

Setting Up the environment:

We used vagrant along with virtual box for this task. For the guest machine we choose Ubuntu 16.04

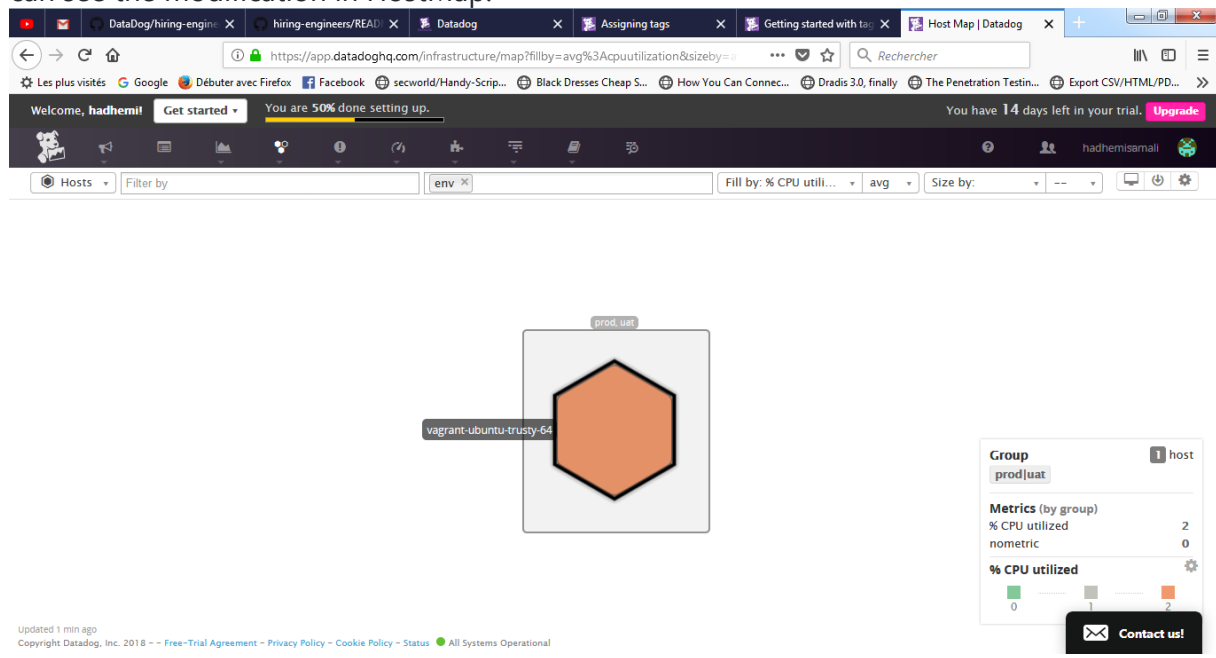
Collecting Metrics:

1. 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

For this task, we added the tags in the config file on the host, under `/etc/datadog-agent/datadog.yaml`:

```
# Set the host's tags (optional)
tags: region:east, env:prod, region:fr, env:uat, env:test, region:west, application:database
```

Making this modification wasn't particularly complicated, we had to wait a while before we can see the modification in HostMap.



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

We choose to install MySQL on the host, and its associated Datadog integration.

The integration process was very clear and easy to follow, except the config file path that was not correct: `conf.d/mysql.yaml` when it is `conf.d/mysql.d/conf.yaml`

2. Configure the Agent to connect to MySQL

Edit [conf.d/mysql.d/conf.yaml](#)

Once we made the configuration, we run the check to see if everything is ok:

```
memory
-----
Total Runs: 2
Metrics: 16, Total Metrics: 32
Events: 0, Total Events: 0
Service Checks: 0, Total Service Checks: 0
Average Execution Time : 0ms

mysql
-----
Total Runs: 2
Metrics: 64, Total Metrics: 127
Events: 0, Total Events: 0
Service Checks: 1, Total Service Checks: 2
Average Execution Time : 89ms

network
-----
Total Runs: 2
Metrics: 26, Total Metrics: 52
Events: 0, Total Events: 0
Service Checks: 0, Total Service Checks: 0
Average Execution Time : 3ms

ntp
---
Total Runs: 2
Metrics: 1, Total Metrics: 2
Events: 0, Total Events: 0
Service Checks: 1, Total Service Checks: 2
Average Execution Time : 39ms
```

