**Name:** Fawzi Hmouda

**Email:** [h.fawzi@hotmail.com](mailto:h.fawzi@hotmail.com)

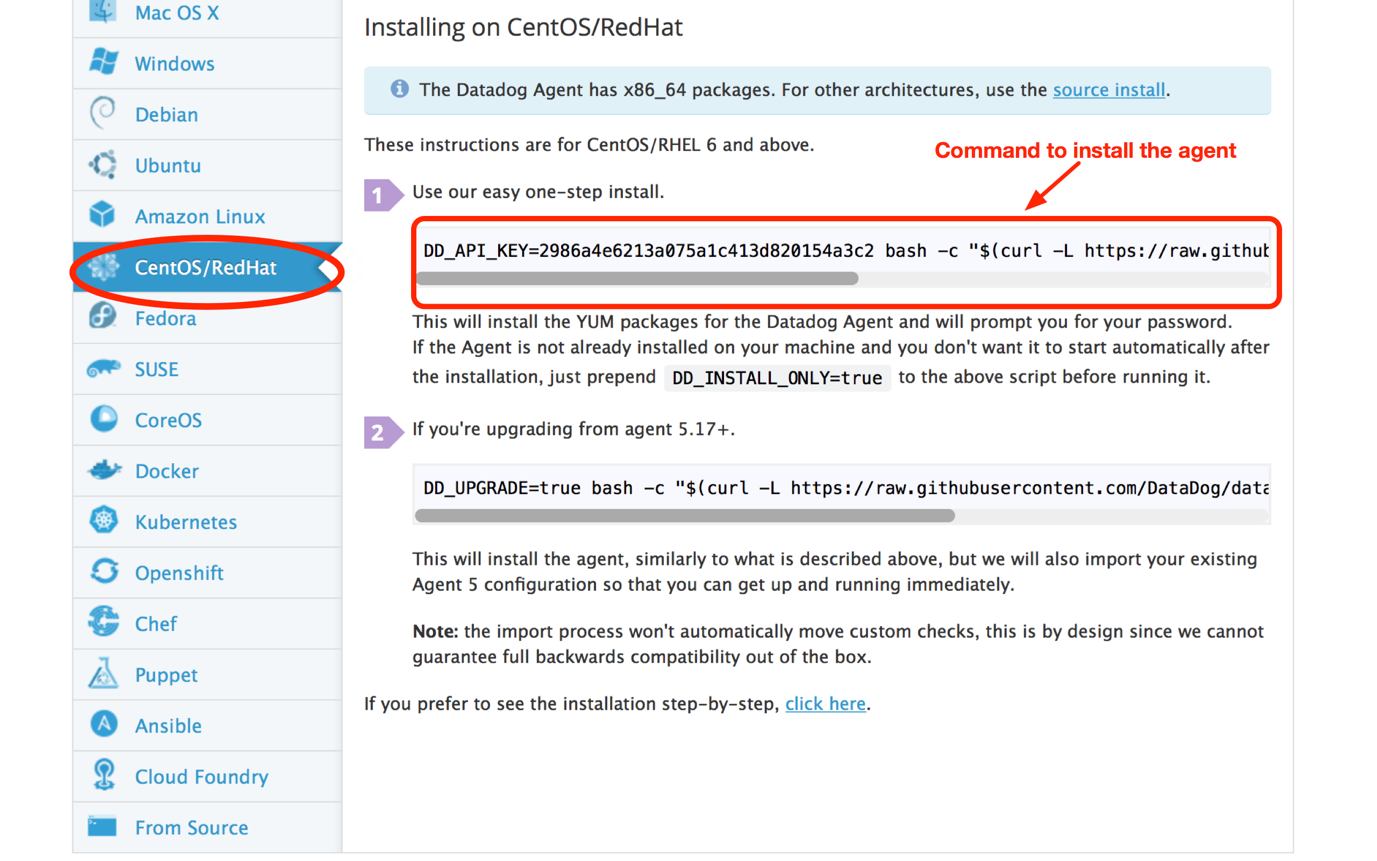
**Solution Engineer,** Sydney, Australia

**1-Prerequisites – Setup the enviroment**

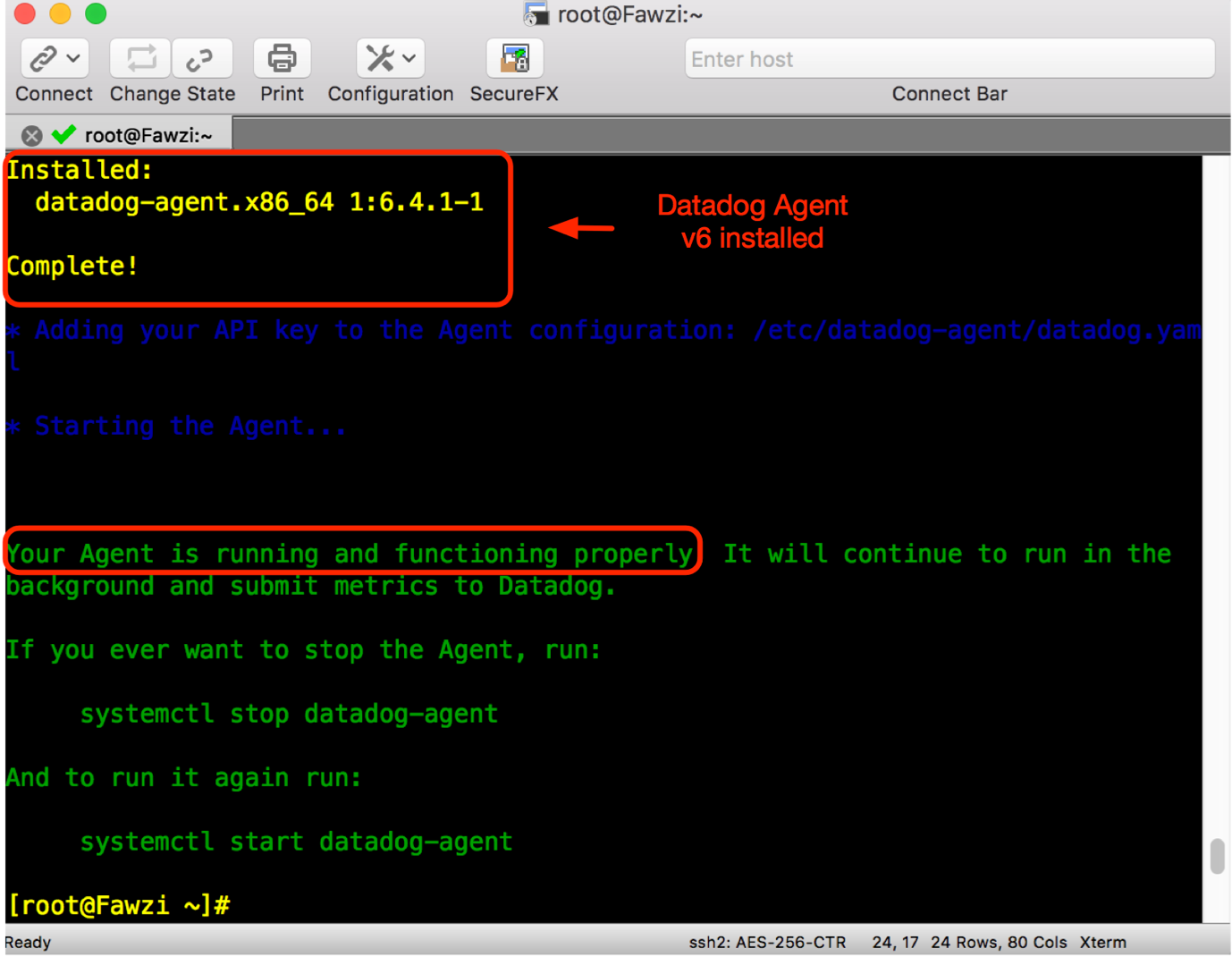
I chose virtualized CentOS Linux server inside VMware Fusion for this exercise as I have a good experience in Linux administration and I setup an SSH server, so I can access my CentOS shell from my local machine (MacOS).

