

Hello Datadog engineers!

My name is Derek Bean and I'm going to walk through the different steps for the GitHub hiring exercise. I'm using a Ubuntu 18.04 VM on my mac for this testing with PostgreSQL for the database engine. After signing up on your site, I've installed the agent via the instructions given for my OS:

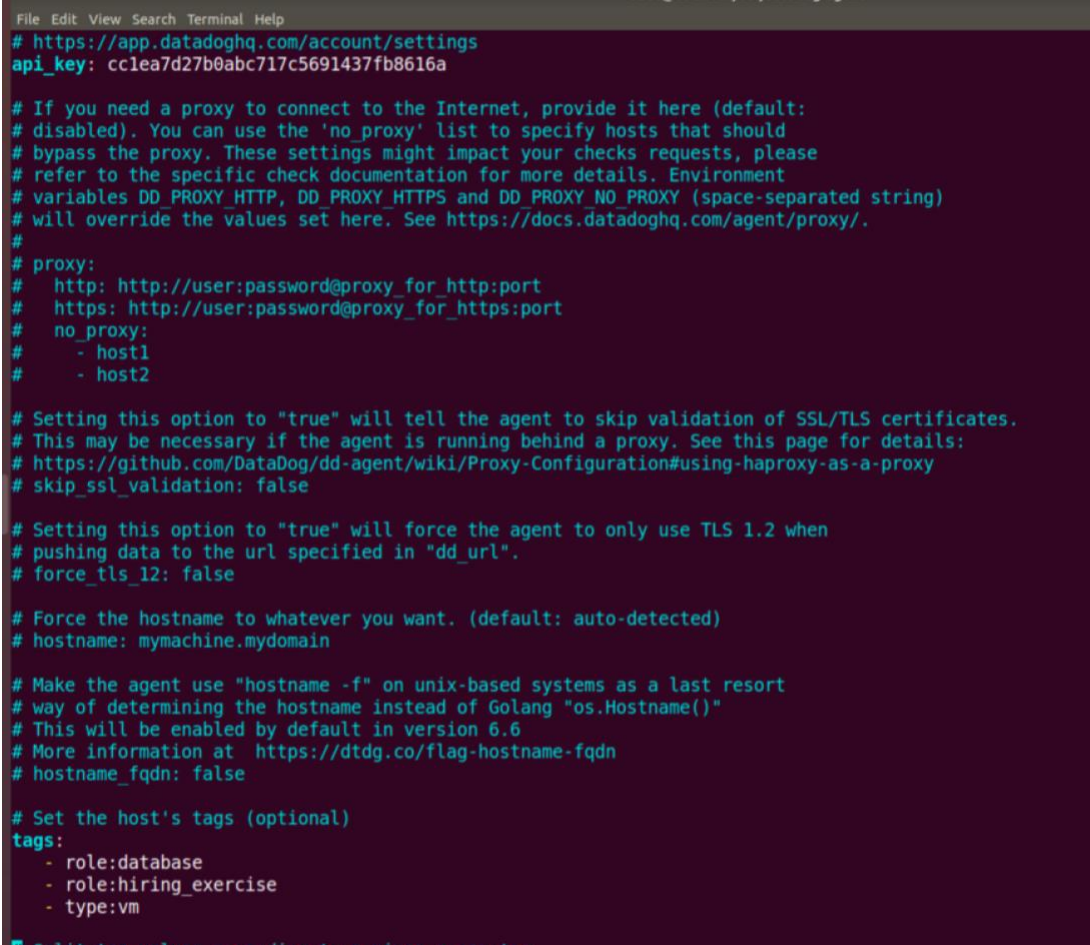
Ubuntu Agent Install:

```
DD_API_KEY=cc1ea7d27b0abc717c5691437fb8616a bash -c "$(curl -L
https://raw.githubusercontent.com/DataDog/datadog-agent/master/cmd/agent/install\_script.sh)"
```

COLLECTING METRICS:

Q: You're looking to add some tags inside the agent config file and show a screenshot of the host and it's tags in the Host Map page:

A: In the file /etc/datadog-agent/datadog.yaml you'll see a section for tags. I've added in a few to define the role and type of this system. After adding them in, you'll need to restart the datadog-agent service, which has to be done anytime a change is made to the configuration.



```
File Edit View Search Terminal Help
# https://app.datadoghq.com/account/settings
api_key: cc1ea7d27b0abc717c5691437fb8616a

# If you need a proxy to connect to the Internet, provide it here (default:
# disabled). You can use the 'no_proxy' list to specify hosts that should
# bypass the proxy. These settings might impact your checks requests, please
# refer to the specific check documentation for more details. Environment
# variables DD_PROXY_HTTP, DD_PROXY_HTTPS and DD_PROXY_NO_PROXY (space-separated string)
# will override the values set here. See https://docs.datadoghq.com/agent/proxy/.
#
# proxy:
#   http: http://user:password@proxy_for_http:port
#   https: http://user:password@proxy_for_https:port
#   no_proxy:
#     - host1
#     - host2

# Setting this option to "true" will tell the agent to skip validation of SSL/TLS certificates.
# This may be necessary if the agent is running behind a proxy. See this page for details:
# https://github.com/DataDog/dd-agent/wiki/Proxy-Configuration#using-haproxy-as-a-proxy
# skip_ssl_validation: false

# Setting this option to "true" will force the agent to only use TLS 1.2 when
# pushing data to the url specified in "dd_url".
# force_tls_12: false

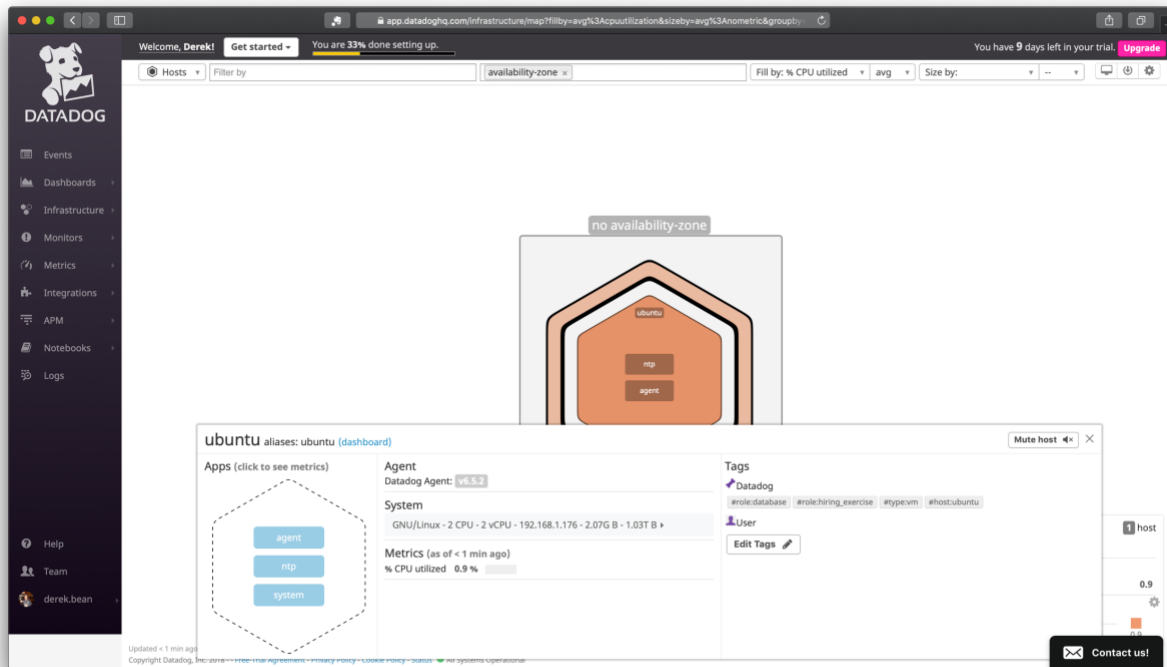
# Force the hostname to whatever you want. (default: auto-detected)
# hostname: mymachine.mydomain

# Make the agent use "hostname -f" on unix-based systems as a last resort
# way of determining the hostname instead of Golang "os.Hostname()"
# This will be enabled by default in version 6.6
# More information at https://dtdg.co/flag-hostname-fqdn
# hostname_fqdn: false

# Set the host's tags (optional)
tags:
  - role:database
  - role:hiring_exercise
  - type:vm
```



**\*\*After several minutes, the changes are now reflected inside the Host Map link online\*\***



Q: Now install a database on your host and install the respective Datadog integration:

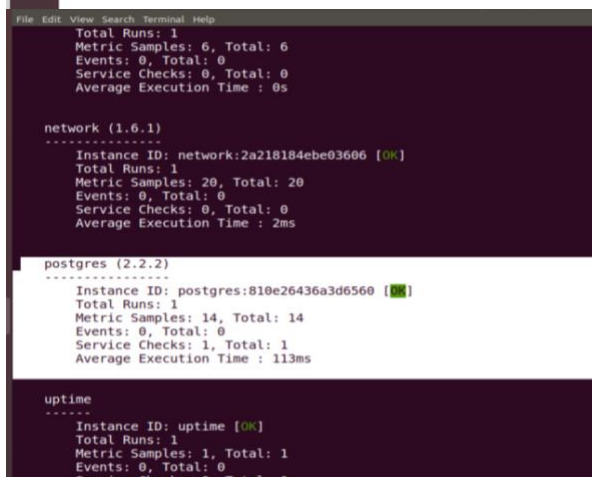
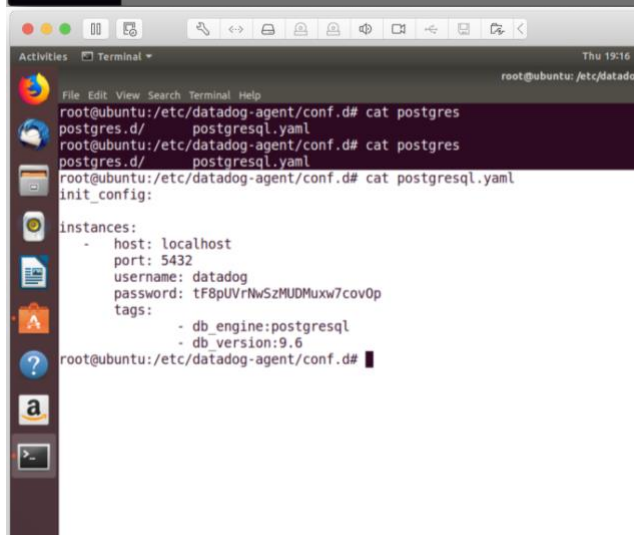
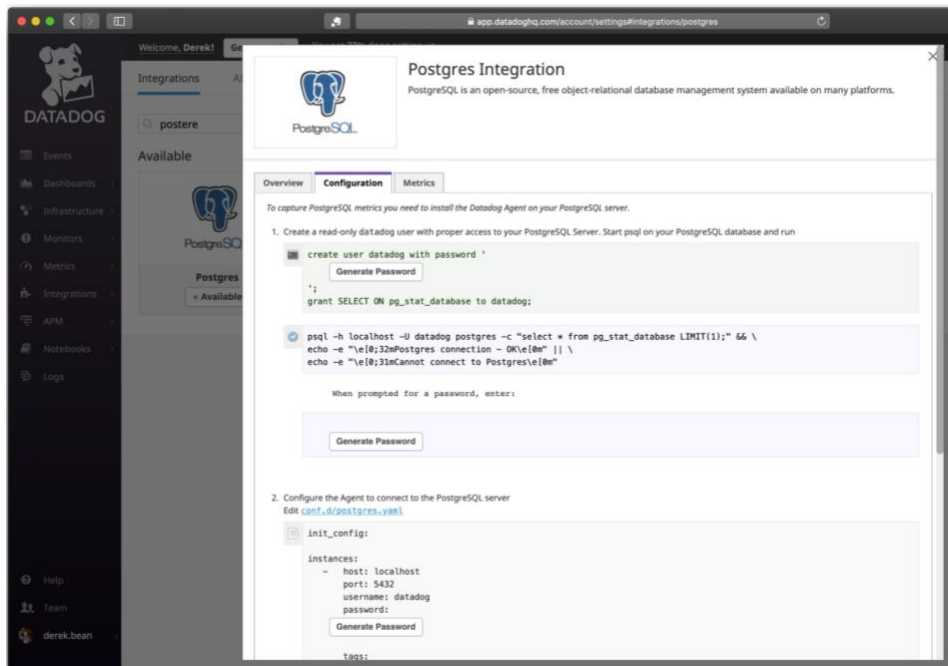
A: I've gone with PostgreSQL and installed on the virtual machine. After going through the configuration steps on the integration page, I tested to see if its reporting back correctly using the command: `datadog-agent status`

```
postgres@ubuntu:/home/dbean$ sudo /usr/lib/postgresql/10/bin/pg_ctl -D /var/lib/postgresql/10/data -l /var/log/postgresql/postgresql-10-main.log -s start
pg_ctl: cannot be run as root
Please log in (using, e.g., "su") as the (unprivileged) user that will own the server process.
postgres@ubuntu:/home/dbean$ /usr/lib/postgresql/10/bin/pg_ctl -D /var/lib/postgresql/10/data -l /var/log/postgresql/postgresql-10-main.log -s start
pg_ctl: another server might be running; trying to start anyway
waiting for server to start....bin/sh: 1: cannot create logfile: Permission denied
stopped waiting
pg_ctl: could not start server
Examine the log output.
postgres@ubuntu:/home/dbean$ netstat -an | grep 54321
postgres@ubuntu:/home/dbean$ netstat -an | grep 5432
tcp        0      0 0.0.0.0:54321        0.0.0.0:*           LISTEN
unix 2      [ ACC ] STREAM LISTENING 68948 /var/run/postgresql/.s.PGSQL.5432
postgres@ubuntu:/home/dbean$ systemctl status postgresql
● postgresql.service - PostgreSQL RDBMS
   Loaded: loaded (/lib/systemd/system/postgresql.service; enabled; vendor preset: enabled)
   Active: active (exited) since Fri 2018-10-12 20:59:00 PDT; 5min ago
     Main PID: 11006 (code=exited, status=0/SUCCESS)
       Tasks: 0 (limit: 2293)
      CGroup: /system.slice/postgresql.service

postgres@ubuntu:/home/dbean$
postgres@ubuntu:/home/dbean$
postgres@ubuntu:/home/dbean$ psql
psql (10.5 (Ubuntu 10.5-0ubuntu0.18.04))
Type "help" for help.

postgres=# \dt
Did not find any relations.
postgres=# \db
      List of tablespaces
   Name | Owner  | Location
-----+-----+-----
 pg_default | postgres | 
 pg_global  | postgres | 
(2 rows)

postgres=# \q
```



Q: Create a custom agent check that submits a metric named `my_metric` with a random value between 0 and 1000:

A: You need to create two files. One for the check file located in `/etc/datadog-agent/checks.d/` called `my_metric.py`

```
File Edit View Search Terminal Help
root@ubuntu:/etc/datadog-agent/checks.d# cat my_metric.py
from checks import AgentCheck
from random import randint

class MyMetricCheck(AgentCheck):
    def check(self, instance):
        self.gauge('my_metric', randint(0, 1000))
root@ubuntu:/etc/datadog-agent/checks.d#
```