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

After reading the documentation of how to create a custom agent check, we created a custom agent check that creates a random value between 0 & 1000 named "my_metric".

```
from checks import AgentCheck

from random import randint

class myCheck(AgentCheck):
    def check(self, instance):
        x = randint(0, 1000)
        self.gauge('my_metric', x)
```

Check that the metric is running correctly:

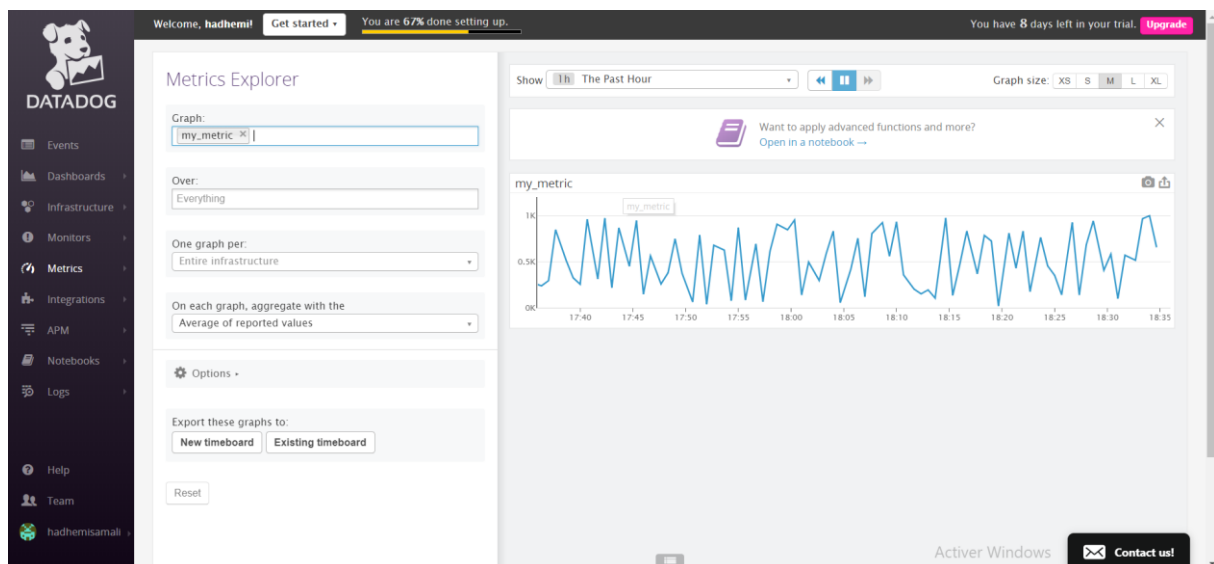
```

my_metric
-----
Total Runs: 1
Metrics: 1, Total Metrics: 1
Events: 0, Total Events: 0
Service Checks: 0, Total Service Checks: 0
Average Execution Time : 0ms

mysql
-----
Total Runs: 4
Metrics: 61, Total Metrics: 243
Events: 0, Total Events: 0
Service Checks: 1, Total Service Checks: 4
Average Execution Time : 123ms

```

Once this is done, we visualize the my_metric metric through the WEB UI.



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

We thought about adding a wait timer in the python code, this will stop the code 45s every time before submitting the metric.

```

from checks import AgentCheck

from random import randint
from time import sleep

class myCheck(AgentCheck):
    def check(self, instance):
        sleep(45)
        x = randint(0, 1000)
        self.gauge('my_metric', x)

```

5. **Bonus Question Can you change the collection interval without modifying the Python check file you created?**

It is mentioned in the documentation that the collection interval can be configured in the YAML files using `min_collection_interval`.