**Agent Installation**

Because this is a fresh installation, there is no agent upgrade, only a fresh installation of the newest version of Datadog agent which is v6 until this date.



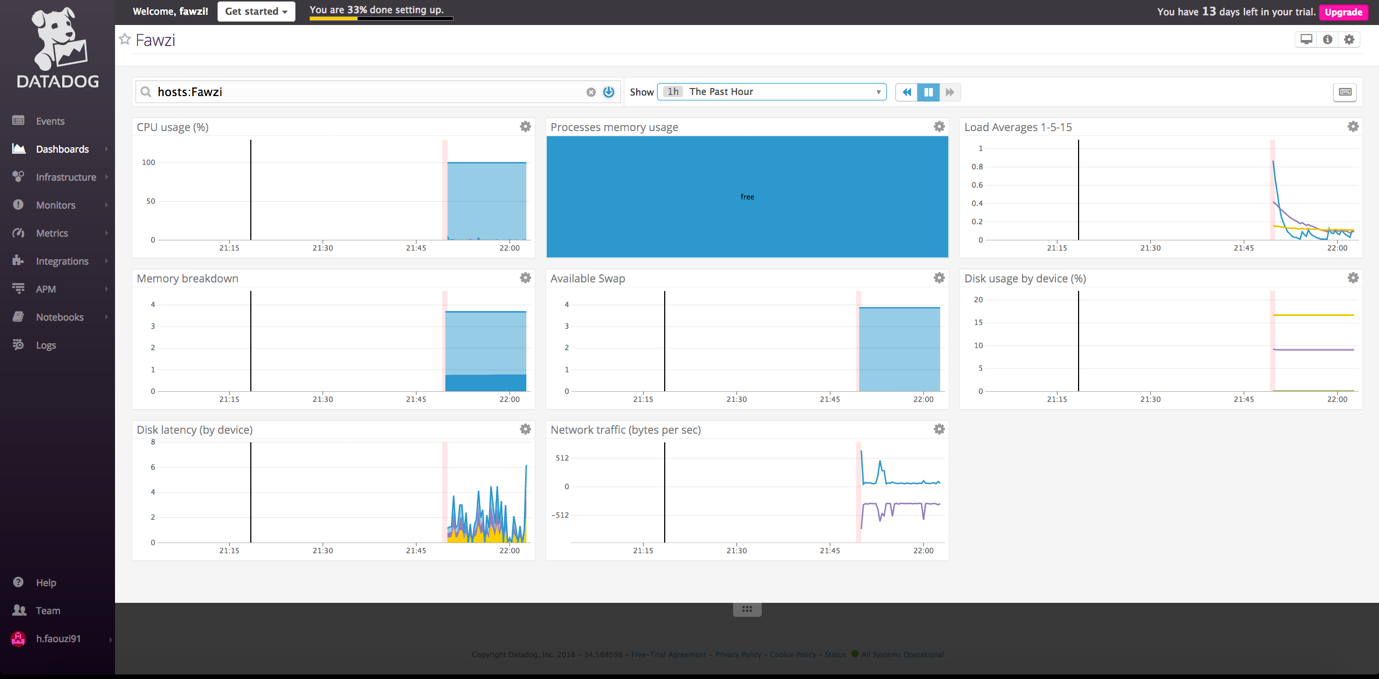
After running the command, the picture bellow shows that the installation is competed, and the agent is running and functioning properly;



Later, if something went wrong we can still troubleshoot the agent and check its functionality ([Agent Troubleshooting](https://docs.datadoghq.com/agent/troubleshooting/)). For some circumstances, we might stop, start, restart and show the status of the agent if required by issuing the following commands;

**#service datadog-agent** (stop/start/restart/status)

The following image is the metrics of our host after installing the agent;



**2-Collecting Metrics**

**Assigning tags**

There is 4 ways of assigning tags: API, UI, Agent configuration file and from integrations. In this task, we are required to use the agent configuration file to assign tags.

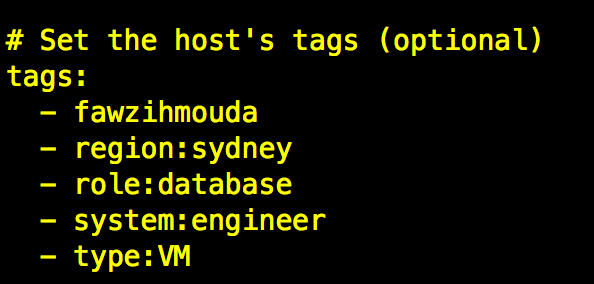
There are difference approaches when assigning tags, as per Datadog and AWS best practice, the format of a tag should be **Key : Value** where a keys can be role, region, APP etc..

If we have a tag assigned at the agent level, all metrics at the integration level will share that particular tag.