and another one for the configuration file in `/etc/datadog-agent/conf.d/my_metric.d/` called `my_metric.yaml`

```
File Edit View Search Terminal Help
root@ubuntu:/etc/datadog-agent/conf.d# cat my_metric.yaml
init_config:

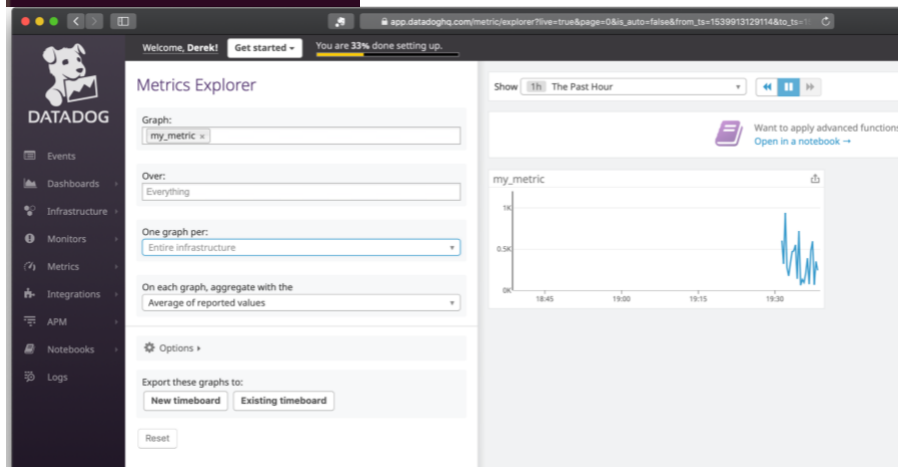
instances:
  [{}]
```

Q: Now change the collection interval to 45 seconds:

A: Update the `/etc/datadog-agent/conf.d/my_metric.d/my_metric.py` file to include the `min_collection_interval` set to 45 seconds as shown below:

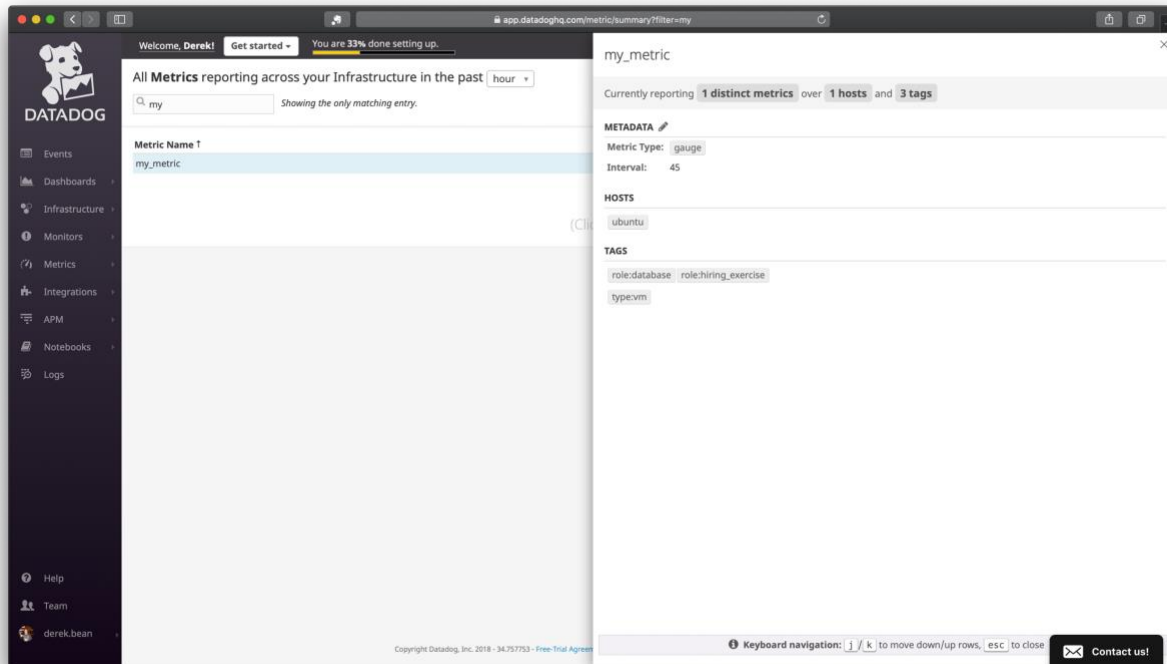
```
File Edit View Search Terminal Help
init_config:

instances:
  - min_collection_interval: 45
```



BONUS QUESTION: Can you change the collection interval without modifying the python check file

A: Yes, you can change it inside the web GUI under the metrics section by typing in 'my\_metric' in the search field. Once that metric comes up you can change the interval under the metadata section as seen below:



## VISUALIZING DATA:

Q: Use the Datadog API to create a timeboard that has the following:

- my\_metric scoped over my host
- any metric from the integration in PostgreSQL with the anomaly function applied
- my\_metric with the rollup function applied to sum up all the points for the past hour into one bucket