`datadog-agent/conf.d/my_metric.d/my_metric.yaml`

```
init_config:

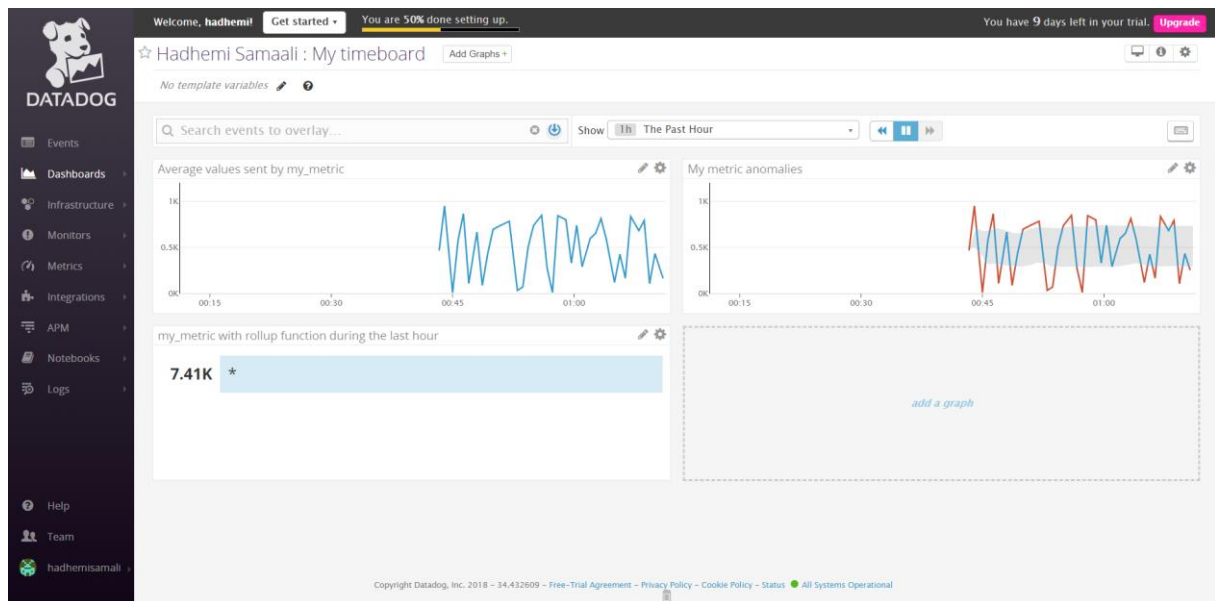
instances:
  - host: localhost
    min_collection_interval: 45
    name: my_metric
```

Visualizing Data

1. **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

It took me a while to read and understand how the Datadog API works, to make the query I sometimes use the graphic UI to understand the syntax.

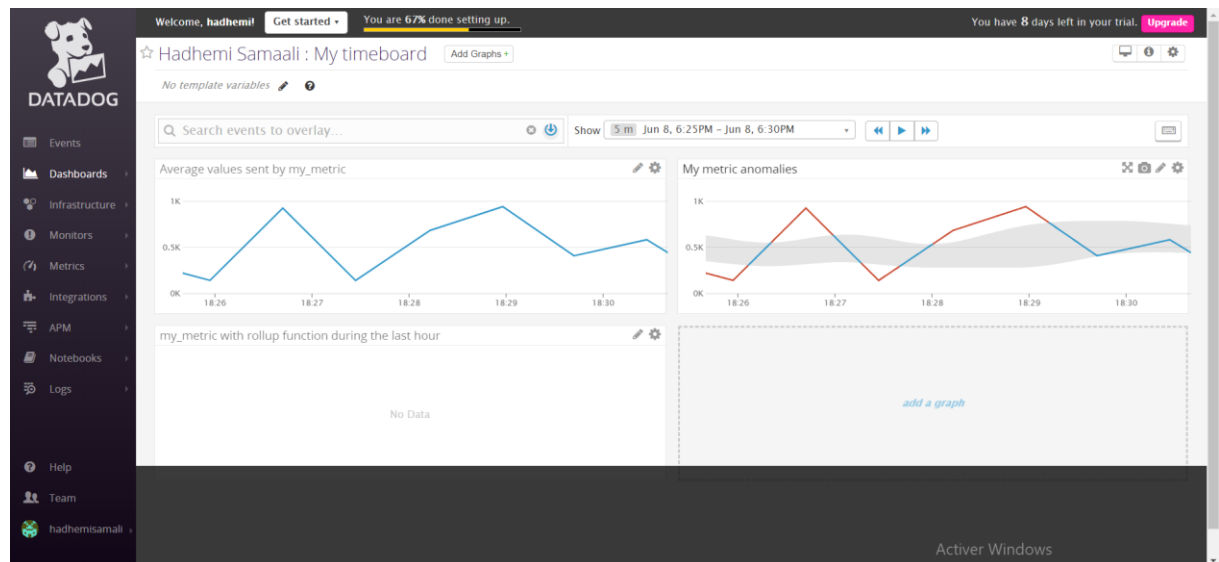
This is the timeboard code along with anomaly detection:



2. **Please be sure, when submitting your hiring challenge, to include the script that you've used to create this Timeboard.**

Done

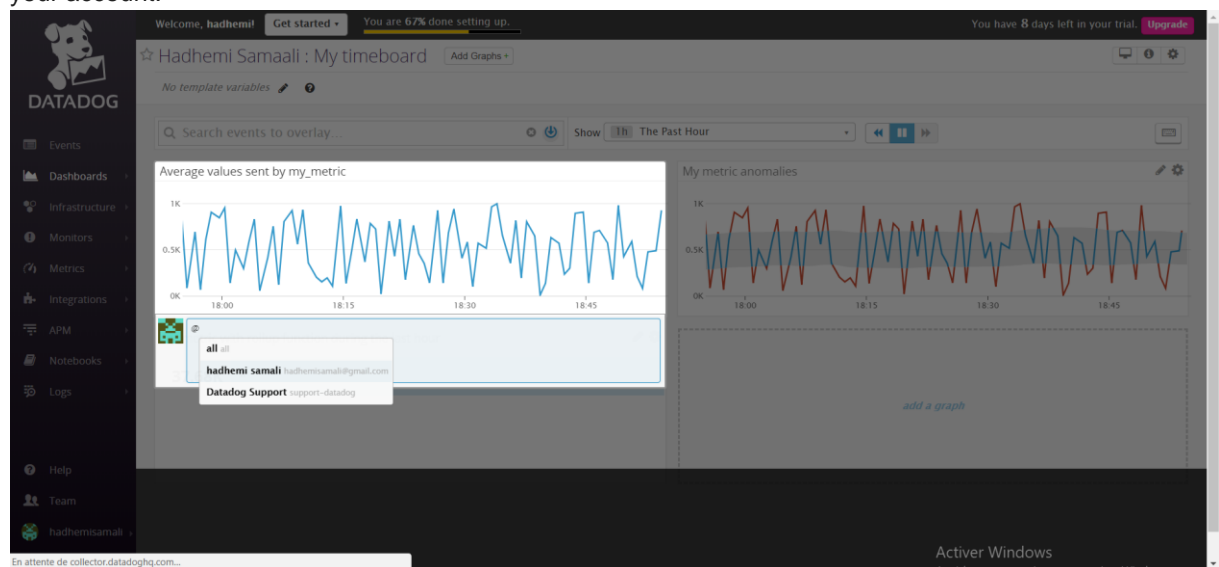
3. **Once this is created, access the Dashboard from your Dashboard List in the UI:**
 - Set the Timeboard's timeframe to the past 5 minutes