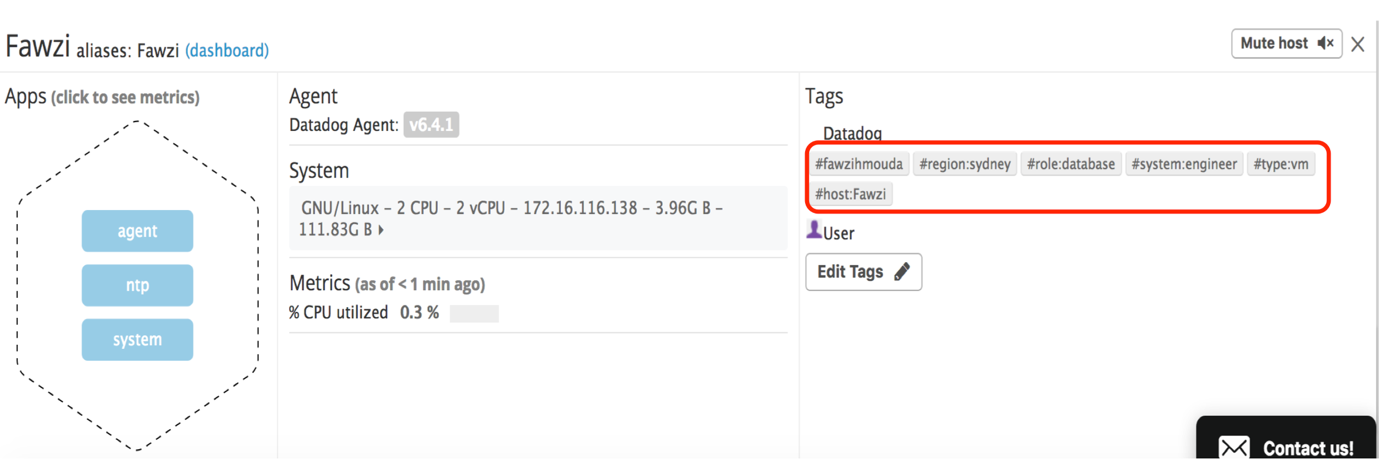
To assign a tag, a configuration file which is the **datadog.yaml** should be edited and it is located under the **datadog-agent** directory;

**/etc/datagog-agent/ datadog.yaml**

The following image shows that tags were added in the configuration file;

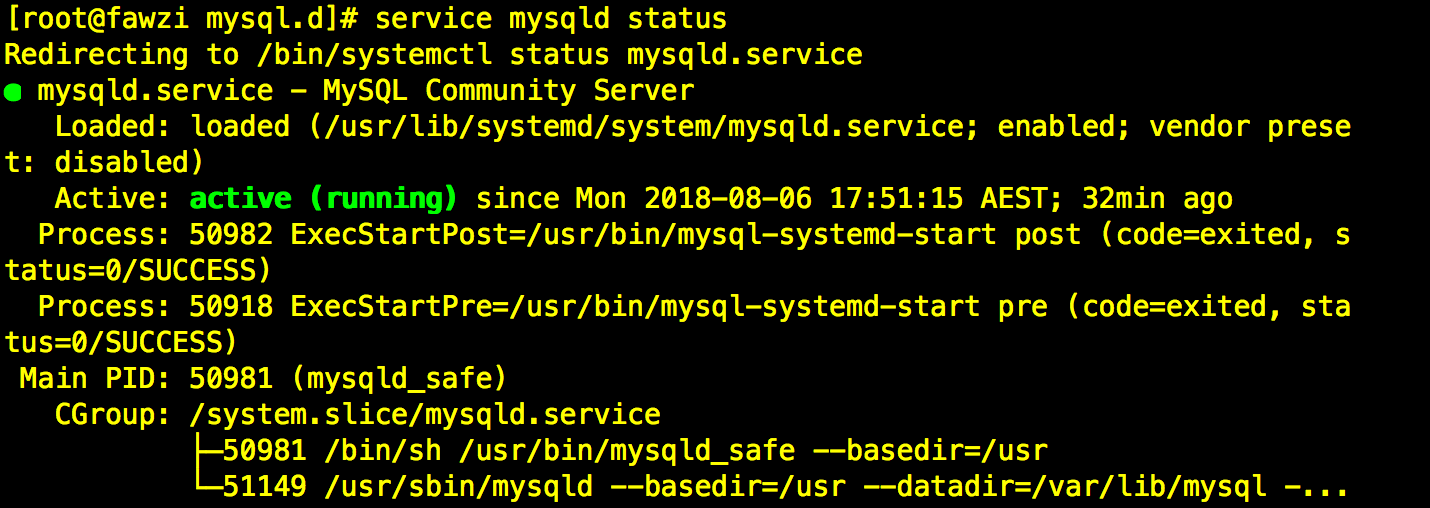


The agent **must be restarted** in order to apply these tags. The following picture is the host map page and it shows that the tags are set to the host according to the datadog.Yaml file above;



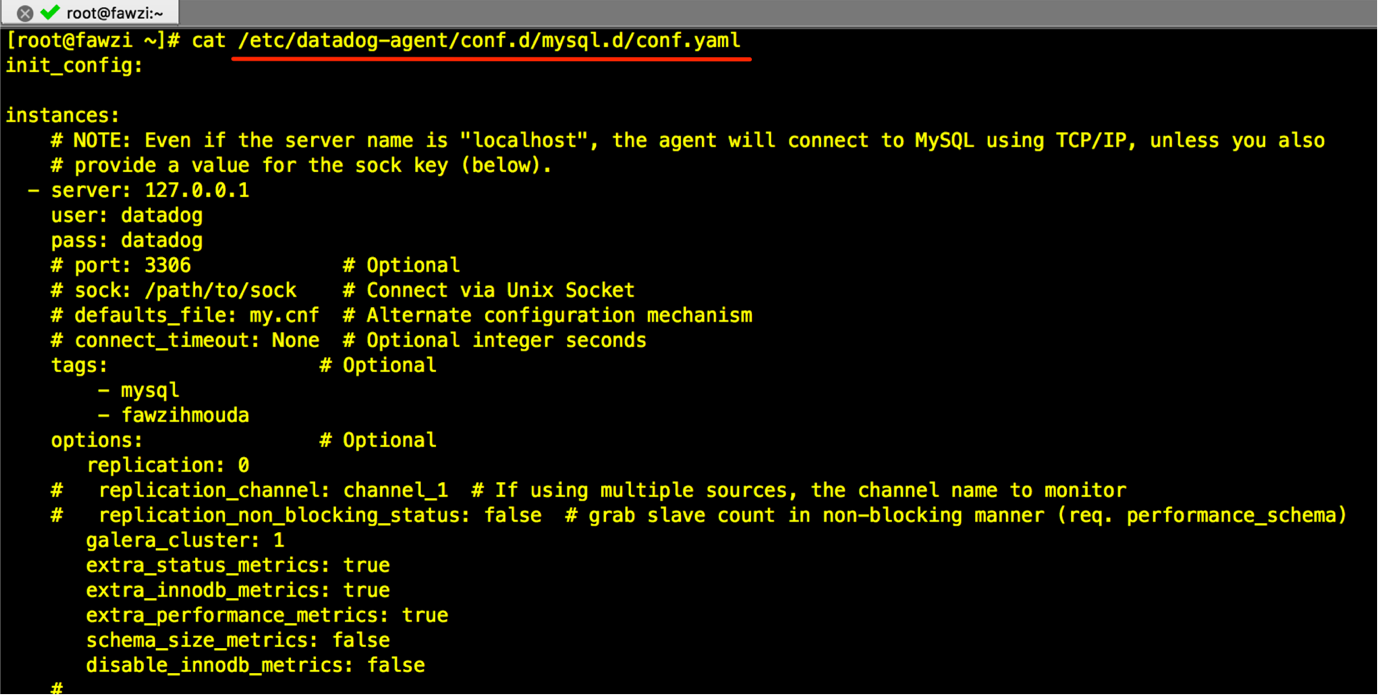
**Database installation**

I chose to install MySQL on my CentOS machine, and the picture bellow shows that MySQL is running properly in order integrate it with datadog;



In order to integrate MySQL into datadog, I followed the link [MySQL Integration](https://docs.datadoghq.com/integrations/mysql/) and then i created a read only datadog user and grant it the required privileges. For the lab purpose, I also granted the user the full metric catalogue.

