions in the following documentation: https://docs.datadoghq.com/tagging/assigning_tags/

I edited the config file '/opt/datadog-agent/etc/datadog.yaml' to include some sample tags, as shown in the image below.

A screenshot of a terminal window showing the configuration of the datadog.yaml file. The text is as follows:

```
# Set the host's tags (optional)
# tags:
#   - mytag
#   - env:prod
#   - role:database
tags: application:database, region:east, region:west
```

Figure 1-0-1. Adding tags in the Agent config file.

I then restarted the Agent using the following commands:

```
>> launchctl stop com.datadoghq.agent
>> launchctl start com.datadoghq.agent
```

To confirm that these tags had been successfully added, I went to the Host Map in the UI. See image below.

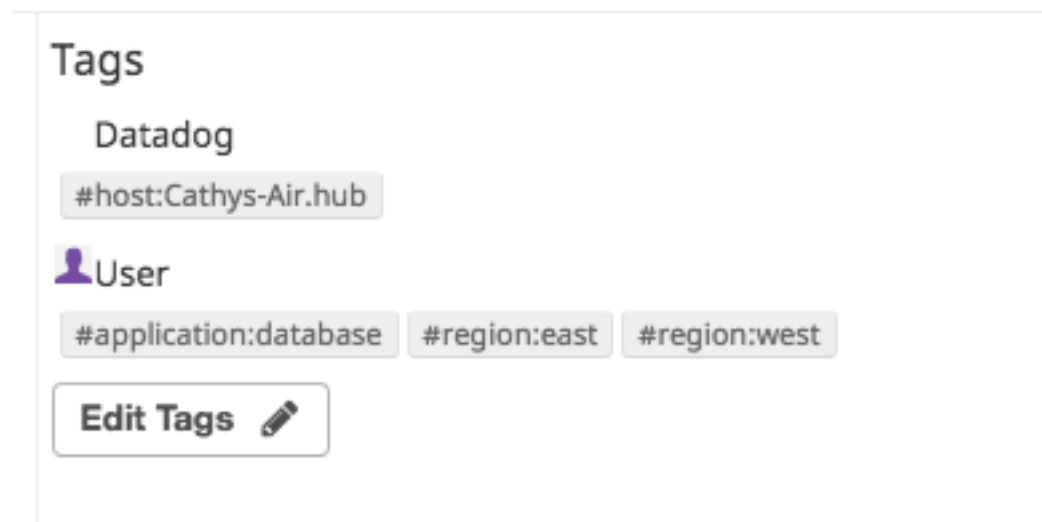


Figure 1-0-2. Tags visible in Host Map.

1.2. Installing Datadog Integration

To install a Datadog Integration, I first downloaded MySQL and then searched for MySQL in Datadog.

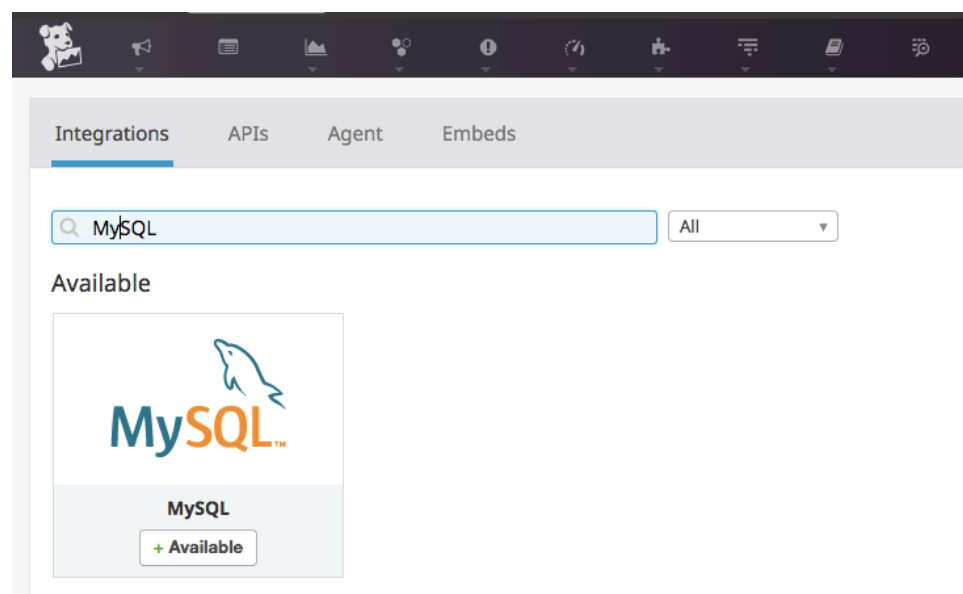


Figure 1-0-3. Searching for MySQL in Datadog.

I clicked on the link and installed it. The image below shows that it has been successfully installed.

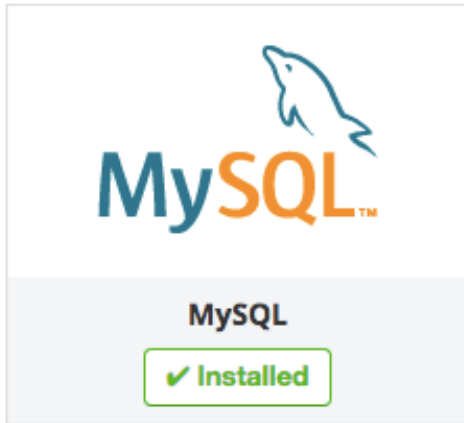


Figure 1-0-4. MySQL integration installed.