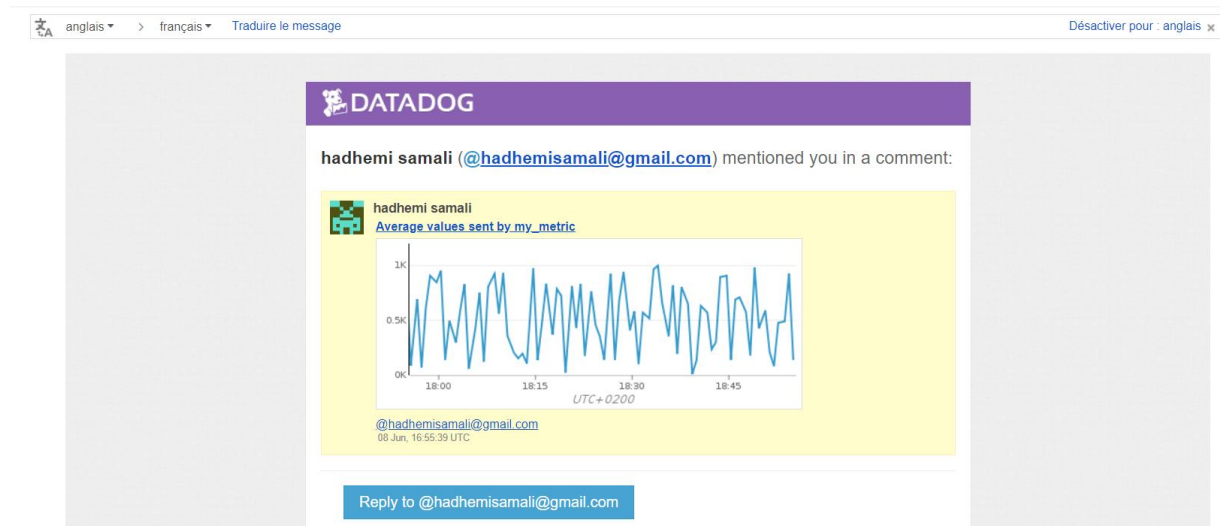


We access the newly created Dashboard, and adjust the Timeframe to the past 5 minutes.

4. Take a snapshot of this graph and use the @ notation to send it to yourself.

All we had to do is to simply choose the camera icon, add your name to the '@' notation to send it to your account.





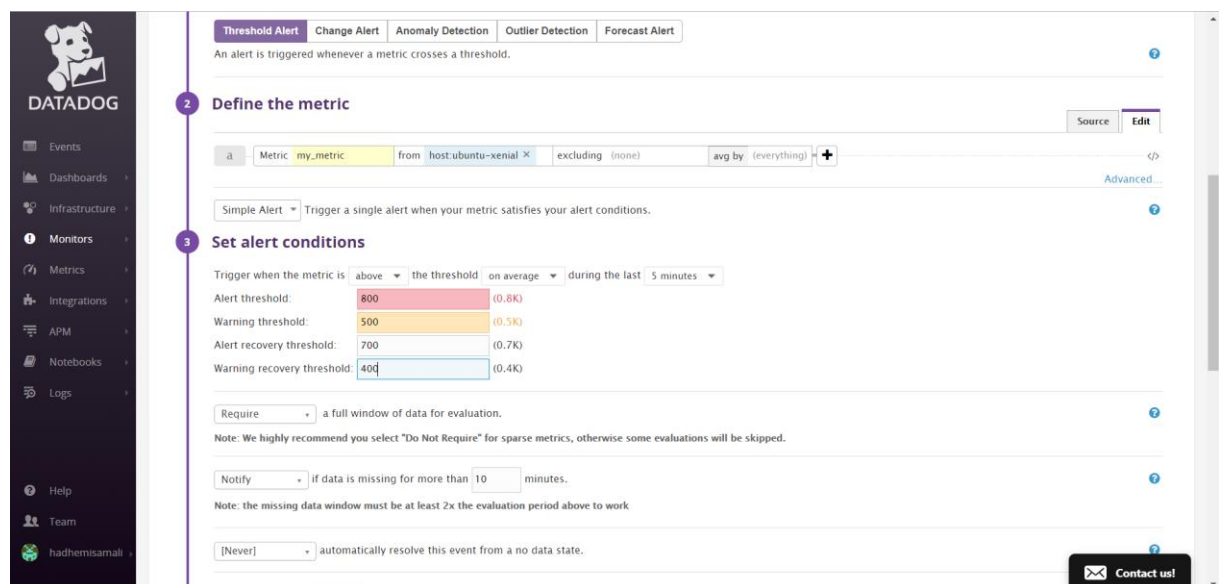
5. Bonus Question: What is the Anomaly graph displaying?