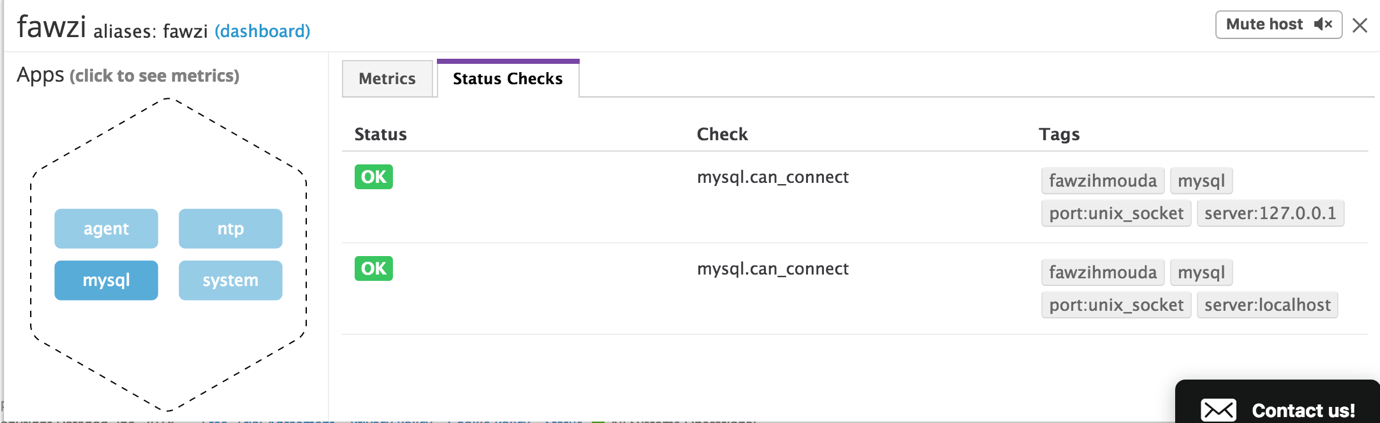
Mysql.yaml which is located under **/etc/datadog-agent/conf.d/mysql.d** must be edited and uncommented in order for the integration to be done;

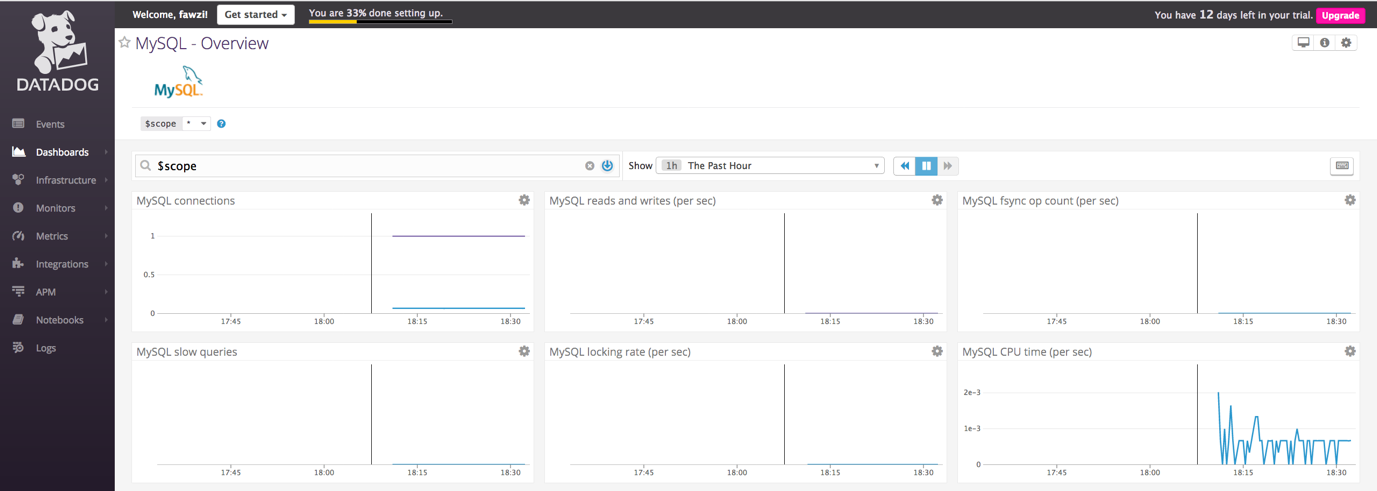


To complete this task, the agent must be restarted and (MySQL Integration) installing button should be pressed to begin the installation;

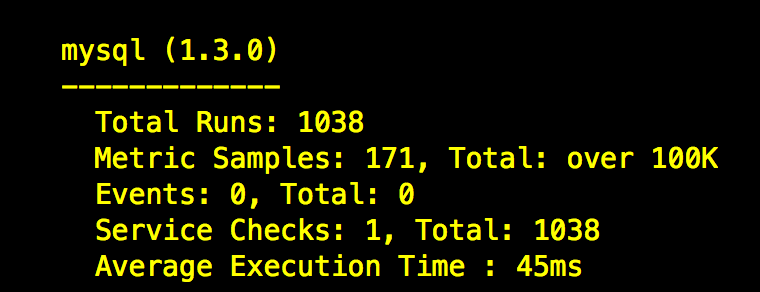


The host map page shows that MySQL integration is working properly and the dashboard (Second Picture) shows that it is collecting MySQL Metrics





another way to check if the integration is correctly done, we can issue the **#datadog-agent status**

****

**Creating a custom agent check**

In this task, the requirement is to create a custom agent check that submits a metric named my\_metric with a random value between 0 and 1000.

I Followed the instruction on how to create an agent check [here](https://docs.datadoghq.com/developers/agent_checks/).

The First step is to create 2 files which are **mycheck.py** and **mycheck.yaml**

!!! files name should match the name of the check module !!!