**NOTE:** To get some actual metrics from PostgreSQL, I used the following code to generate a little load:

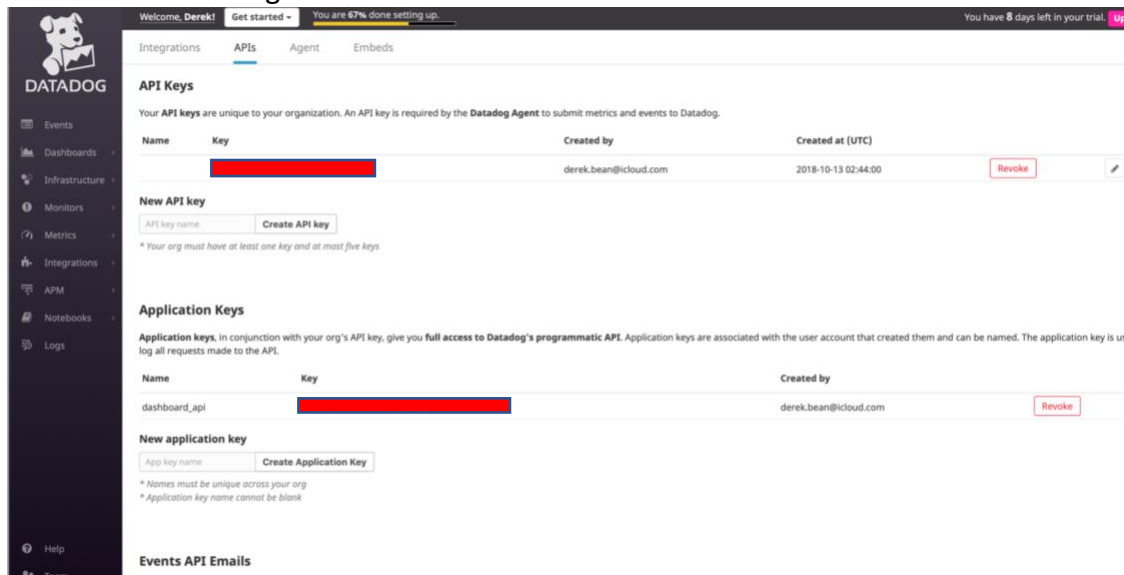
```
...
```

```
CREATE DATABASE dbname;
CREATE TABLE Departments (code VARCHAR(4), UNIQUE (code));
CREATE TABLE Towns (
  id SERIAL UNIQUE NOT NULL,
  code VARCHAR(10) NOT NULL, -- not unique
  article TEXT,
  name TEXT NOT NULL, -- not unique
  department VARCHAR(4) NOT NULL REFERENCES Departments (code),
  UNIQUE (code, department)
);
...
```

```
...
```

```
insert into towns (
  code, article, name, department
)
select
  left(md5(i::text), 10),
  md5(random()::text),
  md5(random()::text),
  left(md5(random()::text), 4)
from generate_series(1, 10000000) s(i);
...
```

A: We have to generate an app key for our API calls. You can do that under the integrations\API menu in the Datadog GUI.



With the the api key and now the app key generated, here's my code to create the dashboard:

...

```
api_key=<API KEY>
app_key=<APP KEY>
```

```
curl -X POST -H "Content-type: application/json" \
-d '{
  "graphs": [{
    "title": "My Metric across my host",
    "definition": {
      "events": [],
      "requests": [{
        "q": "avg:my_metric{host:ubuntu}"
      }]
    },
    "viz": "timeseries"
  }],
  {
    "title": "Anomalies with PostgreSQL Rows Returned",
    "definition": {
      "events": [],
      "requests": [{
        "q": "anomalies(avg:postgresql.rows_returned{*}, \u0027basic\u0027,
2)"
```

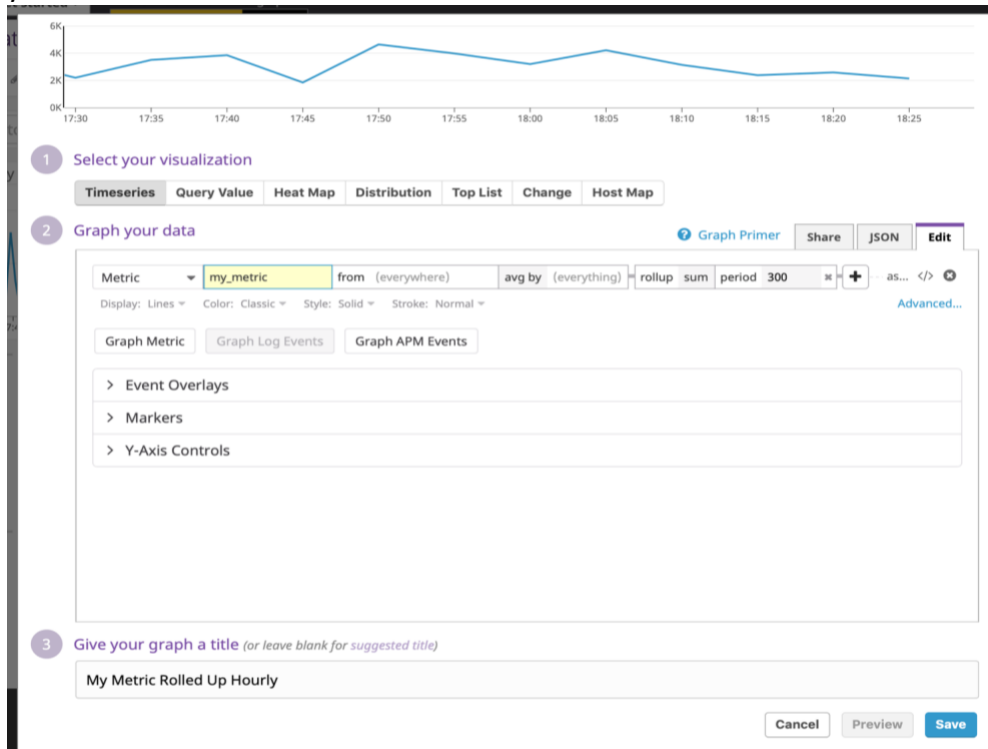


```

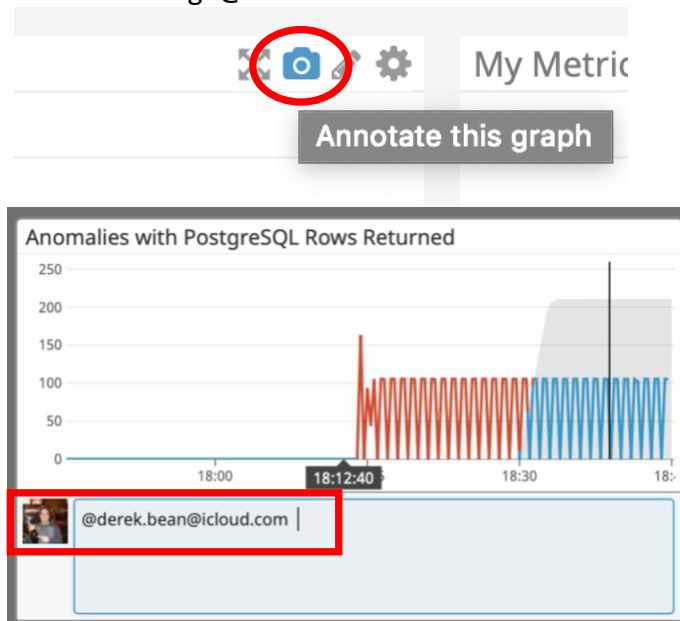
        }
      },
      "viz": "timeseries"
    },
    {
      "title": "My Metric Rolled Up Hourly",
      "definition": {
        "events": [],
        "requests": [{
          "q": "avg:my_metric{*}.rollup(sum, 3600)"
        }]
      },
      "viz": "timeseries"
    }
  ],
  "title": "Data Visualizations",
  "description": "Exercise Dashboard",
  "read_only": "True"
} \
"https://api.datadoghq.com/api/v1/dash?api_key=${api_key}&application_key=${app_key}"
...