Anomaly graph is used to determine when a metric is behaving abnormally, the abnormal behavior is determined based on the past metric values. It will display the anomaly of the metric behavior.

Monitoring Data

1. 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.

Alert recovery threshold: (0.7K)
Warning recovery threshold: (0.4K)

Require [?](#)
Note: We highly recommend you select "Do Not Require" for sparse metrics, otherwise some evaluations will be skipped.

Notify [?](#)
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 [?](#)

4 Say what's happening [Use message template variables](#) [?](#)

[Preview](#) [Edit](#) [Markdown supported](#)

My metric alert

```

{{#is_alert}} This is a notification regarding the alert coming from {{host_ip}} , made by {{value}} {{/is_alert}}
{{#is_warning}}Warning{{/is_warning}}
{{#is_no_data}}No data since 10 min{{/is_no_data}}

@hadhemisamali@gmail.com

```

Tags:

[notify if the monitor has not been reached](#)

[Contact us!](#)

When this monitor sends you an email notification, take a screenshot of the email that it sends you.

2. Bonus Question: Since this monitor is going to alert pretty often, you don't want to be alerted when you are out of the office. Set up two scheduled downtimes for this monitor:

- One that silences it from 7pm to 9am daily on M-F,
- And one that silences it all day on Sat-Sun.

2 Schedule

One-Time **Recurring**

Start Date: 2018/06/09 Time Zone: Europe/Paris
Repeat Every: 1 weeks
Repeat On: ☒ Sun ☐ Mon ☐ Tue ☐ Wed ☐ Thu ☐ Fri ☒ Sat
Beginning: 00:00
Duration: 1 days
Repeat Until: No end date
Summary: From 12:00am to 12:00am the next day
Weekly on Sunday and Saturday

[Preview planned recurrences](#)

3 Add a message

[Preview](#) [Edit](#) Markdown supported

This the weekly silence
@hadhemisamali@gmail.com

2 Schedule

One-Time **Recurring**

Start Date: 2018/06/08 Time Zone: Europe/Paris
Repeat Every: 1 weeks
Repeat On: ☐ Sun ☒ Mon ☒ Tue ☒ Wed ☒ Thu ☒ Fri ☐ Sat
Beginning: 19:00
Duration: 14 hours
Repeat Until: No end date
Summary: From 7:00pm to 9:00am
Weekly on Monday, Tuesday, Wednesday, Thursday, and Friday

[Preview planned recurrences](#)

3 Add a message

[Preview](#) [Edit](#) Markdown supported

This the daily silence @hadhemisamali@gmail.com

Make sure that your email is notified when you schedule the downtime and take a screenshot of that notification.

 DATADOG

hadhemi samali (@hadhemisamali@gmail.com) mentioned you in a comment:



hadhemi samali
hadhemi samali scheduled downtime on [My metric alert](#) from 5:00PM on June 8 to 7:00AM UTC on June 9.
This the daily silence @hadhemisamali@gmail.com
08 Jun, 11:17:05 UTC

Reply to @hadhemisamali@gmail.com

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

 DATADOG

hadhemi samali (@hadhemisamali@gmail.com) mentioned you in a comment:







hadhemi samali
hadhemi samali scheduled downtime on [My metric alert](#) from 10:00PM on June 8 to 10:00PM UTC on June 9.
This is the weekly slience @hadhemisamali@gmail.com
08 Jun, 11:15:34 UTC


Reply to @hadhemisamali@gmail.com


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

When the downtime starts:


 **Datadog** help@datadoghq.com via [outbound.dtdg.co](#)
A moi 

19:00 (Il y a 0 minute)  

 anglais > français [Traduire le message](#) [Désactiver pour : anglais](#)

 DATADOG

A Datadog event mentioned you:



Scheduled downtime on My metric alert started
Scheduled downtime on [My metric alert](#) has started.
Alerting on [My metric alert](#) will be silenced until 7:00AM UTC on June 9.
This the daily silence @hadhemisamali@gmail.com
08 Jun, 17:00:25 UTC

Reply on Datadog

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

Collecting APM Data

1. Bonus Question: What is the difference between a Service and a Resource?

A service is a set of processes that do the same job, for example database services or webapp services. While a resource is a particular action for a service.