1st, **mycheck.py** located under the **checks.d** directory



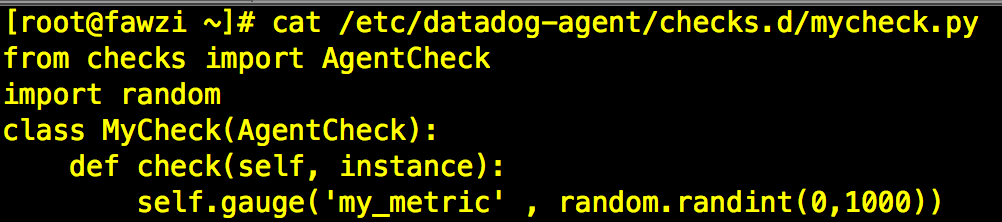
2nd, **mycheck.yaml** located under the **conf.d** directory



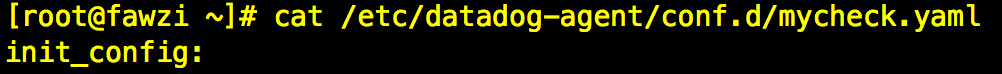
The second step is to create the check inside **mycheck.py**

This check uses the **self.gauge** method to submit the custom metric and the **random.randint(0,1000)** function to create random values between (0 – 1000)

The following picture is the mycheck.py ;



The third step is to edit the **mycheck.yaml** that was created under the **conf.d** directory, the file is pretty simple and it is as the following ;



The picture bellow shows the custom metric dashboard, I added a red marker line to show that the maximum number generated is 1000 which is a requirement for this task

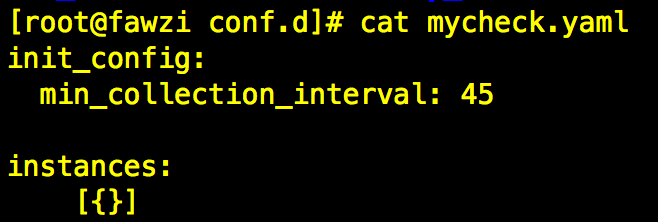


**Changing the metric check interval to 45s**

By default, and according to datadog, custom agent check run ever 20 seconds, to adjust this interval.

we could edit the **mycheck.py** by adding **import random, time** and **time.sleep(45).**

To adjust the data collection interval without touching the python app, we can add **min­­\_collection\_interval** under the **init\_config** section, which is an easier way. The picture bellow shows **min\_collection\_interval** was added to the YAML file followed by 45 which represent the interval time (45 second);



**3-Visualizing Data:**

In order to create a timeboard using datadog API, first we need to create a pair of credentials which are the API\_Key and APP\_KEY. API\_KEY is generated when we first created a datadog account, the second API KEY (App) should be generated from the APIs tab.

Later, a python file should be created, in my case I created a file called **fawzi-timeboard.py** which include the APIs and the program. To run the program, a python datadog module was installed **#pip install datadog**

The requirements for the timeboard are as the following;

1-Custom metric scoped over my host

2-Any metric from my Database (MySQL) with the anomaly function applied, in my case I chose the **mysql.performance.user\_time** metric.

3-My custom metric with the rollup function applied to sum up all the points for the past 1 hour into one bucket.

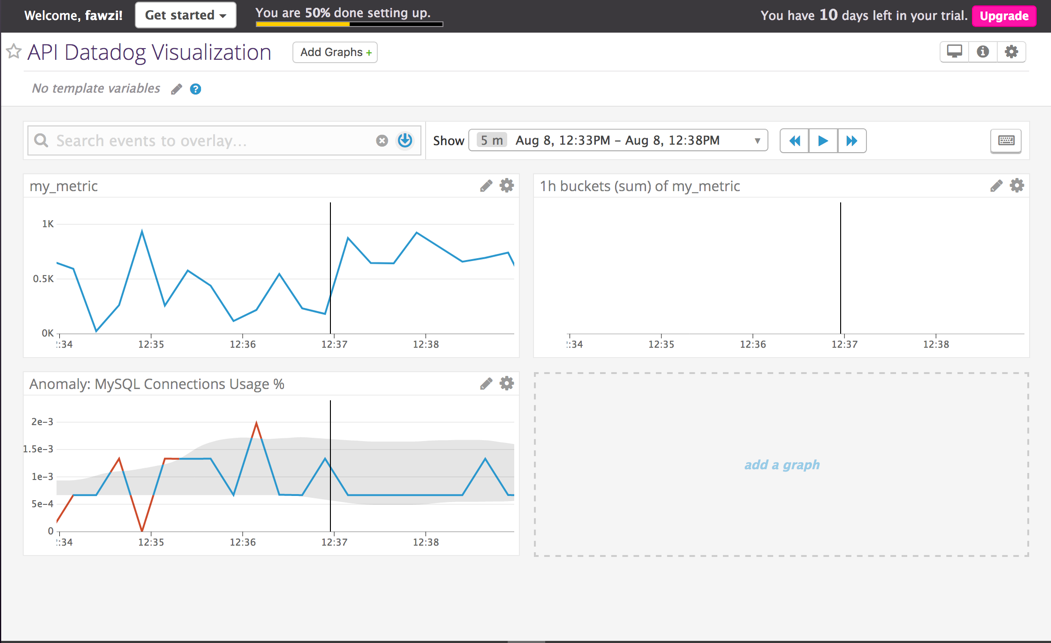
Added metrics to my fawzi-timeboard.py:

"q": "avg:my\_metric{host:fawzi}" graphs the average value of My\_Metric over my host (fawzi).

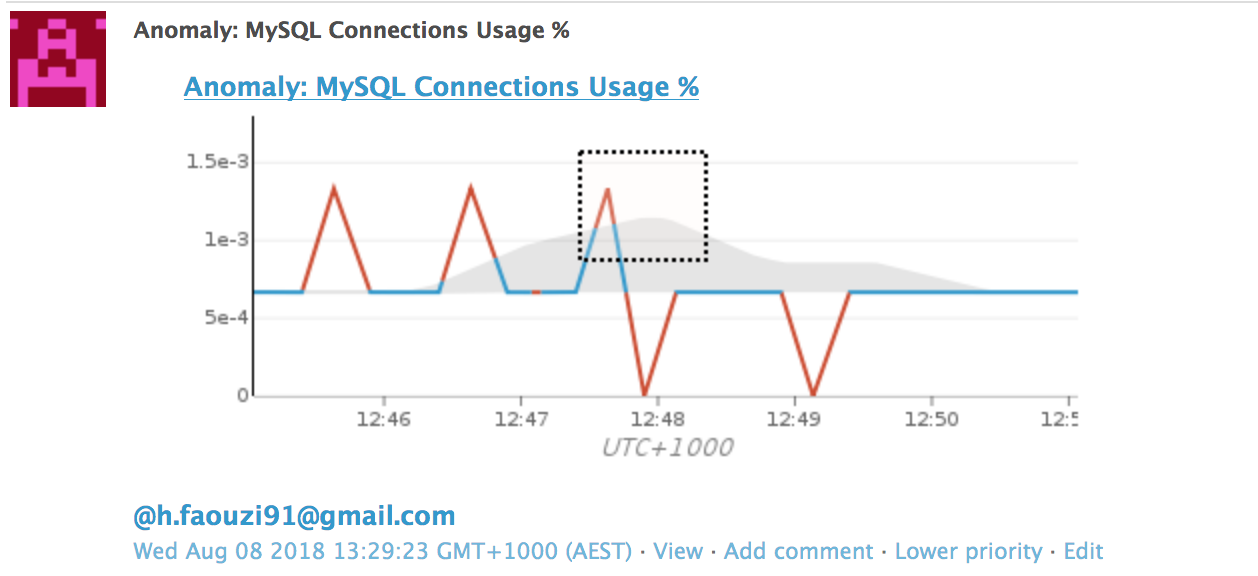
"q": "avg:my\_metric{host:fawzi}.rollup(sum, 3600)" take the sum of my metric over the last 1 hour (3600s) and rolls it up into a single point.