1.3. Creating a Custom Agent Check

To create a custom agent check, I followed the 'Hello World' example described on the website (https://docs.datadoghq.com/developers/agent_checks/). This example created a hello.world metric with a value of 1. After seeing how this was done, I created the 'my_metric' metric as follows:

- Create a file /opt/datadog-agent/etc/conf.d/mycheck.d/mycheck.yaml
This is essentially a dummy config file, with no real contents.

```
Cathys-Air:conf.d cathyswanton$ cat mycheck.yaml  
  
init_config:  
  
instances:  
  [{}]  
  
Cathys-Air:conf.d cathyswanton$
```

Figure 1-0-5. Yaml config file.

- Create a file /opt/datadog-agent/etc/checks.d/mycheck.py
This python file will create a random variable between 0 and 1000 and add it to the metric 'my_metric'

```
from checks import AgentCheck  
from random import randint  
class MyCheck(AgentCheck):  
    def check(self, instance):  
        self.gauge('my_metric', randint(0,1000))  
~
```

Figure 1-0-6. Python file.

- Restart the agent

```
>> launchctl stop com.datadoghq.agent
>> launchctl start com.datadoghq.agent
```

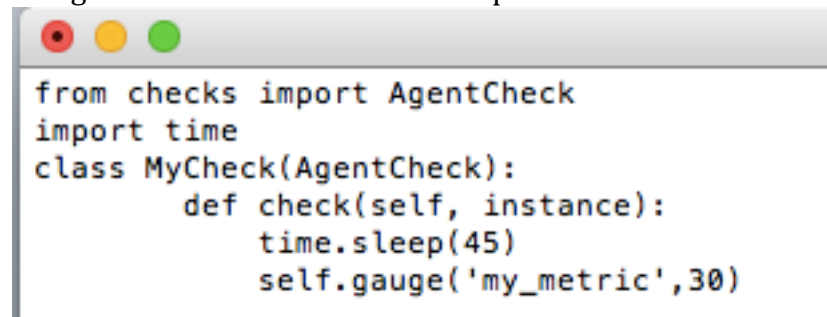
This did not work as expected. When I went to the Datadog UI, I could not see my new metric 'my_metric'. I could however see the 'hello.world' metric which was just being set to 1. The only difference between the two is that 'my_metric' is using randint to generate. I removed this from the code and just set an arbitrary value of 30. Still, I could not see the metric in the UI. Next, I tried removing the import line from the python code. This time I could see my new metric, with the assigned value of 30.

Since the 'import random' was causing some sort of problem, I tried to use the randint source code instead. This didn't work either. At this point, I decided to move on with the rest of the exercise. I decided that it wasn't worth wasting too much time here. There is probably some environment issues here that aren't allowing me to import random (was able to import re and other modules in the code). For the rest of this exercise, I was able to manually change the value of my_metric to test different scenarios, so it did not prevent me from proceeding.

1.4. Changing the Collection Interval

1.4.1. Using the Python check file

To add a 45 second collection interval, you can add a delay in the Python code using the time module. See code implemented below.

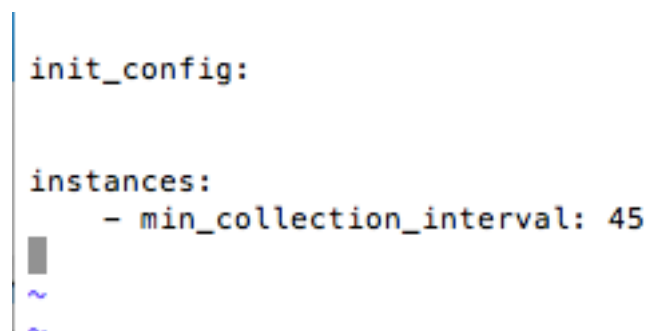


```
from checks import AgentCheck
import time
class MyCheck(AgentCheck):
    def check(self, instance):
        time.sleep(45)
        self.gauge('my_metric', 30)
```

Figure 1-0-7. Changing the collection interval using Python code.

1.4.2. Using the .yaml config file (Bonus Question)

A nicer way of changing the collection interval is to use the .yaml config file. Add a 45 second collection interval as follows:



```
init_config:

instances:
  - min_collection_interval: 45
~
~
```

Figure 1-0-8. Changing the collection interval using the config file.

To validate that this was successfully working, I plotted 'my_metric' and could see that the points were being added to the graph in 45 second intervals. The first image below shows a data point at 16:37:00 and the next is 45 seconds later at 16:37:45.