```

Now change the timeframe to the past 5 minutes and add in an '@' notation to send it to yourself:



Using the camera icon on the right side (Annotate), you can send a capture of the graph to someone using '@' and their email address:

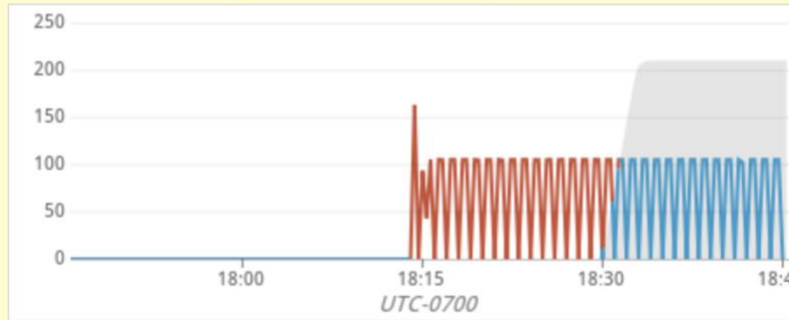


Derek Bean ([@derek.bean@icloud.com](mailto:@derek.bean@icloud.com)) mentioned you in a comment:



**Derek Bean**

[Anomalies with PostgreSQL Rows Returned](#)



[@derek.bean@icloud.com](mailto:@derek.bean@icloud.com)

20 Oct, 01:45:57 UTC

**BONUS QUESTION:** What's the Anomaly graph displaying?

A: An anomaly graph distinguishes between normal and abnormal metric trends on things like web requests, user logins, etc. Due to the fluctuations of these metrics, either by the time of day or by the day of the week, it's tough to set a hard threshold to alert on. Here the gray band shows the anomaly detection algorithm's predicted range based on data from prior weeks.

## MONITORING DATA:

Q: Create a new metric monitor watches the average of your custom metric and will alert if it's over the following values over the past 5 minutes

- warning threshold of 500
- alerting threshold of 800
- notify if there's no data for past 10 minutes

Configure the alert to email when it triggers, Create different messages for alert, warning, and no data states. Include the metric value that caused the monitor to trigger and host ip when the monitor triggers an alert state.

A: I've created a monitor to watch the average of my metric as shown here

**1 Choose the detection method**

Threshold Alert Change Alert Anomaly Detection Outliers Alert Forecast Alert

An alert is triggered whenever a metric crosses a threshold.

**2 Define the metric**

a Metric **my\_metric** from (everywhere) excluding (none) avg 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: Alert recovery threshold (opt)

Warning recovery threshold: Warning recovery threshold

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.

Notify if data is missing for more than **10** minutes.

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

[Never] automatically resolve this event from a no data state.

With the following code for the monitor's message:

```
``{{#is_no_data}}  
{{host.name}} has not received any data from my_metric!  
{{/is_no_data}}
```

```
{{#is_warning}}  
Host {{host.name}} is now in WARNING state! The value is: {{value}}  
{{/is_warning}}
```

{{#is\_alert}}

Host {{host.name}} at {{host.ip}} is now in ALERT state! The value is: {{value}}. Please investigate this immediately @dbean@qti.qualcomm.com

{{/is\_alert}}

@dbean@qti.qualcomm.com

...