"q": "anomalies(avg:mysql.performance.user\_time{\*}, 'basic', 2 shows the anomalies for the percentage of CPU time spend in user space by MySQL

The following picture is what my timeboard looks like in the past 5 minutes:



Snapshot taken of my graph:



**Bonus question:**

The anomaly graph shows where a metric is behaving differently comparing to its behaviour in the past. The graph will display red colour which are the anomalies.

**4-monitoring data:**

In the previous task, a custom metric was created that generate values from 0 to 1000. For this task, we have to create a new Metric Monitor that watches the **average** of custom metric and will alert if it’s **above** the following values over the past **5 minutes**:

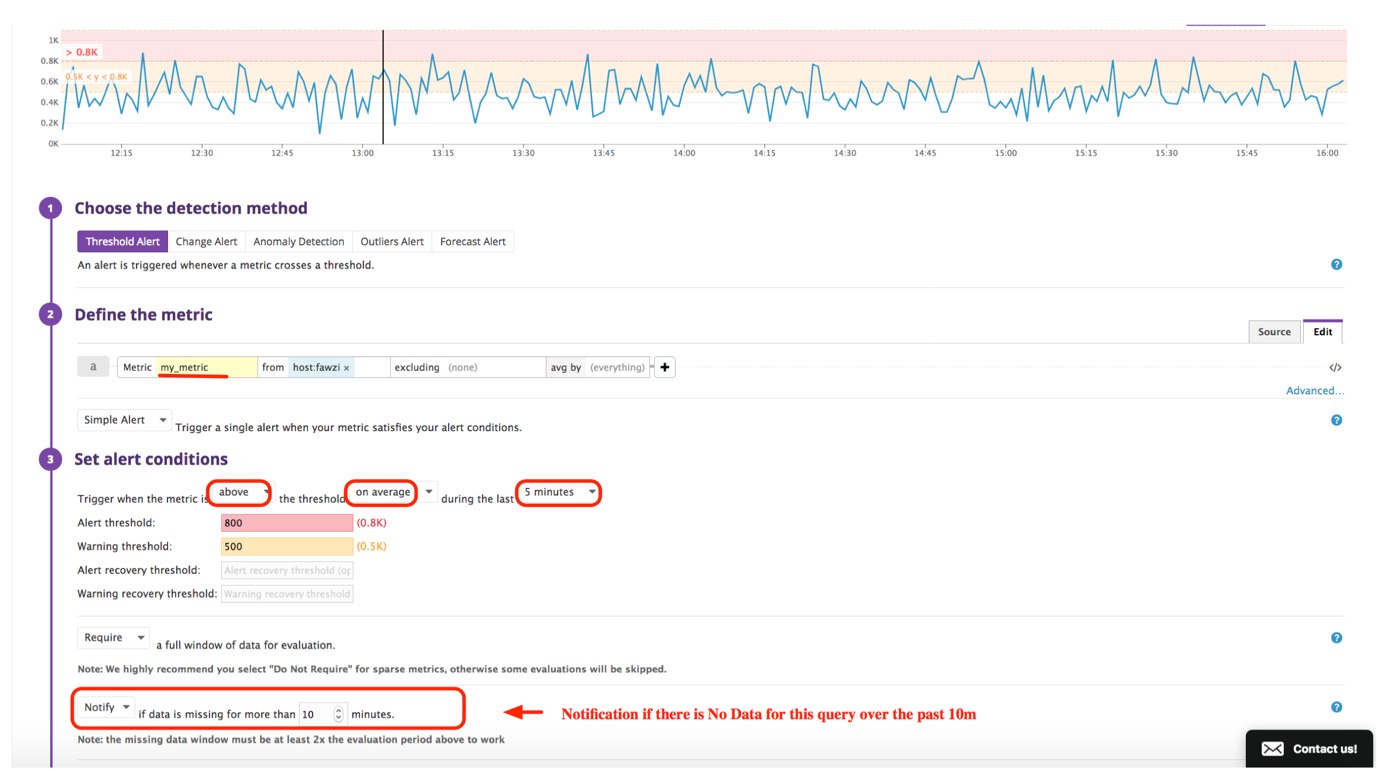
alerts conditions of:

1-Warning threshold of 500

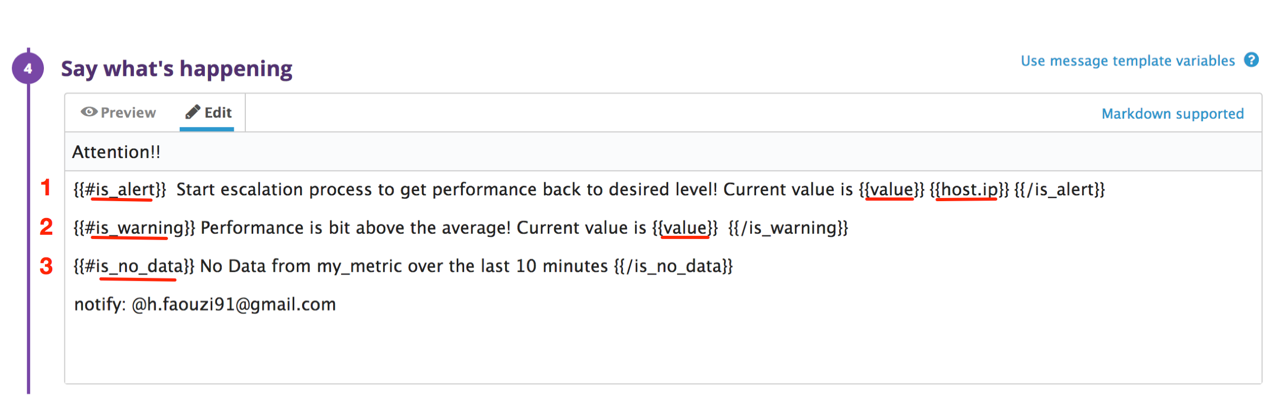
2-Alerting threshold of 800

3-generate a notification if there is no data over the past 10 minutes

For this part, the configuration is straight forward; just fill the required fields as bellow;



In order to manipulate and edit the message received when a value is above the alert or warning threshold or even if there is no data, the message template should be edited with the appropriate syntax;



In the picture above, we have 3 syntax, each one is an alert condition.