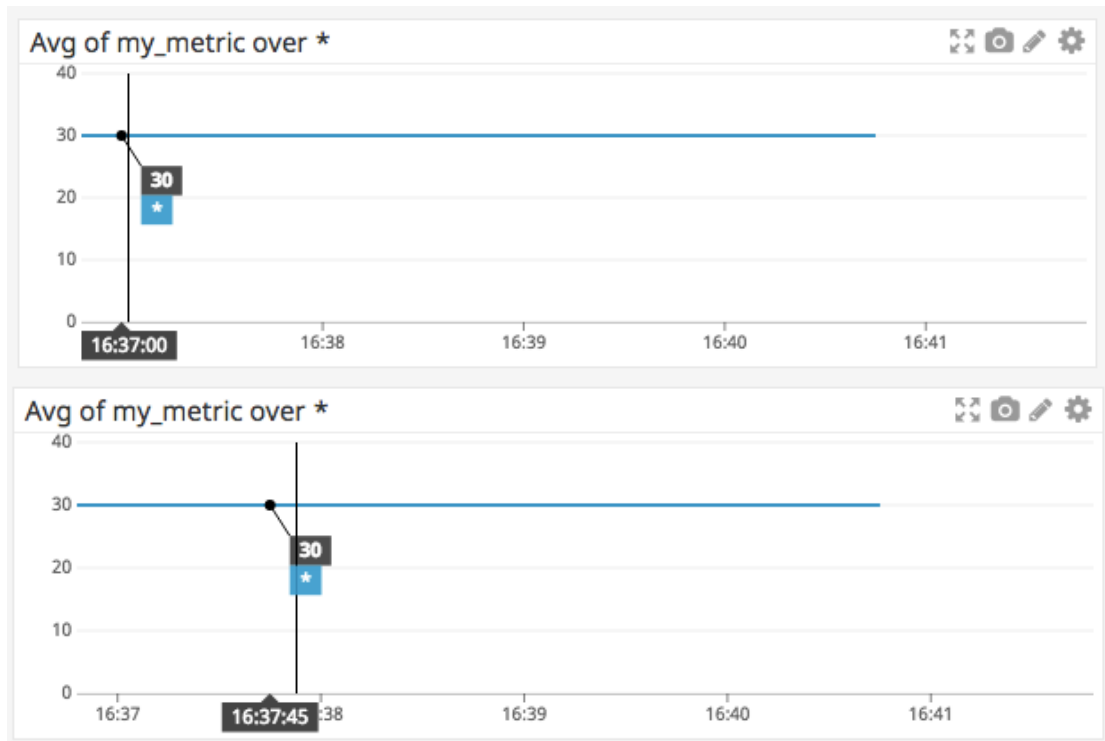


Figure 1-0-9. 45 second collection interval.

2. Visualizing Data

I created a new Timeboard called MyMetric.

2.1. Custom Metric Timeseries

In the new dashboard, I added a Timeseries graph for the average of my custom metric, 'my_metric'. Since I only have one host, there was no need to specify which host.

1 Select your visualization

Timeseries Query Value Heat Map Distribution Top List Change Host Map

2 Graph your data

Graph Primer Share JSON Edit

Metric from avg by as... </>

Display: Lines Color: Classic Style: Solid Stroke: Normal

Advanced...

Graph Metric

- > Event Overlays
- > Markers
- > Y-Axis Controls

3 Give your graph a title (or leave blank for suggested title)

Average of My_Metric

Cancel Preview Save

Figure 2-1. Creating a timeseries for 'my_metric'.

2.2. Using the Anomaly Function

To create an Anomaly, I used the variable 'system.cpu.idle'. The JSON code for the graph is given below. It is plotting an anomaly for system.cpu.idle using the agile method with a bounds value of 2.

```
{
  "viz": "timeseries",
  "status": "done",
  "requests": [
    {
      "q": "anomalies(avg:system.cpu.idle{*}, 'agile', 2, direction='both',
alert_window='last_15m', interval=60, count_default_zero='true',
seasonality='hourly')",
      "type": "line",
      "style": {
        "palette": "dog_classic",
        "type": "solid",
        "width": "normal"
      },
      "conditional_formats": [],
      "aggregator": "avg"
    }
  ],
  "autoscale": true
}
```

2.3. Using the Rollup Function

I added a new timeseries graph to the dashboard for my custom metric. I then changed the JSON code to use the rollup function. The time is in seconds, so to sum all the points in the last hour will require a value of 3600.

```
{
  "viz": "timeseries",
  "status": "done",
  "requests": [
    {
      "q": "avg:my_metric(*).rollup(sum, 3600)",
      "type": "line",
      "style": {
        "palette": "dog_classic",
        "type": "solid",
        "width": "normal"
      },
      "conditional_formats": [],
      "aggregator": "avg"
    }
  ],
  "autoscale": true
}
```

2.4. Setting the Timeframe to 5 Minutes.

To change the timeframe to 5 minutes, I used the mouse to zoom in on the graph.

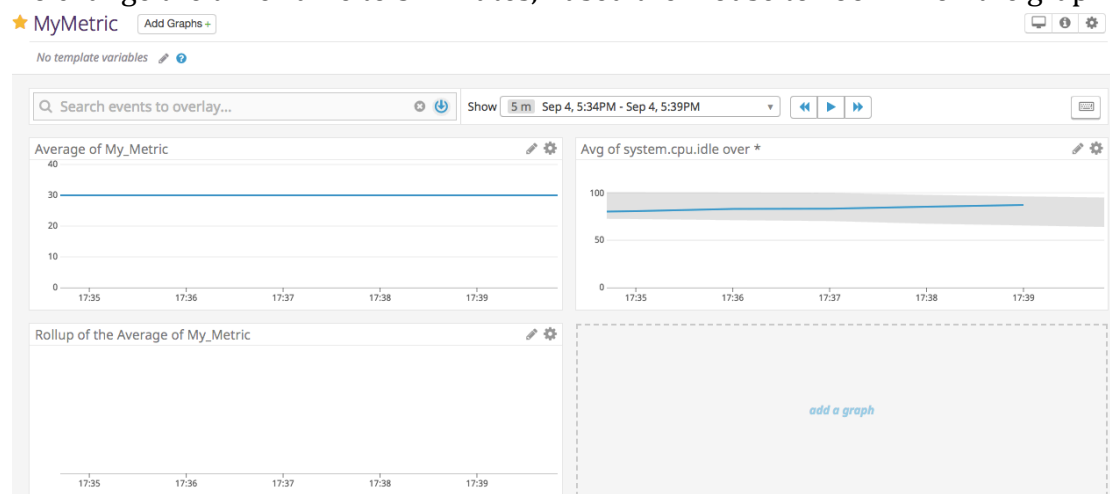


Figure 2-2. Dashboard with 5minute timeframe.

2.5. Take a Snapshot of the Graph.

To take a snapshot of the graph I clicked on the camera icon and used the @ notation to send it to myself. Below is a sample of the email that will be received.