## BONUS QUESTION #2:

Set up two scheduled downtimes for this monitor. One that will silence it from 7pm to 9am daily M-F and one that will do it for Sat and Sun only. Make sure you're notified when you schedule them and take a screenshot.

A:

The screenshot shows the Nagios XI interface for configuring a monitor. The steps are numbered 1 through 4:

- 1** **Monitor**: Select "By monitor name". Monitor: My\_Monitor. Group scope (optional, default all groups): \*
- 2** **Schedule**: One-Time | Recurring. Start Date: 2018/10/21. Time Zone: America/Los\_Angeles. Repeat Every: 1 weeks. Repeat On: ☐ Sun ☒ Mon ☒ Tue ☒ Wed ☒ Thu ☒ Fri ☐ Sat. Beginning: 7:00PM. Duration: 14 hours. Repeat Until: No end date. Summary: From 7:00pm to 9:00am tomorrow. Weekly on Monday, Tuesday, Wednesday, Thursday, and Friday.
- 3** **Add a message**: Preview | Edit. Markdown supported. This is a downtime set for every Monday to Friday from 7pm to 9am. @derek.bean@icloud.com
- 4** **Notify your team**: Derek Bean x

Buttons: Cancel, Save



Derek Bean ([@derek.bean@icloud.com](#)) mentioned you in a comment:



**Derek Bean**

**Derek Bean** scheduled downtime on [My Monitor](#) from **2:00AM** to **4:00PM UTC** on **October 22**.

This is a downtime set for every Monday to Friday from 7pm to 9am.

[@derek.bean@icloud.com](#)

21 Oct, 18:39:25 UTC

[Reply to @derek.bean@icloud.com](#)

To manage your Datadog subscriptions, click [here](#).



Derek Bean ([@derek.bean@icloud.com](#)) mentioned you in a comment:



**Derek Bean**

**Derek Bean** scheduled downtime on [My Monitor](#) from **7:00AM** on **October 20** to **7:00AM UTC** on **October 21**.

Alarm silenced for my\_metric from Sat to Sun all day

[@dbean@qti.qualcomm.com](#)

19 Oct, 22:23:59 UTC

[Reply to @derek.bean@icloud.com](#)

To manage your Datadog subscriptions, click [here](#).

## COLLECTING APM DATA:

Q: Given the following Flask app, instrument this using Datadog's APM solution. Provide a link and a screenshot of of a dashboard with both APM and infrastructure metrics

A:

First you need to install the ddtrace package using:

```
root@ubuntu: cd /opt/datadog-agent/embedded/bin
```

```
root@ubuntu: ./pip install ddtrace
```

Then I edited the `/etc/datadog-agent/datadog.yaml` file to have the following under the APM section and restarted the datadog agent:

apm\_config:

enabled: true

receiver\_port: 8126

extra\_sample\_rate: 1.0

max\_traces\_per\_second: 0

analyzed\_spans:

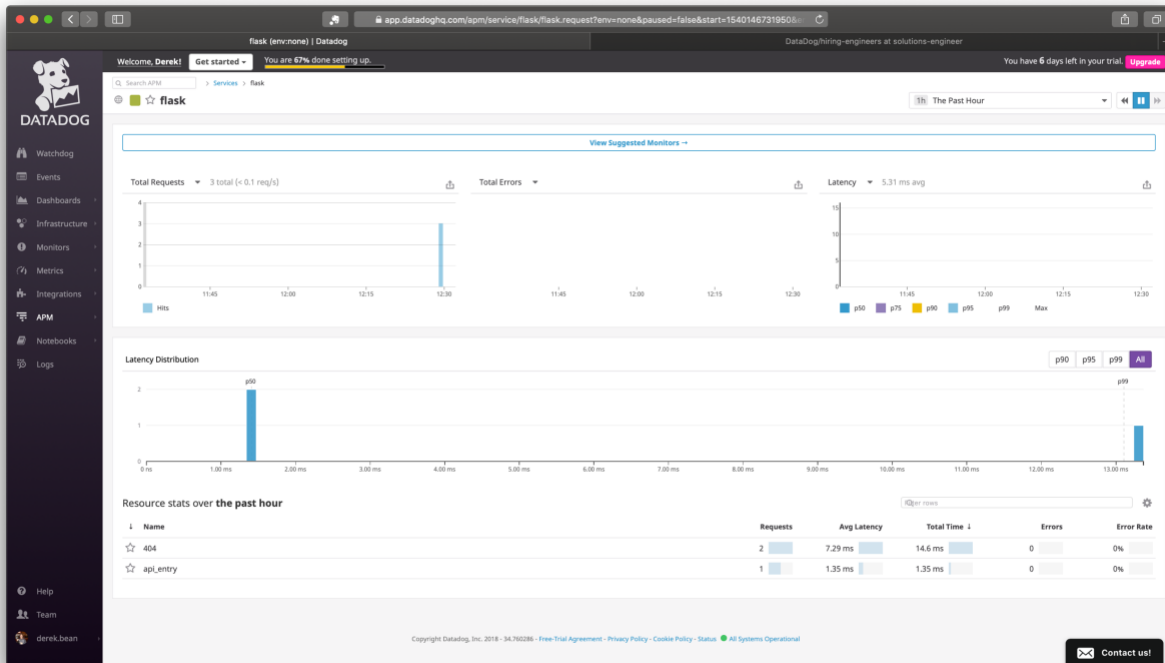
flask|flask.request: 1

From a browser in my VM, I went to <http://0.0.0.0:5050> and did a bunch of requests to that page to generate some stats.