2. Provide a link and a screenshot of a Dashboard with both APM and Infrastructure Metrics.

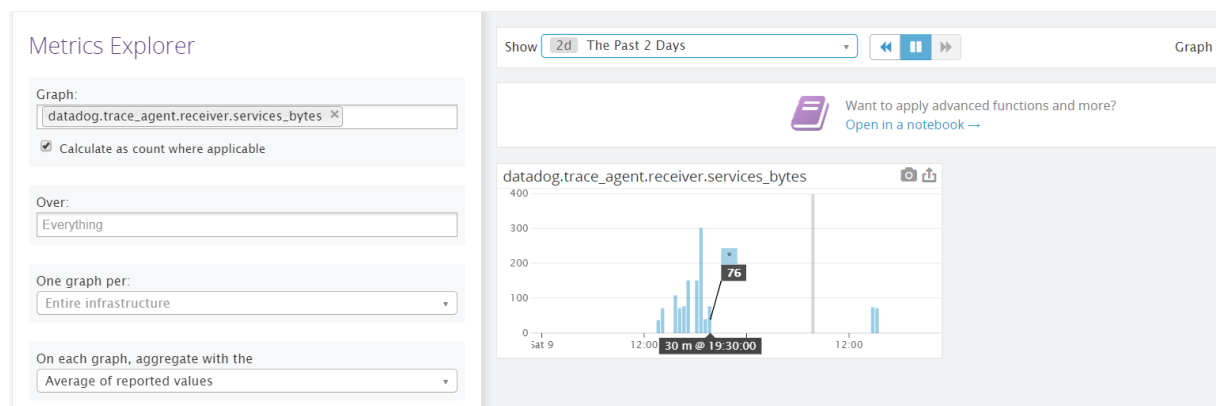
For this APM configuration, we started by installing dd-trace, blinker and flask for python using pip and then setting up the APM configuration in datadog.yaml (enable APM and set env to prod) :

```
# Trace Agent Specific Settings
#
apm_config:
# Whether or not the APM Agent should run
enabled: true
# The environment
# tag that Traces should be tagged with
# Will inherit from "env" tag if none is applied here
env: prod
```

Then we run the flask app using dd-trace :

```
root@ubuntu-xenial:/etc/datadog-agent# ddtrace-run python checks.d/flaskapp.py &
[1] 5754
root@ubuntu-xenial:/etc/datadog-agent# DEBUG:ddtrace.contrib.flask.middleware:flask: initializing trace middleware
2018-06-10 13:03:13,604 - ddtrace.contrib.flask.middleware - DEBUG - flask: initializing trace middleware
DEBUG:ddtrace.writer:resetting queues. pids(old:None new:5754)
2018-06-10 13:03:13,605 - ddtrace.writer - DEBUG - resetting queues. pids(old:None new:5754)
DEBUG:ddtrace.writer:starting flush thread
2018-06-10 13:03:13,605 - ddtrace.writer - DEBUG - starting flush thread
* Serving Flask app "flaskapp" (lazy loading)
* Environment: production
  WARNING: Do not use the development server in a production environment.
  Use a production WSGI server instead.
* Debug mode: off
INFO:werkzeug: * Running on http://127.0.0.1:5050/ (Press CTRL+C to quit)
2018-06-10 13:03:13,609 - werkzeug - INFO - * Running on http://127.0.0.1:5050/ (Press CTRL+C to quit)
DEBUG:ddtrace.api:reported 1 services
2018-06-10 13:03:13,610 - ddtrace.api - DEBUG - reported 1 services
```

However, in our datadog UI we were unable to see the APM metrics, even though we can see the trace agent services receiving bytes:



Dashboard URL:

https://app.datadoghq.com/dash/826035/hadhemis-timeboard-2-jun-2018-1944?live=true&page=0&is_auto=false&from_ts=1528491582880&to_ts=1528664382880&tile_size=m