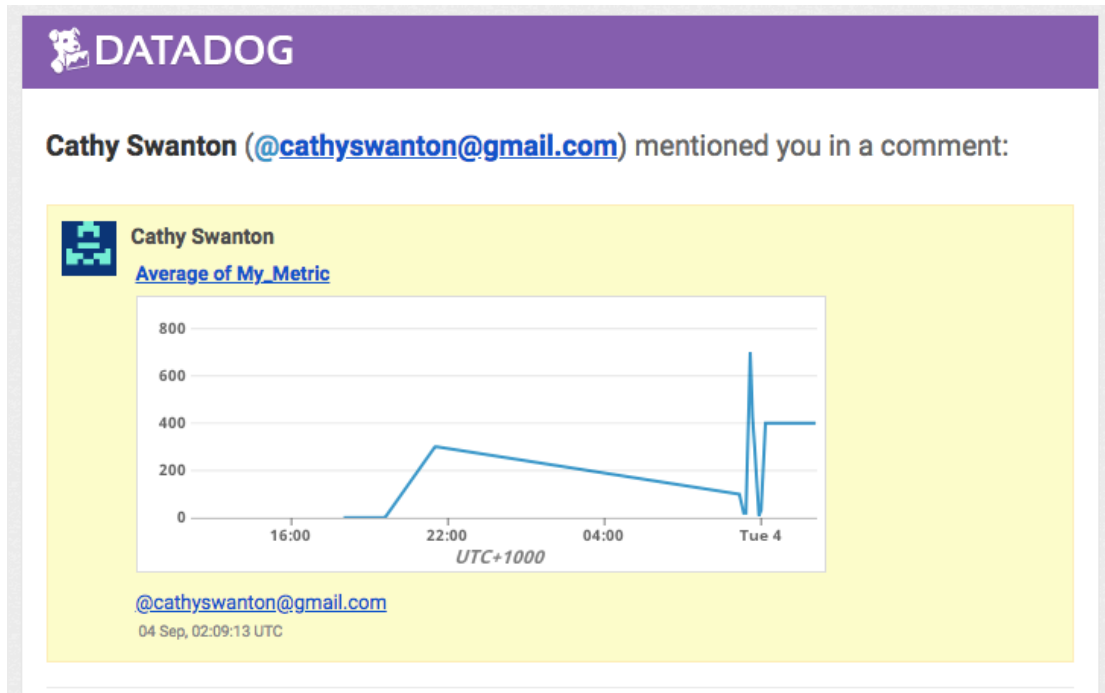


Figure 2-3. Email received from snapshot.

2.6. What is the Anomaly graph displaying?

The anomaly graph displays any 'abnormalities' in the metric `cpu.idle`. Unfortunately, since I only have 1 day of data available, the anomalies are pretty meaningless. Ideally, I would have a couple of weeks worth of data and I would expect to see more normal patterns and therefore the algorithm would better detect anomalies.

However, working with the limited data available, below is a plot of the `cpu.idle` metric over the last day. The grey region shows the bounds, or the range of values that the algorithm would regard as 'normal'. Whenever the actual value is greater or less than these bounds, the trace is red.

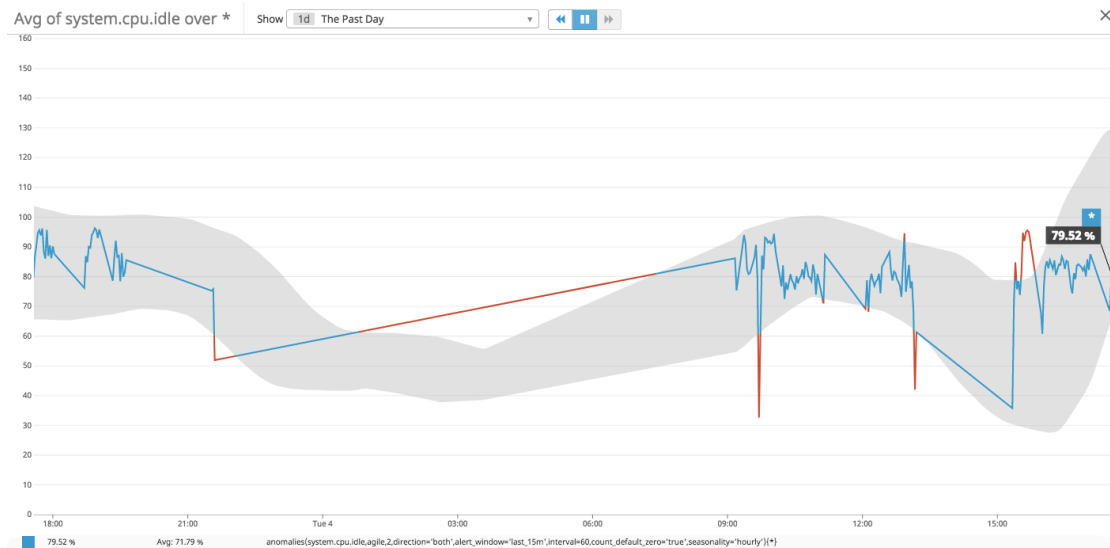


Figure 2-4. Anomaly Graph.

3. Monitoring Data.

3.1. Creating a Metric Monitor

The screenshots below capture the process of creating a metric monitor and configuring the alert messages. The alert conditions send an alert message when the value exceeds 800 and a warning message when it exceeds 500 (red and yellow boxes, respectively, in step 3 in the image below). I have also enabled the option to notify if the data is missing for more than 10 minutes.

2 Define the metric

a Metric **my_metric** from (everywhere) excluding (none) max by (everything) +

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**

Warning threshold: **500**

Alert recovery threshold:

Warning recovery threshold:

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.

if data is missing for more than minutes.