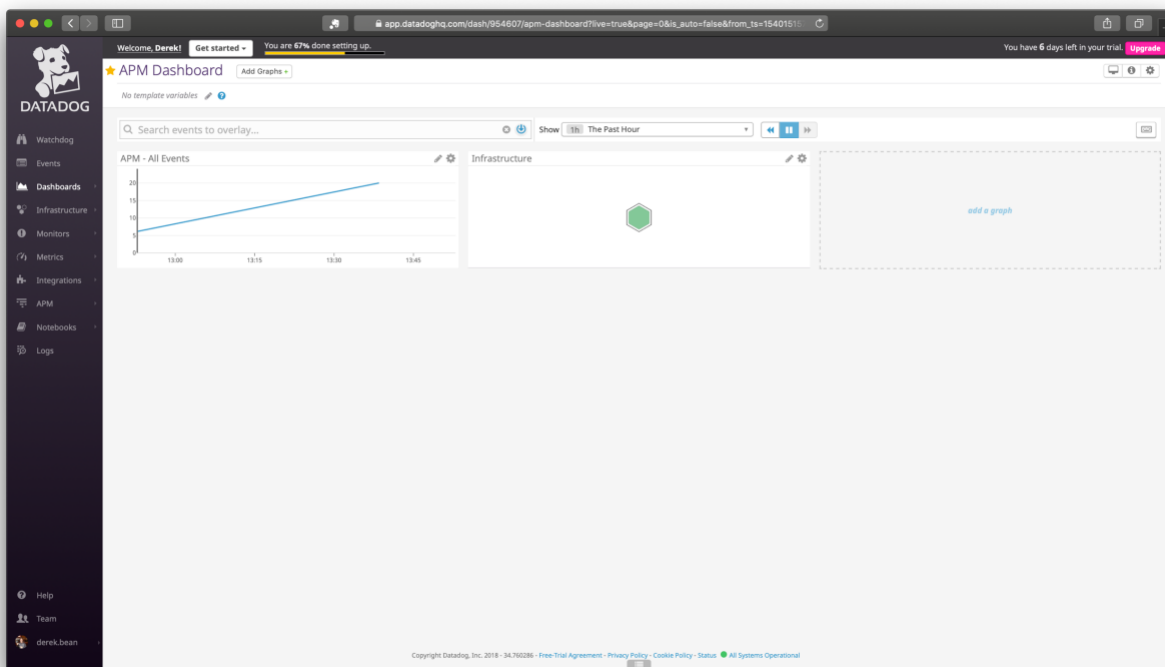
Back in the datadog web page, you'll see stats under the APM > Trace List.

Trace List

Service: All Status: All OK Errors Duration: No minimum					
Date ↓	Type	Service	Resource	Duration	Status
10/21/2018 1:38:59 PM	🌐	flask	api_entry	371 μs	200
10/21/2018 1:38:55 PM	🌐	flask	api_entry	368 μs	200
10/21/2018 1:38:55 PM	🌐	flask	api_entry	457 μs	200
10/21/2018 1:38:52 PM	🌐	flask	api_entry	422 μs	200
10/21/2018 1:38:51 PM	🌐	flask	api_entry	367 μs	200
10/21/2018 1:38:51 PM	🌐	flask	api_entry	436 μs	200
10/21/2018 1:38:49 PM	🌐	flask	api_entry	698 μs	200
10/21/2018 1:38:49 PM	🌐	flask	api_entry	389 μs	200
10/21/2018 1:38:48 PM	🌐	flask	api_entry	368 μs	200
10/21/2018 1:38:47 PM	🌐	flask	api_entry	377 μs	200
10/21/2018 1:38:46 PM	🌐	flask	api_entry	1.60 ms	200



I'm now able to use them in a dashboard as seen below:



From which I shared each of the timelines using the 'share' feature tab:

[https://app.datadoghq.com/graph/embed?token=e273775460e78b4e0c2234745b8c569f6eed0251a88debeb9dbfbf4eabdb1a8f&height=300&width=600&legend=true" width="600" height="300" frameborder="0"](https://app.datadoghq.com/graph/embed?token=e273775460e78b4e0c2234745b8c569f6eed0251a88debeb9dbfbf4eabdb1a8f&height=300&width=600&legend=true)



[</iframe>](https://app.datadoghq.com/graph/embed?token=ff1d51d6567fe49770ce93c39e4399c5c3b2ee6b21bf9c89e0a27b6dcab22ee4&height=300&width=600&legend=true)

BONUS QUESTION #3:

Q: What is the difference between a service and a resource?

A: A service is a set of processes that do the same job (ex. web or database server) whereas a resource is a particular action for a service.

FINAL QUESTION:

Q: Is there anything creative you would use Datadog for?

A: I was thinking that you could use it for McDonald's (or any restaurant for that matter) food trending and analysis. (Yes, sadly I've worked there 😊)

Graphing anomalies over time with the different food items that have been created during the week, the amount that's been wasted (food that's put in a bucket that they count each night for orders that are wrong or something that's made incorrectly), and with each employee on an overlay graph could show if someone is more prone to mistakes or how much you're selling over time to see if there's a pattern. Just a little data analytics could go pretty far in the food industry actually. You could even set up alerting different owners when that store or region hits a different level of profit.