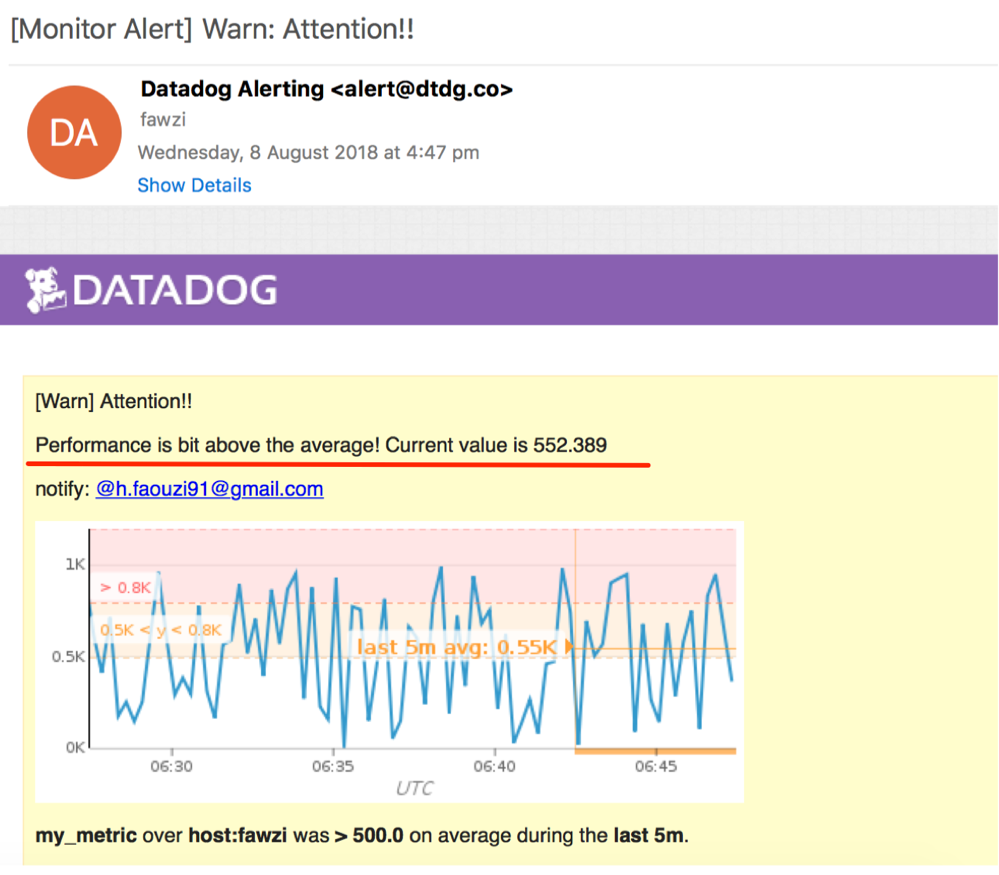
First syntax is for the alert condition, where it generates a message including the metric value that caused the monitor to trigger and host ip

Second syntax is for the warding condition, where it generate a message including the metric value that caused the monitor to trigger

Third syntax is for the no data condition, where it only generate a message

**@h.faouzi91@gmail.com** is to receive an email from the monitor

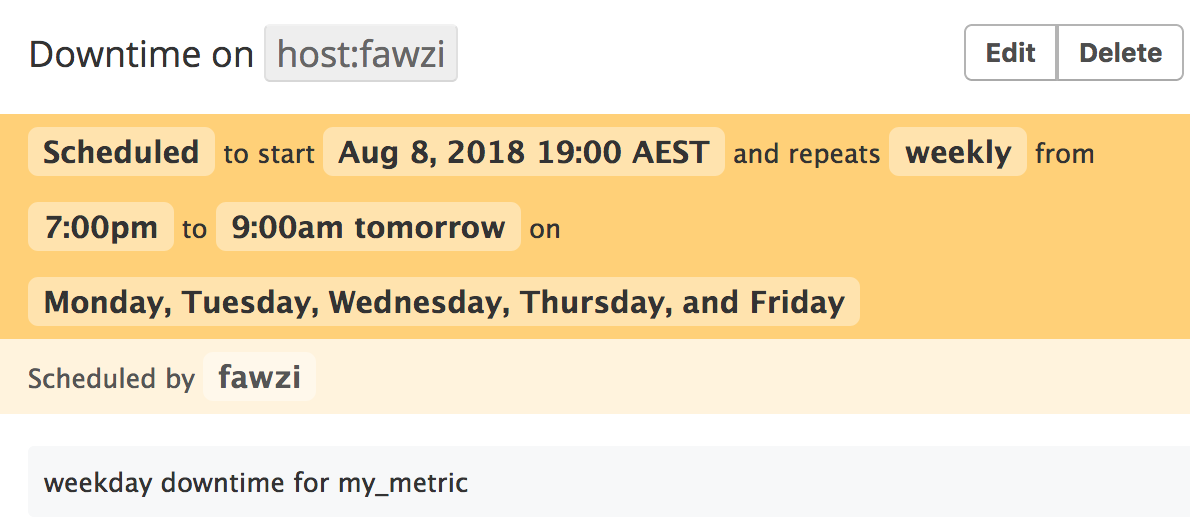
The picture bellow is a capture of my mail inbox, an email from my monitor, and because the value is 552.389, the category of this alert is a **Warning.**