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

automatically resolve this event from a no data state.

Delay evaluation by seconds

Figure 3-1. Alert conditions for metric monitor.

The figure below shows how to configure the monitor alerts. Different messages are sent for the three scenarios – alert, warning and no_data.

4 Say what's happening

Monitor triggered for my_metric

```

{{#is_alert}}
  Alert: my_metric has exceeded 800. Offending host:{{host.name}}
{{/is_alert}}

{{#is_warning}}
  Warning: my_metric has exceeded 500
{{/is_warning}}

{{#is_no_data}}
  No Data: no data has been received for my_metric for more than 10 minutes.
{{/is_no_data}}@cathyswanton@gmail.com

```

Tags:

renotify if the monitor has not been resolved.

5 Notify your team

alert recipients when this alert is modified

Figure 3-2. Configuring the notifications.

Since I could not get the random number generator to work, I just forced each of the scenarios below. I manually set `my_metric` in the Python code to different values so that I could see the warning and alert messages.

The figure below shows the email message received when the value exceeds 800. Unfortunately, the `host.name` variable is not being populated. Not too sure why...

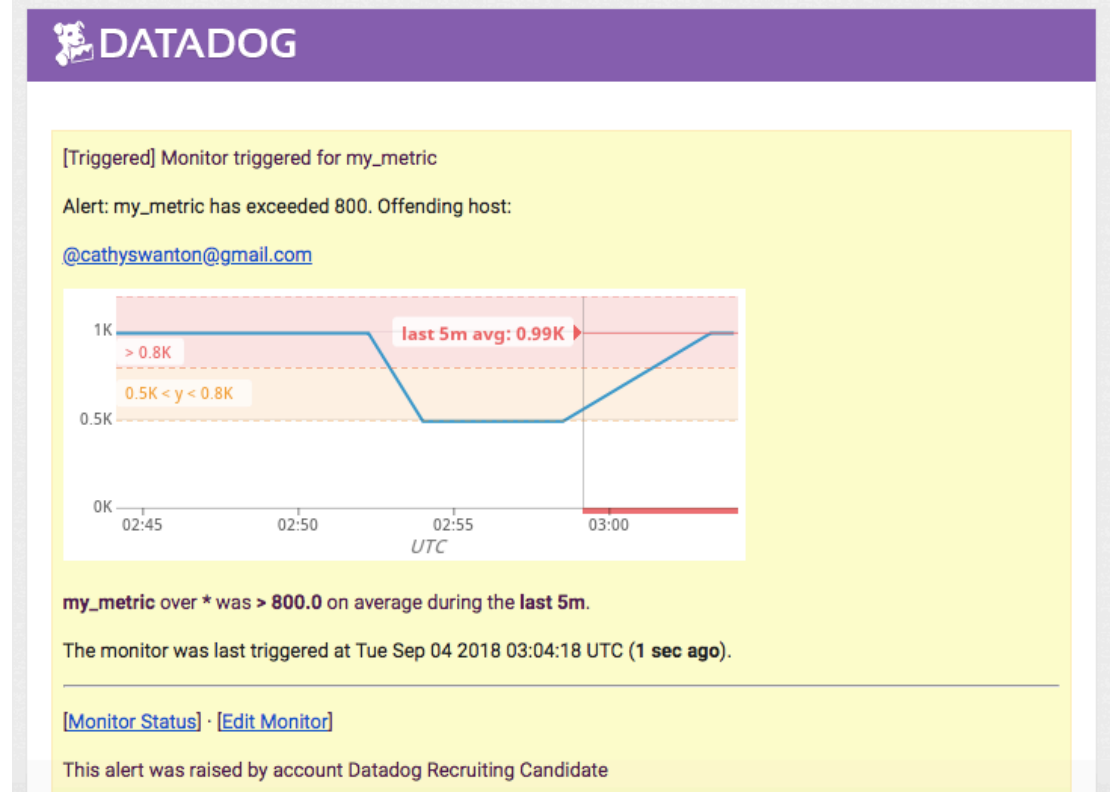


Figure 3-3. Sample Alert message.

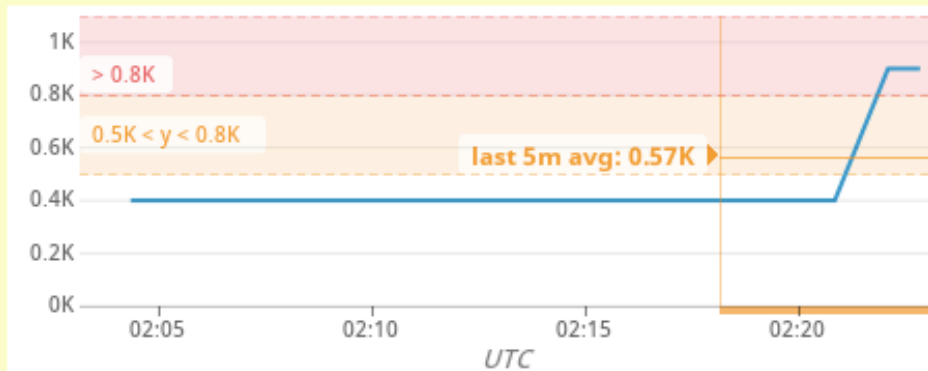
And a sample email for the warning scenario (value between 500 and 800):



[Warn] Monitor triggered for my_metric

Warning: my_metric has exceeded 500

@cathyswanton@gmail.com



my_metric over * was > 500.0 on average during the last 5m.

The monitor was last triggered at Tue Sep 04 2018 02:23:18 UTC (1 sec ago).

Figure 3-4. Sample Warning notification.

And finally, to validate the no_data case, I stopped the Agent for 10 minutes and got the following email.



[No data] Monitor triggered for my_metric

No Data: no data has been received for my_metric for more than 10 minutes.

@cathyswanton@gmail.com

The monitor has been missing data for the last 10m

The monitor was last triggered at Tue Sep 04 2018 02:36:18 UTC (1 sec ago).

[\[Monitor Status\]](#) · [\[Edit Monitor\]](#)

This alert was raised by account Datadog Recruiting Candidate

Figure 3-5. No_Data notification.

3.2. Scheduling Downtime

To schedule a downtime, I used the UI and went to Monitors->Manage Downtime. I scheduled two new downtimes. The first, shown below, silences the monitor for my_metric at 7PM for 14 hours (until 9AM) every day of the week. The image below shows the settings used.

1 Choose what to silence

☒ By monitor name ☐ By monitor tags

Monitor:

Group scope (optional, default all groups):

[Preview affected monitors](#)

2 Schedule

☐ One-Time ☒ Recurring

Start Date: Time Zone:

Repeat Every:

Beginning:

Duration:

Repeat Until:

Summary: **From 7:00pm to 9:00am tomorrow**
Daily until Dec 31, 2018

3 Add a message

☒ Preview ☐ Edit [Markdown supported](#)

Downtime every evening. @cathyswanton@gmail.com

Figure 3-6. Downtime every evening from 7PM to 9AM.

The second downtime I scheduled to start at one minute past midnight on Saturday morning and last for two days, to cover the entire weekend. This downtime will overlap with the evening one set previously, so the monitor will be ignored from 7PM on Friday evening until 9AM on Monday morning. These settings are shown in the image below.

1

Choose what to silence

By monitor nameBy monitor tags

Monitor:

Monitor: Monitor triggered for my_metric

Group scope (optional, default all groups):

* x

Preview affected monitors

?

2

Schedule

One-TimeRecurring

Start Date:2018/09/08Time Zone:Australia/Sydney

Repeat Every:7days

Beginning:00:01

Duration:2days

Repeat Until:Date2018/12/31

Summary:From 12:01am to 12:01am in 2 days
Every 7 days until Dec 31, 2018

Preview planned recurrences

3

Add a message

PreviewEdit

Markdown supported

Downtime every weekend.

Figure 3-7. Downtime scheduled for weekends.

The images below show the email messages received after scheduling the downtimes. Note that the time zone is UTC, so at first the times look off!

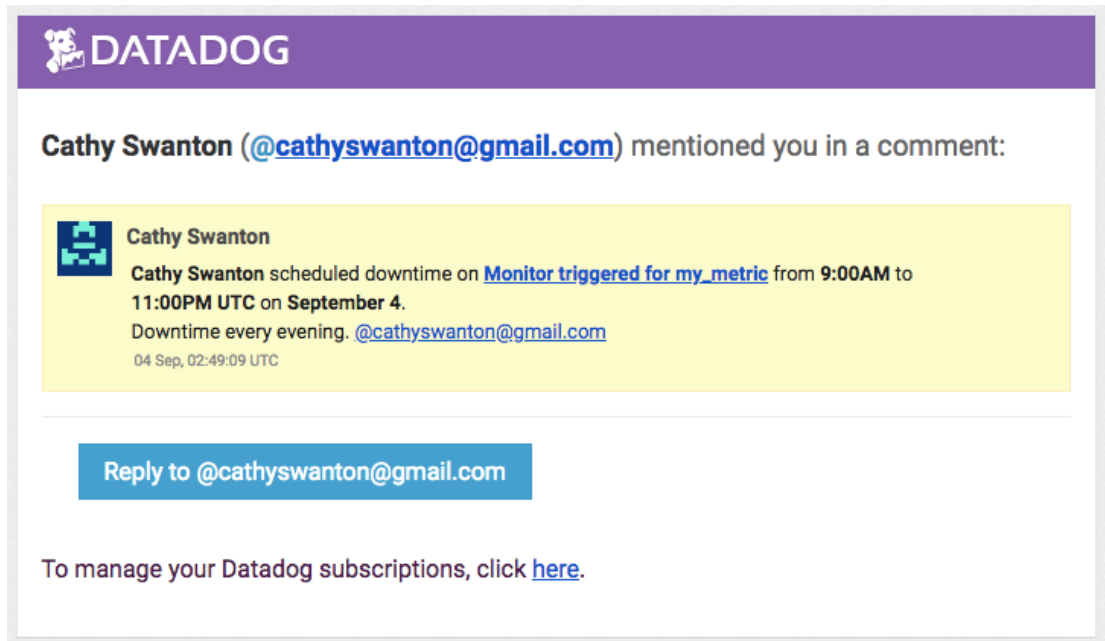


Figure 3-8. Email notification received for evening downtime.