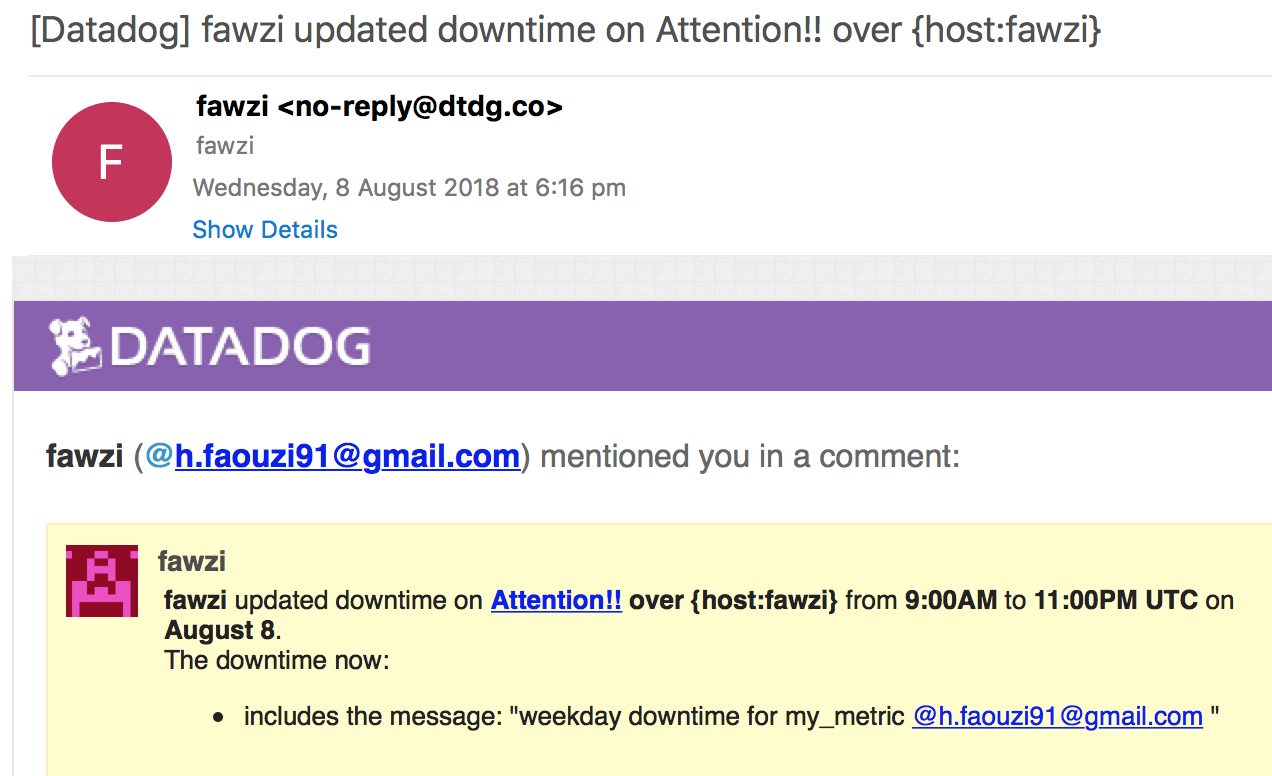


**Bonus Question:**

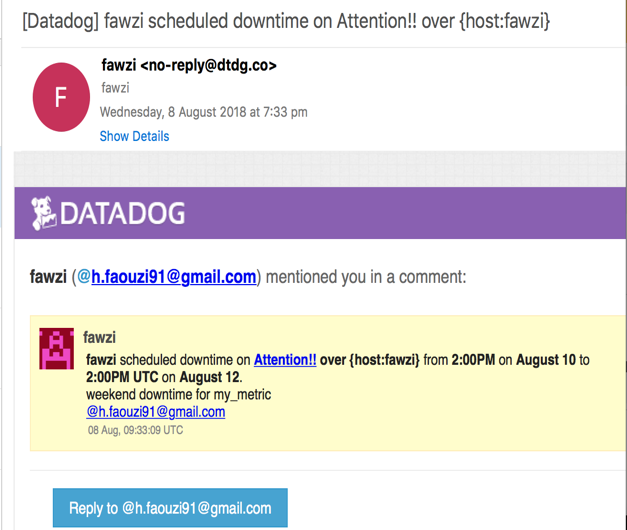
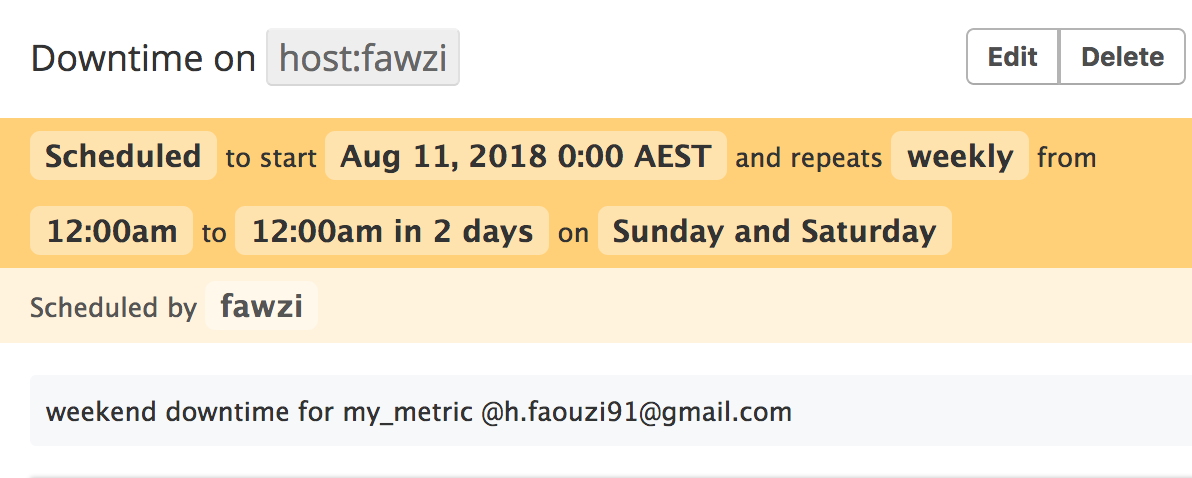
(Manage Downtime) button in the Monitors submenu, and created 2 new recurring scheduled downtimes

**1-silences from 7pm to 9am daily on M-F**

****

****

**2-silences it all day on Sat-Sun**

****

**5-Collection APM**

My final task was to take a simple Python Flask API, change it to use DataDog's program tracing solution. The [instruction](http://pypi.datadoghq.com/trace/docs/#module-ddtrace.contrib.flask) contained everything I needed to use the Flask middleware approach. After modifying the program and starting it with the command python flaskapp.py which the code is located in the code section.

!!! for some reason which I didn’t know, and after all the prerequisite installation were donr including uncommenting APM section in datadog.yaml, datadog cannot collect anything from my app !!!

**Bonus Question:**

Service is a set of processes that do the same job. For example, a simple web application may consist of 2 services: a single webapp service and a single database service.

Resource is a particular action for a service. For a web application: some examples might be a canonical URL, such as /user/home or a handler function like web.user.home. For a SQL database: a resource is the query itself, such as SELECT \* FROM users WHERE id = ?.

**Final Question**

1-We can use datadog to monitor people’s spending when they are using cards payments. We can apply the **TAG** approach on monitor to categorise such as groceries, cash out, retails spending and more as well as comparing the spending between each season.

2-Gather how many times the train in sydney has been delayed and gather statistics on the cost of commute that the delay represents

3- Get at regular intervals the traffic of a given distance to know what is are the best times in different days.