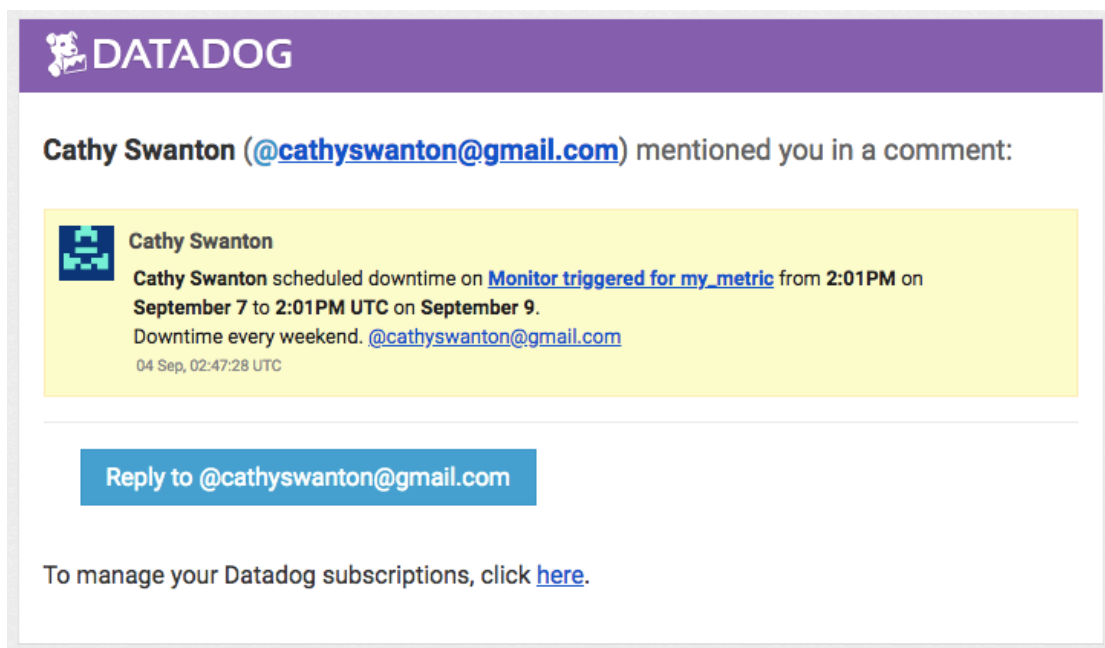


Figure 3-9. Email notification received for weekend downtime.

4. Collecting APM Data

I faced a few issues here. I saved the sample Flask application as `my_app.py` and installed `ddtrace` as instructed. However, when I ran the following command:

```
>> ddtrace-run python my_app.py
```

I got the following error messages:

```
INFO:werkzeug: * Running on http://0.0.0.0:5050/ (Press CTRL+C to quit)
2018-09-05 09:19:22,448 - werkzeug - INFO - * Running on http://0.0.0.0:5050/ (Press CTRL+C to quit)
ERROR:ddtrace.writer:cannot send services to localhost:8126: [Errno 61] Connection refused
2018-09-05 09:19:23,431 - ddtrace.writer - ERROR - cannot send services to localhost:8126: [Errno 61] Connection refused
INFO:werkzeug:127.0.0.1 - - [05/Sep/2018 09:19:34] "GET / HTTP/1.1" 200 -
2018-09-05 09:19:34,065 - werkzeug - INFO - 127.0.0.1 - - [05/Sep/2018 09:19:34] "GET / HTTP/1.1" 200 -
ERROR:ddtrace.writer:cannot send spans to localhost:8126: [Errno 61] Connection refused
2018-09-05 09:19:34,471 - ddtrace.writer - ERROR - cannot send spans to localhost:8126: [Errno 61] Connection refused
INFO:werkzeug:127.0.0.1 - - [05/Sep/2018 09:19:34] "GET /favicon.ico HTTP/1.1" 404 -
2018-09-05 09:19:34,514 - werkzeug - INFO - 127.0.0.1 - - [05/Sep/2018 09:19:34] "GET /favicon.ico HTTP/1.1" 404 -
ERROR:ddtrace.writer:cannot send spans to localhost:8126: [Errno 61] Connection refused
2018-09-05 09:19:35,475 - ddtrace.writer - ERROR - cannot send spans to localhost:8126: [Errno 61] Connection refused
```

Figure 4-1. Error message - connection refused.

I tried multiple fixes, including killing the port:8126, running without the datadog trace, and editing the datadog.yaml configuration file to uncomment the line highlighted below.

```
# apm_config:
# Whether or not the APM Agent should run
# enabled: true
# The environment tag that Traces should be tagged with
```

Figure 4-2. An attempted fix - changing the yaml config file.

None of these fixes worked however. I then stumbled upon these setup instructions: <https://docs.datadoghq.com/tracing/setup/> and found that for MacOS, the TraceAgent is not pre-packaged with the standard Agent, as it is for Windows and Linux. So I then downloaded and ran the trace agent.

(Learning from this: read the instructions first before diving straight in and you will save yourself considerable time!!)

```
Cathys-Air:Downloads cathyswanton$ chmod 770 trace-agent-darwin-amd64-6.4.1
Cathys-Air:Downloads cathyswanton$ ./trace-agent-darwin-amd64-6.4.1 --config /opt/datadog-agent/etc/datadog.yaml
2018-09-05 10:52:04 INFO (main.go:98) - Loaded configuration: /opt/datadog-agent/etc/datadog.yaml
2018-09-05 10:52:05 INFO (trace_writer.go:49) - Trace writer initializing with config: {MaxSpansPerPayload:1000 Flush
s SenderConfig:{MaxAge:20m0s MaxQueuedBytes:67108864 MaxQueuedPayloads:-1 ExponentialBackoff:{MaxDuration:2m0s Growth
2018-09-05 10:52:05 INFO (trace_writer.go:49) - Trace writer initialized with config: {MaxSpansPerPayload:1000 Flush
```

Figure 4-3. Installing trace agent.

After properly reading the instructions, I again ran the ddtrace command with my app. No more error messages this time. I sent some requests to the app, using the following curl commands:

```
Cathys-Air:~ cathyswanton$ curl 0.0.0.0:5050
Entrypoint to the ApplicationCathys-Air:~ cathyswanton$
Cathys-Air:~ cathyswanton$ curl 0.0.0.0:5050/api/apm
Getting APM StartedCathys-Air:~ cathyswanton$
Cathys-Air:~ cathyswanton$ curl 0.0.0.0:5050/api/trace
Posting TracesCathys-Air:~ cathyswanton$
Cathys-Air:~ cathyswanton$
```

Figure 4-4. Sending requests to the app.

Looking at the Datadog UI, I could now see some data (finally 😊).

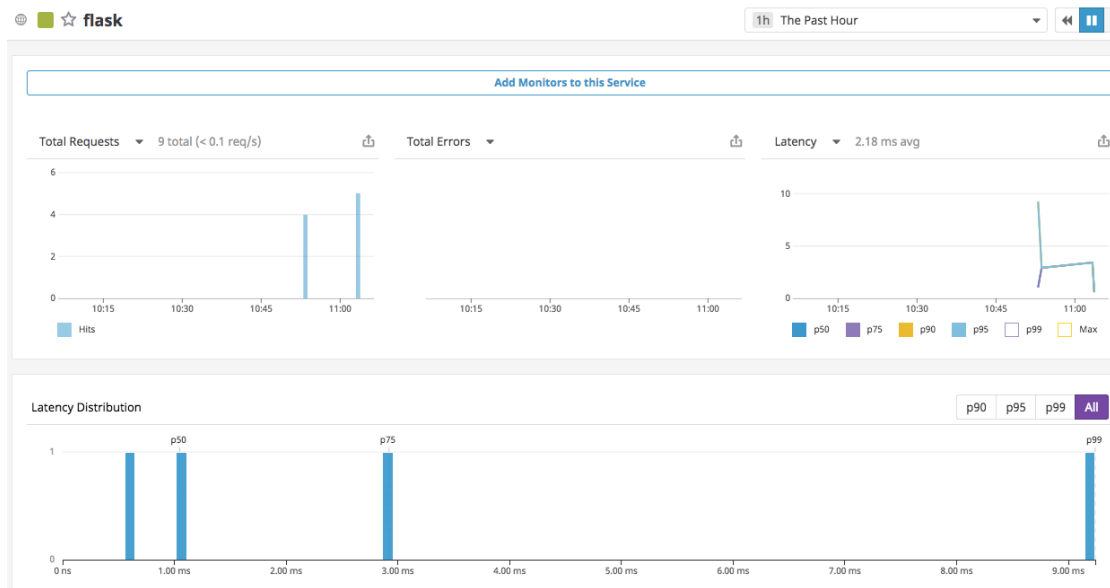


Figure 4-5. APM dashboard.

I then exported the latency graph to a new Dashboard:

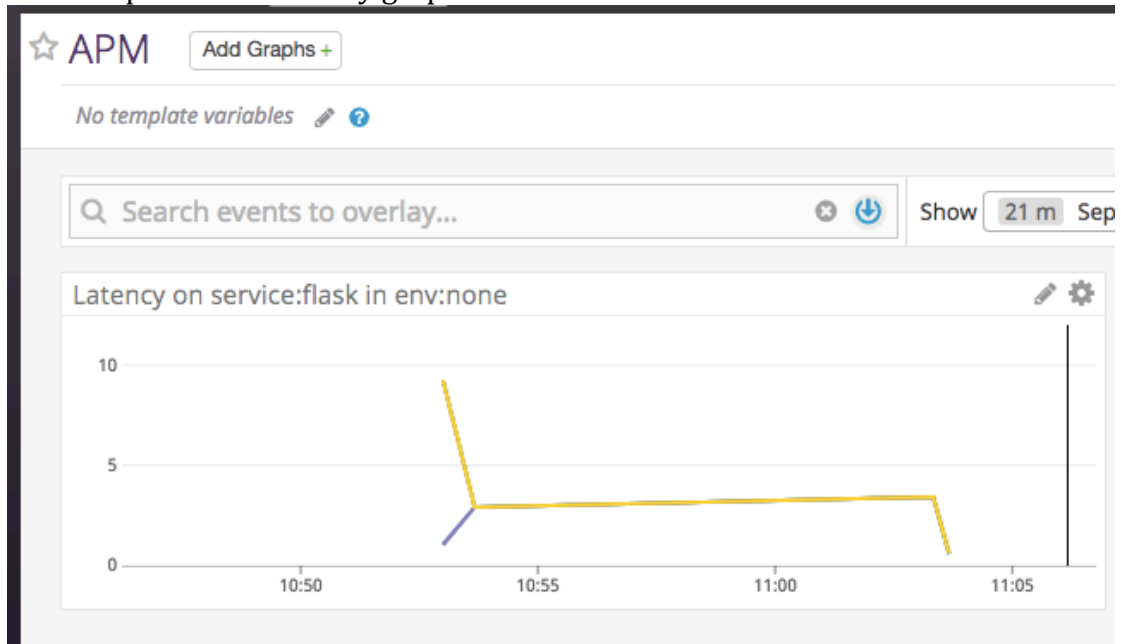


Figure 4-6. Latency graph exported to timeboard.

This is a link to the APM:

<https://app.datadoghq.com/apm/services?start=1536105716744&end=1536109316744&paused=false&env=none&watchdogOnly=false>

Difference between a Service and a Resource

A service is a set of processes that work together to provide a feature set. For example, a web application might consist of two services – a webapp service and a database service. These two services provide different function for the same feature (web application).

A resource is a particular query to a service. Using the web application example again, a resource might be a canonical URL like /user/home.

Final Question

I would use Datadog to monitor how busy my gym is. A pet peeve of mine is going to the gym and finding it packed. Having to queue up to use a machine irritates me and it means it takes me longer to do my routine. To prevent this annoyance, I would use Datadog to monitor how many people are coming and going, so before I leave home I can decide whether or not it is worth my while.

I would setup a monitor to alert me when the metric is below a certain threshold, notifying me that it is a good time to go. I would schedule this monitor to only alert me during times when I would be interested in going – so say I would set a downtime while I am at work, and only get alerts in the evening.